Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-97-02

Proceedings of the Fifth Meeting on Mathematics of Language – MOL5

Tilman Becker and Hans-Ulrich Krieger (eds.)

25–28 August 1997

Deutsches Forschungszentrum für Künstliche Intelligenz

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry of Education, Science, Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

Proceedings
of the
Fifth Meeting on Mathematics of Language
–
MOL5

Tilman Becker and Hans-Ulrich Krieger (eds.)

DFKI-D-97-02

This work has been supported by a grant from The Federal Ministry of Education, Science, Research and Technology (FKZ ITWM-01 IV 701 V0).

ISSN 0946-0098

Proceedings
of the
Fifth Meeting on Mathematics of Language

—

MOL5

Tilman Becker and Hans-Ulrich Krieger (eds.)

July 25, 1997

Abstract

The Fifth Meeting on Mathematics of Language (MOL5) covers all areas of study that deal with the mathematical properties of natural language. These areas include, but are not limited to, mathematical models of syntax, semantics and phonology; computational complexity of linguistic frameworks/theories and models of natural language processing; mathematical theories of language learning; parsing theory; and quantitative models of language.
The 1997 meeting takes place in the wonderfully located Schloss Dagstuhl, the 'International Meeting and Research Center for Computer Science' near Saarbruecken, Germany.

Contents

Characterizing learnability conditions for cue-based learners in parametric language systems

Stefano Bertolo, Kevin Broihier, Edward Gibson and **Kenneth Wexler**
Department of Brain and Cognitive Sciences - MIT
Cambridge, MA 02139
{bertolo,kevin,gibson,wexler}@psyche.mit.edu

Abstract

Applications of Formal Learning Theory to the problem of Language Acquisition have often had a rather narrow scope, due to the difficulty of expressing general results in a vocabulary meaningful for the allied disciplines of Developmental Psycholinguistics and Linguistics. In this paper we provide a complete characterization of the learnability properties of parametric systems of grammars of the kind hypothesized by Chomsky [5] with respect to the class of cue-based learners. In addition, we show that the conditions of the application of our negative learnability results are local, in the sense that they involve inspection of only a fragment of a parameter space and can be verified by standard tools of linguistic analysis.

Parametric Linguistics and Cue-based Learners

If, as it has been proposed by Chomsky [5], human languages all obey a common set of principles and differ from one another only in finitely many respects (often referred to as *parameters*) and in these respects only in finitely many ways (the *values* of the *parameters*), then human language learning can be seen as a search problem in a finite hypothesis space: the child does not need to hypothesize grammars that fall beyond those that are consistent with the common set of principles (often referred to as *Universal Grammar*) and any of the possible assignment of values to the linguistic parameters. However, although finite, this space of hypothesis can still be quite large (recent principled estimates place this number around 2^{40} different possible grammars[1]) and it is therefore imperative for any parametric model of language acquisition to show how such a huge hypothesis space could be searched effectively since this is arguably what children do.

Some linguists (e.g. Dresher and Kaye [7]) have observed that this huge hypothesis space could be searched effectively if children were capable of establishing the value of certain parameters by attending to linguistic events of a particular nature in their environment. In fact, if all parameters are binary valued establishing the value of a parameter eliminates exactly half of the hypotheses from the hypothesis space. Ideally, 40 such observations could be sufficient to single out a grammar out of 2^{40} possible alternatives.

[1]This estimate can been obtained by restricting all parametric variation to the ability or inability of functional heads to attract other heads or maximal projections and by estimating the number of functional head that are required for descriptive adequacy. On this see Roberts [14].

The following artificial example should help to give an idea as to what these observations could amount to. Suppose you were trying to determine, from a collection of positive examples, which one of the following four regular expressions generates the sample:

$$a\left\{\begin{array}{c} b \cup a \\ b^* \end{array}\right\} c\left\{\begin{array}{c} d \cup c \\ d^* \end{array}\right\} ef^*.$$

One way to solve this problem could be to set up a battery of tests to be applied to each one of the positive samples and to make choices about the assignment of value that is appropriate for each parameter depending on the outcome of these tests:

Test input string	Response if test positive
T_1: two a's in a row?	set p_1 to value $b \cup a$
T_2: two b's in a row?	set p_1 to value b^*
T_3: two c's in a row?	set p_2 to value $d \cup c$
T_4: two d's in a row?	set p_2 to value d^*

In this construction, the observable event of a sample string having two a's in a row is taken as a *cue* to the $b \cup a$ value assignment for the first parameter. The goal of the cue-based learning enterprise is to show that it is simultaneously possible to reconstruct linguistic variation parametrically and to single out in each possible target language a set of cues that would allow a learner to acquire the correct setting for each parameter.

A theory of cue-based learners

Although the central intuition about the design philosophy of a cue-based learner emerges quite clearly from the example above, a formal characterization of the class of these algorithms turns out to be quite useful on at least two counts. First of all, a formal definition will make it possible to capture some essential design features in learning algorithms that appear to be prima facie unrelated. Second, by establishing learnability results about the class of cue-based algorithms at large one would automatically have results that can be applied to each individual algorithm. For an application of the results of this paper to the analysis of a parametric language learner based on unambiguous 'superparsing' proposed by Fodor [8] see Bertolo et al. [2].

Since, as we saw, the salient feature of a cue-based learner is to restrict the hypothesis space of a parametric learning problem, we first need to introduce a definition of parameter spaces.

Definition 1 *A parameter space $\mathcal{P}$ is a triple $< par, L, \Sigma >$, where Σ is a finite alphabet of symbols and par is a finite set of sets $\{p_1, \ldots, p_n\}$. Given a p_i in par, its members are enumerated as $v_i^1, \ldots, v_i^{|p_i|}$. Given the cartesian product $\mathbf{P} = p_1 \times p_2 \times \ldots \times p_n$, a parameter vector $\overline{P}$ is a member of $\mathbf{P}$. The function $L : \mathbf{P} \mapsto 2^{\Sigma^*}$ assigns a possibly empty subset of Σ^* to each vector $\overline{P} \in \mathbf{P}$. The expression $\mathcal{L}(\mathbf{P})$ denotes the set $\{L(\overline{P_1}), \ldots, L(\overline{P_{|\mathbf{P}|}})\}$.*

Given a parameter space, it turns out to be useful to be able to refer to an assignment of values to some, but not all of the parameters.

Definition 2 *Let $\mathcal{P}$ be a parameter space. A partial assignment in $\mathcal{P}$ is any subset B of*

$$\bigcup_{p_i \in par} \{p_i\} \times p_i$$

such that for every p_i in par there is at most one $< p_i, v_i^m >$ in B. Given two partial assignments A and B in $\mathcal{P}$, B is said to be A-consistent iff $A \cup B$ is also a partial assignment in $\mathcal{P}$.

Such partial assignments can in turn be used to isolate only those parts of a parameter space that agree on the values assigned to the parameters in a partial assignment. Crucially, such a portion of a parameter space is, by definition 1, itself a parameter space.

Definition 3 *Let $\mathcal{P}$ be a parameter space and $\mathcal{P}[\emptyset] = \mathcal{P}$. If $\mathcal{P}[A]$ is a parameter space $< par^A, L, \Sigma >$ and B is an A-consistent partial assignment in $\mathcal{P}$, then the subspace $\mathcal{P}[A \cup B]$ is the parameter space $< par^{A \cup B}, L, \Sigma >$ such that, given*

$$H = \bigcup_{x \in B} \pi_1(x),$$

if $p_j \notin H$ then $p_j^{A \cup B} = p_j^A$ and if $p_j \in H$ then $p_j^{A \cup B} = \{v_j^m\}$ where v_j^m is the only $v \in p_j$ such that $< p_j, v_j^m > \in B$. Finally, $\mathcal{P}[A \cup \overline{B}]$ is the parameter space $< par^{A \cup \overline{B}}, L, \Sigma >$ where, for every p_i in H, $p_i^{A \cup \overline{B}} = p_i^A - p_i^{A \cup B}$ and, for every p_i not in H, $p_i^{A \cup \overline{B}} = p_i^A$.

We are now ready to formalize the notion of some parameter values being established as a result of observing certain events in the linguistic environment. The function ϕ_C of definition 4 can be seen as a formal representation of the battery of tests discussed in the example above. It is important to notice that definition 4 generalizes our original intuition in two important respects. First of all, it captures the possibility that the learner, upon observation of a linguistic event, could reach different conclusions depending on what its current state of belief (assignment of value to certain parameters) is. Secondly, it allows for the existence of linguistic events that can only be observed comparing n distinct data points (in the case of syntax learning, typically sentences).

Definition 4 *Let $\mathcal{P}$ be a parameter space, B a subset of the set B^* of all partial assignments in $\mathcal{P}$, C a non-empty subset of $\bigcup_{\overline{P} \in \mathbf{P}} L(\overline{P})$ and C^i the cartesian product of C with itself i times. A cue function of window size n for $\mathcal{P}$ is a function*

$$\phi_C : \bigcup_{i=1}^{n} C^i \times B \mapsto B^*$$

such that:

1. *if $\overline{s_i}, \overline{s_j} \in \bigcup_{i=1}^{n} C^i$ are permutations of one another, then, for every B, $\phi_C(\overline{s_i}, B) = \phi_C(\overline{s_j}, B)$ and*
2. *if $\phi_C(\overline{s_i}, A) = B$ and $\phi_C(\overline{s_j}, A \cup B) = C$, then if $\overline{s_k} = \overline{s_i} \circ \overline{s_j}$, then $\phi_C(\overline{s_k}, A) = B \cup C$.*

Restrictions 1) is meant to ensure that the cue functior be 'locally set-driven'.[2]. Restriction 2) is needed to avoic the case of a cue function that reacts to a set of data point: differently than it does to a sequential presentation of one o its partitions.

Finally, a cue-based learner is a learning algorithm tha does all its learning via a cue function. The crucial feature o such learners is the absence of any form of backtracking: a definition 5 shows, if the cue function returns any parameter assignment that is not in agreement with the current assignment, the inconsistent portion of the output of the cue functior is simply discarded.

Definition 5 *Let $\mathcal{P}$ be a parameter space, B^*, B and C as in definition 4 and ϕ_C a cue function of window size n for $\mathcal{P}$. A cue-based learner for $\mathcal{P}$ is a function*

$$\lambda_C : \{\mathcal{P}[A] | A \in B\} \times \bigcup_{i=1}^{n} C^i \mapsto \{\mathcal{P}[A] | A \in B^*\}$$

such that

$$\lambda_C(\mathcal{P}[A], \overline{s}) = \begin{cases} \mathcal{P}[A] & \text{if } \overline{s} \notin \bigcup_{i=1}^{n} C^i \\ \mathcal{P}[A \cup B_A] & \text{if } \phi_C(\overline{s}, A) \text{ is not } A\text{-consistent} \\ \mathcal{P}[A \cup \phi_C(\overline{s}, A)] & \text{otherwise} \end{cases}$$

where B_A is the largest A-consistent subset of $\phi_C(\overline{s}, A)$.

Now that we have formalized what a cue-based learnei does in response to a linguistic event we need to formalize how a sequence of data points can be parsed into a sequence of events compatible with a given window size. This step is necessary in order to investigate the behavior of such a learnei in the limit.

Definition 6 *Let σ be a sequence of elements from a set L and σ_i denote the i-th element in σ. The expression $w(\sigma_i, \sigma, m)$ denotes the sequence of sequences $\sigma^{im} = \sigma_1^{im}, \sigma_2^{im}, \ldots, \sigma_k^{im}$ where*

$$\begin{aligned} \sigma_1^{im} &= \sigma_i \\ \sigma_2^{im} &= \sigma_{i-1}\sigma_i \\ \sigma_3^{im} &= \sigma_{i-2}\sigma_{i-1}\sigma_i \\ &\ldots \\ \sigma_k^{im} &= \sigma_{i-k} \cdots \sigma_{i-2}\sigma_{i-1}\sigma_i \end{aligned}$$

and k is the largest number such that $k \leq m$ and $i - k \geq 1$. The expression $W(\sigma, m)$ then denotes the sequence of sequences $w(\sigma_1, \sigma, m) \circ w(\sigma_2, \sigma, m) \circ \ldots$ where $\circ$ is the concatenation sign.

For example, given the infinite sequence $\sigma = 1, 2, 3, 4, \ldots$, $W(\sigma, 3)$ is the infinite sequence

$$(1), (2), (2, 1), (3), (3, 2), (3, 2, 1), (4), (4, 3), (4, 3, 2), \ldots$$

Finally, we need to define the behavior of a cue-based learner on strings of arbitrary length.

[2] See Wexler and Culicover [16], sec. 2.2 and Osherson, Stob and Weinstein [12], sec. 4.4.2 for definitions and consequences of the general property of being 'set-driven'.

Definition 7 *Let $\mathcal{P}[A]$ be a parameter space, C as in definition 4 , σ a sequence of strings from $\bigcup_{\overline{P}\in\mathbf{P}} L(\overline{P})$ and $\tau = W(\sigma, m)$ with τ^+ denoting the sequence such that $\tau_1 \circ \tau^+ = \tau$. If λ_C is a cue based learner with a window of size m for $\mathcal{P}$, $\lambda_C(\mathcal{P}[A], \tau)$ is defined as $\lambda_C(\lambda_C(\mathcal{P}[A], \tau_1), \tau^+)$.*

This completes the formalization of the notion of a cue-based learner. We now want to show under what conditions exactly such learners are successful. We will do this in two steps. Given a criterion of successful learning, we will first show what characteristic a cue function should have for the corresponding cue-based learner to be successful and we will then show how the existence (or non existence) of the required characteristics could be established by an analysis of the parameter space which can often be grounded on a linguistically respectable vocabulary.

Characterizing conditions for successful cue-based learners

Our characterization of cue-based learners was general enough to include the possibility of a learner reacting to the same input sentence in different ways, depending on what its particular current partial assignment is at that particular moment. Given the possibility of this kind of potential cueing, it will turn out to be useful to have a notion of which parts of a parameter space a cue-based learner will visit in response to data from a given language. The following definition formalizes this notion.

Definition 8 *Given two partial assignments A and B and a $\overline{P} \in \mathbf{P}$ and a ϕ_C of window size m, $T_{\phi_C}(\overline{P})$ denotes the set of all triples $< \overline{s}, A, B >$ such that $\overline{s} \in L(\overline{P})^j$ for some $j \leq m$, $\phi_C(\overline{s}, A) = B$ and B is A-consistent.*

1. *A ϕ_C-chain for $\overline{P}$ is a sequence $t_1, t_2, \ldots, t_n$ of elements of $T_{\phi_C}(\overline{P})$ such that $\pi_2(t_1) = \emptyset$ and, for all $n \leq m$, $\pi_2(t_n) = \pi_2(t_{n-1}) \cup \pi_3(t_{n-1})$.*
2. *A ϕ_C-chain $t_1, t_2, \ldots, t_n$ for $\overline{P}$ is maximal iff there is no $t \in T_{\phi_C}(\overline{P})$ such that $t_1, t_2, \ldots, t_n, t$ is also a ϕ_C-chain for $\overline{P}$ and $\pi_3(t_n) \neq \pi_3(t)$.*
3. *A ϕ_C-trajectory for $\overline{P}$ is a sequence $K_1, \ldots, K_m$ such that for $K_1 = \pi_2(t_1)$ and, for all $1 < i \leq m$, $K_i = \pi_3(t_i)$ for some maximal ϕ_C-chain $t_1, t_2, \ldots, t_n$ for $\overline{P}$.*
4. *The expression $T(\overline{P})$ denotes the set*

$$\{\pi_2(t) | t \text{ is a member of a maximal } \phi_C\text{-chain for } \overline{P}\}$$

Our definition of a cue function is general enough to include the case of a 'silent' cue function that returns an empty set of parameter values in response to every sentence from the target language. It is intuitively obvious why such a function could hardly be useful for a cue-based learner. It is equally intuitive that for a cue function to be of any use to cue-based learner it must satisfy two conditions: it must eventually yield a complete conjecture (that is a conjecture in which every parameter has a value assigned to it) and it should never restrict itself to a subspace that does not contain the target or one of its equivalents. The following definitions are introduced in order to formalize these notions and theorem 2 will show that the intuition presented above is indeed correct.

Definition 9 *Let $\mathcal{P}$ be a parameter space and ϕ_C a cue function for $\mathcal{P}$. ϕ_C is complete iff for every $\overline{P} \in \mathbf{P}$ and every ϕ_C-trajectory T for $\overline{P}$, $|\overline{P}_T| = |par|$, where $\overline{P}_T$ is the union of all the elements of T.*

Definition 10 *Let $\mathcal{P}$ be a parameter space and ϕ_C a cue function of window size m for it. ϕ_C is coherent iff for every $\overline{P} \in \mathbf{P}$ and every partial assignment $A \in T(\overline{P})$ if $\overline{s} \in L(\overline{P})^i$ (with $i \leq m$) and $\phi_C(\overline{s}, A) = B$ then, for some $\overline{P'} \in \mathbf{P}^{A \cup B}$, $L(\overline{P'}) = L(\overline{P})$.*

In order to prove that completeness and coherence of a cue function are jointly necessary and sufficient for the success of a cue based algorithm we need to specify adequately our criterion of success. The following definitions adapt to our parametric scenario Gold's [9] criterion of identification in the limit.

Definition 11 *Let $\mathcal{P}$ be a parameter space, ϕ_C a cue function of window size m. A sequence σ of elements of $\Sigma^* \times \{0, 1\}$ is said to be* for *a language $L(\overline{P})$ iff, for every i, $\sigma_i \in L(\overline{P})$.*

1. *A sequence σ for a language $L(\overline{P})$ is a* text *for that language iff for every $s \in L(\overline{P})$ there is an $\sigma_i =< s, 1 >$.*
2. *A sequence σ for a language $L(\overline{P})$ is an* informant *for that language iff for every $\overline{s} \in L(\overline{P})$ there is an $\sigma_i = \overline{s}$.*
3. *Given a ϕ_C-chain $t_1, \ldots, t_n$ and a sequence σ for $L(\overline{P})$, σ is said to* complete *$t_1, \ldots, t_n$ iff, given $\tau = W(\sigma, m)$*
   (a) *there are $\tau_{i_1}, \ldots, \tau_{i_n}$ such that, for all $1 \leq k \leq n$, $\tau_{i_k} = \pi_1(t_k)$, and*
   (b) *for all $1 \leq k < n$, $i_{k+1} > i_k$ and*
   (c) *for any two $\tau_{i_k}, \tau_{i_{k+1}}$ there is no $\tau_{i_{k'}}$ such that $i_k < k' < i_{k+1}$ and $< \tau_{i_{k'}, \pi_2(t_{k+1}), B} > \in T_{\phi_C}(\overline{P})$ for some $B \neq \pi_3(t_{k+1})$ and for τ_{i_1} there is no τ_j such that $j < i_1$ and $< \tau_{i_j, \emptyset, B} > \in T_{\phi_C}(\overline{P})$ for some $B \neq \pi_3(t_1)$.*
4. *The sequence σ, is said to be C^m-rich for $L(\overline{P})$ iff it completes at least one maximal ϕ_C-chain for $L(\overline{P})$.*

Definition 12 *Given a parameter space $\mathcal{P}$, a cue-based learner λ_C of window size m, an infinite sequence σ of members of $\Sigma^* \times \{0, 1\}$ and the sequence $\tau = W(\sigma, m)$, λ_C is said to be defined on τ iff λ_C is defined on $\overline{\tau_n}$ for every n, where $\overline{\tau_n}$ is the segment containing the first n elements of τ. Let B be a partial assignment for $\mathcal{P}$. λ_C is said to converge on τ to $\mathcal{P}[B]$ iff λ_C is defined on τ and for all but finitely many n $\lambda_C(\mathcal{P}, \overline{\tau_n}) = \mathcal{P}[B]$. Given a sequence σ for a language $L(\overline{P})$, λ_C is said to identify σ iff $\lambda_C(\mathcal{P}, \tau) = \mathcal{P}[B]$, $|\mathbf{P}^B| = 1$ and $L(\overline{P})$ is equivalent to the only $L(\overline{P'})$ such that $\overline{P'} \in \mathbf{P}^B$. Finally, λ_C is said to identify $L(\overline{P})$ on C^m-rich text (or informant) iff it identifies every C^m-rich text (or informant) for $L(\overline{P})$*

Notice that this definition of successful learning differs from Gold's [9] original definition (see also Osherson, Stob and Weinstein [12]) in precisely the respect that is relevant in an application of formal learning theory to the problem of language acquisition. Since the environments in which humans learn their respective target languages does *not* include all the sentences of the target languages, it is useful to have a notion of success that does not require the learner's conjecture

to generate a language that is equivalent to the set of sentences present in the environment, as is the case for Gold's [9] original definition.

Given this definition, it is easy to see that cue-based learners cannot identify languages unless they receive C^m-rich sequences for those languages.

Theorem 1 *Given a parameter space $\mathcal{P}$, a cue-based learner λ_C of window size m and a sequence σ for a language $L(\overline{P})$, λ_C identifies σ only if σ is C^m-rich.*

PROOF. Assume σ is not C^m-rich. Then, by definition 11, σ does not complete any maximal ϕ_C-chain for $L(\overline{P})$. Let $t_1, \ldots, t_m$ be a ϕ_C-chain completed by σ. Since $t_1, \ldots, t_m$ is not maximal, $|\mathbf{P}^{\pi_2(t_m) \cup \pi_3(t_m)}| > 1$. By definition 12, this implies that λ_C does not identify σ.
□

Having established that no identification takes place on sequences that are not C^m-rich we are now ready to prove that cue-based learners are successful on C^m-rich sequences if and only if they rely on a complete and coherent cue function. The proof will be aided by the following lemma, that makes explicit the consequences of restricting one's search to the wrong subspace when this is coupled with a general inability to backtrack.

Lemma 1 *Let $\mathcal{P}[A]$ be a parameter space and L a subset of Σ^* such that $L \notin \mathcal{L}(\mathbf{P}^A)$. Then, for every partial assignment B, $L \notin \mathcal{L}(\mathbf{P}^{A \cup B})$.*

PROOF. Since, by construction, for every partial assignment B $\mathbf{P}^A \supseteq \mathbf{P}^{A \cup B}$, if $L \in \mathcal{L}(\mathbf{P}^{A \cup B})$ then $L \in \mathcal{L}(\mathbf{P}^A)$. □

Theorem 2 *Let $\mathcal{P}$ be a parameter space and λ_C a cue-based learner of window size m. Then λ_C identifies every $L(\overline{P}) \in \mathcal{L}(\mathbf{P})$ on C^m-rich sequences if and only if the corresponding cue function ϕ_C is complete and coherent.*

PROOF. $\Longrightarrow$ Assume ϕ_C is not coherent. Then, there is a $\overline{P} \in \mathbf{P}$, an $A \in T(\overline{P})$ and a $\overline{s} \in L(\overline{P})^i$ (with $i \leq m$) such that $\phi_C(\overline{s}, A) = B$ and, for every $\overline{P'} \in \mathbf{P}^{A \cup B}$, $L(\overline{P'}) \neq L(\overline{P})$. Let $\overline{s} = s_1, s_2, \ldots, s_n$ and σ^A be a sequence of couples from $L(\overline{P})$ such that $\lambda_C(\mathcal{P}, W(\sigma^A, m)) = \mathcal{P}[A]$. This sequence is guaranteed to exist, since $A \in T(\overline{P})$. Then, for every sequence σ for $L(\overline{P})$, λ_C does not identify the sequence $\sigma^A \circ s_1 \circ \ldots \circ s_n \circ \sigma$ and so, in particular, for every sequence σ' for $L(\overline{P})$ such that $\sigma^A \circ s_1 \circ \ldots \circ s_n \circ \sigma'$ is C^m-rich, λ_C does not identify the sequence $\sigma^A \circ s_1 \circ \ldots \circ s_n \circ \sigma'$. In fact, by hypothesis and by the construction of σ^A it follows that $\lambda_C(\mathcal{P}, W(\sigma^A \circ s_1, \ldots s_n, m)) = \mathcal{P}[A \cup B]$. By lemma 1 and the fact that for every $\overline{P'} \in \mathbf{P}^{A \cup B}$, $L(\overline{P'}) \neq L(\overline{P})$ it follows that, for every partial assignment D, for every $\overline{P'} \in \mathbf{P}^{A \cup B \cup D}$, $L(\overline{P'}) \neq L(\overline{P})$. Therefore, since, for every sequence σ for $L(\overline{P})$, $\lambda_C(\mathcal{P}, \sigma^A \circ s \circ \sigma) = \mathcal{P}[A \cup B \cup D]$ for some partial assignment D, λ_C does not identify $\sigma^A \circ s \circ \sigma$.

Assume ϕ_C is not complete. Then, for some $\overline{P} \in \mathbf{P}$, for some ϕ_C-trajectory $T = A_0, \ldots A_k$ and for some $1 \leq i \leq n$, $\pi_i(\overline{P}) = v_i^m$ and $< p_i, v_i^m > \notin \bigcup_{i=0}^{k} A_i$. Let σ^{A_k} be a sequence of couples from $L(\overline{P})$ such that $\lambda_C(\mathcal{P}, W(\sigma^{A_k}, m)) = \mathcal{P}[\bigcup_{i=0}^{k} A_i]$. This sequence is guaranteed to exist, since $\bigcup_{i=0}^{k} A_i \in T(\overline{P})$. So, for every sequence σ (and so, a fortiori, for every sequence σ' such that $\sigma^{A_k} \circ \sigma'$ is C^m-rich) for $L(\overline{P})$, if $\lambda_C(\mathcal{P}[\bigcup_{i=0}^{k} A_i], \sigma) = \mathcal{P}[B]$ then $< p_i, v_i^m > \notin B$. This however, implies that $|\mathbf{P}^B| > 1$ and so λ_C does not identify $\sigma^{A_k} \circ \sigma$.

$\Longleftarrow$ Assume that ϕ_C is complete and coherent. Let σ be a C^m-rich sequence for some $L(\overline{P})$ and $t_1, \ldots, t_k$ the maximal ϕ_C-chain σ completes. From definition 5 it follows immediately that $\lambda_C(\mathcal{P}, W(\sigma, m)) = \mathcal{P}[\pi_2(t_k) \cup \pi_3(t_k)]$. To show that λ_C identifies a sequence σ for $L(\overline{p})$ we need to show that $|\mathbf{P}^{\pi_2(t_k) \cup \pi_3(t_k)}| = 1$ and, for every $L(\overline{P}) = L(\overline{P'})$ for the only $\overline{P'} \in \mathbf{P}^B$. So, assume that λ_C does not identify σ. This implies that, if $\lambda_C(\mathcal{P}, W(\sigma, m)) = \mathcal{P}[\pi_2(t_k) \cup \pi_3(t_k)]$, then either $|\mathbf{P}^{\pi_2(t_k) \cup \pi_3(t_k)}| > 1$ or for the only $\overline{P'} \in \mathbf{P}^B$, $L(\overline{P'}) \neq L(\overline{P})$.

Assume $|\mathbf{P}^{\pi_2(t_k) \cup \pi_3(t_k)}| > 1$. This implies that, for every $\overline{s} \in \bigcup_{i=1}^{m} (C \cap L(\overline{P}))^i$, $\pi_2(t_k) \cup \pi_3(t_k) \supseteq \phi_C(\overline{s}, \pi_2(t_k) \cup \pi_3(t_k))$. But this contradicts the hypothesis that ϕ_C is complete.

Assume instead that $|\mathbf{P}^{\pi_2(t_k) \cup \pi_3(t_k)}| = 1$ and, $L(\overline{P'}) \neq L(\overline{P})$. So, by definition 10, if we can show the existence of a $\pi_2(t_i)$ be such that:

1. $\lambda_C(\mathcal{P}[\pi_2(t_i)], \overline{s_j}) = \mathcal{P}[\pi_2(t_i) \cup \pi_3(t_i)]$;
2. for some $\overline{P'}$ such that $L(\overline{P'}) = L(\overline{P})$, $\overline{P'} \in \mathbf{P}^{\pi_2(t_i)}$ and
3. for every $\overline{P'}$ such that $L(\overline{P'}) \neq L(\overline{P})$, $\overline{P'} \notin \mathbf{P}^{\pi_2(t_i) \cup \pi_3(t_i)}$,

then we have a proof that ϕ_C is not coherent, contrary to the assumption. But such a $\pi_2(t_i)$ is guaranteed to exist from the assumption that $\pi_2(t_1) = \emptyset$ (and so $\overline{P} \in \mathbf{P}^{\emptyset}$) and the assumption that $\overline{P} \notin \mathbf{P}^{\pi_2(t_k) \cup \pi_3(t_k)}$.
□

Expressive cue functions and the Non-Disjunctive Subspace property

Although, as theorem 2 shows, completeness and coherence of cue functions completely characterize the class of successful cue-based learners, they don't capture a desideratum of cue functions that is quite obvious on grounds of psychological plausibility, especially if, as in the example discussed in the first section, one regards cue functions as batteries of tests that can be performed on linguistic events. In particular, it is perfectly possible for a complete and coherent function to receive two input linguistic events $\overline{s}_1$ and $\overline{s}_2$ from the same language $L(\overline{P})$ and return two distinct value assignments v_i^m and v_i^k for the same parameter p_i. Of course, if the function is coherent, this can only happen if one of these two events also belongs to some other language. This is undesirable because, in the presence of a linguistic event that could have been produced by two different languages, it is as if the function were 'partial' to one of these possibilities over the alternative.

A different way to see why functions that are 'partial' are undesirable is the following: ideally, a cue-based learner ought to commit itself to a particular assignment of value to a parameter only if the sequence of data points that cause it to do so somehow 'expresses' that assignment of value. There are several different ways to reconstruct this notion (see Clark [6] for a discussion concerning actual syntactic parameters and Bertolo [1] for a formal definition) but the intuition is that a sentence does not express the value v_i^m of a parameter p_i if it is also a member of languages corresponding to parameter

assignments where p_i is assigned a value different from v_i^m (although the converse does not hold).

This implies that, if we want cue-based learners to be something more than hash-tables and require that their learning behavior be guided by some form, however superficial of linguistic analysis of the input data (as it is done, for example, in Fodor's [8] 'superparsing' algorithm) we have to require that the data they use to return a parameter value must express that parameter value. The following definition formalizes this notion.

In order to formalize this desideratum, we introduce the notion of 'expressiveness'.

Definition 13 *Let $\mathcal{P}$ be a parameter space and ϕ_C a cue function of window size m for it. ϕ_C is expressive iff for every $\overline{P} \in \mathbf{P}$ and every partial assignment $A \in T(\overline{P})$ if $\overline{s} \in L(\overline{P})^i$ (with $i \leq m$) and $\phi_C(\overline{s}, A) = B$ then, for every $< p_i, v_i^m > \in B - A$,*

$$\overline{s} \notin \bigcup_{\overline{P'} \in \mathbf{P}^{A \cup \{< p_i, v_i^m >\}}} L(\overline{P'})^i.$$

Given this definition it is easy to show that expressiveness implies coherence.

Theorem 3 *Let $\mathcal{P}$ be a parameter space and ϕ_C a cue function of window size m for it. If ϕ_C is expressive then it is coherent.*

PROOF. Assume $A \in T(\overline{P})$. Then there is a ϕ_C-trajectory $T = A_1, A_2, \ldots, A_n$ such that $A = \cup A_k$ for some $1 \leq k \leq n$. This implies that there is a $\overline{s'} \in L(\overline{P})^i$ (with $i \leq m$) such that $\phi_C(\overline{s'}, \cup A_{k-1}) = A_k$. Assume $\overline{P} \notin \mathbf{P}^A$. This implies that for $\cup A_{k-1} \in T(\overline{P})$ there is a $\overline{s'} \in L(\overline{P})$ such that $\phi_C(\overline{s'}, \cup A_{k-1}) = A_k$ and, for some $< p_i, v_i^m > \in A_k - \cup A_{k-1}$,

$$\overline{s'} \in \bigcup_{\overline{P'} \in P^{\cup A_{k-1} \cup \{< p_i, v_i^m >\}}} L(\overline{P'})^i.$$

But, by definition 13, this means that ϕ_C is not expressive. □

As a consequence, in order to prove that a parameter space can be learned by a cue-based learner it is sufficient to show the existence of a complete and expressive cue function.

Theorem 4 *Let $\mathcal{P}$ be a parameter space and λ_C a cue-based learner. If the corresponding cue function ϕ_C is complete and expressive, then λ_C identifies every $L(\overline{P}) \in \mathcal{L}(\mathbf{P})$.*

The problem is then to show that there is a property of parameter spaces that is sufficient and necessary for the existence of a complete and expressive cue function and that it is possible to use standard techniques of linguistic analysis to determine whether a parameter space does not have it. In this section we address the first question by showing that such a property exists. We call it the Global Non-Disjunctive Subspace property.

Definition 14 *Let $\mathcal{P}$ be a parameter space and A a partial assignment. $\mathcal{P}[A]$ is said to have the Non-Disjunctive Subspace property up to m (NDS-m) iff if $|\mathbf{P}^A| > 1$ then for every $\overline{P} \in \mathbf{P}^A$ there is an $\overline{s} \in L(\overline{P})^i$ (with $i \leq m$) and an A-consistent partial assignment B such that $\overline{P} \in \mathbf{P}^{A \cup B}$ and, for every $< p_i, v_i^m > \in B - A$,*

$$\overline{s} \notin \bigcup_{\overline{P'} \in \mathbf{P}^{A \cup \{< p_i, v_i^m >\}}} L(\overline{P'})^i.$$

$\mathcal{P}$ has the Global NDS-m (GNDS-m) iff, for every partial assignment A, $\mathcal{P}[A]$ has the NDS-m.

In essence, a parameter space has the GNDS iff for all of its subspaces, each language in the subspace has enough data points to distinguish itself from other languages in terms of *non-disjunctive* parameter assignments (where a *disjunctive* assignment prompted by a linguistic event $\overline{s}$ could be expressed as "event $\overline{s}$ could have been caused either by a language with p_i set to value v_i^m *or* by a language with p_j set to value v_j^k").

The following theorem shows that the GNDS is indeed necessary and sufficient for the existence of a complete and expressive cue function for a parameter space.

Theorem 5 *Let $\mathcal{P}$ be a parameter space. A complete and expressive cue function of window size m ϕ_C for $\mathcal{P}$ exists iff $\mathcal{P}$ has the GNDS-j for some $j \leq m$.*

PROOF.

$\Longrightarrow$ Assume for every $i \leq m$, $\mathcal{P}$ does not have the GNDS-i and ϕ_C is a complete and expressive cue function of window size m for $\mathcal{P}$. Since, for every $i \leq m$, $\mathcal{P}$ does not have the GNDS-i there is a partial assignment A such that $\mathcal{P}[A]$ does not have the NDS-i for every $i \leq m$. Let $\overline{P} \in \mathbf{P}^A$, A' be the partial assignment such that $\overline{P} \in \mathbf{P}^{A \cup A'}$ and $|\mathbf{P}^{A \cup A'}| = 1$, $T = A_1, \ldots, A_n$ a ϕ_C-trajectory for $\overline{P}$ and A_k the first set in T such that $A_k \cap A' \neq \emptyset$. Such a A_k must exist in T since we assumed that ϕ_C is complete. This implies that, for some $j \leq m$, there is an $\overline{s} \in L(\overline{P})^j$ such that $\phi_C(\overline{s}, \cup A_{k-1}) = A_k$. Since, by assumption, ϕ_C is expressive, for every $< p_i, v_i^m > \in A_k - \cup A_{k-1}$

$$\overline{s} \notin \bigcup_{\overline{P'} \in \mathbf{P}^{\cup A_{k-1} \cup \{< p_i, v_i^m >\}}} L(\overline{P'})^j.$$

Also, since by construction $\cup A_{k-1} \subseteq A$, we have that, for every $< p_i, v_i^m > \in A_k - \cup A_{k-1}$ $\mathbf{P}^{\cup A_{k-1} \cup \{< p_i, v_i^m >\}} \supseteq \mathbf{P}^{A \cup \{< p_i, v_i^m >\}}$. This however implies that, for every $< p_i, v_i^m > \in A_k - \cup A_{k-1}$,

$$\overline{s} \notin \bigcup_{\overline{P'} \in \mathbf{P}^{A \cup \{< p_i, v_i^m >\}}} L(\overline{P'})^j.$$

The assumption that $\mathcal{P}[A]$ does not have the NDS-i for every $i \leq m$ implies that, in particular for j, there is a $< p_i, v_i^m > \in A_k - \cup A_{k-1}$ such that

$$\overline{s} \in \bigcup_{\overline{P'} \in \mathbf{P}^{A \cup \{< p_i, v_i^m >\}}} L(\overline{P'})^j$$

hence, a contradiction.

$\Longleftarrow$ Assume $\mathcal{P}$ has the GNDS-m. Then, for every $\overline{P} \in \mathbf{P}$ and every partial assignment A such that $|\mathbf{P}^A| > 1$, the set

$C(A, \overline{P})$, which denotes the set of triples $< \overline{s}, A, B >$ such that $\overline{P} \in \mathbf{P}^A$, $\overline{s} \in L(\overline{P})^m$ and

$$\overline{s} \notin \bigcup_{\overline{P} \in \mathbf{P}^{A \cup \{< p_i, v_i^m >\}}} L(\overline{P})^m \text{ for all } < p_i, v_i^m > \in B - A\}$$

is not empty. Moreover, if B^+ is the subset of B^* such that, if $A \in B^+$ then $|\mathbf{P}^A| > 1$,

$$C(\overline{P}) = \bigcup_{A \in B^+} C(A, \overline{P}) \text{ and } K = \bigcup_{\overline{P} \in \mathbf{P}} C(\overline{P}),$$

then K is the graph of a complete and coherent cue function ϕ_C for $\mathcal{P}$.

To show that ϕ_C is complete, it is sufficient to show that, for every $\overline{P} \in \mathbf{P}$ and every ϕ_C-trajectory $A_0, \ldots, A_n$ for $\overline{P}$

$$a \in \bigcup_{i=0}^{n} A_i \text{ iff } a \in \{< p_1 \pi_1(\overline{P}) >, \ldots, < p_n \pi_n(\overline{P}) >\}.$$

But this follows from the fact that for every sequence $< \overline{s_1}, A_0, A_1 >, < \overline{s_2}, A_1, A_2 >, \ldots < \overline{s_n}, A_{n-1}, A_n >$ of members of $C(\overline{P})$ such that $A_0 = \emptyset$,

$$\bigcup_{i=0}^{n} A_i = \{< p_1, \pi_1(\overline{P}) >, \ldots < p_n, \pi_n(\overline{P}) >\}$$

(since, by hypothesis, $C(A, \overline{P}) \neq \emptyset$ for every A such that $|\mathbf{P}^A| > 1$) and the fact that the corresponding sequence $A_0, A_1, \ldots, A_n$ is by definition 8 a ϕ_C-trajectory for $\overline{P}$.

To show that ϕ_C is expressive, take any $\overline{P}$, $A \in T(\overline{P})$ and $\overline{s} \in L(\overline{P})^m$ such that $\phi_C(\overline{s}, A) = B$. By construction of the graph of ϕ_C $< \overline{s}, A, B > \in K$ and so, for some $\overline{P'}$, $< \overline{s}, A, B > \in C(A, \overline{P'})$. But, by construction, again, if $< \overline{s}, A, B > \in C(A, \overline{P'})$ then, for every $< p_i, v_i^m > \in B - A$,

$$\overline{s} \notin \bigcup_{\overline{P} \in \mathbf{P}^{A \cup \{< p_i, v_i^m >\}}} L(\overline{P})^m.$$

□

Expressing learnability conditions in a linguistically meaningful vocabulary

The existence of a characterizing condition for the cue-based learnability of a parameter space is not in itself an exciting result if it can only be tested by inspecting the set theoretical relationships among all languages generated by the space. It turns out, however, that in order to prove a negative learnability result one can simply verify whether the parameter space under scrutiny possesses a feature that can be stated in a vocabulary that is meaningful from the point of view of linguistic theory.

In particular, it is possible to show that every parameter space that includes clusters of two or more languages that are weakly equivalent to one another (that is, that generate exactly the same set of strings, whatever the internal structure assigned to them) does not have the GNDS.

Theorem 6 *Let $\mathcal{P}$ be a parameter space with $\overline{P'}, \overline{P''} \in \mathbf{P}$ such that $L(\overline{P'}) \supseteq L(\overline{P''})$. Then $\mathcal{P}$ does not have the GNDS-m for any m.*

PROOF. Let $A = \{< p_{i_1} \pi_{i_1}(\overline{P'}) >, \ldots < p_{i_m} \pi_{i_m}(\overline{P'}) >\}$ be the largest set such that for all $1 \leq k \leq m$, $\pi_{i_k}(\overline{P'}) = \pi_{i_k}(\overline{P''})$. Then $\overline{P'}, \overline{P''} \in \mathbf{P}^A$. If, for some m, $\mathcal{P}$ has the GNDS-m, then every partial assignment has the NSP-m, and so, in particular, A has the NSP-m. This implies that for every $\overline{P} \in \mathbf{P}^A$, and so, in particular for $\overline{P'}$ there is a partial assignment B and a $\overline{s'} \in L(\overline{P'})^j$ (with $j \leq m$) such that, for every $< p_i, v_i^m > \in B - A$,

$$\overline{s'} \notin \bigcup_{\overline{P} \in \mathbf{P}^{A \cup \{< p_i, v_i^m >\}}} L(\overline{P})^j.$$

However, by hypothesis, every member of $L(\overline{P'})$ is also a member of $L(\overline{P''})$ and since $\overline{P''} \in \mathbf{P}^{A \cup B}$ it follows that, for every $< p_i, v_i^m > \in B - A$,

$$\overline{s'} \in \bigcup_{\overline{P} \in \mathbf{P}^{A \cup \{< p_i, v_i^m >\}}} L(\overline{P})^j$$

hence a contradiction. □

A direct consequence of theorem 6 and theorem 5 is that once one is able to show that a system of parameters generates at least a couple of languages that are either weakly equivalent or properly include one another, then one has a proof that no expressive cue-based learner exists for the space.

Proving that a system of parameters generates at least a couple of weakly equivalent languages is often fairly easy to do using standard linguistic analyses. For example, it is easy to show that two languages are weakly equivalent if they only different in the value of a parameter that requires or blocks certain kinds of movement in a context that (due possibly to the setting of some other parameters) makes such movements string vacuous. Examples of this kind are endemic in the linguistic literature on parametric variation. For example, in Wu's [18] space of sixteen syntactic and morphological parameters, several languages can be generated by as many as twenty alternative parameter settings. It could be thought that such massive underdetermination could have been an artifact of the relatively small set of data considered by Wu for each language. In Bertolo et al. [2], however, we discuss a space of syntactic parameters where clusters of weakly equivalent languages are still present even if each language is represented by sets of data that are several hundreds times larger than those considered by Wu.

Likewise, a prima facie argument for the existence of actual linguistic systems that contain couples of languages that stand in the subset/superset relationship can be made consulting the vast body of linguistic literature on the so called *Subset Principle* (Rizzi [13], Berwick [4], Wexler and Manzini [17]. But see MacLaughlin [10] for a critical review of these studies.)

Finally, it is important to stress that the existence in a parameter space of clusters of weakly equivalent languages, although sufficient, is by no means necessary for a space to lack the GNDS. Consider for example the parameter space $\mathcal{P}$=$< \{\{0,1\}, \{0,1\}, \{0,1\}, L\{0,1\}\} >$ where each language $L(\overline{P})$ is constructed by *diagonalization* and consists of all the members of $\mathbf{P}$ with the exception of $\overline{P}$ itself as it is shown in the table below.

$\overline{P}$	$L(\overline{P})$							
000		001	010	011	100	101	110	111
001	000		010	011	100	101	110	111
010	000	001		011	100	101	110	111
011	000	001	010		100	101	110	111
100	000	001	010	011		101	110	111
101	000	001	010	011	100		110	111
110	000	001	010	011	100	101		111
111	000	001	010	011	100	101	110	

It is easy to verify that no two languages are equivalent to one another. It is also easy to verify that, whatever the choice of value for, say, p_1, there is no data point in, say, $L(< 000 >)$, that cannot be found also in some of the languages that have p_1 set to the alternative value. Therefore, $\mathcal{P}$ itself does not have the NSP-1 and so, a fortiori, it does not have the GNDS-1. It is also easy to verify that the smallest number k such that $\mathcal{P}$ has GNDS-k is 4.

For a proof that parametric systems of *infinite* languages that are based on the diagonalization construction can indeed be generated by parametric classes of context free grammars, consider the following parameter space $\mathcal{P}=<\{\{0,1\},\{0,1\}\},L,G,\{a,b,g\}>$ where the function L is defined as follows.

$\overline{P}$	$L(\overline{P})$						
00			abg^*	$\cup$	bag^*	$\cup$	bbg^*
01	aag^*			$\cup$	bag^*	$\cup$	bbg^*
10	aag^*	$\cup$	abg^*			$\cup$	bbg^*
11	aag^*	$\cup$	abg^*	$\cup$	bag^*		

A universal part of the grammar, common to all languages, can be represented by the following set of rules:

$$UG = \left\{ \begin{array}{lll} G \rightarrow gG & G \rightarrow \epsilon & B \rightarrow ba \\ C \rightarrow bb & D \rightarrow aa & E \rightarrow ab \end{array} \right\}$$

The two parameters are then:

$$p_1 = \left\{ \; 0 = \left\{ \begin{array}{l} S \rightarrow AG \\ S \rightarrow BG \\ S \rightarrow CG \end{array} \right\} 1 = \left\{ \begin{array}{l} S \rightarrow DG \\ S \rightarrow EG \\ S \rightarrow FG \end{array} \right\} \right\}$$

and

$$p_2 = \left\{ \; 0 = \left\{ \begin{array}{l} A \rightarrow ab \\ F \rightarrow bb \end{array} \right\} 1 = \left\{ \begin{array}{l} A \rightarrow aa \\ F \rightarrow ba \end{array} \right\} \right\}$$

This would seem to indicate that the existence of spaces lacking the GNDS is not just a mathematical curiosity but something a linguistic theory of variation has to treat as a distinct possibility.

This example is particularly instructive because it shows that a simpleminded repair strategy that a cue-based learner could adopt upon encountering a subspace that does not have the NSP, that is, arbitrarily choosing a possible partition of the subspace, is not guaranteed to work in general. In particular, a learner that adopted such a strategy in the parameter space of the example above by first guessing the value of the first parameter, then the value of the second and so on selecting the two alternative values with equal probability would have probability at most 0.25 of selecting the target grammar.[3] This example is therefore helpful to provide an additional motivation for the restriction on cue function imposed by expressiveness. Although expressiveness may not be necessary for learnability when the lack of GNDS in the space is due solely to clusters of weakly equivalent languages, it certainly is when it is not.

Conclusions

In this paper we have provided a formal characterization of a broad class of learning algorithms for parametric space systems that have been advocated in one form or another by linguists and psycholinguists due to their search efficiency in a parametric hypothesis space.

We have shown that, as it was to be expected, such efficiency comes at a cost. Specifically, we have shown that cue-based learners can be successful (and efficiently so, in that case), only if the parametric class of languages that they are supposed to learn enjoys certain set theoretical properties. In particular, we have shown how these properties are related to the size of the memory window the learner can rely on.

Finally, we have shown that, it is always possible to verify 'locally' the application of our general negative learnability result to a particular parameter space. In other words, given a parameter space that cannot be learned by any cue-based learner of a certain window size, one only needs to inspect a subset of all the languages in the class to find out that this is the case. In addition, the conditions for the 'local' application of the negative learnability result can often be stated in a vocabulary that is descriptively meaningful in terms of current linguistic theories of parametric variation.

All the results of this paper depend on the assumption almost universally shared by developmental psycholinguists (see Marcus [11] for a review of the arguments in support of this conclusion) that children cannot rely on systematic negative evidence for language learning. However, we have shown elsewhere (Bertolo et al. [3]) that the results generalize straightforwardly to the case in which cue-based learners receive systematically both positive and negative evidence.

In closing, we wish to list a few questions that can be fruitfully investigated within the model we have proposed.

1. As it has been pointed out repeatedly, what causes cue-based learners to fail on parameter spaces that have a certain structure is their inability to backtrack in the hypotesis space. There are, however, several ways to search efficiently the hypothesis space while keeping a modicum of backtracking. One such way would be to start with some 'default' values for some or all of the parameters and revise them in response to cues (as it is done, for example, in Dresher and Kaye [7]). In that case, the characterizing

[3] Suppose the target language is $L(< 000 >)$. As noted before, no data point could reveal the value of p_1. Choosing randomly the learner has 0.5 probability of choosing the correct value 0. The same is true for parameter p_2, which gives the learner a 0.25 probability of choosing the correct values for both p_1 and p_2. If these two values are correctly retrieved, however, the learner could rely on a conditional form of learning, that is allowed by definition 4 and 'reason' as follows: if the current assignment is $< 00 >$ then observing 001, which, at this point, is unique to $L(< 000 >)$ assigns to p_3 value 0. Similarly, observing 000 assigns to p_3 value 1.

condition for successful learning is going to be different from the one we have provided.

2. Our criterion of success requires learners to yield a complete parameter assignment as their last conjecture. It is conceivable that for many linguistic systems this may not be necessary and that, in some sense, a 'partial' conjecture may be equivalent to many possible ways to complete it. A study of how the success criterion may be relaxed in such cases will require a more fine grained analysis of the properties of the function that maps parameter assignments to languages. Some preliminar results on this problem can be found in Bertolo et al [3].
3. The example by means of which we demonstrate the possibility of implementing a 'diagonal' construction of a parameter space by means of context free grammars reveals that there is nothing intrinsically 'parametric' about the GNDS property that we used to establish our main result. In other words, whenever one specifies a class of grammars (whether finite or infinite) in terms of distinct union sets of sets of rules, it should always be possible to determine whether, for the purpose of efficient learning, certain (sequences of) data can be used to acquire certain (sets of) rules. The same analysis of 'non-backtracking' learning algorithms could then be applied also to grammatical systems such as those investigated by Stabler [15], where the number of possible human languages is not limited in principle as it is in the parametric case.

Acknowledgments

Stefano Bertolo was supported by a post-doctoral fellowship from the McDonnell-Pew foundation, Kevin Broihier (grant DIR 9113607) and Edward Gibson and Kenneth Wexler (grant SBR-9511167) were supported by grants from NSF. We wish to thank Janet Fodor and Ed Stabler for valuable comments on earlier versions of this work.

References

[1] Stefano Bertolo. *Learnability properties of parametric models for natural language acquisition.* PhD thesis, Rutgers University, 1995.

[2] Stefano Bertolo, Kevin Brohier, Edward Gibson, and Kenneth Wexler. Cue-based learners in parametric language systems: applications of general results to a recently proposed learning algorithm based on unambiguous 'superparsing'. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*, August 1997.

[3] Stefano Bertolo, Kevin Brohier, Edward Gibson, and Kenneth Wexler. Cue-based learners in parametric language systems. MIT manuscript. Available at http://www-bcs.mit.edu/~bertolo, August 1997.

[4] Robert Berwick. *The Acquisition of Syntactic Knowledge.* MIT Press, Cambridge, MA, 1985.

[5] Noam Chomsky. *Lectures on Government and Binding.* Foris Publications, Dordrecht, 1981.

[6] Robin Clark. The selection of syntactic knowledge. *Language Acquisition*, 2(2):83–149, 1992.

[7] Elan Dresher and Jonathan D. Kaye. A computational learning model for metrical phonology. *Cognition*, 34:137–195, 1990.

[8] Janet Fodor. Unambiguous triggers. *Linguistic Inquiry.*

[9] Mark E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[10] Dawn MacLaughlin. Language acquisition and the subset principle. *The Linguistic Review*, 12:143–191, 1995.

[11] Gary Marcus. Negative evidence in language acquisition. *Cognition*, 46(1):53–85, 1993.

[12] Daniel Osherson, Michael Stob, and Scott Weinstein. *Systems that learn.* MIT Press, Cambridge, MA, 1986.

[13] Luigi Rizzi. Violations of the wh island constraint and the subjacency condition. In Luigi Rizzi, editor, *Issues in Italian Syntax.* Foris, Dordrecht, 1982.

[14] Ian Roberts. Language change and learnability. In Stefano Bertolo, editor, *Learnability and Language Acquisition: a self contained Tutorial for Linguists.* MIT Manuscript, 1996.

[15] Edward Stabler. Acquiring and parsing languages with movement. Ms. UCLA, 1996.

[16] Kenneth Wexler and Peter Culicover. *Formal Principles of Language Acquisition.* MIT Press, Cambridge, MA, 1980.

[17] Kenneth Wexler and Rita Manzini. Parameters and learnability in binding theory. In Thomas Roeper and Edwin Williams, editors, *Parameter Setting*, pages 41–76. Reidel, Dordrecht, 1987.

[18] Andi Wu. *The Spell-Out Parameters: A Minimalist Approach to Syntax.* PhD thesis, UCLA, 1994.

Evaluating Parsing Schemes with Entropy Indicators

Caroline Lyon
Computer Science Department
University of Hertfordshire
Hatfield AL10 9AB, UK
comrcml@herts.ac.uk

1 Introduction

This paper introduces an objective metric for assessing the effectiveness of a parsing scheme. Information theoretic indicators can be used to show whether a given scheme captures some of the structure of natural language text. We then use this method to support a proposal to decompose the parsing task into computationally more tractable subtasks.

The principle on which the grammar evaluator is based is derived from Shannon's original work with letter sequences [1]. We show how his ideas can be extended to other linguistic entities. We describe a method of representation that enables the entropy of sentences to be measured under different parsing schemes. The entropy is a measure, in a certain sense, of the degree of unpredictability. If the grammar captures some of the structure of language, then the relative entropy of the text should decline after parsing. We can thus objectively assess whether parsers that accord with some linguistic intuition do indeed capture some regularity in natural language.

Natural language can be seen as having a tertiary structure. First, there are the relationships between adjacent words, a structure that can be modelled by Markov processes. Then words can be grouped together into constituents and these constituents are organized in a secondary structure. Thirdly, there are relationships between elements of constituents, such as the agreement between the head of a subject and the main verb. These 3 levels are compatible with levels in the Chomsky hierarchy.

We need to integrate natural language processing at these different levels. The work described in this paper uses a method of representation that enables primary and secondary structure to be modelled jointly. It concludes by suggesting how this approach could facilitate processing at levels 2 and 3.

The paper is organized in the following way. First, we recall Shannon's original work with letter sequences. Then we describe a method of adapting his approach to word sequences. Next, we show how this is not an adequate model for natural language sentences, but can be extended. Using the new representation we can model syntactic constituents, and parsing a sentence is taken to be finding their location. Then we show how the entropy of parsed and unparsed sentences is measured. If the entropy declines after parsing, this indicates that some of the structure has been captuered.

Finally, we apply this entropy evaluator to show that one particular parsing method effectively decomposes declarative sentences into 3 sections. These sections can be partially parsed separately, in parallel, thus reducing the complexity of the parsing task.

2 Shannon's work with letter sequences

Shannon's well known work on characteristics of the English language examined the entropy of letter sequences. He produced a series of approximations to the entropy H of written English, which successively take more of the statistics of the language into account

H_0 represents the average number of bits required to determine a letter with no statistical information. H_1 is calculated with information on single letter frequencies; H_2 uses information on the probability of 2 letters occurring together; H_n, called the n-gram entropy, measures the amount of entropy with information extending over n adjacent letters of text. As n increases from 0 to 3, the n-gram entropy declines: the degree of predictability is increased as information from more adjacent letters is taken into account. If $n-1$ letters are known, H_n is the conditional entropy of the next letter, and is defined as follows.

b_i is a block of $n-1$ letters, j is an arbitrary letter following b_i

$p(b_i, j)$ is the probability of the n-gram b_i, j

$p_{b_i}(j)$ is the conditional probability of letter j after block b_i, that is $p(b_i, j) \div p(b_i)$

$$\begin{aligned} H_n &= -\sum_{i,j} p(b_i, j) * log_2 p_{b_i}(j) \\ &= -\sum_{i,j} p(b_i, j) * log_2 p(b_i, j) + \sum_{i,j} p(b_i, j) * log_2 p(b_i) \\ &= -\sum_{i,j} p(b_i, j) * log_2 p(b_i, j) + \sum_{i} p(b_i) * log_2 p(b_i) \end{aligned}$$

since $\sum_{i,j} p(b_i, j) = \sum_i p(b_i)$

An account of this process can also be found in [2].

The entropy can be reduced if an extra character representing a space between words is introduced. Let H' represent the entropy measures of the 27 letter alphabet. Then, if $n > 0$, $H'_n < H_n$. By introducing an extra element, the number of choices has increased, so $H'_0 > H_0$. The space will be more common than other characters , so $H'_1 < H_1$. Where $n > 1$ the statistical relationships of neighbouring elements are taken into account. Shannon says "a word is a cohesive group of letters with strong internal statistical influences" so the introduction of the space has captured some of the structure of the letter sequence.

	H_0	H_1	H_2	H_3
26 letter	4.70	4.14	3.56	3.3
27 letter	4.76	4.03	3.32	3.1

Table 1: Comparison of entropy for different n-grams, with and without representing the space between words

3 Representing parsed and unparsed text

This type of analysis can be applied to strings of words instead of strings of letters. In order to make this approach computationally feasible we need to partition an indefinitely large vocabulary into a limited number of part-of-speech classes. By taking this step we loose much information: the process is not reversible. However, we aim to retain the information that is needed for one stage of processing, and return later to the actual words at a later stage.

Sometimes, the allocation of part-of-speech tags has been considered a step in parsing. However, we are looking for syntactic structure and call the strings of tags the unparsed text.

Now, at the primary level text can be modelled as a sequence of tags, and Shannon's type of analysis can be extended to word sequences. Punctuation marks can also be mapped onto tags. An experiment with the LOB corpus showed that for sequences of parts-of-speech tags H_2 and H_3 are usually slightly lower if punctuation is included in an enlarged tagset.

However, there is more structural information to be extracted. Our linguistic intuition suggests that there are constituents, cohesive groups of words with internal statistical influences. The entropy indicator will show objectively whether this intuition is well founded.

Furthermore, the statistical patterns of tag sequences can be disrupted at the boundaries of constituents. Consider the probability of part-of-speech tags following each other: some combinations are "unlikely", such as *noun - pronoun* and *verb - auxiliary verb* but they may occur at clause and phrase boundaries in sentences like "The shirt he wants is in the wash.".

An important step extends the representation to handle this. The embedded clause is delimited by inserting boundary markers, or hypertags, like virtual punctuation marks. We represent the sentence as

The shirt { he wants } is in the wash.

The pairs generated by this string would exclude *noun - pronoun*, but include, for instance, *noun - hypertag1*. The part-of-speech tags have probabilistic relationships with the hypertags in the same way that they do with each other. We can measure the entropy of the sequence with the opening and closing hypertags included. If their insertion has captured some of the structure the bipos and tripos entropy should be reduced.

Each class of syntactic elements has a distinct pair of hypertags. Applying automated parsers, one type of syntactic element is found at a time. In this particular case of locating an embedded clause, the insertion of hypertags can be seen as representing "push" and "pop" commands. One level of embedding has been replaced.

4 Entropy measures

We apply the theory outlined above to a corpus of text, taken from engine maintenance manuals. We propose different structural markers, and measure the resulting entropy. Note that the absolute entropy levels depend on a number of variable factors. We are interested in comparative levels, and thus use the term *entropy indicators*.

There is a relationship between tagset size, distribution of tags, number of samples and entropy. For instance, as tagset size is decreased entropy declines, but at the same time grammatical information may be lost. We have to balance the requirement for a small tagset against the need to represent separately each part-of-speech with distinct syntactic behaviour. Another approach to entropy reduction, which would not be helpful, is to expand one element into several that always, or usually, occur together. For instance, we

could reduce the entropy by mapping every instance of *determiner* onto *hypertag1 determiner hypertag2.*

We use linguistic intuition to propose constituents, substrings of tags with certain characteristics that suggest they should be grouped together. Then we investigate the entropy levels of tagged text for the following cases

1. No hypertags (suffix p: plain)
2. Hypertags before and after determiners (suffix d)
3. Arbitrarily placed hypertags: in each sentence before tag position 2, after tag position 5 (suffix a)
4. Hypertags delimiting noun groups (suffix n)
5. Hypertags delimiting subject (suffix s)
6. Hypertags delimiting subject and noun groups (suffix sn)

A noun group is taken to be a noun immediately preceded by an optional number of modifiers, such as "mechanical stop lever" or just "lever".

Results

The data consisted of 351 declarative sentences from manuals from Perkins Engines Ltd. Average sentence length is 18 words, counting punctuation marks as words. The tagset had 32 members, including 4 hypertags. H_0 is 5. Using automated parsers previously developed, the data was prepared automatically, but then manually checked. A summary of results obtained is given in Table 2.

	text	H_1	H_2	H_3
1	text-p	3.962	2.659	2.132
2	text-d	4.086	2.123	1.722
3	text-a	4.135	2.689	2.077
4	text-n	3.981	2.038	1.682
5	text-s	4.135	2.472	1.997
6	text-sn	4.142	1.943	1.612

Table 2: Entropy measures for text with different structural markers

For interest, some text from Shannon's article was also processed in the same way, and produced results in line with these.

Recall that we are interested in the movement of the entropy measure, and do not claim to attach significance to the absolute values. We ask a question with a "yes" or "no" answer: does the entropy decline when the parsing scheme is applied. However, note the results of 6, which combines schemes 4 and 5, that is marking both the noun groups and the subject. We see that the decline in entropy H_2 and H_3 is greater than for either scheme separately.

5 Applying these results to decompose the parsing task

Consider the parser numbers 5 that locates the subject of a sentence. In the corpus used the length of the subject varied from 1 to 12 words, the length of the pre-subject from 0 to 15 words.

As an example of subject location consider a sentence from Shannon's paper which would be represented as

> If the language is translated into binary digits in the most efficient way , { the entropy } is the average number of binary digits required per letter of the original language .

Comparing lines 1 and 5 of Table 2, we see H_2 and H_3 decline for parsed text, so we have captured some of the structure.

Now, locating the subject effectively decomposes a declarative sentence into 3 sections:

pre-subject - subject - predicate.

Of course the first section can be empty. Imperative sentences can also be processed in this way, the lack of an explicit subject being represented by an empty subject section. An automated parser that finds the subject, and thus decomposes the sentence, has already been developed. A prototype is available via telnet for users to try with their own text, and is described in [3, 4].

On examining these concatenated sections we note that other constituents are contained within them and do not cross the boundaries between them. An element or constituent in one section can have dependent links to elements in other sections, such as agreement between the head of the subject and the main verb. However, the constituents themselves - clauses, phrases, noun groups - are contained within one section. Therefore, once the 3 sections have been located, they can then be partially processed separately, in parallel. The complexity of the parsing task can be reduced by decomposing a declarative sentence as a preliminary move.

6 Conclusion

We have shown that entropy indicators can be used to support parsing schemes based on linguistic intuition.

In particular, the entropy indicator supports the decompostion of a sentence into 3 concatenated segments that can be partially processed separately. Since many automatic parsers have difficulty processing longer sentences, we suggest that this decomposition could facilitate the operation of other systems.

References

[1] C E Shannon. Prediction and Entropy of Printed English. *Bell System Technical Journal*, pages 50–64, 1951.

[2] T M Cover and J A Thomas. *Elements of Information Theory.* John Wiley and Sons Inc., 1991.

[3] C Lyon and R Frank. Neural network design for a natural language parser. In *International Conference on Artificial Neural Networks (ICANN)*, pages 105–110, 1995.

[4] C Lyon and R Frank. Using single layer networks for discrete, sequential data: an example from natural language processing. *Neural Computing Applications*, To appear.

Emptiness, Membership and Regular Expressions for Tree Homomorphic Feature Structure Grammars

*Short version**

Tore Burheim†

Abstract

Tree Homomorphic Feature Structure Grammar is a feature structure grammar formalism based on Lexical-Functional Grammar (LFG). It has a strong restriction on the syntax of the equation schemata, but does not have the off-line parsability constraint. In this paper we use modal logic to show that the emptiness and membership problems are decidable for this grammar formalism. We also show that we may allow regular expressions in the feature structure equations without changing the class of languages described.

1 Introduction and some definitions

Feature structure grammars are widely used in computational linguistics. They are grammar formalisms that use feature structures as (one of) their main data structure(s), sometime together with some phrase structure backbone. Feature structures and the way they may be specified are very flexible, but this flexibility has its drawback: the formalisms become almost too powerful in the sense that the membership problem for these grammars in their most general form is undecidable. Restrictions are an answer to this problem. The *off-line parsability constraint* is such a restriction which makes Lexical-Functional Grammar (LFG) [8] decidable. The off-line parsability constraint says that one is not allowed to have non-branching chains in the phrase-structure tree where one category is repeated, or a branching chain with a repeated category but where all the branches between only yield the empty string. However there are other restrictions we may impose to make LFG-like grammar formalisms decidable.

Tree Homomorphic Feature Structure Grammar was introduced in [3]. It is based on Lexical-Functional Grammar (LFG) [8, 5] and work by Colban [4]. The formalism has a context-free phrase structure backbone and adds equations to the nodes in the phrase-structure tree as LFG. These equations describe feature structures. In the formal framework there are two main differences from LFG: First, due to a restriction that is imposed on the equations in the grammar, the referred part of the feature structure is a tree which includes a homomorphic image of the phrase structure tree. Then, with this restriction we do not need the off-line parsability constraint to make the grammar decidable.

1.1 Tree Homomorphic Feature Structure Grammar

In this section we will give a brief introduction to Tree Homomorphic Feature Structure Grammar (THFSG). A broader presentation may be found in [3].

A *feature structure* over a set of attribute symbols $\mathcal{A}$ and value symbols $\mathcal{V}$ is a four-tuple $\langle Q, f_D, \delta, \theta \rangle$ where Q is a finite set of nodes, $f_D : D \to Q$ is a function, called the name mapping, $\delta : Q \times \mathcal{A} \to Q$ is a partial function, called the transition function, and $\theta : Q \to \mathcal{V}$ is a partial function called the atomic value function. We omit the name-domain from the notation, so f will alone denote the name mapping. We extend the transition function to be a

*A full version of this article may be found at The Computation and Language E-Print Archive, `http://xxx.lanl.gov/cmp-lg/` after August 25, 1997.

†Telenor Research and Development, P.O.Box 83, N-2007 Kjeller, Norway. `Tore.Burheim@fou.telenor.no`

function from pairs of nodes and *strings* of attribute symbols as usual.

A feature structure is *describable* if there for every $q \in Q$ is an $x \in D$ and a $u \in \mathcal{A}^*$ such that $\delta(f(x), u) = q$. A feature structure is *atomic* if for every node $q \in Q$: if $\theta(q)$ is defined then is $\delta(q, a)$ not defined for any $a \in \mathcal{A}$. A feature structure is *acyclic* if for every node $q \in Q$, $\delta(q, u) = q$ if and only if $u = \varepsilon$. A feature structure is *well defined* if it is describable, atomic and acyclic.

We will use equations to talk about feature structures, such that a feature structure may or may not satisfy each equation. A feature structure *satisfies* the equation

$$x_1 u_1 \doteq x_2 \tag{1}$$

if and only if $\delta(f(x_1), u_1) = f(x_2)$, and the equation

$$x_3 u_3 \doteq v \tag{2}$$

if and only if $\alpha(\delta(f(x_3), u_3)) = v$, where $x_1, x_2, x_3 \in D$, $u_1, u_3 \in \mathcal{A}^*$ and $v \in \mathcal{V}$. We will call equations like (1) *path equations* and equations like (2) *value equations*. These path and value equations are the only kind of equations we use in THFSG. If E is a set of equations and M is a well defined feature structure such that M satisfies every equation in E we say that M *satisfies* E and we write $M \models E$.

A *Tree Homomorphic Feature Structure Grammar*, THFSG, is a 5-tuple $\langle \mathcal{K}, \mathcal{S}, \Sigma, \mathcal{P}, \mathcal{L} \rangle$ over the set of attribute symbols $\mathcal{A}$ and value symbols $\mathcal{V}$ where $\mathcal{K}$ is a finite set of symbols, called *categories*, $\mathcal{S} \in \mathcal{K}$ is a symbol, called *start symbol*, and Σ is a finite set of symbols, called *terminals*. Further more, $\mathcal{P}$ is a finite set of *production rules*

$$\begin{array}{lcccc} A_0 & \rightarrow & A_1 & \ldots & A_m \\ & & E_1 & & E_m \end{array} \tag{3}$$

where $m \geq 1$, $A_0, \ldots, A_m \in \mathcal{K}$, and for all i, $1 \leq i \leq m$, E_i is a finite set consisting of one and only one equation schema of the form $\uparrow u_1 \doteq \downarrow$ where $u_1 \in \mathcal{A}^*$, and a finite number of equation schemata of the form $\uparrow u_2 \doteq v$ where $u_2 \in \mathcal{A}^+$ and $v \in \mathcal{V}$. At last, $\mathcal{L}$ is a finite set of *lexicon rules*

$$\begin{array}{lcc} A & \rightarrow & t \\ & & E \end{array} \tag{4}$$

where $A \in \mathcal{K}$, $t \in (\Sigma \cup \{\varepsilon\})$, and E is a finite set of equation schemata of the form $\uparrow u_3 \doteq v$ where $u_3 \in \mathcal{A}^+$ and $v \in \mathcal{V}$. The sets $\mathcal{K}$ and Σ are required to be disjoint.

To define constituent structures we use tree domains. Let $\mathcal{N}_+$ be the set of all integers greater than zero. A *tree domain* D is a set $D \subseteq \mathcal{N}_+^*$ of number strings such that if $x \in D$ then all prefixes of x are also in D, and for all $i \in \mathcal{N}_+$ and $x \in \mathcal{N}_+^*$, if $x \cdot i \in D$ then $x \cdot j \in D$ for all j, $1 \leq j < i$. Since clean strings over numerals may be ambiguous we use the sign $\cdot$ to indicate juxtaposition, as in $2 \cdot 11 \cdot 3$. The *out degree* $d(x)$ of an element x in a tree domain D is the cardinality of the set $\{i \mid x \cdot i \in D, i \in \mathcal{N}_+\}$. The set of terminals of D is $term(D) = \{x \mid x \in D, d(x) = 0\}$. The elements of a tree domain are totally ordered lexicographically as follows: $x' \prec x$ if x' is a proper prefix of x, or there exist strings $y, z, z' \in \mathcal{N}_+^*$ and $i, j \in \mathcal{N}_+$ with $i < j$, such that $x' = y \cdot i \cdot z'$ and $x = y \cdot j \cdot z$. A tree domain may be infinite, but we restrict attention to finite tree domains.

A *constituent structure* (c-structure) based on a THFSG-grammar $G = \langle \mathcal{K}, \mathcal{S}, \Sigma, \mathcal{P}, \mathcal{L} \rangle$ is a triple $\langle D, C, E \rangle$ where

- D is a finite tree domain,
- $C : D \rightarrow (\mathcal{K} \cup \Sigma \cup \{\varepsilon\})$ is a function,
- $E : (D - \{\varepsilon\}) \rightarrow \Gamma$ is a function where Γ is the set of all sets of equation schemata in G,

such that $C(x) \in (\Sigma \cup \{\varepsilon\})$ for all $x \in term(D)$, $C(\varepsilon) = \mathcal{S}$, and for all $x \in (D - term(D))$, if $d(x) = m$ then

$$\begin{array}{lcccc} C(x) & \rightarrow & C(x \cdot 1) & \ldots & C(x \cdot m) \\ & & E(x \cdot 1) & & E(x \cdot m) \end{array} \tag{5}$$

is a production or lexicon rule in G, and we say that $license(x)$ is the given rule in (5). The *terminal string* of a constituent structure is the string $C(x_1) \ldots C(x_n)$ such that $\{x_1, \ldots, x_n\} = term(D)$ and $x_i \prec x_{i+1}$ *for all* i, $1 \leq i < n$.

A c-structure $c = \langle D, C, E \rangle$ is *feature consistent* if and only if there exist a well defined feature structure M such that

$$M \models \bigcup_{x \in (D - \{\varepsilon\})} E_{\mathcal{M}}(x) \tag{6}$$

where the $E_{\mathcal{M}}$ is defined as

$$E_{\mathcal{M}}(x \cdot i) = E(x \cdot i)[x/\uparrow, x \cdot i/\downarrow] \tag{7}$$

A string is grammatical with respect to a given grammar if it is the terminal string in a feature consistent c-structure based on that grammar. The set of grammatical strings with respect to a given THFSG grammar G is as usual denoted $L(G)$. Two grammars G and G' are weakly equivalent if and only if $L(G) = L(G')$.

2 Emptiness and membership for THFSG

2.1 Feature productive THFSG

In this section we introduce THFSG which do not contain any $\uparrow\doteq\downarrow$ equation schemata. The reason that we do not want these equations is that we do not want any of the descendants of any node in a c-structure to be mapped to the same node in the feature structure as the node itself. We need this property in the proof of emptiness decidability. We will here show that given any THFSG-grammar we may effective[1] transform this to a THFSG-grammar with the given property such that the language defined by original grammar is non-empty if and only if the language defined by the new one is also non-empty.

Definition 1 (Feature productive THFSG) *A feature productive* [2] THFSG *grammar is a* THFSG *grammar that does not contain any* $\uparrow\doteq\downarrow$ *equation schemata.*

Lemma 1 *There exists an effective procedure which for any* THFSG *grammar G defines a feature productive* THFSG *grammar G' such that $L(G) = \emptyset$ if and only if $L(G') = \emptyset$.*

[1] By "effective" we mean that it can be computed on a Turing Machine (with a proven termination) [7, p146-147]. This must not be confused with polynomial time deterministic algorithms.

[2] The feature productive THFSG is almost identical to nc-LFG defined by Seki et.al. [11]. The differences are that each equation schema in nc-LFG has an attribute string consisting of one and only one attribute symbol, and nc-LFG does not allow the empty string on the right hand side in the phrase structure rules. In their article Seki et.al. prove that nc-LFG is equivalent to finite state translation system based on tree transducer as defined in [10] in the sense that it describes the same class of languages. The emptiness problem is decidable for the yield language of finite state translation system based on tree transducer. It is possible that the decidability of the emptiness problem for feature productive THFSG could be proved using this path.

Proof: Let us first introduce a new notation for the production and lexicon rules. Let

$$\begin{array}{lcccc} A_0 & \rightarrow & A_1 & \ldots & A_n \\ & & E_1 & & E_n \end{array} \tag{8}$$

be any production rule. Then $(A_0, E) \rightarrow (A_1, e_1) \ldots (A_n, e_n)$ is the rule in (8) on *flat format.* Here E is the set of all value equation schemata in $E_1 \cup \ldots \cup E_n$, and each e_i is the path equation schema in E_i, $1 \leq i \leq n$. We do the same for lexicon rules, which give us rules on the format $(A, E) \rightarrow t$, where E is the set of (value) equation schemata in the rule.

As a first step, let G be any THFSG grammar $\langle \mathcal{K}, \mathcal{S}, \Sigma, \mathcal{P}, \mathcal{L} \rangle$. We define $G_\varepsilon = \langle \mathcal{K}, \mathcal{S}, \emptyset, \mathcal{P}, \mathcal{L}_\varepsilon \rangle$ from G as follows: For each lexicon rule $(A, E) \rightarrow t$ in $\mathcal{L}$, $(A, E) \rightarrow \varepsilon$ is a lexicon rule in $\mathcal{L}_\varepsilon$. It is trivial that $L(G) \neq \emptyset$ if and only if $\varepsilon \in L(G_\varepsilon)$ if and only if $L(G_\varepsilon) \neq \emptyset$.

Now we will transform G_ε into a feature productive grammar. First notice that since the empty string is the only possible string in $L(G_\varepsilon)$, the order of the right hand side in the production rules does not matter, any repetition of identical elements is redundant[3], and we may view the right hand sides in production and lexicon rules as sets. We treat them together in $\widehat{\mathcal{PL}}$ as follows: If $(A_0, E) \rightarrow (A_1, e_1) \ldots (A_n, e_n)$ is a production rule in $\mathcal{P}$ then $(A_0, E) \rightarrow \{(A_1, e_1), \ldots, (A_n, e_n)\}$ is a *set rule* in $\widehat{\mathcal{PL}}$, and if $(A, E) \rightarrow \varepsilon$ is a lexicon rule in $\mathcal{L}_\varepsilon$, then $(A, E) \rightarrow \{\varepsilon\}$ is a set rule in $\widehat{\mathcal{PL}}$. Now, we define the following closure on $\widehat{\mathcal{PL}}$: If

$$(A, E) \rightarrow \psi \cup \{(B, \uparrow\doteq\downarrow)\} \tag{9}$$
$$(B, F) \rightarrow \phi \tag{10}$$

are two rules in $\widehat{\mathcal{PL}}$, then the following is also a rule in $\widehat{\mathcal{PL}}$:

$$(A, E \cup F) \rightarrow \psi \cup \phi \tag{11}$$

We see that $\widehat{\mathcal{PL}}$ is a finite set since the constituents of each rule are elements or subsets of finite sets. Now let $\widehat{\mathcal{P}'\mathcal{L}'}$ be the set $\widehat{\mathcal{PL}}$ minus every rule where

[3] This due to the fact that if a node in a c-structure has two daughters labeled with identical category symbols and identical path equation schema, the sets of possible feature consistent sub-c-structures which can be rooted at each of these two nodes are identical, and so are the constraints on the feature structure they may produce.

$\uparrow \doteq \downarrow$ occurs in elements in the set on the right hand side.

It is a trivial procedure to separate $\widehat{\mathcal{PL}}$ into a set of lexicon rules and a set of production rules, and so is the process to separate $\widehat{\mathcal{P}'\mathcal{L}'}$ into a set of lexicon rules and a set of *feature productive* production rules. Hence the grammar G' we get from $\widehat{\mathcal{P}'\mathcal{L}'}$, together with $\mathcal{K}$, $\mathcal{S}$, and $\Sigma' = \emptyset$ is a feature productive THFSG grammar. So, every step in the definition of G' is simple to implement as an effective procedure.

The grammar we get from $\widehat{\mathcal{PL}}$ is weakly equivalent G_ε, since each rule in G_ε exists as a set rule in $\widehat{\mathcal{PL}}$, and each additional rule in $\widehat{\mathcal{PL}}$ corresponds to an internal sub-c-structure based on G_ε. Then we have to show that the grammar we get from $\widehat{\mathcal{P}'\mathcal{L}'}$ is weakly equivalent to the grammar we get from $\widehat{\mathcal{PL}}$. Since $\widehat{\mathcal{P}'\mathcal{L}'} \subseteq \widehat{\mathcal{PL}}$, we only have to consider c-structures with set rules from $\widehat{\mathcal{PL}}$ with $\uparrow \doteq \downarrow$ in elements in the right hand side. By induction on the number of such rules used subsequently in a feature consistent c-structure we have that any sequence of such rules can be replaced with a rule in $\widehat{\mathcal{P}'\mathcal{L}'}$ which yields the same (empty) string and which is feature consistent. Then we may transform any feature consistent c-structure for ε with rules from the $\widehat{\mathcal{PL}}$ to a feature consistent c-structure with only rules from $\widehat{\mathcal{P}'\mathcal{L}'}$. ∎

2.2 The modal language $L^\Box$ and finite acyclic models

There is a trend in computational linguistics to view the structures that is used to collect grammatical information about language strings as models in the model theoretical sense. Different modal languages have shown to be promising to talk about such models [1]. In this section we will present the modal language $L^\Box$ as it is described in [2]. This is a language of propositional modal logic, with a countable set of distinct existential modalities and a countable set of propositional symbols, together with the universal modal operator $\Box$.

Given a set $\mathcal{U}$ of atomic propositions, and $\mathcal{A}$ of modalities, the set of well formed formulas (wff's) in $L^\Box$ is the least set such that if $v \in \mathcal{U}$, $a \in \mathcal{A}$, $\phi, \phi' \in L^\Box$, then v, $\langle a \rangle \phi$, $\neg\phi$, $\phi \vee \phi'$,$\Box\phi \in L^\Box$. We use the standard definitions of the other boolean connectives $\wedge$, $\rightarrow$, $\top$, and $\bot$. The empty disjunction is defined as $\bot$, and the empty conjunction is defined as $\top$.

Given a set of modalities $\mathcal{A}$ and a set of atomic propositions $\mathcal{U}$, a deterministic Kripke model is a triple $\langle Q, \{R_a\}_{a \in \mathcal{A}}, \mathsf{Val} \rangle$ such that Q is a nonempty set of nodes, each $R_a : Q \rightarrow Q$ is a partial function, and $\mathsf{Val} : \mathcal{U} \rightarrow 2^Q$ is a valuation function which assigns a subset of Q to each $v \in \mathcal{U}$. A Kripke model is acyclic if and only if there does not exist any sequence of nodes in Q, where each node is connected to the next by a modality, and at least one node occurs in more than one place in the sequence.

Given a deterministic Kripke model $\langle Q, \{R_a\}_{a \in \mathcal{A}}, \mathsf{Val} \rangle$ we define the satisfaction relation for wff's of $L^\Box$ the usual way:

$$
\begin{array}{lll}
M \models v[q] & \textit{iff} & q \in \mathsf{Val}(v) \\
M \models \langle a \rangle \phi[q] & \textit{iff} & \exists q' \in Q : \langle q, q' \rangle \in R_a \\
 & & \;\&\; M \models \phi[q'] \\
M \models \neg\phi[q] & \textit{iff} & M \not\models \phi[q] \\
M \models \phi \vee \phi'[q] & \textit{iff} & M \models \phi[q] \;\; \textit{or} \;\; M \models \phi'[q] \\
M \models \Box\phi[q] & \textit{iff} & \forall q' \in Q : M \models \phi[q']
\end{array}
$$

The connection between feature structures and Kripke models is well described in [1]. Well defined feature structures where required to be describable, acyclic and atomic in addition to finite. Later we will see that the atomicity constraint can easily be formulated in a formula. The finite and acyclic constraint we define as metaconstraints on the Kripke-models. When we limit the interpretation to deterministic Kripke models we also ensure that the attribute transition function is well defined. But, we need two versions of two theorems (4.3 and 4.4) in [2] which deal with finite *acyclic* deterministic Kripke models. The proofs of these lemmas are elaborations of the proofs in [2] and are omitted here, but may be found in the full version of this article.

Lemma 2 *Let Φ be an $L^\Box$ formula. If Φ is satisfiable in a finite acyclic deterministic Kripke model then it is satisfiable in a finite acyclic deterministic Kripke model with at most $2^{2|\Phi|}$ nodes.*

Lemma 3 *Let Φ be an $L^\Box$ formula. Then it is decidable whether or not Φ is satisfiable in a finite acyclic deterministic Kripke model.*

2.3 From grammar rules to a modal formula

In this section we will define an $L^{\Box}$ formula $\Phi_{\emptyset}$ for each THFSG grammar with the following property: If the grammar is feature productive then $\Phi_{\emptyset}$ is valid in finite acyclic deterministic Kripke models if and only if the language defined by the given grammar is empty. We start by formulating the information we have about any node decorated with a given category in the c-structure.

Definition 2 (Φ_A) *Let $G = \langle \mathcal{K}, \mathcal{S}, \Sigma, \mathcal{P}, \mathcal{L} \rangle$ be a* THFSG *grammar. Let $\mathcal{A}$ be the set of attribute symbols used in G and $\mathcal{V}$ the set of value symbols used in G. Then let $\mathcal{U} = (\mathcal{V} \cup \mathcal{K})$ be the set of atomic propositions and $\mathcal{A}$ the set of modalities in $L^{\Box}$.*

i). For each category $A \in \mathcal{K}$, let R_A be the set of production and lexicon rules in G with A on the left hand side.

ii). If r is the following production rule in G

$$\begin{array}{lcccc} A & \rightarrow & B_1 & \ldots & B_n \\ & & E_1 & & E_n \end{array} \qquad (12)$$

then ϕ_r is the following $L^{\Box}$ formula:

$$E_{\Box} \wedge \langle a_{1,1} \rangle \ldots \langle a_{1,m_1} \rangle B_1 \wedge \ldots \wedge \langle a_{n,1} \rangle \ldots \langle a_{n,m_n} \rangle B_n \qquad (13)$$

where $\uparrow a_{i,1} \ldots a_{i,m_i} = \downarrow$ is the path equation schema in E_i for $i : 1 \leq i \leq n$, and $E_{\Box}$ is the conjunction of all $L^{\Box}$ formulas $\langle a_1 \rangle \ldots \langle a_m \rangle v$ such that $\uparrow a_1 \ldots a_m = v$ is a value equation schema in $E = E_1 \cup \ldots \cup E_n$,

iii). If r is the following lexicon rule in G

$$\begin{array}{lcc} A & \rightarrow & t \\ & & E \end{array} \qquad (14)$$

then ϕ_r is the $L^{\Box}$ formula $E_{\Box}$ defined from E as above.

From i), ii) and iii), let Φ_A be the $L^{\Box}$ formula

$$A \rightarrow \bigvee_{r \in R_A} \phi_r \qquad (15)$$

In section 2.2 we introduced metaconstraints on our models expressing that they must be finite, deterministic and acyclic. Now we define a formula that expresses that the models have to be atomic and that no more than one atomic value can be assigned to each node.

Definition 3 ($\Phi_{\odot}$) *Given any* THFSG *grammar G. Let $\mathcal{A}$ be the set of attribute symbols used in G and let $\mathcal{V}$ be the set of value symbols used in G. Then let $\mathcal{U} = (\mathcal{V} \cup \mathcal{K})$ be the set of atomic propositions and $\mathcal{A}$ the set of modalities in $L^{\Box}$. Then $\Phi_{\odot}$ is the $L^{\Box}$ formula*

$$\bigwedge_{v \in \mathcal{V}} \bigwedge_{a \in \mathcal{A}} v \rightarrow \neg \langle a \rangle \top \quad \wedge \bigwedge_{\substack{v, v' \in \mathcal{V} \\ v \neq v'}} v \rightarrow \neg v' \qquad (16)$$

2.4 The emptiness problem

Lemma 4 (Emptiness with $\Phi_{\emptyset}$) *Given any feature productive* THFSG *grammar $G = \langle \mathcal{K}, \mathcal{S}, \Sigma, \mathcal{P}, \mathcal{L} \rangle$. Then $L(G) = \emptyset$ if and only if*

$$\Box \bigwedge_{A \in \mathcal{K}} \Phi_A \wedge \Box \Phi_{\odot} \rightarrow \neg \mathcal{S} \qquad (17)$$

is valid in finite acyclic deterministic Kripke models.

We call the formula in (17) $\Phi_{\emptyset}$.

Proof: ($\Longleftarrow$) Assume that $L(G) \neq \emptyset$ and that $w \in L(G)$ for a string w. Then there exists a c-structure $c = \langle D, C, E \rangle$ based on G which is supported by a feature structure $M = \langle Q, f, \delta, \theta \rangle$ and which has w as terminal string. Assume that $\mathcal{A}$ is the set of attribute symbols used in G, and that $\mathcal{V}$ is the set of value symbols used in G. We define an acyclic deterministic Kripke model $M' = \langle Q, \{R_a\}_{a \in \mathcal{A}}, \mathsf{Val} \rangle$ from c and M as follows: Let $\{R_a\}_{a \in \mathcal{A}}$ be the set of partial functions R_a, where each R_a is the set

$$\{\langle q, q' \rangle \in Q \times Q \mid \delta(q, a) = q'\} \qquad (18)$$

Since δ is a partial function, so must each R_a be. The valuation function $\mathsf{Val} : (\mathcal{K} \cup \mathcal{V}) \rightarrow 2^Q$ is defined for each $A \in \mathcal{K}$ and each $v \in \mathcal{V}$ as follows:

$$\mathsf{Val}(A) = \{f(x) \mid x \in D - term(D) \ \& \ C(x) = A\} \qquad (19)$$

$$\mathsf{Val}(v) = \{q \in Q \mid \theta(q) = v\} \qquad (20)$$

Since M is acyclic, finite and deterministic, we have from the given definition that M' also must be

acyclic, finite and deterministic. Now we have to show that $M' \models \neg\Phi_\emptyset[q_0]$, for some node q_0, that is

$$M' \models \Box \bigwedge_{A \in \mathcal{K}} \Phi_A \wedge \Box\Phi_\odot \wedge \mathcal{S}\ [q_0] \qquad (21)$$

Let q_0 be $f(\varepsilon)$. Then we have that $M' \models \mathcal{S}[q_0]$ since $C(\varepsilon) = \mathcal{S}$, and since M is atomic and θ is well defined we have directly that $M' \models \Box\Phi_\odot[q_0]$.

Now, given any node $q \in Q$ and any category symbol A such that $M \models A[q]$. From the definition of M' we then know that there exists a nonterminal node $x \in D$ such that $f(x) = q$ and $C(x) = A$. Then there exists a lexicon or production rule *license*(x)

$$\begin{array}{lcccc} A & \rightarrow & C(x{\cdot}1) & \ldots & C(x{\cdot}d(x)) \\ & & E(x{\cdot}1) & & E(x{\cdot}d(x)) \end{array} \qquad (22)$$

that is applied at x. Since c is supported by M all the equations we get from the equation schemata in this rule are satisfied in M. If *license*(x) is a production rule we have by the definition of M' that if $\uparrow a_{i,1} \ldots a_{i,m_i} \doteq\downarrow\in E_i$ so must $M' \models \langle a_{i,1}\rangle \ldots \langle a_{i,m_i}\rangle C(x \cdot i)[q]$ since $\delta(f(x), a_{i,1} \ldots a_{i,m_i}) = f(x{\cdot}i)$ and $f(x{\cdot}i) \in \mathsf{Val}(C(x{\cdot}i))$, for each $i : 1 \leq i \leq d(x)$. Also by the definition of M' for each value equation schemata $\uparrow a'_1 \ldots a'_n \doteq v$ in each $E(xi)$, $M' \models \langle a'_1\rangle \ldots \langle a'_n\rangle v[q]$ since $\theta(\delta(f(x), a'_1 \ldots a'_n)) = v$. Then by the definition of $\phi_{license(x)}$, we have directly that $M' \models \phi_{license(x)}[q]$, and then

$$M' \models A \rightarrow \bigvee_{r \in R_A} \phi_r[q] \qquad (23)$$

Since this is the case for any node $q \in Q$ and any category A, we have that

$$M \models \Box \bigwedge_{A \in \mathcal{K}} \Phi_A[q_0] \qquad (24)$$

Then we have that $M' \models \neg\Phi_\emptyset[q_0]$ and $\Phi_\emptyset$ is not valid in finite acyclic deterministic Kripke models.

($\Longrightarrow$) Assume that $\Phi_\emptyset$ is not valid in acyclic deterministic Kripke models. Then there exists a finite acyclic deterministic Kripke model M' with a node q_0 such that $M' \models \neg\Phi_\emptyset[q_0]$, that is

$$M' \models \Box \bigwedge_{A \in \mathcal{K}} \Phi_A \wedge \Box\Phi_\odot \wedge \mathcal{S}\ [q_0] \qquad (25)$$

From this we will show that there must exist a feature consistent c-structure based on G. Assume that $M' = \langle Q, \{R_a\}_{a \in \mathcal{A}}, \mathsf{Val}\rangle$. Then let $M = \langle Q, f, \delta, \theta\rangle$ be a feature structure where δ and θ are defined as follows:

$$\begin{aligned} \delta &= \{\langle q, a, q'\rangle \mid \langle q, q'\rangle \in R_a \\ &\qquad \& \ R_a \in \{R_a\}_{a \in \mathcal{A}})\} \qquad (26) \\ \theta &= \{\langle q, v\rangle \mid q \in \mathsf{Val}(v) \ \& \ v \in \mathcal{V}\} \qquad (27) \end{aligned}$$

Since $M' \models \Box\Phi_\odot$, M must be atomic and θ well defined, and since M' is deterministic, δ must be a well defined partial function.

We define a c-structure $c = \langle D, C, E\rangle$ and the name mapping function f of M from M' inductively top down in c. At the same time we also show that all the equations we get from the equation schemata sets in c are satisfied in M. To help us with this we will also show that for all $x \in D$ such that $C(x) \in \mathcal{K}$, $f(x) \in \mathsf{Val}(C(x))$.

Initial step (the root node ε): Let $f(\varepsilon) = q_0$, and let $C(\varepsilon) = \mathcal{S}$. Since $M \models \mathcal{S}[q_0]$ so must $q_0 \in \mathsf{Val}(\mathcal{S})$ and then $f(\varepsilon) \in \mathsf{Val}(C(\varepsilon))$. There are no equations attached to ε.

Induction hypothesis: Assume that all nodes above x are licensed by some rule in G, and that, for all nodes above and including x, $f(x)$ is defined and $f(x) \in \mathsf{Val}(C(x))$. Assume also that all the equations we get from the equation schemata sets assigned to these nodes are satisfied in M.

Induction step: Assume that $C(x) = A$ for some A and $f(x) \in \mathsf{Val}(A)$. Let q be $f(x)$. Then $M' \models A[q]$. Since $M' \models \bigwedge_{B \in \mathcal{K}} \Phi_B$, we know that $M' \models \Phi_A$ and since $M' \not\models A \rightarrow \bot[q]$ there must be a rule $r \in R_A$ such that $M' \models \phi_r[q]$. Assume that r is the following rule

$$\begin{array}{lcccc} A & \rightarrow & B_1 & \ldots & B_n \\ & & E_1 & & E_n \end{array} \qquad (28)$$

Then let *license*(x) be r, that is: extend c's tree domain with the nodes $x1, \ldots, xn$, and assign $C(xi) = B_i$ and $E(x{\cdot}i) = E_i$ for each $i : 1 \leq i \leq n$. Now for each $i : 1 \leq i \leq n$: Assume that $\uparrow a_{i,1} \ldots a_{i,m_i} =\downarrow$ is the path equation schema in E_i if r is a production rule. Since $M' \models \phi_r[q]$ we know that $M' \models \langle a_{i,1}\rangle \ldots \langle a_{i,m_i}\rangle B_i[q]$. Then we know that there exists a sequence of nodes $q'_1, \ldots, q'_{m_i}$ such that $\langle q, q'_1\rangle \in R_{a_{i,1}}, \langle q'_1, q'_2\rangle \in R_{a_{i,2}}, \ldots, \langle q'_{m_i-1}, q'_{m_i}\rangle \in R_{a_{i,m_i}}$ and $q'_{m_i} \in \mathsf{Val}(B_i)$. Then let $f(x{\cdot}i) = q'_{m_i}$,

and we have that $f(x \cdot i) \in \mathsf{Val}(C(x \cdot i))$. The path equation we get from the path equation schema in E_i is then satisfied in M. In the same way we see that all the value equations we get from the value equation schemata in E_i must be satisfied in M. If r is a lexicon rule the value equation schemata must be satisfied in the same way. In this case we extend the c-structure with a terminal node $x \cdot 1$, and assigns B_1, which is a terminal symbol, together with E_1 to $x \cdot 1$.

With the definition of f the feature structure is well defined except for describability. However, restricting M to nodes reachable[4] from q_0, does not make any difference in the above arguments, hence we may restrict both Q, δ and θ to these nodes. Since $f(\epsilon) = q_0$ it then became describable.

Since the Kripke model is finite and acyclic the process terminate with terminal symbols.

Then c must be a complete c-structure and since all the equations we get from the equation schemata in c are satisfied in M, M supports c and hence is c feature consistent. Then the terminal string of c is a member of $L(G)$ and $L(G) \neq \emptyset$. ∎

Theorem 1 *The emptiness problem for* THFSG *is decidable.*

Proof: From Lemma 1, 3 and 4. ∎

Theorem 2 *The membership problem for* THFSG *is decidable.*

Proof: From [3] we have that THFSG is closed under intersection with regular languages. In the same place it is described how we can construct a grammar for the intersection. Then we can construct a THFSG grammar for the language $L(G) \cap \{w\}$. This language is non-empty if and only if $w \in L(G)$. ∎

2.5 Consequences of the membership decidability

Theorem 3 *The class of languages described by* THFSG *is not closed under intersection or complement.*

[4] By this we mean all nodes q such that there exist and w where $\delta(q_0, w) = q$

Proof: [6] Given any recursively enumerable set R, does it exists two deterministic context free languages $L_{R,1}$ and $L_{R,2}$, and a homomorphism h such that $R = h(L_{R,1} \cap L_{R,2})$. Now assume that R is not recursive. $L_{R,1}$ and $L_{R,2}$ are describable by two THFSG grammars. If $L_{R,1} \cap L_{R,2}$ had been a THFSG-language, then $R = h(L_{R,1} \cap L_{R,2})$ would also have been since THFSG is closed under homomorphism [3]. But, this that can not be the case since THFSG has a decidable membership problem.

If THFSG had been closed under complement, then $\overline{\overline{L_{R,1}} \cup \overline{L_{R,2}}} = L_{R,1} \cap L_{R,2}$ would have been a THFSG-language since THFSG is closed under union [3]. ∎

Given an ordering on the c-structures we have the following:

Theorem 4 *Given any* THFSG *grammar G, and a string w. Then there exists an effective procedure which give a least feature consistent c-structure for w if $w \in L(G)$ or a "no" answer if $w \notin L(G)$.*

Proof: From Theorem 2 and the fact that the c-structures based on any given grammar is enumerable. ∎

3 Empty value path extension

In the next section we show that we may allow regular expressions in the attribute strings in the equation schemata without changing the class of languages the THFSG grammars describe. In the proof we need the possibility that value equation schemata may have empty attribute strings as in $\uparrow \doteq v$. Recall that we in the definition of THFSG required the value equation schemata to have nonempty attribute strings.

Definition 4 (Empty value path extension) *A* THFSG$^+$ *grammar is a* THFSG *grammar where we allow value equation schemata $\uparrow w = v$ with $w \in \mathcal{A}^*$.*

Theorem 5 *The class of languages described by* THFSG$^+$ *is the same as the class of languages described by* THFSG.

We prove this by changing the equation schemata, and manipulation on the feature structure. It is not particular interesting nor surprising, and the proof may be found in the full version of this article.

4 Regular expressions in the equations

We extend our definitions of equation schemata to include regular expressions over attribute symbols in addition to just strings. Regular expressions are used in LFG-like grammar formalisms to cope with long distance dependencies such as topicalization [9].

Definition 5 (Regular expression equations) *If α_1 and α_3 are regular expressions, then $x_1\,\alpha_1 \doteq x_2$ and $x_3\,\alpha_3 \doteq v$ are regular expression equations.*

A feature structure $\langle Q, f, \delta, \theta\rangle$ satisfies *the regular expression equation $x_1\alpha_1 \doteq x_2$ if and only if there exists a string $u_1 \in L(\alpha_1)$ such that $\delta(f(x_1), u_1) = f(x_2)$ and the regular expression equation $x_3\alpha_3 \doteq v$ if and only if there exists a string $u_3 \in L(\alpha_3)$ such that $\theta(\delta(f(x_3), u_3)) = v$*

Definition 6 (THFSG^{++}) *A* THFSG^{++} *grammar is a* THFSG *grammar where we allow regular expressions over attribute symbols in the equation schemata instead of just strings as in* THFSG.

Theorem 6 *The class of languages described by* THFSG^{++} *is the same as the class of languages described by* THFSG.

In the proof each regular expression is isolated in path equation schemata in non-branching production rules. Then each regular expression are decomposed until they are just strings, by introducing additional new production rules whith unique new category symbols. This give us a THFSG-grammar. The full proof may be found in the full version of this article.

Acknowledgments

I would like to thank Tore Langholm for his advice during the work that this paper is based on, and for comments on earlier versions of the paper.

References

[1] Patrick Blackburn. Structures, languages and translations: the structural approach to feature logic. In C. J. Rupp, M. A. Rosner, and R. L. Johnson, editors, *Constraints, Language and Computation*, pages 1–28. Academic Press, London, England, 1994.

[2] Patrick Blackburn and Edith Spaan. A modal perspective on the computational complexity of attribute value grammar. *Journal of Logic, Language, and Information*, 2:129–169, 1993.

[3] Tore Burheim. A grammar formalism and cross-serial dependencies. In Patrick Blackburn and Maarten de Rijke, editors, *Specifying Syntactic Structure*. CSLI-Publications, 1997.

[4] Erik A. Colban. *Three Studies in Computational Semantics*. PhD thesis, University of Oslo, 1991.

[5] Mary Dalrymple, Ronald M. Kaplàn, John T. Maxwell III, and Annie Zaenen. *Formal Issues in Lexical-Functional Grammar*. Number 47 in CSLI Lecture Notes. CSLI Publications, 1995.

[6] S. Ginsburg, S.A. Greibach, and M.A. Harrison. One-way stack automata. *Journal of the Association of Computing Machinery*, 14(2):389–418, 1967.

[7] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1979.

[8] Ronald M. Kaplan and Joan Bresnan. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. The MIT-Press, Cambridge, Massachusetts, USA, 1982.

[9] Ronald M. Kaplan, John T. Maxwell III, and Annie Zaenen. Functional uncertainty. CSLI Monthly 2:4, 1987.

[10] William C. Rounds. Context-free grammars on trees. In *Conference Record of the ACM Symposium on Theory of Computing*. Association of Computing Machinery, 1969.

[11] Hiroyuki Seki, Ryuichi Nakanishi, Yuichi Kaji, Sachiko Ando, and Tadao Kasami. Parallel multiple context-free grammars, finite-state translation systems, and polynomial-time recognizable subclasses of Lexical-Functional Grammars. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics, ACL'93*, pages 130–139. Association for Computational Linguistics, 1993.

Relating resource-based semantics to categorial semantics*

Mary Dalrymple† Vineet Gupta† John Lamping† Vijay Saraswat‡

July 16, 1997

1 Introduction

We provide a new formulation of the resource-based 'glue' approach to semantics (Dalrymple, Lamping, and Saraswat, 1993; Dalrymple et al., 1997) which better brings out the essential differences and similarities between the glue style and categorial approaches. In particular, we show that many applications of the glue approach use a fragment of linear logic which is equivalent to typed linear lambda calculus. For example, the word 'yawn' might be encoded in the three approaches as approximately:

Categorial	$\lambda x.yawn(x) : \mathrm{N\backslash S}$
Old Glue	$\forall M.\ g \rightsquigarrow M \multimap f \rightsquigarrow yawn(M)$
New Glue	$\lambda M.yawn(M) : g \multimap f$

An essential difference between the categorial approaches and the glue approach is their relation to syntax. Categorial approaches describe syntactic rules, starting from the point of view of how meanings will functionally compose, and using types, like N\S above. The glue approach doesn't try to describe syntactic rules, but rather connects to a separate grammar, and says how to assemble meanings of sentences that have been analyzed by the grammar. It focuses on mediating the differences between the compositional structure of the grammar and the compositional structure of meaning assembly, differences such as occur with quantifier scoping.

A historical difference between the categorial approach and the glue approach has stemmed from the former's use of lambda expressions to manipulate meanings, compared to the latter's use of quantification. This has meant that the actual composition of meanings in the categorial approach is clearly separated from the syntactic types. The original formulation of the glue approach, in contrast, intermixed syntactic information and meanings.

This paper shows that a significant fragment of the glue approach can be reformulated to separate out the meaning composition in a way that is very similar to that of the categorial approaches. Specifically, we show the following:

- A core fragment $\mathcal{C}$ of linear logic (LL) can be used to define semantic assembly in many cases.
- Every formula in $\mathcal{C}$ can be read as an assertion that a particular λ-term has a particular type. These assertions are formulas in System F (Girard 1986).
- The two formulations have equivalent deductive power.

When the glue approach requires only the core fragment, the reformulation allows it to take advantage of one of the primary attractions of the categorial approach: that the syntactic well-formedness of a sentence can be reasoned about strictly in terms of types, yet the types can be labeled with lambda terms so that a meaning term for the sentence can be constructed automatically from a proof that the sentence has the appropriate type, following the well-known Curry-Howard connection between the λ-calculus and intuitionistic logic. This is attractive because it captures formally the intuition that the process of meaning construction is sensitive only to the *types* of the terms being assembled, not their actual *values*. This *Meaning Parametricity* structurally guarantees that no assumptions about the content of the actual meaning are built into the meaning assembly process. Put another way, this captures the intuition that control information is manifest only through type-structure. That is, the *only* way information from phrasal structure can be used to influence the combinations of meanings of lexical entries is through the type-structure of the meanings — in particular, "control" information

*We are grateful to John Fry, David Israel, Mark Johnson, Nissim Francez, Dick Oehrle, Fernando Pereira, and Johann van Bentham for helpful discussion of the issues raised here.

†Xerox PARC, 3333 Coyote Hill Road, Palo Alto CA 94304 USA; {dalrymple,vgupta,lamping}@parc.xerox.com

‡AT&T Research, 180 Park Avenue, P.O. Box 971, Florham Park NJ 07932-0971 USA; vj@research.att.com

cannot be encoded in the specific meaning terms, which live in their own separate world, independent of control or computational concerns.

Under the reformulation of the glue approach, the type structure reflects the parsing results of a separate process (LFG syntactic analysis). This process interfaces with the meaning assembly process by generating types for lexical entries that involve *type constants* (called *roles*, cf. f and g in the example above) which capture more information than just the information about individuals (e) and truth-values (t) inherent in the meaning term; they also capture information about the role of the lexical entry in the syntactic parse. This information is naturally represented by these type constants in the various meaning terms arising from the lexical entries. The *combination* of the meanings is then purely standard.

In the following we first review the categorial approach and the glue approach. We then characterize the fragment of the glue approach that is equivalent to typed lambda calculus and prove the equivalence. We discuss the differences between that calculus and categorial approaches. Finally, we examine applications of glue semantics that use linear logic that exceeds the core fragment $\mathcal{C}$ and discuss why this step was necessary.

2 Categorial grammar

Lexical entries in a typical Categorial Grammar (Lambek, 1958; Oehrle, Bach, and Wheeler, 1988) assign syntactic categories such as N to noun phrases like *Bill*, or more complex categories like N\S for intransitive verbs like *yawned*. N\S represents a category that combines with a N to its left to produce a S. Lexical entries are also associated with typed λ-calculus terms, which are combined via application and abstraction in a manner dictated by the types and the associated parse tree to yield a meaning for the entire utterance. For a sentence like *Bill yawned*, the relevant lexical entries are:

(1)

Bill	*Bill*	:	N
yawned	$\lambda x.yawn(x)$	:	N\S

Given this lexical information, we can produce the following proof that *Bill yawned* is a sentence:

(2) Derivation:

$$\frac{Bill : \mathrm{N} \Rightarrow Bill : \mathrm{N} \quad yawn : \mathrm{N\backslash S} \Rightarrow yawn : \mathrm{N\backslash S}}{Bill : \mathrm{N}, yawn : \mathrm{N\backslash S} \Rightarrow yawn(Bill) : \mathrm{S}}\backslash E$$

It suffices to consider the derivations as inferring types (formulas in intuitionistic logic) — the term corresponding to the meaning can then be built up in a uniform way from the proofs and terms corresponding to the lexical entries.

3 Resource-based semantics and LFG

In this section we provide a brief overview of the resource-based approach. For a more detailed presentation, see Dalrymple et al. (1995c).

Lexical-Functional Grammar (LFG) (Bresnan, 1982; Dalrymple et al., 1995a) assumes two syntactic levels of representation. The constituent structure tree encodes phrasal dominance and precedence relations. Functional structure (*f-structure*) encodes syntactic predicate-argument structure, and is represented as an attribute-value matrix. With Kaplan and Bresnan (1982) and Halvorsen (1983), we claim that the functional syntactic information encoded by f-structures – information about relations such as *subject-of*, *object-of*, *modifier-of*, and so on – is the primary determinant of semantic composition. Those relations are realized by different constituent structures in different languages, but are represented directly and uniformly in the f-structure.

We also assume a *semantic function* σ which maps an f-structure to a *semantic structure* or *role* encoding information about f-structure meaning. In the following, we will use names like f and g for roles except when we wish to establish the relation of the role to its corresponding functional structure; in those cases, we will use names like f_σ or $(f\ \textsc{subj})_\sigma$ for roles corresponding to the functional structures f and $(f\ \textsc{subj})$.

We use linear logic (LL) as the 'glue' for composing a meaning for an utterance from meanings of its constituents. Role, r, is associated with meaning term, M, (drawn from some pre-specified 'meaning logic', usually Montague's intensional logic (Montague, 1974)) via an atomic assertion of the form $r{\rightsquigarrow_\tau}M$, where $\rightsquigarrow$ is an otherwise uninterpreted binary predicate symbol and τ is the semantic type of the entry. Lexical entries contribute either atomic formulas for their roles or a non-atomic LL formula (e.g., $s{\rightsquigarrow_\alpha}M \multimap t{\rightsquigarrow_\beta}N \multimap r{\rightsquigarrow_\tau}P$) that states how a meaning term may be produced by consuming meaning terms associated with related roles.

The LL fragment used allows quantification over roles and over meaning-terms. A meaning M of the role r corresponding to the entire utterance is obtained by finding an LL proof of $r{\rightsquigarrow_\alpha}M$ (where

α is usually t, the type of propositions) from the contributions of the lexical entries (perhaps in the presence of an underlying LL theory). Each such derivable M provides a meaning for the utterance; for the representation in glue logic to be complete, every meaning for the utterance must be derivable in this way.

Names The meaning constructor for a particular occurrence of a name such as 'Bill' establishes an association between a role g_σ and the constant *Bill* representing its meaning:

(3) $g_\sigma \rightsquigarrow_e Bill$

Since the meaning term is an entity, we indicate this by the subscript e on the 'means' relation $\rightsquigarrow_e$.

Verbs The meaning constructor for a verb such as 'yawned' is a glue language formula that can be thought of as instructions for how to assemble the meaning of a sentence with main verb 'yawned': the meaning X for the subject of 'yawned' is consumed, and the sentential meaning is produced.

(4) $\forall X.\ (f\ \text{SUBJ})_\sigma \rightsquigarrow_e X \multimap f_\sigma \rightsquigarrow_t yawn(X)$

Deduction of meaning We will give the names **Bill** and **yawned** to the meaning constructors for the sentence 'Bill yawned'. Note that in the case at hand, f's SUBJ is labeled g, and so we can write g instead of $(f\ \text{SUBJ})$ in the meaning constructor for **yawned**.

(5)
$$f:\begin{bmatrix}\text{PRED} & \text{'YAWN'} \\ \text{SUBJ} & g:\begin{bmatrix}\text{PRED} & \text{'BILL'}\end{bmatrix}\end{bmatrix}$$

bill: $g_\sigma \rightsquigarrow_e Bill$

yawned: $\forall X.\ g_\sigma \rightsquigarrow_e X \multimap f_\sigma \rightsquigarrow_t yawn(X)$

From these premises, LL sanctions the following proof ($\vdash$ stands for the LL entailment relation):

(6)

	bill $\otimes$ **yawned**	*(Premises.)*
$\vdash$	$f_\sigma \rightsquigarrow_t yawn(Bill)$	$X \mapsto Bill$

Quantifiers A generalized quantifier such as 'everyone' can be seen as making a semantic contribution along the following lines: If by giving the arbitrary meaning x to g_σ, the role for 'everyone', we can derive the meaning $S(x)$ for the scope of quantification, then S can be the property that the quantifier requires as its scope, yielding the meaning $every(person, S)$ (Dalrymple et al., 1997). Logically, this means that the semantic constructor for an NP quantifies universally over roles:

(7) **everyone** $\forall H, S.\ (\forall x.\ g_\sigma \rightsquigarrow_e x \multimap H \rightsquigarrow_t S(x)) \multimap H \rightsquigarrow_t every(person, S)$

Notice that the assignment of a meaning to H appears on both sides of the implication, and that the meaning is not the same in the two instances.

The derivation of the meaning of an sentence like 'Everyone yawned' proceeds from the meaning constructors for 'everyone' and 'yawned':

(8) Everyone yawned.

$$f:\begin{bmatrix}\text{PRED} & \text{'YAWN'} \\ \text{SUBJ} & g:\begin{bmatrix}\text{PRED} & \text{'EVERYONE'}\end{bmatrix}\end{bmatrix}$$

everyone: $\forall H, S.\ (\forall M.\ g_\sigma \rightsquigarrow_e M \multimap H \rightsquigarrow_t S(M)) \multimap H \rightsquigarrow_t every(person, S)$

yawned: $\forall M.\quad g_\sigma \rightsquigarrow_e M \multimap f_\sigma \rightsquigarrow_t yawn(M)$

With this, we have the following derivation:

	everyone $\otimes$ **yawned**	(Premises.)
$\vdash$	$f_\sigma \rightsquigarrow_t every(person, yawn)$	$H \mapsto f_\sigma, M \mapsto M,$ $S \mapsto yawn$

4 Analysis of $\mathcal{C}$

The examples presented above lie in the core fragment $\mathcal{C}$. We now turn to an analysis of $\mathcal{C}$ and elucidation of its formal properties.

We first make a notational shift. The glue literature, and the discussion of it above, writes atoms of the form, $r_\sigma \rightsquigarrow_e M$, where r denotes a *syntactic* construction, and its sigma projection, r_σ denotes a place to attach a meaning. For this paper, we want to focus on just the meaning assembly process, and so it is convenient to avoid constants that denote syntactic entities. From now on, our constants will directly denote places to attach meanings. Further, rather than add the type, e or t, as another argument to the $\rightsquigarrow$ relation, we will type our constants and variables. Thus, we will write $r_e \rightsquigarrow M$ to say that the meaning attachment point, r_e, which has type e is associated with meaning M.

4.1 Informal development

If we reconsider the meaning deduction for the sentence 'Everyone yawned', we can remove the meaning terms from the atoms and remove quantification over meaning terms, obtaining the 'stripped' premises:

everyone: $\forall H.\,(g_e \multimap H_t) \multimap H_t$
yawned: $g_e \multimap f_t$

Notice that quantification over H is left, because H ranges over roles, not meanings. Intuitively, the stripped premises yield the stripped deduction:

$$\begin{array}{ll} & \forall H.\,(g_e \multimap H_t) \multimap H_t \\ & g_e \multimap f_t \\ \vdash & f_t \end{array}$$

It is easy to see that stripping meanings from the propositions preserves deductions. That is, that if there is a deduction from some original formulas, then there is an equivalent deduction from the stripped formulas. This is true of any fragment where the notion of stripping makes sense, since the stripped formulas remove some constraints on what formulas can be combined, and add no new constraints.

More interestingly, the implication goes the other way: the meanings do not constrain the deduction. That is, if one starts with original formulas, strips them, and makes a deduction from the stripped formulas, then there is an analogous deduction from the original formulas. Put another way, the meanings can be added back: a deduction from the stripped formulas can be enriched to become a valid deduction on the original formulas. What is more, there is a unique way to enrich the deduction to work on the original formulas. This claim depends on some properties of $\mathcal{C}$ to be presented shortly.

This result can be strengthened to connect with the λ-calculus: a λ-expression can be associated with each stripped term that will record the meaning information, and the meaning of the inferred term can be determined by performing applications and λ-abstractions in correspondence with the proof rules.

Returning to the example, the stripped meanings for **everyone** and **yawned** with λ-expressions added look like:

$$\begin{array}{rl} \lambda S.every(person, S): & \forall H.\,(g_e \multimap H_t) \multimap H_t \\ yawn: & g_e \multimap f_t \end{array}$$

Implication elimination and universal instantiation in the stripped premises now indicates that the corresponding λ-terms should be applied:

$$\begin{array}{ll} & \lambda S.every(person, S) : \forall H.\,(g_e \multimap H_t) \multimap H_t \\ & yawn : g_e \multimap f_t \\ \vdash & \lambda S.every(person, S) yawn : f_t \end{array}$$

or, equivalently:

$$every(person, yawn) : f_t$$

This formulation looks very similar to the categorical approach, with the crucial difference that the types records the roles that result from an independent syntactic analysis.

4.2 Formal development

The core fragment, $\mathcal{C}$. Informally, the fragment has two kinds of terms, those like f that refer to roles, and those like *Bill* that are in the meaning language. The atoms of the fragment are formulas like $f_e \rightsquigarrow Bill$ that relate the two kinds of terms.

Larger formulas are built up in only two ways: by quantification over roles, and by combination of a quantification over a meaning variable and a linear implication (the two together acting as a function definition). An example is $\forall x.f_e \rightsquigarrow x \multimap g_t \rightsquigarrow yawn(x)$, where the meaning of g is set up as a function of the meaning of f.

Definition 1 (Syntax of $\mathcal{C}$)

$$\begin{array}{rl} \langle \text{e-role} \rangle ::= & \langle \text{e-role const} \rangle \\ \langle \text{t-role} \rangle ::= & \langle \text{t-role const} \rangle \mid \langle \text{t-role var} \rangle \\ \langle \text{role} \rangle ::= & \langle \text{t-role} \rangle \mid \langle \text{e-role} \rangle \\ \langle \text{meaning} \rangle ::= & \langle \text{meaning-const} \rangle \\ & \mid \langle \text{meaning-var} \rangle \\ & \mid \langle \text{meaning} \rangle (\langle \text{meaning} \rangle \ldots \langle \text{meaning} \rangle) \\ \langle \text{formula} \rangle ::= & \langle \text{role} \rangle \rightsquigarrow \langle \text{meaning} \rangle \\ & \mid \forall \langle \text{t-role var} \rangle . \langle \text{formula} \rangle \\ & \mid \forall \langle \text{meaning-var} \rangle . \langle \text{formula} \rangle_1 \multimap \langle \text{formu} \\ & \quad \textit{if } generic(\langle \text{meaning-var} \rangle, \langle \text{formula} \rangle_1 \end{array}$$

Notation: We will use the variables f_t, g_t to stand for t-role constants and f_e, g_e for e-role constants. f, g will stand for any role. Meaning constants will be denoted by m, n. In each case the corresponding capital letters will stand for variables of those classes. $\mathcal{M}, \mathcal{N}$ will be used to denote meanings. P, Q, R will stand for formulas.

$generic(\mathcal{M}, P)$ is a meta-predicate that means that P serves strictly to associate $\mathcal{M}$ to a role or roles, but doesn't say anything about its value. For example, $generic(x, f \rightsquigarrow x)$. Genericity guarantees that the last clause of the syntax of formulas in the syntax of C is defininig what is essentially a function. Formally, genericity is defined by structural induction to hold in exactly the following three cases.

1. $generic(\mathcal{M}, f \rightsquigarrow \mathcal{M})$.

2. $generic(\mathcal{M}, \forall F_t.P)$ if $generic(\mathcal{M}, P)$.

3. $generic(\mathcal{M}, \forall N.P \multimap Q)$ if $generic(\mathcal{M}(N), Q)$.

The last clause in this definition is analogous to eta conversion. For example, it is used in showing that $generic(M, \forall N.f_e \rightsquigarrow N \multimap g_t \rightsquigarrow M(N))$, which ties M to the two roles, f_e and g_t.

To prove our results, we define the role and term projections of formulas.

Definition 2 (Role Projections)

$$\begin{aligned}
\{f_t \rightsquigarrow \mathcal{M}\} &= f_t \\
\{f_e \rightsquigarrow \mathcal{M}\} &= f_e \\
\{\forall F_t.P\} &= \forall F_t.\{P\} \\
\{\forall M.P \multimap Q\} &= \{P\} \multimap \{Q\}
\end{aligned}$$

Definition 3 (Meaning Projections)

$$\begin{aligned}
[\![f \rightsquigarrow \mathcal{M}]\!] &= \mathcal{M} \\
[\![\forall F_t.P]\!] &= [\![P]\!] \\
[\![\forall M.P \multimap Q]\!] &= \lambda M.[\![Q]\!]
\end{aligned}$$

The following lemma verifies the intuition that $generic(\mathcal{M}, P)$ essentially means that the meaning of P is $\mathcal{M}$.

Lemma 4 (Genericity) *If $\mathcal{M}$ is generic in a formula P, then $[\![P]\!] = \mathcal{M}$, up to β, η-conversion.*

Proof. This is obviously true for the first two cases. For the third case $P = \forall N.Q_1 \multimap Q_2$, $[\![\forall N.Q_1 \multimap Q_2]\!] = \lambda N.[\![Q_2]\!]$. By induction, $[\![Q_1]\!] = N$ and $[\![Q_2]\!] = \mathcal{M}(N)$, so $[\![\forall N.Q_1 \multimap Q_2]\!] = \lambda N.[\![Q_2]\!] = \lambda N.\mathcal{M}(N) = \mathcal{M}$. □

Lemma 5 *If $[\![P]\!] = [\![Q]\!]$ and $\{P\} = \{Q\}$ then $P = Q$.*

Proof. By induction, and using the Genericity Lemma, since the only information about P in the third case is $\{P\}$. □

Corollary 6 *If $\mathcal{N}$ is generic in P and $\{P\} = \{Q\}$, then there exists an $\mathcal{M}$ such that $Q = P[\mathcal{M}/\mathcal{N}]$, up to α, β, η-renaming.*

Proof. Take $\mathcal{M}$ to be $[\![Q]\!]$. □

Note that the formulas we have are a subset of linear logic with implication and universal quantification. In our formulas, implication always occurs together with a universal quantification. We can combine the Gentzen style proof rules of linear logic, without any loss of power, to get a simpler proof system for our formulas. This is the proof system $\vdash_{\mathcal{C}}$ that we will use in the following theorems. It is formally defined as follows:

$$\frac{f \rightsquigarrow \mathcal{M} \vdash_{\mathcal{C}} f \rightsquigarrow \mathcal{M}' \quad \text{where } \mathcal{M} \equiv_{\alpha,\eta} \mathcal{M}'}{}$$

$$\frac{\Gamma, P, Q, \Delta \vdash_{\mathcal{C}} R}{\Gamma, Q, P, \Delta \vdash_{\mathcal{C}} R}$$

$$\frac{\Gamma, P[f_t/F_t] \vdash_{\mathcal{C}} R}{\Gamma, \forall F_t.P \vdash_{\mathcal{C}} R} \qquad \frac{\Gamma \vdash_{\mathcal{C}} P[G_t/F_t]}{\Gamma \vdash_{\mathcal{C}} \forall F_t.P}(G_t \text{ new})$$

$$\frac{\Gamma \vdash_{\mathcal{C}} P[\mathcal{M}/M] \quad \Delta, Q[\mathcal{M}/M] \vdash_{\mathcal{C}} R}{\Gamma, \Delta, \forall M.P \multimap Q \vdash_{\mathcal{C}} R}$$

$$\frac{\Gamma, P[N/M] \vdash_{\mathcal{C}} Q[N/M]}{\Gamma \vdash_{\mathcal{C}} \forall M.P \multimap Q}(N \text{ new})$$

It is straightforward to show that one can prove $\Gamma \vdash_{\mathcal{C}} R$ if and only if one can prove $\Gamma \vdash_{LL} R$, where $\vdash_{LL}$ is the linear logic proof system.

The first theorem states that meanings can be stripped off from formulas, with no loss in provability. The second states the converse — given a proof with stripped formulas, it is possible to add the meanings to reconstruct the original proof uniquely.

Theorem 7 *If $P_1, \ldots, P_n \vdash_{\mathcal{C}} R$ then $\{P_1\}, \ldots, \{P_n\} \vdash_{LL} \{R\}$*

Proof. Straightforward induction on the proof of $P_1, \ldots, P_n \vdash_{\mathcal{C}} R$. □

Theorem 8 *Let $P_1, \ldots, P_n$ be formulas in $\mathcal{C}$, and S be any LL formula such that $\{P_1\}, \ldots, \{P_n\} \vdash_{LL} S$. Let Σ be a proof of $\{P_1\}, \ldots, \{P_n\} \vdash_{LL} S$. Then there is a unique formula R with $S = \{R\}$ and a unique proof Π of $P_1, \ldots, P_n \vdash_{\mathcal{C}} R$, such that $\{\Pi\} = \Sigma$, where $\{\Pi\}$ is obtained by projecting each formula in Π.*

Proof. Given the proof Σ of $\{P_1\}, \ldots, \{P_n\} \vdash_{LL} S$, and the meanings of $P_1, \ldots, P_n$, we can show by induction that there is a unique meaning that can be attached to each formula that occurs in Σ. The only non-trivial case is the implication, but genericity allows us to immediately compute the meaning of the antecedent of an implication. It is also clear that for each proof rule of linear logic, the rule formed by adding the meanings is a proof rule in $\vdash_{\mathcal{C}}$. □

We now relate proofs in the resource-based semantics to proofs in the categorial semantics. Since we allow universal quantification over t-roles, we need to extend the typed λ-calculus with type abstraction. Thus, as mentioned earlier, we use a restricted version of System F, a generalization of

the typed λ-calculus (see Girard 1986). We restrict the power of System F by limiting type abstraction to t-role types only.

System F. In the following, Γ is a set of $M : S$ pairs where M is a meaning variable, and S is a role projection. A well formed Γ will have distinct variables (so no variable will have more than one role projection) — thus $M : S \in \Gamma, M : S' \in \Gamma \Rightarrow S =_\alpha S'$. Let S, S' denote role projections, other terms are as before.

The proof rules are:

$$\Gamma, M : S \vdash_F M : S \qquad \text{(identity)}$$

$$\frac{\Gamma, M : S \vdash_F \mathcal{N} : S'}{\Gamma \vdash_F \lambda M.\mathcal{N} : S \multimap S'} \qquad (\lambda\text{-intro})$$

$$\frac{\Gamma \vdash_F \mathcal{M} : S \multimap S' \qquad \Delta \vdash_F \mathcal{N} : S}{\Gamma, \Delta \vdash_F \mathcal{M}(\mathcal{N}) : S'} \qquad \text{(appl)}$$

$$\frac{\Gamma \vdash_F \mathcal{M} : S}{\Gamma \vdash_F \mathcal{M} : \forall F_t.S} \qquad (\Lambda\text{-intro, } F_t \text{ new in } \Gamma)$$

$$\frac{\Gamma \vdash_F \mathcal{M} : \forall F_t.S}{\Gamma \vdash_F \mathcal{M} : S[S'/F_t]} \qquad (\Lambda\text{-elim})$$

Note that we did not need to have $\Lambda F_t.\mathcal{M}$ in the Λ-intro rule since we know that F_t cannot occur in $\mathcal{M}$.

Theorem 9 *Let Π be a proof of $P_1, \ldots, P_n \vdash_C R$. Then for any Γ, $\Gamma \vdash_F [\![R]\!] : \{R\}$ in System F whenever each of $\Gamma \vdash_F [\![P_i]\!] : \{P_i\}$ holds in System F. The converse is also true.*

Proof. By induction on the proof tree for $P_1, \ldots, P_n \vdash R$.

If the last rule applied is the identity rule, there is nothing to prove.

If the last rule is the left $\forall$ quantification over t-roles, then we have a proof of $\Gamma \vdash_F \mathcal{M} : \forall F_t.S$, from our assumption that $\Gamma \vdash_F [\![P_i]\!] : \{P_i\}$. Now we can use the Λ-elimination rule to conclude $\Gamma \vdash_F \mathcal{M} : S[S'/F_t]$. Thus we have a proof for all the premises of the antecedent, so by induction we can prove $\Gamma \vdash_F [\![R]\!] : \{R\}$.

If the last rule was the right $\forall$ quantification over t-roles, then we can use the Λ-intro rule to conclude the result.

If the last rule is the left $\forall$ quantification over meaning variables, then we know that

$$\Gamma \vdash_F [\![P_1]\!] : \{P_1\}, \ldots, \Gamma \vdash_F [\![P_n]\!] : \{P_n\}$$

and

$$\Gamma \vdash_F \lambda M.[\![Q_2]\!] : \{Q_1\} \multimap \{Q_2\}$$

Now from the proof of $P_1, \ldots P_i \vdash Q_1[\mathcal{M}/M]$ and the genericity of $\mathcal{M}$ in Q_1 we have a proof of $\Gamma \vdash_F \mathcal{M} : \{Q_1\}$. Now by the application rule we then have $\Gamma \vdash_F [\![Q_2]\!][\mathcal{M}/M] : \{Q_2\}$. Thus from the induction hypothesis, we can prove $\Gamma \vdash_F [\![R]\!] : \{R\}$.

If the last rule is the right $\forall$ quantification over meaning variables, then $R = \forall N.R_1 \multimap R_2$, for some variable N. $\Gamma \vdash_F [\![P_i]\!] : \{P_i\}$ holds in System F for each premise. As N does not occur in Γ, $\Gamma, N : \{R_1\} \vdash_F [\![P_i]\!] : \{P_i\}$ holds in System F, and $\Gamma, N : \{R_1\} \vdash_F N : \{R_1\}$. Thus, by induction $\Gamma, N : \{R_1\} \vdash_F [\![R_2]\!] : \{R_2\}$. Now by λ-intro, we have $\Gamma \vdash_F \lambda N.[\![R_2]\!] : \{R_1\} \multimap \{R_2\}$, which is the required result.

The linearity assures us that each $\Gamma \vdash_F [\![P_i]\!] : \{P_i\}$ will be needed exactly once in the proof of $\Gamma \vdash_F [\![R]\!] : \{R\}$.

The converse is true, because if we drop the meaning terms from the System F rules we get rules that are valid in LL. Now we can add the meanings via the previous theorem. □

5 Beyond the Core Fragment

The core fragment is sufficient to cover many linguistic constructs, including proper nouns, quantifiers, extensional verbs, and a variety of other phenomena. As the above results show, it requires only propositional inference; it is tractable. Whenever possible, it seems best to express linguistic phenomena within the core fragment.

Some linguistic constructs, however, appear to require going beyond the core fragment. The glue semantics approach has the advantage that it is possible to move beyond the core fragment when that is appropriate, while staying within the well developed linear logic system. In fact, the extensions to date have stayed within an only slightly larger fragment of linear logic.

In this section we will very briefly discuss those situations and point out how they affect the above results.

5.1 Intensionality

Intensional verbs require the meaning language to be able to express intensions. To illustrate, one of the readings for "Bill seeks a unicorn" should be

$$seek(Bill, \hat{}\lambda Q.a(\hat{}unicorn, Q))$$

where the meaning language now is the intensional λ-calculus, which we assume the reader is familiar

with.[1]

Intensional verbs can be handled by *extending* the core fragment to allow the meaning language to include intensional expressions also:

$$\begin{aligned}\langle meaning\rangle ::= \;& \lambda\langle meaning\text{-}var\rangle.\langle meaning\rangle \\ |\; & \hat{}\langle meaning\rangle \\ |\; & \check{}\langle meaning\rangle\end{aligned}$$

This extension allows the above conclusion to be derived from the following contributions of 'Bill', 'seek' and 'a unicorn' (labeled **uni**):

$$\begin{aligned}&\textbf{Bill}: h_e \rightsquigarrow \hat{}Bill \\ &\textbf{uni}: \forall H, N.\ (\forall M.\ h_e \rightsquigarrow M \multimap H_t \rightsquigarrow N(M)) \\ &\qquad \multimap H_t \rightsquigarrow \hat{}a(\hat{}unicorn, \hat{}\lambda M.\check{}(N(\hat{}M))) \\ &\textbf{seek}: \forall M_1, M_2.g_e \rightsquigarrow M_1 \\ &\quad \multimap (\forall H, N_1.(\forall N_2.h_e \rightsquigarrow N_2 \multimap H_t \rightsquigarrow N_1(N_2)) \\ &\qquad \multimap H_t \rightsquigarrow M_2(N_1)) \\ &\quad \multimap f_t \rightsquigarrow seek(M_1, \hat{}\lambda Q.\check{}(M_2\lambda x.\hat{}((\check{}Q)(\check{}x))))\end{aligned}$$

In order to have intensions available when they might be needed, all meaning variables now refer to intension, with extensions explicitly taken whenever necessary. This is a departure from Dalrymple et al. (1997). The equivalent formulation here requires more explicit manipulation of intensions, but stays closest to the core fragment.

Since the only change to the core fragment is to allow a wider meaning language, and since the details of the meaning language do not affect the theorems, the theorems above still apply.

5.2 Non-semantic atoms

Recently, Fry (1996) has explored the use of additional atoms in the semantic contributions that do not carry meaning terms, but that are, instead, used to limit the number of possible readings. These can be used to express restrictions on the appearance of negative polarity items such as 'any' or 'ever', for example. Fry proposes that the meaning constructor for an operator like 'nobody', which can license a negative polarity item, is:

$$\begin{aligned}\textbf{nobody}: \quad \forall H, M. \quad & (\forall N.\ (g_e \rightsquigarrow N \otimes \ell) \\ & \multimap (H_t \rightsquigarrow M(N) \otimes \ell)) \\ & \multimap H_t \rightsquigarrow no(person, M)\end{aligned}$$

The non-semantic atom ℓ constitutes a license for negative polarity items that is available only within the scope of the quantifier 'nobody'. The meaning constructor for a negative polarity item such as the sentential adverb 'ever' is:

$$\textbf{ever}: \quad \forall P. \quad (f_t \rightsquigarrow P \otimes \ell) \multimap (f_t \rightsquigarrow ever(P) \otimes \ell)$$

'Ever' can only be used in the presence of a negative polarity license ℓ, and it reinstates the license so it can be used by other negative polarity items.

The language of the core fragment may be extended to allow these kinds of non-semantic propositions (ns-prop, denoted by ℓ, ℓ') to be interspersed:

$$\begin{aligned}\langle ns\text{-}prop\rangle ::= \;& \langle non\text{-}semantic\ atom\rangle \\ |\; & \langle ns\text{-}prop\rangle \multimap \langle ns\text{-}prop\rangle \\ |\; & \langle ns\text{-}prop\rangle \otimes \langle ns\text{-}prop\rangle \\ \langle formula\rangle ::= |\; & \langle ns\text{-}prop\rangle \multimap \langle formula\rangle \\ |\; & \langle ns\text{-}prop\rangle \otimes \langle formula\rangle\end{aligned}$$

Genericity can be extended to these cases:

1. $generic(\mathcal{M}, \ell \multimap P)$ if $generic(\mathcal{M}, P)$.
2. $generic(\mathcal{M}, \ell \otimes P)$ if $generic(\mathcal{M}, P)$.

These definitions allow a non-semantic atom to be added to any term, either conjoined with the term, or as an antecedent of the term. The idea is that they should play no direct role in determining the meaning, but may constrain what deductions are possible.

Role- and meaning-projections may be defined via:

$$\begin{aligned}\{l \multimap P\} &= l \multimap \{P\} \\ \{l \otimes P\} &= l \otimes \{P\}\end{aligned}$$

$$\begin{aligned}[\![l \multimap P]\!] &= [\![P]\!] \\ [\![l \otimes P]\!] &= [\![P]\!]\end{aligned}$$

Given these definitions, all the results above carry through except the last theorem. The last theorem cannot go through because there are no meanings corresponding to the non-semantic atoms.[2]

6 Discussion

We have presented a reformulation of the glue approach that separates out meaning composition so that it is handled by lambda application, in a way that is very similar to that of the categorial approaches. This better points out the essential difference of the glue approach from categorial approaches: the glue approach doesn't use types to describe syntactic rules, but rather uses types to

[1] Here we are assuming Montague's treatment of intensionality. Other approaches can be handled similarly.

[2] The meaning projection rule $[\![l \multimap P]\!] = [\![P]\!]$ could be changed to $[\![l \multimap P]\!] = \lambda u.[\![P]\!]$, with vacuous abstraction over u, to allow the last theorem to go through. However, this would violate the spirit of the meaning projection, and still wouldn't handle the $[\![l \otimes P]\!]$ case.

connect to a separate grammar, and say how to assemble meanings of sentences that have been analyzed by the grammar. The value of the glue approach is in mediating situations, like quantifier scoping, where the compositional structure of the grammar does not align with the compositional structure of meaning assembly.

The reformulation only applies, however, when the sentences of glue approach stick to a core fragment of linear logic. Some linguistic constructs appear to call for sentences beyond that fragment, meaning that they have expressive requirements not readily available in a categorial style.

References

Bresnan, Joan, editor. 1982. *The Mental Representation of Grammatical Relations.* The MIT Press, Cambridge, MA.

Dalrymple, Mary, Ronald M. Kaplan, John T. Maxwell, III, and Annie Zaenen, editors. 1995a. *Formal Issues in Lexical-Functional Grammar.* Center for the Study of Language and Information, Stanford University.

Dalrymple, Mary, John Lamping, Fernando C. N. Pereira, and Vijay Saraswat. 1995b. A deductive account of quantification in LFG. In Makoto Kanazawa, Christopher J. Piñón, and Henriette de Swart, editors, *Quantifiers, Deduction, and Context.* Center for the Study of Language and Information, Stanford, California.

Dalrymple, Mary, John Lamping, Fernando C. N. Pereira, and Vijay Saraswat. 1995c. Linear logic for meaning assembly. In *Proceedings of CLNLP*, Edinburgh.

Dalrymple, Mary, John Lamping, Fernando C. N. Pereira, and Vijay Saraswat. 1997. Quantifiers, anaphora, and intensionality. *Journal of Logic, Language, and Information.* To appear.

Dalrymple, Mary, John Lamping, and Vijay Saraswat. 1993. LFG semantics via constraints. In *Proceedings of the Sixth Meeting of the European ACL*, University of Utrecht, April. European Chapter of the Association for Computational Linguistics.

Fry, John. 1996. Negative polarity licensing at the syntax-semantics interface. Technical Report ISTL-NLTT-1996-11-01, Xerox PARC, Palo Alto, CA.

Girard, J.-Y. 1986. The system F of variable types, fifteen years later. *Theoretical Computer Science*, 45(2):159–192.

Girard, J.-Y. 1989. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press. Translated and with appendices by Y. Lafont and P. Taylor.

Halvorsen, Per-Kristian. 1983. Semantics for Lexical Functional Grammar. *Linguistic Inquiry*, 14(4):567–615.

Kaplan, Ronald M. and Joan Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations.* The MIT Press, Cambridge, MA, pages 173–281. Reprinted in Dalrymple, Kaplan, Maxwell, and Zaenen, eds., *Formal Issues in Lexical-Functional Grammar*, 29–130. Stanford: Center for the Study of Language and Information. 1995.

Lambek, Joachim. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170.

Montague, Richard. 1974. The proper treatment of quantification in ordinary English. In Richmond Thomason, editor, *Formal Philosophy.* Yale University Press, New Haven.

Oehrle, Richard T., Emmon Bach, and Deirdre Wheeler, editors. 1988. *Categorial Grammars and Natural Language Semantics.* D. Reidel, Dordrecht.

Optimality Theory and Generative Complexity

Robert Frank*
Markus Hiller†
Giorgio Satta‡

1 Introduction

Analyses within generative phonology have traditionally been stated in terms of systems of rewrite rules, which, when applied in the appropriate sequence, produce from an underlying representation the appropriate surface form. This view of phonological analysis has been adopted in almost all work in computational phonology, where the goal has been to produce feasible implementations of phonological rule systems for use in generating surface forms from lexical specifications as well as in parsing surface forms into underlying representations. As first pointed out in [6], the effects of phonological rewrite rules can be simulated using only rational relations (equivalently, finite state transducers) with iterative application accomplished by relation composition, a property under which the class of rational relations is closed. Consequently, since the pioneering work in [7] and [9], computational implementations of phonological rule systems have been done using finite state transducers or extensions thereof.

Recently, there has been a shift in focus in much of the work on phonological theory, from systems of rules to sets of constraints on well-formedness. In this paper, we begin an examination of the effects of the move from rule-based to constraint-based theories upon the generative complexity of

*Department of Cognitive Science, Johns Hopkins University, Baltimore, Maryland, rfrank@vonneumann.cog.jhu.edu

†Seminar fuer Sprachwissenschaft, Universitaet Tuebingen, Tuebingen, Germany, markus.hiller@zdv.uni-tuebingen.de

‡Dipartimento di Elettronica ed Informatica, Università di Padova, Padova, Italy, satta@dei.unipd.it

phonological theories. Specifically, we will focus our efforts on the issue of whether the widely adopted constraint-based view known as Optimality Theory (OT), developed by Prince and Smolensky in [10], may be instantiated in a rational relation (a finite state transducer).[1] OT raises a particularly interesting theoretical question in this context: it allows the specification of a ranking among the constraints and allows lower ranked constraints to be violated in order for higher ranked constraints to be satisfied. This violability property means that certain well-known computational techniques for imposing constraints are not directly applicable.

The contribution of this work is stated in what follows. We present a formalization of OT which embodies that theory's notion of constraint violability rather directly. We make the additional assumptions that the mapping from input to possible output forms (the function GEN) is representable as a rational relation and that each constraint may be represented by means of a total function from strings to non-negative integers, with the requirement that the inverse image of each integer be a regular set. These assumptions suffice to capture most of the current phonological analyses within the OT framework that have been presented in the literature. Under the assumptions above, we prove the following results:

- whenever constraints are represented by means of functions with a finite co-domain (as in the case of so called boolean and multi-valued constraints) then OT systems can be expressed by means of rational relations;

- the fact that optimal surface forms may violate some constraints an unbounded number of times allows the theory to express mappings beyond the generative power of rational relations.

Notice that, because of the separation result above, the result about membership in the class of rational relations is an optimal one, in the sense that dropping the finite co-domain requirement takes us outside of the rational relations.

[1] Work on related matters has been presented in [3] and in [11]. None of these papers addresses the general question of whether the input/output mapping specified by OT can be simulated with finite state machinery.

2 A model of OT

In this section we briefly describe the rudiments of OT and introduce a formalization of this theory whose generative complexity will be investigated in the next section .

As in derivational systems, the general form of phonological computation in OT proceeds from an underlying representation (UR). Such a UR is fed as input to the function GEN which produces as output the set of all possible surface realization (SRs) for this UR, called the candidate set. The notion of a possible SR, as realized in [10], is governed by the containment condition, requiring any SR output by GEN to include a representation of the UR as a (not necessarily contiguous) subpart. Thus, an SR must at a minimum include all of the structure that is specified in the UR, but may also include extra structure absent from the UR, called epenthetic structure. This is not to say that all parts of the input are necessarily pronounced at the surface. Rather, the analog of "deletion" may occur by marking that part of the SR corresponding to the deleted material as unparsed, meaning that it is not visible to the phonetic interface.

The candidate set produced by GEN for any UR will in general be infinite, as there is no bound on the amount of epenthetic material which may be added to the UR to produce the SR. The core of the OT machinery is devoted to choosing among the members of this candidate set to determine which is the actual SR. To do this, OT imposes a set of well-formedness constraints on the elements of the candidate set. Note, however, that these constraints are not imposed conjunctively, meaning that the "winning" SR need not, and most often will not, satisfy them all. Instead, OT allows for the specification of a language particular ranking among the constraints, reflecting their relative importance. The candidate SRs are evaluated with respect to the constraints in a number of stages. At each stage the entire candidate set is subjected to one of the constraints, which stage a constraint is applied being determined by the specified constraint ranking. There are two possible outcomes of such an evaluation. The first arises when some members of the candidate set violate the constraint, but others do not. In this case, the constraint permits us to distinguish among the members of the candidate set: those which do not satisfy the constraint are eliminated from the candidate set and are not considered in subsequent constraint evaluation. (Alternatively, if a constraint can be violated multiple times by a single SR, the relevant evaluation compares the number of violations incurred by each of the SRs in the candidate set. Candidates with the fewest violations

are preferred and those with more violations are eliminated.) The second possible outcome from a constraint evaluation ensues when all of the members of the candidate set violate the constraint to the same degree, perhaps massively or perhaps not at all. When this happens, the constraint does not help us in narrowing down the candidate set. Hence, no candidates are eliminated from the candidate set and violations of the constraint do not block any of them from being considered further to be the actual SR. At the end of the last stage, i.e., when all constraints have been applied, what remains is precisely the subset of the candidate set which are the optimal satisfiers of the constraints under their ranking. This set of candidates, which will often contain only a single member is taken as the set of actual SRs for the original UR.

OT makes the strong assumption that the constraints which are used to evaluate the members of the candidate set are universal, and are therefore active in the phonology of every language. What varies from one language to another is the relative ranking of constraints. Thus, as soon as a commitment is made concerning the set of constraints, there is a concomitant commitment concerning the range of possible typological variation: every ordering of the constraints corresponds to a possible phonological system. This concludes our overview of OT. For further discussion of the formal structure of the model and of its empirical consequences, see [10] and references cited therein.

We now turn to the formalization of OT we work with. We denote as $\mathbf{N}$ the set of non-negative integers.

Definition 1 *An* **optimality system** *(*OS*) is a triple* $G = (\Sigma, \textsc{gen}, C)$, *where* Σ *is a finite alphabet,* GEN *is a function from* Σ^* *to* 2^{Σ^*} *and* $C = \langle c_1, \ldots, c_p \rangle$, $p \geq 1$, *is an ordered sequence of total functions from* Σ^* *to* $\mathbf{N}$.

The basic idea underlying the above definition is as follows. If w is a well-formed UR, $\textsc{gen}(w)$ is the non-empty set of all associated SRs, otherwise $\textsc{gen}(w) = \emptyset$. Each function c in C represents some constraint of the grammar. For some SR w, the non-negative integer $c(w)$ is the "degree of violation" that w incurs with respect to the represented constraint. Given a set of candidates S, we are interested in the subset of S which violates c to the least degree, i.e., whose value under the function c is lowest. To facilitate reference to this subset, we define

$$\text{argmin}_c\{S\} \quad = \quad \{w \mid w \in S,\ c(w) = \min\{c(w') \mid w' \in S\}\}.$$

We can now characterize the map an OS induces. We do this in stages, each one representing the evaluation of the candidates according to one of the

constraints. For each $w \in \Sigma^*$ and for $0 \leq i \leq p$ we define a function from Σ^* to 2^{Σ^*}:

$$\mathrm{OT}_G^i(w) = \begin{cases} \textsc{gen}(w) & \text{if } i = 0; \\ \mathrm{OT}_G^{i-1}(w) & \text{if } i \geq 1 \text{ and} \\ & \mathrm{argmin}_{c_i}\{\mathrm{OT}_G^{i-1}(w)\} = \mathrm{OT}_G^{i-1}(w); \\ \mathrm{argmin}_{c_i}\{\mathrm{OT}_G^{i-1}(w)\} & \text{if } i \geq 1 \text{ and} \\ & \mathrm{argmin}_{c_i}\{\mathrm{OT}_G^{i-1}(w)\} \neq \mathrm{OT}_G^{i-1}(w). \end{cases}$$

Function OT_G^p is called the **optimality function** associated with G, and is simply denoted as OT_G. We drop the subscript when there is no ambiguity.

3 OT and rational relations

In this section we present our two main results. Below we take for granted the notion of regular language and rational relation (see for instance [5] and [4]). We also view a one-to-many rational relation as a function from strings to set of strings, in the usual way.

We make a number of specific assumptions concerning the formal nature of an OS. We assume that GEN is specifiable in terms of a rational relation. In addition, we restrict our attention to sets of constraints C such that, for each c in C and $k \in \mathbf{N}$, the set $\{w \mid w \in \Sigma^*,\ c(w) = k\}$, i.e., the inverse image of k under c, is a regular language. Since the question that we focus on in this research is that of determining whether the class of mappings specifiable in OT is beyond the formal power of rational relations, dropping the above assumptions would decide the question by *fiat*. Furthermore, it turns out that nearly all of the constraints that have been proposed in the OT phonological literature satisfy the above restriction on the inverse image. The reason for this is that OT constraints have tended to take the form of local conditions on the well-formedness of phonological representations, where local means bounded in size. Because of this fact, a phonological representation w attests as many violations of a given constraint c as the number of occurrences of strings from some finite set of configurations V_c appearing as substrings of w. Since V_c is finite, it can be represented through some regular expression, and we can use well-known algebraic properties of regular languages to derive the above condition on the inverse image. (See Tesar [11] for further discussion of a related notion of locality in constraints.)

We now start by presenting a sufficient condition for OS's to be implemented through finite state transducers.

Theorem 1 *Let $G = (\Sigma, \text{GEN}, C)$ be an* OS *such that* GEN *is a rational relation and the inverse image under a constraint in C of an integer is a regular set. Function OT_G can be represented as a rational relation if each constraint in C has a finite co-domain.*

Outline of the proof. First, we restrict our attention to constraints having co-domain of size two. We proceed by induction and assume that for $i \geq 1$ we have already been able to represent OT^{i-1} by means of a rational relation R. Let $L(c_i) = \{w \mid w \in \Sigma^*,\ c_i(w) = 0\}$. Consider some UR w and the set $\text{OT}^{i-1}(w) = \{w' \mid wRw'\} = wR$ of associated candidate SR's that are optimal with respect to OT^{i-1}. To select the strings in this set that are optimal with respect to c_i, we check if there are candidates from wR which are members of $L(c_i)$. If the check is successful, (a) we eliminate any non-satisfying candidates by intersecting wR with $L(c_i)$; otherwise (b) we select the whole set of candidates wR. It can be shown that the set L_+ of all UR's for which the above check is successful is a regular language. Thus we can "split" relation R into two relations R_+ and R_-, where R_+ (R_-) is R with its left projection intersected with L_+ ($\Sigma^* - L_+$, respectively). In order to implement step (a) above, we construct a relation R'_+ by intersecting R_+'s right projection with $L(c_i)$. It is not difficult to see that OT^i is represented by the union of R'_+ and R_-. This relation is rational, since both R'_+ and R_- are rational.

The theorem can be proved using the above result and a construction first suggested in [3] that reduces constraints having arbitrarily large finite co-domain to a finite number of constraint having co-domain of size 2. ∎

We now turn to our second result, and show that the condition on the finite co-domain in Theorem 1 is also a necessary condition.

Theorem 2 *There exists an* OS *$G = (\Sigma, \text{GEN}, \langle c \rangle)$ such that* GEN *is a rational relation, the inverse image under c of an integer is a regular set, c has an infinite co-domain, and function OT_G is not a rational relation.*

Outline of the proof. Let $\Sigma = \{a_1, \ldots, a_n, b\}$, $n \geq 2$. For $1 \leq i \leq n$, let $R_i = \{(w, w') \mid w \in \Sigma^*,\ w'$ is w with each occurrence of a_i replaced by $b\}$, and let $\text{GEN} = \cup_{i=1}^{n} R_i$. Finally, let $c(w) = \#_b(w)$, where $\#_b(w)$ denotes the number of occurrences of b within w. Clearly GEN is a rational relation and c satisfies our assumptions. It is not difficult to see that, for each $w \in \Sigma^*$, $\text{OT}_G(w)$ includes string w with each occurrence of a_i replaced by b iff $\#_{a_i}(w) \leq \#_{a_j}(w)$ for each j, $1 \leq j \leq n$. Let R be the binary relation corresponding to $\text{OT}_G(w)$. The right projection of R with the language

denoted by the regular expression $b^*a_2^* \cdots a_n^*$ is the language $\{b^{i_1}a_2^{i_2} \cdots a_n^{i_n} \mid i_1 \leq i_j,\ 1 \leq j \leq n\}$, which is not regular. Hence R cannot be a rational relation. ■

Observe that, in the proof of Theorem 2, if $|\Sigma| = 2$ then R can be realized by a sintax-directed translation schemata; if $|\Sigma| > 2$, then R is even more powerful that the class of translations that correspons to these devices. Since the OS in the proof uses a single constraint, we can conclude that the source of the detected generative complexity resides in the optimization mechanism of OT.

4 Discussion

We have seen that OT systems including constraints which distinguish among unboundedly many levels of violation may give rise to mappings beyond the power of rational relations, whereas systems including only constraints that distinguish among only finitely many levels of violation remain within the class of rational relations. Karttunen [8] argues on empirical grounds that attested phonological processes which mediate between UR and SR can be modelled by a finite state transducer. Though his argument was given in the context of a discussion of systems of rewrite rules, the conclusion, if correct, is completely general. That is, whether the relation between UR and SR is best characterized in terms of sequences of rewriting steps or in terms of OT optimizations, Karttunen's argument suggests that the generative complexity of the resulting mapping need be no greater than that of rational translations. If this empirical argument is on the right track, our results diagnose a formal deficiency with the OT formal system, namely that it is too rich in generative complexity. In addition, our results also suggest a cure: that constraints should be limited in the number of distinctions they can make in levels of violation. We suspect that following this regimen will require a change in the type of optimizations which carried out in OT, from global optimizations over arbitrarily large representations to local optimizations over structural domains of bounded complexity. We leave a study of the empirical and formal implications of this move for future work.

References

[1] Bird, Steven and T. Mark Ellison. 1994. One-level phonology: Autosegmental representations and rules as finite automata. *Computational*

Linguistics, 20(1):55–90.

[2] Bird, Steven and Ewan Klein. 1994. Phonological analysis in typed feature systems. *Computational Linguistics*, 20(3):455–491.

[3] Ellison, T. Mark. 1994. Phonological derivation in optimality theory. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 1007–1013.

[4] Gurari, Eitan. 1989. *An Introduction to the Theory of Computation.* Computer Science Press, New York, NY.

[5] Hopcroft, John E. and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, Reading, MA.

[6] Johnson, C. Douglas. 1972. *Formal Aspects of Phonological Description.* Mouton, The Hague.

[7] Kaplan, Ronald M. and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378. Written in 1980.

[8] Karttunen, Lauri. 1993. Finite-state constraints. In John Goldsmith, editor, *The Last Phonological Rule.* University of Chicago Press, Chicago, pages 173–194.

[9] Koskenniemi, Kimmo. 1984. A general computational model for word-form recognition and production. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 178–181.

[10] Prince, Alan and Paul Smolensky. 1993. Optimality theory: Constraint interaction in generative grammar. Manuscript, Rutgers University and University of Colorado, Boulder.

[11] Tesar, Bruce. 1995. *Computational Optimality Theory.* Ph.D. thesis, University of Colorado, Boulder.

The Relation Between Tree–Adjoining Grammars and Constraint Dependency Grammars

Karin Harbusch
University of Koblenz–Landau
Computer Science Department, Institute of Computational Linguistics
Rheinau 1, D–56075 Koblenz, Germany
Phone: +49 261 9119 463, Fax: +49 261 9119 465
E–mail: harbusch@informatik.uni-koblenz.de

July 18, 1997

Abstract

For the two grammar formalisms *Tree–Adjoining Grammars (TAGs)* and *Constraint Dependency Grammars (CDGs)*, it is known that they are mildly context–sensitive. In both formalisms, natural–language phenomena can appropriately be specified. TAGs provide an adequate domain of locality by defining trees as rules. In a CDG, more or less strictly formulated local constraints are written which must be satisfiable all over the system. For both, polynomial parsers have been proposed in the literature ($O(n^4)$ for CDGs and $O(n^5)$ for TAGs).

Consequently, it seems reasonable to compare the two formalisms in more detail because each provides convincing features to the grammar writer. In the following paper, we show that each TAG can be transformed into an equivalent CDG but not vice versa. This is shown by outlining how a CDG produces the non Tree–Adjoining language L–6 := $a^n b^n c^n d^n e^n f^n$.

1 Introduction

Tree–Adjoining Grammar (TAG) is a well studied formalism for writing natural–language grammars (see, e.g., [Doran et al. 94] for a large natural–language grammar specified in this formalism; furthermore, see section 2 for a brief introduction).

Some remarkable properties are:

- TAGs define an extended domain of locality, i.e. complex phenomena, e.g. long distance movement (cf. [Kroch, Joshi 85]) can be represented by one and the same rule or tree, respectively.

- TAGs are more powerful than Context–Free Grammars[1] (*mildly context–sensitive* — cf. the TAG for the non–context–free language a^n b^n c^n d^n [Joshi 83]),

[1] Here, *a Context–Free Grammar (CFG)* G = ⟨V,T,S,P⟩ is used as described, e.g., in [Hopcroft, Ullman 79]). V is the finite set of *nonterminals*, T is the finite set of terminals ($V \cap T = \emptyset$), $S \in V$ is the *start symbol* and P is the set of *productions* ($P \subseteq N \times (N \cup T)^*$).
A word $\omega := \mu_1\beta\mu_2$ is *directly derivable* from a word $\nu := \mu_1\alpha\mu_2$, where $\mu_1, \mu_2 \in (N \cup T)^* \iff \alpha \in N$, $\beta \in (N \cup T)^*$ and $(\alpha, \beta) \in P$ (written as $\mu_1\alpha\mu_2 \overset{}{\underset{G}{\Rightarrow}} \mu_1\beta\mu_2$).
Furthermore, a word $\bar{\omega}$ is *derivable* from a word $\bar{\nu} \iff \exists$ series $\omega_1, \ldots, \omega_n$ ($n \in \mathbb{N}$) with $\bar{\omega} := \omega_1 \Rightarrow \omega_2 \Rightarrow \ldots \Rightarrow \omega_n := \bar{\nu}$ (written as $\bar{\mu}_1\bar{\alpha}\bar{\mu}_2 \overset{*}{\underset{G}{\Rightarrow}} \bar{\mu}_1\bar{\beta}\bar{\mu}_2$).
A word $\omega \in T^*$ is *in the language* of the Context–Free Grammar G (L(G)) $\iff \exists$ derivation $S \overset{*}{\underset{G}{\Rightarrow}} \omega$.
If in such a derivation $S \overset{*}{\underset{G}{\Rightarrow}} \omega$ the elements on the left–hand side of each applied rule are interpreted as mother nodes and all elements on the right–hand side are interpreted as daughters preserving their order ¿from left to right a *derivation tree* results.

- the word problem is solvable in polynomial time (direct parsers which run in time $O(n^6)$ are described in [Vijay-Shanker, Joshi 85] or [Schabes 90]; furthermore, a parser for bracket grammars which are equivalent to TAGs runs in time $O(n^5)$ [Guan 92]).

Beside the extended domain of locality, for *Constraint Dependency Grammars (CDGs)* the same properties hold (cf. [Maruyama 90a] and see section 3 for a brief introduction). In a CDG, the domain of locality is specified indirectly by a set of constraints. Each constraint cuts off a specific region of the search space solving the combinatory problem of finding values for all variables. For instance, it allows to address a so called modifier without localizing it. So, a set of solutions results. Further constraints can be put on the modifier (e.g., the modifier is the first word, i.e. $mod(x) = word(pos(1))$, makes the result unique). So, all local solutions are tested to satisfy all other constraints as well. Consequently, the result is a possibly empty set of values.

Since the two formalisms exhibit similar properties it should be discussed whether they are equivalent. In the following paper, we show that each TAG can be transformed into an equivalent CDG by extending the constructive proof by [Maruyama 90b] which transforms a CFG into a CDG. For our purposes, we add a new role *tree*. Basically, it relates all adjacent context–free rules in an elementary tree. Furthermore, it relates dislocated context–free rules which belong to the same elementary tree. Since this constraint overgeneralizes, links between root and foot node of an auxiliary tree are introduced similar to the ordering constraints preserving the tree property of context–free rules. Another constraint requires that links can only be nested. It restricts the set of hypotheses to the set of valid TAG derivations. In section 4, the basic context–free transformation and the extensions for TAGs are outlined in more detail. Although the comparison of the two formalisms gives the impression that they are equivalent it is simple to construct a CDG for L–6 $:= a^n b^n c^n d^n e^n f^n$ ($n \geq 1$) which is known not to be a Tree–Adjoining language (TAL). This fact is discussed in section 4.4.

The paper ends addressing some open questions and future work.

2 The Formalism of TAGs

In 1975, the formalism of *Tree Adjoining Grammars* (*TAGs*) was introduced by *Aravind K. Joshi*, *Leon S. Levy* and *Masako Takahashi* (cf. [Joshi et al. 75]). Since then, a wide variety of properties — formal properties as well as linguistically relevant ones — have been studied (see, e.g., [Becker 93] or [Doran et al. 94] for an overview to the recent literature).

A TAG G = (V, T, S, I, A) is a tree–generation system. It consists, in addition to the set of *nonterminals* V, the set of *terminals* T ($V \cap T = \emptyset$), and the *start symbol* S, an extraordinary symbol in V, of two different sets of trees (I and A, called the set of *elementary trees*), which specify the rules of a TAG. Intuitively, the set I of *initial trees* can be seen as context-free derivation trees. This means the start symbol is the root node, all inner nodes are nonterminals and all leaves are terminals. The second set A, the *auxiliary trees*, which can replace a node in an initial tree (which is possibly modified by further adjoinings) during the recursion process, must have the form, so that again a derivation tree results. Accordingly, as in an initial tree all inner nodes are labelled with nonterminals, and beside a special leaf (the *foot node*) all leaves a terminals. The foot node is labelled with the same nonterminal as the root node. Furthermore, it is obligatory that an auxiliary tree derives at least one terminal (i.e. for the leaf string holds $\subseteq (T^+ \; V \; T^* \cup T^* \; V \; T^+)$). During the recursion process *adjoining* of an auxiliary tree β in the inner node X of α, the inner node X — which can also be the root node — of the initial tree α (which can possible be modified by further adjoinings) is eliminated. The incoming edge of X now ends in the root node of the auxiliary tree β which is labelled with the same nonterminal as X and all outgoing edges of X now start in the foot node of β.

The set of all initial trees modified by an arbitrary number of adjoinings (at least zero) is called T(G), the *tree set* of a TAG G.[2] L(G), the *language* of a TAG, is defined as the set containing all

[2]The elements in this set can also be specified by building a series of triples (α_i, β_i, X_i) ($0 \leq i \leq n$) — the *derivation tree* — where $\alpha_0 \in I$, α_i ($1 \leq i \leq n$) is the result of the adjoining of β_{i-1} in node X_{i-1} in α_{i-1}, β_i ($0 \leq i \leq n\text{-}1$) is the auxiliary tree, which is adjoined in node X_i in tree α_i and X_i ($0 \leq i \leq n\text{-}1$) is a unique node number in α_i. This description has the advantage that structurally equal trees in T(G) which result from different adjoinings can uniquely be represented.

leaf strings of trees in T(G) or all trees which can be constructed by adjoining as described in the corresponding derivation, respectively. Here, a leaf string means all labels of leaves in a tree are concatenated in order from left to right.

3 The Formalism of CDGs

The formalism of *Constraint Dependency Grammars (CDGs)* was introduced by *Hiroshi Maruyama* in 1990 (cf. [Maruyama 90a] and [Maruyama 90b]).

Formally speaking, a CDG is a four–tuple ⟨ T, R, L, C⟩ where T is the finite alphabet. Let R = $\{r_1, r_2, ..., r_k\}$ be a finite set of *role-ids*. Roles are like variables, and each role can have a pair $\langle a, d\rangle$ as its value, where the *label* a is a member in the finite set L = $\{a_1, a_2, ..., a_l\}$ and the *modifiee* d is either $1 \leq d \leq n$ or a special symbol *nil*.

C is a *constraint* that an assignment should satisfy. A constraint C is a logical formula in a form $\forall x_1, ..., x_p$: *role*; $P_1 \& ... P_m$ where $x_1, ..., x_p$ are variables and range over the set of roles in an assignment A. Each subformula P_i consists of the following vocabulary:

- variables $x_1, ..., x_p$,
- constants: elements and subsets of T ∪ L ∪ R ∪$\{nil, 1, 2, ...\}$,
- function symbols: $word, pos, rid, lab, mod$,
- predicate symbols: =, <, >, ∈, and
- logical connectors: &, |, ¬, ⇒.

Let a sentence $s = w_1 w_2 ... w_n$ be a finite string on a finite alphabet T. Let R and L be defined as before. The *degree* of a CDG G is the size k of the role–id set R. Suppose that each word w_i (addressed by the position i because the same word can occur in many different positions in a sentence) in a sentence s has k different roles $r_1(i), r_2(i), ..., r_k(i)$. Roles are like variables, and each role can have a pair $\langle a, d\rangle$ as its value. An *analysis* of the sentence s is obtained by assigning appropriate values to the $n \times k$ roles.

An *assignment* A of a sentence s is a function which assigns values to the roles. Given an assignment A, the label and the modifiee of a role x are determined. The following four functions are defined to represent various aspects of the role x assuming that x is an r_j–role of the word at position i (written as $word(i)$):

- $pos(x)$:=the position i,
- $rid(x)$:=the role id r_j,
- $lab(x)$:=the label of x and
- $mod(x)$:=the modifiee of x.

Specifically, a subformula P_i is called a *unary constraint* if P_i contains only one variable, and a *binary constraint* if P_i contains exactly two variables. If all subformulae of C are unary or binary constraints the grammar is of *arity 2*. In the following, only grammars of arity 2 are addressed.

Finally[3], the word problem for CDGs can be defined as follows. A nonnull string s over the alphabet T is *generated* iff there exists an assignment A that satisfies the constraint C. It can be shown that the runtime of the parser for CDGs of arity 2 is $O(n^4)$ (cf. [Maruyama 90a]).

4 The Relation between TAGs and CDGs

In this section, we discuss whether TAGs and CDGs are equivalent. It can be shown (cf. 4.3) that a TAG can be transformed into a CDG but as outlined in section 4.4 the other direction does not work. This is shown by constructing a CDG for a non–TAL. In the following, the transformation of an arbitrary Context–Free Grammar G in *Greibach–Normal Form* into a CDG is presented because the transformation for TAGs extends this proof. Since both proofs require a grammar in Greibach–Normal Form, in section 4.2, the idea how to transform a TAG into Greibach–Normal Form is described. In section 4.3, the transformation of an arbitrary TAG in Greibach–Normal Form into a CDG is presented.

[3]Since here is not the space for an illustrating example, the reader is referred to section 4.4 for a simple grammar written in this formalism.

4.1 How to transform a CFG into a CDG

In [Maruyama 90b] the constructive proof is described how to transform an arbitrary Context-Free Grammar G in *Greibach-Normal Form* (i.e. the finite set of productions P $\subseteq$ V $\times$ (T V*) and $\epsilon \notin$ L(G))[4] into a CDG of degree 2 and arity 2. The basic idea of the construction described below is to define two roles *head* and *body*[5] (degree 2). The two roles differentiate whether the element occurs on the left-hand side or the right-hand side of a rule. Constraint (2) defines the "shake-hand" of these two interpretations iff they refer to the same nonterminal. Furthermore for each context-free rule, all nonterminals on the right-hand side are linked to become a "circle" as described in subformula (3) (cf. figure 1). By this method, all nonterminals on the right-hand side are determined uniquely and are applied in their order from left to right. Together with further constraints specifying that all links are nested[6] (cf. (6)) and some specific constraints reading a lexicon entry by interpreting it as a head role (i.e. termination of the recursion, cf. (1)) and finally another specific rule for the start symbol (cf. (5)), the CDG simulating a Context-Free Grammar is complete and looks as follows:

Let us consider $G_1 = \langle T_1, R_1, L_1, C_1 \rangle$ where T_1 = the alphabet of the CFG, $R_1 = \{head, body\}$, $L_1 = \bigcup X_B \cup X^H \cup \{\#\}$ where $X \in V$, the set of nonterminals. Let $X_1, X_2, \ldots$ be the occurrences of the nonterminal X on the left-hand side of the context-free production rules which are called a *left occurrence of X*. The set of all left occurrences of X is written as X_B. Similarly, X^H is the set of all *right occurrences*. Consequently, each rule of the Context-Free Grammar can be rewritten by replacing each occurrence of a nonterminal symbol by the indexed one, so that every X_* and X^* — subscript and superscript asterisk represents appropriate indices — appears only once in all the production rules in P — which states a necessary prerequisite for CDGs.

The constraint C_1 is a conjunction of the following conditions (x and y are universally quantified):

(1) (Lexical)
$(rid(x) = head \& word(x) = a \Rightarrow \exists$ a rule p: (p) $X_{0*} \rightarrow aX_1^*X_2^*...X_m^*(0 \leq m) \& lab(x) = X_{0*})$.

(2) (head-role — body-role consistency)
$(pos(x) = pos(y) \& rid(x) = head \& rid(y) = body \& lab(x) = X_* \Rightarrow lab(y) \in X^R \cup \{\#\})$.

(3) (Loop)
$((rid(x) = head \& rid(y) = body \& mod(x) = pos(y) \& lab(x) = X_{0*} \& \exists$ p: (p) $X_{0*} \rightarrow aX_1^*X_2^*...X_m^*$
$\Rightarrow pos(x) < pos(y) \& lab(y) = X_m^*)$ &
$(rid(x) = body \& lab(x) = X_i^* \& rid(y) = body \& mod(x) = y \ (2 \leq i \leq m)$
$\Rightarrow pos(y) < pos(x) \& lab(y) = X_{i-1}^*) \&$
$(rid(x) = body \& lab(x) = X_1^* \& rid(y) = head \& mod(x) = y \Rightarrow pos(y) < pos(x) \& lab(y) = X_{0*}))$.

(4) (Terminal)
$(rid(x) = head \& lab(x) = X_* \& \exists$ p: (p) $X \rightarrow a \Rightarrow mod(x) = nil)$.

(5) *(start symbol)*
$((rid(x) = head \& pos(x) = 1 \Rightarrow lab(x) = S_*) \&$
$(rid(x) = body \Rightarrow (pos(x) = 1 \Leftrightarrow lab(x) = \# \Leftrightarrow mod(x) = nil)))$.

(6) (no crossing)
$((pos(x) < pos(y) < mod(x) \Rightarrow pos(x) \leq mod(y) \leq mod(x)) \&$
$(pos(x) < mod(y) < mod(x) \Rightarrow pos(x) \leq pos(y) \leq mod(x)) \&$
$(mod(x) < pos(y) < pos(x) \Rightarrow mod(x) \leq mod(y) \leq pos(x)) \&$
$(mod(x) < mod(y) < pos(x) \Rightarrow mod(x) \leq pos(y) \leq pos(x)))$.

(7) (uniqueness of modifiee with the same label)
$(mod(x) = mod(y) \Rightarrow x = y)$.

For an illustration of rule (3) see figure 1. This construction together with rule (6) and (7) filters only correct parse trees.

[4]The proof can probably be modified to work for an arbitrary Context-Free Grammar according to the Earley parser [Earley 70]. However, the test to check for complete derivations, i.e. completion, is more complicated.

[5]Since the two roles annotate a unique node during constraint satisfaction no argument position is necessary for the individual roles.

[6]This constraint preserves the property that a derivation tree without any crossing of edges is produced.

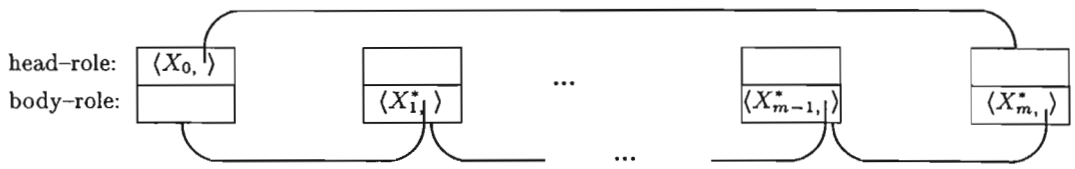


Figure 1: Nested links to preserve the order in a context–free rule

In order to transform a TAG G into a CDG, we basically assume that G is in Greibach–Normal Form. Hence, in the next section the idea how to construct an equivalent TAG in Greibach–Normal Form is described.

4.2 How to transform a TAG into Greibach–Normal Form

A TAG G is in *Greibach–Normal Form* iff each parent–children relation in each elementary tree has the following form. The parent node Z is labelled with a nonterminal, the leftmost daughter is labelled with a terminal and all further children of Z are labelled with nonterminals. I.e. each relation is in Greibach–Normal Form in the context–free sense ($P \subseteq V \times (T\ V^*)$). Furthermore, without loss of generality, $\epsilon \notin L(G)$.

Here is not the space to outline the complete constructive proof. So, basically the crucial cases can be mentioned. Furthermore, an example (cf. figure 2) illustrates the process.

With the finiteness of the set of elementary trees and the set of terminals, it is clear that for all nodes in all elementary trees in time O(1) all terminals in the leftmost position can be predicted[7]. However there remains a subset of the set of auxiliary trees where the foot node is the leftmost leaf of the tree. Obviously, in this case no terminal can be predicted. In order to overcome this problem, new trees where the adjoining of auxiliary trees which are only productive to the right is made explicit in all elementary trees of the grammar. Now, the respective auxiliary trees which are only productive to the right can be replaced by trees which are now productive to the left (cf. β_{12}, α_2 and α_4 illustrating the duplication of trees — note that the two initial trees trees result from the next step — and cf. tree $\beta_{2,2}$ in comparison to β_2 in figure 2). Obviously, exactly the same structures can be produced[8].

Now it is always possible to predict the leftmost terminal for an inner node X. This element becomes the leftmost daughter of X. Furthermore, the structure on the path towards this terminal is preserved as next daughter. Finally, all further daughters remain unchanged when they are nonterminals or a new (non–productive) nonterminal is introduced in order to fulfil the definition (cf. α_1, ..., α_4 in figure 2).

The new nonterminals introducing more depth to the grammar are harmless because this is the well known technique described by [Vijay-Shanker, Joshi 85] transforming a TAG into 2–form. For the termination of the transformation it is important to show that the depth of the second daughter (i.e. the representation of the structure and all its productivity on the path to the terminal is decreasing.

Let us in the following assume that the TAG is in Greibach–Normal Form. Now it is transformed into an equivalent CDG.

4.3 How to transform a TAG in Greibach–Normal Form into a CDG

For our purposes, the above described transformation of a CFG into a CDG (cf. section 4.1) is extended to test for the completeness of all elementary trees in a TAG derivation by the new role

[7] *Predicted* means here the same as in Earley parsing (cf. [Earley 70]) where all rules which can be expanded for the currently considered X are enumerated recursively but without repetitions of circles. Here, only the leftmost position is of interest. Accordingly, only a finite number of paths of finite length are predicted.

[8] Note, that the constructive proof is more complicated for the case that auxiliary trees which are only productive to the right can be adjoined on the path from the root node to the foot node of an auxiliary tree which is itself only productive to the right. Here the same method of making explicit an adjoining which is only productive to the right is repeatedly applied.

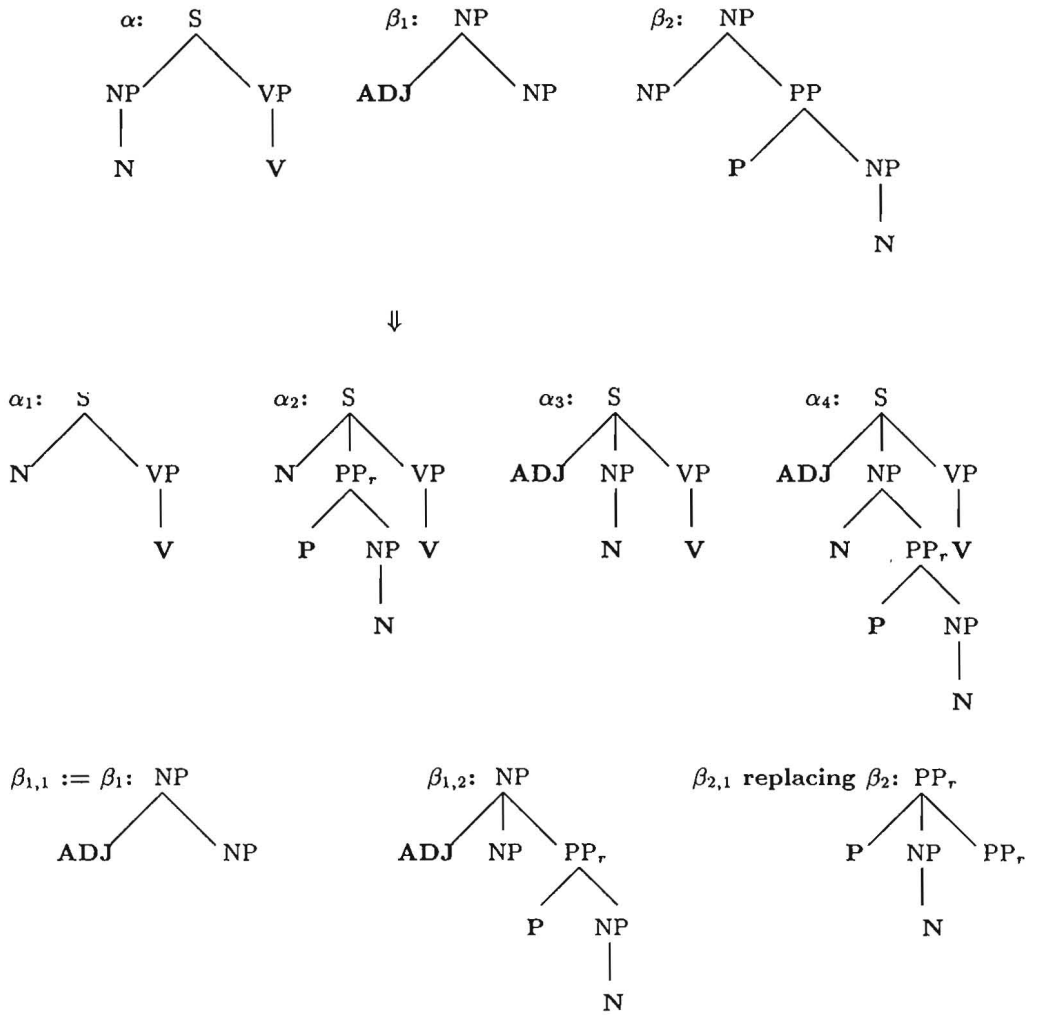


Figure 2: The transformation of an example grammar

tree. Consequently, the degree is 3. But note that the arity remains 2.

In the TAG formalism, only specific context–free rules can be combined. Therefore, the conclusion in subformula (2) is restricted in the following way:

1. $(lab(y) = X^j \& tree(y) = x)$ if X_* and X^j both refer to the same node in the same elementary tree, i.e. two neighboring context–free rules have been identified.

2. $(lab(y) = X^j)$ if j refers to a foot node in the TAG, i.e. an adjoining interrupts the testing for a complete elementary tree. This is expressed by leaving the *tree* role undetermined.

Subformula (3) is extended to relate all *tree* roles in the same way as *head* and *body* are treated. So, the restriction that each element must have a *tree* modifier (subformula (11)) is always satisfied inside a context–free rule. Additionally, the following rules are required:

(8) $((rid(x) = head \& lab(x) = X_k$ where k refers to the root of an auxiliary tree$) \& (rid(y) = body \& lab(y) = X^m$ where m refers to the foot node of the same auxiliary tree$) \Rightarrow tree(x) = y)$.

(9) $(pos(x) = pos(y) \& rid(x) = body \& lab(x) = X^k \& rid(y) = head \& lab(y) = X_m$ where m refers to the root node of an auxiliary tree $\Rightarrow tree(x) = tree(y))$.

(10) $((rid(x) = body \& lab(x) = X^k$ where k does not refer to the foot node of an auxiliary tree$) \& (rid(y) = head \& lab(y) = X_m$ where k and m refer to the same node in an elementary tree $\Rightarrow tree(x) = y)$.

(11) All *tree* links are nested and there exists exactly one *tree* modifier for each element.

These — locally overgenerating — constraints organize the TAG derivation in the following manner. If the derivation consists of an initial tree without adjoinings, subformulae (2) and (3) can only relate the *tree* roles all over the system as if a depth–first left–to–right parser would operate. Iff an adjoining occurs (second alternative of subformula (2)) the foot node remains without a *tree* role. All pointers inside the auxiliary tree realize the "shake–hand" according to (2) until the root node is reached or further adjoinings are detected which prevent the "shake–hand" of the *body* and the *head*. Subformula (8) hypothesizes all candidates for a complete adjoining. Subformula (9) represents that an adjoining — as detected in (8) — can be ignored. Subformula (10) tests whether there exist subtrees of "interrupted" elementary trees (i.e. *heads* which can combine with *bodies* not directly neighboring in the derivation tree). The final subformula restricts the existing hypotheses, which allow for mixing parts of different subtrees, by strictly nesting *tree* modifiers and by making them unique in the same manner it is stated for *mod*. Since it is intuitively clear that the treatment of the *tree* role stretches the "shake–hand" of the *body* – *head* identification in subformula (2), exactly a TAG derivation is stipulated by the system described above. In figure 3 the application of these rules during the identification of an adjoining in a TAG–derivation tree is illustrated.

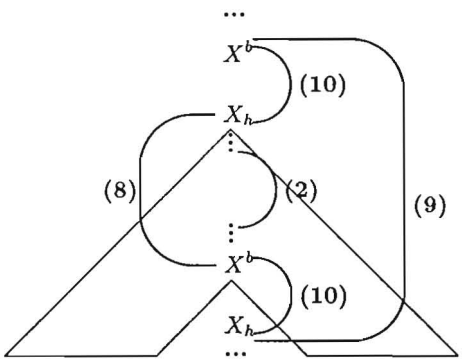


Figure 3: Tree links in a TAG–derivation tree

Consequently, each TAG can be transformed into a CDG and runs the constraint satisfaction process described in [Maruyama 90a] which has a time complexity of $O(n^4)$. Up to now, this result could not be proven for a direct TAG parser. As mentioned in the final section, it is an open question whether the increasing number of rules resulting from transforming the TAG into Greibach–Normal Form would undermine the better runtime.

4.4 A CDG for L–6

Here it is shown that CDGs are more powerful than TAGs. In the following, a CDG for L–6 = $a^n b^n c^n d^n e^n f^n$ ($n \geq 1$) is described.

Let us consider $G_2 = \langle T_2, R_2, L_2, C_2 \rangle$ where $T_2 = \{a, b, c, d, e, f\}$, $R_2 = \{\text{partner}\}$, $L_2 = \text{l}$. The constraint C_2 is a conjunction of the following conditions (x and y are universally quantified):

(1) $(word(pos(x)) = a \Rightarrow word(mod(x)) = f \& pos(x) < pos(mod(x)))$,
i.e. the partner of a is f and f follows a.
(2) $(word(pos(x)) = b \Rightarrow word(mod(x)) = e \& pos(x) < pos(mod(x)))$.
(3) $(word(pos(x)) = c \Rightarrow word(mod(x)) = d \& pos(x) < pos(mod(x)))$.
(4) $(pos(x) < pos(y) < mod(x) \& word(x) \in \{a, b, c\} \Rightarrow pos(x) < mod(y) < mod(x))$,
i.e. only nested partner links pointing from the left to the right can occur.
(5) $(mod(x) = mod(y) \Rightarrow x = y)$, i.e. partner link are unique.
(6) $(word(pos(x)) = f \Rightarrow word(mod(x)) = e, pos(mod(x)) < pos(x))$.
(7) $(word(pos(x)) = e \Rightarrow word(mod(x)) = d, pos(mod(x)) < pos(x))$.
(8) $(word(pos(x)) = d \Rightarrow word(mod(x)) = c, pos(mod(x)) < pos(x))$.
(9) $(mod(x) < pos(y) < pos(x) \& word(x) \in \{d, e, f\} \Rightarrow mod(x) < mod(y) < pos(x))$.

Since here is not the space to show the formal proof we give an intuitive idea why exactly L–6 is generated. Subformulae (1) — (3) relate a and f, b and e, and c and d, respectively. (4) stipulates that all partner links stated in (1) — (3) must be nested. According to constraint (5), a unique structure results. However, the a's, b's and c's remain unordered. Their ordering is realized by the constraints (5) — (9). (5) — (8) associate f and e, e and d, and d and c, respectively. Furthermore, the modifiee must occur to the left of the element. Constraint (9) stipulates crossed links here. Consequently, L–6 is the only language satisfying all constraints.

Obviously, the control of further pairs of terminals in front of a or between c and d can easily be stated. So, a CDG of arity 2 can produce language of the form $a_1^n ... a_{2m}^n$, $a_i \in$ T.

5 Final Discussion

In the paper, we have demonstrated that each TAG can be translated into an equivalent CDG. The other direction does not hold because CDGs are more powerful than TAGs. Consequently, TAGs can indirectly be parsed in $O(n^4)$. The implementation of the constructive proof described in section 4 should answer the open question how much does the grammar constant (which is assumedly cubic after transforming the TAG in Greibach–Normal Form) influence the average runtime of a parser running the constraint–satisfaction algorithm described in [Maruyama 90a] in comparison to a direct TAG parser.

The example in section 4.4 reminds of the class of *set–local Multi–Component TAGs (MC–TAG)* in Weir's dissertation. But is remains an open problem how powerful CDGs are.

References

[Becker 93] T. Becker. HyTAG: A New Type of Tree Adjoining Grammars for Hybrid Syntactic Representation of Free Word Order Languages. PhD thesis, University of the Saarland, Saarbrücken/Germany, 1993.

[Doran et al. 94] C. Doran, D. Egedi, B.A. Hockey, B. Srinivas, M. Zaidel. XTAG System — A Wide Coverage Grammar for English. In the Proceedings of the *15th COLING*, Kyoto/Japan, 1994.

[Earley 70] J. Earley. An efficient context–free parsing algorithm. *Communications of the Association for Computing Machinery*, 13(2):94–102, 1970.

[Guan 92] Y. Guan. A $O(n^5)$ recognition algorithm for coupled parenthesis rewriting systems. In A.K. Joshi, editor. Proceedings of the *2nd International TAG+ Workshop, Philadelphia, PA/USA*, 1992.

[Hopcroft, Ullman 79] J.E. Hopcroft, J.D. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Massachusetts/USA, 1979.

[Joshi 83] A. K. Joshi. Factoring recursion and dependencies : An aspect of Tree Adjoining Grammars (TAG) and a comparison of some formal properties of TAGs and GPSGs and PLGs and LFGs. In Proceedings of the *21st ACL, Cambrigde, MA*, pp. 7–15, 1983.

[Joshi et al. 75] A.K. Joshi, S. Levy, M. Takahashi. Tree Adjoining Grammars. *Journal of Computer and Systems Science*, 6(2):272–284, 1975.

[Kroch, Joshi 85] T. Kroch A.K. Joshi. Linguistic relevance of Tree Adjoining Grammars. Technical report, Philadelphia, PA/USA, 1985. MS–CIS–85–16.

[Maruyama 90a] H. Maruyama. Structural Disambiguation with Constraint Propagation. In the Proceedings of the *28th Annual Meeting of the Association for Computational Linguistics (ACL–90)*, Pittsburgh, Philadelphia/USA, 1990.

[Maruyama 90b] H. Maruyama. Constraint Dependency Grammar. Research Report RT0044, IBM Research, Tokyo Research Laboratory, Tokyo/Japan, 1990.

[Schabes 90] Y. Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, Philadelphia, PA/USA, 1990.

[Vijay-Shanker, Joshi 85] K. Vijay-Shanker, A. K. Joshi. Some computational properties of Tree Adjoining Grammars. In Proceedings of the *23rd ACL, Chicago, IL/USA*, pp. 82–93, 1985.

Compositionality: Similarity versus Interpretability

Herman Hendriks
Utrecht Institute of Linguistics OTS
Trans 10, 3512 JK Utrecht, The Netherlands
+31-(0)30-2536183 (phone)
Herman.Hendriks@let.ruu.nl (e-mail)

Abstract

The present paper studies the general implications of the principle of compositionality for the organization of grammar. It will be argued that Janssen's (1986) requirement that syntax and semantics be *similar* algebras is too strong, and that the more liberal requirement that syntax be *interpretable* into semantics leads to a formalization that can be motivated and applied more easily, while it avoids the technical complications that encumber Janssen's formalization. Moreover, this alternative formalization even allows one to further 'complete' the formal theory of compositionality, in that it is capable of clarifying the role played by *model-theoretic interpretation* and *meaning postulates*, two aspects that received little attention in Janssen (1986) and Montague (1970).

In its most general form, the principle of compositionality states that 'the meaning of an expression is a function of the meanings of its parts and of the way they are syntactically combined' (Partee 1984, p. 281). In other words: the meaning of an expression is determined completely by the meanings of its parts plus the information which syntactic rules have been used to build that expression out of those parts. The principle is also known as 'Frege's principle', but this attribution is at best a tribute according to Janssen (1986), who gives a formalization of the principle which is based on Montague's paper 'Universal Grammar' (1970). Janssen's formalization and the framework defined in Montague (1970) are, roughly speaking, 'different views of the same mathematical object' (Janssen 1986, Part 1, p. 91). The main difference is that Janssen employs many-sorted algebras, whereas Montague uses one-sorted algebras (though with much additional structure). As a consequence of this, Janssen's approach has various advantages (see Hendriks 1993, p. 136). The term 'many-sorted algebra' stems from Adj (1977). We define the notion as follows:[1]

(1) $\langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ is a *many-sorted algebra* of *signature* π (π-algebra) iff
- (*a*) S is a non-empty set (of *sorts*);
- (*b*) $(A_s)_{s\in S}$ is an indexed family of sets (A_s is the *carrier* of s);
- (*c*) Γ is a set (of *operator indices*);
- (*d*) π (the *type-assigning function*) assigns to each $\gamma \in \Gamma$ a pair $\langle\langle s_1, \ldots, s_n\rangle, s_{n+1}\rangle$, where $n > 0$, $s_1 \in S$, $\ldots$, $s_{n+1} \in S$; and
- (*e*) $(F_\gamma)_{\gamma\in\Gamma}$ is an indexed family (of *operators*) such that if $\pi(\gamma) = \langle\langle s_1, \ldots, s_n\rangle, s_{n+1}\rangle$, then $F_\gamma : A_{s_1} \times \ldots \times A_{s_n} \rightarrow A_{s_{n+1}}$.

To cut a long story short, Janssen argues that the compositionality principle dictates the following: (A) the syntax is a π-algebra $A = \langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ with

[1] Our terminology deviates from Janssen (1986), where a pair $\langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ meeting the requirements in (1) is called a 'many-sorted algebra of signature (S, Γ, π)' (1986, Part 1, p. 43).

generating family $H = (H_s)_{s \in S}$; (B) the semantic domain is an ω-algebra $M = \langle (M_t)_{t \in T}, (G_\delta)_{\delta \in \Delta} \rangle$ such that for some $\sigma : S \to T$ and $\rho : \Gamma \to \Delta$: A is (σ, ρ)-similar to M; and (C) meaning assignment is a (σ, ρ)-homomorphism from $T_{A,H}$, the term algebra of A with respect to H, to M.

Ad (A): S is the set of syntactic categories and for each $s \in S$, the set A_s is the set of expressions of category s. Γ is the set of syntactic operator indices and for each $\gamma \in \Gamma$, syntactic operator F_γ of type $\pi(\gamma) = \langle \langle s_1, \ldots, s_n \rangle, s_{n+1} \rangle$ is some total function $A_{s_1} \times \ldots \times A_{s_n} \to A_{s_{n+1}}$ that yields a compound expression a_{n+1} of category s_{n+1} for every sequence $a_1, \ldots, a_n$ of expressions of respective categories $s_1, \ldots, s_n$. And for each $s \in S$, the set H_s is the set of non-compound (lexical) expressions of category s. Ad (B): T is the set of semantic types and for each $t \in T$, the set M_t is the set of semantic objects of type t. Δ is the set of semantic operator indices and for each $\delta \in \Delta$, semantic operator G_δ of type $\omega(\delta) = \langle \langle t_1, \ldots, t_n \rangle, t_{n+1} \rangle$ is some total function $M_{t_1} \times \ldots \times M_{t_n} \to M_{t_{n+1}}$ that yields a semantic object m_{n+1} of type t_{n+1} for every sequence $m_1, \ldots, m_n$ of semantic objects of respective types $m_1, \ldots, m_n$. Algebra A and M are (σ, ρ)-*similar* iff σ and ρ are *bijections* $\sigma : S \to T$ and $\rho : \Gamma \to \Delta$ such that for all $\gamma \in \Gamma$: if $\pi(\gamma) = \langle \langle s_1, \ldots, s_n \rangle, s_{n+1} \rangle$, then $\omega(\rho(\gamma)) = \langle \langle \sigma(s_1), \ldots, \sigma(s_n) \rangle, \sigma(s_{n+1}) \rangle$. Ad (C): the term algebra $T_{A,H}$ of A with respect to H is invoked on account of the phenomenon of (non-lexical) ambiguity, which entails that one cannot in general speak of the meaning of an expression, but only of the meaning of an expression relativized to a certain so-called derivational history. The carriers $T_{A,H,s}$ of this term algebra $T_{A,H} = \langle (T_{A,H,s})_{s \in S}, (F^T_\gamma)_{\gamma \in \Gamma} \rangle$ of signature π consist of symbols which can be seen as representations of the derivational histories of the expressions of the generated syntactic algebra. Finally, the requirement that meaning assignment be a (σ, ρ)-*homomorphism* from $T_{A,H}$ to M means that it has to be a function $f : \bigcup_{s \in S} T_{A,H,s} \to \bigcup_{t \in T} M_t$ such that (*i*) f respects the sorts in that for all s: $f[T_{A,H,s}] \subseteq M_{\sigma(s)}$; and (*ii*) f respects the operators: if $\pi(\gamma) = \langle \langle s_1, \ldots, s_n \rangle, s_{n+1} \rangle$ and $\tau_1 \in T_{A,H,s_1}, \ldots, \tau_n \in T_{A,H,s_n}$, then $f(F_\gamma(\tau_1, \ldots, \tau_n)) = G_{\rho(\gamma)}(f(\tau_1), \ldots, f(\tau_n))$.

In this paper we will argue that the above formalization of the compositionality principle is largely correct—except, however, for a seemingly minor point which will turn out to have rather far-reaching ramifications. We will show that these complications can be solved by replacing the requirement of (σ, ρ)-similarity between the syntactic algebra A and the semantic algebra M by the requirement that the syntactic algebra A be (σ, ρ)-*interpretable* in the semantic algebra M, by which we will mean that σ and ρ have to be *functions* (and not necessarily bijections) $\sigma : S \to T$ and $\rho : \Gamma \to \Delta$ such that for all $\gamma \in \Gamma$: if $\pi(\gamma) = \langle \langle s_1, \ldots, s_n \rangle s, s_n \rangle, s_{n+1} \rangle$, then $\omega(\rho(\gamma)) = \langle \langle \sigma(s_1), \ldots, \sigma(s_n) \rangle, \sigma(s_{n+1}) \rangle$. We will defend this claim by arguing that (σ, ρ)-similarity is too strong from the point of view of explicating the intuitive idea of compositionality, but that, on the other hand, (σ, ρ)-interpretability is a notion that *can* be motivated in this way. Besides, the fact that bijections are functions entails that a π-algebra A is (σ, ρ)-interpretable in ω-algebra M whenever A and M are (σ, ρ)-similar (the converse does not hold, however). Thus (σ, ρ)-similarity is stronger than (σ, ρ)-interpretability, so that the latter notion is more easily applicable in principle. In fact, the requirement that the domains of syntax and semantics constitute similar algebras does lead to actual problems of applicability, since in practice it is generally not the case that there are bijections $\sigma : S \to T$ from the syntactic categories to the semantic types and $\rho : \Gamma \to \Delta$ from the syntactic operator indices to the semantic operator indices that are respected by the meaning assignment homomorphism. Usually, the syntactic and the semantic algebra fail to be (σ, ρ)-similar, since (*a*) some semantic types do not correspond to syntactic categories (so that σ is not surjective); (*b*) different syntactic categories correspond to one and the same semantic type (so that σ is not injective); (*c*) some semantic operators do not figure as the counterpart of a syntactic operator (so that ρ is not

surjective); and/or (*d*) different syntactic operators correspond to one and the same semantic operator (so that ρ is not injective). Moreover, as will be shown below, if a formal logical language is used as an auxiliary translation language, syntactic operators may correspond to semantic operators that—though definable in terms of the operators of the semantic algebra—are themselves not actually present in the semantic algebra (so that ρ is not even a function).

In order to bridge such gaps of dissimilarity between algebras, Janssen invokes the notion of a 'safe deriver'. This notion is introduced in the course of giving a definition of a Montague grammar, which, in its most simple form, consists of a many-sorted algebra and a homomorphic interpretation. However, 'one always uses, in practice, some formal (logical) language as auxiliary language, and the language of which one wishes to define the meanings is translated into this formal language. Thus the meaning assignment is performed indirectly. The aspect of translating into an auxiliary language is, in my opinion, unavoidable for practical reasons, and I therefore wish to incorporate this aspect in the definition of a Montague grammar' (Janssen 1986, Part 1, p. 81). This definition is given in (2), and the situation can be sketched as in (3) (cf. Janssen 1986, Part 1, pp. 75 and 82):

(2) A *Montague grammar* consists of:
a *syntactic* π-algebra $A = \langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ generated by $H = (H_s)_{s\in S}$;
a *logical* ω-algebra $B = \langle (B_t)_{t\in T}, (K_\delta)_{\delta\in\Delta}\rangle$;
a *semantic* ω-algebra $M = \langle (M_t)_{t\in T}, (G_\delta)_{\delta\in\Delta}\rangle$ similar to B;
an *interpretation* homomorphism $\mathcal{I}$ from B to M;
an algebra $D(B)$ similar to A, where D is a *safe deriver*; and
a *translation* homomorphism tr from $T_{A,H} = \langle (T_{A,H,s})_{s\in S}, (F^T_\gamma)_{\gamma\in\Gamma}\rangle$, the term algebra of A with respect to H, to $D(B)$.

(3)
$$\begin{array}{ccc} & & T_{A,H} \\ & & \downarrow tr \\ B & \Longrightarrow & D(B) \\ \downarrow \mathcal{I} & & \downarrow \mathcal{I} \\ M & \Longrightarrow & M' \end{array}$$

In general, a *deriver* D is a function from algebras to algebras: 'a method to obtain new algebras from old ones', and a deriver D is *safe* for algebra P iff for all algebras Q and all surjective homomorphisms $\mathcal{I}$ from P to Q there is a unique algebra Q' such that for the restriction $\mathcal{I}'$ of $\mathcal{I}$ to $D(P)$ it holds that $\mathcal{I}'$ is a surjective homomorphism from $D(P)$ to Q' (Janssen 1986, Part 1, p. 76).[2] Janssen's deriver D is the composition of four basic derivers, viz., AddOp, AddSorts, DelOp and DelSorts,[3] which, by adding operators, adding sorts, deleting operators, and deleting sorts, respectively, transform the logical algebra B into an algebra $D(B)$ = DelSorts(DelOp(AddSorts(AddOp(B)))) which is similar to the syntactic algebra A. As regards the question whether is it really necessary to incorporate this laborious process of deriving a similar algebra $D(B)$ in four steps from the original logical algebra B into the general definition, it can be noted that Janssen emphasizes repeatedly that the possibility of a homomorphism presupposes similarity: 'A mapping is called a homomorphism if it respects the structures of the algebras involved. This is only possible if the two algebras have a similar structure.' (Janssen

[2] Janssen offers no arguments why this should define the safeness of a deriver. The only motivation given is the following: 'The requirement that $\mathcal{I}'$ is a surjective homomorphism is important. If we would not require this, then Q' would in most cases not be unique. An extreme example arises when $D(P)$ is an empty algebra. Then there are infinitely many algebras Q' such that $\mathcal{I}'$ is a homomorphism from $D(P)$ to Q', but only one such that $\mathcal{I}'$ is a surjective homomorphism from $D(P)$ to Q'.' (Janssen 1986, Part 1, p. 76).

[3] In fact, the deriver DelSorts replaces the more complicated and problematic deriver SubAlg actually proposed by Janssen (see Hendriks 1993, p. 162, for motivation).

1986, Part 1, pp. 21–22; see also pp. 67–70). Nevertheless, it can also be observed that if, instead of similarity, interpretability is assumed, we are done in one step: we only need to consider the addition of operators to the logical algebra.

With respect to this aspect of deriving a new syntactic algebra from the syntactic algebra of the logical language, Janssen notes: 'In one respect this attempt [to formalize the compositionality principle] probably has not been successful: the description of how to obtain new algebras out of old ones. There is no general theory which I could use here, and I had to apply *ad hoc* methods.' (Janssen 1986, Part 1, p. 42; see also p. 83). Contrary to this, however, we feel that the appropriate conclusion to be drawn is that the very notion of a 'safe deriver' is *ad hoc*, since it is an artefact created by the requirement of similarity—a requirement which, as we pointed out above, is itself undermotivated in view of the conditions imposed by the compositionality principle. Accordingly, we will now show that the addition of operators to the logical algebra is not, as Janssen puts it, the 'most important' deriver, but the only 'deriver' that has to be taken into account at all.

The basic idea of using a formal logical language as an auxiliary translation language is simply that a term in the term algebra of the generated syntactic algebra is indirectly assigned the interpretation $\mathcal{I}(\beta)$ of the expression β of the logical language that serves as the translation of the term. Thus, each syntactic term t is associated with a unique translation $tr(t)$, and this translation induces the interpretation $\mathcal{I}(tr(t))$: 'the principal use of translations is the semantical one of inducing interpretations' (Montague 1970, p. 232). For such an indirect interpretation assignment to be compositional, the composition $tr \circ \mathcal{I}$ of the translation and interpretation step has to be a homomorphism, i.e., a function, which entails that the logical language must be unambiguous. Therefore, the generated algebra of a logical language is as a rule a *free* algebra.[4]

Furthermore, formal logical languages usually have a *model-theoretic* interpretation, which means that their interpretation homomorphism $\mathcal{I}$ is defined pointwise: on the basis of a class[5] $\mathcal{M}$ of models for the logical language, the interpretation of logical expressions β is specified by separately defining $in_m(\beta)$ for each $m \in \mathcal{M}$. Of course, the point of this model-theoretic set-up is that an expression can have different interpretations in different models: there is not in general a single object that serves as the interpretation of an expression β in all models m. Consequently, in order to be able to talk about 'the' interpretation $\mathcal{I}(\beta)$ of an expression β, one has to incorporate the models into the concept of interpretation: $\mathcal{I}(\beta)$ is that function from models to interpretations in models such that $\mathcal{I}(\beta)(m) = in_m(\beta)$ for all $m \in \mathcal{M}$.

Translations $tr(t)$ of terms t in the term algebra $T_{A,H} = \langle (T_{A,H,s})_{s \in S}, (F^T_\gamma)_{\gamma \in \Gamma} \rangle$ of the generated syntactic algebra are defined by providing a mapping tr which (i) associates each term t that corresponds to a generator h of category s in the syntactic algebra with some expression of type $\sigma(s)$ in the logical algebra $B = \langle (B_t)_{t \in T}, (K_\delta)_{\delta \in \Delta} \rangle$; and (ii) associates each term algebra operator F^T_γ of type

[4] In conformity with L.T.F. Gamut's adage that 'Logical languages wear their meanings on their sleeves' (p.c.).

[5] Here the word 'class' is used deliberately rather than 'set', since the collection of models for a logical language is generally not a set in the sense of axiomatic set theory. In typed logic, for instance, each non-empty set E gives rise to a distinct domain $D_{E,e}$, so that there are at least as many frames—and, consequently, models—as there are (non-empty) sets. This means that the collection $\mathcal{M}$ of models is too large to be countenanced as a set: it is a proper class. Moreover, if $\mathcal{M}$ is a proper class, then the interpretations $\mathcal{I}(\beta)$ that will be defined below must be proper classes as well: these collections contain for all $m \in \mathcal{M}$ exactly one ordered pair $\langle m, a \rangle$ and are, hence, just as large as $\mathcal{M}$. Finally, proper classes do not correspond to set-theoretical objects, so they cannot be constituents of sets and ordered pairs (which are a special kind of sets). Therefore, also the notions $\Phi^{\mathcal{I}}_\gamma$, $K^{\mathcal{I}}_\delta$, $\mathcal{I}_t$, $\mathcal{S}$ and $\mathcal{S}'$, which will be defined below in terms of $\mathcal{I}(\beta)$, do not necessarily correspond to sets. The use of calligraphic letters for these notions is meant to visualize the set-theoretical proviso of the present footnote.

$\langle\langle s_1,\ldots,s_n\rangle,s_{n+1}\rangle$ with some function $\Phi_\gamma : B_{\sigma(s_1)} \times \ldots \times B_{\sigma(s_n)} \to B_{\sigma(s_{n+1})}$, whereby $tr(F_\gamma^T(t_1,\ldots,t_n))$ is defined as $\Phi_\gamma(tr(t_1),\ldots,tr(t_n))$. It is worth mentioning that the logical algebra is usually exploited 'at a higher level' in the process of translation. Thus, terms corresponding to generators of the syntactic algebra need not be translated into generators of the logical algebra. And, more importantly, the functions Φ_γ associated with the operators F_γ^T of the term algebra do not necessarily coincide with the operators K_δ that are actually present in the logical algebra. Let $\Phi_\gamma : B_{\sigma(s_1)} \times \ldots \times B_{\sigma(s_n)} \to B_{\sigma(s_{n+1})}$ be such a function. Define $\Phi_\gamma^{\mathcal{I}}$, *the relation $\mathcal{I}$-induced by Φ_γ*, as the class $\{\langle\langle\mathcal{I}(\beta_1),\ldots,\mathcal{I}(\beta_n)\rangle,\mathcal{I}(\beta_{n+1})\rangle \mid \langle\langle\beta_1,\ldots,\beta_n\rangle,\beta_{n+1}\rangle \in \Phi_\gamma\}$, and call a function Φ_γ *$\mathcal{I}$-functional* iff $\Phi_\gamma^{\mathcal{I}}$ is a function.[6] Moreover, let $\mathcal{I}_t = \{\mathcal{I}(\beta) \mid \beta \in B_t\}$, where $B = \langle(B_t)_{t\in T},(K_\delta)_{\delta\in\Delta}\rangle$ is the logical algebra. Of course, for all $\delta \in \Delta$, K_δ is $\mathcal{I}$-functional, so that $\mathcal{I}$ is a (surjective) homomorphism from B to the semantic algebra $\mathcal{S} = \langle(\mathcal{I}_t)_{t\in T},(K_\delta^{\mathcal{I}})_{\delta\in\Delta}\rangle$. As for the compositionality of an indirect interpretation assignment in terms of a translation homomorphism tr from the syntactic term algebra $T_{A,H} = \langle(T_{A,H,s})_{s\in S},(F_\gamma^T)_{\gamma\in\Gamma}\rangle$ to the 'derived' logical algebra $B' = \langle(B_t)_{t\in T},(\Phi_\gamma)_{\gamma\in\Gamma}\rangle$ and an interpretation homomorphism $\mathcal{I}$ from the logical algebra $B = \langle(B_t)_{t\in T},(K_\delta)_{\delta\in\Delta}\rangle$ to the semantic algebra $\mathcal{S} = \langle(\mathcal{I}_t)_{t\in T},(K_\delta)_{\delta\in\Delta}\rangle$, note that the structure $\mathcal{S}' = \langle(\mathcal{I}_t)_{t\in T},(\Phi_\gamma^{\mathcal{I}})_{\gamma\in\Gamma}\rangle$ is an algebra—and $\mathcal{I}$, consequently, a homomorphism from B' to $\mathcal{S}'$—if and only if for all $\gamma \in \Gamma$, Φ_γ is $\mathcal{I}$-functional.[7] As the composition of two homomorphisms is again a homomorphism, we know then that $tr \circ \mathcal{I}$ is a homomorphism from $T_{A,H}$ to $\mathcal{S}'$.

This raises a question: which operators $\Phi_\gamma : B_{t_1} \times \ldots \times B_{t_n} \to B_{t_{n+1}}$ are $\mathcal{I}$-functional, given a homorphism $\mathcal{I}$ from logical algebra $B = \langle(B_t)_{t\in T},(K_\delta)_{\delta\in\Delta}\rangle$ to semantic algebra $\mathcal{S} = \langle(\mathcal{I}_t)_{t\in T},(K_\delta^{\mathcal{I}})_{\delta\in\Delta}\rangle$? A partial answer to this question is that the class of operators that are $\mathcal{I}$-functional for *all* $\mathcal{I}$ includes the *polynomial* operators over the algebra B. The class of polynomial operators over B consists of elementary operators—projection functions and constant functions—plus operators that are definable as compositions of these elementary operators and the operators in $(K_\delta)_{\delta\in\Delta}$. On the other hand, it is also obvious that for a *particular* homomorphism $\mathcal{I}$ from the logical algebra B to a *specific* semantic algebra $\mathcal{S}$, there are always non-polynomial $\mathcal{I}$-functional operators.

Nonetheless, there are good reasons for disregarding operators over the logical algebra B that are only $\mathcal{I}$-functional for *some* homorphism $\mathcal{I}$. For even though formal logical languages B usually come with a particular class of models $\mathcal{M}$ which determines a specific semantic algebra $\mathcal{S}$ and homorphism $\mathcal{I}$ from B to $\mathcal{S}$,[8] this is generally not the class of models in which the translations of the expressions in the syntactic term algebra are interpreted. Most Montague grammar fragments contain a set MP of so-called *meaning postulates*, sentences of the logical language[9] which are intended to reduce the class $\mathcal{M}$ of all models to the subclass $\mathcal{M}^{MP}$ of models in which all meaning postulates in MP are true (or valid, in the case

[6] I.e., iff there are no $\langle\langle\varepsilon_1,\ldots,\varepsilon_n\rangle,\varepsilon\rangle \in \Phi_\gamma^{\mathcal{I}}$ and $\langle\langle\varepsilon'_1,\ldots,\varepsilon'_n\rangle,\varepsilon'\rangle \in \Phi_\gamma^{\mathcal{I}}$ with $\langle\varepsilon_1,\ldots,\varepsilon_n\rangle = \langle\varepsilon'_1,\ldots,\varepsilon'_n\rangle$ and $\varepsilon \neq \varepsilon'$.

[7] Strictly speaking, it is not the $\Phi_\gamma : B_{\sigma(s_1)} \times \ldots \times B_{\sigma(s_n)} \to B_{\sigma(s_{n+1})}$ themselves, but their restrictions $\Phi_\gamma|tr = \Phi_\gamma \cap ((tr[T_{A,H,s_1}] \times \ldots \times tr[T_{A,H,s_n}]) \times tr[T_{A,H,s_{n+1}}])$ which must be $\mathcal{I}$-functional for $tr \circ \mathcal{I}$ to be a homomorphism. But in view of the fact that every $\mathcal{I}$-functional $\Phi : tr[T_{A,H,s_1}]\times\ldots\times tr[T_{A,H,s_n}] \to tr[T_{A,H,s_{n+1}}]$ can be extended to an $\mathcal{I}$-functional $\Phi' : B_{\sigma(s_1)}\times \ldots \times B_{\sigma(s_n)} \to B_{\sigma(s_{n+1})}$ (simply avoid non-equivalent values for equivalent arguments outside $tr[T_{A,H,s_1}] \times \ldots \times tr[T_{A,K,s_n}]$), there is for every non-$\mathcal{I}$-functional $\Phi_\gamma : B_{\sigma(s_1)} \times \ldots \times B_{\sigma(s_n)} \to B_{\sigma(s_{n+1})}$ with $\mathcal{I}$-functional $\Phi_\gamma|tr$ an $\mathcal{I}$-functional $\Phi'_\gamma : B_{\sigma(s_1)} \times \ldots \times B_{\sigma(s_n)} \to B_{\sigma(s_{n+1})}$ such that $\Phi'_\gamma|tr = \Phi_\gamma|tr$. The latter entails that an algebra $B' = \langle(B_t)_{t\in T},(\Phi_\gamma)_{\gamma\in\Gamma}\rangle$ that mediates in an indirect interpretation homomorphism $tr \circ \mathcal{I}$ can always be replaced by an algebra $B'' = \langle(B_t)_{t\in T},(\Phi'_\gamma)_{\gamma\in\Gamma}\rangle$ in which all operators are $\mathcal{I}$-functional.

[8] There is, however, some latitude here. E.g., typed logics have 'standard' as well as 'generalized' models, etc.

[9] Given their function of reducing the class of models, for that matter, it does not even seem essential that meaning postulates are expressions of (or expressible in) the logical language.

of intensional logics). The interpretation $\mathcal{I}(\beta)$ of logical expressions β is reduced accordingly: $\mathcal{I}^{MP}(\beta) = \{\langle m, in_m(\beta)\rangle \mid m \in \mathcal{M}^{MP}\}$. Let $\mathcal{I}_t{}^{MP} = \{\mathcal{I}^{MP}(\alpha) \mid \alpha \in B_t\}$. Then $\mathcal{I}^{MP}$ can be construed as a homomorphism from the logical algebra B to the semantic algebra $\mathcal{S}^{MP} = \langle(\mathcal{I}_t{}^{MP})_{t\in T}, (K_\delta^{\mathcal{I}^{MP}})_{\delta\in\Delta}\rangle$. The addition of meaning postulates affects the class of $\mathcal{I}$-functional operators in a fairly inscrutable manner: given an initial homomorphism $\mathcal{I}$ and some set MP of meaning postulates, the $\mathcal{I}^{MP}$-functionality of an operator over a logical algebra B cannot be predicted from its $\mathcal{I}$-functionality.[10] Hence it is a safe strategy to allow only those operators over B which are $\mathcal{I}$-functional for *all* homomorphisms $\mathcal{I}$. We noted above that the class of these universally $\mathcal{I}$-functional operators always includes the polynomial operators over the logical algebra B. Moreover, for the languages of typed logic which are commonly used in Montague grammar fragments and whose syntax constitutes a free algebra B in which each type contains infinitely many generators (viz., the variables of that type), it can be shown that the polynomial operators over B actually exhaust the class of universally $\mathcal{I}$-functional operators.[11] Let the *polynomial closure* $\Pi(B)$ of $B = \langle(B_t)_{t\in T}, (K_\delta)_{\delta\in\Delta}\rangle$ be the algebra $\langle(B_t)_{t\in T}, \mathrm{POL}^B\}\rangle$, where POL^B denotes the set of polynomial operators over B. When we incorporate the restriction to universally $\mathcal{I}$-functional—i.e., polynomial—operators, we eventually arrive at the situation sketched in (4):

(4)

$$\begin{array}{ccc} & & T_{A,K} \\ & & \downarrow tr \\ B & & \Pi(B) \\ \downarrow \mathcal{I} & & \downarrow \mathcal{I}^{MP} \\ \mathcal{S} & & \Pi(\mathcal{S}^{MP}) \end{array}$$

Summing up, the main advantage of the picture sketched in (4) over the approach outlined in (3) above seems to be that there is no need for a separate process of explicitly deriving algebras. On the one hand, there is a model-theoretically interpreted logic which determines the translation algebra. On the other hand, there is a grammar fragment consisting of a syntactic algebra, a translation homomorphism from its term algebra to the translation algebra, and a set of meaning postulates. Given the grammar fragment, both the interpretation algebra and the interpretation epimorphism from the translation algebra to the interpretation algebra are induced automatically. This makes the relationship between the grammar of our fragment and the logic that we use in specifying its semantics not only more perspicuous, but also more general: there is no need to readjust our logical tools to every fragment in which we may wish to employ them, in keeping with Montague's idea, who 'viewed the use of an intermediate language as motivated by [...] the expectation (which has been amply realized in practice) that a sufficiently well-designed language such as his Intensional Logic with a known semantics could provide a convenient tool for giving the semantics of various fragments of various natural languages' (Partee and Hendriks 1997, p. 24).

[10] Some results in this area can be distilled from Van Benthem (1980), Section 3.

[11] A proof of this result which originates from F. Wiedijk is presented in Appendix 1 of Janssen (1986, Part 1, pp. 189–192) (cf. Van Benthem (1980), footnote 7, for a one-sorted counterpart). It can be noted that an operator Φ_γ over algebra B is universally $\mathcal{I}$-functional iff the deriver $\mathrm{AddOp}[\{\Phi_\gamma\}]$ is *safe* for B in the sense of the definition quoted above, but that, contrary to what Janssen's motivation for safeness (see footnote 2 above) expresses, it is not so much the ***uniqueness*** as the ***existence*** of the algebra Q' which is at stake.

References

• Adj (J.A. Goguen, J.W. Thatcher, E.G. Wagner, J.B. Wright) (1977). 'Initial Algebra Semantics and Continuous Algebras'. *Journal of the Association for Computing Machinery* 24, 68–95.

• Benthem, J. van (1980). 'Universal Algebra and Model Theory. Two Excursions on the Border.' Report ZW–7908. Department of Mathematics, Groningen University.

• Gamut, L.T.F. (1991). *Logic, Language and Meaning. Volume I: Introduction to Logic. Volume II: Intensional Logic and Logical Grammar*. University of Chicago Press, Chicago and London.

• Hendriks, H. (1993). *Studied Flexibility. Categories and Types in Syntax and Semantics*. ILLC Dissertation Series 1993–5. Institute for Language, Logic and Computation, University of Amsterdam.

• Janssen, T. (1986). *Foundations and Applications of Montague Grammar. Part 1: Philosophy, Framework, Computer Science. Part 2: Applications to Natural Language*. CWI Tracts 19 and 28, Amsterdam.

• Montague, R. (1970). 'Universal Grammar'. *Theoria* **36**, 373–398. Page references concern the reprint as Chapter 7 of R. Thomason (ed.) (1974), *Formal Philosophy. Selected Papers of Richard Montague*. Yale University Press, New Haven.

• Partee, B. (1984). 'Compositionality'. In F. Landman and F. Veltman (eds.) (1984), *Varieties of Formal Semantics. Proceedings of the Fourth Amsterdam Colloquium*. Foris, Dordrecht.

• Partee, B., and H. Hendriks (1997). 'Montague Grammar'. Chapter 1 in J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, Elsevier Science Publishers, Amsterdam, 5–92.

A Dependency-based Approach to Bounded & Unbounded Movement

Mark Hepple

Dept. of Computer Science
University of Sheffield
Regent Court, Portobello Street
Sheffield S1 4DP, UK
hepple@dcs.shef.ac.uk

1 Introduction

This paper addresses the treatment of movement phenomena within *multimodal* categorial, or type-logical, grammar systems. Multimodal approaches allow different modes of logical behaviour to be displayed within a single system. Intuitively, this characteristic corresponds to making available different modes of linguistic description within a single formalism. A key benefit of taking a multimodal approach is that it allows us to choose, for any linguistic phenomenon addressed, a level of description that encodes only the aspects of linguistic structure that are relevant to the treatment of that phenomenon. In practice, this means that we may lexically encode linguistic information which is relevant to one phenomenon but not another, but can discard such information where it is not needed. This characteristic means that the analysis of each phenomenon need focus only on relevant distinctions, allowing analyses to be simpler and more elegant.

In this paper, we are concerned particularly with how locality constraints on movement should be handled, both for bounded and unbounded movement cases. A central claim of the paper is that the treatment of such locality conditions requires representations that encode dependency (i.e. head-dependent distinctions).

2 Multimodal Categorial Grammar

The multimodal categorial approach used here makes available multiple modes of construction, realised in syntax via different product operators $\circ_\alpha$ (each with associated implicationals[1] $\stackrel{\alpha}{\rightarrow}$, $\stackrel{\alpha}{\leftarrow}$), whose behaviour reflects the axioms (e.g. associativity) governing the corresponding operator in the underlying interpretive semantics. Further axioms allow *interaction* between modes (e.g. $x \circ_i (y \circ_j z) = (x \circ_i y) \circ_j z$), and 'linkage' (e.g. $x \circ_i y \longrightarrow x \circ_j y$), i.e. movement from one mode to another. Axioms divide into three classes: (i) *mode internal axioms*, which involve only a single modality, e.g. the familiar associativity axiom $x \circ_i (y \circ_i z) = (x \circ_i y) \circ_i z$; (ii) *interaction axioms*, involving more than one modality, e.g. $x \circ_i (y \circ_j z) = (x \circ_i y) \circ_j z$; (iii) *linkage*

[1] These 'associated implications' correspond to the connectives that are typically notated as $\backslash$ and $/$ in familiar categorial systems such as the associative Lambek calculus, which is a *unimodal* system having a single associative product operator (typically notated as $\bullet$).

or *inclusion axioms*, allowing movement from one mode to another, e.g. $x \circ_i y \longrightarrow x \circ_j y$.[2] Intuitively, the move from one mode to another allowed by a linkage axiom is akin to movement from one description of a linguistic object to an alternative, less informative, description.

We adopt a *labelled* natural deduction formulation, employing inference rules (1–3) below.[3] Labelled formulae take the form: $m \vdash A : s$, with A a type, s a 'semantic' lambda term, and m a *marker* term, the latter being a structured object built up as deduction proceeds, that records information used in ensuring appropriate structure sensitivity in deduction. Hence this system is an instance of a *labelled deductive system* (Gabbay [2]). In linguistic derivations, lexical assumptions have lexically provided marker and semantic components (loosely, the word's 'string' or 'phonology' and its meaning). In all other assumptions (i.e. any additional assumptions, used in hypothetical reasoning, that are eventually discharged), these terms are a simple variable. The role of marker terms here, in recording the proof's significant structural information, closely parallels that of structured configurations of types in various sequent and natural deduction logical formulations, but differing perhaps in that they provide a somewhat more concise/readable representation of the proof's significant structural information. In a linguistic context, a marker may be viewed as providing a description of the linguistic structure of the object derived.

(1)
$$\frac{s \vdash \mathrm{A}\xleftarrow{\alpha}\mathrm{B} : a \qquad t \vdash \mathrm{B} : b}{(s \circ_\alpha t) \vdash \mathrm{A} : (a\ b)}\xleftarrow{\alpha}\mathrm{E} \qquad \frac{\begin{array}{c}[v \vdash \mathrm{B} : v] \\ (s \circ_\alpha v) \vdash \mathrm{A} : a\end{array}}{s \vdash \mathrm{A}\xleftarrow{\alpha}\mathrm{B} : \lambda v.a}\xleftarrow{\alpha}\mathrm{I}$$

(2)
$$\frac{t \vdash \mathrm{B} : b \qquad s \vdash \mathrm{B}\xrightarrow{\alpha}\mathrm{A} : a}{(t \circ_\alpha s) \vdash \mathrm{A} : (a\ b)}\xrightarrow{\alpha}\mathrm{E} \qquad \frac{\begin{array}{c}[v \vdash \mathrm{B} : v] \\ (v \circ_\alpha s) \vdash \mathrm{A} : a\end{array}}{s \vdash \mathrm{B}\xrightarrow{\alpha}\mathrm{A} : \lambda v.a}\xrightarrow{\alpha}\mathrm{I}$$

(3)
$$\frac{\begin{array}{c}[v \vdash \mathrm{B} : v],[w \vdash \mathrm{C} : w] \qquad t \vdash \mathrm{B}\circ_\alpha\mathrm{C} : b \\ s[(v \circ_\alpha w)] \vdash \mathrm{A} : a\end{array}}{s[t] \vdash \mathrm{A} : [b/(v \circ w)].a}\circ_\alpha\mathrm{E} \qquad \frac{s \vdash \mathrm{A} : a \qquad t \vdash \mathrm{B} : b}{(s \circ_\alpha t) \vdash \mathrm{A}\circ_\alpha\mathrm{B} : \langle a, b\rangle}\circ_\alpha\mathrm{I}$$

Additional *structural rules*, which directly reflect axioms of the underlying semantics, act to modify the form of the marker and thereby affect the derivability relation. For example, the associativity rule [a] in (4), which mirrors the associativity axiom $(x \circ_i (y \circ_i z)) = ((x \circ_i y) \circ_i z)$, is needed to enable derivation of the 'simple composition' theorem $\mathrm{X}\xleftarrow{i}\mathrm{Y}, \mathrm{Y}\xleftarrow{i}\mathrm{Z} \Rightarrow \mathrm{X}\xleftarrow{i}\mathrm{Z}$, as illustrated in (7). Note that a system with only a single modality plus the rules (1–4) constitutes a formulation of the associative Lambek calculus. Further examples of structural rules (permutation and linkage) are shown in (5,6). Proof (8) illustrates how the linkage rule allows modality change within an implicational type.

[2]Hepple [4, 5] and Moortgat & Oehrle [9] introduce multimodal frameworks which, like the one to be described in this paper, allow juxtaposition of different levels of the substructural hierarchy of logics, with movement between levels allowed by linkage axioms. Interestingly, the two groups take precisely opposing views as to what constitutes an appropriate pattern of linkage between levels. Kurtonina [7] shows that both views are have well-founded interpretive semantics. There are other proposals that are also multimodal, in the sense of including multiple groups of operators within a single system, with patterns of derivability between different operators, e.g. Morrill [11], Morrill & Solias [12].

[3]A formula in square brackets here indicates an assumption that is discharged by a rule's use. For example, the $[\xleftarrow{\alpha}\mathrm{I}]$ rule indicates that given a proof of a formula of type A which rests on an assumption of type B, we can discharge that assumption to construct a proof of a formula with type $\mathrm{A}\xleftarrow{\alpha}\mathrm{B}$. Note that in (3), $s[(v \circ_\alpha w)]$ and $s[t]$ refer to marker terms that are identical except that where $(v \circ_\alpha w)$ appears as a subterm in the former, t appears instead in the latter.

(4) $$\dfrac{s[(x \circ_i (y \circ_i z))] \vdash \mathrm{A} : a}{s[((x \circ_i y) \circ_i z)] \vdash \mathrm{A} : a}\,[\mathrm{a}]$$

(5) $$\dfrac{s[(x \circ_i y)] \vdash \mathrm{A} : a}{s[(x \circ_j y)] \vdash \mathrm{A} : a}\,[i/j]$$

(6) $$\dfrac{s[(x \circ_i y)] \vdash \mathrm{A} : a}{s[(y \circ_i x)] \vdash \mathrm{A} : a}\,[\mathrm{p}]$$

(7) $$\dfrac{\dfrac{\dfrac{x \vdash \mathrm{X}\leftarrow^{i}\mathrm{Y} : x \qquad \dfrac{y \vdash \mathrm{Y}\leftarrow^{j}\mathrm{Z} : y \qquad [z \vdash \mathrm{Z} : z]}{(y \circ_i z) \vdash \mathrm{Y} : (yz)}}{(x \circ_i (y \circ_i z)) \vdash \mathrm{X} : (x(yz))}}{((x \circ_i y) \circ_i z) \vdash \mathrm{X} : (x(yz))}\,[\mathrm{a}]}{(x \circ_i y) \vdash \mathrm{X}\leftarrow^{i}\mathrm{Z} : \lambda z.(x(yz))}$$

(8) $$\dfrac{\dfrac{\dfrac{x \vdash \mathrm{X}\leftarrow^{i}\mathrm{Y} : x \qquad [y \vdash \mathrm{Y} : y]}{(x \circ_i y) \vdash \mathrm{X} : (xy)}}{(x \circ_j y) \vdash \mathrm{X} : (xy)}\,[i/j]}{x \vdash \mathrm{X}\leftarrow^{j}\mathrm{Y} : \lambda y.(xy)}$$

Regarding the linear order (i.e. word order) consequences of proofs, note that we cannot simply look to the order of assumptions as they are written on the page, since not all modalities carry simple ordering import (and hence likewise their associated connectives). However, we can 'read off' order information from a proof's marker term, *provided* it is constructed only using modalities that do have simple linear import (i.e. are not subject to any permutative axioms). Only proofs that have such markers can serve adequately as linguistic derivations.[4]

3 Categorial Analysis of Movement

The basic treatment of movement rests on being able to derive a type of the form $\mathrm{Y}\leftarrow^{a}\mathrm{Z}$ (informally a phrase Y missing a subphrase Z) for the material forming the extraction domain.[5] Movement is allowed by assigning the displaced element an additional 'movement' type such as $\mathrm{X}\leftarrow^{i}(\mathrm{Y}\leftarrow^{j}\mathrm{Z})$ which can (for leftward movement) prefix to the extraction domain. For example, a relative pronoun might have a type $\mathrm{Rel}\leftarrow^{i}(\mathrm{s}\leftarrow^{j}\mathrm{np})$, so it can combine with a 'sentence missing np' to give a relative clause.

We shall illustrate this approach in relation to a multimodal system, having three modalities: n (non-associative, non-permutative), a (associative, non-permutative), and c (associative, permutative), for which we assume the structural rules [a] and [p] above to be appropriately conditioned. Further, we assume that the schematic linkage rule $[i/j]$ has permissible instances $[n/a]$, $[n/c]$ and $[a/c]$. Consider a relative pronoun type $\mathrm{Rel}\leftarrow^{n}(\mathrm{s}\leftarrow^{c}\mathrm{np})$, and the derivation (9) it allows of the relative clause *who saw kim*.

[4]There are some further aspects to a more complete presentation of the particular multimodal approach described here, which are described in detail in Hepple [6]. In particular, it is shown that explicit marker terms within proofs can be eliminated provided that proof term representations (i.e. the lambda terms encoding functional structure) are augmented with modality information, since marker terms can be directly computed from such enriched proof terms, and hence including explicit marker terms within proofs would then be redundant. The correctness tests on inference rule uses, performed above upon marker terms, can instead then be based upon the enriched proof terms. This development of the system allows for an approach were lexical items may be associated with string components that may be complex terms built using the operators of the proof term algebra (rather than just simple atoms), a move which amounts to allowing lexical encoding of partial proof structure.

[5]This general approach to extraction, depending on the 'flexible deduction' characteristic of many categorial systems, has been widely used within categorial work, and adapts ultimately from the proposals of Ades & Steedman [1].

(9)
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathit{who} \vdash \mathrm{Rel}{\leftarrow^{n}}(\mathrm{s}{\leftarrow^{c}}\mathrm{np}) : \mathbf{who'}} \quad \dfrac{}{\mathit{v} \vdash \mathrm{np} : \mathit{v}} \quad \dfrac{}{\mathit{saw} \vdash (\mathrm{np}{\rightarrow^{n}}\mathrm{s}){\leftarrow^{n}}\mathrm{np} : \mathbf{saw'}} \quad \dfrac{}{\mathit{kim} \vdash \mathrm{np} : \mathbf{kim'}}}{(\mathit{saw} \circ_{n} \mathit{kim}) \vdash \mathrm{np}{\rightarrow^{n}}\mathrm{s} : (\mathbf{saw'}\ \mathbf{kim'})}}{(\mathit{v} \circ_{n} (\mathit{saw} \circ_{n} \mathit{kim})) \vdash \mathrm{s} : (\mathbf{saw'}\ \mathbf{kim'}\ \mathit{v})}}{(\mathit{v} \circ_{c} (\mathit{saw} \circ_{n} \mathit{kim})) \vdash \mathrm{s} : (\mathbf{saw'}\ \mathbf{kim'}\ \mathit{v})}\,[n/c]}{((\mathit{saw} \circ_{n} \mathit{kim}) \circ_{c} \mathit{v}) \vdash \mathrm{s} : (\mathbf{saw'}\ \mathbf{kim'}\ \mathit{v})}\,[\mathrm{p}]}{(\mathit{saw} \circ_{n} \mathit{kim}) \vdash \mathrm{s}{\leftarrow^{c}}\mathrm{np} : \lambda \mathit{v}.(\mathbf{saw'}\ \mathbf{kim'}\ \mathit{v})}}{(\mathit{who} \circ_{n} (\mathit{saw} \circ_{a} \mathit{kim})) \vdash \mathrm{Rel} : \mathbf{who'}(\lambda \mathit{v}.(\mathbf{saw'}\ \mathbf{kim'}\ \mathit{v}))}$$

Such extraction derivations involve an additional assumption (here of type np) appearing in the canonical place of the displaced phrase, which is subsequently discharged in an [$\leftarrow^{a}$I] step, creating an implicational type $\mathrm{Y}{\leftarrow^{a}}\mathrm{Z}$ — a 'Y missing Z'. The structure sensitivity of extraction depends crucially on this [$\leftarrow^{a}$I] step, requiring the immediate subproof's marker to restructure to the form $(s \circ_{\alpha} v)$ (v being the discharged assumption's marker variable). Whether this restructuring is possible or not depends on the structural characteristics of the proof (or, if you prefer, the structure of the phrase), and on the specific modality α required, which is itself determined by the 'movement category' of the displaced phrase, e.g. modality c for the category $\mathrm{Rel}{\leftarrow^{n}}(\mathrm{s}{\leftarrow^{c}}\mathrm{np})$ in the above proof. This modality allows linkage inferences that change the marker to a weaker description that allows use of the permutation rule, and hence this relative pronoun type can allow subject extraction, as it does in (9), but can equally well allow object extraction, as in the derivation (10) of *who kim saw.*

(10)
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathit{who} \vdash \mathrm{Rel}{\leftarrow^{n}}(\mathrm{s}{\leftarrow^{c}}\mathrm{np}) : \mathbf{who'}} \quad \dfrac{}{\mathit{kim} \vdash \mathrm{np} : \mathbf{kim'}} \quad \dfrac{}{\mathit{saw} \vdash (\mathrm{np}{\rightarrow^{n}}\mathrm{s}){\leftarrow^{n}}\mathrm{np} : \mathbf{saw'}} \quad \dfrac{}{\mathit{v} \vdash \mathrm{np} : \mathit{v}}}{(\mathit{saw} \circ_{n} \mathit{v}) \vdash \mathrm{np}{\rightarrow^{n}}\mathrm{s} : (\mathbf{saw'}\ \mathit{v})}}{(\mathit{kim} \circ_{n} (\mathit{saw} \circ_{n} \mathit{v})) \vdash \mathrm{s} : (\mathbf{saw'}\ \mathit{v}\ \mathbf{kim'})}}{(\mathit{kim} \circ_{a} (\mathit{saw} \circ_{a} \mathit{v})) \vdash \mathrm{s} : (\mathbf{saw'}\ \mathit{v}\ \mathbf{kim'})}\,[n/a]^{*}}{((\mathit{kim} \circ_{a} \mathit{saw}) \circ_{a} \mathit{v}) \vdash \mathrm{s} : (\mathbf{saw'}\ \mathit{v}\ \mathbf{kim'})}\,[\mathrm{a}]}{((\mathit{kim} \circ_{a} \mathit{saw}) \circ_{c} \mathit{v}) \vdash \mathrm{s} : (\mathbf{saw'}\ \mathit{v}\ \mathbf{kim'})}\,[a/c]}{(\mathit{kim} \circ_{a} \mathit{saw}) \vdash \mathrm{s}{\leftarrow^{c}}\mathrm{np} : \lambda \mathit{v}.(\mathbf{saw'}\ \mathit{v}\ \mathbf{kim'})}}{(\mathit{who} \circ_{n} (\mathit{kim} \circ_{a} \mathit{saw})) \vdash \mathrm{Rel} : \mathbf{who'}(\lambda \mathit{v}.(\mathbf{saw'}\ \mathit{v}\ \mathbf{kim'}))}$$

Using instead a movement category $\mathrm{Rel}{\leftarrow^{n}}(\mathrm{s}{\leftarrow^{a}}\mathrm{np})$, requiring modality a for the [$\leftarrow$I] step, we find that object extraction is still possible, as in (11), but that subject extraction is not since we cannot now move to a marker allowing permutation. The third alternative of using a movement category requiring modality n for the introduction step would allow neither subject nor object extraction (i.e. since neither [p] nor [a] could be used). Note then that the two connectives in a movement type such as $\mathrm{Rel}{\leftarrow^{i}}(\mathrm{s}{\leftarrow^{j}}\mathrm{np})$ play very different roles. The principle connective ($\leftarrow^{i}$) serves to *assign structure* in the usual way, whereas the second embedded connnective ($\leftarrow^{j}$) instead serves to *test structure.*

(11)

$$\dfrac{who \vdash \mathrm{Rel}\overset{n}{\leftarrow}(\mathrm{s}\overset{a}{\leftarrow}\mathrm{np}) : \mathbf{who'} \qquad \dfrac{\dfrac{\dfrac{\dfrac{kim \vdash \mathrm{np} : \mathbf{kim'} \qquad \dfrac{saw \vdash (\mathrm{np}\overset{n}{\rightarrow}\mathrm{s})\overset{n}{\leftarrow}\mathrm{np} : \mathbf{saw'} \qquad v \vdash \mathrm{np} : v}{(saw \circ_n v) \vdash \mathrm{np}\overset{n}{\rightarrow}\mathrm{s} : (\mathbf{saw'}\ v)}}{(kim \circ_n (saw \circ_n v)) \vdash \mathrm{s} : (\mathbf{saw'}\ v\ \mathbf{kim'})}}{(kim \circ_a (saw \circ_a v)) \vdash \mathrm{s} : (\mathbf{saw'}\ v\ \mathbf{kim'})}\,[n/a]^*}{((kim \circ_a saw) \circ_a v) \vdash \mathrm{s} : (\mathbf{saw'}\ v\ \mathbf{kim'})}\,[\mathrm{a}]}{(kim \circ_a saw) \vdash \mathrm{s}\overset{a}{\leftarrow}\mathrm{np} : \lambda v.(\mathbf{saw'}\ v\ \mathbf{kim'})}}{(who \circ_n (kim \circ_a saw)) \vdash \mathrm{Rel} : \mathbf{who'}(\lambda v.(\mathbf{saw'}\ v\ \mathbf{kim'}))}$$

Let us contrast this approach to handling movement constraints to what we perhaps might view as more 'standard' alternatives, which address a single description of linguistic structure and then apply some relatively complex constraint in deciding the permissibility of a given extraction in terms of the relation between the extracted phrase and its canonical position. For a tree-based approach to linguistic description, such a constraint could perhaps be formulated (or reformulated) as some non-trivial tree-traversing automaton. For the present approach, the absolute characterisation of a movement constraint is quite simple, i.e. does a given phrase have a description $(s \circ_\alpha v)$, for some α? The real complexity of the constraint can be seen to lie with the initial rich lexical encoding of linguistic structure and the system of alternative descriptions that the given multimodal approach allows, from which the individual description is selected.

In what follows, our discussion of how to treat locality constraints within this general approach will not be illustrated by proofs for specific cases, but will instead focus directly on marker systems, and whether or not they allow restructuring of the above kind as is appropriate for characterising particular locality behaviour.

4 Dependency and Locality

An earlier account of locality constraints within a type-logical framework is offered by Hepple [3] (extending proposals due to Morrill [10]), which makes available multiple *unary* modalities, that can be marked on particular phrases to make them boundaries for movement. One criticism that can be levelled against this approach is that it is essentially stipulative, in that it allows boundaries to be specified in a way that is independent of other aspects of structure. In contrast, the aim here is to construct an account of locality that is rooted in a more broadly motivated account of linguistic structure, i.e. developing what appears to be an adequate description of linguistic structure for broader purposes, and then using it as the basis for formulating an account of locality constraints.[6]

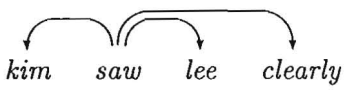
One syntactic distinction that might be encoded by modalities is dependency, i.e. the asymmetry between heads and dependents. This idea was introduced, within categorial work, by Moortgat & Morrill [8], who employ it in a type-logical encoding of metrical trees. A central claim of this paper is that, within a multimodal setting, structures encoding dependency provide an appropriate basis for addressing locality in movement. In particular, such structures allow for the use of a notion of 'domain' for locality which might be termed the 'domain of a head' — i.e. that domain consisting of a head plus its dependents — and allow us to distinguish the cases where the movement of some element does or does not stay within some given domain.

[6]The account of locality constraints to be presented here develops earlier proposals outlined in Hepple [4].

Let us notate left-headed modes using $\succ$, and right-headed ones using $\prec$ (a notation intended to be reminiscient of the 'arrow structures' of dependency grammar, with heads 'pointing' at their dependents), i.e. so that x is head in $(x \succ y)$. A derivation of (e.g.) *Kim saw Lee clearly* might yield a marker such as:

$$(kim \prec ((saw \succ lee) \succ clearly)).$$

The use of binary operators here gives hierarchical structures in which some 'head' elements may be complex. This contrasts with the 'flatter' structures of dependency grammar, where all heads are lexical elements, as in e.g.:



To bridge this gap, it is useful to go beyond our purely binary structures to a recursive notion of R-*head* where a single atomic element has multiple dependents, e.g. in $((y \prec x) \succ z)$, atomic x is R-head, having R-dependents y, z (which are the 'immediate dependents' of the 'projections' of x). Additionally, let the 'maximal head' of an expression be the R-head within it that dominates all other R-heads (i.e. they are contained within its R-dependents).

Let us consider a multimodal approach whose modalities include ones that encode the head-dependent asymmetry, as indicated above. The modalities used in specifying lexical types will be ones that are structurally restrictive (and hence more 'informative'), encoding (we might expect) linear order and bracketting (i.e. being non-associative and non-commutative), as well as headedness (i.e. head-dependent asymmetry). This level can be linked to other structurally more-liberal levels, whose behaviour allows for different possibilities of dependency-sensitive movement. Let us imagine one such level (notated $\dot{\prec}$, $\dot{\succ}$), which maintains a head-dependent distinction, and consider how this level may be used in characterising constraints on movement. Note that we are only concerned with locality constraints, and not any putative order-related constraints, and so we shall firstly assume that the following axiom applies at this level, which freely reorders heads and dependents, whilst maintaining the distinction between them, and which hence serves to undermine any effects of order upon what is derivable:

$$\text{[ax1]:} \quad x \dot{\succ} y \;=\; y \dot{\prec} x$$

The crucial division between where movement is restricted to be within local domains (in the 'domain of a head' sense indicated above) depends on our choice of axioms from the following two:

$$\text{[ax2]:} \quad x \dot{\prec} (y \dot{\succ} z) \;=\; (x \dot{\prec} y) \dot{\succ} z$$

$$\text{[ax3]:} \quad x \dot{\succ} (y \dot{\succ} z) \;=\; (x \dot{\succ} y) \dot{\succ} z$$

Although the axiom [ax2] does not preserve hierarchical head-dependent structure, it does preserve R-heads and R-dependents (i.e. the markers it equates will have the same R-heads, each of which will have the same set of R-dependents). In combination with [ax1], this axiom will permit restructuring that allows any dependent of the maximal head of an expression to 'move up' to topmost hierarchical position, so that a marker α may restructure to the form $\beta \dot{\succ} x$ (for some β) *iff* x is a dependent of the maximal head of α. Consequently, the corresponding implication ($\dot{\Leftarrow}$) could be used for a version of bounded movement of dependents, i.e. allowing an element to move to the periphery of the domain of its R-head but

not beyond. However, this system appears too restrictive for most purposes. For example, it is well known that adverbial adjuncts cannot in general be extracted from embedded clauses (as illustrated in (12)). However, purely head-bounded movement is too restrictive for this phenomenon, as shown by (13b).

(12) a. John [$_{vp}$ [$_{vp}$ opened the box] [$_{adv}$ with a crowbar]]

b. How$_i$ did John [$_{vp}$ [$_{vp}$ open the box] $\__i$]

c. *How$_i$ do you remember that John [$_{vp}$ [$_{vp}$ opened the box] $\__i$]
(* under intended reading)

(13) a. John wants [$_{vp}$ [$_{vp}$ to leave] tomorrow]

b. When$_i$ does John want [$_{vp}$ [$_{vp}$ to leave] $\__i$]

In contrast to [ax2], the axiom [ax3] preserves neither heads nor dependents in general (nor, indeed, either R-heads or R-dependents). The move from $x \succ (y \succ z)$ to $(x \succ y) \succ z$ that it allows might be viewed as a 'non-local' restructuring whereby an 'embedded' dependent moves up a level. In conjunction with [ax1] and [ax2], this axiom will allow an embedded dependent to move up to topmost hierarchical position, and hence be extracted. More specifically, this system will allow a marker α to restructure to the form $\beta \succ x$ (for some β) for *any* (atomic) x within α *except* its maximal head. Hence, such a system appears to be *too* liberal to be useful. A complementary observation, however, is that in a system with [ax1] and [ax3] (either with or without [ax2]), a marker α can restructure to the form $\beta \prec x$ (for some β), with x atomic, *iff* x is the maximal head of α. Consequently, the corresponding implication ($\overset{\prec\cdot}{\leftarrow}$) could be used in implementing a bounded form of head movement, allowing a head to move to the edge of its 'domain' (i.e. consisting of itself plus dependents) but not beyond. A possible use is in handling the bounded movement of the finite verb in the main clauses of V2 (Verb-Second) languages such as Dutch and German. The requisite movement types for finite verbs might be generated by a lexical rule such as:

$$\mathrm{V} \Rightarrow \mathrm{s}_m/(\mathrm{s} \overset{\prec\cdot}{\leftarrow} \mathrm{V}) \qquad (\mathrm{V\ a\ finite\ verb})$$

As we have seen, the restructuring allowed by [ax2] alone is insufficient, but free involvement of [ax3] gives a system that is too liberal. However, an intermediate position between these two extremes is possible, which involves restricting the action of the 'non-local' axiom [ax3], and in particular linking its use to further distinctions encoded by modalities. Let us consider just one of many possible distinctions that might be invoked. Various linguistic approaches acknowledge a distinction between head-complement and head-adjunct relations. We might use different operators to encode these different dependencies, e.g. such as $\succ_c$, $\succ_a$ (c for complement, a for adjunct), so that our example *Kim saw Lee clearly* might yield a marker such as:

$$(kim \prec_c ((saw \succ_c lee) \succ_a clearly)).$$

These modes might be linked to others ($\dot{\succ}_c$, $\dot{\succ}_a$) which preserve both headedness and the complement/adjunct distinction, but which are otherwise more liberal in being subject to variants of the axioms [ax1], [ax2] and [ax3]. A modified [ax3], in particular, might be restricted to apply in only certain cases, e.g. taking the form:

$$x \succ_i (y \succ_j z) \;=\; (x \succ_i y) \succ_j z)) \qquad \text{where } \langle i, j \rangle \in \{\ldots\}$$

For cases of $\langle i, j \rangle$ pairs that are *not* allowed, the effect is that 'j-dependents' may not move up out of 'i-domains', so that 'i-domains' are islands to extraction of 'j-dependents'. For example, the island status of adjuncts, illustrated by (14), could be enforced by disallowing all pairs $\langle i, j \rangle$ in which i corresponds to a head-adjunct relation.

(14) a. Kim filed the articles without telling Lee.

b. *Who did Kim file the articles without telling?

The above example hopefully illustrates the point that this approach seeks to ground an analysis of locality constraints within a detailed representation of linguistic structure, exploiting the distinctions that this representation encodes rather than being merely a stipulative overlay.

References

[1] Ades, A.E. and Steedman, M.J. 1982. 'On the order of words.' *Linguistics and Philosophy*, **4**. 517–558.

[2] Gabbay, D.M. 1996. *Labelled deductive systems. Volume I.* Oxford University Press.

[3] Hepple, M. 1990. *The Grammar and Processing of Order and Dependency: A categorial approach.* PhD Thesis, University of Edinburgh.

[4] Hepple, M. 1993. 'A general framework for hybrid substructural categorial logics.' Ms, IRCS, UPenn. Available as IRCS Report 94-14.

[5] Hepple, M. 1995. 'Mixing Modes of Linguistic Description in Categorial Grammar.' *Proceedings EACL-7.*

[6] Hepple, M. 1996. 'Head Promotion and Obliqueness in a Multimodal Grammar', *Proofs and Linguistic Categories*, C. Casadio (ed), Proceedings of the Third Roma Workshop, Rome, April 1996.

[7] Kurtonina, N. 1994. *Frames and Labels: A Modal Analysis of Categorial Inference.* PhD thesis, Utrecht University.

[8] Moortgat, M. & Morrill, G. 1991. 'Heads and Phrases: Type Calculus for Dependency and Constituency.' To appear: *Journal of Language, Logic and Information.*

[9] Moortgat, M. & Oehrle, R. 1994. 'Adjacency, dependency and order'. *Proceedings of Ninth Amsterdam Colloquium.*

[10] Morrill, G. 1990. 'Intensionality and Boundedness,' *Linguistics and Philosophy*, **13**, 699–726.

[11] Morrill, G. 1995. 'Clausal Proofs and Discontinuity', *Bulletin of the Interest Group in Pure and Applied Logics*, Vol. 3, No. 2,3.

[12] Morrill, G. & Solias, M.T. 1993. 'Tuples, Discontinuity, and Gapping in Categorial Grammar.' *Proc. of EACL-6*, Utrecht.

Agreement Modalities

Dirk Heylen

UiL – OTS, Utrecht University

Abstract Recently, several extensions to the Lambek calculus have been proposed that incorporate important aspects of the expressivity offered by feature structures as used in constraint-based grammars. In this paper we focus on a specific extension using unary modal operators to express feature information. We compare this to a constraint-based approach, like categorial unification grammar, and an extension of the Lambek calculus that uses layering of a categorial logic over a feature logic. We illustrate the approach with a simple example of agreement relations between adjectives and nouns in Dutch.

In the first section we highlight a number of aspects of feature structures, their logic and their use as descriptive devices in theories of grammar. In the second section we present the formal details of a multimodal type-logical grammar and extensions to categorial grammars that include means to express some aspects of feature structures. In the last section we illustrate their use.

1 Feature Structures

1.1 Definition

One could define a feature structure as a decorated labelled directed connected acyclic rooted graph with no two edges with the same label originating at the same node. Many variations to this definition are possible. For instance, in some frameworks cyclic graphs are allowed. Some frameworks require decorations on all nodes, others only on terminal nodes (nodes from which no edges emanate), many require that each node can only be decorated by a single decoration, etc.

To provide a more formal definition (inspired by definitions in [1]) we first parameterise the definition with respect to a signature $\langle \mathcal{L}, \mathcal{A} \rangle$, a pair of non-empty sets thought of as the set of possible labels on the edges, the *features*, and the set of atomic information that can decorate nodes, *types* or *sorts*. A feature structure of signature $\langle \mathcal{L}, \mathcal{A} \rangle$ then is an ordered triple $\langle N, \{R_l\}_{l \in \mathcal{L}}, \{Q_\alpha\}_{\alpha \in \mathcal{A}} \rangle$, where N is a non-empty set of nodes; for each $l \in \mathcal{L}$, R_l is a partial function on N and for all $\alpha \in \mathcal{A}, Q_\alpha$ is a unary relation on N.

The feature structures used in HPSG to model linguistic expressions are of the variety called *typed feature structures*. This variety is well-documented in [4]. The signature for such structures is slightly more complicated. The set of sorts is assumed to be partially ordered, $\sqsubseteq$, and appropriateness conditions are defined that restrict the domain and range of the functions R_l in terms of the decorations on the nodes.

In order to talk about feature structures themselves, we need a language. Many feature structure description languages have been proposed in the literature. Here, we choose the multi-modal propositional language L^N as described in [1]. The signature for this language is $\langle \mathcal{L}, \mathcal{A}, \mathcal{B} \rangle$. The language contains an $\mathcal{L}$ indexed collection of distinct modalities, a set $\mathcal{A}$ of sort symbols, and a set $\mathcal{B}$ of nominals. The set $\mathcal{F}$ of well-formed formulas is defined as follows.

$\mathcal{F} ::= \mathcal{A} \mid \mathcal{B} \mid \Diamond_l \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid \neg \mathcal{F}$

Other Boolean connectives can be defined as usual as can the dual box modalities. We will write $\langle l \rangle$ for $\Diamond_l$.

The feature structures as defined above can be viewed as Kripke models for this language. To interpret nominals we add to the definition of feature structure unary relations Q_β, $\beta \in \mathcal{B}$ which must be singleton subsets of N: a nominal is true at exactly one node in the structure. Using nominals we can talk about reentrancies (structure sharing). Given a feature structure $M = \langle N, n, \{R_l\}_{l \in \mathcal{L}}, \{Q_\gamma\}_{\gamma \in \mathcal{A} \cup \mathcal{B}} \rangle$ we can specify the truth definition as follows:

$M, n \models \alpha$ iff $n \in Q_\alpha$	$M, n \models \phi \wedge \psi$ iff $M, n \models \phi$ and $M, n \models \psi$
$M, n \models \beta$ iff $n \in Q_\beta$	$M, n \models \phi \vee \psi$ iff $M, n \models \phi$ or $M, n \models \psi$
$M, n \models \neg\phi$ iff $M, n \not\models \phi$	$M, n \models \langle l \rangle \phi$ iff $\exists n' (R_l(n) = n' \wedge M, n' \models \phi)$

We now turn to an illustration of the use of the language.

1.2 Constraint-based Grammars

In a constraint-based grammar like HPSG, languages are modelled as sets of totally well-typed, and sort resolved feature structures[1] that satisfy the grammar, G; a formula from the feature description language. A feature structure $M = \langle N, n, \{R_l\}_{l \in \mathcal{L}}, \{Q_\gamma\}_{\gamma \in \mathcal{A} \cup \mathcal{B}} \rangle$ is in the language if it satisfies G. We make this idea a bit more precise by defining a categorial unification grammar ([3]) in this way. We assume a set of features and sorts as follows:

$\mathcal{L} = \{$CAT, VAL, ARG, FUNCTOR, ARGUMENT$\}$

$\mathcal{A} = \{$ *bot, sign, category, word, phrase, basic, complex, s, np, n* $\}$

$\mathcal{B} = \{i, j, ...\}$

We furthermore assume that all types are subtypes of *bot*; *word* and *phrase* are subtypes of *sign*; *basic* and *complex* are subtypes of *category* and *s, np, n* are subtypes of *basic*, i.e $bot \sqsubseteq sign, bot \sqsubseteq$, etc. The functions interpreting the features are restricted as follows.

CAT: $sign \rightarrow category$ VAL: $complex \rightarrow category$ ARG: $complex \rightarrow category$

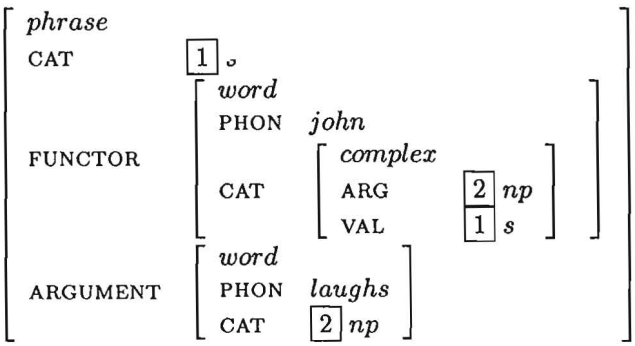
FUNCTOR: $phrase \rightarrow sign$ ARGUMENT: $phrase \rightarrow sign$

The essence of the categorial grammar formula is given as the following formula which in fact we want to hold for all nodes of sort *sign*[2].

$(word \vee (phrase \wedge \langle cat \rangle i \wedge \langle functor \rangle (sign \wedge \langle cat \rangle (\langle val \rangle i \wedge \langle arg \rangle j)) \wedge \langle argument \rangle (sign \wedge \langle cat \rangle j)))$

This requires that either feature structures in our language are of sort *word* or *phrase* and it requires the latter to have CAT, FUNCTOR, and ARGUMENT features where the FUNCTOR has a complex category such that the value for FUNCTOR|ARG is reentrant with the value of ARGUMENT|CAT and the value for FUNCTOR|VAL is reentrant with the value of CAT. This is the essence of the application schema.

Of course, a more realistic grammar will also contain other features to cover more dimensions of linguistic description (morphosyntax, phonology, semantics etc.). We also need to expand the grammar by providing more specific descriptions of words (the lexicon). Adding a feature like PHON : $word \rightarrow phonology$, and $john, laughs$, as subtypes of *phonology* and replacing *word* in the formula above by $(word \wedge \langle$PHON$\rangle john \wedge \langle$CAT$\rangle np) \vee (word \wedge \langle$PHON$\rangle laughs \wedge \langle$CAT$\rangle\langle$ARG$\rangle np \wedge \langle$CAT$\rangle\langle$VAL$\rangle s)$ allows the following feature structure in our language.



1.3 Properties

The basic components of feature structures are *features* and *sorts*. The former are used to name a property or parameter of classification, the latter as a value for the parameter. In some linguistic frameworks feature structures are simply defined as sets of feature-value pairs. Such structures are assumed to describe objects for which all the properties, expressed by the feature-value pairs, hold

[1]This means that each node must be decorated by a sort. No node of a certain sort can have an outgoing arc that is not appropriate for that sort. If a feature is appropriate for a sort then there must be an edge labelled with this feature for each node decorated by that sort. Also, each node must be decorated by a sort that has no subsorts, in other words, that is maximal.

[2]Note that this formula does not express this idea precisely. Actually we need a recursive constraint. See [2].

simultaneously. These are about the simplest definitions of feature structures available, which shows that this conjunctive aspect is central to the notion of feature structure. The basic use of such simple structures is to allow cross-classification. Also, the use of such sets as categories defines an information ordering corresponding to the subset ordering. It provides a simple way to express generalisations and the combination or unification of information by taking the union of two sets (but taking care that the functional nature of features is retained).

The feature structures we have considered them above also allow structures as complex values of features. The phrasal signs have FUNCTOR and ARGUMENT features whose values are again *signs*. This recursion is needed if we want our feature structures to describe linguistic trees as in HPSG.

Reentrancy (structure sharing) or the nominals of the description language, are essentially used to enforce equality of certain values. Feature percolation principles (think of the head feature principle in HPSG) that enforce equality of feature values in different parts of the linguistic tree are formulated as reentrancies.

The extensions to the type-logical grammars we discuss in the next section do not encode all these properties of feature structures.

2 Categories and Feature Structures

2.1 Categorial/Type-logical Grammars

In this paragraph we define a basic multimodal categorial language and logic. We assume a set $\mathcal{A}$ of basic types and a set $\mathcal{I}$ of indices on the type-forming connectives. Formulas are then defined by the following grammar.

$\mathcal{F} ::= \mathcal{A} \mid \mathcal{F} \bullet_i \mathcal{F} \mid \mathcal{F} \backslash_i \mathcal{F} \mid \mathcal{F} /_i \mathcal{F}$.

Deductions in the next section are written in a Gentzen-style sequent presentation. We will use rules such as the following.

Elimination		*Introduction*	
$\dfrac{\Delta \Rightarrow Y \quad \Gamma[X] \Rightarrow Z}{\Gamma[X/_i Y \circ_i \Delta] \Rightarrow Z}$	$\dfrac{\Delta \Rightarrow Y \quad \Gamma[X] \Rightarrow Z}{\Gamma[\Delta \circ_i Y \backslash_i X] \Rightarrow Z}$	$\dfrac{\Gamma \circ_i Y \Rightarrow X}{\Gamma \Rightarrow X/_i Y}$	$\dfrac{Y \circ_i \Gamma \Rightarrow X}{\Gamma \Rightarrow Y \backslash_i X}$

The semantics for the language is provided in terms of frame semantics, with some domain N (of linguistic resources) and ternary accessibility relations, R_i, interpreting $\bullet_i$. To define the interpretation in a model $\mathbf{M}$ we further assume a family of subsets of the domain, one for each basic category: Q_b.

$$
\begin{array}{lcl}
v(b) & = & \{x \mid x \in Q_b\} \\
v(A \bullet_i B) & = & \{x \mid \exists y \exists z [R_i(x,y,z) \;\&\; y \in v(A) \;\&\; z \in v(B)]\} \\
v(C/_i B) & = & \{y \mid \forall x \forall z [(R_i(x,y,z) \;\&\; z \in v(B)) \Rightarrow x \in v(C)]\} \\
v(A \backslash_i C) & = & \{z \mid \forall x \forall y [(R_i(x,y,z) \;\&\; y \in v(A)) \Rightarrow x \in v(C)]\}
\end{array}
$$

In this framework a grammar is specified by providing a lexicon that assigns categories to lexical linguistic resources. For a general model to qualify as appropriate for the lexicon, we assume that the linguistic resources in the lexicon are in the model and we require the valuation function v to be compatible with the lexical type assignment. Grammaticality is defined in terms of deduction as in [10]: the grammar assigns some type B to a non-empty string of lexical resources $x_1, ..., x_n$, provided there are lexical type specifications, $A_1, ..., A_n$ (such that A_1 is in the lexical assignments of x_1 etc.) and we can deduce B from $\circ(A_1, ..., A_n)$ in the type logic. By $\circ(A_1, ..., A_n)$ is meant, any of the possible products of the formulas $A_1, .., A_n$ in that order.

2.2 Feature Extensions

An important difference between the constraint-based and the type-logical grammars is that the former define grammaticality on the basis of the satisfiability of formulas making up the grammar and the lexicon (find a model that makes the formula true) whereas the latter are defined on the basis of derivability and hence the validity of formulas (the formula must be true in all models). This

has repercussions on the way that feature structures can be used in type-logical grammars, relating to unification and underspecification. If we would just transplant the constraint-language in the type-logical setting then there are formulas that defined grammatical structures in the constraint-language (true in some model) that are no longer grammatical in the type-logical grammar (not true in all models).

There are many options to choose from when adding feature structure-like objects to type-logical grammars. First, there is the issue of the precise notion of feature structure one wants to embed in the categorial framework. For instance, one could use only flat, non-recursive structures (like feature structures as sets of feature-value pairs); one could choose to ignore the features and leave them implicit (as we will do below); and one could choose to exclude structure-sharing or reentrancy. Next, one has to choose a specific feature description-language. Some of the options are: a propositional language (see [9]), a predicational language (see [11]) or a modal language ([12]). Related to the choice of description language is the way of combining the feature and type logic. Some choices of layering and double layering the logics are discussed in [5], [6] and [7]. One particular option of fibering a feature logic with the type logic is presented below. The other extension we discuss adds unary modalities to the type-logic to express feature information.

Propositional One way to refine the category structure of Lambek-style categorial grammars is to replace atomic basic categories by formulas taken from some feature logic ([6]). Such systems preserve the inferential capacities of the categorial logic, while allowing a way to decompose at least basic categories into features. In the system proposed by [9] the feature logic is a simple fragment of propositional logic. Crucially, functors and arguments do not combine through unification but the argument position of the functor must subsume the argument category: $X/Y \bullet Z \rightarrow X$ if Y subsumes Z ($Y \sqsubseteq Z$, i.e. $v(Z) \subseteq v(Y)$).

We now give a more formal definition of this language and its interpretation. The categorial language extended with booleans can be characterised as follows. We assume a set $\mathcal{A}$ of basic categories or sorts. Formulas are then defined by the following grammar.

$\mathcal{T} ::= \mathcal{A} \mid \mathcal{T} \wedge \mathcal{T} \mid \mathcal{T} \vee \mathcal{T}$

$\mathcal{F} ::= \mathcal{T} \mid \mathcal{F} \bullet_i \mathcal{F} \mid \mathcal{F} \backslash_i \mathcal{F} \mid \mathcal{F} /_i \mathcal{F}.$

We assume similar interpretation clauses as for the language presented above. The interpretation for the terms of the language is now slightly more complicated.

$v(A \wedge B) = \{x \mid x \in v(A) \ \& \ x \in vB\} = v(A) \cap v(B)$

$v(A \vee B) = \{x \mid x \in v(A) \vee x \in vB\} = v(A) \cup v(B)$

As for the proof-theoretic characterisation of the logic we can adopt the usual rules for $\wedge$ and $\vee$ from propositional logic. More about this can be found in [6] (where soundness, completeness and decidability are proven).

This approach can be modified in various ways. One could for instance add other boolean constructors, replace the sorts by feature-value pairs, add an ordering to the sorts, or indeed replace the boolean terms by a complete feature logic as the one presented in the previous section. One could also consider moving $\wedge, \vee$ a level up to connect formulas.

As it stands the feature part is very restricted. We have only sorts, and no features. However, we do have the means for multiple classification using $\wedge$ but no recursive structure (embedding). An information ordering is defined by the logic of $\wedge, \vee$ as well, which makes underspecification (to a certain extent as we will see shortly) and generalisations possible. The language does not include the means to talk about equality, so there is no device matching reentrance. Because the feature-language is plugged in at the atomic level (the level of the basic types of the categorial language), we can only decompose basic categories into morphosyntactic information (see [3] for a discussion on such a restriction).

Modal We now turn to another way to add feature decorations to a type-logical grammar which is also quite restricted in the kind of feature structures it actually encodes. In this case we use unary modal operators as described in [10] to express feature-like information. We specify the syntax as follows.

$\mathcal{F} ::= \mathcal{A} \mid \mathcal{F} \bullet_i \mathcal{F} \mid \mathcal{F} \backslash_i \mathcal{F} \mid \mathcal{F} /_i \mathcal{F} \mid \Diamond_i \mathcal{F} \mid \Box_i^{\downarrow} \mathcal{F}$

To account for the semantics, we add the following clauses.

$v(\Diamond_i C) = \{x \mid \exists y[R_i(x,y) \ \& \ y \in v(C)]\}$

$v(\Box_i^{\downarrow} C) = \{x \mid \forall y[R_i(y,x) \Rightarrow y \in v(C)]\}$

In Gentzen format, the rules for the unary operators we will be using are the following.

$$\frac{(\Gamma)^{\Diamond_i} \Rightarrow Z}{\Gamma \Rightarrow \Box_i^{\downarrow} Z} \qquad \frac{\Gamma[X] \Rightarrow Z}{\Gamma[(\Box_i^{\downarrow} X)^{\Diamond_i}] \Rightarrow Z}$$

Our analysis also relies on inclusion postulates, marked $\sqsubseteq$, for the unary modalities which will be used to encode features. The inclusion postulates can be seen as part of the 'signature' familiar from typed-unification logic. They express a kind of subsumption relation between features. The distribution postulates, marked **A**, will be used to enforce agreement.

$$\underset{\textit{inclusion postulate}}{\frac{\Gamma[(X)^{\Diamond_j}] \Rightarrow Z}{\Gamma[(X)^{\Diamond_i}] \Rightarrow Z}} \sqsubseteq \qquad \underset{\textit{distribution postulate}}{\frac{\Gamma[\Delta_1^{\Diamond_k} \bullet_i \Delta_2^{\Diamond_k}] \Rightarrow Z}{\Gamma[(\Delta_1 \bullet_i \Delta_2)^{\Diamond_k}] \Rightarrow Z}} \ \mathbf{A}$$

Our deductions will be abbreviated wherever convenient. We will write $\langle i \rangle$ and $[i]$ instead of $\Diamond_i$ and $\Box_i^{\downarrow}$. For more details on formal aspects of this calculus, the reader should consult [10]. In the next section we provide an elaborate illustration of the way this language is used to encode feature information.

3 Features and Modalities

3.1 The problem

The paradigm of Dutch agreement phenomena that concerns us here is illustrated by the following data.

*de/*het jongen*	'the boy'	*het/*de kind*	'the child'
*de aardige/*aardig jongen*	'the nice boy'	*een aardige/*aardig jongen*	'a nice boy'
*het aardige/*aardig kind*	'the nice child'	*een *aardige/aardig kind*	'a nice child'

Dutch nouns bear grammatical gender. Neuter nouns combine with the definite determiner *het*, non-neuter nouns with *de*. The indefinite determiner *een* can combine with both. As the above examples show, the form of the adjective varies with the context in a particular way. If the determiner is indefinite and the noun is neuter, the adjective is not inflected. In all other cases the adjective is inflected.

The analysis of this construction is interesting when considering feature extensions to the Lambek calculus for the following reason. Bayer and Johnson ([9]), defend the view that a theory modelling agreement phenomena in terms of the requirement that arguments must be *subsumed* by, or logically imply, the corresponding argument specification of a predicate or functor category, is superior to a theory that assumes *unification* (see also [8]). Bouma (in postings to the CG-mailing list, January 1997) challenges this position by arguing that the agreement phenomena in Dutch cannot be treated in a subsumption-based setting without missing generalisations. In this paper we take up Bouma's challenge and provide a subsumption-based analysis of the Dutch constructions in which the generalisations are not lost using the mixed multimodal calculus presented above.

In a constraint-based analysis of this construction, we can use the lexical assignments given below. Note that we assume an extended signature to accomodate for the morphological information. ⟨CM⟩ groups together ⟨CAT⟩ and ⟨MOR⟩ information. ⟨VAL*⟩ abbreviates ⟨CAT⟩⟨VAL⟩⟨CM⟩, ⟨ARG*⟩ abbreviates ⟨CAT⟩⟨ARG⟩⟨CM⟩.

⟨PHON⟩*de* ∧ ⟨CM⟩(⟨VAL*⟩⟨CAT⟩*np* ∧ ⟨ARG*⟩(⟨CAT⟩*n* ∧ ⟨MOR⟩(⟨GEN⟩*de* ∧ ⟨DEF⟩+)))

⟨PHON⟩*het* ∧ ⟨CM⟩(⟨VAL*⟩⟨CAT⟩*np* ∧ ⟨ARG*⟩(⟨CAT⟩*n* ∧ ⟨MOR⟩(⟨GEN⟩*het* ∧ ⟨DEF⟩+)))

⟨PHON⟩*een* ∧ ⟨CM⟩(⟨VAL*⟩⟨CAT⟩*np* ∧ ⟨ARG*⟩(⟨CAT⟩*n* ∧ ⟨MOR⟩⟨DEF⟩−))

⟨PHON⟩*jongen* ∧ ⟨CM⟩(⟨CAT⟩*n* ∧ ⟨MOR⟩⟨*gen*⟩*de*)

⟨PHON⟩*kind* ∧ ⟨CM⟩(⟨CAT⟩*n* ∧ ⟨MOR⟩⟨*gen*⟩*het*)

$\langle \text{PHON} \rangle aardig \wedge \langle \text{CM} \rangle (\langle \text{VAL}^* \rangle (\langle \text{CAT} \rangle n \wedge \langle \text{MOR} \rangle (i \wedge \langle \text{GEN} \rangle het \wedge \langle \text{DEF} \rangle -)) \wedge \langle \text{ARG}^* \rangle (\langle \text{CAT} \rangle n \wedge \langle \text{MOR} \rangle i))$
$\langle \text{PHON} \rangle aardige \wedge \langle \text{CM} \rangle (\langle \text{VAL}^* \rangle (\langle \text{CAT} \rangle n \wedge \langle \text{MOR} \rangle (i \wedge \neg (\langle \text{GEN} \rangle het \wedge \langle \text{DEF} \rangle -))) \wedge \langle \text{ARG}^* \rangle (\langle \text{CAT} \rangle n \wedge \langle \text{MOR} \rangle i))$

The crucial point of this example is that the combination of an adjective with a noun carries morphosyntactic information arising from the adjective as well as the noun. This can be seen when we combine the adjectives and the noun to yield *aardig kind*. Unification and reentrancies can be presented as in the following formula that is made to look like a feature structure.

⟨CM⟩	[1] ⟨MOR⟩ [3]		
⟨FUNCTOR⟩	⟨PHON⟩	*aardig*	
	⟨CM⟩	⟨VAL*⟩ [1]	⟨CAT⟩*n*
			⟨MOR⟩[3](⟨GEN⟩*het* ∧ ⟨DEF⟩−)
		⟨ARG*⟩ [2]	⟨CAT⟩*n*
			⟨MOR⟩[3]
⟨ARGUMENT⟩	⟨PHON⟩	*kind*	
	⟨CM⟩ [2]	⟨CAT⟩*n*	
		⟨MOR⟩ [3]	⟨GEN⟩ *het*

As a result of the application schema, the morphological information of the noun unifies with the information on the argument position of the adjective and the information on the argument position is reentrant with the information on the result position. The effect is that the modifier (1) mediates the information of the noun to the combination and (2) adds information of its own.

The same procedure can be carried out for further combinations with the determiners. Of course not all combinations are grammatical. In the case of the ungrammatical *aardig jongen*, the various constraints at the node labelled [3] are incompatible: at the adjective position the gender is required to be *het*, whereas at the noun position it is required to be *de*. There is no feature structure that satisfies this formula.

Now consider an approach based on subsumption. We can use the following lexical assignments.

een	$np/(n \wedge indef)$	de	$np/(n \wedge de)$
het	$np/(n \wedge het)$	jongen	$n \wedge de \wedge indef \wedge def$
kind	$n \wedge het \wedge indef \wedge def$	aardig	$(n \wedge het \wedge indef)/(n \wedge het \wedge indef)$
aardige	$(n \wedge de \wedge def \wedge indef)/(n \wedge de)$	aardige	$(n \wedge het \wedge def)/(n \wedge def)$

The fact that nouns are neutral with respect to definiteness is indicated by 'overspecification' (see [8], [9]). So *jongen* is both *indef* and *def* at the same time. A problem arises with the specification of *aardige*, for which we need two entries. Using a single assignment such as $(n \wedge \neg(het \wedge indef))/(n \wedge \neg(het \wedge indef))$ does not work. To see this, consider the case of the ungrammatical *een aardige kind*. The single assignment fails to exclude the phrase. The reason is that the values on the argument category and the resultant category are unrelated: the assignment subsumes the category $(n \wedge de \wedge indef)/(n \wedge het \wedge def)$.

This constrasts with the constraint-based analysis and the analysis we propose below. Bouma notes that such a multiplication of lexical assignments (which may be worse if you consider languages with richer nominal inflection, such as German) may be considered as 'missing a generalisation' by some. Although, missing this generalisation need not be considered such a great burden on the grammar, it would be interesting nevertheless to try and construct a grammar that is subsumption based and that needs only one lexical assignment. This we will do in the next section. Our main purpose, however, is to illustrate the expressive power of the multimodal approach to grammar writing which allows us to encode feature distribution principles mimicking in part the effect of unification.

3.2 A multimodal Analysis

The multimodal analysis of this construction uses the following ingredients.

1. A selection of resource modes on unary modalities expressing featural information: d, h, o, db, do, hb, ho, $-ho$.

2. A selection of resource modes on binary modalities expressing the mode of combination: *specifier/head (= s), modifier/head (= m).*
3. Inclusion postulates expressing an ordering on the feature modes (see below).
4. Interaction postulates expressing the distribution of features over combination modes.
5. Lexical assignments as follows.

de	$\text{NP}/[db]\text{N}$	het	$\text{NP}/[hb]\text{N}$	een	$\text{NP}/[o]\text{N}$
jongen	$[d]\text{N}$	kind	$[h]\text{N}$		
aardig	$[ho](\text{N}/\text{N})$	aardige	$[-ho](\text{N}/\text{N})$		

Let us review the features/sorts used briefly. $[d]$ is used for a word like *jongen*, to mark its grammatical non-neuter gender: it combines with the definite determiner *de*. Similarly, $[h]$ marks neuter words, those that combine with the definite determiner *het*. The definite determiners require their complements to be 'definite' and have the appropriate gender. We use b for definite and o for indefinite. $[db]$ is the requirement put on nouns and their projections by the definite non-neuter determiner *de*, $[hb]$ is the requirement put on nouns by the definite neuter determiner *het*. The adjectives divide into inflected and uninflected ones. The latter occur only in indefinite neuter environments, which is signalled by $[ho]$ whereas the former occur in all other environments, hence $[-ho]$, to be thought of as "not neuter and indefinite".

First, we consider the derivations for the grammatical *de aardige jongen* and the ungrammatical *het aardige jongen*.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{N}/\text{N} \circ_m \text{N} \Rightarrow \text{N}}{([-ho](\text{N}/\text{N}))^{\langle -ho\rangle} \circ_m ([d]\text{N})^{\langle d\rangle} \Rightarrow \text{N}}}{([-ho](\text{N}/\text{N}))^{\langle db\rangle} \circ_m ([d]\text{N})^{\langle db\rangle} \Rightarrow \text{N}}\,\sqsubseteq}{([-ho](\text{N}/\text{N}) \circ_m [d]\text{N})^{\langle db\rangle} \Rightarrow \text{N}}\,\mathbf{A}}{[-ho](\text{N}/\text{N}) \circ_m [d]\text{N} \Rightarrow [db]\text{N}}}{\text{NP} \Rightarrow \text{NP}}}{\text{NP}/[db]\text{N} \circ_s [-ho](\text{N}/\text{N}) \circ_m [d]\text{N} \Rightarrow \text{NP}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\textit{fail}}{([-ho](\text{N}/\text{N}))^{\langle hb\rangle} \circ_m ([d]\text{N})^{\langle hb\rangle} \Rightarrow \text{N}}\,\sqsubseteq}{([-ho](\text{N}/\text{N}) \circ_m [d]\text{N})^{\langle hb\rangle} \Rightarrow \text{N}}\,\mathbf{A}}{[-ho](\text{N}/\text{N}) \circ_m [d]\text{N} \Rightarrow [hb]\text{N}}}{\text{NP} \Rightarrow \text{NP}}}{\text{NP}/[hb]\text{N} \circ_s [-ho](\text{N}/\text{N}) \circ_m [d]\text{N} \Rightarrow \text{NP}}$$

The crucial points to note about these derivations are the following. First, the agreement relation between the adjective and the noun is expressed through the distribution of the agreement features over both the modifier and the noun. We assume the distribution principles:

$$\dfrac{\Gamma[\Delta_1^{\langle db\rangle} \circ_m \Delta_2^{\langle db\rangle}] \Rightarrow Z}{\Gamma[(\Delta_1 \circ_m \Delta_2)^{\langle db\rangle}] \Rightarrow Z}\,\mathbf{A} \qquad \dfrac{\Gamma[\Delta_1^{\langle hb\rangle} \circ_m \Delta_2^{\langle hb\rangle}] \Rightarrow Z}{\Gamma[(\Delta_1 \circ_m \Delta_2)^{\langle hb\rangle}] \Rightarrow Z}\,\mathbf{A}$$

Second, the derivations rely on several inclusion postulates which are needed to make the inferences marked '$\sqsubseteq$'. In the case of *de aardige jongen* we need the rules:

$$\dfrac{\Gamma[(X)^{\langle -ho\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle db\rangle}] \Rightarrow Z}\,\sqsubseteq \qquad \dfrac{\Gamma[(X)^{\langle d\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle db\rangle}] \Rightarrow Z}\,\sqsubseteq$$

We do not want the inclusion postulate relating $\langle d\rangle$ and $\langle hb\rangle$ in our grammar, because this would imply that checking whether something is a neuter word amounts to checking whether it is a non-neuter definite word, so this why the derivation for *het aardige jongen* fails.

Now consider the similar derivations for *een aardige jongen* en *een aardig kind*.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{N}/\text{N} \circ_m \text{N} \Rightarrow \text{N}}{([-ho](\text{N}/\text{N}))^{\langle -ho\rangle} \circ_m ([d]\text{N})^{\langle d\rangle} \Rightarrow \text{N}}}{([-ho](\text{N}/\text{N}))^{\langle o\rangle} \circ_m ([d]\text{N})^{\langle o\rangle} \Rightarrow \text{N}}\,\sqsubseteq}{([-ho](\text{N}/\text{N}) \circ_m [d]\text{N})^{\langle o\rangle} \Rightarrow \text{N}}\,\mathbf{A}}{[-ho](\text{N}/\text{N}) \circ_m [d]\text{N} \Rightarrow [o]\text{N}}}{\text{NP} \Rightarrow \text{NP}}}{\text{NP}/[o]\text{N} \circ_s [-ho](\text{N}/\text{N}) \circ_m [d]\text{N} \Rightarrow \text{NP}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{N}/\text{N} \circ_m \text{N} \Rightarrow \text{N}}{([ho](\text{N}/\text{N}))^{\langle ho\rangle} \circ_m ([h]\text{N})^{\langle h\rangle} \Rightarrow \text{N}}}{([ho](\text{N}/\text{N}))^{\langle o\rangle} \circ_m ([h]\text{N})^{\langle o\rangle} \Rightarrow \text{N}}\,\sqsubseteq}{([ho](\text{N}/\text{N}) \circ_m [h]\text{N})^{\langle o\rangle} \Rightarrow \text{N}}\,\mathbf{A}}{[ho](\text{N}/\text{N}) \circ_m [h]\text{N} \Rightarrow [o]\text{N}}}{\text{NP} \Rightarrow \text{NP}}}{\text{NP}/[o]\text{N} \circ_s [ho](\text{N}/\text{N}) \circ_m [h]\text{N} \Rightarrow \text{NP}}$$

We have used the following inclusion postulates to account for these derivations.

$$\frac{\Gamma[(X)^{\langle -ho\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle o\rangle}] \Rightarrow Z} \sqsubseteq \frac{\Gamma[(X)^{\langle d\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle o\rangle}] \Rightarrow Z} \sqsubseteq \frac{\Gamma[(X)^{\langle ho\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle o\rangle}] \Rightarrow Z} \sqsubseteq \frac{\Gamma[(X)^{\langle h\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle o\rangle}] \Rightarrow Z} \sqsubseteq$$

And we have used the distribution postulate:

$$\frac{\Gamma[\Delta_1^{\langle o\rangle} \circ_m \Delta_2^{\langle o\rangle}] \Rightarrow Z}{\Gamma[(\Delta_1 \circ_m \Delta_2)^{\langle o\rangle}] \Rightarrow Z} \mathbf{A}$$

However, we run into problems with the ungrammatical indefinites **een aardig jongen* en **een aardige meisje*. With the distribution postulates assumed in the previous derivations, we can also derive these ungrammatical noun phrases.

The problem is easily diagnosed. In the indefinite case, we have to worry about the *gender* agreement between the noun and the adjective as well. We can do this by not allowing the feature o to distribute over the adjective-noun combinations, but only allow the more specific postulates for *do* and *ho*.

$$\frac{\Gamma[\Delta_1^{\langle do\rangle} \circ_m \Delta_2^{\langle do\rangle}] \Rightarrow Z}{\Gamma[(\Delta_1 \circ_m \Delta_2)^{\langle do\rangle}] \Rightarrow Z} \mathbf{A} \quad \frac{\Gamma[\Delta_1^{\langle ho\rangle} \circ_m \Delta_2^{\langle ho\rangle}] \Rightarrow Z}{\Gamma[(\Delta_1 \circ_m \Delta_2)^{\langle ho\rangle}] \Rightarrow Z} \mathbf{A}$$

We replace the inclusion postulates involving o with the following:

$$\frac{\Gamma[(X)^{\langle do\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle o\rangle}] \Rightarrow Z} \sqsubseteq \frac{\Gamma[(X)^{\langle ho\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle o\rangle}] \Rightarrow Z} \sqsubseteq \frac{\Gamma[(X)^{\langle -ho\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle do\rangle}] \Rightarrow Z} \sqsubseteq \frac{\Gamma[(X)^{\langle d\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle do\rangle}] \Rightarrow Z} \sqsubseteq \frac{\Gamma[(X)^{\langle h\rangle}] \Rightarrow Z}{\Gamma[(X)^{\langle ho\rangle}] \Rightarrow Z} \sqsubseteq$$

The derivation for *een aardige jongen* now looks different, because we can no longer distribute $\langle o\rangle$ over the adjective and the noun. We first have to apply one of the inclusion postulates, turning the check for indefiniteness into a check for indefiniteness neuter, $\langle do\rangle$, or indefiniteness non-neuter, $\langle ho\rangle$. In this case the indefiniteness non-neuter option $\langle do\rangle$ works. In the ungrammatical case *een aardig jongen*, neither of the two will work.

Discussion The principal characteristics of the grammar fragment are that (1) features are encoded as unary modalities; (2) these are ordered hierarchically by means of inclusion postulates and (3) the distribution of features in syntactic structure is partly defined in terms of distribution laws encoded as interaction postulates.

It is interesting, with respect to this last point, to compare the set-up with feature-distribution principles in a theory like HPSG. The need for various HPSG principles, *head-feature principle, valence principle, immediate dominance, slash-inheritance, etc.* is a reflection of the fact that not all information in syntactic structure is governed by a single distribution law. The functional nature of categories and the formation of constituents by application in categorial grammars combine the effects of only a few principles in HPSG. Interaction postulates complement this.

On the subject of unification versus subsumption we would like to point out, that in many cases structure sharing is used in HPSG where simple equality of information is needed. With the distribution principles described above we cannot enforce structure sharing, we can only make sure that the same information is found in different places. Our conjecture is that such a restricted mechanism is sufficient for writing grammars.

Summarising, the grammar fragment above illustrates some of the expressive capacities of multimodal categorial logic. It shows:

- how feature information can be introduced on both basic and complex types;
- how feature information can be organized by means of inclusion postulates;
- how feature distribution principles can be implemented by means of interaction postulates;
- how such postulates may obviate the need for (i.e. replace) unification in grammatical description.

As in the propositional extension our "feature structures" do not have attribute names but only sorts. In the example we discussed we only used simple modalities but this is not a principled restriction. We could stack features or use a more complicated logic for the labels to add the conjunctive aspect from ordinary feature structures.

Conclusion We have discussed three varieties of categorial grammars that include the means to decompose categories into some kind of feature bundles (a classic move in linguistics). The categorial unification grammar differs from the two type-logical grammars in that they define grammaticality in terms of satisfiability instead of validity. The first type-logical grammar replaces basic categories with feature structures, the second uses unary modalities as feature decorations. Both embody only limited aspects of the notion of feature structure as it is standardly found in constraint-based theories, but this restriction need not be considered a drawback. The modal approach benefits from the possibility of defining modal postulates that allow the specification of more complicated patterns of feature checking, mimicking the effect of unification.

References

[1] Patrick Blackburn. Structures, languages and translations: the structural approach to feature logic. In C.J. Rupp, M. A. Rosner, and R.L. Johnson, editors, *Constraints, Language and Computation*, pages 1–27. Academic Press, London, 1994.

[2] Patrick Blackburn and Edith Spaan. A modal perspective on the computational complexity of attribute value grammar. *Journal of Logic, Language, and Information*, 2(2):129–169, 1993.

[3] Gosse Bouma. *Nonmonotonicity and Categorial Unification Grammar.* PhD thesis, R.U. Groningen, The Netherlands, 1993.

[4] Bob Carpenter. *The Logic of Typed Feature Structures.* Cambridge University Press, Cambridge, 1992.

[5] Jochen Dörre, Dov Gabbay, and Esther König. Fibred semantics for feature-based grammar logic. *Journal of Logic, Language, and Information*, 5(3-4):387–422, October, 1996.

[6] Jochen Dörre and Suresh Manandhar. On constraint-based Lambek calculi. In Patrick Blackburn and Maarten de Rijke, editors, *Logic, Structures and Syntax.* Reidel, Dordrecht, to appear.

[7] Nissim Francez. On the direction of fibring feature logics with concatenation logics. Nancy, LACL 1997.

[8] Dirk Heylen. On the proper use of booleans in categorial logic. In *Formal Grammar*, pages 71–84, Prague, 1996.

[9] Mark Johnson and Sam Bayer. Features and agreement in Lambek categorial grammar. In Glyn Morrill and Richard Oehrle, editors, *Formal Grammar*, pages 123–137, Barcelona, 1995.

[10] Michael Moortgat. Categorial type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language.* Elsevier, 1996.

[11] Glyn Morrill. *Type Logical Grammar.* Kluwer Academic Publishers, Dordrecht, 1994.

[12] Koen Versmissen. *Grammatical Composition: Modes, Models, Modalities.* PhD thesis, Utrecht University, Utrecht, 1996.

BUBBLE TREES AND SYNTACTIC REPRESENTATIONS[1]

Sylvain KAHANE[2]

Abstract. A new mathematical object, the bubble tree, is introduced and applied to the syntactic representation of sentences. Two themes are explored: first, the comparison of dependency and constituency models; second, the application of bubble trees to a particular syntactic model.

1. Introduction

One of the major issues in modern linguistics is the lack of a common language among linguists (as opposed to, say, mathematics), which leads to problems of finding correspondences between different idiolects. In particular, two differents models can hide more similarities than it appears at first sight. In Section 4, the two main models of syntactic representation, dependency and constituency, will be compared with the support of a common representational device, the **bubble tree**. Intuitively, bubble trees are trees whose nodes are bubbles which in turn contain sub-bubbles linked to other bubbles and so on. The only formal study of such mathematical structures, as far as I know, is by Gladkij 1968.

Section 5 describes some complex syntactic phenomena, such as coordination, extraction and word order, using representations based on bubble trees. Detailed linguistic descriptions and computational applications cannot be presented in this short communication. They will be the subject of a further communication.

2. Prerequisites

A **tree** can be viewed as an oriented graph or as a binary relation $\lhd$ (in this case we will call it a **tree relation**) ($x \lhd y$ if and only if (y, x) is a link of the corresponding graph). A tree relation induces a **domination relation** $\preceq$ defined by $x \preceq y$ if and only if $x = x_1 \lhd \cdots \lhd x_n = y$ $(n \geq 0)$. The **root** of a tree is the only node which dominates all other nodes. A **terminal node** is a node without dependents.

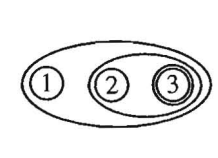
A **constituency tree** on X (Bloomfield 1933, Chomsky 1957) is a four-tuple $(X, \mathcal{B}, \varphi, \lhd)$ where $\mathcal{B}$ is the set of **constituents**, $\lhd$ is a relation on $\mathcal{B}$ and φ is a map from $\mathcal{B}$ to the non-empty[3] subsets of X (which describes the **content** of constituents) such that:

P1. $\lhd$ is a tree relation.

P2. Any one-element subset of X is the content of one and only one terminal node.

P5. If $\alpha \lhd \beta$, then $\varphi(\alpha) \subseteq \varphi(\beta)$.

A **dependency tree** on X (Tesnière 1934, 1959, Hays 1960, Lecerf 1961) is in fact a plain tree on X.[4] Any dependency tree $(X, \lhd_1)$ induces a constituency tree $(X, \mathcal{B}, \varphi, \lhd_2)$: each node x produces two constituents, noted x and $\bar{x}$ such that $\varphi(x) = \{x\}$ and $\varphi(\bar{x})$ is the **projection** of x, i.e. the set of nodes of X dominated by x (variant: when x is a terminal node x for $\lhd_1$, x and $\bar{x}$ can be identified). The relation $\lhd_2$ is $\lhd_1$ on the bar-constituents and $x \lhd_2 \bar{x}$ for every $x \in X$.



A dependency tree and
the corresponding constituency tree

To give a constituency tree heads (resp. co-heads) means to choose a (resp. a set of) head sub-constituent(s) in each constituent (Pittman 1948). So a **co-headed constituency tree** on X is a quintuple $(X, \mathcal{B}, \varphi, \lhd, \tau)$, where $(X, \mathcal{B}, \varphi, \lhd)$ is a constituency tree and τ a map from $\bar{\mathcal{B}}$ (= the subset of $\mathcal{B}$ of non terminal constituents) to the non-empty subsets of $\mathcal{B}$ such that $\beta \lhd \alpha$ for each $\beta \in \tau(\alpha)$. If $\tau(\alpha)$ has a single element, α is said to be **headed**, otherwise α is said to be **co-headed** or **sub-headed** (in the latter case, α is considered to be a potentially headed constituent for which the head is subspecified). A **head node** of a bubble α is a node obtained in descending from α following only head constituents. The **kernel** of α is the subset of **head nodes** of α.

1. The present paper was read and commented on by Anne Abeillé, David Beck, Dick Hudson and Igor Mel'čuk. I thank them.

2. TALANA (Univ. Paris 7) and Univ. Paris 10 - Nanterre. E-mail: sk@ccr.jussieu.fr

3. That does not mean that we ignore empty constituents. Elements of X are abstract nodes which are associated with signs, which do not need a phonological realisation.

4. In a more general way, dependency trees are trees whose nodes are labelled with grammatical categories and links with functional attributes, but we are not directly concerned with the labelling here.

A headed constituency tree induces a dependency tree (Lecerf 1961, Gaifman 1965, Robinson 1970), but the dependency relation is not explicit. We will introduce equivalent structures to headed constituency trees, which makes dependency relations more explicit.

3. Bubble trees

A **bubble tree** is a four-tuple $(X, \mathcal{B}, \varphi, \lhd)$, where X is the set of **basic nodes**, $\mathcal{B}$ is the set of **bubbles**, φ is a map from $\mathcal{B}$ to the non-empty subsets of X (which describes the **content** of bubbles) and $\lhd$ is a relation on $\mathcal{B}$ verifying P1, P2[5] and

P3. If $\alpha, \beta \in \mathcal{B}$, then $\varphi(\alpha) \cap \varphi(\beta) = \emptyset$ or $\varphi(\alpha) \subseteq \varphi(\beta)$ or $\varphi(\beta) \subseteq \varphi(\alpha)$.

P4. If $\varphi(\alpha) \subset \varphi(\beta)$, then $\alpha \prec \beta$. If $\varphi(\alpha) = \varphi(\beta)$, then $\alpha \preceq \beta$ or $\beta \preceq \alpha$.

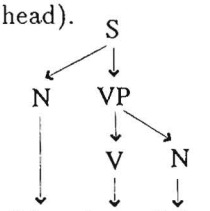
Bubble trees are thus defined.[6] The relation $\lhd$ is called **dependency-embedding relation**. Two sub-relations of $\lhd$ are considered, the **dependency relation** $\lhd\!\!\lhd$ defined by $\alpha \lhd\!\!\lhd \beta$ if $\alpha \lhd \beta$ and $\varphi(\alpha) \cap \varphi(\beta) = \emptyset$ and the **embedding relation** $\lhd\!\!\!\cdot$ defined by $\alpha \lhd\!\!\!\cdot \beta$ if $\alpha \lhd \beta$ and $\alpha \subseteq \beta$. We will say that α **depends** on β if $\alpha \lhd\!\!\lhd \beta$ and α is **directly embedded** in β if $\alpha \lhd\!\!\!\cdot \beta$. In the following figures, $\lhd\!\!\lhd$ will be represented by links and $\lhd\!\!\!\cdot$ by inclusion of bubbles. The **projection** of a bubble α is the union of the contents of all the bubbles dominated by α, including α.

4. Comparison between dependency and constituency

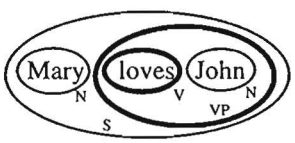
4.1. Various representations of the same structure

Consider a co-headed constituency tree $B_1 = (X, \mathcal{B}_1, \varphi_1, \lhd_1, \tau)$. We associate it with a bubble tree $B_2 = (X, \mathcal{B}_2, \varphi_2, \lhd_2)$ where $\mathcal{B}_2 = \mathcal{B}_1$, $\lhd_2 = \lhd_1$ and $\alpha \lhd\!\!\!\cdot_2 \beta$ if and only if $\alpha \in \tau(\beta)$. Such a bubble tree is called a **bi-tree**. Note that $\varphi_2(\alpha)$ is the kernel of α in B_1, and $\varphi_1(\alpha)$ is the projection of α in B_2. Therefore, co-headed constituency trees and bi-trees are in one-to-one correspondence. A bi-tree corresponding to a headed constituency tree is called a **stratified tree**; in this case, every bubble contains a unique element (the head of the constituent). Any stratified tree T trivially induces a dependency tree T'; to obtain T' from T, it is sufficient to collapse all bubbles with the same content.[7]

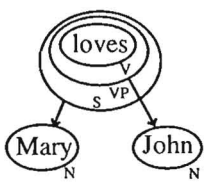
Example. We will give four representations of an X-bar tree of *John loves Mary* (X-bar trees are headed constituency trees; τ is generally encoded in the node labelling; here τ is represented by lining up any node with its head).



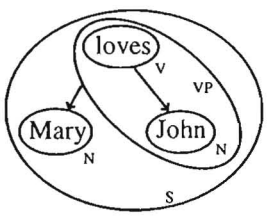
An X-bar tree



the same X-bar tree



the corresponding stratified tree



the corresponding Gladkij tree

Bi-trees can be easily characterised. The bubble β is a **sub-bubble** of α if $\beta \subseteq \alpha$ and $\beta \prec \alpha$. The bubble β is an **immediate sub-bubble** of α if β is a sub-bubble of α but not a sub-bubble of a sub-bubble of α. Note that an immediate sub-bubble of α is either directly embedded in α or depends on another immediate sub-bubble of α. A bubble tree is a **bi-tree** if and only if any immediate sub-bubbles of any bubble α is directly embedded in α. In other words, a link cannot be included in a bubble.

There is another way to encode a headed constituency tree with a bubble tree, but this does not work with a co-headed constituency tree. A headed constituency tree $B_1 = (X, \mathcal{B}_1, \varphi_1, \lhd_1, \tau)$ can be associated with a bubble tree $B_3 = (X, \mathcal{B}_3, \varphi_3, \lhd_3)$ where $\mathcal{B}_3 = \mathcal{B}_1$, $\varphi_3 = \varphi_1$ and $\alpha \lhd_3 \beta$ if and only if either $\alpha = \tau(\beta)$ or there exists γ such that $\alpha \lhd_1 \gamma$ and $\beta = \tau(\gamma)$. The resulting bubble tree is called a **Gladkij tree**.[8] A combination of both types of representation will give us hybrid representations of bi-trees and Gladkij trees (for example Vergne's model (Vergne 1994) uses such structures).[9]

---

5. P2 can be weakened to authorize a "lexical" bubble to have descendents.

6. Gladkij 1968 defines a particular case of bubble trees (he supposes in particular that each link must be contained in a bubble and that there must exist a bubble which contains all the nodes). Moreover, $\lhd$ and $\lhd\!\!\lhd$ are not clearly distinguished and P1 is only partially stated.

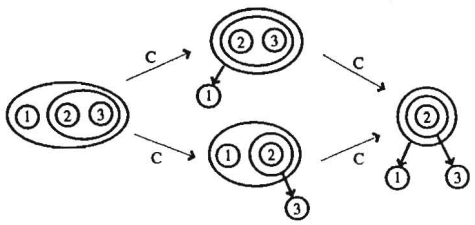
7. If the stratified tree is labelled by categories, the induced dependency tree must be labelled by sequences of categories.

8. This kind of bubble trees are bubble trees in the sense of Gladkij 1968. But Gladkij's formal and linguistic interpretations of these trees are very different from what is offered here.

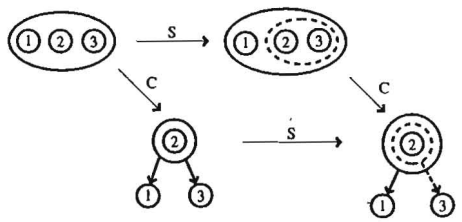
9. A dependency tree $T=(X, \lhd_1)$ is said to be **compatible** with a bubble tree $B=(X, \mathcal{B}, \varphi, \lhd_2)$ if for each link $x \lhd_1 y$ of T, there exists a link $\alpha \lhd\!\!\lhd_2 \beta$ of B with $x \in \varphi(\alpha)$ and $y \in \varphi(\beta)$. Note that the dependency tree induced by a stratified tree is compatible with it and is also compatible with the corresponding Gladkij tree. Therefore, the two representations can be mixed without problems: the induced dependency tree can be recovered by compatibility.

4.2. Concentration vs stratification, dependency vs constituency

The bi-tree $B_1 = (X, \mathcal{B}, \varphi_1, \lhd)$ is said to be more **concentrated** than the bi-tree $B_2 = (X, \mathcal{B}, \varphi_2, \lhd)$ if for every α, $\varphi_1(\alpha) \subseteq \varphi_2(\alpha)$. The **concentration** is a partial order on bi-trees. Maximal elements are stratified trees and minimal elements constituency trees. **Concentrating** a given bi-tree consists in changing an immediate sub-bubble β of a bubble α (which has at least two immediate sub-bubbles) into a dependent bubble of α.



To **stratify** a constituency tree consists simply in adding any intermediate constituents. For a bi-tree, this is generally more complicated, because we want to ensure the commutativity between the operations of concentration and stratification. **Stratification** of a given bi-tree consists in adding to some non-terminal bubble α an immediate sub-bubble β and in distributing α's dependencies among α and β. A bi-tree B_1 is said to be more **stratified** than the bi-tree B_2 if B_1 is obtained by stratifying B_2. The stratification is a partial order on bi-trees. Stratification has no maximal elements. Minimal stratified trees are dependency trees or, more exactly, bi-trees corresponding to the headed constituency trees associated with these dependency trees.



Note that concentration does not change stratification and vice-versa, and the two operations commute (see figure above). Clearly, concentration measures the degree of dependency or headedness and stratification, the degree of constituency. Contemporary syntactic models are generally split into dependency models (MTT (Mel'čuk 1988), WG (Hudson 1990), ...) and constituency models (GB (Chomsky 1981), G/HPSG (Gazdar & al. 1985, Pollard & Sag 1994), LFG (Kaplan & Bresnan 1982), TAG (Joshi 1987), CG (Bar-Hillel 1953, Moortgaat 1988), ...). In fact, all these models use dependency (and of course constituency, in the sense that any dependency tree canonically induces a constituency tree) and their respective classification depends on how these models are presented (i.e. whether the dependency relation is explicit or not), not on concentration and stratification. For example, GB is a real dependency model (c-command and government cannot be defined without the notion of head) and it is even more concentrated than Tesnière's model.[10]

There is a third important operation on bubble trees: **granularisation**. The **aggregation** of a bi-tree consists in collapsing together two "adjacent" basic nodes. A bi-tree B_1 is said to be more **granular** than B_2 if B_2 is obtained by aggregating B_1. For example, GB is more granular than other models because it considers two nodes V and $INFL$ where other models consider only the node V. More generally, models that work with morphemes are more granular than models that work with words.

5. A particular model based on bubble trees

We will now defend a particular syntactic representation using bubble trees. Our basic representation is a standard dependency tree which is roughly equivalent to Tesnière's stemma or the Surface Syntactic Structure of MTT, the deep structure of WG, the *f*-structure of LFG, the derivation tree of TAG, the *d*-structure of GB or the subcategorization structure of G/HPSG. These theories differ most in the representation of some complex phenomena such as coordination or extraction, to which we now turn.

On the one hand, the **well-formedness** of syntactic structures is controlled, by some general principles such as projectivity, coordination principles, ... (see §§5.1-2) and, on the other, by lexical frames[11] (a **lexical frame**, which is a part of a given lexical entry *a*, describes an acceptable structural environment for *a*, that is, the nature and the government (= *régime*) of its arguments or the nature of its head if it is a modifier)[12].

5.1. Coordination

Dependency links are a good way to formalize subordination. But coordination is an orthogonal operation and must be formalize in an orthogonal way. Bubbles offer us a good solution.

10. Tesnière's stemma is not exactly a tree as most linguists think. Some phrases, such as determinant-noun, auxiliary-participle, complementizer-verb ..., are grouped in a bubble, called a nucleus. Thus, the stemma is a bubble tree with co-head bubbles.

11. The border line between lexical frames and general principles is not clear; for example, agreement rules can either be general principles directly applied to the structure or can be encoded in lexical frames. Nevertheless, in the latter case, an agreement rule must not be introduced independently in each particular lexical frame, but must be stated by a rule at a meta-level: agreement rules belong to a meta-lexicon, which controls the correctness of elements of the lexicon, i.e. the lexical frames).

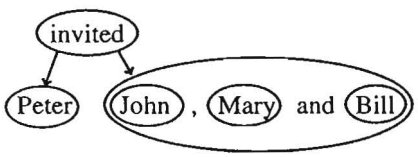
12. It can be supposed that a lexical frame can deal only with adjacent nodes of the node occupied by the lexical unit in question (this is generally called the locality principle). In dependency structures, the locality principle assumes that a lexical unit controls only its governor and dependents. In a certain sense, that defines the head; the head of a constituent is the lexical unit which determines the possibilities of insertion of the constituent (Pittman 1948, Garde 1967, Mel'čuk 1988). Nevertheless, these possibilities do not necessarily depend on only one lexical unit and co-heads can be relevant.

Roughly speaking, coordination boils down to the fact that two or more elements together occupy one syntactic position. These elements can be grouped in a bubble, called a **coordination bubble**, which occupies this position. Paradoxically, our description of coordination rests on the notion of head, but cannot be properly encoded in a plain dependency tree.

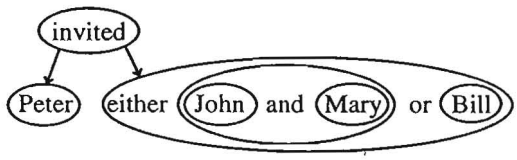
Note that coordination can be developped in two ways:

- the **iterativity** of coordination is the fact that an illimited number of elements can be coordinated and that a coordination bubble can have an illimited number of elements;

- the **recursivity** of the coordination is the fact that coordination bubbles can be coordinated, as in *Peter invited John, Mary and Bill*; the recursivity is linguistically limited to one step and must be well marked (by special words, such as *either*, or prosody).

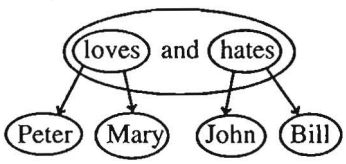


Peter invited John, Mary and Bill
(iterativity of coordination)

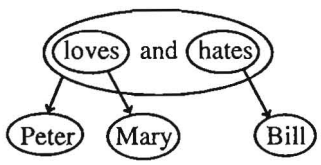


Peter invited either John and Mary or Bill
(recursivity of coordination)

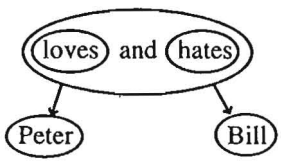
Sharing. Coordinated elements necessarily share their governor (if there is one) and they can share all or part of their dependents. Sharing is constrained: for example, in English, it is easier for two coordinated verbs to share their subject than their object (? *Mary loves and Peter hates Bill*). The whole sharing (*Peter loves and hates Bill*) is a special case of coordination with particular constraints (it is generally called lexical coordination).



Peter loves Mary and John hates Bill



Peter loves Mary and hates Bill



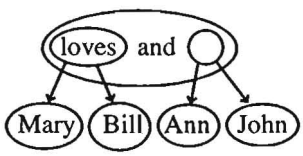
Peter loves and hates Bill

Coordination and lexical frames. A coordination bubble is a special kind of co-headed bubbles. Nevertheless, a coordination bubble requires a particular interpretation. First, the coordination bubble contains two different sorts of objects: coordinated elements on the one hand, and coordinating conjunctions on the other hand. Second, lexical frames apply to coordination bubbles (for example, the verb agrees with its subject which can be a coordination bubble), but also to the coordinated elements by taking into account that the valency (= sub-categorization) of any coordinated element is the union of the valency of every coordination bubble containing it (it is this resulting graph that Tesnière 1959 and Hudson 1990, ∞ adopt as a representation of the coordination). In particular, lexical order rules, such as "the subject is before its governor", apply to a dependent of a bubble just as they do to a dependent of lexical node: thus, *Peter loves and hates Bill*, for example, is the only possible projection of its syntactic structure.

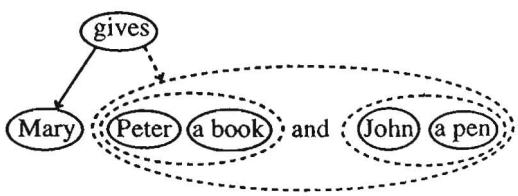
We will now describe two particular kinds of coordination, gapping coordination and valency slot coordination.

Gapping coordination. If two clauses with the same main verb are coordinated, the second occurrence of the verb can be omitted. So we have a verbal bubble with an empty phonological realisation.

Valency slot coordination. A valency slot bubble is a subset of the valency of a governing element grouped in a bubble (with a single link to governor); two valency slot bubbles of the same kind can be coordinated.

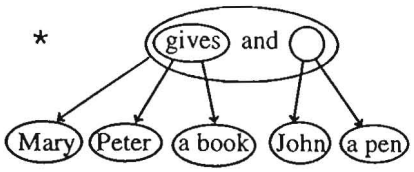


Mary loves Bill and Ann, John



Mary gives Peter a book and John a pen

Gapping coordinations and valency slot coordinations are close, and formally they could be represented in the same way: opposite, we propose representing a valency slot coordination in the same way as a gapping coordination. In fact, valency slot coordination are closer to ordinary coordination than gapping coordination; gapping coordination is more constrained. For example, in French, with the coordinating cunjunction *ainsi que*, only valency slot coordination is possible: *Pierre donne un livre à Pierre, ainsi qu'un crayon à Jean* vs ⁇ *Marie parle à Paul, ainsi qu'Anne à Jean.* The same is true for *as well as* in English: *Peter drinks coffee at 11 as well as tea at 4* vs ⁇ *Peter drank coffee as well as John tea.*



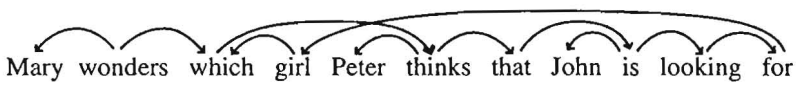
Mary gives Peter a book and John a pen
(as gapping coordination)

On the other hand, gapping coordinations are not valency slot coordinations, because the governing verb agrees with only the first subject. Moreover, our valency slot representation would pose problems for the control of the linear order of gapping coordinations, because we have to assume that the projection of a valency slot bubble, as well as a coordination bubble, must be continuous (see §5.2)

5.2. Word order, projectivity and nuclei

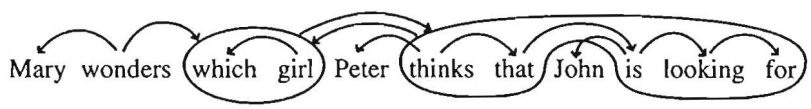
A **dependency** (resp. **constituency**) **structure** is a dependency (resp. constituency) tree on a *linearly ordered* set X. A dependency structure on X is said to be **projective** if links do not cross each other and no link covers an ancestor. A constituency structure is said to be **continuous** if the content of any constituent is continuous. Generally, constituency structures are always supposed to be continuous. Note that a dependency structure is projective if and only if the induced constituency structure is continuous (Lecerf 1961, Gladkij 1966).

Very often, the syntactic dependency tree of a sentence is not projective. The most common type of non-projectivity in English is due to extraction (topicalisation, interrogation, relativisation ...), which we will now study. Following Tesnière 1959 or Hudson ∞, we think that it is better to associate two nodes to a *wh*-word because it assumes two functions: a pronominal function and the same function as other conjunctions, such as *that*, which have the role of subordinating a verb. Both nodes can be combined: we obtain a structure which is not exactly a tree because the *wh*-word has two governors, but which does not make formal problems.



Note that the above "tree" is not projective. Nevertheless, the linear order of such constructions is very constrained and it is not possible to totally renounce projectivity. We need a weaker property which allows all the possible linearisations, but which, combined with the order constraints of the lexical frames, allows only the possible linearisations. Our solution is to use bubble trees.

Roughly, a node of a clause can be **extracted** (= topicalized, relativized, interrogated ...) only if it depends on some main verbal nucleus, where **verbal nuclei** of English are verbs and complex units such as auxiliary–participle (*be eating, have eaten*), verb–infinitive (*want to eat*), verb–conjunction–verb (*think that eat*), verb–preposition (*look for*) and all units built by transitivity from these (*thinks that is looking for*).[13] The extractee is a nominal nucleus containing a *wh*-pronoun. **Nominal nuclei** are nouns (*who*) and complex units such as determinant–noun (*which girl, whose girl*) and noun–noun complement (*the daughter of which man*). Verbal and nominal nuclei can be represented by bubbles and certain links can or must be allocated to the nucleus such as the governing link of the extractee.



The previous ordered bubble tree represented is projective in a sense that I will now make clear.

A linearly ordered bubble-tree is said to be **projective** if bubblinks do not cross each other and no bubblink covers an ancestor or a co-head (where a **bubblink** is either a bubble or a link).[14]

13. Every language can have nuclei, but each language develops its own proper types of nuclei. For example, in French, the nucleus verb–preposition does not exist (**Marie se demande quelle fille Pierre parle à*), but there exists a nucleus verb–subject and verb–direct object (*l'homme dont la fille dort, l'homme dont Pierre aime la fille*) and no other nucleus verb-complement (**l'homme dont Pierre parle à la fille*).

14. The first half of this property is stated in Gladkij 1968.

The signifiance of projectivity depends on the way nuclei are encoded. Although the two properties are not equivalent, projectivity can be described in this case by saying that the projection of every bubble is continuous.[15]

Attention: nuclear structure does not take the place of dependency structure. The former is simply **superimposed** on the latter and it is to the former that projectivity applies. There is fundamental difference between nuclei and coordination bubbles: the nucleus is a marking of a particular string of dependencies which must be consider from a certain point of view as a whole, whereas the coordination bubble is an orthogonal operation which must not be describe in term of dependencies.

5.3. Nuclei and coordination

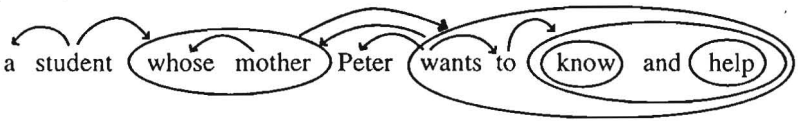
It is well known that there is some constraints between extraction and coordination (Ross 1967):

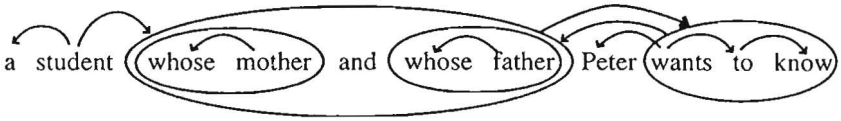
* *a student whose mother Peter knows and helps Mary*

* *a student whose mother and his father Peter knows*

In all the models which I know —constituency or dependency based—, these constructions must be blocked by complicate special constraints.

In our model, possible cases of coordination are of course allowed because a node of a nucleus can naturally be occupied by a coordination bubble:





On the other hand, non acceptable cases of coordination are blocked without additional constraints by our constraint on the extraction (§5.2) and our particular representation of the *wh*-words. Thus, * *a student whose mother Peter knows and helps Mary* is blocked because the nominal nucleus *whose mother* is not a complement of the main verbal nucleus *knows and helps* (but only to *knows*) and * *a student whose mother and his father Peter knows* is blocked because *whose* and *his* do not have the same lexical frame and the two nominal nuclei cannot be coordinated (*wh*-words own very special lexical frame: they have two governors !).

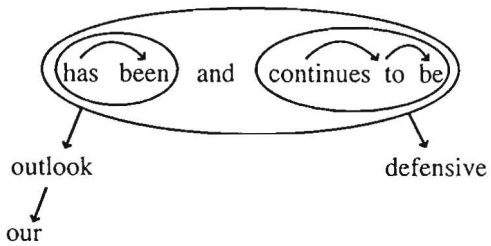
5.4. Coordination and nuclei

The notion of nucleus also allows us to explain some facts of coordination which cannot be described without extending our definition of coordination. We claim that a verbal nucleus can be coordinated with a verb or with another verbal nucleus:

Our outlook ***has been*** *and* ***continues to be*** *defensive.*

a picture that Peter ***likes*** *and* ***is trying to buy.***

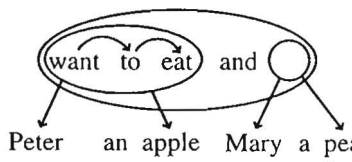
Verbal nuclei in coordination are more constrained than verbal nuclei in extraction: not all kinds of verbal nuclei can be coordinated.



Gapping coordination is possible with verbal nuclei as it is with a verb:

Peter ***wants to eat*** *an apple and Mary, a pear.*

Mary ***is looking for*** *a beautiful landscape and Peter, a pretty girl.*[16]



Nuclei also intervene in a valency slot coordination, in the sense that two nodes can be grouped in valency slot bubble if their governors belong to the same nucleus and are thus coordinated: *John went to London in April and Boston in June.*

Other kinds of nuclei can also be coordinated. For example, one can find in a dictionary this definition: *hedonism:* ***pursuit of*** *or* ***devotion to*** *pleasure.*

15. Projectivity can be ensured in other equivalent ways. For example, all the links of the nucleus can be allocated to the head node (that is what is done by Hudson ∞ with raising or by G/HPSG with the slash feature (see Kahane 1996)), or the links can be labelled as extra- and intranuclear links and projectivity stated in terms of which kinds of links can cut or cover each other (see again Kahane 1996).

16. Such a coordination (with the ellipsis of the preposition) is impossible in French where the nucleus verb-preposition does not exist.

5.5. Conclusion

Constituency structures, as well as plain dependency structures, are too poor: they force us to put in a same dimension subordination and coordination. But subordination and coordination are two orthogonal linguistic operations and we need a two dimentional formalism to capture this, such that bubble trees.

References

BAR-HILLEL Yehoshua, 1953, *A quasi-arithmetical notation for syntactic description*, Language **29.1**, 47-58.

BLOOMFIELD Leonard, 1933, *Language*, New York: Holt, 566 p.

CHOMSKY Noam, 1957, *Syntaxic structures*, La Haye: Mouton, 116 p.

CHOMSKY Noam, 1981, *Lectures on Government and Binding*, Dortrecht: Foris.

GAIFMAN Haïm, 1965, "Dependency systems and phrase-structure systems", *Information and Control 8*, 304-337; Rand Corporation, 1961, RM-2315.

GARDE Paul, "Ordre linéaire et dépendance syntaxique: contribution à une typologie", *Bull. Soc. Ling. Paris* **72.1**, 1-26.

GAZDAR Gerald, KLEIN Ewan, PULLUM Geoffrey K. & SAG Ivan A., 1985, *Generalised Phrase Structure Grammar*, Cambridge, Mass.: Harvard Univ. Press.

GLADKIJ Aleksej V., 1966, *Leckii po matematičeskoj linguistike dlja studentov NGU*, Novosibirsk (French transl: *Leçons de linguistique mathématique, fasc. 1*, 1970, Paris: Dunod).

GLADKIJ Aleksej V., 1968, "On describing the syntactic structure of a sentence" (in Russian with English summary), Computational Linguistics **7**, Budapest, 21-44.

HAYS David, 1960, "Grouping and dependency theory", Rand Corporation, RM-2646.

HUDSON Richard A., 1990, *English Word Grammar*, Oxford: Blackwell.

HUDSON Richard A., ∞, "Discontinuity", e-preprint (ftp.phon.ucl.ac.uk).

JOSHI Aravind K., 1987, "Introduction to Tree Adjoining Grammar", in A. Manaster Ramer (ed.), *The Mathematics of Language*, Amsterdam: J. Benjamins, 87-114.

KAHANE Sylvain, 1996, "If HPSG were a dependency grammar ...", *Proc. 3rd TALN Conf.*, Marseille, 45-49.

KAPLAN Ronald M. & BRESNAN Joan, 1982, "LFG: a formal system for grammatical representation", in J. Bresnan (ed.), *The mental representation of grammatical relations*, Cambridge, Mass.: MIT Press.

LECERF Yves, 1961, "Une représentation algébrique de la structure des phrases dans diverses langues natuelles", *C. R. Acad. Sc. Paris* **252**, 232-234.

MEL'CUK Igor, 1988, *Dependency syntax: theory and practice*, Albany, NY: State Univ. Press NY.

MOORTGAT 1988, *Categorial investigations*, Dordrecht: Foris.

POLLARD Carl & SAG Ivan A., 1994, *Head-Driven Phrase Structure Grammar*, CSLI series, University of Chicago Press.

ROBINSON Jane J., 1970, "Dependency structures and transformational rules", *Language* **46.2**, 259-85.

ROSS John, 1967, "Contrainst on variables in syntax", Doctoral dissertation, MIT (Published as *Infinite syntax !*, Norwood, N.J.: Ablex, 1986).

TESNIÈRE Lucien, 1934, "Comment construire une syntaxe", *Bull. Fac. Lettres Strasbourg* **7**, 12ème année, 219-229.

TESNIÈRE Lucien, 1959, *Éléments de syntaxe structurale*, Paris: Kliencksieck, 670 p.

VERGNE Jacques, 1994, "A non recursive sentence segmentation applied to parsing of linear complexity in time", *Proc. 1st Int. Conf. on New Methods in Lang. Proc.*, Manchester.

Local Tree Description Grammars

Laura Kallmeyer

Universität Tübingen
lk@sfs.nphil.uni-tuebingen.de

1 Introduction

A lot of interest has recently been paid to constraint-based definitions and extensions of Tree Adjoining Grammars (TAG). Examples are the so-called quasi-trees (see Vijay-Shanker (1992) and Rogers (1994)), D-Tree Grammars (see Rambow et al. (1995)) and Tree Description Grammars (TDG) (see Kallmeyer (1996a,b)). The latter are grammars consisting of a set of formulas denoting trees. TDGs are derivation-based where in each derivation step a conjunction is built of the old formula, a formula of the grammar and additional equivalences between node names of the two formulas. This formalism is more powerful than TAGs. TDGs offer the advantages of MC-TAG (see Joshi (1987a)) and D-Tree Grammars for natural languages, and they allow underspecification. However, the problem is that TDGs might be unnecessarily powerful for natural languages. To solve this problem, in this paper, I will propose local TDGs, a restricted version of TDGs. Local TDGs still have the advantages of TDGs but they are semilinear and therefore more appropriate for natural languages.

First, the notion of semilinearity is defined. Then local TDGs are introduced, and, finally, semilinearity of local Tree Description Languages is proven.

2 Semilinearity

Let N be the set of non-negative integers. For $(a_1, \cdots, a_n), (b_1, \cdots, b_n) \in N^n$ and $m \in N$ we define: $(a_1, \cdots, a_n) + (b_1, \cdots, b_n) := (a_1 + b_1, \cdots, a_n + b_n)$ and $m(a_1, \cdots, a_n) := (ma_1, \cdots, ma_n)$.

For some alphabet $X = \{a_1, \cdots, a_n\}$ with some (arbitrary) fixed order of the elements, a function $p : X^* \to N^n$ is called a Parikh-function, if:
For all $w \in X^*$: $p(w) := (|w|_{a_1}, |w|_{a_2}, \cdots, |w|_{a_n})$, where $|w|_{a_i}$ is the number of occurences of a_i in w. For all $L \subseteq X^*$: $p(L) := \{p(w) | w \in L\}$.

Two strings $x_1, x_2 \in X^*$ are letter equivalent if they contain equal number of occurences of each symbol, i.e. if $p(x_1) = p(x_2)$ for some Parikh-function p. Two languages $L_1, L_2 \subseteq X^*$ are letter equivalent if every element in L_1 is letter equivalent to an element in L_2 and vice-versa, i.e. if $p(L_1) = p(L_2)$ for some Parikh-function p.

Definition 1 *(Semilinearity)*

1. *Let $x_0, x_1, \cdots, x_m, 0 \leq m$ be in N^n. A linear subset of N^n is a set $\{x_0 + n_1x_1 + \cdots + n_mx_m \mid n_i \in N$ for $1 \leq i \leq m\}$.*

2. *The union of finitely many linear subsets of N^n is a semilinear subset of N^n.*
3. *A language $L \subseteq X^*$ is semilinear, if there is a Parikh-function p such that $p(L)$ is a semilinear subset of N^n.*

Proposition 1 *(Parikh-Theorem) Each context free language is semilinear.*

Clearly, each language that is letter equivalent to a semilinear language is semilinear as well. Because of the Parikh-Theorem (proven by Parikh (1966)), this means that for some language L, in order to prove the semilinearity of L, it is sufficient to show that L is letter equivalent to a context free language.

Semilinearity is an important language property because it seems plausible that natural languages are semilinear (see Joshi (1987b) and Vijay-Shanker et al. (1987)). As far as I know, the only example of a possibly non-semilinear phenomenon is case stacking in Old Georgian (see Michaelis and Kracht (1996)). Since it is not clear whether there is really a (theoretically) infinite progression of stacking possible, there is no reason to assume natural languages not to be semilinear, as long as these are the only examples of nonsemilinear phenomena. If natural languages are semilinear, then it is desirable that the languages generated by grammar formalisms intended to capture human language capacity are semilinear as well.

3 Local TDGs

The tree logic used for local TDGs is the same as for TDGs (see Kallmeyer (1996b)). It is similar to the logic proposed by Rogers (1994) for TAGs. The logic is a quantifier-free first order logic with variables K (node names), binary relations $\lhd$ (parent or immediate dominance), $\lhd^*$ (dominance), $\prec$ (linear precedence) and $\approx$ (equality), a symbol δ for the labelling function, sets of constants N and T for the nonterminal and terminal labels, and logical connectives $\neg$, $\wedge$ and $\vee$. Satisfaction is defined with respect to special models (finite labelled trees) and variable assignments. ϕ_1 entails ϕ_2 ($\phi_1 \models \phi_2$) for two formulas ϕ_1, ϕ_2 iff all finite labelled trees satisfying ϕ_1 with respect to an assignment g also satisfy ϕ_2 with respect to g. A sound, complete and decidable notion of syntactic consequence, $\phi_1 \vdash \phi_2$, can be defined for this logic.

In the formulas in TDGs (descriptions) certain subtrees are uniquely described together with dominance relations between these trees. A negation free, disjunction free satisfiable formula ϕ is a *description* if there is at least one $k \in node(\phi)$ ($k \in K$ occuring in ϕ) such that $\phi \vdash k \lhd^* k'$ for all $k' \in node(\phi)$ (k is called *minimal* in ϕ), and if for all k_1, k_2, k_3:

- If $\phi \vdash k_1 \lhd k_2 \wedge k_1 \lhd^* k_3$, then either $\phi \vdash k_1 \approx k_3$ or there is a k_4 with $\phi \vdash k_1 \lhd k_4 \wedge k_4 \lhd^* k_3$.
- If $\phi \vdash k_1 \lhd k_2 \wedge k_1 \lhd k_3$, then either $\phi \vdash k_2 \prec k_3$ or $\phi \vdash k_2 \approx k_3$ or $\phi \vdash k_3 \prec k_2$.

To guarantee that in each derivation step, descriptions with disjoint sets of node names can be chosen, an equivalence relation on $\{(\phi, K_\phi); \phi$ is a description and

$K_\phi \subseteq node(\phi)\}$ is needed: $(\psi_1, K_{\psi_1}) \approx_K (\psi_2, K_{\psi_2})$ iff ψ_1 and ψ_2 only differ in a bijection (variable renaming) $f_K : K \to K$ with $K_{\psi_2} = f_K(K_{\psi_1})$.

A *TDG* is a tuple $G = (N, T, D, \phi_S)$, such that:

1. N and T are pairwise disjoint finite sets, the nonterminals and terminals
2. D is a finite set of equivalence classes $\overline{(\psi, K_\psi)}$ (wrt $\approx_K$), such that for all $(\psi, K_\psi) \in \overline{(\psi, K_\psi)}$, ψ is a description with constants N and T. ψ is called an *elementary description* of G, and each $k \in K_\psi$ is called *marked* in ψ.
3. ϕ_S is a description (with constants N and T), the *start description.*

In a derivation step $\phi_1 \overset{\psi}{\Rightarrow} \phi_2$, the result ϕ_2 is the conjunction of ϕ_1, an elementary ψ and equivalences $k_1 \approx k_2$ with $k_1 \in node(\phi_1)$ and $k_2 \in \{k; k$ minimal in ψ or $k \in K_\psi\}$. The main idea of local TDGs is to restrict the derivation mode such that all $k_1 \in node(\phi_1)$ used for new equivalences occur in one single elementary ψ_d that was added before. Furthermore, each $k_1 \in node(\phi_1)$ can be used but once to introduce a new equivalence. Then the derivation step only depends on ψ_d, and the derivation process can be described by a context-free grammar. Doing this, letter equivalence of local TDLs (the string languages of local TDGs) and context-free languages can be shown, and, consequently, local TDLs are semilinear.

To understand the intuitions behind the definition of local TDGs, it is helpful to have an idea of the semilinearity proof. In this proof, for a given local TDG G_T a letter equivalent context-free grammar G_{CF} is obtained as follows: The nonterminals in G_{CF} describe "states" of elementary descriptions used in the course of a derivation. For a derived description ϕ in the corresponding derivation in G_{CF} there is one nonterminal Z_{ψ_d} for each start or elementary description ψ_d added in the course of the derivation of ϕ. Z_{ψ_d} specifies in which way the names of ψ_d can be used in a new derivation step. For each $k \in node(\psi_d)$, Z_{ψ_d} gives information about whether k has a parent or daughter in ϕ, whether k is minimal or does not dominate any other name in ϕ and whether k is strongly dominated by a name k' such that $\phi \vdash \delta(k') \approx X$ for some label X. (A strong dominance in ϕ is a conjunct $k_1 \lhd^* k_2$ in ϕ that is not entailed by the rest of ϕ, i.e. ϕ without this conjunct. Notation: $\phi \vdash_s k_1 \lhd^* k_2$.)

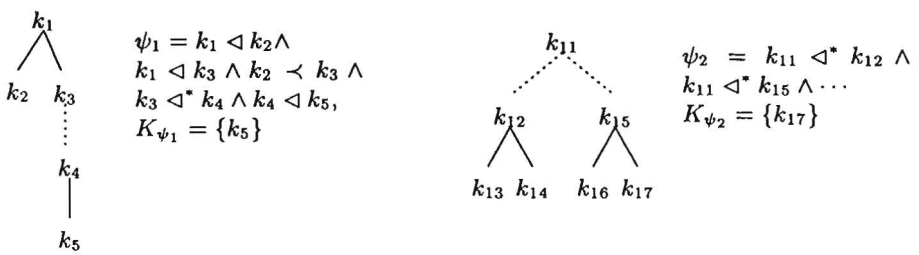


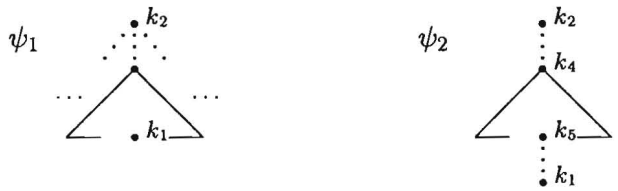
Figure 1: non-local elementary descriptions

For the old description ϕ in a derivation step the following should hold: Only for the elementary ψ_d (in ϕ) used in this derivation step may the state change. Therefore "subtree descriptions" (e.g. the part with k_{12}, k_{13}, k_{14} in ψ_2 in Fig. 1)

must not be inserted into strong dominances $\phi \vdash_s k \lhd^* k'$ with $k' \notin node(\psi_d)$. To guarantee this the form of the descriptions is restricted by defining local descriptions. The descriptions of Fig. 1 for example are not local. If k_{13} or k_{14} was marked, then ψ_2 would be local.

Definition 2 *(Local description) An elementary description ψ in a TDG G is local, if for all $k_1, k_2, k_3 \in node(\psi)$:*

1. *If $\psi \vdash k_1 \approx k_2$, then $k_1 = k_2$.*
2. *If $\psi \vdash_s k_2 \lhd^* k_1$ and $\psi \vdash_s k_3 \lhd^* k_1$, then $k_2 = k_3$.*
3. *If $\psi \vdash_s k_1 \lhd^* k_2$ and $\psi \vdash_s k_1 \lhd^* k_3$, then either $k_2 = k_3$ or: k_1 is minimal or marked in ψ and there are $k_4, k_5 \in K_\psi$ with $\psi \vdash k_2 \lhd^* k_4$ and $\psi \vdash k_3 \lhd^* k_5$.*
4. *If $k_1 \in K(\psi)$ and k_2 is marked or minimal in ψ with $k_1 \neq k_2$ and $\psi \vdash k_2 \lhd^* k_1$, such that there is no further marked name between k_1 and k_2, then:*
   - *There is a $k \in node(\psi)$ with $\psi \vdash_s k_2 \lhd^* k$ and $\psi \vdash k \lhd^* k_1$, and for all $k_3 \in K_\psi$: if $\psi \vdash k \lhd^* k_3$, then $\psi \vdash k_1 \lhd^* k_3$.*
   - *If there are k_4, k_5 with $\psi \vdash k_4 \lhd^* k_5$, $\psi \vdash_s k_2 \lhd^* k_4$ and $\psi \vdash_s k_5 \lhd^* k_1$, then: there is an $X \in N$ with $\psi \vdash \delta(k_i) \approx X$ for all $i \in \{1,2,4,5\}$, and if there is a k with $\psi \vdash_s k_2 \lhd^* k$, then $k = k_4$ holds.*



By this definition two kinds of marked names k_1 with k_2 as next marked or minimal name dominating k_1 are allowed: first (see ψ_1) names k_1 with no $k \neq k_2$ strongly dominating k_1. The second type (see ψ_2) are marked names where underspecification can occur. This is the case, if k_2 strongly dominates some k_4 and k_1 is strongly dominated by some k_5. k_4, k_5, k_1 and k_2 then have the same labels, and there are no other names strongly dominated by k_2. Generally, names k that are not marked or minimal do not strongly dominate more than one name.

A *local TDG* is a TDG $G = (N, T, D, \phi_S)$ where ϕ_S and all elementary descriptions are local. As already mentioned, the main idea of local derivation is to use for new equivalences only names from one elementary ψ_d in the old description ϕ_1, and to use each $k \in node(\phi_1)$ at most once.

Definition 3 *(Local derivation) Let G be a local TDG. For an elementary ψ in G and descriptions ϕ_1, ϕ_2 with $\phi_S \stackrel{*}{\Rightarrow}_l \phi_1$ and $node(\psi) \cap node(\phi_1) = \emptyset$: $\phi_1 \stackrel{\psi}{\Rightarrow}_l \phi_2$ holds (ϕ_2 is locally derived from ϕ_1 in one step), if there is a ψ_d with $\psi_d = \phi_S$ or $\phi_S \stackrel{*}{\Rightarrow}_l \phi \stackrel{\psi_d}{\Rightarrow}_l \phi' \stackrel{*}{\Rightarrow}_l \phi_1$, such that:*

1. *$\phi_2 \vdash \phi_1 \wedge \psi$.*
2. *For all $k_1 \in node(\phi_1), k_2 \in node(\psi)$ such that $\phi_2 \vdash k_1 \approx k_2$, there is a k_1' with $\phi_1 \vdash k_1 \approx k_1'$ and*

(i) $k'_1 \in node(\psi_d)$, *and* k_2 *is marked or minimal in* ψ.

(ii) *For all* k *with* $\phi_1 \vdash k'_1 \approx k$: *either* $k'_1 = k$, *or* $\psi_d \neq \phi_S$ *and* $\phi' \vdash k'_1 \approx k$.

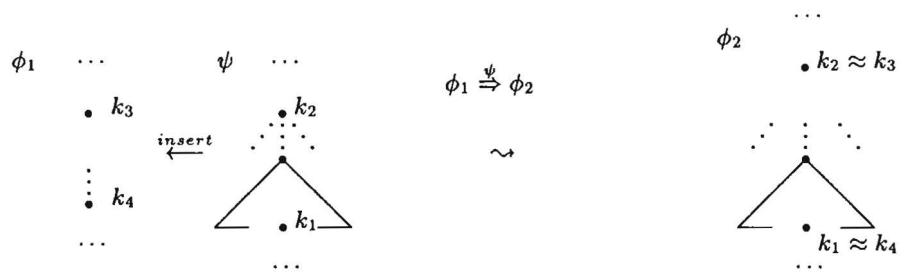
(iii) *If* k_m *is the next marked or minimal name dominating* k_2 *and there are* k'_m, k'_2 *with* $\psi \vdash_s k_m \lhd^* k'_m$ *and* $\psi \vdash k'_m \lhd^* k'_2 \wedge k'_2 \lhd k_2$, *then: There is a* k *with* $\phi_1 \vdash_s k \lhd^* k'_1$ *such that for all* k': *if* $\psi \vdash k'_m \lhd^* k'$, *then* $\phi_2 \vdash k \lhd^* k'$.

(iv) *If there is no* $k_3 \in K_\psi$, $k_2 \neq k_3$, *such that* $\psi \vdash k_2 \lhd^* k_3$, *then either* k'_1 *is a leaf name in* ϕ_1 *or* k_2 *is a leaf name in* ψ. *(*k *is a leaf name in* ϕ *iff for all* k': *If* $\phi \vdash k \lhd^* k'$, *then* $\phi \vdash k \approx k'$.*)*

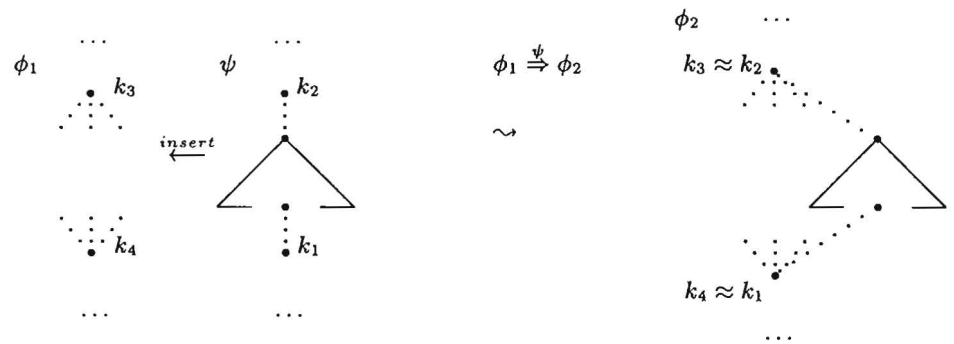
(v) *If there is a* $k_3 \in K_\psi$ *with* $\psi \vdash k_2 \lhd^* k_3$ *and* $k_2 \neq k_3$, *if there is no marked name between* k_2 *and* k_3, *and if there are* k'_2, k'_3 *with* $\psi \vdash_s k_2 \lhd^* k'_2$ *and* $\psi \vdash_s k'_3 \lhd^* k_3$ *and* $\psi \vdash k'_2 \lhd^* k'_3$, *then: If* $k_4 \in node(\psi_d)$ *with* $\phi_2 \vdash k_4 \approx k_3$, *then for all* $k \in node(\phi_1)$: $\phi_1 \not\vdash k'_1 \lhd k \vee k \lhd k_4$.

3. For all ϕ_3 *such that 1. and 2. hold for* ϕ_3: *If* $\phi_2 \vdash \phi_3$, *then* $\phi_3 \vdash \phi_2$.

(i) makes sure that all $k \in node(\phi_1)$ used in one derivation step are from one elementary ψ_d. (ii) says that each name can be used only once for a derivation step. Because of (iii), parent relations in ϕ_2 come from exactly one of the descriptions ϕ_1 or ψ, and everything between two marked or minimal names in ψ must be inserted into one single strong dominance. With (iv) a $k \in K_\psi$ not dominating any other $k' \in K_\psi$ either is a leaf name or it is identified with a leaf name in ϕ_1. Because of *1.* and *3.*, ϕ_2 must entail ψ and ϕ_1, and ϕ_2 must be maximally underspecified.



For $k_1, k_2 \in node(\psi)$ either marked or minimal with no marked names in between and with $\phi_2 \vdash k_4 \approx k_1 \wedge k_3 \approx k_2$ for $k_3, k_4 \in node(\phi_1)$: Either there is no $k \neq k_2$ with $\psi \vdash_s k \lhd^* k_1$. Then the derivation step is as in the preceding figure. Or, if there is such a k, (see *(v)*) the derivation step has the form:



In a local TDG G, $L^l_D(G)$ is the set of descriptions that can be locally derived

from ϕ_S. The tree language is the set of *minimal trees* of these descriptions. A minimal tree of ϕ is a tree that satisfies ϕ such that all parent relations in the tree are already described in ϕ. The set of strings yielded by these trees is the string language.

Local TDGs are still powerful enough to describe $\{a_1^n a_2^n \cdots a_k^n\}$ and copy languages. Local Tree Description Languages (TDL) are a true superset of Tree Adjoining Languages. With local TDGs, as with MC-TAGs, several subtree descriptions can be added simultaneously, and subsertion-like derivation steps as in D-Tree Grammars are possible. Furthermore, in cases of scope ambiguities, underspecified representations can be derived (see Kallmeyer (1996b, 1997)).

4 Semilinearity of local TDLs

Proposition 2 *Local TDLs are letter equivalent to context-free languages.*

Proof (outline): Let $G_T = (N_T, T, D, \phi_S)$ be a local TDG such that without loss of generality for all elementary or start descriptions ϕ and all $k \in node(\phi)$ there is a $X \in N_T \cup T \cup \{\epsilon\}$ with $\phi \vdash \delta(k) \approx X$.
Construction of a letter equivalent context-free grammar $G_{CF} := (N_C, T, P, S)$: The nonterminals are states Z of the form $Z = \phi_Z \wedge \xi_Z$ with: $\phi_Z = \phi_S$ or ϕ_Z elementary in G_T (one representative for each class in D is chosen). ξ_Z is a conjunction of formulas $parent(k)$, $child(k)$, $leaf(k)$, $minimal(k)$, $dom_\uparrow(k, X)$ or $derive(k)$ or their negations with $k \in node(\phi_Z)$ and $X \in N_T$. For each state $Z = \phi_Z \wedge \xi_Z$ for all $k \in node(\phi_Z)$ and all such formulas $\psi = parent(k), \cdots$ either ψ or $\neg\psi$ must occur in ξ_Z.
Additionally N_C contains a start symbol S different from all other nonterminals. Let $Z^\sim = \phi_Z^\sim \wedge \xi_Z^\sim$ be equivalent to one $Z \in N$ ("equivalent" means that Z and $Z^\sim$ only differ in a bijection K). We define: A description ϕ with $\phi_S \stackrel{*}{\Rightarrow}_l \phi$ entails $Z^\sim$, $\phi \models Z^\sim$, as follows:

1. $\phi \models parent(k)$ iff there is a k' such that $\phi \models k' \lhd k$.
2. $\phi \models child(k)$ iff there is a k' such that $\phi \models k \lhd k'$.
3. $\phi \models leaf(k)$ iff k is a leaf name in ϕ.
4. $\phi \models minimal(k)$ iff k is minimal in ϕ.
5. $\phi \models derive(k)$ iff there are ϕ_1, ϕ_2 such that $\phi_S \stackrel{*}{\Rightarrow}_l \phi_1 \Rightarrow_l \phi_2 \stackrel{*}{\Rightarrow}_l \phi$, $k \in node(\phi_1)$ and $\phi_2 \models k \approx k'$ for one $k' \notin node(\phi_1)$.
6. $\phi \models dom_\uparrow(k, X)$ iff there is a k' with $\phi \vdash_s k' \lhd^* k \wedge \delta(k') \approx X$.
7. Apart from this, $\phi_1 \models \phi_2$ is defined as before.

Productions P:

1. If $Z_S \in N$ with $Z_S = \phi_S \wedge \xi_S$ and $\phi_S \models \xi_S$ and if $t_1, \cdots, t_n$ are all occurences of terminals in ϕ_S, then $S \rightarrow t_1 \cdots t_n Z_S \in P$.
2. Let Z and Z' be states for the same elementary or start description, Z_{new} a state for some elementary ψ, and $t_1, \cdots, t_n$ all occurences of terminals in ψ. $Z \rightarrow t_1 \cdots t_n Z' Z_{new} \in P$ iff the following holds:

For all ϕ, $\phi_S \stackrel{*}{\Rightarrow}_l \phi$ entailing a $Z^{\sim} = \phi^{\sim} \wedge \xi^{\sim}$ equivalent to Z: There is a ϕ' with $\phi \stackrel{\psi}{\Rightarrow}_l \phi'$ and $Z'^{\sim} = \phi'^{\sim} \wedge \xi'^{\sim}$ and $Z^{\sim}_{new} = \phi^{\sim}_{new} \wedge \xi^{\sim}_{new}$ equivalent to Z' and Z_{new} such that $\phi' \models Z'^{\sim} \wedge Z^{\sim}_{new}$. Furthermore $\phi^{\sim} = \phi'^{\sim}$ and $\psi = \phi^{\sim}_{new}$ hold and $\phi^{\sim}$ is the elementary ψ_d (see Def. 3) used in this derivation step.

3. For all $Z \in N$, $Z = \phi_Z \wedge \xi_Z$:
   $Z \rightarrow \epsilon \in P$ iff for all k in ϕ_Z: if X is the label of k, then either $parent(k)$ or $dom_{\uparrow}(k, X)$ or $derive(k)$ or $minimal(k)$ is in ξ_Z.

G_{CF} is unique and it is a context-free grammar.
By induction on the length n of the derivation the following can be shown:

$S \stackrel{n+1}{\Rightarrow} w_n$ wrt G_{CF} without applying ϵ-productions, and $Z_1, \cdots Z_n$ are all occurences of nonterminals in w_n
iff there is a derivation $\phi_S \stackrel{n}{\Rightarrow}_l \phi_n$ wrt G_T such that there are pairwise different $Z^{\sim}_1, \cdots, Z^{\sim}_n$ with $Z^{\sim}_i = \phi^{\sim}_i \wedge \xi^{\sim}_i$ equivalent to Z_i, with:

- The elementary or start descriptions that have been used in course of the derivation of ϕ_n, are exactly $\phi^{\sim}_1, \cdots, \phi^{\sim}_n$.
- $\phi_n \models Z^{\sim}_i$ for all $1 \leq i \leq n$.

With the ϵ-productions the following holds for w_n, ϕ_n as above: $w_n \stackrel{*}{\Rightarrow} w'_n$ can be derived by applying only ϵ-productions and $w'_n \in T^*$ iff ϕ_n has a minimal tree.
In general: $\phi_S \stackrel{*}{\Rightarrow}_l \phi$ wrt G_T, ϕ has a minimal tree yielding the string w iff there is a w' letter equivalent to w such that $S \stackrel{*}{\Rightarrow} w'$ wrt G_{CF}.

□

As a corollary local TDLs are semilinear.

5 Conclusion

TDGs have been developed to give a constraint-based TAG-extension that offers the advantages of MC-TAGs and D-Tree Grammars, and to introduce underspecification to TAGs. However, TDGs seem to be unnecessarily powerful for natural languages. For this reason I have presented local TDGs in this paper, a restriction of TDGs that is still much more powerful than TAGs. Local TDGs also have the advantages of MC-TAGs and D-Tree Grammars, and even underspecified representations are still possible in local TDGs (see Kallmeyer (1996b, 1997)). By describing the derivation process by a context-free grammar, I have proven that local TDGs are semilinear, which indicates that they really are an interesting alternative to other formalisms developed for natural language processing.

Acknowledgments

For critical discussions and helpful comments, I would like to thank Tom Cornell and Frank Morawietz.

References

Joshi, A. K.: 1987a, An introduction to Tree Adjoining Grammars, *in* A. Manaster-Ramer (ed.), *Mathematics of Language*, John Benjamins, Amsterdam.

Joshi, A. K.: 1987b, Tree adjoining grammars: How much contextsensitivity is required ro provide reasonable structural descriptions?, *in* D. Dowty, L. Karttunen and A. Zwicky (eds), *Narural Language Parsing*, Cambridge University Press.

Kallmeyer, L.: 1996a, Tree Description Grammars, *in* D. Gibbon (ed.), *Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference*, Mouton de Gruyter, Berlin.

Kallmeyer, L.: 1996b, Underspecification in Tree Description Grammars, *Arbeitspapiere des SFB 340 81*, Universität Tübingen. To appear in: Hans Peter Kolb and Uwe Mönnich (eds.) *The Mathematics of Syntactic Structures.*

Kallmeyer, L.: 1997, A syntax-semantics interface with Synchronous Tree Description Grammars, *Proceedings of Formal Grammar*, Aix en Provence. to appear.

Michaelis, J. and Kracht, M.: 1996, Semilinearity as a syntactic variant. Abstract, *Logical Aspects of Computational Linguistics*, Nancy.

Parikh, R.: 1966, On context free languages, *Jounal of the ACM* **13**, 570–581.

Rambow, O., Vijay-Shanker, K. and Weir, D.: 1995, D-Tree Grammars, *Proceedings of ACL.*

Rogers, J.: 1994, *Studies in the Logic of Trees with Applications to Grammar Formalisms*, PhD thesis, University of Delaware.

Vijay-Shanker, K.: 1992, Using descriptions of trees in a tree adjoining grammar, *Computational Linguistics* **18**(4), 481–517.

Vijay-Shanker, K., Weir, D. J. and Joshi, A. K.: 1987, Characterizing structural descriptions produced by various grammatical formalisms, *Proceedings of ACL*, Stanford.

Dynamic Lambda Calculus

Michael Kohlhase, Susanna Kuschert
Universität des Saarlandes

The goal of this paper is to lay a logical foundation for discourse theories by providing an algebraic foundation of compositional formalisms for discourse semantics as an analogon to the simply typed λ-calculus. Just as that can be specialized to type theory by simply providing a special type for truth values and postulating the quantifiers and connectives as constants with fixed semantics, the proposed dynamic λ-calculus $\mathcal{DLC}$ can be specialized to λ-DRT by essentially the same measures, yielding a much more principled and modular treatment of λ-DRT than before; $\mathcal{DLC}$ is also expected to eventually provide a conceptually simple basis for studying higher-order unification for compositional discourse theories.

Over the past few years, there have been a series of attempts [Zee89, GS90, EK95, Mus96, KKP96, Kus96] to combine the Montagovian type theoretic framework [Mon74] with dynamic approaches, such as DRT [Kam81]. The motivation for these developments is to obtain a general logical framework for discourse semantics that combines compositionality and dynamic binding.

Let us look at an example of compositional semantics construction in λ-DRT which is one of the above formalisms [KKP96, Kus96]. By the use of β-reduction we arrive at a first-order DRT representation of the sentence $\texttt{A}^i$ `man sleeps.` (i denoting an index for anaphoric binding.)

$$(\lambda Q.\ \boxed{\begin{array}{c}U_i\\ man(U_i)\end{array}} \otimes Q(U_i))\ (\lambda X.\ \boxed{sleep(X)}) \longrightarrow^*_\beta \boxed{\begin{array}{c}U_i\\ man(U_i)\end{array}} \otimes \boxed{sleep(U_i)}$$

$$\longrightarrow_\delta \boxed{\begin{array}{c}U_i\\ man(U_i)\\ sleep(U_i)\end{array}}$$

where $\otimes$ is the λ-DRT conjunction operator that intuitively merges two DRSes by unifying the sets of discourse referents and that of conditions.

Unfortunately, the above mentioned unified formalisms have failed so far to duplicate a key aspect of type theory that has lead to interesting linguistic analyses in computational linguistics. Type theory (or higher-order logic) is a two-layered formalism, where the algebraic content (the behaviour of higher-order functions) is neatly packaged into a formalism of its own, namely simply typed λ-calculus; while the logical content (the specific semantics of connectives and quantifiers) is built on top of it. Thus the use of type theory allows us to deal with the complexities of natural language semantics on two distinct levels: the simply typed λ-calculus provides the theory of β-reduction (which is the motor of compositionality) whereas the logical side of semantics is dealt with by a system that is rather like predicate logic. By focussing on known mechanisms for dealing with each of the two subsystems, it has proved possible to use type theoretic techniques for natural language processing systems.

- Higher-order unification [Hue75] solves equations in the simply typed λ-calculus and leads to analyses of ellipsis [DSP91], and focus [GK96, Pul94]. Note that these accounts are inadequate for the treatment of the logical structure, so they make insufficient predictions about quantifiers and connectives.

- First-order automated theorem proving [Fit90] is used to reason about the logical structure of natural language, for presuppositions, and to integrate world knowledge into natural language semantics. Note that these approaches normally cannot capture higher-order aspects of the semantics like compositionality or underspecified (e.g. elliptic-) semantic elements.

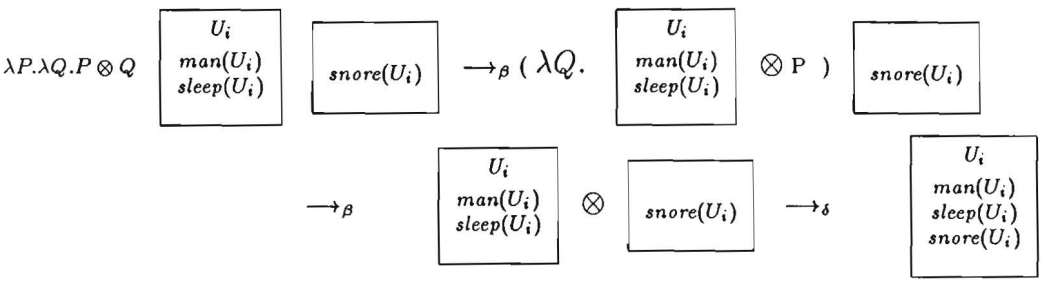
- Only recently, logic formalisms for higher-order theorem proving [Koh95] have appeared that are generalizations of both higher-order unification and automated theorem proving. These can be used to integrate world knowledge into the unification-based approaches [GKvL96].

The goal of this paper is to lay the foundation for analyses like the above by providing an algebraic foundation of compositional formalisms for discourse semantics as an analogon to the

simply typed λ-calculus. Just as that can be specialized to type theory by simply providing a special type o for truth values and postulating the quantifiers and connectives as constants with fixed semantics, the proposed *Dynamic Lambda-Calculus* ($\mathcal{DLC}$) can be specialized to λ-DRT [KKP96] by essentially the same measures, yielding a much more principled and modular treatment of λ-DRT than before.

However, we expect the benefits from a clean separation of the structural and logic parts of compositional discourse semantics will not be restricted to this. In particular, $\mathcal{DLC}$ is expected to serve as the formal basis for the development of higher-order unification algorithms for compositional formalisms for discourse semantics (the topic of a future talk, though), which in turn can be expected to lead to dynamic analyses of ellipses, focus, corrections, ..., corresponding to those discussed above. First experiments with the formal system have shown that these will be more intuitive than those for the static case.

The central theme of DRT is the establishment of anaphoric binding in discourse; λ-DRT, similar to the other above mentioned formalisms, establishes such bindings in a Montagovian-like construction process given coindexation of the pronoun with its antecedent through the syntactical analysis. As an example, consider continuing the above sentence by $\texttt{He}_i$ `snores`. The representation of the discourse may be constructed from the representation of the two sentences by applying them to $\lambda P.\lambda Q.P \otimes Q$ and reducing, arriving at



Note that in this process the free variable in the representation of $\texttt{He}_i$ `snores`, standing for the pronoun $\texttt{he}_i$, has in the course of β-reduction been captured by the declaration of the discourse referent U_i in the representation of the first sentence, which is part of the representation of $\texttt{a}^i$ `man`. Indeed, the capturing of free variables, *the* thing impossible in pure λ-calculus, is the driving force of the establishment of anaphoric binding in λ-DRT.

The proposed formalism $\mathcal{DLC}$ focuses on the interaction of dynamic binding (declaration of discourse referents), function abstraction and function application. It is spelt out by the interaction of two distinct abstraction operators, the well-known λ-, and the dynamic δ-operator. The *capturing of free variables*, motivated above, will be one of the central themes. This leads us to a thorough study of the variables known from λ-calculus and the variables that can thus be captured.

Most of the burden of the interaction of the two kinds of variables, and in particular the respective abstraction mechanisms that go with them, is carried by an elaborate type system that takes into account structural properties of dynamic systems (the *mode*), such as binding power and the accessibility relation. In first experiments, the use of these binding properties have proven very useful in different linguistic applications.

This abstract only gives a first idea about the details of $\mathcal{DLC}$; the full paper can be found via http://coli.uni-sb.de/~kuschert/academic.html.

1 The Syntax of $\mathcal{DLC}$

The key to understanding the character of compositional dynamic formalisms is the identification of functional and dynamic properties and the interaction of these. [KKP96, Kus96] already observe that we can locate two different kinds of variables together with two kinds of abstractions of very different nature: the well-known standard λ-abstraction, used for the compositional construction of representations, and a dynamic abstraction to be called δ-abstraction. Most prominent of their differences is the fact that δ-abstraction may capture free variables on β-reduction which breaks a taboo in standard λ-calculus. In essence this means that the two abstraction operators have quite different notions of scope: Whereas δ-abstraction is boundless with respect to function application,

it is bounded by the notion of accessibility, motivated and defined by some linguistic theory (in particular DRT).

Observing that both these properties are of a structural nature, the central idea of $\mathcal{DLC}$ is to encode the necessary information for variable capture and the accessibility relation in an elaborate type system. Such information must include knowledge about δ-bound variables, which have the power to capture variables, and knowledge about free variables, which are liable to being captured.

As a consequence, $\mathcal{DLC}$'s types include *mode* information for the variables that are visible from the outside of an expression: In addition to the standard λ-calculus types (formed from basic types and functional application) there are types of the form $\Gamma\#\alpha$ for any α, where the mode Γ marks a variable with a '+' if the variable has capturing power, and with a '−' if it is prone to being captured (i.e. if it is a free variable)[1]. As a first example, if the expression **A** has type $X^-, U^+, \Gamma\#\alpha$, we know that **A** is of standard (type theoretic) type α (e.g. if α is the base type o, then **A** is a proposition) and **A** contains at least the free variable X and the δ-bound variable U. λ-bound variables in **A** do not appear in the mode of its type, since they are not visible to the outside of **A**.

The set $\mathcal{V}$ of (typed) variables is partitioned into two distinct sets of variables of inherently different character: the set of *functional variables* $\mathcal{V}^{fun}$ and the set of *dynamic variables* $\mathcal{V}^{dyn}$. In a mode, a positive declaration of a variable U will always overwrite a negative declaration, i.e. $U^+, U^- = U^+ = U^-, U^+$. We use Γ^+ (Γ^-) for the *positive* (*negative*) submode of Γ. As an example, if $\Gamma = U^+, Y^-, P^+$, then $\Gamma^+ = U^+, P^+$, and $\Gamma^- = Y^-$.

Members of the set $\mathcal{T}$ of $\mathcal{DLC}$*-types*, or simply *types*, are constructions of the form $\beta = \Gamma\#\alpha$, and *function types* of the form $\alpha \to \beta \in \mathcal{T}$ (# shall bind stronger than $\to$), where $\alpha, \beta \in \mathcal{T}$, built up from a fixed set $\mathcal{BT}$ of *base types*. We call a type *simple*, iff it is a base type or a functional type. We identify a simple type α with type $\emptyset\#\alpha$. Note that simple types may contain modes on functional level. Furthermore, we identify $\Gamma\#(\Delta\#\alpha)$ with $(\Gamma, \Delta)\#\alpha$. Thus we can always rewrite a type β such that α is simple in $\beta = \Gamma\#\alpha$. In this case we call α the *characteristic type* and Γ the *characteristic mode* of β.

We shall need to allow for the arguments of functional expressions to carry free variables, and therefore define extensions $\overline{\alpha}$ and $\overline{\overline{\alpha}}$ of a type α, given some negative modes Γ_i (i.e. $\Gamma_i^+ = \emptyset$): if α is of base type, then both $\overline{\alpha}$ and $\overline{\overline{\alpha}}$ equal α. If $\alpha = \Delta\#(\alpha_1 \to (\ldots \to \alpha_n))$, then $\overline{\alpha} = \Delta\#(\Gamma_1\#\alpha_1 \to (\Gamma_2\#\alpha_2(\ldots \to [\bigcup_{i=1\ldots(n-1)} \Gamma_i]\#\alpha_n)))$ and $\overline{\overline{\alpha}} = \Delta\#(\Gamma_1\#\alpha_1 \to (\Gamma_2\#\alpha_2 \to (\ldots \to \Gamma_n\#\alpha_n))$, where $\Delta \cap \Gamma_i = \emptyset$. Unless stated otherwise, when we speak of α in the context of a given $\overline{\alpha}$ or $\overline{\overline{\alpha}}$, we assume α to be the minimal type, that is, taking away all negative variables on all levels.

We extract the top-level positive (negative) variables of a type α by α^+ (α^-), defined as $\Gamma^+ \cup \beta^+$ ($\Gamma^- \cup \beta^-$) if $\alpha = \Gamma\#\beta$, and $\alpha^+ = \emptyset$ ($\alpha^- = \emptyset$) if α is a simple type. Further, we shall need a function which collects the variables of all levels of a type, the *binding potential of* α, defined as $\mathcal{BP}(\alpha) = \alpha^+ \cup \alpha^-$, if α is not a functional type and $\mathcal{BP}(\alpha) = \mathcal{BP}(\beta) \cup \mathcal{BP}(\gamma)$, if $\alpha = \beta \to \gamma$. We write the positive (negative) parts of α's binding potential as $\mathcal{BP}^+(\alpha)$ ($\mathcal{BP}^-(\alpha)$). The name *binding potential* reflects that $\mathcal{BP}(\alpha)$ gives full information on which variables contribute to the binding behaviour of an expression which is of type α. The binding behaviour needs to contain both positive and negative variables. Further, we need to consider all functional levels of the type, since they give information on what happens upon function application.

Lastly, we need to define a *substitution of modes in a type* $[\Delta/X]\alpha$ by

$$
\begin{array}{lll}
[\Delta/X]\alpha & = [\Delta/X]\Gamma\#[\Delta/X]\alpha' & \text{if } \alpha = \Gamma\#\alpha' \\
& = [\Delta/X]\beta \to [\Delta/X]\gamma' & \text{if } \alpha = \beta \to \gamma \\
& = \alpha & \text{if } \alpha \text{ base and simple} \\
[\Delta/X]\Gamma & = \Delta, (\Gamma - X^-) & \text{if } X^- \in \Gamma \\
& = [\Delta/X]n, [\Delta/X](\Gamma - n) & \text{for some } n \in N \\
& = \Gamma & \text{else} \\
[\Delta/X]n & = [\Delta/X]n & \text{(i.e. substitutions for placeholders are not resolved)}
\end{array}
$$

Defining the well-formed formulae of $\mathcal{DLC}$, we start from a signature Σ of typed constants, consisting of two disjunct subsets Σ^l and Σ^p (for logical constants and parameters, i.e. the non-

[1] Modes will also include natural numbers acting like de-Bruijn indices; however, their motivation is too involved for the extend of this abstract and may safely be ignored for a first understanding of $\mathcal{DLC}$. Still, we will mention them in the definitions for completeness.

logical constants, respectively). For reasons of minimality, we assume that Σ does not contain any constant functions — these may be constructed from atomic expressions by functional application.

We assume that Σ^l contains at least the operators ∂, ∂^s and ∂^o which will be called *dynamification operators*:

$$\begin{array}{ll} \partial & :(\alpha \to \beta \to \gamma) \to (\Gamma \# \alpha) \to (\Delta \# \beta) \to (\Gamma \cup \Delta) \# \gamma \\ \partial^s & :(\alpha \to \beta \to \gamma) \to (\Gamma \# \alpha) \to (\Delta \# \beta) \to (\Gamma \cup \Delta) \# \gamma \text{ where } \Delta^+ \cup \Gamma^- = \emptyset \\ \partial^o & :(\alpha \to \beta) \to (\Gamma \# \alpha) \to \Gamma \# \beta \end{array}$$

Well-formed formulae will be defined by means of an inference system, where the judgment $\mathbf{A}: \alpha$ holds, iff $\mathbf{A}$ is a formula of type α.

Definition 1.1 (Well-formed Formulae of $\mathcal{DLC}$). The syntactic category of *well-formed formulae* consists of constants, variables, *applications* ($\mathbf{AB}$), and *λ-abstractions* ($\lambda X_\alpha.\mathbf{A}$), *dynamic abstractions* ($\delta U_\alpha.\mathbf{A}$). The inference system for the judgment schema of *well-typedness with respect to some variable context* $\mathcal{A}$, denoted by $\mathcal{A} \vdash \mathbf{A}: \Gamma \# \alpha$, is given by the following schemata[2]:

$$\frac{\Sigma^r(c) = \alpha}{\mathcal{A} \vdash c: \overline{\alpha}}\ \textit{wff:par} \qquad \frac{\Sigma^l(c) = \alpha}{\mathcal{A} \vdash c: \alpha}\ \textit{wff:lconst} \qquad \frac{A \notin \mathcal{BP}(\overline{\overline{\alpha}})}{\mathcal{A}, [A: \overline{\alpha}] \vdash A: A^- \# \overline{\overline{\alpha}}}\ \textit{wff:var}$$

$$\frac{\mathcal{A}, [U: \overline{\overline{\beta}}] \vdash \mathbf{A}: \alpha}{\mathcal{A}, [U: \overline{\overline{\beta}}] \vdash \delta U_\beta.\mathbf{A}: U^+ \# \alpha}\ \textit{wff:dyn}$$

$$\frac{\mathcal{A}, [X: \gamma] \vdash \mathbf{A}: \alpha \quad X \notin \mathbf{Dom}(\Gamma^-) \quad X \notin \mathcal{BP}(\gamma) \quad \gamma = \overline{\overline{\beta}}}{[0/X]\mathcal{I}(\mathcal{A}) \vdash (\lambda X_\beta.\mathbf{A}): \Gamma^- \# \mathcal{I}(\gamma) \to [\Gamma^-/X]\mathcal{I}(\alpha)}\ \textit{wff:abs}$$

$$\frac{\mathcal{A} \vdash \mathbf{A}: \Gamma^- \#(\Delta^- \# \alpha \to \beta) \quad \mathcal{A} \vdash \mathbf{B}: \alpha}{\mathcal{D}([\alpha^-/0]\mathcal{A}) \vdash \mathbf{AB}: \Gamma^- \# \mathcal{D}([\alpha^-/0]\beta)}\ \textit{wff:app}$$

If $\mathcal{A} \vdash \mathbf{A}: \Gamma \# \alpha$ for some $\mathcal{A}$, where α is simple, we shall also write $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma; \Gamma)$. Note that α depends on the choice of $\mathcal{A}$.

Note that the definition of $\mathcal{DLC}$ well-formedness is a simple extension of the well-formedness in λ-calculus, being extended only by the definition of δ-abstraction and the management of modes. Note in particular that a variable is a member of its own mode, and that free variables of a function argument are allowed in through the Γ^- on top level and through the $\overline{\overline{\beta}}$ on functional level. Observe that the top level free variables are free in the result type also exactly if the λ-abstraction is not empty. Apologies for this rather brief exposition come with an example for a $\mathcal{DLC}$ type derivation. $\mathcal{B}, \mathcal{B}'$ and $\mathcal{B}''$ are variable contexts such that $\mathcal{B} = \mathcal{B}' - [U: \ldots] = \mathcal{B}'' - [F: \ldots]$.

$$\cfrac{\cfrac{\mathcal{A}, [U: \overline{\overline{\theta}}], [P: \overline{\overline{\gamma}}] \vdash P: P^- \# \overline{\overline{\gamma}}}{\mathcal{A}, [U: \overline{\overline{\theta}}], [P: \overline{\overline{\gamma}}] \vdash \delta U.P: U^+, P^- \# \overline{\overline{\gamma}}}}{\mathcal{A}, [U: \overline{\overline{\theta}}] \vdash \lambda P.\delta U.P: (\Gamma^- \# \overline{\overline{\gamma}}) \to [\Gamma^-/P]U^+, P^- \# \overline{\overline{\gamma}}}$$

$$\frac{\mathcal{B}', [F: \Delta'^- \# \alpha \to \Delta''^- \# \beta] \vdash F: F^- \#(\Delta'^- \# \alpha \to \Delta''^- \# \beta) \quad \mathcal{B}'', [U: \overline{\overline{\theta}}] \vdash U: U^- \# \overline{\overline{\theta}}}{\mathcal{B}, [U: \alpha], [F: \Delta'^- \# \alpha \to \Delta''^- \# \beta] \vdash F(U): F^-, \Delta''^- \# \beta} \boxed{\begin{array}{l} \Delta'^- \# \alpha = \\ U^- \# \overline{\overline{\theta}} \end{array}}$$

[2] $\mathcal{D}$ and $\mathcal{I}$ are defined as the decrement and increment on natural numbers respectively, i.e. $\mathcal{I}(U^+, Y^-, 1, P^+ \#(\alpha \to \beta) = U^+, Y^-, 2, P^+(\mathcal{I}(\alpha) \to \mathcal{I}(\beta))$, and likewise for $\mathcal{D}()$. For purpose of this abstract and a first understanding of $\mathcal{DLC}$, ignore these as well as the introduction and substitution of numbers in modes.

The latter sub-construction illustrates how both, the type extension (introducing Δ'^- which is matched with U^-) and the negative mode Γ^- of the application rule (being matched with F^-) are used to derive $F(U)$'s type. Now, we derive the type of $\mathbf{A} = \lambda F.(\lambda P.\delta U.P)F(U)$, calling the above subderivations (1) and (2) respectively.

$$\frac{\dfrac{(1)\quad(2)}{\mathcal{B},[U:\alpha],[F:\ldots]\vdash(\lambda P.\delta U.P)F(U):U^+,F^-,\Delta''^-\#\beta}\ \boxed{\begin{array}{l}\Gamma^-\#\bar{\bar{\gamma}}\\ F^-,\Delta''^-\#\beta\end{array}}\ =}{\mathcal{B},[U:\alpha]\vdash\lambda F.(\lambda P.\delta U.P)F(U):(\Xi^-\#(U^-\#\alpha\rightarrow\Delta''^-\#\beta))\rightarrow U^+,\Xi^-,\Delta''^-\#\beta}$$

Observe that Δ''^- is not yet specified in $\mathbf{A}$'s type. This depends on what kind of function will be substituted for F. E.g. if we apply $\mathbf{A}$ to $\lambda X.s(X):\Theta^-\#\epsilon\rightarrow\Theta^-\#o$ given $[X:\epsilon]\in\mathcal{B}$ and some type o, we get $\Delta''^- = U^-$. Alternatively, if we apply $\mathbf{A}$ to $\lambda X.r:(\Theta^-\#\epsilon)\rightarrow o$, then $\Delta''^- = \emptyset$.

In the first of these alternatives, i.e. $\mathbf{A} = (\lambda P.\delta U.P)s(U)$, we may also observe how variable capture is mirrored in the types: here, the Δ''^- in the type of $F(U)$ contains a negative U^- by the specification of F. The positive U^+, which comes from $\lambda P.\delta U.P$, overwrites the U^-. Thus the key to the structural modeling of variable capturing is the overwriting effect of positive variables. Note that capturing works regardless of whether the positive variable occurs in the functor or in the argument.

If the mode of an expression contains no positive variables, we call that expression *static*, and *dynamic* otherwise. In the same spirit, we call $\lambda X.\mathbf{A}$ a *static* or *functional abstraction*, and $\delta U.\mathbf{A}$ a *dynamic abstraction*, since it adds to the degree of dynamicity. Further, for a variable A we distinguish between A being *functionally bound* (bound by a λ-operator) in an expression $\mathbf{A}$, defined as in standard λ-calculus, and A being *dynamically bound* (bound by a δ-operator) in $\mathbf{A}$, if A occurs positively in $\mathbf{A}$'s type. A is *free* in $\mathbf{A}:\alpha$, iff $A\in\alpha^-$, and we write $A\in\mathcal{FV}(\mathbf{A})$.

Substitution (for functional variables) in $\mathcal{DLC}$ is defined very much like its respective notion in λ-calculus, except that capturing of dynamic variables through substitution is perfectly allowed. With $\mathcal{DLC}$-types we may also express λ-calculus substitutability by means of types only: An expression $\mathbf{B}:\beta$ is *substitutable* for a functional variable Y in an expression $\mathbf{A}:\alpha$, iff $\mathcal{BP}(\alpha)\cap\beta^- = \emptyset$. The definition of substitution must be extended by a clause for dynamic expressions, which is trivial though: since U is not a functional variable, we have $[\mathbf{B}/Y]\mathbf{A} = \delta U.([\mathbf{B}/Y]\mathbf{C})$. We note that $\mathcal{DLC}$ substitution is type-preserving.

In $\mathcal{DLC}$, we may use the well-studied β-reduction of type theory for the semantic construction process, in the way suggested by Montague. In a similar way, we also need a so-called δ-reduction, by which means dynamic expressions are constructed from their dynamic constituents. A particular feature of $\mathcal{DLC}$ is, however, that α-renaming known from type theory is not be confined to λ-bound variables alone but extended to δ-bound variables. In fact, α-renaming of dynamically bound variables is a problem for the existing systems for compositional discourse semantics mentioned in the introduction and $\mathcal{DLC}$ evolved from an attempt to understand full α-conversion. Facilitated by the richer type system, we are able to define a joint α-conversion rule for functionally and dynamically abstracted variables (let A and B be either both functional or dynamic variables); the operator C_A^B changes B for A everywhere in $\mathbf{A}$ itself, including the abstractions, *and* also in its type.

$$\frac{\mathcal{A}\vdash\mathbf{D}:\beta\quad A_\gamma\notin\mathcal{BP}(\beta)\quad B_\gamma\notin\mathcal{A}}{(\mathcal{A}\vdash\mathbf{D})=_\alpha([B/A]\mathcal{A}\vdash C_A^B(\mathbf{D}))}$$

The α-rule may be applied if the variable A to be changed in $\mathbf{A}$ does not occur in the binding potential of $\mathbf{A}$, and B is a new variable (not in $\mathcal{A}$). This means that a variable, whether functional or dynamic, can be renamed if its scope is closed off (note that the scope of free variables can be considered boundless), even over functional application, i.e. it does not have an effect on any of $\mathbf{A}$'s arguments. Note that the substitution $[B/A]\mathcal{A}$ is non-empty, if A is a dynamic variable. As usual, we will assume a built-in α-equality, i.e. that expressions are syntactically equal, if they are α-equal.

In addition to the above α-rule and the standard (λ-calculus) β- and η-rules, $\mathcal{DLC}$ also has a δ-rule which defines that $\partial\mathbf{R}(\delta U.\mathbf{A})\mathbf{B}\longrightarrow_\delta\delta U.(\partial\mathbf{RAB})$ and $\partial\mathbf{R}.\mathbf{A}(\delta U.\mathbf{B})\longrightarrow_\delta\delta U.(\partial\mathbf{RAB})$, and if both $\mathbf{A}$ and $\mathbf{B}$'s types are static, $\partial\mathbf{RAB}\longrightarrow_\delta\mathbf{RAB}$. Similarly for ∂^s and ∂^o.

The induced equality shows that ∂ and ∂^s are dynamification operators for binary relations $\mathbf{R}$, a symmetric and an asymmetric one respectively, and that ∂^o is a dynamification operator for unary relations $\mathbf{R}$. We have chosen ∂, ∂^s and ∂^o as primitives for $\mathcal{DLC}$ instead of λ-DRT's $\otimes$-operator (and the asymmetric equivalents of other theories). Indeed, we are convinced that it will be possible to characterize the dynamic operators of other approaches in relation to known binary relations by means of these dynamification operators.

2 The Semantics of $\mathcal{DLC}$

The main focus on the definition of $\mathcal{DLC}$'s semantics will be the capturing of dynamic variables, as before. Here it means that the interpretation of the variable to be captured must be such that it can be mirrored onto the interpretation of the bound variable as a side-effect of function application. This effect shall be modelled by delaying the interpretation of the variable until after function application. A general technique to do this is the use of intensionalization. The underlying idea of $\mathcal{DLC}$'s semantics is to use this technique implicitely and let the interpretation process itself guard *all* dynamic variables (instead of using explicit $^\wedge$ and $^\vee$-operators). λ-bound variables will be interpreted as usual, by an assignment function φ that is added to the interpretation function.

The basis of interpretation of $\mathcal{DLC}$-expressions will be a dynamic pre-structure, a straightforward extension of the well-known pre-structure of Henkin models of standard λ-calculus by a domain for (potentially) dynamic structures.

Definition 2.1 (Dynamic Pre-Structures). Let $\mathcal{D}_\mathcal{T}$ be a typed collection of sets and $\mathcal{I}: \Sigma \longrightarrow \mathcal{D}$ be a typed total function, then we call the triple $\mathcal{A} := (\mathcal{D}, \underline{@}, \mathcal{I})$ a *dynamic pre-structure*, if

1. $\mathcal{D}_{\alpha\to\beta} \subseteq \mathcal{F}(\mathcal{D}_\alpha; \mathcal{D}_\beta)$
2. $\mathcal{D}_{\Gamma\#\alpha} = \mathcal{F}(\mathcal{B}_\Gamma; \mathcal{D}_\alpha)$

where $\mathcal{B}_\Gamma = \bigcup_{\Delta \supseteq \Gamma} \mathcal{F}(\Delta^{dyn}; \mathcal{D})$[3] is the set of *variable assignments* for the mode Γ, also called Γ-*states*.

The function $\mathcal{I}$ must be defined such that[4]

$$\begin{array}{ll} \mathcal{I}(\partial) & = \Lambda\mathcal{R}, \mathcal{A}, \mathcal{B}.\{a^1 \cup b^1 \mapsto \mathcal{R}(a^2, b^2) \mid a \in \mathcal{A}, b \in \mathcal{B}, a^1 \big|\big| b^1\} \\ \mathcal{I}(\partial^s) & = \Lambda\mathcal{R}, \mathcal{A}, \mathcal{B}.\{a^1 \cup b^1 \mapsto \mathcal{R}(a^2, b^2) \mid a \in \mathcal{A}, b \in \mathcal{B}, a^1 \big|\big| b^1\} \\ \mathcal{I}(\partial^o) & = \Lambda\mathcal{R}, \mathcal{A}.\{a^1 \mapsto \mathcal{R}(a^2) \mid a \in \mathcal{A}\} \end{array}$$

and the constant $\underline{@}: (\Gamma\#\alpha \to \beta) \to (\Delta\#\alpha) \to (\Gamma \cup \Delta\#\beta)$ is defined as $\partial@$ where $@$ is the static (standard) application operator.

The collection $\mathcal{D}$ is called the *carrier set* or the *frame of* $\mathcal{A}$, the set $\mathcal{D}_\alpha$ the *universe of type* α, $\underline{@}$ is the *dynamic application operator*, and the function $\mathcal{I}$ is the *interpretation of constants*.

Definition 2.2 (Denotation $\mathcal{I}_\varphi(\mathbf{A})$). Let $\mathcal{A} = (\mathcal{D}, \underline{@}, \mathcal{I})$ be a dynamic pre-structure and φ be a partial assignment function for λ-bound variables. The *denotation* $\mathcal{I}_\varphi(\mathbf{A})$ of a well-formed formula $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma; \Gamma)$ is defined inductively as follows:

1. $\mathcal{I}_\varphi(c) = \mathcal{I}(c)$ for any $c \in \Sigma$
2. $\mathbf{A}$ is a variable: If $\mathbf{A} = X \in \mathcal{V}^{fun}$, then $\mathcal{I}_\varphi(X) = \varphi(X)$,
   if $\mathbf{A} = U \in \mathcal{V}^{dyn}$, then $\mathcal{I}_\varphi(U) = \{t \mapsto t(U) \mid t \in \mathcal{B}_{[U:\alpha]}\}$
3. $\mathcal{I}_\varphi(\delta U.\mathbf{D}) = \mathcal{I}_\varphi(\mathbf{D})\big|\big|_U$[5]
4. $\mathcal{I}_\varphi(\lambda X.\mathbf{D}) = \Lambda\mathcal{A}.\mathcal{I}_{\varphi,[\mathcal{A}/X]}(\mathbf{D})$[6]
5. $\mathcal{I}_\varphi(\mathbf{AB}) = \mathcal{I}_\varphi(\mathbf{A})\underline{@}\mathcal{I}_\varphi(\mathbf{B})$

[3] We write Δ^{dyn} to denote the set of dynamic variables in Δ.

[4] Two partial functions f_1 and f_2 agree, $f_1 \big|\big| f_2$, if for all $X \in \mathbf{Dom}(f_1) \cap \mathbf{Dom}(f_2)$ we have $f_1(X) = f_2(X)$.

For all $s \in \mathbf{Dom}(\mathcal{I}_\varphi(\mathbf{A}))$ we call $\mathcal{I}_\varphi(\mathbf{A})(s)$ the *static meaning* of $\mathbf{A}$ with respect to state s and assignment φ.

From the definition of the variables as described above, everything else is simple. Since the set of Γ-states has already been built up recursively in the course of the interpretation of some expression $\mathbf{A}$, the δ-abstraction has little to add in the interpretation of $\delta U.\mathbf{A}$. Note that U may not occur in $\mathbf{A}$ itself, thus the denotation of $\mathcal{A}$ may include Γ-states which are not defined on U. In this case, these states have to be eliminated since the mode of $\delta U.\mathbf{A}$ does contain U.

Note that the job of dynamic binding on the semantic side is mainly done by the dynamification operators: they dynamify a static operation by coordinating the states of two dynamic objects and applying the static operator on the static meanings belonging to the respective states. This coordination is facilitated by the agreement condition $a^1 \big|\big| b^1$ and the set union $a^1 \cup b^1$.

3 Application to λ-DRT

We introduced $\mathcal{DLC}$ as an algebraic foundation for (existing) compositional formalisms for discourse semantics. Let us demonstrate this for one of such systems, the λ-DRT [KKP96, Kus96]. To arrive at λ-calculus, we only need to fix the set of base types to $\mathcal{BT} = \{e, o\}$ (individuals and truth values), and specify the set Σ^l, the set of logical constants, thus:

symbol	*type*
$\sqsupset_\Gamma$	$(\Gamma\#o) \to (\Gamma^-\#o)$
$\underline{\wedge}_{\Gamma\Delta}$	$(\Gamma\#o) \to (\Delta\#o) \to (\Gamma, \Delta\#o)$
$\underline{\vee}_{\Gamma\Delta}$	$(\Gamma\#o) \to (\Delta\#o) \to (\Gamma^-, \Delta^-\#o)$
$\underline{\Rightarrow}_{\Gamma\Delta}$	$(\Gamma\#o) \to (\Delta\#o) \to ((\Gamma^-, (\Delta^-/\Gamma^+)))\#o$

The types of these constants fully reflect DRT's notion of accessibility which is, in effect, a specification of the dynamic behaviour of certain linguistic constructs.

Note that the formalization in $\mathcal{DLC}$ allows a finer analysis of dynamic objects than in [Kus96], since the mode Γ records the exact degree of dynamicity of a DRS — the λ-DRT type t for DRSes is now expressed by the collection of dynamic types of the form $\Gamma\#o$. This points to an important and useful property of $\mathcal{DLC}$, the merging of DRSes and conditions: these are not inherently different, but DRSes are merely dynamic conditions. For one, this means that the awkward distinction between the two has been dropped. Further, this means that the logical constants now are conglomerations of the respective static and dynamic operator, e.g. $\underline{\wedge}$ unites the static $\wedge$ and the $\otimes$-operator used in [Kus96].

As an example, let us have a look on the type of the representation of a sentence similar to the one quoted in the introduction. The representation of **every man** with its type is $\mathcal{B}, [U : e] \vdash \lambda Q.(\delta U.\mathsf{man}(U) \underline{\Rightarrow} Q(U)) : (\Delta^-\#(U^-\#e \to \Gamma''^-\#o)) \to \Delta^-, (\Gamma''^-/U)\#o$, and **sleeps** is represented by $\mathcal{A} \vdash \lambda X.\mathsf{sleep}(X) : \Gamma^-\#e \to \Gamma^-\#o$. Thus, on application of the two, Δ^- will be instantiated by $\emptyset$, since $\Gamma^-\#\alpha \to \Gamma^-\#o$ matches perfectly on $U^-\#e \to \Gamma''^-\#o$), giving $\Gamma^- = U^- = \Gamma''^-$. With these, the result type $\Delta^-, (\Gamma''^-/U)\#o$ gives $\emptyset\#o$. This is as one would expect since $(\delta U.\mathsf{man}(U)) \underline{\Rightarrow} \mathsf{sleep}(U))$ has no dynamic potential.

The step from $\mathcal{DLC}$'s semantics to a semantics for λ-DRT again is simple. We fix the carrier sets for the basic types to the standard carrier sets for expressions of types e and o, the universe of individuals and the set of truth values respectively. The semantics of the logical constants is summarized below; for $\underline{\wedge}$ we have a simple dynamification of the static $\wedge$, whereas the interpretation of the other constants needs to take into account some notion of quantification.

[5] We define $f\big|\big|_U$ to be the restriction of f to those elements that contain U in their domain, i.e. $f\big|\big|_U = \{s \mapsto f(s) \mid U \in \mathbf{Dom}(s)\}$.

[6] We use Λ for lambda-abstraction in the meta-language with the intuitive meaning.

$$
\begin{array}{lcl}
\mathcal{I}_\varphi(\sqsupset_\Gamma)\underline{@}\mathcal{A} & = & \{t \mapsto r \mid t \in \mathcal{B}_{\Gamma^-}, r = \mathrm{T}, \text{if for all } a^1 \mapsto a^2 \in \mathcal{A} \text{ such that} \\
 & & \quad a^1|_{-\Gamma+} = t \text{ we have } a^2 = \mathrm{F}, \text{ else } r = \mathrm{F}\} \\
\mathcal{I}_\varphi(\underline{\wedge}_\Gamma\Delta)\underline{@}\mathcal{A}\underline{@}\mathcal{B} & = & \partial\underline{@} \wedge \underline{@}\mathcal{A}\underline{@}\mathcal{B} \\
\mathcal{I}_\varphi(\underline{\vee}_\Gamma\Delta)\underline{@}\mathcal{A}\underline{@}\mathcal{B} & = & \{a^1|_{\Gamma^-} \cup b^1|_{\Delta^-} \mapsto a^2 \vee b^2 \mid a^1 \mapsto a^2 \in \mathcal{A}, b^1 \mapsto b^2 \in \mathcal{B}, a^1 \big|\big| b^1\} \\
\mathcal{I}_\varphi(\underline{\Rightarrow}_\Gamma\Delta)\underline{@}\mathcal{A}\underline{@}\mathcal{B} & = & \{t \mapsto r \mid t \in \mathcal{B}_{\Gamma^-,(\Delta^-/\Gamma^+)}, r = \mathrm{T} \text{ if for all } a^1 \mapsto \mathrm{T} \in \mathcal{A} \\
 & & \quad \text{such that } \mathbf{Dom}(a^1|_{-\Gamma+}) \subseteq \mathbf{Dom}(t) \text{ there} \\
 & & \quad \text{exists a } b^1 \mapsto \mathrm{T} \in \mathcal{B} \text{ such that } a^1 \big|\big| b^1|_{-\Delta+} \text{ and} \\
 & & \quad a^1|_{-\Gamma+} \cup b^1|_{-\Delta+} = t, \text{ else } r = \mathrm{F}\}
\end{array}
$$

4 Conclusion

We are convinced that $\mathcal{DLC}$ with its new typing system constitutes a powerful algebraic basis for a whole class of logical systems combining functional and dynamic logics and that its further study may reveal more of the properties of the interplay of their features. In particular, we hope that in the same way as the types guide the higher order unification of standard λ-calculus, this type system may be useful for dynamic higher order unification.

First experiments have been done to use $\mathcal{DLC}$ instead of the static λ-calculus for applications of natural language understanding. It turned out that the type information does indeed improve the processing power. Further work to be done abounds.

References

[DSP91] Mary Dalrymple, Stuart Shieber, and Fernando Pereira. Ellipsis and higher-order unification. *Linguistics and Philosophy*, 14:399–452, 1991.

[EK95] Jan van Eijck and Hans Kamp. Representing discourse in context. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*. Elsevier Science B.V., 1995.

[Fit90] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer Verlag, 1990.

[GK96] Claire Gardent and Michael Kohlhase. Focus and higher–order unification. In *Proceedings of the 16th International Conference on Computational Linguistics*. Copenhagen, 1996.

[GKvL96] Claire Gardent, Michael Kohlhase, and Noor van Leusen. Corrections and Higher-Order Unification. In *Proceedings of KONVENS96*, pages 268–279. De Gruyter, Bielefeld, Germany, 1996.

[GS90] J. Groenendijk and M. Stokhof. Dynamic Montague Grammar. In L. Kálmán and L. Pólos, editors, *Papers from the Second Symposium on Logic and Language*, pages 3 – 48. Budapest, Akadémiai Kiadó, 1990.

[Hue75] Gérard P. Huet. An unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[Kam81] H. Kamp. A theory of truth and semantic representation. In J. Groenendijk, Th. Janssen, and M. Stokhof, editors, *Formal Methods in the Study of Language*, pages 277 – 322. Mathematisch Centrum Tracts, Amsterdam, 1981.

[KKP96] Michael Kohlhase, Susanna Kuschert, and Manfred Pinkal. A type-theoretic semantics for λ-DRT. In P. Dekker and M. Stokhof, editors, *Proceedings of the 10th Amsterdam Colloquium*, pages 479–498. ILLC, Amsterdam, 1996.

[Koh95] Michael Kohlhase. Higher-order tableaux. In *Proceedings of the Tableau Workshop*, pages 294–309, Koblenz, Germany, 1995.

[Kus96] Susanna Kuschert. *Higher Order Dynamics: Relating operational and denotational semantics for λ-DRT*. CLAUS-Report 84, Universität des Saarlandes, 1996.

[Mon74] R. Montague. The proper treatment of quantification in ordinary English. In R. Montague, editor, *Formal Philosophy. Selected Papers*. Yale University Press, New Haven, 1974.

[Mus96] R. Muskens. Combining Montague semantics and discourse representation. *Linguistics and Philosophy*, 14:143 – 186, 1996.

[Pul94] Stephen G. Pulman. Higher order unification and the interpretation of focus. Technical Report CRC-049, SRI Cambridge, UK, 1994.

[Zee89] H. Zeevat. A compositional approach to DRT. *Linguistics and Philosophy*, 12:95–131, 1989.

Restructuring in Romance and Tree Adjoining Grammar[1]

Seth Kulick
Institute for Research in Cognitive Science
University of Pennsylvania

1 Introduction

Much of the investigation of Tree Adjoining Grammar (TAG) as a formalism for natural language ([KJ85], [Fra92]) has focused on determining which members of the TAG family of formalisms are necessary to handle various natural language phenomena. Basic TAG allows one tree to adjoin (or substitute) into another tree. [Wei88] proposed a family of multi-component (MCTAG) extensions to basic TAG, which allow the grammar to consist of tree sets and not just single trees. Tree-local MCTAG requires that all members of a tree set adjoin into distinct nodes of a single elementary tree, and set-local allows trees from one multi-component set to adjoin into distinct nodes of any of the trees from another multi-component set. Set-local MCTAG is known to have a greater generative capacity ([Wei88]).

In this paper I consider how adequate TAG is to handle clitic climbing in Romance languages. [Ble94] has previously argued that tree-local MCTAG is inadequate to handle clitic climbing in Romance, and that set-local MCTAG is required. I argue in this paper against that view, and maintain that tree-local MCTAG is sufficient to handle clitic climbing and related constructions, commonly grouped together as "restructuring" constructions.

2 A Brief Look at Restructuring in Romance

"Restructuring" in Romance[2] refers to constructions in which normally clausal-bound operations appear to take place across a clausal boundary. One such construction is clitic placement[3]. An object clitic[4] usually appears on the verb of the clause that it is associated with, and in most cases cannot appear on the verb of a higher clause[5]:

(1) a. Luis insistió en comer las manzanas amarillas
Luis insisted on eating the yellow apples

b. Luis insistió en comer*las*

c. * Luis *las* insistió en comer

But for a certain class of verbs, commonly called "trigger" verbs following the [AP83] usage, clitics can be placed higher, called "clitic climbing". As (2c) shows, an object clitic from the lower infinitival can appear on the higher verb, if that verb if *querer*, in contrast to *insistir* in (1c). As (3) shows, a clitic can even climb over two trigger verbs, where *tratar de* is also a trigger verb.

(2) a. Luis quiere comer las manzanas amarillas
Luis wants to eat the yellow apples

b. Luis quiere comer*las*

[1] I would like to thank Tilman Becker, Tonia Bleam, David Embick, Robert Frank, Heidi Harley, Beth Ann Hockey, Aravind Joshi, Brian Kinstler, Anthony Kroch, Jeff Lidz, Miriam Meyerhoff, and Owen Rambow for valuable conversation and advice. I would also like to thank Carmen Rio Rey and Marisel for their native speaker judgements. This work was supported by NSF grant SBR8920230 and ARO grant DAAH04-94-G-0426.

[2] All the following examples, unless otherwise noted, are in Spanish, which I will use as representative of Romance in general, although there are some differences which will be pointed out.

[3] Two others are the long reflexive passive, and long *tough*-movement, which I cannot discuss here for reasons of space.

[4] A clitic is an unstressed pronominal item associated with an argument of a verb. I will not discuss here the various arguments concerning whether they are are base-generated as inflectional affixes or result from movement of an argument, since under either view the same locality conditions must be accounted for, and that is the main issue of concern.

[5] These examples are taken from [AP83]. In Spanish, clitics appear after a nonfinite verb, and before a finite verb.

c. Luis *las* quiere comer

(3) a. Luis quiere tratar de comer*las*
Luis wants to try to eat them

b. Luis *las* quiere tratar de comer

Both trigger verbs (*quiere*, *tratar de*), are usually analyzed as subject-control verbs, but some trigger verbs can also be apparent object-control verbs, such as *permitir*. (4) shows that both the clitic *te* and *lo* can be moved up to *quiere*. Some other verbs in this class are *mandar* (command), *ordenar* (order), *sugerir* (suggest), *aconsejar* (advise), and *enseñar* (teach)[6]. One other class of trigger verbs (5) that I will discuss consists of just two - *hacer* (make) and *dejar* (let), commonly referred to as causatives[7] [8].

(4) a. Mari quiere permitir*te* ver*lo*
Mari wants to permit you to see it

b. Mari quiere permitir*telo* ver

c. Mari *telo* quiere permitir ver

(5) a. El *me* hizo decir*lo*

b. El *me lo* hizo decir
He me it made to say
He made me say it

3 Clitic Climbing and TAG

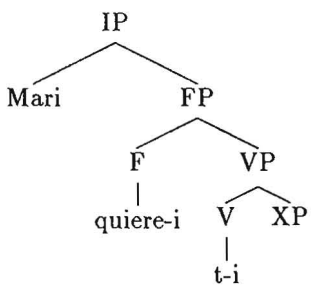
Example (4c) was used as a crucial case in [Ble94]'s argument that tree-local MCTAG was insufficient to handle clitic-climbing. To illustrate the argument, consider first a simpler case such as (6).

(6) a. Mari quiere ver*lo*

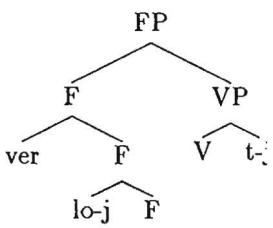
b. Mari *lo* quiere ver
Mari wants to see it

(6a), without clitic climbing, would be derived by (7b) substituting into the XP node of (7a)[9]. [Ble94] analyzes the case with clitic climbing by splitting the tree for a clause from which a clitic climbs into a multi-component set, as in (8ab), so that (6b) would be derived by (8b) substituting into (7a) while (8a) adjoins into (7a) at F.

(7) (a)



(b)



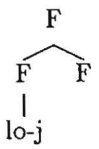
[6] It should be noted that judgements differ on how acceptable clitic climbing is with these verbs (e.g., [Bor88], [Luj80], [Suñ80], [AR75]). Also note that the object of *permitir* has dative case, although it can't be seen here in the clitic form, but it can if it was a full NP. Thus these verbs are usually considered as taking a dative NP argument which controls the PRO in an infinitival complement. Spanish is unique, as far as I know, among modern Romance in allowing clitic climbing with this class of verbs.

[7] For many speakers, clitic climbing with *hacer* is obligatory, and (5a) would be ungrammatical.

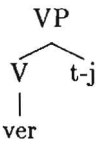
[8] One other class of trigger verbs, that I will not discuss here, is the perception verbs. The causatives and perception verb cases of clitic climbing are sometimes referred to as Clause Union, with the other trigger verbs grouped together as Clause Reduction. Restructuring is sometimes used to refer only the latter group of verbs. I am using it as a general cover term here.

[9] XP is taken to mean that (7a) can take either an FP or VP complement, where FP is a functional projection.

(8) (a)

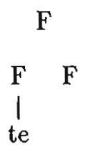


(b)

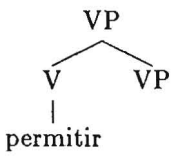


This much requires just tree-local MCTAG[10]. But now consider a sentence with two trigger verbs, such as (4c). This will require another multi-component tree set, for *permitir*, parallel to that of *ver*, as in (9ab).

(9) (a)



(b)



The derivation takes place by (8ab) adjoining and substituting into (9ab), respectively, and then the two complex components adjoin and substitute into (7). This is a set-local, not tree-local MCTAG derivation.

4 Clitic Climbing - Another Approach

The trigger verbs can be broken down into two basic categories, depending on whether the subject of the complement must be coreferential with that of the trigger verb. I will refer to the case of when such coreference is necessary as the "auxiliary-like" category. The second category, without the coreference, includes the *permitir* class and the causatives.

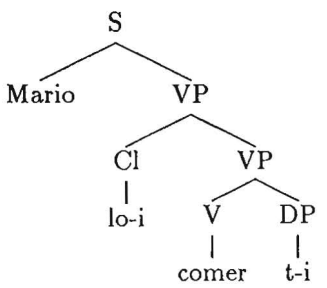
4.1 Auxiliary-like

Consider for the moment just the first category, and a sentence like (2c), repeated here:

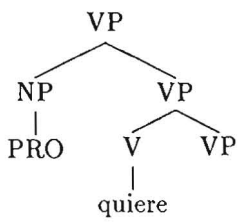
(10) Luis *las* quiere comer

What must be the analysis if set-local MCTAG is excluded? Assuming that *las* must be in the same tree as *comer*, then *quiere* must be adjoining in between them[11]. The trees that would be used are shown in (11), and (11b) would adjoin into (11a) at the lower VP node[12].

(11) (a)



(b)



[10] It should be noted, though, that this is not the usual definition of tree-local MCTAG, in which both components *adjoin* into another tree. Here, one component adjoins, while the other substitutes in. Leaving aside the formal issues, this seems to me like a not-all-together innocuous method of derivation. For example, one could easily then have a raising-to-object derivation for *I believe John to be a liar*, in which a lower clause is split into a multi-component set, *John* and *to be a liar*, with a higher clause of *I believe S*, and *to be a liar* substitutes into the S node of the *believe* tree, while *John* adjoins (or substitutes) into the *believe* tree. Whether or not this is desirable, it's important to note that pure TAG's formal inability to accomplish raising-to-object was one of the original arguments in favor of it [KJ85].

[11] Another possibility is that *Luis* and *quiere* can multi-component adjoin around *las*, but I put that aside here. In any case, it certainly does not affect my argument that set-local MCTAG is not needed. Also, if a sentence with two intervening trigger verbs was used, as in (3c) *Luis las quiere tratar de comer*, then even with tree-local MCTAG, *tratar* would have to adjoin in by itself.

[12] And there must be some type of argument merger/unification happening between *Mario* and PRO.

The proposed analysis handles the "auxiliary-like" trigger verbs by adjoining them almost as if they actually were auxiliaries or raising verbs. That is, it claims that that they have a subcategorization in which they are more "defective" and so can adjoin in a manner not usually done. In fact, not only does this analysis allow TAG to handle clitic climbing, but it is well supported by the linguistic facts, as I will now briefly discuss.

First, a raising or auxiliary analysis is appropriate for many of the verbs in this category. Most of the verbs in this category are either modals[13] like *deber* (must) (12a), aspectuals like *comenzar a* (start) (12b), or motion verbs like *volver a*. The latter is an interesting case, because, as [Str81] points out, while it can mean either "again" or "return (in order to)" when there is no clitic climbing, as in (13a), it can only have the aspectual reading when there is clitic climbing, in (13b).

(12) a. *La* debió comprar en 1950
She must have bought it in 1950

b. *Lo* comenzó a escribir
She started to write it [Suñ80]

(13) a. Ana volvió a empezar a copiar*la*
Ana again began to copy it *or*
Ana returned (in order) to begin to copy it

b. Ana *la* volvió a empezar a copiar
Ana again began to copy it
*Ana returned (in order) to begin to copy it [Str81]

[Luj80] and [Pic85] argue for Spanish and Catalan respectively that the aspectuals do not impose any selectional restrictions on the subject, and could be treated as raising verbs[14] However, it is apparently the case that in at least some dialects of Italian, these aspectuals do impose some selectional restrictions. Whether root modals assign a theta-role to its subject has been a matter of debate, but they are commonly assumed to not be raising verbs[15], but it is certainly not unreasonable, in TAG terms, to adjoin them as auxiliaries.

So a raising or auxiliary analysis is already appropriate for most of the verbs in this category. However, there do seem to be some verbs in this category that are subject-control verbs, in particular *tratar de* (try), and *querer* (want). However, there is a good amount of evidence[16] (e.g., [Nap81], [Ros90], [Pic85]) indicating that, as [Nap81] concluded in her thorough analysis of clitic climbing in Italian, a trigger verb is limited in interpretation when clitic climbing takes place as compared to when it doesn't, and that "these limitations basically involve a kind of weakening or bleaching of the lexical value of the verb."

I will only mention here one of many pieces of data, in this case from Catalan ([Pic90]). In Catalan (and Spanish), a common analysis is that plural null pronominals may be interpreted as arbitrary in reference if they are external arguments of a predicate, although that is not true for an internal argument, as shown in (14) for Catalan. However, in (15) the null subject is interpreted as if it were the internal argument of *passar*, not the external argument of *want*. In other cases a control, non-trigger verb with an embedded *passar* clause would allow the arbitrary reading. So under the TAG view argued for here, in (15), *pro* is acting as the internal argument of *passar* because it *is* the internal argument of *passar*, and *volen* is simply adjoining in[17].

(14) a. Sembla que *pro* hi passen
seems that there pass by
It seems that they are passing by there
*It seems that someone is passing by there

[13] Unlike in English, modal verbs in Romance are not distinguished from other verbs in terms of lacking any verbal morphology.

[14] [Pic85] does not actually treat them as raising verbs, for reasons that are irrelevant here.

[15] Although epistemic modals perhaps are, which I won't discuss here.

[16] A good amount, but certainly not sufficient. Much of this evidence requires further checking and examination of a wider range of cases than has been discussed in the literature.

[17] In the TAG tree for *passar*, *pro* would be moved from the object position to subject position, but that is a move internal to an elementary tree and is not a problem.

b. Em penso que *pro* m'enreden
I think that me-are-fooling
I think that they are fooling me
I think that someone is fooling me

(15) Sembla que *pro* hi volen passar
seems that there want to pass by
It seems that they want to pass by there
*It seems that someone wants to pass by the

It's worth noting in this connection that "want" is a modal in German and that [Mey97] argues that for Bislama, a Melanesian creole, *wantem* (want) and *traem* (try) can participate in an auxiliary-like manner in constructions that are similar to complex predicates[18].

Note also that since the derivation of a sentence such as (3c) requires that *quiere* adjoins into a tree for *Luis las comer*, then the final subject[19] must be the same as the subject of the embedded clause. It is formally impossible for it to be otherwise. This of course rules out object-control verbs as trigger verbs, and in fact this is in general true[20]. This appears to be a problem for verbs like *permitir*, to which we now turn.

4.2 The *permitir* class and the causatives

The causatives are an interesting case, since they exhibit monoclausal behavior, in a number of ways, aside from clitic climbing, most notably in Case marking. I adopt the approach here, following recent work on the phrase structure of causatives, that the causative verb is a "spell-out" of a light verb that is associated with the lower verb, and in TAG terms that would mean that the causative and its complement are in fact part of one TAG tree. [SH88] argue, in a TAG framework, that there is strong evidence for this from Italian, since the passive can be constructed with the complement object promoted to matrix subject position, as in (16), and so the causative+complement verb forms a complex item in the lexicon over which the passive can operate[21].

(16) a. Questo libro e stato fatto leggere a tutti gli studenti
this book is been made read to all the students
This book was made to be read by all the students

[SH88] argue that since other Romance languages do not permit this long passive, they have a true bi-clausal structure, unlike Italian, which must be monoclausal. However, I will instead extend the analysis of Italian to the causatives for the Romance languages in general, and assume that the long passive gets ruled out for other reasons. So if it assumed that in a sentence like (5) that *hacer* and *decir* are in the same TAG tree, then there is obviously no problem in handling the clitic movement of *lo* without extending TAG.

As for the *permitir* verbs, I analyze them as causatives[22], in which *permitir* is also analyzed as the spell-out of the causation in the TAG tree, leaving aside the details of this for now. Aside from the suspicious semantic nature of these verbs, support for this analysis is given by the fact that in some languages, "teach", one of the *permitir* class of verbs, as *enseñar*, is in fact a morphological causative, meaning "cause to learn", as in the Polynesian language Mōari (whakaako : whaka- 'make; cause' and ako 'learn').[23] [24]

[18] I thank Miriam Meyerhoff for this reference and for a discussion of the relevant properties.

[19] By this mean only the item that appears as the subject in the final derived tree. I am not referring to the technical Relational Grammar use of "final".

[20] [Kay89], in his analysis, specifically tries to rule out object-control verbs.

[21] They suggest that *fare* and the complement must form a "lexically derived complex verb". For my purposes here, it is only necessary that *fare* and the complement verb are co-anchors of a TAG tree. Under a TAG variant such as that of [Ram94], such a long passive would be possible by performing the passive after sufficient incremental generation of the proper tree, but the tree manipulation required by this analysis is beyond the power of TAG.

[22] Thanks to Tony Kroch for this suggestion. [Kay89] also suggests that they be considered as "covert causatives"

[23] Thanks to Heidi Harley and Miriam Meyeroff for pointing this out to me.

[24] There are some differences between *hacer* and *dejar* and the *permitir* verbs ([Bor88], [Moo91]). I won't discuss these differences here, but they do not seem significant enough to cause a problem. Also, it should be noted that even the uncontroversial causatives *hacer* and *dejar* do not act identically. [Bor88] also gives a partially-unified analysis of the *permitir* verbs and the causatives, by treating the latter as object-control verbs under one subcategorization.

This type of analysis, which solves the problem of the verbs in this category by increasing the size of the elementary trees, has always been an option in TAG, of course[25]. This is a dangerous option, though, since it can easily become a hack that overcomes any problem by making bigger trees, while losing the explanatory power of the locality of other cases. In the case of the causatives, though, there is good reason to take this approach, since it is limited to a small number of verbs, and to some extent this analysis is even forced by the empirical properties of the causatives[26].

5 Some Consequences

5.1 Revisiting the Problematic Case

Now consider again the derivation of (4c). Although I won't go into detail on the phrase structure of the trees, it can be easily derived, without needing set-local MCTAG, or even tree-local MCTAG. One initial tree will consist of *Mari telo permitir ver*, where *permitir* is the spell out of the causation, and *permitir* and *ver* are the co-anchors of the tree. Then *quiere* simply adjoins in.

5.2 Embedding of Causatives

Since the verbs likes *querer* simply adjoin in, there is no formal limit on how far the clitic can climb over verbs such as these, those in the "auxiliary-like" category. The same is not true for the causatives. If clitic climbing is handled by movement within one tree, what happens when one causative is on top of the other? This is currently being investigated, but current indications are that in Spanish, at least, a clitic cannot climb over two *permitir* verbs, and the causatives may perhaps admit that only marginally. This is a complex matter and cannot be discussed here in full, but for example, (17a) shows *la* having climbed from *comprar* to *permitir*. As (17b) shows, it cannot climb higher to *ordenar*[27]

(17) a. El doctor *le* ordenó permitir*la* comprar a Juan
the doctor her ordered to-permit-it to-buy Juan
The doctor ordered her to permit Juan to buy it

b. * El doctor *sela* ordenó permitir comprar a Juan

5.3 Constraints on Multiple Clitic Movement

It has long been recognized that two clitics associated with a verb cannot be split[28], what [AP83] called the "multiple clitic constraint" (MCC):

(18) a. Puedo mandár*tela*
I can send it to you

b. * *Te* puedo mandár*la*

c. *Te la* puedo mandár

Under the analysis proposed here, all that needs to be said is that argument clitics belonging to the same verb are together on the same node in the elementary tree for that verb, a reasonable assumption. Then they obviously cannot separate for clitic climbing, as in (18b), because there is no such "climbing". The clitics *tela* never move, they simply get stretched away from the lower verb, and they cannot separate.

[25] The earliest TAG work [KJ85] in fact proposed allowing another level of embedding in an elementary tree to handle certain constructions in English and Italian.

[26] A similar TAG analysis for French causatives has been suggested by Anne Abeillé. [Moo91] refers to some unpublished work by C. Rosen in a Relational Grammar framework that makes the same basic distinction between the two sets of trigger verbs as is done here, in which she classifies them as "auxiliary triggers" and "serial triggers".

[27] when *le* and *la* are together on a verb, they appear as *sela*.

[28] [Kay89] claims that this is not so for some nonstandard dialects of Italian and Spanish. I leave this aside for now.

A more interesting case of a constraint on clitic movement has to do with an object clitic "climbing" to one of the *permitir* verbs, what [AP83] called "intersecting clitic climbing" and what [Ble94] grouped together with the MCC as "bandwagon effects". Essentially, if a clitic in the embedded clause of *permitir* climbs to *permitir*, then it and the clitic associated with *permitir* are stuck together:

(19) a. Mari quiere permitir*te* ver*lo*
Mari wants to permit you to see it
b. Mari *te* quiere permitir ver*lo*
c. Mari quiere permitir*telo* ver
d. Mari *te lo* quiere permitir ver
e. * Mari *te* quiere permitir*lo* ver
f. * Mari *lo* quiere permitir*te* ver

[Ble94] derived this constraint from the use of set-locality. If *te* and *lo* both climb, then the tree sets in (8ab) and (9ab) would both be used, and (8a) would have to adjoin into (9a), thus ensuring that the clitics have to stay together, ruling out (19e). (19f) is ruled out because if (8a) adjoined into (7a), while (8b) substituted into a tree for *permitirte*, which then substituted into (7a), set-locality would be violated.

Under the approach here, it would have to be said that if *lo* moves internally in the *permitir-ver* tree, then it appears on the same node as *te*. Then, again, since *quiere* is simply adjoining in, the clitics cannot be separated. Of course, the clitics do not have to be on the same node in the *permitir-ver* tree, since *te* can climb by itself, while *lo* doesn't have to climb at all. But if *lo* does climb internally to the tree, it must be on the same node as *te*, which must be a constraint stated locally on the TAG tree, since otherwise (19ef) could be derived. This is perhaps one area in which [Ble94]'s analysis has an advantage over the one here, since in her analysis it is argued that the facts follow from the set-locality of the derivation.

However, it may be the case that it is actually desirable to treat this as a local constraint. As [Moo91] points out, there are cases in which the clitics of the lowest clause must stay lower, such as when the downstairs verb is ditransitive:

(20) a. *Te* permitió mandár*mela*
S/he permitted you to send it to me
b. * *Te me la* permitió mandár

The obvious thing to say is that a verb (*permitir*) cannot have two dative clitics on it, such as *te* and *me*. Under the analysis here, this would be a constraint that rules out (20) by simply prohibiting such a situation in an elementary tree - such trees are filtered out before any derivation begins. For [Ble94], since *permitir* can take a reduced complement, and since both clitics for *mandar* can climb in other situations (e.g., (18c)), this would have to be a constraint during the derivation (as well as on elementary trees, of course) that a verb cannot have two dative clitics, a situation that arises as a result of movement simulated by the sort of MCTAG system that [Ble94] uses (see footnote 10). Although this is very far from a definitive argument, it does at least suggest that there are some advantages to the analysis here, with regard to the behavior of multiple clitics.

6 Conclusion

I have argued that set-local MCTAG is not needed to handle restructuring in Romance. In fact, even tree-local MCTAG has not been used. Although there are many questions remaining over details of phrase structure, it seems to be a fairly successful approach. A crucial area of investigation is to further examine the properties of doubled verbs of the *permitir* class or the causatives, and to extend the analysis to the perception verbs, keeping in mind the various similarities and differences between these classes. Finally, given the similarities between long distance scrambling (LDS) in German and Restructuring in Romance ([Sab95]), I hope to be able to extend this analysis to handle LDS, which has also been shown to be a problem for TAG.

References

[AP83] Aissen and Perlmutter. Clause reduction in spanish. In *Studies in Relational Grammar*. University of Chicago Press, 1983.

[AR75] Judith Aissen and Alberto M. Rivas. The proper formulation of the spurious-*se* rule in Spanish. In *Proceedings of the First Annual Meeting of the Berkeley Linguistics Society*, pages 1–15, 1975.

[Ble94] Tonia Bleam. Clitic climbing and the power of tree adjoining grammar. In *Symposium on Tree Adjoining Grammar*, 1994. To Appear.

[Bor88] Ivonne Bordelois. Causatives: From lexicon to syntax. *Natural Language and Linguistic Theory*, 6:57–93, 1988.

[Fra92] Robert Frank. *Syntactic Locality and Tree Adjoining Grammar: Grammatical, Acquisition and Processing Perpectives*. PhD thesis, University of Pennsylvania, 1992.

[Kay89] Richard S. Kayne. Null subjects and clitic climbing. In Osvaldo Jaeggli and Kenneth J. Safir, editors, *The Null Subject Parameter*, pages 239–261. Kluwer, Dordrecht, 1989.

[KJ85] Anthony Kroch and Aravind K. Joshi. The Linguistic Relevance of Tree Adjoining Grammars. Technical Report MS-CIS-85-16, University of Pennsylvania, 1985.

[Luj80] Marta Luján. Clitic promotion and mood in Spanish verbal complements. *Linguistics*, 18:381–484, 1980.

[Mey97] Miriam Meyerhoff. *Be i no gat: Constraints on Null Subjects in Bislama*. PhD thesis, University of Pennsylvania, 1997.

[Moo91] John Moore. *Reduced Constructions in Spanish*. PhD thesis, University of Santa Cruz, 1991.

[Nap81] Donna Jo Napoli. Semantic interpretation vs. lexical governance. *Language*, 57:841–887, 1981.

[Pic85] M. Carme Picallo. *Opaque Domains*. PhD thesis, CUNY, 1985.

[Pic90] M. Carme Picallo. Modal verbs in Catalan. *Natural Language and Linguistic Theory*, 8:285–312, 1990.

[Ram94] Owen Rambow. *Formal and computational aspects of natural language syntax*. PhD thesis, University of Pennsylvania, 1994.

[Ros90] Sara Thomas Rosen. *Argument Structure and Complex Predicates*. Garland Publishing, 1990.

[Sab95] Joachim Sabel. On parallels and differences between clitic climbing and long scrambling and the economy of derivations. In *Proceedings of NELS*, 1995.

[SH88] Beatrice Santorini and Caoline Heycock. Remarks on Causatives and Passive. Technical Report MS-CIS-88-33, University of Pennsylvania, 1988.

[Str81] Judith Strozer. An alternative to restructuring in Romance syntax. In H. Contreras and J. Klausenburger, editors, *Proceedings of the Tenth Anniversary Symposium on Romance Languages*, Seattle, 1981. University of Washington.

[Suñ80] Margarita Suñer. Clitic promotion in Spanish revisited. In F. Nuessel, editor, *Contemporary Studies in Romance Languages*, Bloomington, 1980. Indiana University Linguistics Club.

[Wei88] David Weir. *Characterizing mildly context-sensitive grammar formalisms*. PhD thesis, University of Pennsylvania, 1988.

Towards a model-theoretic characterization of indexed grammars

Tore Langholm

Preliminary Version

Recent work by Kracht, Rogers and others has opened up a systematic study of the relation between two alternative ways of specifying syntactic structures. These can be viewed as the possible end results of the construction processes described by various grammars, or alternatively as the potential models for the sentences of appropriate logics. Both perspectives open up the possibility of describing an infinite set of trees by finite means, either as the set of trees generated by a certain grammar, or as the set of models of a certain sentence. With a given grammar type or logical language, many sets will not be identifiable in this way, but stronger types of grammar or stronger languages will allow the identification of more sets of trees.

A notable result is the correspondence pointed out by Rogers between the context-free grammars and the monadic second-order language $L^2_{\emptyset,P}$ over finite ordered trees, saying roughly that the sets of trees generated from context-free grammars are exactly the sets definable by the sentences of $L^2_{\emptyset,P}$.

The language mentioned is equipped with individual variables ranging over nodes, set variables and set constants (the members of P) ranging over sets of nodes, truth-functional connectives and existential (and thus universal) quantifiers for both types of variables, together with two binary relation symbols $\lhd$ and $\prec$ which are restricted to denote immediate dominance and (weak) linear precedence, respectively. In addition, various (second-order) definable relations are included, such as identity and dominance.

The precise correspondence between $L^2_{\emptyset,P}$ and context-free grammars is this: Following Kracht, allow for a context-free grammar to contain a (finite) *set* of start symbols, rather than just a single one, and define a context-free feature grammar (cffg) to be a pair of a cfg and a classification scheme γ, i.e., a function from the (terminal and non-terminal) symbols of the grammar into the powerset of the set P of set constants. Moreover, define a feature tree to be a finite ordered tree where each node is decorated with a subset of P. Now a feature tree τ can be viewed as generated from a cffg iff the grammar generates some phrase-structure tree which is mapped to τ when symbols are replaced by their images under γ. Similarly a feature tree can be viewed as a structure satisfying or falsifying the

sentences of $L^2_{\emptyset,P}$ in the obvious way where a set constant decorates a node iff the node is contained in its interpretation.

Now Rogers has shown that for any set A of trees of bounded out-degree (i.e., any set for which there exists some finite number n such that no node in any contained tree has more than n children) there is a sentence of $L^2_{\emptyset,P}$ satisfied by exactly the members of A iff there is a cffg generating exactly the members of A.

Over the years a rich selection of grammar types stronger than the context-free has been developed, and it is of interest to determine if any of these can be related to corresponding extensions of $L^2_{\emptyset,P}$. The indexed grammars constitute a natural starting point for such investigations. An indexed grammar can be considered to generate a phrase-structure tree of the same type as those generated by context-free grammars, after the index-strings have been deleted. Moreover, paired with a classification scheme it can also be used to generate feature trees. The difference will of course be that such indexed feature grammars (ifg's) will be capable of identifying more sets of feature trees, but exactly *which* new sets are identifiable in this way depends on the exact variety of indexed grammars considered. Various alternative formulations ("normal forms") exist which are equivalent in the sense of generating the same string languages but which are seen to differ when attention is shifted to include the trees. A reasonable point of departure is the simple formulation given by Hopcroft and Ullman, allowing productions of the forms $A \rightarrow \alpha$ and $A \rightarrow Bf$ and $Af \rightarrow \alpha$. Any such grammar (or, more accurately, any *pair* of such a grammar and a classification scheme) corresponds to a sentence in an extension $L^2_{\emptyset,P,\emptyset}$ of $L^2_{\emptyset,P}$ obtained by the introduction of unary function variables. (The empty set at the third subscript position signifies that no function constants are added, only function variables.)

This extension of $L^2_{\emptyset,P}$ is perhaps also the obvious first extension to consider, but the leap from $L^2_{\emptyset,P}$ to $L^2_{\emptyset,P,\emptyset}$ is more radical than the step from context-free to indexed grammars: while the emptiness problem for indexed grammars is decidable, it is easily discovered that satisfiability of general $L^2_{\emptyset,P,\emptyset}$-sentences is undecidable.[1] Hence at the very least there can be no algorithm mapping the sentences of $L^2_{\emptyset,P,\emptyset}$ to corresponding ifg's. But in fact many sentences of $L^2_{\emptyset,P,\emptyset}$ will not correspond to any ifg, algorithmically or otherwise. The reason is simple; it was observed by several authors already in the late sixties and early seventies that the tree sets generated by indexed grammars are much like the string sets generated by context-free grammars; and in particular it can be shown that the class of feature tree sets generated by ifg's is not closed under intersection. Hence the ifg's correspond to *no* language on feature trees containing a general conjunction operator.

Hence what one can hope for is to relate the ifg's to a suitable *fragment* of $L^2_{\emptyset,P,\emptyset}$ which is not closed under conjunction. One approach along these lines is

[1] In fact, it is also undecidable when some finite bound has been introduced on the acceptable out-degree of feature trees. To simplify the discussions, the existence of such an implicit bound is assumed in the sequel. In a later version this will be brought to the surface in an explicit discussion.

to narrow the attention to sentences implying certain axioms that restrict the interaction between the unary functions. Let F be a finite set of unary function symbols, containing the distinguished element 1; the following axioms hold some interest.

Ax_1 $\bigwedge_{f\in F} \forall x(f(x) = f(1(x)) = 1(f(x)))$

Ax_2 $\bigwedge_{f\in F} \forall x\forall y(f(x) = f(y) \rightarrow f(x) = 1(x) \vee 1(x) = 1(y) \vee 1(y) = f(y))$

Ax_3 $\bigwedge_{f,g\in F; f\neq g} \forall x(f(x) = 1(x) \vee 1(x) = g(x))$

Ax_4 $\bigwedge_{f\in F} \forall x\exists Y(Y(f(x)) \wedge \forall z(Y(z) \rightarrow \bigwedge_{g\in F} Y(g(z))) \wedge (Y(x) \rightarrow x = f(x)))$

Ax_5 $\forall x\forall y((\bigwedge_{f\in F}(f(x) = 1(x) \wedge f(y) = 1(y))) \rightarrow 1(x) = 1(y))$

A set of functions satisfying these axioms corresponds to an assignment of stacks over $F - \{1\}$ to the elements of the domain. The function denoted by 1 can be imagined to take each node to a "canonical" node decorated by the same stack. Note in particular that $f(x) = 1(x)$ for "most" f and x; this would "encode" that f is not the top symbol of the stack decorating x. On the other hand, if $f(x)$ differs from $1(x)$ then this would encode that f *is* the top symbol of that stack, and popping that symbol yields the stack decorating the node $f(x)$. More precisely, the following representation result can be shown.

> A finite model $\langle D, 1^{\mathcal{M}}, f^{\mathcal{M}}, \ldots\rangle$ satisfies $Ax_1 - Ax_5$ iff $1^{\mathcal{M}}(1^{\mathcal{M}}(a)) = 1^{\mathcal{M}}(a)$ for all $a \in D$, and there exists a bijection ϕ from the range of $1^{\mathcal{M}}$ onto a downwards closed set of stacks over $F - \{1\}$ such that
>
> - if $TOP(\phi(1^{\mathcal{M}}(a))) = f$ then $f^{\mathcal{M}}(a)$ is the $b \in range(1^{\mathcal{M}})$ such that $\phi(b) = POP(\phi(1^{\mathcal{M}}(a)))$
> - otherwise (for instance when $\phi(1^{\mathcal{M}}(a))$ is empty) $f^{\mathcal{M}}(a) = 1^{\mathcal{M}}(a)$

Note that this holds for all finite models, including feature trees. It enables us to talk about stacks decorating the nodes without moving to a two-sorted logic with a separate domain of stacks.

Now let $G = \langle V, T, I, R, S\rangle$ be an indexed grammar as described by Hopcroft and Ullman, with the slight generalization that S is now a *subset* of V. Let Sy be $V \cup T$, and assume that $A_1, \ldots, A_m$ are the elements of this union. Below they will be treated syntactically as set variables. Let $i_1, \ldots, i_q$ be the members of I; these will be treated as unary function variables. In addition, 1 is a fresh unary function variable. Moreover, let γ be a classification scheme over Sy, i.e., a function from Sy into the powerset of the set P of set constants. The following abbreviations are used.

$1(x) \equiv 1(y)$:	$1(x) = 1(y)$.	
$i(x) \equiv 1(y)$:	$1(x) \neq i(x) \wedge i(x) = 1(y)$	for $i \in I$.
$1(x) \equiv j(y)$:	$1(x) = j(y) \wedge j(y) \neq 1(y)$	for $j \in I$.

$root(x)$:	$\neg\exists y\ y \lhd x.$
$leaf(x)$:	$\neg\exists y\ x \lhd y.$
$children(x, y_1, \ldots, y_n)$:	$y_1 \prec y_2 \wedge y_1 \neq y_2 \wedge \ldots \wedge y_{n-1} \prec y_n \wedge y_{n-1} \neq y_n$ $\wedge\ \forall z(x \lhd z \leftrightarrow z = y_1 \vee \ldots \vee z = y_n).$
$start(x)$:	$\bigvee_{A\in S} A(x).$
$stop(x)$:	$\bigvee_{A\in T} A(x).$
$empty(x)$:	$\bigwedge_{i\in I} i(x) = 1(x).$
$partition$:	$\forall x(A_1(x) \vee \ldots \vee A_m(x)) \wedge \bigwedge_{1\leq k<l\leq m} \neg\exists x(A_k(x) \wedge A_l(x)).$
$classification$:	$\bigwedge_{A\in S_Y} \forall x(A(x) \rightarrow \bigwedge_{p\in\gamma(A)} p(x) \wedge \bigwedge_{p\in(P-\gamma(A))} \neg p(x)).$
Ax_0:	$\forall x(root(x) \rightarrow empty(x)).$

For each rule $r = A \rightarrow B_1 \ldots B_n$ let $\varphi_r(x)$ be the formula

$$A(x)\wedge\exists y_1 \ldots \exists y_n(children(x, y_1, \ldots, y_n)\wedge B_1(y_1)\wedge 1(x) \equiv 1(y_1)\wedge\ldots\wedge B_n(y_n)\wedge 1(x) \equiv 1(y_n)).$$

For each rule $r = A \rightarrow B\ i$ let $\varphi_r(x)$ be the formula

$$A(x) \wedge \exists y(children(x, y) \wedge B(y) \wedge 1(x) \equiv i(y)).$$

For each rule $r = A\ i \rightarrow B_1 \ldots B_n$ let $\varphi_r(x)$ be the formula

$$A(x)\wedge\exists y_1 \ldots \exists y_n(children(x, y_1, \ldots, y_n)\wedge B_1(y_1)\wedge i(x) \equiv 1(y_1)\wedge\ldots\wedge B_n(y_n)\wedge i(x) \equiv 1(y_n)).$$

Finally let $\varphi_{G,\gamma}$ be the sentence

$$\begin{aligned}
&\exists 1\exists i_1 \ldots \exists i_q(\ Ax_0 \ \wedge\ Ax_1 \ \wedge Ax_2 \ \wedge\ Ax_3 \ \wedge \\
&\qquad \exists A_1 \ldots \exists A_m(\ partition \ \wedge\ classification \ \wedge \\
&\qquad\qquad \forall x\ (\ (root(x) \rightarrow start(x)) \ \wedge \\
&\qquad\qquad\quad (stop(x) \rightarrow leaf(x)) \ \wedge \\
&\qquad\qquad\quad (\neg stop(x) \rightarrow \textstyle\bigvee_{r\in R} \varphi_r(x))))).
\end{aligned}$$

The axioms Ax_4 and Ax_5 are not listed explicitly, but can be shown to follow from $\varphi_{G,\gamma}$. (More accurately, they are satisfied by any pair of a feature tree and a variable assignment on $\{1, i_1, \ldots, i_q\}$ that satisfies the subformula of $\varphi_{G,\gamma}$ obtained by deletion of the $q+1$ outermost quantifiers.) It is then straightforward to check that the ifg (G, γ) generates exactly the feature treesthat satisfy $\varphi_{G,\gamma}$. Now the quest ion is which results are obtainable in the opposite direction. For instance, is it the case that any $L^2_{\emptyset,P,\emptyset}$-sentence of the form

$$\exists 1\exists i_1 \ldots \exists i_q(Ax_0 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge \varphi)$$

or perhaps the form

$$\exists 1\exists i_1 \ldots \exists i_q(Ax_0 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge Ax_4 \wedge Ax_5 \wedge \varphi)$$

corresponds to an ifg, provided φ contains no quantification over function variables? We hope to be able to obtain alternative characterizations of the corresponding classes of tree sets in the future, but at first sight they do not appear

to correspond to any simple version of ifg's. On the other hand, something very close to the ifg's considered above appears to be captured when an additional restriction is put on the use of function variables inside of φ. For any f and g from the list $1, i_1, \ldots i_q$, with at least one of the two being equal to 1, let $f/g(x)$ be an abbreviation of $\exists y(y \lhd x \wedge f(x) \equiv g(y))$. Now restrict φ to contain occurrences of function variables only inside positively occurring subformulas of this type. Note that the $\varphi_{G,\gamma}$ above can easily be rewritten to have the inner φ satisfy this condition. It appears that under this restriction the sentences

$$\exists 1 \exists i_1 \ldots \exists i_q (Ax_0 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge \varphi)$$

or

$$\exists 1 \exists i_1 \ldots \exists i_q (Ax_0 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge Ax_4 \wedge Ax_5 \wedge \varphi)$$

can be shown to correspond to a slight generalization of indexed (feature) grammars in the sense of Hopcroft and Ullman, with the before-mentioned generalization to a *set* of start symbols, and with a slight generalization of the rule format which in particular includes a provision for "memory loss" to occur, in the sense of permitting productions like $A \rightarrow B*$, which allows the state of the stack to be changed arbitrarily when passing from A to B. While it is more general and flexible, such a grammar type still appears to be a variety of "indexed grammar" in the sense of allowing exactly the indexed string languages to be generated.

We have not chosen to define this more precisely at this point, since a detailed proof has not yet been produced. However, we expect to resolve this shortly, and will almost certainly have a positive result of this sort ready for the MOL meeting.

Note also that no conjecture has been made concerning the original formulation of indexed grammars given by Aho. This constitutes an alternative generalization of the form given by Hopcroft and Ullman in which more than one stack symbol may be pushed onto the stack in a single step. At first sight it may appear that such a situation is easily describable by a sentence of the approximate form

$$\exists y(y \lhd x \wedge f(g(h(x))) = 1(y)),$$

but this will not work with the given axioms, since the intermediate stack situations may not decorate any nodes, in which case no appropriate denotations can be had for the subterms $h(x)$ and $g(h(x))$.

Selected References

Kracht, Marcus, 1995. Syntactic Codes and Grammar Refinement, *Journal of Logic, Language and Information* **4**, pp. 41–60 & 359–380.

Rogers, James, 1996. A Model-Theoretic Framework for Theories of Syntax, *Proceedings of the 34th Annual Meeting of the ACL*, pp. 10–16.

Generative Capacity Matters

Alexis Manaster Ramer *and Walter Savitch †

Abstract

We discuss the relevance of mathematical results on weak generative capacity. We contend that such results can (still) be relevant but that they must be handled with more subtle attention to detail than they normally receive.

1 Introduction

The question of weak generative capacity of natural language has at times been considered a critical issue in Linguistics and at other times been considered irrelevant. Moreover, some segments of the community have moved from one position to the other and back again. When Chomsky set up the framework from which most mathematical linguistics has evolved he wrote

> ... the main problem of immediate relevance to the theory of language is that of determining where in the hierarchy of devices the grammars of natural languages lie. It would, f.e., be extremely interesting to know whether it is in principle possible to construct a phrase structure grammar for English (even though there is good motivation of other kinds for not doing so). (Chomsky, 1959)

Later he took the contrary view. Current attitudes in linguistics, pure as well as computational, say that weak generative capacity is at best of marginal significance. And some (notably, Chomsky 1986) go so far as to claim that such issues can in principle have no significance whatsoever in the linguistic arena. However, if history is any guide to the future, this view is likely to swing back again at some time.

Part of the problems in dealing with weak generative capacity may stem from the fact that the mathematical context seduces us into a simplistic way of viewing things. With a formal mathematical definition of "context-free language" the question of whether or not a particular (well defined) language

*Wayne State University Univ. manaster@umich.edu

†University of California, San Diego. wsavitch@ucsd.edu

is context-free or not is a mathematically precise question with a definitive yes or no answer. Perhaps because of this setting, we tend to think that the question "Does weak generative capacity matters to linguistics?" should also have a simple yes or no answer. Moreover, we also tend to think that, if the answer is yes, then the answer is a very broad yes that applies to almost all of linguistics, and if the answer is no, then weak generative capacity does not ever matter in linguistics inquiry. We contend that this is too simplistic a view.

The topic of weak generative capacity is a mathematically precise area with mathematically precise results. There are no significant problems with the mathematics. The theorems are by and large correct. The only lasting controversy has to do with whether or not the mathematics applies to real natural language. In the simplistic case, somebody claims that some natural language is not context-free using some data and some theorem deriving a result from the data. [Postal, Culy, Shieber, Manaster Ramer] When such results are called into question, it is the data that is questioned. The theorem holds, but if the data does not hold up under scrutiny, then the theorem becomes irrelevant.

(There are cases where the mathematics itself is called into question. See for example, Pullum and Gazdar 1982. However, these cases are rare. Most of these cases can arguably be considered a misinterpreting of the data, and in any event, mistakes in mathematics can be checked and do not stand the test of time.)

Claims that a given language is or is not context-free are easy to understand and the relevance of the mathematics is, as we have seen, easy to characterize. With mathematical results of a more general nature, the situation is a bit more subtle, but is basically the same. The issue almost never is whether or not the mathematics is correct. The issue always is whether or not the mathematics is relevant to linguistics. For example, when somebody proposes a model for natural language or some portion of natural language, it is typically true that the model describes a clearly wider classes of languages than natural language (or that portion of natural language under study). All proposed models from Chomsky's context-sensitive grammars and transformational grammars to various more recent models over generate. In particular, the so called "weakly context-sensitive grammars" over generate. In these cases it is not clear whether the grammar model should be considered a relevant model or not. In cases where the model grossly overgenerates, as when it characterizes all recursive or all recursively enumerable language, the model is clearly suspect. In those cases the grammar can model any algorithmic process what so ever and so it is not at all clear that it models any particular process, such as natural language grammar. In situations where the model only modestly over generates the relevance of the model is less open to attack, but in all cases, the issue is the relevance of the model.

In almost all cases where the mathematical results have fallen into disrepute, it is the relevance of the result that is questioned, not the mathematics. Currently, most results in mathematical linguistics have received criticism against their relevance to linguistics. It may appear that there is something intrinsic that forces us to choose between mathematical rigor and linguistics relevance. We content that things are not that pessimistic. We contend that mathematical results can be linguistically relevant. However, this relevance requires that we simultaneously become both mathematically and linguistically sophisticated in our analysis. We cannot simply take stock theorem and their stock intuitive interpretation and apply them to empirical data without any further analysis.

Before going on, we should point out that this misuse of stock theorems and their simplistic interpretation is not limited to linguistics. This is a general phenomenon in science. One of the most glaring current examples is the misuse of NP-completeness results in many branches of science. Scientists who learn a few basic techniques manage to prove that some model is NP-complete for some problems and then conclude that the problem is "computationally impossible." A more careful look at the situation almost always reveals that the application of the mathematics to the real world situation is very glib and not clearly applicable. In these cases the result says something about computation in the model, but it does not say the model is computationally impossible. To take a very simple case, the so called traveling salesman problem is NP-complete. This says that it is NP-complete to compute a maximally economic route of travel for a traveling salesman (or anybody else). Yet, we compute economical schedules everyday, including schedules for traveling salesmen. In this case there are two mismatches between the model and the real world. The real world allows for approximate solutions and the real word, in these cases, deals with finite models while the mathematics deals with infinite models. The application of the NP-result says something, but what it says is much more subtle than what people interpret it to say.

There is a certain intrinsic problem in applying mathematical results to any real world situation. The mathematics is always an analogy to the real situation. As with all analogies, the mathematics correctly models some aspects of the real situation and misses other aspects. The researcher must show that the mathematics correctly models enough relevant features of the real world situation and misses only irrelevant features. We content, however, that a good approximation of this goal is not beyond reach.

In this paper we content that one does not have to choose between mathematical rigor and linguistics insight. If one uses more precision in applying mathematical results, one can have both mathematical rigor and linguistic relevance. We illustrate this with a small example from morphology which applies the principle of avoiding excess generative capacity. The example we have chosen happens to involve languages which are sets of words rather

than sets of sentences and happens to be concerned with regular, rather than context-free languages. These two features mean that it is that much easier to present briefly, and, moreover, it happens to represent a more or less finished piece of work (while we have nothing comparable to offer at the moment in the domain of syntax or of context-free languages).

2 An Example from Morphology

There has been a lot of interest lately in using finite- state models for morphology, which of course implies that the set of word forms of a language is a regular language. Recent work by Creider et al. (1994) discusses a number of problems with the finite-state approach, all of which boil down to the fact that a simple finite automaton will process words from left to right, yet languages sometimes exhibit phenomena which can only be described this way with some significant loss of naturalness. For example, Creider et. al. mention a language in which some roots and some suffixes trigger a change in the vowels of the entire word, including any prefixes. This kind of vowel harmony, which can be triggered by a non-initial morpheme in the word, is a problem for left-to-right processing. Another problem is the existence of morphological patterns where a root can take a given suffix S only if it is preceded by a given prefix P. Thus, English allows *joy*, *enjoy*, *enjoyable*, but not **joyable*.

Creider et al. propose a model which elegantly handles all such problems. Their model is a modified version of Rosenberg's (1967) two-tape nondeterministic finite automaton (NFA). In their ingenious modification, instead of having two tapes, they assume a two-way-infinite tape and two heads, one of which can only move to the left (and scans prefixes) and the other of which can only move to the right (and scans suffixes). The automaton starts out by positioning these heads on the root, which is found by guessing nondeterministically.

The problem with the Creider et. al. model is, as Creider et. al. point out, that such automata generate linear context-free languages, a proper superset of regular languages. Yet the extra power is not required for anything in the kinds of examples Creider et al. discuss. So, in light of our opening remarks, what we need instead is to look for a model which has the extra descriptive power that Creider et al.'s model has but which does NOT generate nonregular languages.

Such a model is easy to construct once we analyze what it is that Creider et al.'s new model does as opposed to what a simple (left-to-right) NFA does. Specifically, we would seem to want a kind of nondeterministic one-tape Turing machine which can move its head freely to the left as well as the right but which is somehow prevented from accepting nonregular languages.

This can be accomplished by restricting how much the machine can write

on its tape (and hence how much memory it has). Specifically, we will allow the machine, like in Creider et al.'s model, to guess nondeterministically the location of the root of the word being processed before it does any processing of prefixes or suffixes. In order for the guess to do any good, the machine must presumably mark the position of the root on the tape in some way. That is the only time it is allowed to write. The details here are unimportant, but basically, we can have a set of distinguished symbols to be used for marking the root, and we will permit the machine to write only once. Assuming that the input is presented as a sequence of morphemes, this should suffice for the linguistic purposes, since we can always find the root. (If the input were written as a sequence of phonemes, we would need to mark the beginning as well as the end of the root). At the same time it clearly suffices to guarantee that only regular languages can be accepted.

The model can be described in more detail as follows: It is a highly restricted one-tape nondeterministic Turing machine. The initial tape configuration consists of the input string delimited by left and right end markers. The machine has one head, which is not allowed to move outside of the end markers. For concreteness, let's say that the head starts on the left end marker in a designated start state, although these details are of no consequence to the results being proved. The machine is nondeterministic and accepts by entering a designated accepting state.

If the one head on our model were a read-only head, then the machine would be a standard two-way nondeterministic finite state automata, and so would accept exactly the class of regular languages. However, our model is allowed to perform a very limited amount of writing. Specifically, the machine in the designated start state continually moves left and remains in the start state until it nondeterministically decides to overwrite a single symbol. After it writes this single symbol, it never again overwrites a symbol and never again returns to the start state. Thus, the model is allowed to write at most one symbol and after that behaves as a two-way nondeterministic finite automata.

For notational simplicity, we assume that the start state is not an accepting state, that for each input symbol a, there is only one possible symbol a' that can be written in place of a, that the symbols a' are disjoint from the input symbols a, and that the machine goes to a unique state after its one allowed write move. Again, the result does not depend on these simplifying details. It is, however, important to note that different symbols a and b can be overwritten with possibly different symbols a' and b'. We will refer to the symbols that the machine is allowed to write as "primed symbols." We will call a machine like this a "write-once machine." To see that these write-once machines accept exactly the class of regular languages, let us develop a small amount of notation. Let M be one of these write-once machines and let L be the language accepted by M. Our goal is to show that L is a regular language.

After the machine M has written its one allowed symbol, it then behaves just like a two-way finite state acceptor and so we can define an associated finite state acceptor M'. M' is simply M with the start state removed and with notational adjustments so that it starts computing where M left off. Since M' is an ordinary two-way finite-state automaton it accepts some regular language L'. Without loss of generality, we can assume that M' also checks the input to make sure it contains exactly one primed symbol so each string in L' contains exactly one primed symbol. It will follow that L is a regular language because L can be obtained from the regular language L' by an operation that always takes regular languages to regular languages. Specifically let h be the homomorphism that removes primes. In our prime notation $h(a') = a$ and $h(a) = a$ for each input symbol a of our original write-once machine M. With this homomorphism h, $L = h(L')$. Now, h is a homomorphism, L' is a regular language, and the regular languages are closed under homomorphism [See, for example, Hopcroft and Ullman, 1979]; so L is a regular language.

Thus, the language accepted by any of these write-once machines must be a regular language. Since it is trivial to see that any regular language can be accepted by one of these write-once machines, it follows that these write-once machines accept exactly the class of regular languages.

The write-once machines defined exactly as we have defined them are adequate for the linguistic purpose at hand. However, it is of interest to note in passing that, if you make almost any minor change that does not change the spirit of the definition, then the machines still accept only regular languages, although in some cases the proof becomes a bit more cumbersome. In particular, the write-once machine need not go to a unique state after writing its one symbol, but may go to a state that depends on the symbols read and written and it will still accept only regular languages. Also, the write-once machine can be redefined so that it can move back and forth (rather than only to the right) before it writes its one symbol and it will still accept only regular languages. While it is natural to assume that the primed and unprimed symbols are disjoint, the result still holds if we relax this assumption, although the proof would then be given in terms of finite-state transducers rather than homomorphisms.

3 Conclusion

The reason we chose to dwell on this very simple example is that it seems to illustrate what should have been done in syntax in the period of the late fifties and early sixties, and what should always be our attitude towards the development of new models in linguistics. At that time, having concluded that phrase structure grammar was an inadequate model of language, Chomsky introduced a model which happened to generate an even broader class

of languages, although there was no evidence that even the full context-sensitive generative capacity of phrase structure grammars was required for the analysis of natural language syntax. The goal was to capture all the cases missed by the phrase structure grammar and the most straight forward application of the mathematics says that to get more add more power. However, the linguistic goal is not to get more sentences but to get more precision in capturing sentences and that requires looking at the mathematics with more subtle attention to real world data. A theory which allowed all context-sensitive languages to be generated, such as certain versions of phrase structure grammar, was already much too powerful and should have been rejected–or constrained, instead of being replaced with an even more powerful theory than that of transformations.

Generative capacity is one of many considerations which can be invoked when judging the validity or utility of a model of human language. There are many possible criteria by which a linguistic theory may be judged and hence a variety of ways in which it can turn out to be wanting. Adequate but not excessive generative capacity is surely one such criterion, since that amounts to fitting the most conspicuous language data to the model. In this paper we have contended, and hope to have illustrated, the point that generative capacity, if handled with enough care to the details of the particular case at hand, can produce results that are both mathematically rigorous and linguistically insightful. Moreover, this attention to the details of the linguistic case at hand is more subtly important than has been reflected in the treatment generative capacity typically receives.

One can argue endlessly about which considerations are more, and which are less, important, but the fact remains that, at the end of the day, if a given theory is found to be wanting with respect to any of the various possible criteria, then the theory is wanting. In some cases generative capacity may turn out to be the crucial consideration that results in sharpening our linguistics insights. Certainly, historically, arguments about inadequacy in generative capacity played a vital role in convincing many experts of the need for transformational rules in preference to various kinds of phrase structure or combinatorial possibilities (see, for example, Bar-Hillel and Shamir, 1960). Future research need not, indeed should not, abandon the study of generative capacity, but the study of generative capacity must grow in linguistic sophistication and not be either abandoned nor allowed to generate continually more powerful mathematics with only casual checks of the links between the linguistics and the mathematics.

References

[1] Bar-Hillel, Y. and E. Shamir, "Finite state languages: formal representations and adequacy problems," 1960, as reprinted in *Language and*

Information (Y. Bar-Hillel, ed.), 1964, Addison-Wesley, Reading Mass, 87-98.

[2] Chomsky, N., "On certain formal properties of grammars," *Information and Control* **2** (1959), 137-167.

[3] Chomsky, N. *Aspects of the Theory of Syntax*, 1965, MIT Press, Cambridge, Mass.

[4] Chomsky, N., *Knowledge of Language: Its Nature, Origins, and Use*, 1986, New York, Praeger.

[5] Creider, Chet, J. Hankamer, and D. Wood, 1994. "Preset two-head automata: a basis for morphological analysis of natural languages," in *Mathematical Linguistics and Related Topics*, (G. Paun, ed.), 1994, Publishing House of the Romanian Academy of Sciences, Bucharest.

[6] Culy, C., "The complexity of the vocabulary of Bambara," *Linguistics and Philosophy* **8** (1985), 345-351. (Reprinted in Savitch, et. al., 1987.)

[7] Hopcroft, J. and J. Ullman, *Introduction to Automata Theory, Languages and Computation*, 1979, Addison-Wesley, Reading Mass.

[8] Manaster Ramer, A., "Subject-verb agreement in respective coordinations," *Computational Linguistics* **13** (1987), 64-65.

[9] Pullum, G. K. and G. Gazdar, "Natural languages and context-free languages", *Linguistics and Philosophy* **4** (1982), 471-504. (Reprinted in Savitch, et. al., 1987.)

[10] Rosenberg, A. L., "A machine realization of linear context-free languages," *Information and Control* **10** (1967), 175-188.

[11] Savitch, W. J., E. Bach, W. Marsh, and G. Safran-Naveh, *The Formal Complexity of Natural Language*, 1987, D. Reidel Publishing Co., Dordrecht, Holland.

[12] Shieber, S. M., "Evidence against the context-freeness of natural language," *Linguistics and Philosophy* **8** (1983), 333-343. (Reprinted in Savitch, et. al., 1987.)

How to Solve Domain Equations Involving Path Equalities

M. Andrew Moshier

January 1997

1 Introduction

In (Moshier 96), the author shows that the domain of feature structures ordered by subsumption can be constructed using only the tools available in domain theory and logic. That is, one can think of the subsumption ordering on feature structures as arising naturally from a domain of trees together with the need to reason about path equality in those trees. The results in (Moshier 96) show that feature structures with subsumption ordering in fact form a most general solution to the problem of constructing a domain of tree-like objects for which assertions of path equality make sense. This partly justifies (from the domain theorists' view) the interest in feature structures that marks much of computational linguistics and logic programming research. The earlier paper left two important and closely related bits of unfinished business, however, both of which are the main subjects of this work.

First, "domain of tree-like objects" simply meant a domain that maps continuously (with some side conditions satisfied) onto a specific domain defined to represent trees in an obvious way. That is, there was no requirement that this new domain actually carry any tree structure of its own, but only that it map in a certain way to a domain that happens to carry the desired tree structure. As noted in the paper, the domain of feature structures actually does carry a tree-like structure of its own: it is a non-initial solution to the very domain equation for which the domain of trees is an initial solution. Nevertheless, the construction used in the paper cannot offer an explanation for this. It seemed to the author at the time that it is no coincidence that feature structures are tree-like in this stronger sense of being a non-initial solution to the tree-defining domain equation, but the constructions considered in the earlier paper simply take a domain, and not the domain equation that determines the domain, as data.

Second, the notions of "path" and "path equality" are defined with respect to the domain of trees, and not with respect to the determining domain equation. That is, again the link between the original domain equation and the domain of feature structures is lost. The earlier work adequately (and completely, in light of the universal construction) explains how the domain of feature structures arises from the domain of feature trees plus a definition of path equality. What is missing is any explanation of how paths and path equalities relate, if in fact they do, to the very domain equation that characterizes what is meant by "tree" (and hence, at least informally, "path") in the first place.

The two problems sketched above are resolved here, by demonstrating that the domain of feature structures can be constructed directly from a domain equation (or system of equations) that defines our notion of trees. The construction stands on a proof of existence for a solutions for a certain class of problems in domain theory. Importantly, the construction is motivated entirely by concerns

common in domain theory, principally the notions of continuity and universality. The significance of this is that the results reported here provide a natural justification for our interest in feature structures with respect to subsumption and unification, and by virtue of the general existence proof, show how a wide variety of similar domains can be constructed entirely from formal specifications of thier domain theoretic properties.

2 Review

The universal construction investigated in (Moshier, JOLLI 4:111-143, 1995) can be simplified for the purposes of this paper as follows. A *domain* is a partially ordered set having a least element and suprema for all directed subsets. Domains constitute the objects of a category DCPO in which the arrows are simply maps that preserve suprema of directed sets. Maps are not generally required to preserve least element. A domain X can also be described as a topological space (known as the *Scott topology* of X) by taking suprema of directed sets as limit points. That is, take $U \subseteq X$ to be open provided that for all directed $D \subseteq X$, $\bigvee D \in U$ if and only $D \cap U \neq \emptyset$. Scott topologies are significant because the arrows of DCPO are precisely the maps between domains that are continuous with respect to Scott topologies. On this observation, morphisms in DCPO are called *continuous maps*. Also, the Scott open sets of a domain A are important because they can be regarded as characterizing the partial order on A precisely as an order of information content. That is, $a \leq b$ holds in A if and only if every open neighborhood of a is also a neighborhood of b. So the order on A essentially answers the question of how much information with respect to membership in opens each element contains.

A subset $J \subseteq X$ of a domain X is called *inductive* provided that every directed subset of J has its supremum also in J. Let $I(X)$ denote the collection of inductive subsets of X. Then one can easily check that $I(X)$ is closed under finite unions and intersections, and thus is a distributive lattice. In fact, open sets are always inductive, so $\sigma(X)$ is a sublattice of $I(X)$. If f is continuous from X to Y, then f^{-1} sends inductive sets to inductive sets. So, f^{-1} can be regarded as a homomorphism from $I(Y)$ and $I(X)$.

Fix a 2211 algebra $L = \langle L, \wedge, \vee, \top, \bot \rangle$ and subalgebra L_0. We take L_0 to denote the inclusion map from L_0 to L as well. Then we are concerned primarily with *interpretations* of L in $I(X)$ and $\Sigma(X)$, i.e., homomorphisms from L to $I(X)$ and to $\Sigma(X)$. Specifically, we define two categories CP (constraint problems) and CD (constraint domains) as follows. Objects of CP are pairs $\langle X, [\![\cdot]\!] \rangle$ where X is a domain and $[\![\cdot]\!]$ is a homomorphism from L to $I(X)$ which sends L_0 to $\sigma(X)$. In other words, $[\![\cdot]\!]$ interprets a token in $\mathcal{L}$ as an inductive set and in particular, a token in L_0 as an open set. Arrows of CP from $\langle X, [\![\cdot]\!] \rangle$ to $\langle X', [\![\cdot]\!]' \rangle$ are continuous maps $f : X \to X'$ so that

$$\begin{aligned} [\![\cdot]\!] &\leq f^{-1} \circ [\![\cdot]\!]' \\ [\![\cdot]\!] \circ L_0 &= f^{-1} \circ [\![\cdot]\!]' \circ L_0 \end{aligned}$$

where homomorphisms are ordered pointwise. The objects of CD are pairs $\langle X, [\![\cdot]\!] \rangle$ so that $[\![\cdot]\!]$ is a homomorphism from L to $\Sigma(X)$. Arrows from $\langle X, [\![\cdot]\!] \rangle$ to $\langle X', [\![\cdot]\!]' \rangle$ are continuous maps $f : X \to X'$ so that

$$[\![\cdot]\!] = f^{-1} \circ [\![\cdot]\!]'$$

The central results of (Moshier 1995) simplify to the following two theorems.

Theorem 2.1. *The forgetful functor from CD to CP has a right adjoint, i.e., for each object $\langle X, [\![\cdot]\!] \rangle$ of CP, there is an object $\langle X^*, [\![\cdot]\!]^* \rangle$ of CD and an arrow $u\colon X^* \to X$ in CP so that for any other arrow v from an object $\langle Y, [\![\cdot]\!]' \rangle$ of CD to $\langle X, [\![\cdot]\!] \rangle$, there is a unique CD arrow $\hat{v}$ for which $v = u \circ \hat{v}$.*

Fix a set Σ, and let L be the free algebra generated by tokens $w \doteq v$ for $w, v \in \Sigma^*$. Also, let L_0 be the subalgebra generated by tokens $w \doteq w$ for $w \in \Sigma^*$. For an arbitrary domain S, let FT be an initial solution to the domain equation

$$X \simeq S + [\Sigma_\perp \to_s X_\perp] \tag{1}$$

Here $A_\perp$ is the partial order obtained by adding a new bottom element to A, $+$ constructs the strict sum (disjoint union except that least elements are identified), and $\to_s$ constructs the strict function space (continuous functions that preserve least element). Elements of FT can be seen as trees with leaves labeled by elements of S and edges labeled by elements of Σ. In FT, each $w \in F^*$ determines a partial map $t \mapsto t/w$ from FT to FT that picks out a subtree by following the path w, provided that the path exists in t. Notice that elements $t \in \mathsf{FT}$ for which t/w is defined form a Scott open set in FT. Now define $[\![\cdot]\!]\colon L \to I(\mathsf{FT})$ by

$$[\![w \doteq v]\!] = \{t \in \mathsf{FT} \mid t/w = t/v \text{ both defined}\}$$

Thus $[\![\cdot]\!]$ picks out inductive subsets of FT and in particular, open subsets when restricted to L_0.

Theorem 2.2. *Feature structures with subsumption form a pre-order that is equivalent to the domain* FT^* *in* $\langle \mathsf{FT}^*, [\![\cdot]\!]^* \rangle$*, the right adjoint to* $\langle \mathsf{FT}, [\![\cdot]\!] \rangle$*. Moreover, the counit of the adjunction (the required continuous map from* FT^* *to* FT*) is precisely the "unfolding" map that sends a feature structure to its underlying tree.*

The first result provides a general construction (via adjunction) for building domains from (i) a domain X and (ii) an interpretation of a positive logic L in the inductive subsets of X. The resulting domain X^* interprets the same positive logic in open subsets of the result. The adjunction ensures that X^* is the most general domain in which such an interpretation is possible, while maintaining a systematic relationship with X.

The second result simply demonstrates that the general construction yields feature structures in the specific case of a tree domain and path equations.

3 Results

On closer inspection of the above results, the reader will notice that the construction of the domain of feature structures involves two very distinct stages. First, we specify FT, a domain of trees, via the standard technique of domain equations. Then, with no apparent reference to the domain equation, we define L, L_0 and $[\![\cdot]\!]$. Clearly, however, L, L_0 and $[\![\cdot]\!]$ are somehow related to the original domain equation. In particular, the domain FT^*, resulting from the adjunction in Theorem 2.2, turns out to be a non-initial solution to the domain equation. The results here show that this is not an accident. The data L, L_0 and $[\![\cdot]\!]$ can be systematically derived from any system of domain equations with certain side definitions. The adjunction in Theorem 2.2 from such data is then guaranteed to yield a non-initial solution to the system. Furthermore, this solution can be characterized as enjoying a certain universal property with respect to the system of equations with its side definitions.

What we usually call "features" are essentially partial continuous maps, i.e., partial maps from one domain to another (in the cases of FT and FT*, the features run from the domain back to the same domain). In addition, the domain of definition for a feature is an open set, and the feature (viewed as a map) is continuous on this open set. We take these properties as a "qualitative" definition of features. Thus, we take a *feature from domain A to domain B* to be a pair $f = (U, m)$, where $U \subseteq A$ is open and $m\colon U \to B$ is continuous. For example, in FT, each label $l \in \Sigma$ determines $\cdot/l$, a partial continuous map having an open set as its domain of definition. Thus these partial maps $\cdot/l$ can be taken as features from FT to FT.

Features compose in the following way. Let (U, m) be a natural feature from A to B and let (V, n) be a natural feature from B to C. Then the set

$$(V \circ U) = U \cap m^{-1}(V)$$

is open in A. Moreover, elements of $(V \circ U)$ are sent via m to elements of V, and $x \mapsto n(m(x))$ is defined and continuous for elements of $V \circ U$. Thus we can take $(V, n) \circ (U, m)$ to be $(V \circ U, n \circ m)$, with $n \circ m$ understood as being defined only on $V \circ U$.

Equality of features relates to inductive sets as by the follow lemma.

Lemma 3.1. *For domains A and B, and features (U, m) and (V, n) from A to B, the set $e((U, m), (V, n)) = \{a \in U \cap V \mid m(a) = n(a)\}$ is inductive in A. Furthermore $e((U, m), (U, m)) = U$, and hence is open.*

Notice that the "featurehood" of $\cdot/l$ doesn't actually depend on much about FT. Rather, for any domain of the form

$$F(X) = S + [\Sigma_\perp \to_s X_\perp] \tag{2}$$

we can define a similar feature from $F(X)$ to X. Namely, $\cdot/l$ is defined only on the elements of $F(X)$ corresponding to strict functions $f \in [\Sigma_\perp \to_s X_\perp]$ for which $f(l) \neq \perp$. For such elements f, we have $f/l = f(l)$. In an important, technical sense, the definition of $\cdot/l$ as a feature depends only on the functor F defined in (2). This suggests that certain families of features may be taken as *natural*.

Given two functors $G\colon \mathcal{A} \Rightarrow \mathsf{DCPO}$ and $H : \mathcal{A} \Rightarrow \mathsf{DCPO}$, define a *natural feature from G to H* as a family of features $\{(U_a, m_a)\}_a$ indexed by objects in the category $\mathcal{A}$, so that for all arrows $f\colon a \to b$ in $\mathcal{A}$, it is the case that

$$\begin{aligned} U_a &\subseteq G(f)^{-1}(U_b) \qquad \text{so the notation } U(f) \text{ is functorial} \\ m_b \circ U(f) &= H(f) \circ m_a \end{aligned}$$

The composition of features extends to natural features between functors, in the sense that for natural features $\{(U_a, m_a)\}_a$ from G to H and $\{(V_a, n_a)\}_a$ from H to I, the family of features $\{(V_a \circ U_a, n_a \circ m_a)\}_a$ constitutes a natural feature from G to I.

In the case of F as defined above, each $l \in \Sigma$ determines a natural feature $\cdot/l$ from F to 1_{DCPO}. In words, for any domain of the form $F(X)$ it makes perfect sense to speak about the "feature" l, regardless of whether $F(X)$ is anything like a domain of trees or feature structures. Roughly, this is because the functor F itself specifies what we mean by "tree-like" with respect to having a root and immediate children, and the property of possessing a child along an edge labelled l is a local matter that says nothing about the shape the children take. In other words, the particular features from which paths are built in FT arise directly from the defining equation (1).

Suppose that $\langle e\colon F(A) \to A, p\colon A \to F(A)\rangle$ is a (possibly non-initial) solution to the equation (1). That is, e and p are such that $ep \leq \mathsf{id}_A$ and $pe = \mathsf{id}_{F(A)}$. (here p is called a *projection*, e, an *embedding*). Then it is possible in a uniform way to define *paths* in A. Specifically, what is needed is to connect the natural features $\cdot/l$, which are (at the object A) features from $F(A)$ to A, to features from A back to itself. The map p gives the means of making this connection. That is, more generally if $\{(U_X, m_X)\}$ is a natural feature from F to 1_{DCPO}, and $\langle e\colon F(A) \to A, p\colon A \to F(A)\rangle$ $X \simeq F(X)$, then $(p^{-1}(U_A), m_A \circ p)$ is a feature from A to A. Through composition of features, this provides a notion of *path* in A. ay are determined precisely by the
original functor F (or more generally, the system of functors) that specifies the space of solutions.

Lemma 3.2. *Given a (continuous) functor* $F\colon \mathsf{DCPO} \Rightarrow \mathsf{DCPO}$ *and a natural feature* $\{(U_X, m_X)\}_X$ *from* F *to* 1_{DCPO}, *each solution* $\langle e\colon F(A) \to A, p\colon A \to F(A)\rangle$ *of* $X \simeq F(X)$ *determines a feature* $(p^{-1}(U_A), m_A \circ p)$ *from* A *to* A. *Moreover, this feature is natural with respect to* p.

Fix a functor $F\colon \mathsf{DCPO} \Rightarrow \mathsf{DCPO}$ and Φ a set of natural features from F to 1_{DCPO}. Then in any solution $\langle e\colon F(A) \to A, p\colon A \to F(A)\rangle$ determines features from A to A and hence also paths. That is, take the set P_Φ to be the least set of features from A to A so that $(A, \mathsf{id}_A) \in P_\Phi$ and if $(U, m) \in P_\Phi$ and $(V, m) \in \Phi$, then $(V \circ U, n \circ m) \in P_\Phi$.

Take L^Φ to be the free 2211 algebra generated from $P_\Phi \times P_\Phi$. Generators are written $w \doteq v$ for $w, v \in P_\Phi$. Also, take L_0^Φ as the subalgebra generated by the diagonal elements $w \doteq w$.

Thus each solution of $X \simeq F(X)$ determines a constraint problem, $\langle A, [\![\cdot]\!]\rangle$ where the homomorphism $[\![\cdot]\!]$ is given by $[\![w \doteq v]\!] = e(w, v)$. This leads to our main theorem.

Theorem 3.3. *Fix functor* $F\colon \mathsf{DCPO} \Rightarrow \mathsf{DCPO}$ *and set* Φ *of natural features from* F *to* 1_{DCPO}. *The right adjoint of the forgetful functor from constraint domains to constraint problems preserves solutions of* $X \simeq F(X)$, *as well as projections that commute with solutions.*

In the motivating application for this theorem, the domain of feature structures arises as the right adjoint of the initial solution to (1). The theorem shows that (i) this right adjoint is guaranteed to be a solution to (1) as well, and (ii) because the adjunction also preserves projections that commute with solutions, feature structures can be seen as an initial solution for (1) subject to the requirement that the solution also must be a constraint domain.

In the full paper, the main results are show to generalize to systems of equations and appropriateness conditions for features. Also, the notion of n-ary features is considered in some detail.

4 Conclusion

The results reported here show that feature structures with their subsumption order can be constructed via methods, the motivations for which are found entirely amongst the standard concerns of domain theory, i.e., continuity and the central importance of topology, solutions of domain equations, naturality and universality. This is important for the simple reason that is provides a formal justification for the claim that feature structures are, in a fundamental way, the most natural objects for the job they are typically put to in unification grammar formalisms. Without this sort of external justification, we might agree that feature structures are useful in practice, but will not be able to explain to anyone out of the community of users, what exactly makes them useful. The results here provide this explanation.

An Efficient Recognition Algorithm for Multiple Context-Free Languages

Ryuichi Nakanishi[1] Keita Takada[2] Hiroyuki Seki[1]
{nakanisi, seki}@is.aist-nara.ac.jp

[1] Graduate School of Information Science, Nara Institute of Science and Technology
[2] Application Engineering Laboratory, Matsushita Electronics Corporation

Abstract: Valiant proposed an $O(n^2)$ time algorithm which reduces the recognition problem for context-free languages (CFLs) to the boolean matrices multiplication problem. By this algorithm, the recognition problem for CFLs can be solved in $O(\max\{n^2, M(n)\})$ time where n is the length of an input string and $M(k)$ is the time needed for multiplying two $k \times k$ boolean matrices. The best known value for $M(k)$ is $O(k^{2.376})$. Multiple context-free grammars (MCFGs) were introduced to denote the syntax of natural languages. By the known fastest algorithm, the recognition problem for multiple context-free languages (MCFLs) can be solved in $O(n^e)$ time where e is a constant which depends only on a given MCFG G, called the degree of G.

In this paper, we propose an algorithm which reduces the recognition problem for MCFLs to the boolean matrices multiplication problem. By this algorithm, the recognition problem for MCFLs can be solved in $O(n^{e'-3i'+1} \cdot M(n^{i'}))$ time where e' and i' are constants which depend only on a given MCFG ($e' \leq e$, $i' \geq 1$). The time complexity of this algorithm is less than that of the forementioned algorithm unless $e' = e$ and $i' = 1$.
keywords multiple context-free grammar, boolean matrices multiplication, recognition algorithm, parsing, formal grammar

1 Introduction

It is often pointed out that the generative capacity of context-free grammars (*CFGs*) is too weak to generate natural languages, and there are various extensions of CFGs introduced to define the syntax of natural languages. *Multiple context-free grammar* (*MCFG*) [3][9] is one of such extensions of CFGs. A nonterminal of an MCFG derives tuples of strings while a nonterminal of a CFG derives strings. In an MCFG, it is possible to account for structures involving discontinuous constituents such as "respectively" sentences or inverted sentences in a simple manner.

Head grammars (*HGs*) [6] and tree adjoining grammars (*TAGs*) [2] were also proposed to denote the syntax of natural languages. The generative power of MCFGs is properly stronger than those of CFGs, TAGs and HGs, and properly weaker than that of context-sensitive grammars [3][9]. Linear context-free rewriting systems (*LCFRSs*) were also proposed to denote the syntax of natural languages [11]. LCFRSs which have strings as their domain are essentially the same formalism as MCFGs except that LCFRSs are required to satisfy non-erasing condition (Condition 1.3(c) of Lemma 2.1 in this paper). However, the generative power of LCFRSs is equal to that of MCFGs.

In [4][9], the recognition problem for multiple context-free languages (MCFLs) was shown to be decidable in $O(n^e)$ time, where n is the length of an input string and e is a constant which depends only on a given MCFG G, called the *degree* of G (see Section 2).

Cocke-Kasami-Younger (CKY) algorithm is a well-known $O(n^3)$ time recognition algorithm for context-free languages (CFLs). Based on CKY algorithm, Valiant proposed a recognition algorithm for CFLs. Valiant's algorithm reduces the recognition problem for CFLs to the $n \times n$ boolean matrices multiplication problem in $O(n^2)$ time [10]. So, the time complexity of the algorithm is $O(\max\{n^2, M(n)\})$, where $M(k)$ is the time needed for multiplying two $k \times k$ boolean matrices. The best known value for $M(k)$ is $O(k^{2.376})$ [1]. As an extension of [10], an $O(M(n^2))$ time recognition algorithm for tree-adjoing languages was proposed [7].

In this paper, as an extension of [7], we propose an $O(n^{e'-3i'+1} \cdot M(n^{i'}))$ time recognition algorithm for MCFLs which uses multiplications of boolean matrices, where e' and i' are constants which depend only on a given MCFG ($e' \leq e$, $i' \geq 1$). The time complexity of the algorithm is less than the complexity $O(n^e)$ of the algorithm in [4][9] unless $e' = e$ and $i' = 1$.

2 Preliminaries

We present the definition of MCFG [3][9][5] and some related concepts.

Definition 2.1 : An MCFG G is a 4-tuple (N, T, P, S) defined by the following (1) through (4).

(1) N is a finite set of *nonterminal symbols*. For each $A \in N$, a positive integer $d(A)$, called the *dimension* of A, is defined. A nonterminal symbol A generates $d(A)$-tuples of terminal strings. The *dimension* of G, denoted $d(G)$, is defined as $\max\{d(A) \mid A \in N\}$. Sometimes A is written as $(A^{[1]}, \ldots, A^{[d(A)]})$. $A^{[1]}, \ldots, A^{[d(A)]}$ are called *component symbols* of A. Let $N_{CMP} = \{A^{[i]} | A \in N, 1 \leq i \leq d(A)\}$.

(2) T is a finite set of *terminal symbols*.

(3) P is a finite set of *production rules*. Sometimes a nonterminal symbol, a terminal symbol and a production rule are called a nonterminal, a terminal and a rule, respectively. A rule, say p, has a form of $p : (A^{[1]}, \ldots, A^{[d(A)]}) \rightarrow (\gamma_1, \ldots, \gamma_{d(A)})$, where $A \in N$, $\gamma_1, \cdots, \gamma_{d(A)} \in (N_{CMP} \cup T)^*$. Each rule p satisfies the following condition.

Right-linearity: For each nonterminal B and each component symbol $B^{[i]}$ of B $(1 \leq i \leq d(B))$, $B^{[i]}$ appears in the right-hand side (rhs) of p at most once.

$A = (A^{[1]}, \ldots, A^{[d(A)]})$ is called *the nonterminal in the left-hand side (lhs) of* p. $B \in N$ is called *a nonterminal*

in the rhs of p if some component symbol $B^{[i]}$ $(1 \leq i \leq d(B))$ of B appears in the rhs of p. Also we call γ_k $(1 \leq k \leq d(A))$ the k-th component of the rhs of p. A rule p is called *terminating* if no component symbol appears in the rhs of p. Otherwise it is called *nonterminating*.

(4) $S \in N$ is *the start symbol* with $d(S) = 1$. □

The *degree* of a rule $p: A \to \cdots$, denoted $e(p)$, is defined as the sum of the two numbers, $d(A)$ and the number of component symbols in the rhs of p. The *degree* of an MCFG G, denoted $e(G)$, is defined as $\max\{e(p) \mid p$ is a rule in $G\}$.

$L_G(A)$, the *language* generated by a nonterminal A, is defined by the following (D1) through (D3).

(D1) $(\alpha_1, \ldots, \alpha_{d(A)}) \in L_G(A)$ if there exists a terminating rule $(A^{[1]}, \ldots, A^{[d(A)]}) \to (\alpha_1, \ldots, \alpha_{d(A)})$, where $\alpha_k \in T^*$ $(1 \leq k \leq d(A))$.
(D2) Let $p: (A^{[1]}, \ldots, A^{[d(A)]}) \to (\gamma_1, \ldots, \gamma_{d(A)})$ be a nonterminating rule and let $B_1, \ldots, B_n$ be the nonterminals in the rhs of p. Suppose $(\alpha_{i,1}, \ldots, \alpha_{i,d(B_i)}) \in L_G(B_i)$ $(1 \leq i \leq n)$. Then $L_G(A)$ contains the $d(A)$-tuple of strings obtained from $(\gamma_1, \ldots, \gamma_{d(A)})$ by replacing each component symbol $B_i^{[j]}$ with $\alpha_{i,j}$ $(1 \leq i \leq n, 1 \leq j \leq d(B_i))$.
(D3) $L_G(A)$ has no element other than those obtained by (D1) or (D2).

We often write $A \stackrel{*}{\Rightarrow} (\alpha_1, \ldots, \alpha_{d(A)})$ instead of $(\alpha_1, \ldots, \alpha_{d(A)}) \in L_G(A)$. We call α_k $(1 \leq k \leq d(A))$ *the string derived from* $A^{[k]}$ in the derivation $A \stackrel{*}{\Rightarrow} (\alpha_1, \ldots, \alpha_{d(A)})$. If we obtain $A \stackrel{*}{\Rightarrow} (\alpha_1, \ldots, \alpha_{d(A)})$ by applying the above (D1) or (D2) with a rule p to derivations $B_i \stackrel{*}{\Rightarrow} (\alpha_{i,1}, \ldots, \alpha_{i,d(B_i)})$ $(1 \leq i \leq n$, $n = 0$ if d is obtained by applying (D1) with a terminating rule $p)$, then we say that $A \stackrel{*}{\Rightarrow} (\alpha_1, \ldots, \alpha_{d(A)})$ is *a derivation rooted by* p.

$L(G)$, the *language* generated by an MCFG $G = (N, T, P, S)$, is defined to be $L_G(S)$. The language generated by an MCFG is called a *multiple context-free language* (*MCFL*). The *degree* $e(L)$ of an MCFL L is defined to be $\min\{e(G) \mid L(G) = L\}$.

Example 2.1 : Consider an MCFG $G = (N, T, P, S)$, where $N = \{S, A, B\}$ $(d(S) = 1, d(A) = d(B) = 2)$, $T = \{a, b, c, d\}$, $P = \{p_1: S^{[1]} \to A^{[1]}B^{[1]}A^{[2]}B^{[2]}$,
$p_2: (A^{[1]}, A^{[2]}) \to (aA^{[1]}, cA^{[2]})$,
$p_3: (A^{[1]}, A^{[2]}) \to (a, c)$,
$p_4: (B^{[1]}, B^{[2]}) \to (bB^{[1]}, dB^{[2]})$,
$p_5: (B^{[1]}, B^{[2]}) \to (b, d)\}$.
$L_G(A) = \{(a^n, c^n) | n \geq 1\}$, $L_G(B) = \{(b^m, d^m) | m \geq 1\}$, and $L_G(S) = L(G) = \{(a^n b^m c^n d^m) | n, m \geq 1\}$. By definition, $d(G) = 2$, $e(p_1) = 5$, $e(p_2) = e(p_4) = 4$, $e(p_3) = e(p_5) = 2$, $e(G) = 5$ and hence $e(L(G)) \leq 5$. □

Let A be the nonterminal of the lhs of a rule p and $B_1, \ldots, B_n$ be the nonterminals of the rhs of p $(n = 0$ if p is terminating). We sometimes write the rule as $p: A \to f[B_1, \ldots, B_n]$. The following lemma gives a normal form of MCFG. A proof is found in [8].

Lemma 2.1: Let G be an MCFG such that $\epsilon \notin L(G)$. An MCFG G' can be constructed from G such that $L(G') = L(G)$, $e(G') \leq e(G)$ and G' satisfies the following Conditions 1 through 3.

Condition 1: For every rule $p: A \to (\gamma_1, \ldots, \gamma_{d(A)}) = f[B_1, \ldots, B_n]$,
1.1 $\gamma_k \neq \epsilon$ for every $1 \leq k \leq d(A)$,
1.2 if p is terminating, then $d(A) = 1$ and $|\gamma_1| = 1$,
1.3 if p is nonterminating, then
(a) $\gamma_k \in N^*_{CMP}$ for every $1 \leq k \leq d(A)$, i.e., no terminal symbol appears in the rhs of p,
(b) $n = 2$,
(c) **Non-erasing condition**: every component symbol $B_i^{[j]}$ $(i = 1, 2$ and $1 \leq j \leq d(B_i))$ appears in γ_k for some $1 \leq k \leq d(A)$, and
(d) no pair $B^{[j]}, B^{[k]}$ $(1 \leq j < k \leq d(B), B = B_1, B_2)$ of component symbols of the same nonterminal appear adjacently in the rhs, i.e., component symbols of B_1 and B_2 appear alternately.
(e) There exist $i (1 \leq i \leq d(A))$ such that $|\gamma_i| \geq 2$.
Condition 3: $2d(G') + 1 \leq e(G') \leq 3d(G')$. □

By Lemma 2.1, we assume that a given MCFG satisfies Conditions 1 through 3 in the rest of the paper.

3 Algorithm

Let $G = (N, T, P, S)$ be an MCFG of dimension m and let $w = a_1 a_2 \cdots a_n$ be an input string where $n \geq 1$ and $a_i \in T$ $(1 \leq i \leq n)$. The proposed algorithm uses a $2m$-dimensional matrix M of size $\underbrace{(n+1) \times (n+1) \times \cdots \times (n+1)}_{2m}$. Each entry of M is a subset of N. We design the algorithm so that the following 1 and 2 are equivalent when the algorithm terminates (see Lemma 3.2).

1. $A \stackrel{*}{\Rightarrow}$
   $(a_{l_1+1} \cdots a_{r_1}, a_{l_2+1} \cdots a_{r_2}, \ldots, a_{l_{d(A)}+1} \cdots a_{r_{d(A)}})$.
2. $A \in M[l_1, r_1, l_2, r_2, \ldots, l_{d(A)}, r_{d(A)}, \underbrace{0, 0, \ldots, 0, 0}_{2(m-d(A))}]$

If the above two conditions are equivalent, then $S \stackrel{*}{\Rightarrow} a_1 \cdots a_n$ if and only if $S \in M[0, n, 0, \ldots, 0]$. In what follows, we abbreviate
$M[l_1, r_1, l_2, r_2, \ldots, l_i, r_i, \underbrace{0, 0, \ldots, 0, 0}_{2(m-i)}]$ $(i \leq m)$ as $M[l_1, r_1, l_2, r_2, \ldots, l_i, r_i]$, omitting the tailing 0s for short. Suppose that a rule

$$p: \begin{pmatrix} A^{[1]} \\ A^{[2]} \end{pmatrix} \to \begin{pmatrix} B^{[1]}C^{[2]} \\ C^{[3]}B^{[2]}C^{[1]} \end{pmatrix} \qquad (*1)$$

belongs to P. Then, there exists a derivation rooted by p:

$$\text{(d1)} \begin{pmatrix} A^{[1]} \\ A^{[2]} \end{pmatrix} \stackrel{*}{\Rightarrow} \begin{pmatrix} a_{l_1+1} \cdots a_{r_1} \\ a_{l_2+1} \cdots a_{r_2} \end{pmatrix}$$

where $r_1 \leq l_2$ or $r_2 \leq l_1$ (i.e., $a_{l_1+1} \cdots a_{r_1}$ and $a_{l_2+1} \cdots a_{r_2}$ are non-overlapping), if and only if there exist derivations

$$\text{(d2)} \begin{pmatrix} B^{[1]} \\ B^{[2]} \end{pmatrix} \stackrel{*}{\Rightarrow} \begin{pmatrix} a_{lb_1+1} \cdots a_{rb_1} \\ a_{lb_2+1} \cdots a_{rb_2} \end{pmatrix} \text{ and}$$

$$\text{(d3)} \begin{pmatrix} C^{[1]} \\ C^{[2]} \\ C^{[3]} \end{pmatrix} \stackrel{*}{\Rightarrow} \begin{pmatrix} a_{lc_1+1} \cdots a_{rc_1} \\ a_{lc_2+1} \cdots a_{rc_2} \\ a_{lc_3+1} \cdots a_{rc_3} \end{pmatrix}, \text{ where}$$

(b1) $l_1 = lb_1$, (b2) $r_1 = rc_2$, (b3) $l_2 = lc_3$, (b4) $r_2 = rc_1$, and
(c1) $rb_1 = lc_2$, (c2) $rc_3 = lb_2$, (c3) $rb_2 = lc_1$
for some lb_i, rb_i $(i = 1,2)$, lc_i, rc_i $(i = 1,2,3)$ in $\{0,1,\ldots,n\}$.

From already known derivations such as (d2) and (d3), the algorithm checks whether the conditions (c1) through (c3) hold by boolean matrices multiplication. If they are satisfied, then the algorithm obtains the information on the longer derivation (d1). More precisely, the algorithm performs the following (P1) through (P4).

(P1) Before analyzing the input string w, construct linear lists *end_with_B*(p), *alter_B*(p), *end_with_C*(p), *alter_C*(p) and *end_of_A*(p) for each rule $p \in P$. Each list has no duplicate element. Possible elements in these lists are $\mathbf{lB}_1, \ldots, \mathbf{lB}_{d(B)}, \mathbf{rB}_1, \ldots, \mathbf{rB}_{d(B)}, \mathbf{lC}_1, \ldots, \mathbf{lC}_{d(C)}, \mathbf{rC}_1, \ldots, \mathbf{rC}_{d(C)}$. For example, $\mathbf{lB}_i$ (resp. $\mathbf{rB}_i$) stands for the left-end (resp. right-end) of the string derived from $B^{[i]}$. Let $\mathbf{L_B} = \{\mathbf{lB}_i | 1 \le i \le d(B)\}$, $\mathbf{R_B} = \{\mathbf{rB}_i | 1 \le i \le d(B)\}$ and $\mathbf{LR_B} = \mathbf{L_B} \cup \mathbf{R_B}$. Similarly, let $\mathbf{L_C} = \{\mathbf{lC}_i | 1 \le i \le d(C)\}$, $\mathbf{R_C} = \{\mathbf{rC}_i | 1 \le i \le d(C)\}$ and $\mathbf{LR_C} = \mathbf{L_C} \cup \mathbf{R_C}$. Sometimes we treat the lists as sets when the order of the elements in them are not significant. These lists are defined as follows.

(i) *end_with_B*(p) contains $\mathbf{lB}_i$ (resp. $\mathbf{rB}_i$) if and only if $B^{[i]}$ is the leftmost (resp. rightmost) symbol of some component in the rhs of p. *alter_B*(p) is defined as $\mathbf{LR_B}$ − *end_with_B*(p). For rule p in (∗1), *end_with_B*$(p) = [\mathbf{lB}_1]$ and *alter_B*$(p) = [\mathbf{rB}_1, \mathbf{lB}_2, \mathbf{rB}_2]$.

(ii) *end_with_C*(p) and *alter_C*(p) are defined in the same way as in (i). For rule p in (∗1), *end_with_C*$(p) = [\mathbf{rC}_2, \mathbf{lC}_3, \mathbf{rC}_1]$ and *alter_C*$(p) = [\mathbf{lC}_2, \mathbf{rC}_3, \mathbf{lC}_1]$.

(iii) For $1 \le i \le d(A)$, the i-th element of *end_of_A*(p) is a pair $(\mathbf{lB}_j/\mathbf{lC}_j, \mathbf{rB}_k/\mathbf{rC}_k)$ if and only if $B^{[j]}/C^{[j]}$ is the leftmost symbol of the i-th component in the rhs of p and $B^{[k]}/C^{[k]}$ is the rightmost symbol of the same component. For rule p in (∗1), *end_of_A*$(p) = [(\mathbf{lB}_1, \mathbf{rC}_2), (\mathbf{lC}_3, \mathbf{rC}_1)]$.

By Condition 1.3 in Lemma 2.1, we have:

Proposition 3.1: $|alter\_B(p)| = |alter\_C(p)|$. □

(P2) Construct two boolean matrices B_p and C_p from a current matrix M (which contains the information on sub-derivations such as the above (d2) and (d3)), *end_with_B*(p), *alter_B*(p), *end_with_C*(p) and *alter_C*(p). We will explain the construction by using rule p in (∗1). For d_j $(0 \le d_j \le n,\ 1 \le j \le q)$, let $\langle d_q \cdot d_{q-1} \cdots d_1 \rangle$ denote the $(n+1)$-ary number, i.e., let

$$\langle d_q \cdot d_{q-1} \cdots d_1 \rangle = \sum_{j=1}^{q} (d_j \times (n+1)^{j-1}).$$

From *end_with_B*$(p) = [\mathbf{lB}_1]$ and *alter_B*$(p) = [\mathbf{rB}_1, \mathbf{lB}_2, \mathbf{rB}_2]$, construct $\langle n \rangle \times \langle n \cdot n \cdot n \rangle$ matrix B_p such that $B_p[\langle lb_1 \rangle, \langle rb_1 \cdot lb_2 \cdot rb_2 \rangle] = 1$ iff $B \in M[lb_1, rb_1, lb_2, rb_2]$.

$\langle lb_1 \rangle \setminus \langle rb_1 \cdot lb_2 \cdot rb_2 \rangle$

	$\langle 0\cdot 0\cdot 0\rangle$	$\langle 0\cdot 0\cdot 1\rangle$	$\cdots$	$\cdots$	$\langle n\cdot n\cdot n\rangle$
$\langle 0\rangle$	·	·	·	·	·
$\langle 1\rangle$	·	·	·	·	·
$\langle n\rangle$	·	·	·	·	·

Figure 1: B_p

From *alter_C*$(p) = [\mathbf{lC}_2, \mathbf{rC}_3, \mathbf{lC}_1]$ and *end_with_C*$(p) = [\mathbf{rC}_2, \mathbf{lC}_3, \mathbf{rC}_1]$, construct $\langle n \cdot n \cdot n \rangle \times \langle n \cdot n \cdot n \rangle$ matrix C_p in the same way as B_p.

$\langle lc_2 \cdot rc_3 \cdot lc_1 \rangle \setminus \langle rc_2 \cdot lc_3 \cdot rc_1 \rangle$

	$\langle 0\cdot 0\cdot 0\rangle$	$\langle 0\cdot 0\cdot 1\rangle$	$\cdots$	$\cdots$	$\langle n\cdot n\cdot n\rangle$
$\langle 0\cdot 0\cdot 0\rangle$	·	·	·	·	·
$\langle 0\cdot 0\cdot 1\rangle$	·	·	·	·	·
⋮					
$\langle n\cdot n\cdot n\rangle$	·	·	·	·	·

Figure 2: C_p

Note that since $|alter\_B(p)| = |alter\_C(p)|$ by Proposition 3.1, the width of B_p and the height of C_p are equal, and hence $B_p \times C_p$ is well-defined. For example, if $B \overset{*}{\Rightarrow} (a_3a_4a_5, a_{13}a_{14})$ and $C \overset{*}{\Rightarrow} (a_{15}, a_6a_7, a_{10}a_{11}a_{12})$, then $B \in M[2,5,12,14]$ and $C \in M[14,15,5,7,9,12]$. We set $B_p[\langle 2 \rangle, \langle 5 \cdot 12 \cdot 14 \rangle] = 1$ and $C_p[\langle 5 \cdot 12 \cdot 14 \rangle, \langle 7 \cdot 9 \cdot 15 \rangle] = 1$.

(P3) Compute $A_p = B_p \times C_p$. Suppose that $B_p[\langle lb_1 \rangle, \langle rb_1 \cdot lb_2 \cdot rb_2 \rangle] = 1$ and $C_p[\langle lc_2 \cdot rc_3 \cdot lc_1 \rangle, \langle rc_2 \cdot lc_3 \cdot rc_1 \rangle] = 1$. If $\langle rb_1 \cdot lb_2 \cdot rb_2 \rangle = \langle lc_2 \cdot rc_3 \cdot lc_1 \rangle$, or equivalently, the forementioned conditions (c1) through (c3) holds, then $A_p[\langle lb_1 \rangle, \langle rc_2 \cdot lc_3 \cdot rc_1 \rangle] = 1$. In the above example, $B_p[\langle 2 \rangle, \langle 5 \cdot 12 \cdot 14 \rangle] = 1$ and $C_p[\langle 5 \cdot 12 \cdot 14 \rangle, \langle 7 \cdot 9 \cdot 15 \rangle] = 1$.

$$A_p = B_p \times C_p = \langle 2 \rangle \overset{\langle 5\cdot 12\cdot 14\rangle}{\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}} \times \overset{\langle 7\cdot 9\cdot 15\rangle}{\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}} \langle 5 \cdot 12 \cdot 14 \rangle$$

$$= \langle 2 \rangle \overset{\langle 7\cdot 9\cdot 15\rangle}{\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}}$$

Therefore, $A_p[\langle 2 \rangle, \langle 7 \cdot 9 \cdot 15 \rangle] = 1$.

(P4) By using A_p, update M. The i-th element $(\mathbf{lB}_j/\mathbf{lC}_j, \mathbf{rB}_k/\mathbf{rC}_k)$ of *end_of_A*(p) represents that the left-end of the string derived from $A^{[i]}$ is the left-end of the string derived from $B^{[j]}/C^{[j]}$ and the right-end of the string derived from $A^{[i]}$ is the right-end of the string derived from $B^{[k]}/C^{[k]}$. For p in (∗1), *end_of_A*$(p) = [(\mathbf{lB}_1, \mathbf{rC}_2), (\mathbf{lC}_3, \mathbf{rC}_1)]$. So, if $A_p[\langle 2 \rangle, \langle 7 \cdot 9 \cdot 15 \rangle] = 1$, then we add A to $M[2,7,9,15]$ and we know $A \overset{*}{\Rightarrow} (a_3 \cdots a_7, a_{10} \cdots a_{15})$.

Here, we present the recognition algorithm. Let $G = (N, T, P, S)$ be an MCFG. For a nonterminal $A \in N$, a $2d(A)$-tuple $v = (l_1, r_1, \ldots, l_{d(A)}, r_{d(A)})$ of integers in $\{0,1,\ldots,n\}$ is called a position vector for A. Let V_A

be the set $\{(l_1, r_1, \ldots, l_{d(A)}, r_{d(A)}) | \ l_i, r_i \in \{0, 1, \ldots, n\}$ $(1 \leq i \leq d(A))\}$ of all position vectors for A.

procedure *MAIN*
$G = (N, T, P, S)$: an MCFG, $m = d(G)$
Input:

$w = a_1 a_2 \cdots a_n$ $(a_i \in T, 1 \leq i \leq n)$: an input string

Variable:

M : a $2m$-dimensional matrix of size $\underbrace{(n+1) \times (n+1) \times \cdots \times (n+1)}_{2m}$ whose entries are subsets of N

1. Set all the entries of M to $\emptyset$;
2. **for** $i = 1$ **to** n
   for each terminating rule $X_j \to a_i$
   add X_j to $M[i-1, i]$;
3. **repeat** (3.1) n times
   (3.1) **for** each nonterminating rule $p : A \to f[B, C]$
   (a) $CONSTRUCT\_B_pC_p(p)$;
   (b) compute $A_p = B_p \times C_p$;
   (c) $TRANS\_A_p\_TO\_M(p)$;
4. **if** $S \in M[0, n]$ **then** accept **else** reject
end of procedure *MAIN*

Definition 3.1: Let $p : A \to f[B, C]$ be a nonterminating rule and let

$\alpha = (l_1, r_1, \ldots, l_{d(A)}, r_{d(A)}) \in V_A$,
$\beta = (lb_1, rb_1, \ldots, lb_{d(B)}, rb_{d(B)}) \in V_B$,
$\gamma = (lc_1, rc_1, \ldots, lc_{d(C)}, rc_{d(C)}) \in V_C$.
Also let $end\_with\_B(p) = [s_1, \ldots, s_h]$,
$alter\_B(p) = [s_{h+1}, \ldots, s_{2d(B)}]$,
$end\_with\_C(p) = [t_1, \ldots, t_k]$,
$alter\_C(p) = [t_{k+1}, \ldots, t_{2d(C)}]$,
$end\_of\_A(p) = [(u_1, v_1), \ldots, (u_{d(A)}, v_{d(A)})]$

where $s_i \in \mathbf{LR_B}$ $(1 \leq i \leq 2d(B))$, $t_i \in \mathbf{LR_C}$ $(1 \leq i \leq 2d(C))$, $u_i \in \mathbf{L_B} \cup \mathbf{L_C}$, $v_i \in \mathbf{R_B} \cup \mathbf{R_C}$ $(1 \leq i \leq d(A))$.
We define mappings φ_β and φ_γ as
$\varphi_\beta(\mathbf{lB}_i) = lb_i$, $\varphi_\beta(\mathbf{rB}_i) = rb_i$ $(1 \leq i \leq d(B))$ and
$\varphi_\gamma(\mathbf{lC}_i) = lc_i$, $\varphi_\gamma(\mathbf{rC}_i) = rc_i$ $(1 \leq i \leq d(C))$. Also define

$\Phi_{p,B}(\beta) = \langle \varphi_\beta(s_1) \cdots \varphi_\beta(s_h) \rangle$,
$\Psi_{p,B}(\beta) = \langle \varphi_\beta(s_{h+1}) \cdots \varphi_\beta(s_{2d(B)}) \rangle$,
$\Phi_{p,C}(\gamma) = \langle \varphi_\gamma(t_{k+1}) \cdots \varphi_\gamma(t_{2d(C)}) \rangle$,
$\Psi_{p,C}(\gamma) = \langle \varphi_\gamma(t_1) \cdots \varphi_\gamma(t_k) \rangle$,
$\Phi_{p,A}(\alpha) = \langle d_{11} \cdots d_{1h} \rangle$, $\Psi_{p,A}(\alpha) = \langle d_{21} \cdots d_{2k} \rangle$,

where $d_{1i} = l_j$ (resp. r_j) if $s_i = u_j$ (resp. v_j) for some $1 \leq j \leq d(A)$,
$d_{2i} = l_j$ (resp. r_j) if $t_i = u_j$ (resp. v_j) for some $1 \leq j \leq d(A)$. □

Example 3.1 : Consider rule p in (*1) again. Let $\beta = (2, 5, 12, 14)$ and $\gamma = (14, 15, 5, 7, 9, 12)$. In this case, $\varphi_\beta(\mathbf{lB}_1) = 2$, $\varphi_\beta(\mathbf{rB}_1) = 5$, $\varphi_\beta(\mathbf{lB}_2) = 12$, $\varphi_\beta(\mathbf{rB}_2) = 14$, $\varphi_\gamma(\mathbf{lC}_1) = 14$, $\varphi_\gamma(\mathbf{rC}_1) = 15$, $\varphi_\gamma(\mathbf{lC}_2) = 5$, $\varphi_\gamma(\mathbf{rC}_2) = 7$, $\varphi_\gamma(\mathbf{lC}_3) = 9$ and $\varphi_\gamma(\mathbf{rC}_3) = 12$. So, $\Phi_{p,B}(\beta) = \langle 2 \rangle$, $\Psi_{p,B}(\beta) = \langle 5 \cdot 12 \cdot 14 \rangle$, $\Phi_{p,C}(\gamma) = \langle 5 \cdot 12 \cdot 14 \rangle$ and $\Psi_{p,C}(\gamma) = \langle 7 \cdot 9 \cdot 15 \rangle$. Also let $\alpha = (2, 7, 9, 15)$. Since $end\_of\_A(p) = [(\mathbf{lB}_1, \mathbf{rC}_2), (\mathbf{lC}_3, \mathbf{rC}_1)]$, we have $\Phi_{p,A}(\alpha) = \langle 2 \rangle$ and $\Psi_{p,A}(\alpha) = \langle 7 \cdot 9 \cdot 15 \rangle$. □

The following procedure constructs 2-dimensional boolean matrices B_p and C_p from $end\_with\_B(p)$, $alter\_B(p)$, $end\_with\_C(p)$ and $alter\_C(p)$.

procedure $CONSTRUCT\_B_pC_p(p)$
for each $\beta \in V_B$
   if $B \in M[\beta]$ **then** $B_p[\Phi_{p,B}(\beta), \Psi_{p,B}(\beta)] = 1$
   else $B_p[\Phi_{p,B}(\beta), \Psi_{p,B}(\beta)] = 0$;
for each $\gamma \in V_C$
   if $C \in M[\gamma]$ **then** $C_p[\Phi_{p,C}(\gamma), \Psi_{p,C}(\gamma)] = 1$
   else $C_p[\Phi_{p,C}(\gamma), \Psi_{p,C}(\gamma)] = 0$;
end of procedure $CONSTRUCT\_B_pC_p(p)$

By the definition of $end\_with\_B(p)$ and $alter\_B(p)$,

$$\{\varphi_\beta(s_1), \ldots, \varphi_\beta(s_{2d(B)})\} = \{lb_1, rb_1, \ldots, lb_{d(B)}, rb_{d(B)}\}.$$

Therefore, $\{(\Phi_{p,B}(\beta), \Psi_{p,B}(\beta)) \mid \beta \in V_B\}$ has one-to-one correspondence with V_B and hence $CONSTRUCT\_B_pC_p(p)$ assigns a value exactly once for each entry of B_p. The same property holds for C.
The following procedure translates the address of each entry of A_p which contains 1 to the address of an entry of M, and adds A to that entry of M.

procedure $TRANS\_A_p\_TO\_M(p)$
for each $\alpha = (l_1, r_1, \ldots, l_{d(A)}, r_{d(A)}) \in V_A$
   if $A_p[\Phi_{p,A}(\alpha), \Psi_{p,A}(\alpha)] = 1$ **and**
   $\{l_1 + 1, \ldots, r_1\}, \{l_2 + 1, \ldots, r_2\}, \ldots,$
   $\{l_{d(A)} + 1, \ldots, r_{d(A)}\}$ are not overlapping
   then add A to $M[\alpha]$;
end of procedure $TRANS\_A_p\_TO\_M(p)$

We can prove the following lemma which states the correctness of the algorithm. A proof is found in [8].

Lemma 3.2: The followings are equivalent for each nonterminal A after Step 3 is executed:
(1) $A \stackrel{*}{\Rightarrow} (a_{l_1+1} \cdots a_{r_1}, \ \ldots, \ a_{l_{d(A)}+1} \cdots a_{r_{d(A)}})$.
(2) $A \in M[l_1, r_1, \ldots, l_{d(A)}, r_{d(A)}]$. □

Theorem 3.3: When algorithm *MAIN* terminates, $S \stackrel{*}{\Rightarrow} a_1 \cdots a_n$ if and only if $S \in M[0, n]$. □

4 Complexity

In this section, we will analyze the time complexity of the algorithm. Let $G = (N, T, P, S)$ be an MCFG with $d(G) = m$ and $e(G) = e$. Obviously, linear lists $end\_with\_B(p)$, $alter\_B(p)$, $end\_with\_C(p)$, $alter\_C(p)$ and $end\_of\_A(p)$ $(p \in P)$ can be constructed in $O(1)$ time. Procedures $CONSTRUCT\_B_pC_p(p)$ and $TRANS\_A_p\_TO\_M(p)$ can be executed in $O(n^{2m})$ time since the numbers of entries of A_p, B_p, C_p and M are all at most $O(n^{2m})$. Next, we will evaluate the time needed for computing $B_p \times C_p$. Assume that the sizes of boolean matrices B_p and C_p are $(n+1)^q \times (n+1)^r$ and $(n+1)^r \times (n+1)^s$, respectively. Let t be $\min\{q, r, s\}$ and u, v be the rest of them. $B_p \times C_p$ can be computed in time

$$(n+1)^{u-t} \cdot (n+1)^{v-t} \cdot M((n+1)^t) = (n+1)^{(q+r+s)-3t} \cdot M((n+1)^t). \quad (*2)$$

Definition 4.1 : For a rule $p: A \rightarrow f[B,C]$, define the number $i(p)$, called the *multiplication unit* of p, as $d(A)+d(B)+d(C)-2 \cdot \max\{d(A), d(B), d(C)\}$. □

Lemma 4.1: $e(p) = q+r+s$ and $i(p) = \min\{q,r,s\}$.
Proof By the construction of B_p and C_p,

$$q+r = 2d(B),\ r+s = 2d(C),\ q+s = 2d(A)\ \ (*3)$$

and hence $q+r+s = d(A)+d(B)+d(C)$. Moreover, $e(p) = d(A)+d(B)+d(C)$ by the definition of degree, Conditions 1.3(b)(c) in Lemma 2.1 and right-linearity. Therefore, $e(p) = q+r+s$. By (*3),

$$q = d(A)+d(B)-d(C), r = -d(A)+d(B)+d(C),$$
$$s = d(A)-d(B)+d(C),$$

which imply $\min\{q,r,s\} = d(A)+d(B)+d(C)-2 \cdot \max\{d(A),d(B),d(C)\} = i(p)$. □

Note that $i(p)$ satisfies $0 \leq i(p) \leq m$. By (*2), Lemma 4.1 and [1], $B_p \times C_p$ can be computed in $O(n^{e(p)-3i(p)} \cdot M(n^{i(p)})) < O(n^{e(p)-0.624i(p)})$ time. Hence, (a) through (c) of Step 3.1 in $MAIN$ can be executed in $\max\{O(n^{2m}),\ O(n^{e(p)-0.624i(p)})\}$ time for a rule p.

Let p' be a rule such that $e(p') - 0.624i(p') = \max\{e(p) - 0.624i(p) \mid p \in P\}$, and let $e' = e(p')$, $i' = i(p')$. Since the number of rules is a constant, Step 3.1 can be executed in $\max\{O(n^{2m}),\ O(n^{e'-0.624i'})\}$ time. Moreover, the following lemma holds [8].

Lemma 4.2 : $i' \geq 1$ and $e' - 0.624i' > 2m$. □

By Lemma 4.2, Step 3.1 can be executed in $O(n^{e'-0.624i'})$ time. As Step 3 dominates algorithm $MAIN$, it can be executed in $O(n^{e'-0.624i'+1})$ time.

5 Conclusions

In this paper, we proposed a recognition algorithm for MCFLs which uses multiplication of boolean matrices. Let p' be a rule such that $e(p') - 0.624i(p') = \max\{e(p) - 0.624i(p) \mid p \in P\}$, and let $e' = e(p')$, $i' = i(p')$. Its time complexity is $O(n^{e'-0.624i'+1})$, where n is the length of an input string. We compare the time complexity of the algorithm proposed in this paper with $O(n^e)$, which is the time complexity of the known fastest algorithm. If $e' \neq e$ i.e. $e \geq e'+1$, then $O(n^e) > O(n^{e'-0.624i'+1})$ by Lemma 4.2. Consider the case that $e' = e$. If $i' \geq 2$, then $O(n^e) > O(n^{e'-0.624i'+1})$. Only when $e' = e$ and $i' = 1$, $O(n^{e'-0.624i'+1}) = O(n^{e+0.376}) > O(n^e)$ and the known algorithm is faster.

In [10] and [7], a technique is proposed which reduces the number of multiplication of boolean matrices by recursive multiplication of submatrices. By this technique, the time complexity of the algorithm in [10] is improved from $O(nM(n))$ to $O(M(n))$ and the algorithm in [7] is improved from $O(n(M(n^2))$ to $O(M(n^2))$. If this technique can be applied to our algorithm, then the time complexity of ours may be improved from $O(n^{e'-0.624i'+1})$ to $O(n^{e'-0.624i'})$.

Parallel multiple context-free grammars (*PMCFGs*) were introduced as an extension of MCFGs [3][9]. Recently, we extend the recognition algorithm proposed in this paper to recognize the class of languages generated by PMCFGs [12]. By this algorithm, some MCFLs are recognized faster than the recognition algorithm proposed in this paper.

References

[1] D. Coppersmith and S. Winograd: "Matrix Multiplication via Arithmetic Progressions," Proc. 19th Annual ACM Symp. Theory of Computing, 1-6, 1987, also in J. Symbolic Computation, **9**, pp.251-280, 1990.

[2] A. K. Joshi, L. Levy and M. Takahashi: "Tree Adjunct Grammars," J. Comput. System Sci., **10**, 1, pp.136-163, 1975.

[3] T. Kasami, H. Seki and M. Fujii: "Generalized Context-Free Grammars and Multiple Context-Free Grammars," Trans. IEICE, **J71-D-I**, 5, pp.758-765, 1988 (*in Japanese*).

[4] T. Kasami, H. Seki and M. Fujii: "On the Membership Problem for Head Languages and Multiple Context-Free Languages," Trans. IEICE, **J71-D-I**, 6, pp.935-941, 1988 (*in Japanese*).

[5] R. Nakanishi: "A Study on Some Context-Free Grammar Based Formalisms," Ph. D. dissertation, Osaka University, 1994.

[6] C. J. Pollard: "Generalized Phrase Structure Grammars, Head Grammars, and Natural Language," Ph. D. dissertation, Stanford University, 1984.

[7] S. Rajasekaran and S. Yooseph: "TAL Recognition in $O(M(n^2))$ Time," Proc. 33rd Annual Meeting of Assoc. for Comput. Ling., pp.166-173, 1995.

[8] K. Takada: "An Efficient Recognition Algorithm for Multiple Context-Free Languages," Master's Thesis, Osaka University, 1996.

[9] H. Seki, T. Matsumura, M. Fujii and T. Kasami: "On Multiple Context-Free Grammars," Theoretical Computer Science, **88**, pp.191-229, 1991.

[10] L. G. Valiant: "General Context-Free Recognition in Less than Cubic Time," J. Comput. and System Sci., **10**, pp.308-315, 1975.

[11] K. Vijay-Shanker, D. J. Weir and A. K. Joshi: "Characterizing Structural Descriptions Produced by Various Grammatical Formalisms," Proc. 25th Annual Meeting of Assoc. Comput. Ling., pp.104-111, 1987.

[12] H. Nii, R. Nakanishi and H. Seki: "An Efficient Recognition Algorithm for Parallel Multiple Context-Free Languages", IEICE Technical Report, **COMP96-82**, pp.31-39, 1997(*in Japanese*).

Solving the correct-prefix property for TAGs

Mark-Jan Nederhof *

University of Groningen
Faculty of Arts — Humanities Computing
P.O. Box 716
NL-9700 AS Groningen
The Netherlands
markjan@let.rug.nl

Abstract

We present a new upper bound for the computational complexity of the parsing problem for TAGs, under the constraint that input is read from left to right in a way that errors in the input are observed as soon as possible, which is called the *correct-prefix property.*

The former upper bound was $\mathcal{O}(n^9)$, which we now improve to $\mathcal{O}(n^6)$, which is the same as that of practical parsing algorithms for TAGs *without* the additional constraint of the correct-prefix property. Thereby we show that the correct-prefix property does not require significant additional costs.

1 Introduction

Traditionally, parsers and recognizers for regular and context-free languages process input from left to right. If a syntax error occurs in the input they often detect that error immediately after its position is reached. The position of the syntax error can be defined as the last input symbol of the shortest prefix which cannot be extended to be a correct sentence in the language L.

In formal notation, this prefix for a given erroneous input $w \notin L$ is defined as the string va, where $w = vax$, some x, such that $vy \in L$, for some y, but $vaz \notin L$, for any z. (The symbols v, w, ... denote strings, and a denotes an input symbol.) The occurrence of a in w indicates the error position.

If the error is detected as soon as it is reached, then all prefixes of the input that have been processed at preceding stages are *correct* prefixes, or more precisely, they are prefixes of some correct strings in the language. Hence, we speak of the *correct-prefix property.*[1]

For context-free and regular languages, the correct-prefix property can be enforced without additional costs of space or time. Strangely enough, it has been claimed by [SW95] that this property is problematic for the weakly context-sensitive languages represented by tree-adjoining grammars (TAGs): the best practical parsing algorithms for TAGs have time complexity $\mathcal{O}(n^6)$ [VSJ85] (see [Sat94] for lower theoretical upper bounds), whereas the only published algorithm with the correct-prefix property, viz. that in [SJ88], has complexity $\mathcal{O}(n^9)$.

In this paper we present an algorithm that fulfils the correct-prefix property and operates in $\mathcal{O}(n^6)$ time. This algorithm merely recognizes input, but it can be extended to be a parsing algorithm, with the ideas from [Sch91].

2 Notation

For a good introduction to TAGs, the reader is referred to [Jos87]. In this section we merely summarize our notation.

A tree-adjoining grammar is a 4-tuple (Σ, NT, I, A), where Σ is the set of *terminals*,

*This research was carried out within the framework of the Priority Programme Language and Speech Technology (TST). The TST-Programme is sponsored by NWO (Dutch Organization for Scientific Research).

[1] We adopt this term from [SSS88]. In some publications, the term *valid prefix property* is used.

I is the set of *initial trees* and A is the set of *auxiliary trees*. We refer to the trees in $I \cup A$ as *elementary trees*. The set NT, the set of *non-terminals*, does not play any role in this paper.

For each leaf M in an elementary tree, except when it is a foot, we define $label(M)$ to be the label of the node, which is either a terminal from Σ or the empty string ϵ. For each other node M we define $Adjunct(M)$ as the set of auxiliary trees that can be adjoined at M. For each non-leaf node M we define $children(M)$ as the list of daughter nodes.

We refer to the root of an elementary tree t as R_t. We refer to the foot of an auxiliary tree t as F_t.

For technical reasons, we assume an additional node for each elementary tree t, which we denote by $\top$. This node has only one daughter, viz. the actual root node R_t. We also assume an additional node for each auxiliary tree t, which we denote by $\bot$. This is the daughter of the actual foot node F_t.

The input to the recognition algorithm is given by $a_1 a_2 \cdots a_n$, where n is the length of the input.

3 The algorithm

The algorithm operates by means of least fixed-point iteration: a table is gradually filled with elements derived from other elements, until no more new ones can be found. A certain collection of *steps* indicate how table elements are to be derived from others.

For the description of the steps we use a pseudo-formal notation. Each step consists of a list of antecedents and a consequent. The antecedents are the conditions under which an incarnation of the step is executed. The consequent is a new table element that the step then adds to the parse table, unless of course it is already present. An antecedent may be a table element, in which case the condition that it represents is membership in the table.

The main table elements, or *items*, are 6-tuples $[h,\ N \to \alpha \bullet \beta,\ i,\ j,\ f_1,\ f_2]$. Here, N is a node from some elementary tree, and $\alpha\beta$ is the list of the daughter nodes of N. The daughters in α, together generate the input from position i to j. The whole tree generates input from position h onwards.

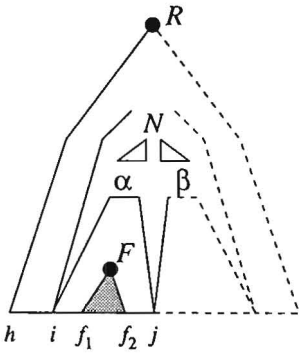


Figure 1: An item $[h,\ N \to \alpha \bullet \beta,\ i,\ j,\ f_1,\ f_2]$

Internal in the tree there may be adjunctions. Furthermore, the tree in which N occurs may itself be an auxiliary tree, in which case it is adjoined in another tree. Then, the foot may be dominated by one of the daughters in α, and the part of the input generated by the foot is given by the interval (f_1, f_2). When the tree is not an auxiliary tree, or when the foot is not dominated by one of the daughters in α, then f_1 and f_2 both have the dummy value "–".

See Figure 1 for a pictorial representation of the meaning of items. The solid lines indicate what has been established; the dashed lines indicate what is merely predicted. The shaded subtree indicates the lower half of a tree in which the present tree has been adjoined. This tree may not exist, as explained above. R and F are the root and foot of the elementary tree to which N belongs; F may not exist.

There is one special kind of item, with only 5 fields instead of 6. This is used as intermediate result in the adjunctor steps to be discussed in Section 3.5.

3.1 Initializer

The initializer step predicts initial trees t starting at position 0; see Figure 2.

Init

$$\frac{t \in I}{\vdash\ [0,\ \top \to \bullet\ R_t,\ 0,\ 0,\ -,\ -]}$$

3.2 Scanner

The scanner steps try to shift the dot rightward in case the next node in line is labelled with a

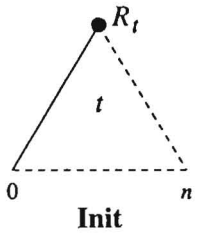


Figure 2: The initialization

terminal or ϵ, which means the node is a leaf but not a foot; see Figure 3.

Scan 1

$$\begin{array}{l} [h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2], \\ label(M) = a_{j+1} \\ \vdash \\ [h,\ N \to \alpha M \bullet \beta,\ i,\ j+1,\ f_1,\ f_2] \end{array}$$

Scan 2

$$\begin{array}{l} [h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2], \\ label(M) = \epsilon \\ \vdash \\ [h,\ N \to \alpha M \bullet \beta,\ i,\ j,\ f_1,\ f_2] \end{array}$$

3.3 Predictor

The first predictor step predicts a fresh occurrence of an auxiliary tree t, indicated by a shaded area in Figure 4. The second predicts a list of daughters γ lower down in the tree, abstaining from adjunction at the current node M. The third predicts the lower half of a tree in which the present tree t was adjoined.

Pred 1

$$\begin{array}{l} [h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2], \\ t \in Adjunct(M) \\ \vdash \\ [j,\ \top \to \bullet\ R_t,\ j,\ j,\ -,\ -] \end{array}$$

Pred 2

$$\begin{array}{l} [h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2], \\ children(M) = \gamma, \\ \text{adjunction not obligatory at } M \\ \vdash \\ [h,\ M \to \bullet\ \gamma,\ j,\ j,\ -,\ -] \end{array}$$

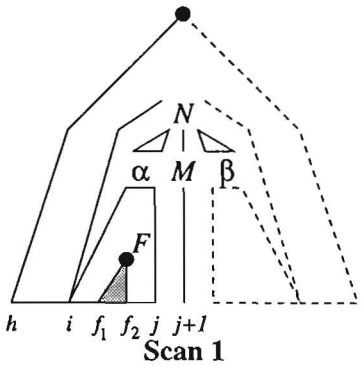


Figure 3: The first scanner step

Pred 3

$$\begin{array}{l} [j,\ F_t \to \bullet \perp,\ k,\ k,\ -,\ -], \\ [h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2], \\ t \in Adjunct(M), \\ children(M) = \gamma \\ \vdash \\ [h,\ M \to \bullet\ \gamma,\ k,\ k,\ -,\ -] \end{array}$$

3.4 Completor

The first completor step completes recognition of the lower half of a tree in which an auxiliary tree t was adjoined, and asserts recognition of the foot of t; see Figure 5. The second and third completer steps complete recognition of a list of daughter nodes γ, and initiate recognition of the list of nodes β to the right of the mother node of γ.

Comp 1

$$\begin{array}{l} [h,\ M \to \gamma \bullet,\ k,\ l,\ f_1',\ f_2'], \\ t \in Adjunct(M), \\ [j,\ F_t \to \bullet \perp,\ k,\ k,\ -,\ -], \\ [h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2] \\ \vdash \\ [j,\ F_t \to \perp \bullet,\ k,\ l,\ k,\ l] \end{array}$$

Comp 2

$$\begin{array}{l} [h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ -,\ -], \\ M \text{ dominates foot of tree}, \\ [h,\ M \to \gamma \bullet,\ j,\ k,\ f_1,\ f_2] \\ \vdash \\ [h,\ N \to \alpha M \bullet \beta,\ i,\ k,\ f_1,\ f_2] \end{array}$$

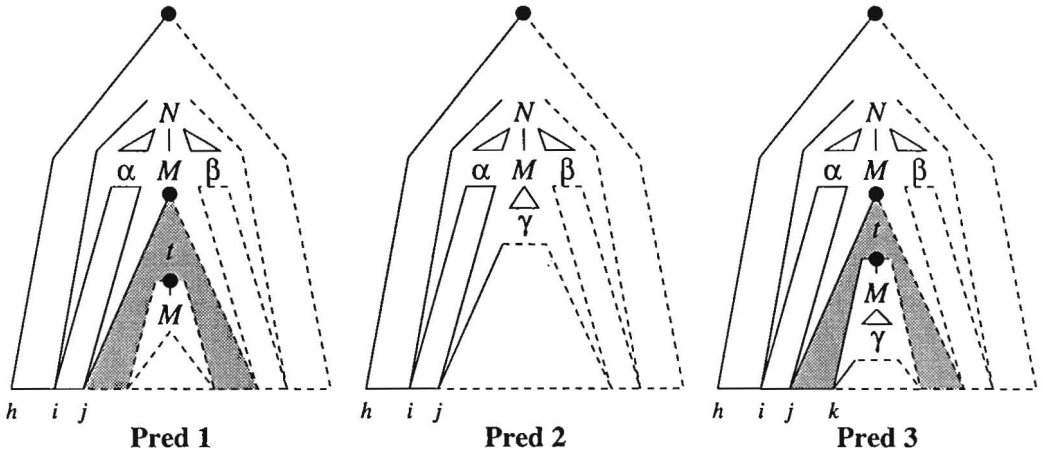


Figure 4: The three predictor steps

Comp 3

$$\begin{array}{l}[h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2],\\ M \text{ does not dominate foot of tree,}\\ [h,\ M \to \gamma \bullet,\ j,\ k,\ -,\ -]\\ \vdash\\ [h,\ N \to \alpha M \bullet \beta,\ i,\ k,\ f_1,\ f_2]\end{array}$$

3.5 Adjunctor

The adjunctor steps perform the actual recognition of an adjunction of an auxiliary tree t in another tree at some node M. The first adjunctor step deals with the case that other tree is again adjoined in a third tree (the two darkly shaded areas in Figure 6) and M dominates the foot node. The second adjunctor step deals with the case that either the other tree is an initial tree, or has the foot elsewhere, i.e. not dominated by M.

The two respective cases of adjunction are realised by step **Adj 0** plus step **Adj 1**, and by step **Adj 0** plus step **Adj 2**. The auxiliary step **Adj 0** introduces items of a somewhat different form than considered up to now, viz. $[M \to \gamma \bullet,\ j,\ k,\ f_1',\ f_2']$. The interpretation is suggested in Figure 7: at M a tree has been adjoined. The adjoined tree and the lower half of the tree that M occurs in together generate the input from j to k. In the case that M dominates a foot note, as suggested in the figure, f_1' and f_2' have a value other than "–".

The reason that the auxiliary step is needed for each case is that otherwise 8 variables would be involved in one step, resulting in a complexity of $\mathcal{O}(n^8)$. See Section 5 for more explanation.

Adj 0

$$\begin{array}{l}[j,\ \top \to R_t \bullet,\ j,\ k,\ f_1,\ f_2],\\ [h,\ M \to \gamma \bullet,\ f_1,\ f_2,\ f_1',\ f_2'],\\ t \in \mathit{Adjunct}(M)\\ \vdash\\ [M \to \gamma \bullet,\ j,\ k,\ f_1',\ f_2']\end{array}$$

Adj 1

$$\begin{array}{l}[M \to \gamma \bullet,\ j,\ k,\ f_1',\ f_2'],\\ M \text{ dominates foot of tree } t',\\ [h,\ F_{t'} \to \bot \bullet,\ f_1',\ f_2',\ f_1',\ f_2'],\\ [h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ -,\ -]\\ \vdash\\ [h,\ N \to \alpha M \bullet \beta,\ i,\ k,\ f_1',\ f_2']\end{array}$$

Adj 2

$$\begin{array}{l}[M \to \gamma \bullet,\ j,\ k,\ -,\ -],\\ [h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1',\ f_2']\\ \vdash\\ [h,\ N \to \alpha M \bullet \beta,\ i,\ k,\ f_1',\ f_2']\end{array}$$

4 Properties

The first claim we make about the algorithm pertains to its correctness as recognizer:

Claim 1 *After completion of the algorithm, the item* $[0,\ \top \to R_t \bullet,\ 0,\ n,\ -,\ -]$, *for some* $t \in I$, *is in the table if and only if the input is in the language described by the grammar.*

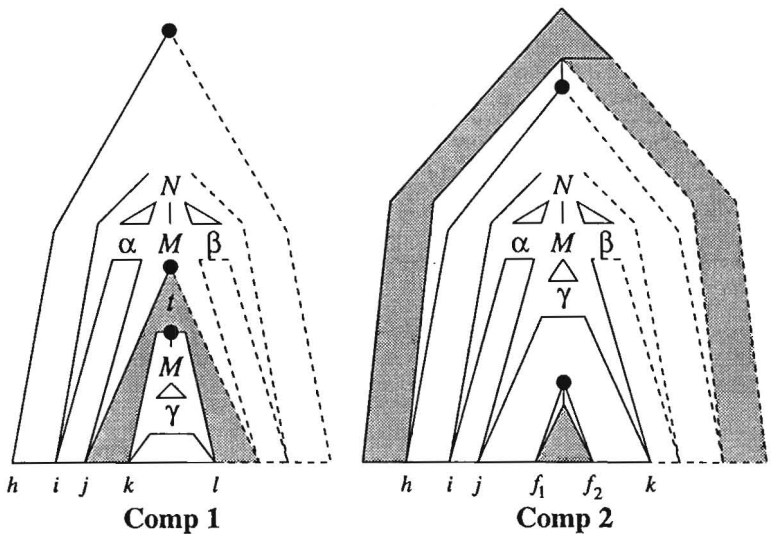


Figure 5: Two of the completor steps

The intuition behind the proof of the "if" part is that for trees constructed from the grammar we can indicate a left-to-right depth-first tree traversal that is matched by corresponding steps of the algorithm. When nodes are visited, corresponding items are added to the table, as suggested earlier by Figure 1.

The "only if" part can be proven along with the second claim:

Claim 2 *The algorithm satisfies the correct-prefix property, provided the grammar is reduced.*

A TAG is reduced if it does not contain any elementary trees that cannot be part of any parse tree.[2]

The proof requires us to refine the interpretation of items $[h,\ N \rightarrow \alpha \bullet \beta,\ i,\ j,\ f_1,\ f_2]$. Apart from the requirements suggested in Figure 1, we require that the elementary tree is part of a parse tree, of which the part to the left of that elementary tree generates the input from position 0 to h. Given the correctness of this interpretation, the second claim follows: if the input up to position i has been read, then there is a string y such that $a_1 \cdots a_i y$ is in the language. This y is the concatenation of strings generated by β, by the nodes to the right of N, etc. For the "only if" part of the first claim we consider the interpretation of $[0,\ \top \rightarrow R_t \bullet,\ 0,\ n,\ -,\ -]$.

[2] One reason why an auxiliary tree might not be a part of any parse tree is that at some node it may have obligatory adjunction of itself, leading to "infinite adjunction".

The interpretation of items can be proven correct by verifying that if the items of the antecedents of some step satisfy the interpretation, then so does the item of the consequent. A slight technical problem is caused by the obligatory adjunctions. This can be solved by noting that for each node with an obligatory adjunction some finite adjunction at that node exists, since the grammar is reduced.

The full proofs are straightforward but tedious. Furthermore, they do not depart in any significant way from those for existing recognition algorithms for TAGs [VSJ85, SJ88, Lan88, Sch91], and therefore including the full proofs here does not seem desirable.

5 Complexity

The steps presented in pseudo-formal notation in Section 3 can easily be composed into an actual algorithm [SSP95]. As a first approximation, the complexity of such an algorithm is given by $\mathcal{O}(n^p)$, where p is the largest number of input positions in any antecedent. A more refined analysis excludes the variables for input positions that only occur once in a step, the so called *don't-cares*. This is because an implicit intermediate step $I \vdash I'$ may be applied that reduces an item I with q input positions to another item I' with $q' \leq q$ input positions, omitting the don't-cares. That reduced item I' then takes the place of I in the antecedent of the actual step.

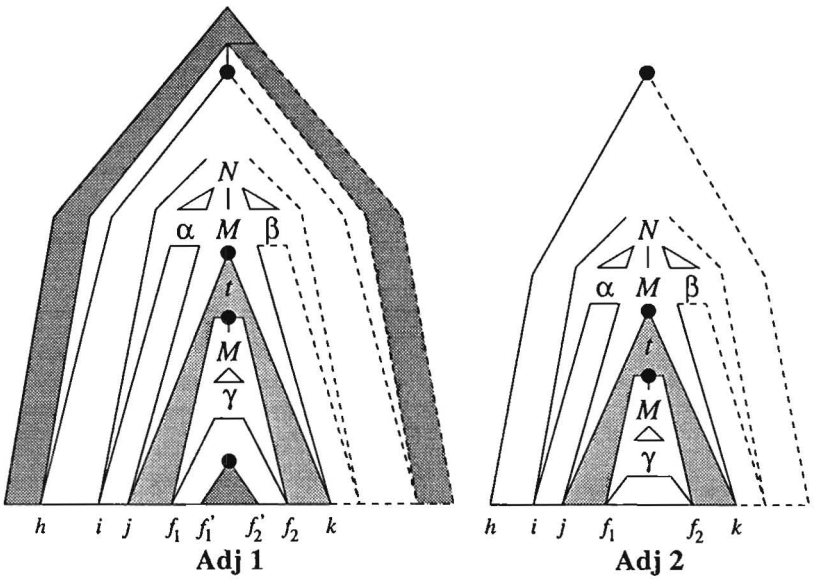


Figure 6: The two adjunctor steps, implicitly combined with **Adj 0**

For example, there are 9 variables in **Comp 1**, of which i, f_1, f_2, f_1', f_2' are all don't-cares, since they occur only once in that step. Therefore, the contribution of this step to the overall time-complexity is $\mathcal{O}(n^4)$ rather than $\mathcal{O}(n^9)$.

Under these considerations, the maximum number of relevant variables for input positions per step is 6. Thereby, the complexity of left-to-right recognition for TAGs under the constraint of the correct-prefix property is $\mathcal{O}(n^6)$.

In terms of the size of the grammar, the complexity is $\mathcal{O}(|G|^2)$, since at most 2 elementary trees are simultaneously considered in a single step.

6 Further research

In [SVS90] an attempt was made to add further sophistication to left-to-right parsing for TAGs: the idea of LR parsing, as usually applied to grammars with an underlying context-free structure, was extended to TAGs.[3] For some TAGs, the parser is even deterministic; in fact, determinism was the primal objective of that work.

The comparison of our work with that in [SVS90] raises a few questions. The algorithm in the present paper operates in a top-down manner, being very similar to Earley's algorithm [Ear70], which is emphasised by the use of the "dotted" items. As shown in [NS94], a family of parsing algorithms (viz. top-down, left-corner, PLR, ELR, and LR parsing [Ned94]) can be carried over to head-driven parsing. An obvious question is whether such parsing techniques can also be used to produce variants of left-to-right parsing for TAGs. Thus, one may conjecture, for example, the existence of an LR-like parsing algorithm for arbitrary TAGs that operates in $\mathcal{O}(n^6)$ and that has the correct-prefix property.

The definition of such an algorithm is not at all straightforward. The additional benefit of LR parsing, in comparison to, for example, left-corner parsing, lies in the ability to process multiple grammar rules simultaneously. If this is to be carried over to TAGs, then one needs to decide in what way multiple elementary trees can be handled simultaneously. A straightforward combination of this objective with the mechanism we used to ensure the correct-prefix property does not seem useful, except for the most simple cases when a TAG contains many, almost

[3] It seems obvious that that algorithm is actually incorrect. It accepts input that is not in the language. This is because the action that matches the parts of an elementary tree to the "south" and "north-east" of an adjunction to that to the "north-west" is defective. The verification merely checks the *number* of terminals in the "north-west" part, which is insufficient to ensure that the same elementary tree is used. This observation does not seem to have appeared in print before, although it is very easy to demonstrate, and has been confirmed by personal communication with colleagues. I see no possibilities for a straightforward patch.

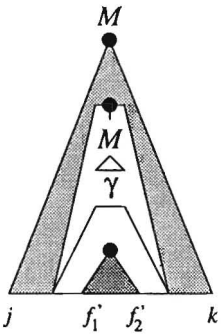


Figure 7: An item $[M \rightarrow \gamma \bullet, j, k, f_1', f_2']$

identical, elementary trees.

Therefore, further research is needed not only to precisely define such left-to-right algorithms for TAGs, but also to determine whether there are any benefits for practical grammars.

Acknowledgements

An error in a previous version of this paper was found and corrected with the help of Giorgio Satta.

References

[Ear70] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, February 1970.

[Jos87] A.K. Joshi. An introduction to tree adjoining grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 87–114. John Benjamins Publishing Company, Amsterdam, 1987.

[Lan88] B. Lang. The systematic construction of Earley parsers: Application to the production of $\mathcal{O}(n^6)$ Earley parsers for tree adjoining grammars. Unpublished paper, December 1988.

[Ned94] M.J. Nederhof. An optimal tabular parsing algorithm. In *32nd Annual Meeting of the ACL, Proceedings of the Conference*, pages 117–124, Las Cruces, New Mexico, USA, June 1994.

[NS94] M.J. Nederhof and G. Satta. An extended theory of head-driven parsing. In *32nd Annual Meeting of the ACL, Proceedings of the Conference*, pages 210–217, Las Cruces, New Mexico, USA, June 1994.

[Sat94] G. Satta. Tree-adjoining grammar parsing and Boolean matrix multiplication. *Computational Linguistics*, 20(2):173–191, 1994.

[Sch91] Y. Schabes. The valid prefix property and left to right parsing of tree-adjoining grammar. In *Proc. of the Second International Workshop on Parsing Technologies*, pages 21–30, Cancun, Mexico, February 1991.

[SJ88] Y. Schabes and A.K. Joshi. An Earley-type parsing algorithm for tree adjoining grammars. In *26th Annual Meeting of the ACL, Proceedings of the Conference*, pages 258–269, Buffalo, New York, June 1988.

[SSP95] S.M. Shieber, Y. Schabes, and F.C.N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36, 1995.

[SSS88] S. Sippu and E. Soisalon-Soininen. *Parsing Theory, Vol. I: Languages and Parsing*, EATCS Monographs on Theoretical Computer Science, volume 15. Springer-Verlag, 1988.

[SVS90] Y. Schabes and K. Vijay-Shanker. Deterministic left to right parsing of tree adjoining languages. In *28th Annual Meeting of the ACL, Proceedings of the Conference*, pages 276–283, Pittsburgh, Pennsylvania, USA, June 1990.

[SW95] Y. Schabes and R.C. Waters. Tree insertion grammar: A cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–513, 1995.

[VSJ85] K. Vijay-Shankar and A.K. Joshi. Some computational properties of tree adjoining grammars. In *23rd Annual Meeting of the ACL, Proceedings of the Conference*, pages 82–93, Chicago, Illinois, USA, July 1985.

Parametric Types for Typed Attribute Value Logic

(Preliminary Version)

Gerald Penn
SFB 340, Universität Tübingen
gpenn@sfs.nphil.uni-tuebingen.de

1 Motivation

Parametric polymorphism has been combined with inclusional polymorphism to provide natural type systems for Prolog ([DH88]), HiLog ([YFS92]) and constraint resolution languages ([Smo89]), and, more recently, HPSG-like grammars to classify lists and sets of linguistic objects ([PS94],Figure 1). This abstract summarizes work in progress on the incorporation of parametric types into the typed

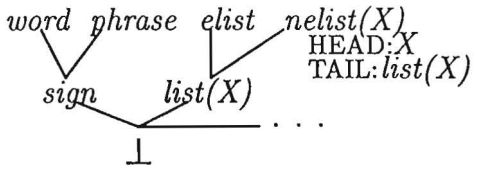


Figure 1: A fragment of the HPSG type signature.

attribute value logic of [Car92]. This logic is distinguished by its strong interpretation of *appropriateness*, a set of conditions which tell us which features an object of a given type can have, and which types a feature's value can have. Its interpretation, *total well-typedness*, says that every feature structure must have an appropriate value for all and only the appropriate features of its type. In contrast to other logics which are concerned with models of the feature terms themselves (e.g. SRL,[Kin89]), it takes feature structures themselves to represent partial information states obtained through the closure of inference procedures (e.g. total well-typing) over a description, relative to some type signature.

In this context, the relevant question to ask is what different kinds of information one can represent relative to a signature[1] with parametric types, than relative to a signature without them. This enquiry has yielded an interpretation of parametric types with several specific properties necessary to conform to their current usage by linguists and implementors who work with feature-based formalisms. What is at stake, however, is not just how to represent lists and sets in HPSG. Parametric types have a wide range of possible applications throughout knowledge representation, including HPSG.

Previous approaches have required that every parameter of a subtype should be a parameter of all of its supertypes; thus, it would not be possible to encode Figure 1 because $\bot \sqsubseteq \mathit{list(X)}$.[2] The present one eliminates this restriction by requiring the existence of a most general type (which [Car92]'s logic requires anyway), which is then used during type-checking and inferencing to interpret new parameters. All previous approaches deal only with fixed-arity terms; and none but one use a feature logic, with the one (CUF,[Dor92]) only permitting one parametric type, lists, which are simply hard-wired into that implementation. The present approach provides a generalization of appropriateness which permits both unrestricted parametricity and incremental feature introduction.

This enquiry has also provided a better understanding of the trade-off between using features or more refined typing, with or without parametric polymorphism, in a type signature — a degree of freedom with respect to which linguistic applications typically commit themselves on the basis of arbitrary criteria. The purpose of this paper, however, is not to argue for some principle or heuristic to improve these criteria, nor even to argue that parametric types are necessary for linguistics at all.

[1] By "signature," I refer to a partial order of types plus feature appropriateness declarations. The partial order itself, I shall refer to as a "type (inheritance) hierarchy."

[2] In this paper, the most general type will be called $\bot$.

Its purpose is simply to illustrate both the expressive and computational consequences of adding them to a signature.

In particular, section 5 proves that parametric types are not simply a macro language for types. They significantly extend the expressive power of finite type signatures. In spite of this, feature-based NLP systems can use parametric types efficiently. The two most common previous approaches have been to use the most general instance of a parametric type, e.g. *nelist(⊥)* without its appropriateness, or manually to "unfold" a parametric type into a non-parametric subhierarchy which suffices for a fixed grammar (e.g. Figure 2). The former does not suffice even for fixed

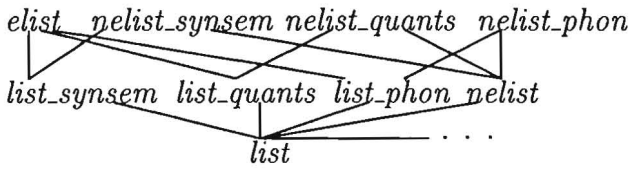


Figure 2: A manually unfolded subhierarchy.

grammars because it simply disables type checking on feature values. The latter is error-prone, inconvenient, and subject to change with the grammar. Section 6 provides two methods for unfolding parametric signatures automatically.

2 Parametric Type Hierarchies

Parametric types are not types. They are functions which provide access or a means of reference to a set of types (their image) by means of argument types, or "parameters" (their domain). Figure 1 has only unary functions; but in general, parametric types can be n-ary functions over n-tuples of types.[3] This means that hierarchies which use parametric types are not "type" hierarchies, since they express a relationship between functions (here, we can regard simple types as nullary parametric types):

Definition 1: A *parametric (type) hierarchy* is a partial order, $\langle P, \sqsubseteq_P \rangle$, plus a partial *argument assignment function*, $a_P : P \times P \times \mathsf{Nat} \rightarrow \mathsf{Nat} \cup \{0\}$, in which P consists of (simple and) parametric types, (i.e. no ground instances of parametric types), including the simple type, $\bot$. For p arity n, q arity m, $a_P(p,q,i)$, written $a_p^q(i)$, is only defined when $p \sqsubseteq_P q$ and $1 \leq i \leq n$, and only attains a value between 0 and m. If there is at least one non-simple parametric type, the hierarchy is *properly parametric.*

The argument assignment function encodes the identification of parameters between a parametric type and its parametric subtype. The number, n, refers to the nth parameter of a parametric type, with 0 referring to a parameter which has been dropped. In practice, this is normally expressed by the names given to type variables. In the parametric type hierarchy of Figure 1, *list* and *nelist* share the same variable, X, because $a_{list}^{nelist}(1)$ is 1. If $a_{list}^{nelist}(1) = 0$, then *nelist* would use a different variable name. As a more complicated example, in Figure 3, $a_b^d(1) = 1$, $a_b^d(2) = 3$, $a_c^d(2) = 2$, $a_c^d(1) = 0$, and $a_\bot$ and a_e are undefined ($\uparrow$) for any pair in $P \times \mathsf{Nat}$.

[3] In this paper, "parametric type" will refer to such a function, written as the name of the function, followed by the appropriate number of "type variables," variables which range over some set of types, in parentheses, e.g. *list(X)*. "Type" will refer to both "simple types," such as $\bot$ or *elist*; and "ground instances" of parametric types, i.e. types in the image of a parametric type function, written as the name of the function followed by the appropriate number of actual type parameters in parentheses, such as *list(⊥)*, *set(psoa)* or *list(set(⊥))*. I will use letters t, u, and v to indicate types; capital letters to indicate type variables; capitalised words to indicate feature names; p, q, and r for names of parametric types; and g to indicate ground instances of parametric types, where the arguments need not be expressed.

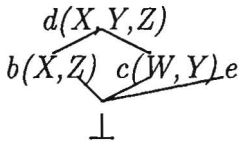


Figure 3: A subtype that inherits type variables from more than one supertype.

3 Induced Type Hierarchies

The relationship expressed between two functions by $\sqsubseteq_P$, informally, is one between their image sets under their domains,[4] while each image set internally preserves the subsumption ordering of its domain. It is, thus, possible to think of a parametric type hierarchy as "inducing" a non-parametric type hierarchy, populated with the ground instances of its parametric types, which obeys both of these relationships.

Definition 2: Let $\bar{\cdot} : I(P) \rightarrow P$ be the function which maps ground instances, $p(t_1, \ldots, t_n)$ back to their parametric types, $p(X_1, \ldots, X_n)$, in P. Given parametric type hierarchy, $\langle P, \sqsubseteq_P, a \rangle$, the *induced (type) hierarchy*, $\langle I(P), \sqsubseteq_I \rangle$, is defined such that:

- $I(P)$ is the smallest set, I, such that, for every parametric type, $p(X_1, \ldots, X_n) \in P$, and for every tuple, $\langle t_1 \ldots t_n \rangle \in I^n$, $p(t_1, \ldots, t_n) \in I$.
- For $g_1 = p(t_1, \ldots, t_n)$, and $g_2 = q(u_1, \ldots, u_m)$, $g_1 \sqsubseteq_I g_2$ iff $\bar{g}_1 \sqsubseteq_P \bar{g}_2$, and, for all $1 \leq i \leq n$, either $a_p^q(i) = 0$ or $t_i \sqsubseteq_I u_{a_p^q(i)}$.

Note that, in the case of $n = 0$, this function maps simple types to themselves, and that, therefore, $I(P)$ contains all of the simple types of P. In the case where g_1 is simple, $g_1 \sqsubseteq_I g_2$ iff $\bar{g}_1 \sqsubseteq_P \bar{g}_2$.

Figure 4 shows a fragment of the type hierarchy induced by Figure 1. If *list* and *nelist* had not

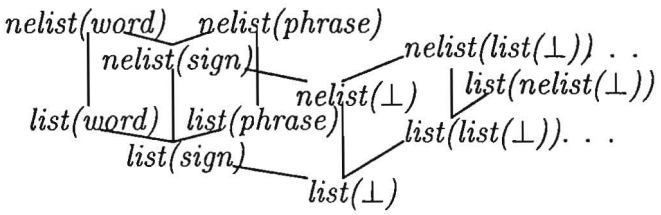


Figure 4: Fragment induced by Figure 1.

shared the same type variable ($a_{list}^{nelist}(1) = 0$), then it would have induced the type hierarchy in Figure 5. In the hierarchy induced by Figure 3, *b(e,e)* subsumes types *d(e,Y,e)*, for any type *Y*,

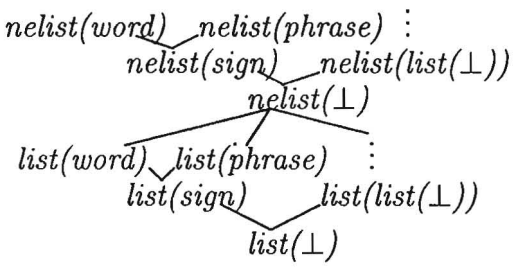


Figure 5: Another possible induced hierarchy.

for example *d(e,c(e,e),e)*, or *d(e,b(⊥,e),e)*, but not *d(c(⊥,e),e,e)*, since $e \not\sqsubseteq_I c(\bot, e)$. Also, for any types, *W*, *X*, and *Z*, *c(W,e)* subsumes *d(X,e,Z)*.

[4] One can restrict these domains with "parametric restrictions," a parallel to appropriateness restrictions on feature values. This abstract assumes that these domains are always the set of all types. This is the most expressive case of parametric types, and the worst case, computationally.

The present approach permits parametric types in the signature, but only ground instances in a grammar relative to that signature. If one must refer to "some list" or "every list" within a grammar, one may use $list(\bot)$, while still retaining groundedness. An alternative to this approach would be to attempt to deal with parametric types themselves directly within descriptions. From a processing perspective, this is problematic when closing such descriptions under total well-typing, as observed in [Car92]. The most general satisfier of the description, $list(X) \wedge$HEAD:HEAD$\doteq$TAIL:HEAD, for example, is an infinite feature structure of the infinitely parametric type, *nelist(nelist(...* because X unifies with *nelist(X)*. [5]

Induced type hierarchies have the following nice property, which allows us to speak of an induced unification operation:[6]

Theorem 1: If $\langle P, \sqsubseteq_P \rangle$ is a join semilattice, then $\langle I(P), \sqsubseteq_I \rangle$ is a join semilattice. In particular, given $g_1 = p(t_1, \ldots, t_n), g_2 = q(u_1, \ldots, u_m) \in I(P)$, $g_1 \sqcup_I g_2$ is $g_3 = r(v_1, \ldots, v_s)$, where $\bar{g}_1 \sqcup_P \bar{g}_2 = \bar{g}_3$, and, for all $1 \leq k \leq s$,

$$v_k = \begin{cases} t_i \sqcup_I u_j & \text{if there exist } i \text{ and } j \text{ such that} \\ & \quad a_p^r(i) = k \text{ and } a_q^r(j) = k \\ t_i & \text{if there is such an } i, \text{ but no such } j \\ u_j & \text{if there is such a } j, \text{ but no such } i \\ \bot & \text{if there is no such } i \text{ or } j. \end{cases}$$

So $g_1 \sqcup_I g_2 \uparrow$, if $\bar{g}_1 \sqcup_P \bar{g}_2 \uparrow$, or there exist i, j, and $k \geq 1$ such that $a_p^r(i) = k$, and $a_q^r(j) = k$, but $t_i \sqcup_I u_j \uparrow$.

In the induced hierarchy of Figure 3, for example, $b(e, \bot) \sqcup_I b(\bot, e) = b(e, e)$; $b(e, e) \sqcup_I c(\bot) = d(e, \bot, e)$; and $b(e, e)$ and $b(c(\bot), e)$ are not unifiable, as e and $c(\bot)$ are not unifiable. Note that joins in an induced hierarchy do not always correspond to joins in a parametric hierarchy. In those places where a_P attains 0, types can unify without a corresponding unification in their parameters. Such is the case in Figure 5, where every instance of $list(X)$ ultimately subsumes $nelist(\bot)$. One may also note that induced hierarchies can have not only deep infinity, where there exist infinitely long subsumption chains, but broad infinity, where certain types can have infinite supertype branching factors, as in the case of $nelist(\bot)$ or, in Figure 1, *elist*.

4 Appropriateness

So far, we have formally considered only type hierarchies, and no appropriateness. Appropriateness constitutes an integral part of a parametric type signature's expressive power, because the scope of its type variables extends to include it.

Definition 3: A *parametric (type) signature* is a parametric hierarchy, $\langle P, \sqsubseteq_P, a_P \rangle$, along with finite set of features, $Feat_P$, and a partial *(parametric) appropriateness function*, $Approp_P : Feat_P \times P \rightarrow Q$, where $Q = \bigcup_{n \in \mathsf{Nat}} Q_n$, and each Q_n is the smallest set satisfying the equation, $Q_n = \{1, \ldots, n\} \cup \{p(q_1, \ldots, q_k) | p \in P \text{ arity } k, q_i \in Q_n\}$, such that:

1. (Feature Introduction) For every feature $f \in Feat_P$, there is a most general parametric type $Intro(f) \in P$ such that $Approp_P(f, Intro(f))$ is defined
2. (Upward Closure / Right Monotonicity) For any $p, q \in P$, if $Approp_P(f, p)$ is defined and $p \sqsubseteq_P q$, then $Approp_P(f, q)$ is also defined and $Approp_P(f, p) \sqsubseteq_Q Approp_P(f, q)$, where $\sqsubseteq_Q$ is defined as $\sqsubseteq_{I(P)}$ with natural numbers interpreted as universally quantified variables (i.e. $a(1) \sqsubseteq_Q b(1)$ iff $\forall x a(x) \sqsubseteq_P c(x)$)
3. (Parameter Binding) For every $p \in P$ of arity n, for every $f \in Feat_P$, if $Approp_P(f, p)$ is defined, then $Approp_P(f, p) \in Q_n$.

[5] Occasionally, one also sees parameterized lists used in HPSG with general descriptions as parameters, e.g. $list($LOCAL : CAT : HEAD : *verb*$)$. Attempting to interpret these either as types or macro descriptions is also quite problematic, in general, as explained in the full version of this paper.

[6] The proofs of these theorems can be found in the full version of this paper

The special construction of Q is required to ensure that only those natural numbers less than or equal to the arity of a given parametric type are used in its appropriateness declaration. $Approp_P$ maps a feature and the parametric type for which it is appropriate to its value restriction on that parametric type. The first two of these conditions are the usual conditions on appropriateness, taken straight from [Car92]. The third says that the natural numbers in its image refer to the parametric variables of the appropriate parametric type — we can use one of these parameters wherever we would normally use a type. Notice that ground instances of parametric types are permitted as value restrictions, as well as instances of parametric types whose arguments are bound to these parametric variables, as well as the parametric variables themselves. The first is used in HPSG for features such as SUBCAT, whose value must be $list(synsem)$; whereas the second and third are used in the appropriateness specification for $nelist(X)$ in Figure 1. The use of parameters in appropriateness restrictions is what conveys the impression that ground instances of lists or other parametric types are somehow derived from their parameter types.

A parametric signature induces a type hierarchy as defined above, along with the appropriateness conditions on its ground instances, determined by the substitution of actual types for parametric variables. We can also prove that this is a bona fide signature:

Theorem 2: If $Approp_P$ satisfies properties (1)–(3) in Definition 3, then $Approp_{I(P)}$ satisfies properties (1) and (2).

5 Equivalence

Parametric signatures can encode universes of information states that non-parametric signatures cannot. Taken literally, this means that there are parametric signatures whose feature structures do not correspond, respecting unification, to those of any non-parametric signature, abstracting away from issues such as whether information is represented by subtypes or by feature values:[7]

Definition 4: Two type signatures, P and Q, are *equivalent* ($P \approx_S Q$) if there exists an isomorphism (w.r.t. unification) between the totally well-typed feature structures of P and those of Q.

Of course, for the purposes of processing with a parametric signature, we only need to ensure that there is a non-parametric signature into which we can embed its feature structures:

Definition 5: Type signature, P, *subsumes* signature Q ($P \sqsubseteq_S Q$) if there exists an embedding (w.r.t. unification) from the totally well-typed feature structures of P to those of Q.[8]

Both of these definitions naturally extend to parametric type signatures, simply by substituting their induced type signatures. Even with this weaker notion of correspondence, some parametric signatures will still be a problem:

Theorem 3: For any finite parametric signature, P, for which there is a maximal simple type, s, a maximal non-simple type, $q(X_1, \ldots, X_n)$, a non-maximal non-simple type, $p(Y_1, \ldots, Y_m)$, and a subtype $r \sqsupseteq p$ for which a_p^r attains zero, then there is no finite non-parametric signature, N, for which $P \sqsubseteq_S N$.

There are many, however, which are better behaved:

Definition 6: Parametric type signature, P is *persistent* if:

- a_P never attains zero, and
- For every $f \in Feat_P$ and $p \in P$ such that $Approp_P(f,p)$ exists, $Approp_P(f,p) \in Q_n$, where n is the smallest number such that $Approp_P(f, Intro(f)) \in Q_n$.

Theorem 4: For any persistent parametric signature, P, there is a finite non-parametric signature, N, such that $P \sqsubseteq_S N$.

[7] Taken as models of information states, it makes sense to reason, and compute, with feature structures induced by signatures in this way. If we are interested in finding models of feature terms themselves, however, then, as [Mos95] observes, not all models of one signature may be suitable models of other signatures which correspond in this sense.

[8] Actually, these mappings must also respect equivalence classes under alphabetic variance, the discussion of which I omit for simplicity; but the results of this section hold in this case.

Persistence means that parameters do not disappear as one moves up (more specific) in subtyping or down (more general) in appropriate value restrictions. Loosely speaking, persistence, combined with the normal restrictions on appropriateness, allows us to treat parameter values like feature values. Notice that Figure 1 satisfies the conditions of Theorem 3 (*s=phrase,q=nelist,p=list,r=elist*). If we change *elist* to *elist(X)*, i.e. give every kind of list its own empty list, then it satisfies the conditions of Theorem 4. This does not mean that users can dispense with parametric types in HPSG, however. It means that implementors can pretend to handle parametric types in HPSG, when, in fact, they are secretly doing their computations without them. To force users to use the non-parametric signatures directly is dangerous — unless the stronger correspondence, $\approx_S$, holds, they will be creating a collection of totally well-typed information states, some of which correspond to no real state relative to the parametric signature; and the implementor can do nothing to warn them. Theorem 3 gives us some indication of when it does not hold; but I believe:
Conjecture: For any properly parametric signature, P, with appropriate features, there exists no finite non-parametric signature, N, such that $P \approx_S N$.

These statements do not span all cases of parametric signatures; but they show that the use of parameters in appropriate value restrictions and the ability to drop parameters play a central role in the extra expressive power of parametric signatures. Figure 1, furthermore, satisfies the conjecture, with or without the repair to *elist*. Of course, if one possesses a powerful enough theory of relations (e.g. [Meu97]), one can use them as constraints in a theory to force an equivalence between any two signatures relative to that theory.

6 Finiteness

For the purposes of feature-based NLP, one cannot simply unfold any parametric type signature into its induced signature at compile-time and then proceed as usual. This is particularly true for systems which precompile all of their type operations, as many induced signatures contain infinitely many types.[9] On the other hand, given that one will only see finitely many ground instances of parametric types in a theory, it is certainly desirable to perform some precompilation specific to those instances, which will involve some amount of unfolding. What is needed is a way of precomputing, given a signature and a grammar, what part of the induced hierarchy will be needed at run-time, so that type operations can be compiled only on that part.[10]

There are essentially two ways by which one can identify this part. Both of them work even when Theorem 3 says that there is no embedding in general, because here, we only need to be concerned with an embedding which is good enough for a fixed grammar, much as Figure 2 may be a satisfactory approximation of Figure 1 for some grammars. The first is to use all ground instances of a *bounded parametric depth*:
Definition 7: Given a parametric hierarchy, P, the *parametric depth* of a type, $t = p(t_1, \ldots, t_n) \in I(P)$, $\delta(t)$, is 0 if $n = 0$, and $1 + \max_{1 \leq i \leq n} \delta(t_i)$ if $n > 0$. A type hierarchy, $I \subseteq I(P)$ is of *bounded parametric depth* if, for some k, for every type $t \in I$, $\delta(t) \leq k$.
So, for example, $\delta(\text{list}(\text{list}(\text{list}(\bot)))) = 3$.

If P is finite, then, trivially, I is finite. The virtues of this approach are that it is very easy to conceptualize which types belong to such a fragment, and that, in many cases,[11] a theory requires only ground instances of some bounded parametric depth anyway. Its principal problem is that it typically does not produce a subalgebra (i.e. closed under $\sqcup_{I(P)}$), although this is not a problem for every attribute-value logic (notably SRL), or for every implementation. Depth-bounded hierarchies

[9] In fact, without parametric restrictions (fn. 4), the induced hierarchy of any properly parametric hierarchy will be infinite.

[10] Systems where some type operations are computed at run-time, can still memoize computations on instances of parametric types as they are witnessed to achieve much of the same savings, perhaps along with some precompilation.

[11] The suggestion (p. 396, fn. 2) in [PS94] that the domain of the type variable in Figure 1 might be restricted to non-parametric sorts would place a bound of $k = 1$, although there has been a limited use of lists of lists in HPSG, e.g. [MSI94].

can also contain many unnecessary types for a given theory.

The second way is to identify some set of ground instances (a *generator set*) which are necessary for computation, and close that set under $\sqcup_{I(P)}$:

Theorem 5: If parametric hierarchy, P, is a finite join semilattice, and $G \subseteq I(P)$, is finite, then the subalgebra of $I(P)$ generated by G, $I(G)$, is finite.

There is also a polynomial time algorithm for computing $I(G)$. One can easily construct a generator set: simply collect all ground instances of types attested in the grammar, or collect them and add all of the simple types, or add the simple types along with some extra set of types distinguished by the user at compile-time. When the generator set is chosen in this way, knowing which types will eventually belong to $I(G)$ (which, in general, is more difficult) becomes far less important because, whatever they are, they will be sufficient for computation with the grammar. In fact, $I(G)$ will be the least set of types which is adequate for unification-based processing with the given grammar, even though $I(P)$ may be infinite.

These techniques have another application. One can translate a signature to a $\approx_S$-equivalent one in which formerly feature-encoded information is expressed by the use of additional subtypes. This is particularly desirable in systems that precompile their type operations, since this translation can increase run-time efficiency. The problem is that this generally involves an exponential type explosion, which cannot be avoided with bit vectors or other disjunctive type encodings because there can be exponentially more *maximal* types. In certain cases, however, one can automatically translate a feature encoding to a parametric type encoding in polynomial time, simply by using one parameter for every former feature.[12] The resulting signature can then be unfolded to provide as many types as a given grammar requires.

References

[Car92] Carpenter, B., 1992. *The Logic of Typed Feature Structures.* Cambridge University Press.

[DH88] Dietrich, R. and Hagl, F., 1988. A Polymorphic Type System with Subtypes for Prolog. Proceedings of the 2nd European Symposium on Programming, Spring LNCS 300, pp. 79–93.

[Dor92] Dorna, M., 1992. Erweiterung der Constraint-Logiksprache CUF um ein Typsystem. Diplomarbeit, Universität Stuttgart.

[MSI94] Manning, C., Sag, I., and Iida, M., 1994. The Lexical Integrity of Japanese Causatives. To appear in G. Green and R. Levine eds., Readings in HPSG, Cambridge.

[Kin89] King, P. J., 1989. *A Logical Formalism for Head-Driven Phrase Structure Grammar.* Ph.D. thesis, University of Manchester.

[Meu97] Meurers, D., Richter, F., Sailer, M., and Winhart, H., 1997. Ein HPSG-Fragment des Deutschen, Teil 1: Theorie. Arbeitspapiere des SFB 340. Universitt Tbingen

[Mos95] Moshier, M. A., 1995. Featureless HPSG. Unpublished ms. Presented at *MOL 4.*

[PS94] Pollard, C. and Sag, I., 1994. *Head-Driven Phrase Structure Grammar.* University of Chicago Press.

[Smo89] Smolka, G., 1989. Logic Programming over Polymorphically Order-Sorted Types. Ph.D. Dissertation, Universität Kaiserslautern.

[YFS92] Yardeni, E., Früwirth, T. and Shapiro, E., 1992. Polymorphically Typed Logic Programs. In F. Pfenning, ed., *Types in Logic Programming.* MIT Press, pp. 63–90.

[12] The full version of this paper elaborates on the cases for which this translation can be $\approx_S$-equivalent. In general, it requires the use of parametric restrictions (fn. 4).

A Polynomial Model for Unrestricted Functional Uncertainty

Owen Rambow
CoGenTex, Inc.
840 Hanshaw Road, Suite 11
Ithaca, NY 14850-1589
USA
owen@cogentex.com

1 Introduction: LFG and TAG

LFG (Kaplan and Bresnan, 1982) is based on the now fairly standard assumption that there are several interrelated but independent levels of representation. LFG's c-structure is a representation of phrase structure, and is derived by the underlying context-free grammar (CFG). f-structure is a representation of the functional structure of a sentence, using categories such as SUBJECT, OBJECT, and so on. C- and f-structure are related by functional constraints associated with CFG rules, called *functional schemata.* As has been discussed in the LFG and related literature (Maxwell and Kaplan, 1993), parsing grammars that are associated with functional constraints is computationally costly, the run time being exponential in the length of the input string in the worst case.

Burheim (1996) proposes to replace the CFG-based characterization of c-structure with a tree adjoining grammar (TAG) (see (Joshi and Schabes, 1991) for an overview of TAG; also see (Kameyama, 1986) for an earlier proposal to relate LFG and TAG). The elementary structures in TAG are trees; the extended domain of locality (i.e., extended over that of CFG) allows us to devise lexicalized grammars in which each elementary structure is associated with exactly one lexical item and its entire projection, and which has positions for all the arguments of the predicate structure of the lexical item. In LFG terms, in a TAG we "pre-assemble" into a single tree all those c-structure rules whose left-hand side nonterminal will be associated with the same f-structure predicate through the use of the $\uparrow=\downarrow$ equation (which indicates syntactic projection in LFG). In a TAG derivation, the action of substituting or adjoining a tree to another corresponds directly to making the lexical item of the first tree an argument or an adjunct of the lexical item of the second tree. Thus, we can annotate the leaves of elementary trees with functional schemata such as $\downarrow=\uparrow$ OBJECT (which says that the tree substituted at this node will be the object of the anchor). In fact, if we assume that each functional schema is associated with a single node in a tree, then the f-structure can be read off from the derivation structure (which records which trees were composed how during the derivation).[1] More precisely, the derivation structure is a tree and it represents syntactically determined coreference (such as in control constructions) using features, while in

[1] For a discussion of some problematic cases in which f-structure (the syntactic dependency structure) and derivation structure do not match, see (Rambow et al., 1995). Additional problems arise with control, which however can be dealt with in a number of ways, which will not be discussed here.

LFG f-structure is represented as a feature structure with a notation suggesting reentrancy. Therefore, to obtain a standard LFG f-structure, the derivation tree of TAG would have to be converted to the feature structure-based notation. This can be achieved in a standard and deterministic manner. It is possible to devise a methodology for deriving TAGs from LFG grammars which obey certain restrictions (i.e., "compiling" LFG into TAG). There should be considerable benefits for efficiency in parsing because of this pre-assembly.[2]

Unfortunately, many grammars in the LFG framework do not meet the criteria necessary for easy conversion to a TAG grammar. There are two reasons for this. First, the same functional schema can be used at different nodes in a derivation tree which themselves are associated with the same f-structure. This will have the effect of inducing two dependent derivations from two different nodes, and has been used in the LFG literature to handle the crossing dependencies of Dutch (Bresnan et al., 1983). While TAG can handle these Dutch constructions, it cannot derive the phrase structure proposed by Bresnan et al. (1983), which has two "spines", one carrying the nouns, and the other carrying, in the same order, the verbs. These types of cases can, however, be derived by multiple CFGs (Seki et al., 1991), which are the same formalism as string-based LCFRS (Weir, 1988), and which can be seen as a generalization of TAG. Seki et al. (1993) show that a particular type of restricted LFG is weakly equivalent to MCFG/LCFRS. This is significant, since MCFG/LCFRS is known to have restricted generative capacity and to be polynomially parsable.

The second reason why many LFG grammars cannot simply be converted to a TAG grammar is the use of *functional uncertainty* (Kaplan and Zaenen, 1989) in functional schemata. In functional uncertainty, a functional equation uses regular expressions, which are interpreted as a shorthand for an infinite set of "ordinary" functional equations. This device is used in LFG grammars to handle long-distance extraction of *wh*-words in English, and for a variety of word order phenomena in West Germanic (Zaenen and Kaplan, 1995). Using functional equations such as $\downarrow=\uparrow$ XCOMP*OBJECT, one can specify that, for instance, the filler at a specifier position of CP (i.e., the left daughter of an S node) is in fact the object of an arbitrarily deeply embedded clause (designated by XCOMP). (Since the Kleene-star notation also allows the deletion of the XCOMP, we also get that the position may be filled by the same clause's object.) Functional uncertainty is not considered by Seki et al. (1993). Vijay-Shanker and Joshi (1989) show that functional uncertainty is a corollary in TAG, but this is of course only true for those cases in which TAG can provide an analysis of long-distance extraction. As Becker et al. (1991) and Rambow et al. (1995) argue, there are cases in which TAG cannot provide a linguistically motivated analysis (word order in West Germanic, *wh*-movement in non-*wh*-initial languages such as Kashmiri); functional uncertainty also covers such cases. This paper proposes, for the first time, a TAG-related framework that directly models functional uncertainty.[3]

[2]Also see (Maxwell and Kaplan, 1993) for a discussion of how modifying the c-structure CFG in a less radical manner can also lead to improvements in parsing.

[3]Burheim (1996) mentions a possible treatment of functional uncertainty, but the approach is not worked out in any detail.

2 UVG-RegDL

Rambow (1994) introduces a new formalism called UVG-DL (unordered vector grammars with dominance links).[4] In UVG-DL, the elementary structures are sets of context-free productions, which are linked by *dominance links*. A *dominance link* links a rhs nonterminal of a production to the lhs nonterminal of a different production in the same set. A derivation proceeds like a CFG derivation, except that (a) if a single production from an instance of a set is used, then all other productions from that set must also be used in the same derivation (all exactly one time); and (b), in the derivation tree, the dominance links introduced by application of rules from instances of sets must correspond to dominance relations.

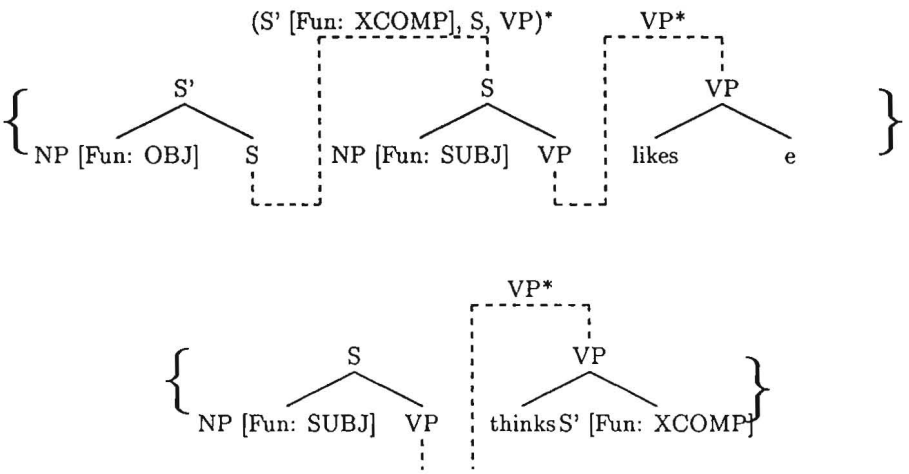


Figure 1: Elementary structures for *like* and *think*

In UVG-RegDL (Unordered Vector Grammars with Regular Expression-Restricted Dominance Links), introduced here, the dominance links are furthermore augmented with regular expressions (regexps). During a derivation, these regexps are interpreted as path conditions in the (context-free) derivation tree between the rhs nonterminal of the relevant instance of the dominating rule and the lhs nonterminal of the relevant instance of the dominated rule. They can be seen as a generalization of the subsertion-insertion constraints of DTG (Rambow et al., 1995). We refrain from a formal definition for lack of space, but discuss an example. In Figure 1, two elementary structures are shown. In the set for *like*, we see that the direct object has been moved above the subject, and the sister node to the direct object dominates the S node of *likes* through a link. This represents a long-distance *wh*-movement (or "topicalization"). The regexp annotation on the link says that the path that the link "stretches" over in a derivation may only contain nodes labeled VP, S, and S' [Fun: XCOMP]. (We consider bounded feature structures to convenient notational extensions to node labels.) This annotation thus prevents island violations (and corresponds to a largely similar stipulation in all linguistic theories.) Furthermore, in both sets, the two VP nodes are connected by a link which is annotated VP*, reflecting the fact that no clause boundary may intervene between two VP nodes of the same verbal projection. A derivation structure is shown in Figure 2. The dom-

[4] DTG, introduced in (Rambow et al., 1995), can be seen for the purposes of this paper as UVG-DL equipped with a definition of derivation defined with respect to the elementary sets, not the elementary trees or rules in the sets.

inance links connecting the two VP pairs have collapsed, but the link connecting the object to rest of the *likes* clause has been stretched over a path with labels S, VP, and S' [Fun: XCOMP] (top-to-bottom), and thus conform with its regexp.

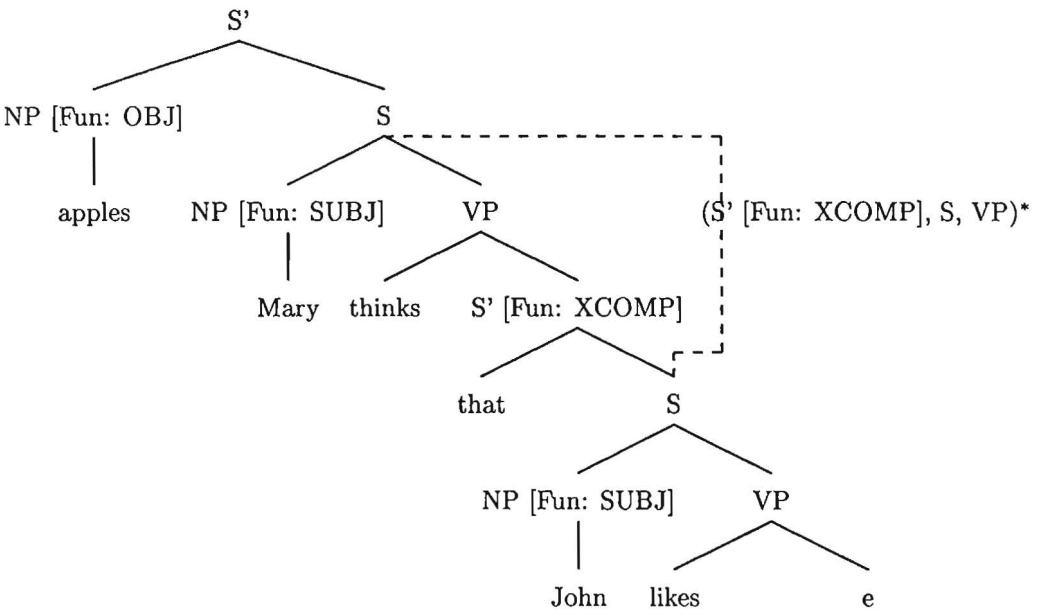


Figure 2: Derived structure for *Apples, Mary thinks John likes*

The principal formal results of this paper are that, like UVG-DL, UVG-RegDL generates only context-sensitive languages when lexicalized (i.e., every set of productions contains at least one terminal symbol) and that it is polynomially parsable when lexicalized. (Note that in linguistic applications, the restriction to lexicalization is standard.)

To see that $\mathcal{L}$(UVG-RegDL) $\subseteq \mathcal{L}$(CSG), let G be a UVG-RegDL. We construct a linear bounded automaton (LBA) M which recognizes exactly $L(G)$. M starts with the input word w on the tape and then proceeds to derive w from the start symbol of the grammar. Whenever a rule is used whose rhs contains a nonterminal at which a link starts, this link is recorded in an annotation following the nonterminal in the sentential form. The number of possible links is fixed by the grammar and hence each link can be encoded by a single symbol. In addition, we must keep track of the path this link is stretched over. For this purpose, the associated regular expression is converted to a deterministic finite automaton. In addition to recording the open links, we record the state of each open link. Again, the number of states per link is determined by the grammar, and each state can also each be expressed by a single tape symbol. The state transitions are expressed in the finite control of the LBA (as are, of course, the rules of G). Once the derivation has completed, the input copy of w and the derived copy are compared and M goes into an accepting state if they are equal. Crucially, since G is assumed to be lexicalized, the number of open links is linearly bounded in n, the length of w. Therefore, the derivation can be carried out in a space linearly bounded in n.

The proof of polynomial parsability is based on the construction of a CKY-style parser for UVG-RegDL. The CKY parser is a parser for CFG, except that the nonterminal entries in the squares of the parse matrix must be augmented with

information about open links. We represent the open links using an array of integers indexed on the links in the grammar and the states associated with the state machine of that link's regexp. In order to see that the resulting parser operates in time polynomial in n, we need to estimate the possible number of entries in the squares of the matrix. The maximal number of different combination of open links is linearly bounded by n^L, where L is a grammar constant designating the number of different links in the grammar. Furthermore, each link can be in a finite number of different states. Let S_L be the maximal number of states of a state machine associated with a regular expression on a link in G. Then the number of states that links may be in is LS_L, and the number of entries in each square of the matrix is in $O(n^{LS_L})$. As overall complexity we get $O(n^{LS_L+3})$.

This complexity result is polynomial in the length of the input string, but of course exponential in the size of the grammar. However, as Becker and Rambow (1995) discuss, in fact the exponent will be much smaller, since the types of "stretchable" dominance links in a grammar are quite limited and can be related to specific linguistic phenomena (such as, in English, Raising and *wh*-movement). In bottom-up parsing, it need only be recorded which type of link is being traced, not exactly which elementary set the link came from. Therefore, the exponent does not reflect the number of lexical items in a language, but rather the number of long-distance constructions that the language supports. Furthermore, it is possible to develop parsing strategies that do not hypothesize a "stretched" link unless no parse without stretching is available.

3 Outlook: Modeling LFG

Frequently, what seems to an outsider to be a notational variant is considered by followers of a linguistic framework to be a crucially different representation which fails to express the framework's deep insights. It is therefore important to address the following issue: on the face of it, UVG-RegDL really does not look much like LFG at all. Most importantly, there is no f-structure.

The formalism introduced above, UVG-RegDL, shares the same extended domain of locality of TAG. Again, the sets of phrase-structure rules can be seen as "pre-assembling" parts of c-structure. Therefore, if we annotate the leaf nodes of a set with functional annotations (as done in the examples), we can again read off the f-structure from the derivation structure (with the same remark about syntactically determined coreference as for the case of TAG). The notion of derivation structure we are using is exactly that defined for DTG (Rambow et al., 1995): it records not when and how individual context-free rules from sets are used, but rather when and how the entire sets are used. Roughly the same kind of heuristics that would allow us to derive a TAG from annotated c-structure rules will allows us to derive a UVG-RegDL. In addition, we can now deal with functional schemata involving unrestricted functional uncertainty (which result in dominance links annotated with regexps).

Several extensions are possible to make UVG-RegDL look even more like LFG. First, we can use a synchronized extension to UVG-RegDL to model the construction of the f-structure more explicitly (see (Rambow and Satta, 1996)). However, we are still left with the fact that f-structure is not defined as a tree, but as an attribute-value structure, a generalization of a tree. However, if we think of a UVG-RegDL set as a description of a tree rather than as a rewriting system (following (Vijay-Shanker, 1992)), then we can see that that description can also be specialized into

an attribute-value structure, with the addition of some additional constraints. It is therefore possible to envisage a synchronous extension to UVG-RegDL, which, without changing the essential formal properties, comes very close to modeling the formal machinery of LFG. This is a topic of ongoing research.

4 Linguistic Relevance: Germanic Word Order

There are (at least) two reasons why one may want to study formal properties of linguistic theories. First, this may be done for its own sake or for the sake of implementing efficient parsers. But second, and no less important, such a study allows us to compare linguistic analyses expressed in different frameworks, because the formal study allows us to relate formal devices in the two frameworks. Setting aside right now the issue of how to best represent LFG in terms of a UVG-RegDL-style formalism, let us consider the two LFG analyses of West Germanic word order presented in (Bresnan et al., 1983) and in (Zaenen and Kaplan, 1995) (which revises the previous analysis). As mentioned above, the earlier analysis suggests that the cross-serial dependencies of Dutch are, in c-structure, represented by two dependent derivations (of the nouns and of the verbs). This is achieved by using the same functional schema at different points in the derivation (c-)structure. The later paper rejects this analysis on theoretical, formal, and linguistic grounds. Instead, Zaenen and Kaplan (1995), considering a wider range of data from Dutch, Swiss German, and Standard German, uses functional uncertainty to relate nominal arguments to their verbs. Cross-serial dependencies, mandatory in Dutch (but not in the other languages), which functional uncertainty cannot impose, are achieved by a special type of LP rules.

In terms of the rewrite systems discussed in this paper, this shift corresponds exactly to the shift from using the "locality" of MCFG/LCFRS (which, we have seen, corresponds to the type of device used in (Bresnan et al., 1983)) to the use of "vertical context conditions" (of which functional uncertainty can be seen as an instance) as implemented by dominance links in formalisms such as UVG-DL, UVG-RegDL, and DTG. Becker et al. (1992) argue that MCFG/LCFRS are not powerful enough to derive German (and Swiss German) scrambling, while Rambow et al. (1995) argue for the use of DTG based on different types of arguments, including evidence from *wh*-movement in languages such as Kashmiri, the relation to dependency (i.e., functional structure), and the desire for monotonic derivations. What is striking is that the evidence adduced in (Zaenen and Kaplan, 1995) in favor of the functional uncertainty-based analysis is again completely independent of the evidence discussed in the context of formal rewrite systems by Becker et al. (1992) and Rambow et al. (1995). This can be taken as strong independent evidence that, across different approaches to syntax, the use of "vertical context" is emerging as a necessary and standard device. This generalization is only meaningful if backed up by the kind of formal analysis sketched in this paper.

At the same time, it is interesting to note that, under the "vertical context condition" approaches (both LFG and TAG-related), Dutch cross-serial dependencies, which simple TAG can easily derive, pose a problem. It has been conjectured that UVG-DL and DTG cannot derive the copy language $\{ww|w \in \{a,b\}^*\}$, and the same appears to be true for UVG-RegDL. Similarly, Zaenen and Kaplan (1995) must use a special type of ad-hoc LP rule to obtain the correct ordering for Dutch. We conclude that either UVG-RegDL and the current formal framework for LFG need to be refined in order to provide a more elegant derivation of the Dutch facts, or that the linguistic interpretation of the Dutch facts needs to be re-evaluated

(for example, Dutch competence grammar allows for free scrambling of nominal arguments but other factors effectively rule out all word orders but the cross-serial ordering). This issue remains open.

Bibliography

Becker, T., Joshi, A., and Rambow, O. (1991). Long distance scrambling and tree adjoining grammars. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL'91)*, pages 21–26. ACL.

Becker, T. and Rambow, O. (1995). Parsing non-immediate dominance relations. In *Proceedings of the Fourth International Workshop on Parsing Technologies*, Prague.

Becker, T., Rambow, O., and Niv, M. (1992). The derivational generative power, or, scrambling is beyond LCFRS. Technical Report IRCS-92-38, Institute for Research in Cognitive Science, University of Pennsylvania. A version of this paper was presented at MOL3, Austin, Texas, November 1992.

Bresnan, J., Kaplan, R., Peters, S., and Zaenen, A. (1983). Cross-serial dependencies in Dutch. *Linguistic Inquiry*, 13:613–635.

Burheim, T. (1996). Aspects of merging lexical-functional grammar with lexicalized tree-adjoining grammar. Unpublished abstract, University of Bergen.

Joshi, A. K. and Schabes, Y. (1991). Tree-adjoining grammars and lexicalized grammars. In Nivat, M. and Podelski, A., editors, *Definability and Recognizability of Sets of Trees*. Elsevier.

Kameyama, M. (1986). Characterising Lexical Functional Grammar (LFG) in terms of Tree Adjoining Grammar (TAG). Unpublished Manuscript. Dept. of Computer and Information Science, University of Pennsylvania.

Kaplan, R. M. and Bresnan, J. W. (1982). Lexical-functional grammar: A formal system for grammatical representation. In Bresnan, J. W., editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Mass.

Kaplan, R. M. and Zaenen, A. (1989). Long distance dependencies, constituent structure, and functional uncertainty. In Baltin, M. and Kroch, A., editors, *Alternative Conceptions of Phrase Structure*. University of Chicago Press, Chicago. IL.

Maxwell, J. T. and Kaplan, R. M. (1993). The interface between phrasal and functional constraints. *Computational Intelligence*, 19(4):571–590.

Rambow, O. (1994). Multiset-valued linear index grammars. In *32nd Meeting of the Association for Computational Linguistics (ACL'94)*. ACL.

Rambow, O. and Satta, G. (1996). Synchronous models of language. In *34th Meeting of the Association for Computational Linguistics (ACL'96)*, pages 116–123. ACL.

Rambow, O., Vijay-Shanker, K., and Weir, D. (1995). D-Tree Grammars. In *33rd Meeting of the Association for Computational Linguistics (ACL'95)*, pages 151–158. ACL.

Seki, H., Matsumura, T., Fujii, M., and Kasami, T. (1991). On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Seki, H., Nakanishi, R., Kaji, Y., Ando, S., and Kasami, T. (1993). Parallel multiple context-free grammars, finite state translation systems, and polynomial-time recognizable subclasses of lexical-functional grammar. In *31st Meeting of the Association for Computational Linguistics (ACL'93)*, pages 121–129, Columbus, OH. ACL.

Vijay-Shanker, K. (1992). Using descriptions of trees in a Tree Adjoining Grammar. *Computational Linguistics*, 18(4):481–518.

Vijay-Shanker, K. and Joshi, A. K. (1989). Long distance dependencies in LFG and TAG. In *27th Meeting of the Association for Computational Linguistics (ACL'89)*, Vancouver, B.C.

Weir, D. J. (1988). *Characterizing Mildly Context-Sensitive Grammar Formalisms.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania.

Zaenen, A. and Kaplan, R. M. (1995). Formal devices for linguistic generalizations: West Germanic word order in LFG. In Dalrymple, M., Kaplan, R. M., Maxwell, J., and Zaenen, A., editors, *Formal Issues in Lexical-Functional Grammar*, pages 215–239. CSLI Publications, Stanford, CA.

A Unified Notion of Derived and Derivation Structures in TAG (Preliminary Version)

James Rogers, University of Central Florida, jrogers@cs.ucf.edu

Generalizing CFGs

It has been recognized for some time that TAGs can be viewed as a particular sort of generalization of CFGs. Vijay-Shanker, Weir, and Joshi [VSWJ87] and Weir [Wei88], for instance, note a number of parallels between CFLs and TALs—both in their formal properties and in the fact that a number of characterizations of the CFLs can be generalized to provide characterizations of the TALs and TAG tree sets. Some of these generalizations iterate naturally, constructing infinite hierarchies of languages falling between the CFLs and the CSLs. This view of CFLs and TALs as adjacent levels of natural hierarchies of language classes is attractive in that it supports the transfer of results between the two and provides a route to more powerful formalisms which come with many of their properties already established.

But there is a discontinuity between the CFGs and the TAGs and their higher-order generalizations. The relationship between the set of structures that characterize the derivations of a CFG (i.e., its derivation trees) and the set of structures it derives is extremely simple regardless of whether we understand it to derive trees (in which case the relationship is identity) or strings (in which case it is the yield). This is true because the derivation structures embody the effect of the derivation steps—the expansion of a non-terminal into a string—directly. This is not the case for the traditional derivation structures for TAGs, where a derivation step involves the expansion of a non-terminal into a tree. If the derivation structures are taken to be trees themselves they cannot directly embody the actual expansion. Rather they just encode which trees are adjoined, which tree they adjoin into, and at which node of that tree the adjunction takes place. To obtain the derived tree from the derivation tree one must actually carry out these adjunctions. Thus the derived structure is not a substructure of the derivation structure. This separation of derived and derivation structures is extremely useful in that it abstracts away from the form of the derived structures. This allows formalisms which derive quite distinct classes of structures to be compared. Indeed, such comparison is the main thrust of Weir's dissertation. It does, however, introduce an asymmetry that mars the analogy between CFGs and TAGs and, we argue here, masks regularities that can be identified in the progression from regular sets, through CFLs and TALs, and beyond.

In this paper we take the parallel between string substitution in CFGs and tree adjunction in TAGs absolutely at face value. Just as we understand a CF rewrite rule to license the expansion of a node in a tree into a string of child nodes, we interpret an auxiliary tree in a TAG as licensing the expansion of a node in a three-dimensional tree-like structure into a tree of child nodes. Because we define these structures uniformly across an arbitrary range of dimensions, we

provisionally refer to them as (labeled) *tree manifolds*. These (3-dimensional) tree manifolds encode derivations in TAGs in precisely the same way that trees (2-dimensional tree manifolds) encode derivations in CFGs. There is, in fact, an easy mapping between TAG derivation trees of the traditional sort and these structures. Note that there are two distinct notions of immediate dominance in these structures, the immediate dominance relations in the elementary trees of the TAG and the relationship between a node and the nodes of the tree that expands it. When we take the maximal set of points with respect to this second relation—the two-dimensional yield of the three-dimensional structure—we get exactly the phrase-structure tree the derivation generates. When we then take the maximal set of points with respect to the domination relation within that tree—the one-dimensional yield of the two-dimensional structure—we get the string the derivation generates. In this way the operation taking the derivation structure into the derived phrase-structure tree is a higher-dimensional analog of the operation taking the derivation structure of a CFG into the string it generates. We take, then, these 3-dimensional tree manifolds to be both the derived and derivation structures of the TAG in exactly the same way that derivation trees can be taken to be both the derived and derivation structures of a CFG.

Tree Manifolds

We build the formal notion of labeled tree manifolds by analogy with *tree domains* [Gor67] in which nodes in a tree are assigned addresses which are strings in $\mathbb{N}^*$ with the root at address ϵ and the children of the node at address w at $w0$, $w1$,... in left-to-right order. A *labeled* tree domain includes a mapping assigning labels from some alphabet Σ to the nodes in the domain. Tree domains are subject to two well-formedness properties: they must be downward closed ($wv \in T \Rightarrow w \in T$ for all $w, v \in \mathbb{N}^*$) and they must be left-sibling closed ($wi \in T$ and $j < i \Rightarrow wj \in T$ for all $w \in \mathbb{N}^*$, $i, j \in \mathbb{N}$). These two properties can be expressed uniformly if one employs unary encoding for the natural numbers. Thus $n \in \mathbb{N}$ is encoded as 1^n and node addresses become sequences of strings of '1's (e.g., the address 1031 becomes $\langle\langle 1\rangle, \langle\rangle, \langle 111\rangle, \langle 1\rangle\rangle$).[1] The left-sibling closure property then becomes a downward closure property at the level of these strings of '1's: $s \cdot \langle wv\rangle \in T \Rightarrow s \cdot \langle w\rangle \in T$ for all $w, v \in \{1\}^*$, $s \in (\{1\}^*)^*$. From this perspective, strings are just a specialization of the notion of labeled tree domain—a prefix closed set of sequences of '1's (rather than sequences of such sequences) along with a labeling function.

One can interpret the address of a node in a tree domain as the sequence of string addresses one follows in traversing the path from the root to that node. This gives us the point of view we need in generalizing to higher dimensions. An address of a node in a 3-dimensional tree manifold is the sequence of tree addresses one follows in traversing the path to it from the root. Thus a 3-dimensional tree manifold is a set of sequences of sequences of sequences of '1's

[1] We could be pedantic and employ sequences of empty sequences $\langle\rangle$, but the notation is difficult enough to read as it stands.

(a third-order sequence of '1's). As with the first two levels this set of sequences must also be prefix closed. In general, an i-dimensional tree manifold will be a set of i^{th} -order sequences of '1's that is hereditarily prefix closed.

There is, however, a problem here. In the case of tree domains we are expanding nodes to linear structures. There is no ambiguity about how these structures fit together; the rightmost node dominated by a given node must immediately precede the leftmost node dominated by its immediate right sibling. In the case of 3-dimensional tree manifolds, however this is not the case. In expanding a node into a tree, any one of the nodes in the yield of that tree could be taken to dominate the subtree rooted at the original node.[2] If the set of maximal points (in the third dimension) are to form a coherent tree, each of the trees expanding a node must have a distinguished node at which the subtree will be spliced; in TAG terminology there must be a designated foot node. Projecting this down to the string level one gets a requirement that each string must be headed. Thus we will take our string addresses to be sequences of 'l's, designating positions to the left of the head, or 'r's, designating positions to the right of the head. Tree addresses are, again, sequences of such sequences, and the spine of a tree is just the path that follows heads at each step. The designated foot node, then, is the maximal node at an address that is a sequence of empty sequences. Note that we do not need to modify the downward closure property in any way.[3]

Tree Manifold Grammars and Automata

This notion of tree manifolds gives us a natural hierarchy of grammars. A grammar at level n is a finite set of labeled *local* (depth one in the major dimension) n-dimensional tree manifolds. The set of structures derived by such a grammar is the set of labeled n-dimensional tree manifolds that can be constructed by concatenating the local tree manifolds in the grammar. Note that maximal nodes of the tree manifold form *trivial* tree manifolds—consisting of a node and an empty set of children. As these must each be licensed by a tree manifold included the grammar just like every other local tree manifold, we can identify the set of symbols labeling trivial tree manifolds in the grammar as the terminals of the grammar, albeit terminals that potentially may be rewritten. Similarly, we will usually be interested in sets of tree manifolds in which the root is labeled with (one of) a designated (set of) start symbol(s). At the 2-dimensional level this gives us a straightforward generalization of CFGs in the form of sets of local trees [GS84]. A 1-dimensional tree manifold grammar consists of pairs of labels licensing the expansion of labeled nodes into labeled nodes to form strings, with the label of each node in the string depending only on the label of its predecessor. These generate the *strict 2-locally testable* languages, a much weaker class

[2]Note that our operation expanding nodes to trees is adjunction-like—the subtree rooted at the node is preserved as a single subtree, its children may not be split into multiple subtrees.

[3]While this approach is successful for the 3-dimensional case, its generalization to the higher-dimensional cases is yet incomplete. For our purposes here the first four levels suffice. (0-dimensional tree manifolds, of course, are just points.)

than the regular languages. At the 3-dimensional level we get TAGs generalized in the sense that the label of the node at which the adjunction is taking place, the label of the root of the adjoined tree, and the label of its foot may all be distinct. The fact that the possibility of adjunction depends only on the label of the node makes these grammars look superficially like *pure* TAGs—those without adjoining constraints. The requirement that maximal nodes be licensed, however, has the same effect as Obligatory Adjoining (OA) constraints while the fact that auxiliary trees may adjoin at dissimilarly labeled nodes has the same effect on generative capacity as Null Adjoining (NA) constraints. While one cannot, in the strictest sense, directly express Selective Adjoining (SA) constraints in these grammars, as we will see shortly they can generate any tree set generated by a TAG with such constraints. By generalization from the 2-dimensional terminology, we will refer to the sets licensed by these grammars as *local* sets of tree manifolds.

We also get a natural hierarchy of automata. An n-dimensional tree manifold automaton is a finite set of pairs associating labels (in the alphabet Σ) and local n-dimensional tree manifolds labeled with states from a finite set Q. A run of the automaton on a Σ-labeled n-dimensional tree manifold is an assignment of states to the nodes of the manifold in which the states assigned to each local manifold are associated by the automaton with the label of its root.[4] Such an automaton accepts a labeled n-dimensional tree manifold relative to some set of initial states iff there is a run of the automaton in which the root is assigned a state in that set. In the one dimensional case we get ordinary Finite State Automata. In the two dimensional case we get exactly the (non-deterministic) tree automata and in the three dimensional case we get exactly TAGs with adjoining constraints, modulo the generalization permitting adjunction of auxiliary trees at dissimilarly labeled nodes. Again by generalization from the 2-dimensional terminology, we will refer to the sets licensed by these automata as *recognizable* sets of tree manifolds.

It should be noted that the hierarchy of languages associated with the grammars and automata of this hierarchy is *not* the hierarchy of Linear Context-Free Re-writing Systems of Weir's dissertation [Wei88], but rather appears to be closer to the hierarchies of [VSWJ87].

Analysis

The key property of the local sets is the fact that the set of structures that may expand a node depends only on the label of that node—in other words, the features that constrain the form of the structure must be explicit in its labeling. This is the property that distinguishes them from the recognizable sets, where these features may be "hidden" in the state. It is this ability to hide features that allows us to capture SA constraints directly in the automata. Note, though, that a simple lift of Thatcher's [Tha67] proof that the yield languages of recognizable sets of trees are CFLs establishes that 3-dimensional tree manifold grammars and automata are equivalent in the sets of trees they

[4] Here we have notions of licensing of terminal nodes and restriction to tree manifolds with roots assigned states in a distinguished set analogous to the similar notions in the grammars.

yield (and, *a fortiori*, in the sets of strings they yield). The construction of the proof also dispels the apparent paradox in the fact that tree manifold grammars can directly express OA and NA constraints but not SA constraints while the strong generative capacity of pure TAGs is generally understood to be unaffected by SA constraints and extended by OA or NA constraints. In capturing SA constraints in a pure TAG one must extend the labels to explicitly encode the constraints in exactly the way that Thatcher's construction extends the labels of the grammar to encode states. In both contexts the classes of tree sets are equivalent modulo a projection. Of course, in the case of tree manifolds there is no need to extend the labels of the maximal nodes, and so trees they yield may have the original labels. The distinction between the local and recognizable sets arises at the level of the derivation structure—at the level of trees in the CF case, at the level of tree manifolds in the TAG case. This level has to do with the way in which the elementary structures are analyzed, with the structure of strings in CFGs, with the structure of trees in TAGs. It is at this level that the properties of elementary trees can be formalized and it is at this level that the linguistic significance of the distinction between local and recognizable sets of tree manifolds will arise if it exists.

One can limit the grammars to generate only pure TAG tree sets by restricting them to local tree manifolds in which the root node and the root and foot of the child tree are identically labeled, and by requiring all nodes to be licensed to have null expansions. Thus the distinction between pure TAGs and TAGs in general, from this point of view, is a consequence of stipulations that have been placed on the form of the grammar. To the extent that the formalism is intended to capture characteristics of natural language, such stipulations are explicit expressions of the theory of syntax it embodies. Here they are consequences of two of the most basic aspects of the linguistic motivation of TAGs. The restriction on the labeling of the local tree manifolds is a consequence of the fact that auxiliary trees are intended to capture recursive structures. The requirement that every label be licensed as a terminal is a consequence of fact that initial trees are intended to capture the minimal non-recursive structures of the languages—and thus every "sentential form" of a derivation will be in the language. Adjoining constraints, in part, have the effect of moderating the formal effect of these restrictions.

Schabes and Shieber [SS94] have proposed an alternative notion of derivation in TAG in which multiple trees may be adjoined at a single node. One of their motivations is the fact that when a phrase is multiply modified the auxiliary trees encoding the modifiers must nest when they adjoin to the node rooting that phrase. Nevertheless, the significant relationships are between the modifier trees and the modified node. By admitting derivation trees with each of the modifiers adjoined at the same node, those relationships can be expressed locally even though the trees are not locally related in the derived tree. From the standard point of view, the restriction that no more than one tree adjoin to a node is primarily stipulative. If we take our derivation structures to be tree manifolds, however, the restriction becomes a simple physical fact. Schabes and Shieber's conception of derivation can, nonetheless, be accommodated if one recognizes

that their goal is to express what, from the point of view of tree manifolds, are non-local relationships.[5] As shown by Joshi and Levy [JL77] there is a very rich class of such constraints that can be employed in defining sets of trees without leaving the realm of recognizable sets. In fact, in their original conception, adjoining constraints were expressed by such *domination* and *proper analysis predicates* [JLT75]. By lifting Joshi and Levy's proofs to the level of tree manifolds one can show that such predicates do not extend the capacity of TAGs with only SA, OA and NA constraints.

This leads us, at last, to the original motivation for these studies. We expect that tree manifolds will provide us with the class of models needed to extend the model theoretic techniques we have applied to local and recognizable sets of trees to apply to TAG tree sets. This approach allows the properties of the derived and derivation structures to be expressed as ordinary logical predicates, providing an extremely natural way of capturing linguistic intuitions, including those that might be expressed by domination and proper analysis predicates. More importantly, this approach provides a uniform framework for formalizing linguistic theories. Whereas in the past its applicability has been limited to theories that license CF (and regular) string languages, the extension to TAG tree sets and higher dimensional generalizations will enable us to address a range of mildly context-sensitive formalisms.

References

[Gor67] Saul Gorn. Explicit definitions and linguistic dominoes. In John F. Hart and Satoru Takasu, editors, *Systems and Computer Science, Proceedings of the Conference held at Univ. of Western Ontario, 1965.* Univ. of Toronto Press, 1967.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[JL77] Aravind K. Joshi and Leon S. Levy. Constraints on structural descriptions: Local transformations. *SIAM Journal of Computing*, 6(2):272–284, 1977.

[JLT75] Aravind K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10:136–163, 1975.

[Shi94] Stuart M. Shieber. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385, 1994.

[SS90] Stuart M. Shieber and Yves Schabes. Synchronous tree-adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics*, volume 3, pages 253–258, Helsinki, 1990. Association for Computational Linguistics.

[SS94] Yves Schabes and Stuart M. Shieber. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124, 1994.

[5] In the case of synchronous TAGs, it appears that the variation between Shieber and Schabes's original definition [SS90] and the revised definition of [Shi94, SS94] manifests itself as a variation in the permissible mappings between tree manifolds much as the modifications to the revised definition explored in [Shi94] are expressed as variations in the permissible mappings between derivation structures.

[Tha67] J. W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, 1:317–322, 1967.

[VSWJ87] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. On the progression from context-free to the tree adjoining languages. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 389–401. John Benjamins, Amsterdam, 1987.

[Wei88] David J. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms.* PhD thesis, University of Pennsylvania, 1988.

Separating Dependency from Constituency in a Tree Rewriting System*

Anoop Sarkar
Department of Computer and Information Science
University of Pennsylvania
200 South 33rd St, Philadelphia PA 19104
anoop@linc.cis.upenn.edu

1 Introduction

We define a new grammar formalism called Link-Sharing Tree Adjoining Grammar (LSTAG) which arises directly out of a concern for distinguishing the notion of constituency from the notion of relating lexical items in terms of linguistic dependency[1](Mel'čuk, 1988; Rambow and Joshi, 1992). This work derives directly from work on Tree Adjoining Grammars (TAG) (Joshi, Levy, and Takahashi, 1975) where these two notions are conflated. The set of derived trees for a TAG correspond to the traditional notions of constituency while the derivation trees of a TAG are closely related to dependency structure (Rambow and Joshi, 1992). A salient feature of TAG is the extended domain of locality it provides for stating these dependencies. Each elementary tree can be associated with a lexical item giving us a lexicalized TAG (LTAG)(Joshi and Schabes, 1991). Properties related to the lexical item such as subcategorization, agreement, and certain types of word-order variation can be expressed directly in the elementary tree (Kroch, 1987; Frank, 1992). Thus, in an LTAG all of these linguistic dependencies are expressed locally in the elementary trees of the grammar. This means that the predicate and its arguments are always topologically situated in the same elementary tree.

However, in coordination of predicates, e.g. (1), the dependencies between predicate and argument cannot be represented in a TAG elementary tree directly, since several elementary trees seem to be 'sharing' their arguments.

(1) a. Kiki frolics, sings and plays all day.
b. Kiki likes and Bill thinks Janet likes soccer.

The idea behind LSTAG is that the non-local nature of coordination as in (1) (for TAG-like grammar formalisms) can be captured by introducing a restricted degree of *synchronized* parallelism into the TAG rewriting system while retaining the existing *independent* parallelism[2](Engelfriet, Rozenberg, and Slutzki, 1980; Rambow and Satta, to appear). We believe that an approach towards coordination that explicitly distinguishes the dependencies from the constituency gives a better formal

*Thanks to Christy Doran, Aravind Joshi, Nobo Komagata, Owen Rambow, and B. Srinivas for their helpful comments and discussion.

[1]The term dependency is used here broadly to include formal relationships such as case and agreement and other relationships such as filler-gap.

[2]It is important to note that while the adjunction operation in TAGs is "context-free", synchronized parallelism could be attributed to the TAG formalism due to the string wrapping capabilities of adjunction, since synchronized parallelism is concerned with how strings are derived in a rewriting system. We note this as a conjecture but will not attempt to prove it here.

understanding of its representation when compared to previous approaches that use tree-rewriting systems which conflate the two issues, as in (Joshi, 1990; Joshi and Schabes, 1991; Sarkar and Joshi, 1996) which have to represent sentences such as (1) with either unrooted trees or by performing structure merging on the derived tree. Other formalisms for coordination have similar motivations: however their approaches differ, e.g. CCG (Steedman, 1985; Steedman, 1997b) extends the notion of constituency, while generative syntacticians (Moltmann, 1992; Muadz, 1991) work with three-dimensional syntactic trees.

2 Synchronized Parallelism

The terms *synchronized* parallelism and *independent* parallelism arise from work done on a family of formalisms termed parallel rewriting systems that extend context-free grammars (CFG) by the addition of various restrictive devices (see (Engelfriet, Rozenberg, and Slutzki, 1980))). Synchronized parallelism allows derivations which include substrings which have been generated by a common (or shared) underlying derivation process[3]. Independent parallelism corresponds to the instantiations of independent derivation processes which are then combined to give the entire derivation of a string[4]. What we are exploring in this paper is an example of a mixed system with both independent and synchronous parallelism.

In (Rambow and Satta, to appear) it is shown that by allowing an unbounded degree of synchronized parallelism we get systems that are too unconstrained. However, interesting subfamilies arise when the synchronous parallelism is bounded to a finite degree, i.e. only a bounded number of subderivations can be synchronized in a given grammar. The system we define has this property.

3 LSTAG

We first look at the formalism of Synchronous TAG (STAG)(Shieber and Schabes, 1990) since it is an example of a tree-rewriting system that has synchronized parallelism.

As a preliminary we first informally define Tree Adjoining Grammars (TAG). For example, Figure 1 shows an example of a tree for a transitive verb *cooked*. Each node in the tree has a unique address obtained by applying a Gorn tree addressing scheme. For instance, the object *NP* has address 2.2. In the TAG formalism, trees can be composed using the two operations of *substitution* (corresponds to string concatenation) and *adjunction* (corresponds to string wrapping). A history of these operations on elementary trees in the form of a derivation tree can be used to reconstruct the derivation of a string recognized by a TAG. Figure 2 shows an example of a derivation tree and the corresponding parse tree for the derived structure obtained when $\alpha(John)$ and $\alpha(beans)$ substitute into $\alpha(cooked)$ and $\beta(dried)$ adjoins into $\alpha(beans)$ giving us a derivation tree for *John cooked dried beans*. Trees that adjoin are termed as *auxiliary trees*, trees that are not auxiliary are called *initial*. Each node in the derivation tree is the name of an elementary tree. The labels on the edges denote the address in the parent node where a substitution or adjunction has occured.

Definition 1 *In a* TAG $G = \{\gamma \mid \gamma$ *is either an* initial *tree or an* auxiliary *tree* $\}$, *we will notate adjunction (similarly substitution) of trees* $\gamma_1 \ldots \gamma_k$ *into tree* γ *at*

[3]The Lindenmayer systems are examples of systems with only synchronous parallelism and it is interesting to note that these L systems have the anti-AFL property (where none of the standard closures apply).

[4]CFG is a formalism that only has independent parallelism.

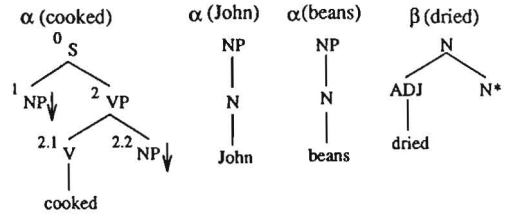


Figure 1: Example of a TAG

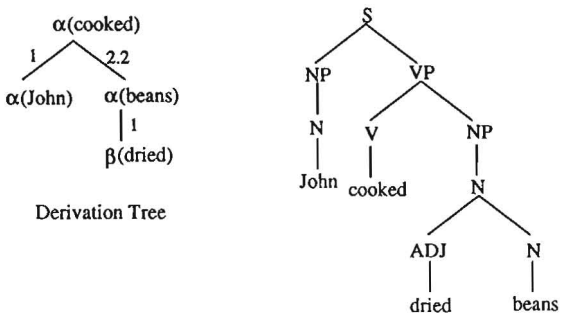


Figure 2: Example of a derivation tree and corresponding parse tree

addresses $a_1 \ldots a_k$ *giving a derived tree* γ' *as*

$$\gamma' = \gamma[a_1, \gamma_1] \ldots [a_k, \gamma_k]$$

Definition 2 *Given two standard* TAGs G_L *and* G_R *we define (from (Shieber, 1994)) a* STAG *as* $\{\langle \gamma, \gamma', \frown \rangle \mid \gamma \in G_L, \gamma' \in G_R\}$, *where* $\frown$ *is a set of* links *from a node address in* γ *to a node address in* γ'. *A derivation proceeds as follows:*

- *for* $\gamma = \langle \gamma_L, \gamma_R, \frown \rangle$, *pick a link member* $a_L \frown_i a_R$, *where the a's are node addresses and* $\frown_i \in \frown$. *For simplicity, we refer to* $\frown$ *as* link *and its elements* $\frown_i$ *as link* members.
- *adjunction (similarly substitution) of* $\langle \beta_L, \beta_R, \frown' \rangle$ *into* γ *is given by*

  $$\langle \gamma'_L, \gamma'_R, \frown'' \rangle = \langle \gamma_L[a_L, \beta_L], \gamma_R[a_R, \beta_R], \frown'' \rangle$$

  where all links in $\frown$ *and* $\frown'$ *are included in* $\frown''$ *except* $\frown_i$.
- $\langle \gamma'_L, \gamma'_R, \frown'' \rangle$ *is now a derived structure which can be further operated upon.*

In (Abeillé, 1992; Abeillé, 1994) STAGs have been used in handling non-local dependencies and to seperate syntactic attachment from semantic roles. However, STAG cannot be used to seperate the dependencies created in (pairs of) derivation trees for coordinate structures from the constituency represented in these derivation trees. In this particular sense, STAG has the same shortcomings of a TAG. Also the above definition of the inheritance of links in derived structures allows STAG to derive strings not generable by TAG (Shieber, 1994). We look at a modified version of STAGs which is weaker in power than STAGs as defined in Defn 2. We call this formalism *Link-Sharing* TAG (LSTAG).

Definition 3 *An* LSTAG G *is defined as a 4-tuple* $\langle G_L, G_R, \Delta, \Phi \rangle$ *where* G_L, G_R *are standard* TAGs, Δ *and* Φ *are disjoint sets of sets of links and for each pair* $\gamma = \langle \gamma_L, \gamma_R \rangle$, *where* $\gamma_L \in G_L$ *and* $\gamma_R \in G_R$, $\delta_\gamma \in \Delta$ *is a subset of* links *in* γ *and* $\phi_{\gamma_R} \in \Phi$ *is a distinguished subset of* links *with the following properties:*

- *for each link* $\frown \in \phi_{\gamma_R}$, $\eta \frown \eta$, *where* η *is a node address in* γ_R. *i.e.* ϕ_{γ_R} *is a set of reflexive links.*
- δ_R *and* ϕ_{γ_R} *have some canonical order* $\prec$.
- *adjunction (similarly substitution) of* $\langle \beta_L, \beta_R \rangle$ *into* γ *is given by*

$$\langle \gamma'_L, \gamma'_R \rangle = \langle \gamma_L[a_L, \beta_L], \gamma_R[a_R, \beta_R] \rangle$$

and for all $\gamma_i \in \delta_\gamma, \beta_i \in \phi_{\beta_R} (1 \leq i \leq n)$ *(card(δ_γ)* $\geq$ *card(β_R))*

$$\delta_\gamma \sqcup \phi_{\beta_R} \stackrel{def}{=} \frown_{\gamma_1} \sqcup \frown_{\beta_1} \cup \ldots \cup \frown_{\gamma_n} \sqcup \frown_{\beta_n}$$

where

$$\frown_{\gamma_1} \prec \frown_{\gamma_2}, \ldots, \frown_{\gamma_{n-1}} \prec \frown_{\gamma_n}$$

and

$$\frown_{\beta_{R_1}} \prec \frown_{\beta_{R_2}}, \ldots, \frown_{\beta_{R_{n-1}}} \prec \frown_{\beta_{R_n}}$$

- $\frown_i \sqcup \frown_j$ *is a set of links defined as follows. If* $a_{L_i} \frown_i a_{R_i}$ *and* $a_{R_j} \frown_j a_{R_j}$, *then*

$$\frown_i \sqcup \frown_j \stackrel{def}{=} \{a_{L_i} \frown a_{R_i}\} \cup \{a_{L_i} \frown a_{R_j}\}$$

- $\langle \gamma'_L, \gamma'_R \rangle$ *is the new derived structure with new set of links* $\delta_\gamma \sqcup \phi_{\beta_R}$.

Φ is used to derive synchronized parallelism in G_R. The ordering $\prec$ is simply used to match up the links being shared via the (non-local) sharing operation $\sqcup$.

This ordering $\prec$ can be defined in terms of node addresses or "first argument $\prec$ second argument", i.e. ordering the arguments of the two predicates being coordinated.

It is important to note that only the links in Φ are used non-locally and they are always exhausted in a single adjunction (or substitution) operation. No links from Δ are ever inherited unlike STAGs. Hence, non-locality is only used in a restricted fashion for the notion of 'sharing'.

4 Linguistic Relevance

To explain how the formalism works consider sentence (2).

(2) John cooks and eats beans.

Consider a LSTAG $G = \{\gamma, \beta, \alpha, \upsilon\}$ partially shown in Fig. 3(a) and Fig. 3(b). α and υ are analogously defined for *John* and *beans* respectively (see Fig. 1). In Fig. 3(a) $\delta_\gamma = \{1, 2\}^5$and $\phi_{\gamma_R} = \{\}$, while for Fig. 3(b) $\delta_\gamma = \{\}$ and $\phi_{\gamma_R} = \{1, 2\}$.

It is important to note that our initial motivation about seperating dependency from the constituency information is highlighted in β (see Fig. 3(b)) where the first projection will only contribute information about constituency in a derivation tree while the second projection will contribute only dependency information in a derivation tree. We conjecture that this is true for all the structures defined in an

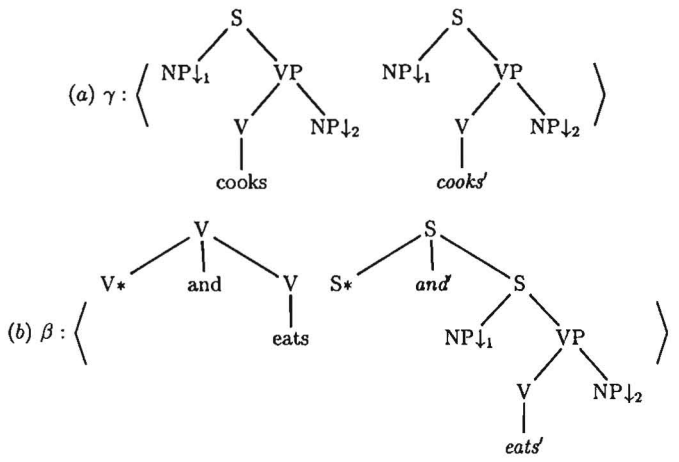


Figure 3: Trees γ and β from LSTAG G

LSTAG. the kind of questions addressed in (Rambow, Vijay-Shanker, and Weir, 1995) can perhaps be answered within the framework of LSTAG[6].

The derived structure after β adjoins onto γ is shown in Fig. 4(a). Fig. 5(a) shows the derived tree after the tree α (for *John*) substitutes into γ. Notice that due to link sharing, substitution is shared, effectively forming a "tangled" derived tree[7]. In Figs. 4 and 5 the derivation trees are also given (associated with each element). The derivation structure for the second element in Fig. 5(b) is a directed acyclic derivation graph which gives us information about dependency we expect. The derivation tree of the first element in Fig. 5(b), on the other hand, gives us information about constituency.

The notion of link sharing is closely related to the schematization of the coordination rule in (Steedman, 1997b) shown below in combinatory notation.

$$\begin{aligned} bxy &\equiv bxy \\ bfg &\equiv \lambda x.b(fx)(gx) \\ bfg &\equiv \lambda x.\lambda y.b(fxy)(gxy) \\ &\ldots \end{aligned}$$

Link sharing is used to combine the interpretation of the predicate arguments f and g (e.g. *cooks*, *eats*) of the conjunction b with the interpretation of the arguments of those predicates $x, y, \ldots$. However, it does this within a tree-rewriting system, unlike the use of combinators in (Steedman, 1997b).

[5]We are just using numbers $1, 2, \ldots$ to denote the links rather than use the Gorn notation to make the trees easier to read. Here, link number 1 stands for $1 \frown 1$ and 2 stands for $2.2 \frown 2.2$

[6]In (Rambow, Vijay-Shanker, and Weir, 1995) a new formalism called D-Tree Grammars was introduced in order to bring together the notion of derivation tree in a TAG with the notion of dependency grammar (Mel'čuk, 1988). Perhaps the kind of questions addressed in (Rambow, Vijay-Shanker, and Weir, 1995) can also be handled using the current framework. Such an application of the formalism would motivate the need for trees like γ in Fig. 3 independent of the coordination facts since they would be required to get the dependencies right.

[7]While this notion of sharing bears some resemblance to the notion of *joining node* in the three-dimensional trees used in (Moltmann, 1992; Muadz, 1991) the rules for semantic interpretation of the derivations produced in a LSTAG is considerably less obscure than the rules needed to interpret 3D trees; crucially because elementary structures in a TAG-like formalisms are taken to be semantically minimal without being semantically void.

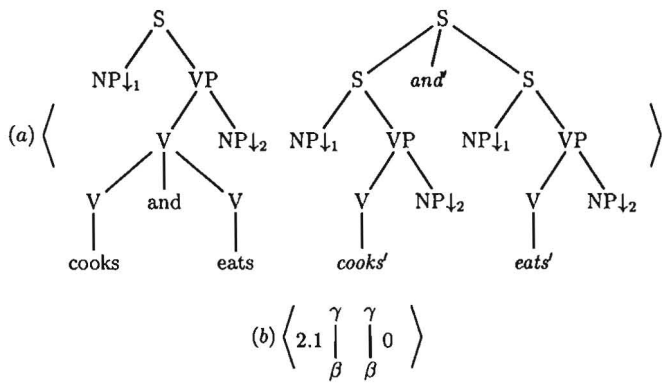


Figure 4: Derived and derivation structures after β adjoins into γ.

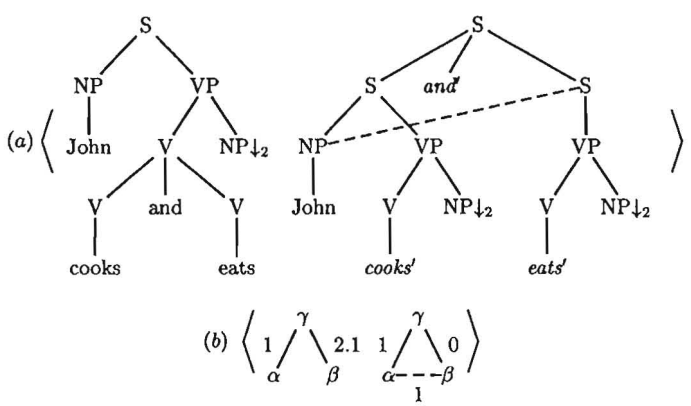


Figure 5: Substitution of α

5 Restrictions

Having defined the formalism of LSTAG, we now define certain restrictions on the grammar that can be written in this formalism in order to capture correctly certain facts about coordinate structures in English.

For instance, we need to prohibit elementary structures like the one in Fig. 6 because they give rise to ungrammatical sentences like (3).

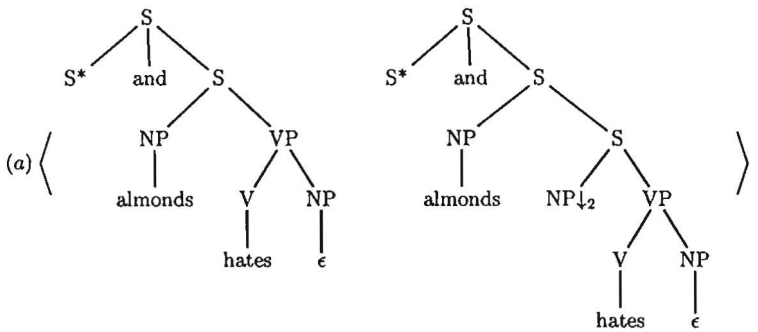


Figure 6: Discontiguous elementary structure

(3) *Peanuts John likes and almonds hates. (Joshi, 1990)

However, such restrictions in the context of TAGs have been discussed before. (Joshi, 1990) rules out (3) by stating a requirement on the lexical string spelled out by the elementary tree. If the lexical string spelled out is not contiguous then it cannot coordinate. This requirement is stated to be a phonological condition and relates the notion of an *intonational phrase* (IP) to the notion of appropriate fragments for coordination (in the spirit of (Steedman, 1997a)). It is important to note that the notions of phrase structure for coordination and intonational phrases defined in (Joshi, 1990) for TAG are not identical, whereas they *are* identical for CCG (Steedman, 1997a).

We can state an analogous restriction on the formation of elementary structures in a LSTAG, one that is motivated by the notion of link sharing. The left element of an elementary structure in a LSTAG cannot be composed of discontinuous parts of the right element. For example, in Fig. 6 the segment $[_S[_{NP_\downarrow}][_{VP}]]$ from the right element has been excised in the left element. This restriction corresponds to the notion that the left element of a structure in a LSTAG represents constituency.

6 Conclusion

We have presented a new tree-rewriting formalism called Link-Sharing Tree Adjoining Grammar (LSTAG) which is a variant of synchronous TAGs (STAG). Using LSTAG we defined an approach towards coordination where linguistic dependency is distinguished from the notion of constituency. Appropriate restrictions on the nature of elementary structures in a LSTAG were also defined. Such an approach towards coordination that explicitly distinguishes dependencies from constituency gives a better formal understanding of its representation when compared to previous approaches that use tree-rewriting systems which conflate the two issues (see (Joshi and Schabes, 1991; Sarkar and Joshi, 1996)). The previous approaches had to represent coordinate structures either with unrooted trees or by performing structure merging on the parse tree. Moreover, the linguistic analyses presented in (Joshi

and Schabes, 1991; Sarkar and Joshi, 1996) can be easily adopted in the current formalism.

References

Abeillé, Anne. 1992. Synchronous TAGs and French Pronominal Clitics. In *Proc. of COLING-92*, pages 60–66, Nantes, Aug 23–28.

Abeillé, Anne. 1994. Syntax or Semantics? Handling Nonlocal Dependencies with MC-TAGs or Synchronous TAGs. *Computational Intelligence*, 10(4):471–485.

Engelfriet, J., G. Rozenberg, and G. Slutzki. 1980. Tree transducers, L systems, and two-way machines. *Journal of Computer and System Science*, 43:328–360.

Frank, Robert. 1992. *Syntactic locality and Tree Adjoining Grammar: grammatical, acquisition and processing perspectives.* Ph.D. thesis, University of Pennsylvania,IRCS-92-47.

Joshi, A. and Y. Schabes. 1991. Tree adjoining grammars and lexicalized grammars. In M. Nivat and A. Podelski, editors, *Tree automata and languages.* North-Holland.

Joshi, Aravind. 1990. Phrase Structure and Intonational Phrases: Comments on the papers by Marcus and Steedman. In G. Altmann, editor, *Computational and Cognitive Models of Speech.* MIT Press.

Joshi, Aravind and Yves Schabes. 1991. Fixed and flexible phrase structure: Coordination in Tree Adjoining Grammar. In *Presented at the DARPA Workshop on Spoken Language Systems*, Asilomar, CA.

Joshi, Aravind K., L. Levy, and M. Takahashi. 1975. Tree Adjunct Grammars. *Journal of Computer and System Sciences.*

Kroch, A. 1987. Subjacency in a tree adjoining grammar. In A. Manaster-Ramer, editor, *Mathematics of Language.* J. Benjamins Pub. Co., pages 143–172.

Mel'ĉuk, I. 1988. *Dependency Syntax: Theory and Practice.* State University of New York Press, Albany.

Moltmann, Friederike. 1992. On the Interpretation of Three-Dimensonal Syntactic Trees. In Chris Barker and David Dowty, editors, *Proc. of SALT-2*, pages 261–281, May 1-3.

Muadz, H. 1991. *A Planar Theory of Coordination.* Ph.D. thesis, University of Arizona, Tucson, Arizona.

Rambow, O. and A. Joshi. 1992. A formal look at dependency grammars and phrase-structure grammars, with special consideration to word-order phenomena. In *Intern. Workshop on the Meaning-Text Theory*, pages 47–66, Arbeitspapiere der GMD 671. Darmstadt.

Rambow, O. and G. Satta. to appear. Independent parallelism in finite copying parallel rewriting systems. *Theor. Comput. Sc.*

Rambow, O., K. Vijay-Shanker, and D. Weir. 1995. D-Tree Grammars. In *Proceedings of the 33rd Meeting of the ACL.*

Sarkar, Anoop and Aravind Joshi. 1996. Coordination in TAG: Formalization and implementation. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'96)*, Copenhagen.

Shieber, S. 1994. Restricting the weak generative capacity of synchronous tree adjoining grammars. *Computational Intelligence*, 10(4):371–385, November.

Shieber, Stuart and Yves Schabes. 1990. Synchronous Tree Adjoining Grammars. In *Proceedings of the* 13^{th} *International Conference on Computational Linguistics (COLING'90)*, Helsinki, Finland.

Steedman, Mark. 1985. Dependency and coordination in the grammar of Dutch and English. *Language*, 61:523–568.

Steedman, Mark. 1997a. Information Structure and the Syntax-Phonology Interface. manuscript. Univ. of Pennsylvania.

Steedman, Mark. 1997b. *Surface Structure and Interpretation: Unbounded and Bounded Dependency in Combinatory Grammar.* Linguistic Inquiry monograph. MIT Press.

Some algebraic properties of higher order modifiers

R. Zuber
CNRS, Paris
rz@ccr.jussieu.fr

Modifiers are expressions of category C/C, for any C: they combine with expressions of a category C to form others still in the category C. Thus depending on the choice of C there are various modifiers. Informally a modifier is *a higher order modifier*, HOM for short, if its argument expression denotes an object of an order higher than the order of sets of individuals. Thus adjectives and adverbs are not HOMs since their arguments denote properties. In this paper I will show (1) the existence of HOMs, more specifically of modifiers modifying determiners and (2) propose an algebraic analysis of them extending the analysis of "simple" modifiers as proposed by Keenan. The algebraic notion of *atom* and, consequently, of atomicity of the relevant Boolean algebras will play an essential role in this analysis.

The theoretical background is that of the theory of generalized quantifiers in conjunction with Boolean semantics as developed by Keenan (Keenan 1983, Keenan and Faltz 1985). This means in particular that all logical types D_C (denotations of expressions of the category C) form atomic (and complete) Boolean algebras and the partial order defined in them can be interpreted as the relation of *generalized entailment*. This relation is defined for any category C. The set of functions from the algebra A onto the algebra B is noted AfB. Thus a modifier of category C/C denotes in $D_{C/C}$. Given that C can vary this means that there might exists *categorially ambiguous* modifiers. This is indeed the case, in particular with so-called *focus particles*, as we will see.

Not all elements of the set CfC are necessary for the interpretation of natural language modifiers. For instance concerning (extensional) adjectival and adverbial modifiers Keenan distinguishes a specific set $R(P)$ of (positively) *restricting functions* which interpret such modifiers. Thus $f \in BfB$ is positively restrictive, relative to algebra B, iff $f(x) \leq x$ for any $x \in B$. The set $R(B)$ of all restricting functions (relative to a given algebra B) forms a Boolean algebra (denoted by $R(B)$) with the operation of meet and join defined pointwise and the complement f' of f is defined as $f'(x) = x \cap (f(x))'$, where $(f(x))'$ is the complement of $f(x)$ in B. A sub-class of restricting functions is constituted by *(positively) intersecting functions* $INT(B)$ defined as: $f \in INT$ iff $f(x) = x \cap f(1_B)$ for all $x \in B$. Thus any positively intersecting function is a positively restricting function and, in addition, the set $INT(B)$, considered as an algebra, is a sub-algebra of the corresponding restricting algebra $R(B)$

(Keenan 19983).

Some additional tools are needed: a class $NR(B)$ of modifier interpreting functions called *negatively restricting*, and its sub-class $NINT(B)$ of *negatively intersecting* functions. By definition (cf. Zuber 1997a) $f \in NR(B)$ iff $f(x) \leq x'$ for any $x \in B$. The set $NR(B)$ forms a Boolean algebra with operations as in BfB except that the complement operation is relativised to the negative identity function $f(x) = x'$. Thus the algebra $NR(B)$ is a factor algebra (of B) generated by the negative identity function. If B is atomic then $NR(B)$ is atomic and if B is complete then $NR(B)$ is complete. The set $NINT(B)$ is defined as: $f \in NINT(B)$ iff $f(x) = x' \cap f(0_B)$, for any $x \in B$. Thus negatively intersecting functions are negatively restricting and, in addition one can show that $NINT(B)$ forms a sub-algebra of $NR(B)$ and that $NINT(B)$ is isomorphic to B (Zuber 1997a). A simple class of positively intersecting and negatively intersecting functions is indicated in

Fact 1 *Let* **B** *be a Boolean algebra. Then for any* $\alpha \in B$ *the function* $f_\alpha(x) = x \cap \alpha$ *is positively intersecting and the function* $g_\alpha(x) = x' \cap \alpha$ *is negatively intersecting (in* **B***)*

I will also make use of the notion of *intersective* and *co-intersective* functions as defined by Keenan (cf. Keenan 1993). By definition, F, a function of type $< 1, 1 >$, is intersective iff for all properties X, Y, V, Z, if $X \cap Y = V \cap Z$ then $F(X)(Y) = F(V)(Z)$. Similarly, F is co-intersective iff for all properties X, Y, V, Z if $X - Y = V - Z$ then $F(X)(Y) = F(V)(Z)$. In particular intersective functions interpret determiners in exclusion clauses. Thus the determiner *No...but A* is interpreted by the intersective function F such that $F(X)(Y)$ iff $X \cap Y = A$ and in addition this function is an atom in the algebra of intersective functions. Similarly the determiner *All...but A* is interpreted by the co-intersective function F such that $F(X)(Y)$ iff $X - Y = A$; this function is an atom in the algebra of co-intersective functions.

There are many constructions involving HOMs. For instance in slavic languages there is a specific comparative construction which can be considered as a modifier of the adjectival modifier (cf. Zuber 1997b). Among all possible constructions involving HOMs I consider here mainly modifiers one finds in exclusion ($EXCL$) clauses and in inclusion ($INCL$) clauses and in some related constructions. The $EXCL$ clauses are represented by schemas like *NP but/except E* as instanciated in (1). Similarly $INCL$ clauses, represented by schemas like *NP including E*, are instanciated in (2):

(1) NP but/except E
(1a) All students but Leo were sleeping
(1b) All students but five failed
(1c) No teacher but Lea and Leo drank
(1d) All student but the Albanian (students) went to the library
(2) NP, including E
(2a) Most students, including Max were singing
(2b) Some students, including the Albanian ones, were unhappy
(2c) All teachers, including the oldest one, were at the party
(2d) Five teachers, including Leo, slept in the library

(2e) All students, including the five at the back, were listening

In the above schemas (1) and (2), the NP is the first argument of the $EXCL$ or $INCL$ clause and the expression E is the second argument. As we will se the expression E will be considered as being logically complex.

$EXCL$, but not $INCL$, clauses have been extensively studied. One notices similar problems in both cases: thus, there are severe restrictions on the type of expressions which can occur as the first argument in the above schemas: these are so-called quantifier constraints. For instance it is well-known that in $EXCL$ clauses only the quantifiers *all/each* and *no* can occur on this position. In $INCL$ clauses there are similar, although weaker, restrictions: they cannot contain monotone decreasing quantifiers on this position:

(3) *No student/?at most three students, including Leo went to the pool

My analysis basically takes into account the case of $EXCL$ clauses and the case of $INCL$ clauses. I also indicate how the more general case of focus particles which are logically related to these clauses can be treated.

One can distinguish two approaches to $EXCL$ clauses. Keenan considers that they result from the application of a discontinous determiner to a common noun. Thus *All students but Leo* is a result of the application of the (discontinous) determiner *All...but Leo* to the common noun *students*. Such determiners denote co-intersective functions and consequently the NP corresponding to the $EXCL$ clause denotes the value of this function at the property corresponding to *students*. In fact Keenan shows that in general exclusion determiners denote in the set of intersective or co-intersective functions (Keenan 1993). Of course his approach is compatible with an approach in which the analysis of exclusion determiners is pushed further to the point showing their syntactic or semantic composition.

Under the second approach, proposed in particular by Moltmann (Moltmann 1995, Moltmann 1996), the $EXCL$ clauses result, syntactically, by the application of some functional expressions to NPs: one gets a NP in the form of an $EXCL$ clause by applying the "complement expression" *but/except NP* to a quantified NP (in fact to NPs of the form *All CN* or *No CN*). So, although in general NPs are rarelt modified, we have to do in this case, according to Moltmann, with a modification of NPs.

I am going to suggest that the modification also takes place in $EXCL$ (and $INCL$) clauses but what is modified is not an NP but a determiner. Furthermore, the determiner which is modified occurs as a "logical constituant" in the second argument in $EXCL$ or $INCL$ clauses: it is a a "logical part" of the expression E in the above schemas. Thus exclusion determiners that Keenan treats globally are complex determiners obtained by a modification of simpler ones and the modifier corresponds to the function interpreting the expression *No... but/except* or *All...but/except.*

From a purely formal point of view it is not important how functional depen-

dence is established in a complex expression in which various elements can be considered as arguments or as functions. However, if we consider that it is the expression E (or rather its part) which varies in $EXCL$ clauses then the range of possible arguments is much greater than if it is the first NP that varies. Indeed, in $EXCL$ clauses the variation of the first argument, the quantified NP, is very limited, which is counter-intuitive for an expression to be considered as argument.

To determine the logical form of the expression E and the part of it which is modified, one observes that in the following examples the expressions in (a) are equivalent to those in (b):

(4a) No student but Leo
(4b) No student but Leo who is a student
(5a) All students but five
(5b) All students but the five students
(6a) All students but the Albanian ones
(6b) All students but the Albanians who are students

These and similar observations indicate that the second argument in exclusion clauses (the expression E in the sechema) are definite NPs, i.e. NP which are interpreted by filters. This is even more obvious given that (7) is not acceptable in comparison with (8):

(7) *Most students, including five
(8) Most students, including the five (at the back)

Given that in both arguments in an $EXCL$ clause we have the same common noun which can vary, the modified element is not a definite NP but a determiner creating such an NP. For instance the clauses in (4a) and (4b) correspond to (9), where X varies over properties:

(9) No (X) but Leo, who is (X)

Thus, I claim that semantically the second argument is a quantifier of type $< 1,1 >$ which I call *filter creating function* (or FCF). By definition, for any property A, $f_A \in FCF$ iff $f_A(X) = \emptyset$ if A is not a subset of X and otherwise it is equal to the filter generated by A. So the expression E will be interpreted by a FCF determined by a property A. This property is the specific propery indicating exception or inclusion. For instance in (1a) and (4b) it corresponds to the singleton whose only element is the referent of Leo, in (1c) it corresponds to the set of two elements, roughly **Leo** and **Lea** and in (2b) and (6b) the property A corresponds to a set of specific Albanian students.
Concerning FCF one proves the following:

Fact 2 *If $f \in FCF$ then f is intersective*

Thus $EXCL$ clauses can be considered as resulting from the application of the modifier *All... but/except* or *No... but/except* to an expression denoting an intersective function. So we have two types of exclusion modifiers and the

functions they denote: those which are based on *No* and those which are based on *All*. Observe now that when the function is based on the quantifier *No*, it is a restricting function, and when it is based on *All*, it is a negatively restricting function. For instance (4a) entails *Leo, who is a student* and (6a) entails *not the Albanian students*. Furthermore, the property A determines not only an intersective function but also atoms of the algebra of intersective functions and of co-intersective functions: for any property A the function $i_A(X)(Y) = 1$ iff $X \cap Y = A$ is an atom (based on A) in the algebra of intersective functions and the function $c_A(X)(Y) = 1$ iff $X - Y = A$ is an atom (based on A) in the algebra of co-intersective functions (Keenan 1993). It follows from this and the semantics of exclusion clauses that

Fact 3 $\mathbf{NO-but\ (F_A)}$ *is the atom based on* **A** *of the intersective algebra and* $\mathbf{ALL-but\ (F_A)}$ *is the atom based on* **A** *of the co-intersective algebra.*

So the restricting function interpreting the exclusion modifier based on **No**, the function $\mathbf{NO-but}$ associates with any $F_A \in FCF$ the atom based on **A** (and thus contained in F_A) of the algebra of intersective functions. Similarly the negatively restricting function based on **All** interpreting the exclusion modifier, the function $\mathbf{ALL-but}$ associates with any $F_A \in FCF$ the atom based on **A** (and thus not contained in F_A) of the algebra of co-intersective functions. Furthermore, the restricting function based on **No** is not intersecting since it is not monotone increasing. The negatively restricting function based on **All** is not negatively intersecting since it is not monotone decreasing.

In many languages there exists a lexicalized modifier which corresponds, under one of its categorization, to the modifier $\mathbf{No-but}$. In English it is the "particle" *only*. It is easy to show that with the appropriate categorisation *No student but Leo* is equivalent to *only the student who is Leo*. As a restricting function, and thus an element of the Boolean algebra, **Only** has a negation, which, interestingly enough, is the denotation of the "particle" *Also*. Thus we have $\mathbf{NO-but(F_A) = ONLY(F_A)}$ and $\mathbf{ONLY'(F_A) = ALSO(F_A)}$.

The above observation will help us to analyse $INCL$ clauses. One might think that the simplest way to represent them is to use the fact 1: for instance the clause *Some(X), including A* would be represented by a positively intersecting function based on *Some* defined as: $\mathbf{SOME-incl(F_A) = SOME \cap F_A}$. Similar definitions can be given for including functions interpreting *Most..., including A* or *At least five..., including A*. There seems to be, however, an empirical problem with such representations since they do not account for the fact that including clauses in many cases must be interpreted as involving the universe which contains more elements than the universe of just the determiners on which such clauses are based. Thus the clauses *All/most/some/ students, including Leo and Lea* all entail that there are at least three students. In other words the "inclusion" in $INCL$ clauses is a "strict inclusion". If this second interpretation is accepted (which is possible in all cases, whereas non-strict inclusion isn't) one obtains interesting algebraic properties of $INCL$ clauses and an interesting relationship between them and $ECXCL$ clauses.

Observe that the "strict inclusion" interpretation is the interpretation which entails the negation of the exclusion modifier based on *No* and thus it entails

Also, the negation of *only*. For instance *All/most/some (X), including Leo* all entail *not only (X) who is Leo*. So, a bit more formally any $INCL$ clause of the form $\mathbf{D-incl(F_A)}$, with $\mathbf{D}$, an appropriate determiner, should entail $\mathbf{ONLY'(F_A)}$. From this follows

Fact 4 *If* $\mathbf{D-incl}$ *is a function denoted by the inclusion modifier based on the det* $\mathbf{D}$, *then* $\mathbf{D-incl(F_A) = D \cap ONLY'(F_A) = D \cap F_A \cap (NO-but(F_A))'}$.

Thus, informally, any function denoted by an inclusion modifier is a positively restricting function which associates with any filter creating function $\mathbf{F_A}$ the meet of this function and of the co-atom determined by $\mathbf{A}$.

There are also *negative inclusion*, $NINCL$, clauses such as *No student, not even Leo* or, maybe, *Few students, in particular not Albanian students*. For their description we need the notion of the *postnegation* $\mathbf{D-not}$ of a quantifier of type $< 1,1 >$: for any $\mathbf{X}$, $\mathbf{D-not(X) = D(X)-not}$, where $\mathbf{D(X)-not}$ is the postnegation of the quantifier of type $< 1 >$.. Now, one notices that all $NINCL$ clauses above entail the post-negation of *Not only the student Leo* (or, roughly, *No student, even not Leo* entails *Also the student who is Leo (did) not*). By analogy with positive $INCL$ clauses we have

Fact 5 *If* $\mathbf{D-negincl}$ *is a function denoted by the negative inclusion modifier based on the det* D, *then* $\mathbf{D-negincl(F_A) = D \cap ALSO(F_A)-not = D \cap (F_A-not) \cap ALL'-but(F_A)}$, *where* $\mathbf{ALL'-but}$ *is the complement of the restricting function* $\mathbf{ALL-but}$

Thus negative inclusion clauses such as *No, even not A*, or *Few, not even A*, are interpreted by negatively restricting functions. One has to check whether these functions are negatively intersecting.

There are restrictions on the type of the determiner D which occur in expressions discussed in the fact 3, 4 and 5. Various known constraints on possible occurrences can be explained by languages universals concerning the co-directionality of monotonicity which must take place in some conjunctions. If $INCL$ or $EXCL$ clauses contain connectives (*including* or *except*) then it is possible that these connectives, like standard Boolean connectives force particular arguments.

$EXCL$ and $INLC$ clauses, which have been shown to contains HOMs and thus to involve restrictive functions in their interpretation, have been related to focus particles. These are well-known to be categorially ambiguous (being HOMs) expressions. The algebraic analysis proposed for $EXCL$ and $INCL$ clauses extends easily to "logical' focus particles of any category like *Only* and *Also*: they also are interpreted by restricting functions and these functions essentially involve the atomicity of the corresponding denotational algebra. For instance for the particle $ONLY$ which can have in its scope expressions of various categories, we have

Fact 6 *If the scope of Only is E of the category* $\mathbf{C}$, *and Only(E) is also of the category* $\mathbf{C}$ *then* $\mathbf{ONLY(E)}$ *is an atom in the algebra* $\mathbf{D_C}$.

Similarly with the categorially ambiguous particle $ALSO$: when it is applied to an expression of the category C, the resulting expressions denotes co-atoms

of the algebra D_C.

The case of the particle *Even* is more complicated, since this particle has a strong pragmatic import. The "purely logical" content of *Even* can be captured by considering that $\mathbf{Even(F_A(X))}$ is equivalent to $\mathbf{All(X), including(F_A(X))}$.

I would like to conclude by makin some general remarks concerning the above results. . The expressions in $EXCL$ and $INCL$ clauses which have been interpreted by restricting functions are still complex exprerssions. Thus *No/all but* are composed of the determiner and the connective *but.* Similarly with inclusion clauses, where one finds the connective *including.* So one can ask what is the meaning of these connectives and whether they are binary or unary. The results presented above show that, informally, we have with HOMs four logically related situations represented by the following schemas:

(10) A and only A
(11) A and not only A
(12) Not-A and only not-A
(13) Not-A and not only not-A

These schemas strongly remind us of the traditional square of oppositions. Furthermore, looking at these schemas one notices that HOMs involve unary, and not binary, connectives, since only one variable is involved. Furthermore, the connectives represented by the above schemas neither correspond directly to nor are generalisations of the classical unary propositional connectives. One can easily check that in the algebra of functions which interpret classical unary propositional connectives all positively restrincting functions are positively intersecting and all negatively restricting ones are negatively intersecting. The connectives represented by the schemas above are not intersecting since they are neither monotone increasing nor monotone decreasing.

My second remark is related to the preceding one. Keenan conjectured that to interpret (extensional) natural language modifiers one needs only positively restricting functions. He considered, however, mainly modifiers of lower order. Keenan's claim has been challenged precisely in the case of HOMs (Zuber 1996). Indeed, the analysis presented here might suggest that we need also negatively restricting functions. For instance the function **All...but** is negatively restricting. This is not obvious, however, since we have the following:

Fact 7 $\mathbf{F}$ *is intersective iff* $\mathbf{F - not}$ *is co-intersective*

From this fact it fallows that the quantifier $\mathbf{All - but(F_A)}$ is equivalent to $\mathbf{No - but(F_A - not)}$. In other words, the negatively restricting function interpreting determiner modifiers can be replaced by a positively restricting function applied to the postnegation of the argument. Whether a similar way out is always possible is another question.

References

[1] Keenan, E.L. (1983) Boolean Algebra for linguists, *Working Papers in Linguistics*, UCLA

[2] Keenan, E.L. (1993) Natural Language, Sortal Reducibility and Generalized Quantifiers, *J. of Symbolic logic* 58-1, p.314-325

[3] Keenan, E.L. and Faltz, L.M. (1985) *Boolean Semantics for Natural Language,* D. Reidel Publishing Company, Dordrecht

[4] Moltmann, F. (1995) Exception sentences and polyadic quantification, *Linguistics and Philosophy*

[5] Moltmann, F. (1996) Resumptive Quantifiers in Exception Sentences, in Kanazawa, M. et al. (eds.) *Quantifiers, Deduction, and Context*, CSLI Publications, Stanford, p.139-170

[6] Zuber, R. (1996) 81. Two Semantic Components of Noun Phrases, in Dubach Green, A. and Motapanyane, V. (eds). *Proceedings of ESCOL'96*, Cornell University Press, 1996, p. 347-354

[7] Zuber, R. (1997a) On negatively restricting Boolean algebras, *Bulletin of the section of logic*, 26-1, p. 50-54

[8] Zuber, R. (1997b) The category of modifiers and comparatives in Polish, in Junhganns, U. and Zybatov, G. (eds.) *Formale Slavistik*, Verveurt Verlag, Frankfurt am Main, 1997, p. 523-532

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

-Bibliothek, Information
und Dokumentation (BID)-
PF 2080
67608 Kaiserslautern
FRG

Telefon (0631) 205-3506
Telefax (0631) 205-3210

e-mail
dfkibib@dfki.uni-kl.de
WWW
http://www.dfki.uni-sb.de/dfkibib

Veröffentlichungen des DFKI

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder (so sie als per ftp erhaeltlich angemerkt sind) per anonymous ftp von ftp.dfki.uni-kl.de (131.246.241.100) im Verzeichnis pub/Publications bezogen werden. Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or (if they are marked as obtainable by ftp) by anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) in the directory pub/Publications.
The reports are distributed free of charge except where otherwise noted.

DFKI Research Reports

1997

RR-97-04
Serge Autexier, Dieter Hutter
Parameterized Abstractions used for Proof-Planning
13 pages

RR-97-03
Dieter Hutter
Using Rippling to Prove the Termination of Algorithms
15 pages

RR-97-02
Stephan Busemann, Thierry Declerck, Abdel Kader Diagne, Luca Dini, Judith Klein, Sven Schmeier
Natural Language Dialogue Service for Appointment Scheduling Agents
15 pages

RR-97-01
Erica Melis, Claus Sengler
Analogy in Verification of State-Based Specifications: First Results
12 pages

1996

RR-96-06
Claus Sengler
Case Studies of Non-Freely Generated Data Types
200 pages

RR-96-05
Stephan Busemann
Best-First Surface Realization
11 pages

RR-96-04
Christoph G. Jung, Klaus Fischer, Alastair Burt
Multi-Agent Planning
Using an *Abductive*
EVENT CALCULUS
114 pages

RR-96-03
Günter Neumann
Interleaving
Natural Language Parsing and Generation
Through Uniform Processing
51 pages

RR-96-02
E.André, J. Müller , T.Rist:
PPP-Persona: Ein objektorientierter Multimedia-Präsentationsagent
14 Seiten

RR-96-01
Claus Sengler
Induction on Non-Freely Generated Data Types
188 pages

1995

RR-95-20
Hans-Ulrich Krieger
Typed Feature Structures, Definite Equivalences, Greatest Model Semantics, and Nonmonotonicity
27 pages

RR-95-19
Abdel Kader Diagne, Walter Kasper, Hans-Ulrich Krieger
Distributed Parsing With HPSG Grammar
20 pages

RR-95-18
Hans-Ulrich Krieger, Ulrich Schäfer
Efficient Parameterizable Type Expansion for Typed Feature Formalisms
19 pages

RR-95-17
Hans-Ulrich Krieger
Classification and Representation of Types in TDL
17 pages

RR-95-16
Martin Müller, Tobias Van Roy
Title not set
0 pages

Note: The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

RR-95-15
Joachim Niehren, Tobias Van Roy
Title not set
0 pages

Note: The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

RR-95-14
Joachim Niehren
Functional Computation as Concurrent Computation
50 pages

RR-95-13
Werner Stephan, Susanne Biundo
Deduction-based Refinement Planning
14 pages

RR-95-12
Walter Hower, Winfried H. Graf
Research in Constraint-Based Layout, Visualization, CAD, and Related Topics: A Bibliographical Survey
33 pages

RR-95-11
Anne Kilger, Wolgang Finkler
Incremental Generation for Real-Time Applications
47 pages

RR-95-10
Gert Smolka
The Oz Programming Model
23 pages

RR-95-09
M. Buchheit, F. M. Donini, W. Nutt, A. Schaerf
A Refined Architecture for Terminological Systems: Terminology = Schema + Views
71 pages

RR-95-08
Michael Mehl, Ralf Scheidhauer, Christian Schulte
An Abstract Machine for Oz
23 pages

RR-95-07
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt
The Complexity of Concept Languages
57 pages

RR-95-06
Bernd Kiefer, Thomas Fettig
FEGRAMED
An interactive Graphics Editor for Feature Structures
37 pages

RR-95-05
Rolf Backofen, James Rogers, K. Vijay-Shanker
A First-Order Axiomatization of the Theory of Finite Trees
35 pages

RR-95-04
M. Buchheit, H.-J. Bürckert, B. Hollunder, A. Laux, W. Nutt,
M. Wójcik
Task Acquisition with a Description Logic Reasoner
17 pages

RR-95-03
Stephan Baumann, Michael Malburg, Hans-Guenther Hein, Rainer Hoch,
Thomas Kieninger, Norbert Kuhn
Document Analysis at DFKI
Part 2: Information Extraction
40 pages

RR-95-02
Majdi Ben Hadj Ali, Frank Fein, Frank Hoenes, Thorsten Jaeger,
Achim Weigel
Document Analysis at DFKI
Part 1: Image Analysis and Text Recognition
69 pages

RR-95-01
Klaus Fischer, Jörg P. Müller, Markus Pischel
Cooperative Transportation Scheduling
an application Domain for DAI
31 pages

1994

RR-94-39
Hans-Ulrich Krieger
Typed Feature Formalisms as a Common Basis for Linguistic Specification.
21 pages

RR-94-38
Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne,
Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger,
Klaus Netter, Günter Neumann, Stephan Oepen, Stephen P. Spackman.
DISCO–An HPSG-based NLP System and its Application for Appointment Scheduling.
13 pages

RR-94-37
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 1: Overview.
54 pages

RR-94-36
Manfred Meyer
Issues in Concurrent Knowledge Engineering. Knowledge Base and Knowledge Share Evolution.
17 pages

RR-94-35
Rolf Backofen
A Complete Axiomatization of a Theory with Feature and Arity Constraints
49 pages

RR-94-34
Stephan Busemann, Stephan Oepen, Elizabeth A. Hinkelman,
Günter Neumann, Hans Uszkoreit
COSMA – Multi-Participant NL Interaction for Appointment Scheduling
80 pages

RR-94-33
Franz Baader, Armin Laux
Terminological Logics with Modal Operators
29 pages

RR-94-31
Otto Kühn, Volker Becker, Georg Lohse, Philipp Neumann
Integrated Knowledge Utilization and Evolution for the Conservation of Corporate Know-How
17 pages

RR-94-23
Gert Smolka
The Definition of Kernel Oz
53 pages

RR-94-20
Christian Schulte, Gert Smolka, Jörg Würtz
Encapsulated Search and Constraint Programming in Oz
21 pages

RR-94-19
Rainer Hoch
Using IR Techniques for Text Classification in Document Analysis
16 pages

RR-94-18
Rolf Backofen, Ralf Treinen
How to Win a Game with Features
18 pages

RR-94-17
Georg Struth
Philosophical Logics—A Survey and a Bibliography
58 pages

RR-94-16
Gert Smolka
A Foundation for Higher-order Concurrent Constraint Programming
26 pages

RR-94-15
Winfried H. Graf, Stefan Neurohr
Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces
20 pages

RR-94-14
Harold Boley, Ulrich Buhrmann, Christof Kremer
Towards a Sharable Knowledge Base on Recyclable Plastics
14 pages

RR-94-13
Jana Koehler
Planning from Second Principles—A Logic-based Approach
49 pages

RR-94-12
Hubert Comon, Ralf Treinen
Ordering Constraints on Trees
34 pages

RR-94-11
Knut Hinkelmann
A Consequence Finding Approach for Feature Recognition in CAPP
18 pages

RR-94-10
Knut Hinkelmann, Helge Hintze
Computing Cost Estimates for Proof Strategies
22 pages

RR-94-08
Otto Kühn, Björn Höfling
Conserving Corporate Knowledge for Crankshaft Design
17 pages

RR-94-07
Harold Boley
Finite Domains and Exclusions as First-Class Citizens
25 pages

RR-94-06
Dietmar Dengler
An Adaptive Deductive Planning System
17 pages

RR-94-05
Franz Schmalhofer, J. Stuart Aitken, Lyle E. Bourne jr.
Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

RR-94-03
Gert Smolka
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

RR-94-02
Elisabeth André, Thomas Rist
Von Textgeneratoren zu Intellimedia-Präsentationssystemen
22 Seiten

RR-94-01
Elisabeth André, Thomas Rist
Multimedia Presentations: The Support of Passive and Active Viewing
15 pages

DFKI Technical Memos

1996

TM-96-02
Harold Boley
Knowledge Bases in the World Wide Web:
A Challenge for Logic Programming
8 pages

TM-96-01
Gerd Kamp, Holger Wache
CTL — a description Logic with expressive concrete domains
19 pages

1995

TM-95-04
Klaus Schmid
Creative Problem Solving
and
Automated Discovery
— An Analysis of Psychological and AI Research —
152 pages

TM-95-03
Andreas Abecker, Harold Boley, Knut Hinkelmann, Holger Wache,
Franz Schmalhofer
An Environment for Exploring and Validating Declarative Knowledge
11 pages

TM-95-02
Michael Sintek
FLIP: Functional-plus-Logic Programming
on an Integrated Platform
106 pages

TM-95-01
Martin Buchheit, Rüdiger Klein, Werner Nutt
Constructive Problem Solving: A Model Construction Approach towards Configuration
34 pages

1994

TM-94-05
Klaus Fischer, Jörg P. Müller, Markus Pischel
Unifying Control in a Layered Agent Architecture
27 pages

TM-94-04
Cornelia Fischer
PAntUDE – An Anti-Unification Algorithm for Expressing Refined Generalizations
22 pages

TM-94-03
Victoria Hall
Uncertainty-Valued Horn Clauses
31 pages

TM-94-02
Rainer Bleisinger, Berthold Kröll
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

TM-94-01
Rainer Bleisinger, Klaus-Peter Gores
Text Skimming as a Part in Paper Document Understanding
14 pages

DFKI Documents

1997

D-97-03
Andreas Abecker, Stefan Decker, Knut Hinkelmann, Ulrich Reimer
Proceedings of the Workshop „Knowledge-Based Systems for Knowledge Management in Enterprises“ 97
167 pages

D-97-01
Thomas Malik
NetGLTool Benutzeranleitung
40 Seiten

1996

D-96-07
Technical Staff
DFKI Jahresbericht 1995
55 Seiten

Note: This document is no longer available in printed form.

D-96-06
Klaus Fischer (Ed.)
Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems
63 pages

D-96-05
Martin Schaaf
Ein Framework zur Erstellung verteilter Anwendungen
94 pages

D-96-04
Franz Baader, Hans-Jürgen Bürckert, Andreas Günter, Werner Nutt (Hrsg.)
Proceedings of the Workshop on Knowledge Representation and Configuration WRKP'96
83 pages

D-96-03
Winfried Tautges
Der DESIGN-ANALYZER - Decision Support im Designprozess
75 Seiten

D-96-01
Klaus Fischer, Darius Schier
Ein Multiagentenansatz zum Lösen von Fleet-Scheduling-Problemen
72 Seiten

1995

D-95-12
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'95 Workshop:
KRDB-95 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
61 pages

D-95-11
Stephan Busemann, Iris Merget
Eine Untersuchung kommerzieller Terminverwaltungssoftware im Hinblick auf die Kopplung mit natürlichsprachlichen Systemen
32 Seiten

D-95-10
Volker Ehresmann
Integration ressourcen-orientierter Techniken in das wissensbasierte Konfigurierungssystem TOOCON
108 Seiten

D-95-09
Antonio Krüger
PROXIMA: Ein System zur Generierung graphischer Abstraktionen
120 Seiten

D-95-08
Technical Staff
DFKI Jahresbericht 1994
63 Seiten

Note: This document is no longer available in printed form.

D-95-07
Ottmar Lutzy
Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse
74 Seiten

D-95-06
Markus Steffens, Ansgar Bernardi
Integriertes Produktmodell für Behälter aus Faserverbundwerkstoffen
48 Seiten

D-95-05
Georg Schneider
Eine Werkbank zur Erzeugung von 3D-Illustrationen
157 Seiten

D-95-04
Victoria Hall
Integration von Sorten als ausgezeichnete taxonomische Prädikate in eine relational-funktionale Sprache
56 Seiten

D-95-03
Christoph Endres, Lars Klein, Markus Meyer
Implementierung und Erweiterung der Sprache $\mathcal{ALCP}$
110 Seiten

D-95-02
Andreas Butz
BETTY
Ein System zur Planung und Generierung informativer Animationssequenzen
95 Seiten

D-95-01
Susanne Biundo, Wolfgang Tank (Hrsg.)
PuK-95, Beiträge zum 9. Workshop „Planen und Konfigurieren", Februar 1995
169 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

1994

D-94-15
Stephan Oepen
German Nominal Syntax in HPSG
— On Syntactic Categories and Syntagmatic Relations —
80 pages

D-94-14
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 2: User Guide.
72 pages

D-94-12
Arthur Sehn, Serge Autexier (Hrsg.)
Proceedings des Studentenprogramms der 18. Deutschen Jahrestagung für Künstliche Intelligenz KI-94
69 Seiten

D-94-11
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
65 pages

Note: This document is no longer available in printed form.

D-94-10
F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (Eds.)
Working Notes of the 1994 International Workshop on Description Logics
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-94-09
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht 1993
145 Seiten

D-94-08
Harald Feibel
IGLOO 1.0 - Eine grafikunterstützte Beweisentwicklungsumgebung
58 Seiten

D-94-07
Claudia Wenzel, Rainer Hoch
Eine Übersicht über Information Retrieval (IR) und NLP-Verfahren zur Klassifikation von Texten
25 Seiten

Note: This document is no longer available in printed form.

D-94-06
Ulrich Buhrmann
Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien
117 Seiten

D-94-04
Franz Schmalhofer, Ludger van Elst
Entwicklung von Expertensystemen: Prototypen, Tiefenmodellierung und kooperative Wissensevolution
22 Seiten

D-94-03
Franz Schmalhofer
Maschinelles Lernen: Eine kognitionswissenschaftliche Betrachtung
54 Seiten

Note: This document is no longer available in printed form.

D-94-02
Markus Steffens
Wissenserhebung und Analyse zum Entwicklungsprozeß eines Druckbehälters aus Faserverbundstoff
90 pages

D-94-01
Josua Boon (Ed.)
DFKI-Publications: The First Four Years 1990 - 1993
75 pages

Proceedings of the Fifth Meeting on Mathematics of Language – MOL5

Tilman Becker and Hans-Ulrich Krieger (eds.)

D-97-02
Document