A Uniform Constraint-based Framework for the Verification of Infinite State Systems

DISSERTATION

zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes
von

Supratik Mukhopadhyay

Saarbrücken
2000

Contents

1 Introduction **1**
- 1.1 Perspective . . . 1
- 1.2 A Brief History of Computer-Aided Verification . . . 3
- 1.3 Synopsis of this Dissertation . . . 5
- 1.4 Contributions of this Dissertation . . . 6

2 Preliminaries **9**
- 2.1 Transition Systems . . . 9
  - 2.1.1 Equivalences between Transition Systems . . . 10
- 2.2 Specification Logics . . . 11
- 2.3 Logic Programs and Datalog . . . 12
- 2.4 OLDT Resolution . . . 13
  - 2.4.1 Extension of an OLDT structure . . . 14
- 2.5 Constraint Query Languages . . . 15
  - 2.5.1 Finite Automata and Constraint Query Languages . . . 17
  - 2.5.2 Pushdown Processes . . . 18
  - 2.5.3 Tree Automata . . . 18
  - 2.5.4 Alternating Automata . . . 19
  - 2.5.5 Automata on Infinite Words: Büchi Automata . . . 19
- 2.6 The Logic CTL . . . 20
  - 2.6.1 Model Checking for CTL: Programs with Oracles . . . 20
- 2.7 Computing Model-Theoretic Semantics . . . 21
  - 2.7.1 Dowling-Gallier Graphs . . . 22
  - 2.7.2 Immediate Consequence Operator . . . 23
  - 2.7.3 Magic Sets Transformation . . . 24
- 2.8 Constraint Domains . . . 24
  - 2.8.1 The Constraint Domain of Infinite Trees . . . 25
  - 2.8.2 Constraint Simplification . . . 28

3 Model Checking for Timed Logic Processes **29**
- 3.1 Introduction . . . 29
- 3.2 Timed Automata . . . 30
- 3.3 Timed Logic Processes . . . 33
- 3.4 Translation of Timed Automata into TLPs . . . 34
- 3.5 Logic of Safety and Bounded Liveness ($\mathcal{L}_s$) . . . 36

3.6 Product Program . . . 37
3.7 The Trim Operation on Constraints . . . 39
3.8 Extension of OLDT Resolution to Constraints . . . 49
3.9 Full Disjunction . . . 51
3.10 Unbounded Liveness Properties . . . 52
3.11 Implementation . . . 53
3.12 Model Checking for TCTL formulas . . . 57
3.13 Transient Behavior of Real Time Systems . . . 59
3.13.1 Detecting Transience . . . 60
3.13.2 Construction of Nonground Büchi Automata . . . 60
3.14 Related Work . . . 62

4 The Stratified μ-calculus **65**
4.1 Introduction . . . 65
4.2 Stratification . . . 66
4.3 Backward Analysis . . . 68
4.4 Perfect Models . . . 69
4.5 Tabled Resolution . . . 71
4.6 Convergence in Timed Automata . . . 76
4.7 Checking Convergence . . . 77

5 Beyond Region Graphs: Symbolic Forward Analysis of Timed Automata **81**
5.1 Introduction . . . 81
5.2 The Constraint Transformer $\varphi \mapsto [\![w]\!](\varphi)$ . . . 82
5.3 Zone Trees and Symbolic Forward Analysis . . . 85
5.4 RQ Automata . . . 87
5.5 Future Work . . . 94

6 Accurate Widenings and Boundedness Properties **97**
6.1 Introduction . . . 97
6.2 Intuition Behind the Accurate Widening Framework . . . 100
6.3 Timed Automata, Constraints and Model Checking . . . 102
6.3.1 Inclusion Abstractions . . . 103
6.4 Widening Rules . . . 104
6.5 Related Work . . . 113

7 Compositional Termination Analysis of Symbolic Forward Analysis for Infinite-State Systems **117**
7.1 Introduction . . . 117
7.2 Infinite State Systems . . . 118
7.3 Parallel Composition . . . 119
7.4 Constraints Representing Sets of States . . . 120
7.5 Bound Variables and Initialized Strings . . . 121
7.6 Constraint Trees and Symbolic Forward Analysis . . . 122
7.7 Compositional Reasoning about Termination . . . 125

7.8 Related Work 134

8 Constraint Transformer Monoids: A Unified Algebraic Framework for Abstract Symbolic Forward Analysis of Infinite State Systems 137
8.1 Introduction 137
8.2 Infinite State Systems 138
8.3 Constraint Transformer Monoids 138
8.4 Coverings of Constraint Transformer Monoids 139
8.5 Constraint Trees and Symbolic Forward Analysis 141
8.6 Abstract Constraint Trees and Abstract Symbolic Forward Analysis 142
8.7 Applications 144
8.7.1 Timed Automata 144
8.7.2 The Two-process Bakery Algorithm 145
8.8 Summary and Related Work 147

9 Conclusions 149
9.1 Summary 149
9.2 Future Work 149

Abstract

Automatic verification of infinite state systems is an important area of research. Unlike its finite state counterpart, in spite of the existence of a large body of theoretical and practical results on automatic verification of infinite state systems, there does not exist a uniform framework that is applicable to a large class of systems and that facilitates description of procedures that solves the verification problem for infinite state systems in practice as well as providing tools for reasoning about the termination conditions of such procedures. The purpose of this dissertation is to provide a uniform framework that (1) allows description of infinite state systems at their own level of granularity, (2) allows specifying their properties at a high level, (3) allows description of procedures, that can solve in practice the verification problems for infinite state systems, in a declarative fashion, (4) provides tools to reason about the termination conditions for such procedures, (5) facilitates derivation of abstractions for verification as well as easy incorporation of optimization techniques, (6) allows clear separation of the logical aspects of verification from the combinatorial ones, (7) allows combination of deductive (proof-theoretic) methods with model-theoretic ones and (8) provides, for free, data structures for implicit representation of possibly infinite sets of states.

Zusammenfassung

Automatische Verifikation von Systemen mit unendlichem Zustandsraum ist ein wichtiges Forschungsgebiet. Doch im Gegensatz zum Fall endlicher Zustandsräume und trotz einer grossen Anzahl an theoretischen und praktischen Resultaten über automatische Verifikation von Systemen mit unendlichem Zustandsraum, existiert kein einheitliches Rahmenwerk, das sich auf eine grosse Klasse von Systemen anwenden liesse und das die Beschreibung von Prozeduren, die das Verifikationsproblem von Systemen mit unendlichem Zustandsraum in der Praxis lösen, unterstützen würde, sowie das Werkzeuge zum Beweis der Termination solcher Prozeduren zur Verfügung stellen würde. Das Ziel dieser Dissertation ist es, ein einheitliches Rahmenwerk zu liefern, das (1) die Beschreibung von Systemen mit unendlichem Zustandsraum erlaubt, (2) die Spezifikation ihrer Eigenschaften auf einer hohen Ebene erlaubt, (3) die Beschreibung von Prozeduren, die das Verifikationsproblem fuer Systeme mit unendlichem Zustandsraum lösen, in einer deklarativen Art und Weise erlaubt, (4) Werkzeuge zum Beweis von Terminationsbedingungen solcher Prozeduren zur Verfügung stellt, (5) die Herleitung von Abstraktionen zur Verifikation ebenso wie die einfache Einbindung von Optimierungstechniken unterstützt, (6) eine klare Trennung der logischen Seiten der Verifikation von den kombinatorischen erlaubt, (7) eine Kombination deduktiver (beweistheoretischer) Methoden mit modelltheoretischen erlaubt und (8) umsonst Datenstrukturen für die implizite Darstellung von Systemen mit unendlichem Zustandsraum bereitstellt.

Extended Abstract

Automatic verification of infinite state systems is an important area of research. Unlike its finite state counterpart, in spite of the existence of a large body of theoretical and practical results on automatic verification of infinite state systems, there does not exist a uniform framework that is applicable to a large class of systems and that facilitates description of procedures that solves the verification problem for infinite state systems in practice as well as providing tools for reasoning about the termination conditions of such procedures. The purpose of this dissertation is to provide a uniform framework that (1) allows description of infinite state systems at their own level of granularity, (2) allows specifying their properties at a high level, (3) allows description of procedures, that can solve in practice the verification problems for infinite state systems, in a declarative fashion, (4) provides tools to reason about the termination conditions for such procedures, (5) facilitates derivation of abstractions for verification as well as easy incorporation of optimization techniques, (6) allows clear separation of the logical aspects of verification from the combinatorial ones, (7) allows combination of deductive (proof-theoretic) methods with model-theoretic ones and (8) provides, for free, data structures for implicit representation of possibly infinite sets of states.

The two main currents that run through this dissertation are constraints and logic. Using an intricate and exquisite interplay between constraints and logic, we provide a uniform constraint-based framework for the verification of infinite state systems. The key idea is that the verification problems for infinite state systems can be naturally viewed as constraint satisfaction problems. This idea leads to the observation that temporal properties of infinite state systems can be described as model-theoretic semantics of constraint databases. This connection allows description of the system as well as specifying properties about it at a high level in a declarative fashion.

The methods employed for computing (or checking membership in) model-theoretic semantics of constraint databases are deductive ones. Thus our methodology replaces the conventional graph-theoretic techniques for automatic verification of infinite state systems by uniform deductive ones. By employing specialized deduction strategies, optimized local and symbolic procedures for automatic verification are obtained in a natural manner. This way we unify, extend and explain in a uniform manner the seemingly different procedures behind the success of several existing verification tools for infinite state systems. Due to the closely knit interplay between logic and constraints, it has been possible for us to design practical procedures that can verify properties of infinite state systems specified in richer logics. As constraints are used within a logical landscape, it is easier to reason about the correctness of these procedures within our framework. Since constraints can represent possibly infinite sets of states, we obtain, for free, data structures for implicit representation of possibly infinite sets of states. Also since the algorithmic aspect of solving constraints is separated from the logical one, our framework

provides modular solutions to verification problems.

Theoretical investigations of infinite-state systems have so far concentrated on decidability results; using our framework, we investigate the specific procedures that are used in practice to decide verification problems. Our framework presents basic concepts and properties that are useful for reasoning about sufficient termination criteria for procedures solving the verification problem for infinite state systems in practice, and also for deriving those criteria. These criteria can be obtained in the form of syntactic sufficient conditions on the individual components composed with *asynchronous* parallel composition. The central notions here are constraint transformers associated with sequences of transitions of an infinite state system and *constraint trees* labeled with successor constraints. We show interesting examples of systems for which the sufficient termination conditions derived using our framework guarantee the termination of the procedures solving the verification problem for such systems in practice. We also provide a unified algebraic framework for deriving abstractions for the verification of a large class of infinite state systems and for reasoning about their accuracy. The central notions involved are those of *constraint transformer monoids* and *coverings* between constraint transformer monoids. Due to the choice of constraints as data structures representing possibly infinite sets of states, the abstractions, most of which are presented as widening rules, are easily implementable using constraint-based operations. We show interesting examples in which the abstractions derived using our framework force the termination of otherwise nonterminating verification procedures without losing any accuracy in the process.

Finally, to demonstrate the applicability of our framework, we show that many verification problems can be solved by a natural translation to our framework. This fact is corroborated by the encouraging results obtained by applying an implementation based on our framework to practical verification problems. We have also identified sufficiently expressive fragments of the propositional μ calculus such that our framework, that uses disjunctive constraints as the data structure for representing and manipulating sets of states, is especially suitable for verification problems in which the properties specified are in these fragments.

Ausführliche Zusammenfassung

Automatische Verifikation von Systemen mit unendlichem Zustandsraum ist ein wichtiges Forschungsgebiet. Doch im Gegensatz zum Fall endlicher Zustandsräume und trotz einer grossen Anzahl an theoretischen und praktischen Resultaten über automatische Verifikation von Systemen mit unendlichem Zustandsraum, existiert kein einheitliches Rahmenwerk, das sich auf eine grosse Klasse von Systemen anwenden liesse und das die Beschreibung von Prozeduren, die das Verifikationsproblem von Systemen mit unendlichem Zustandsraum in der Praxis lösen, unterstützen würde, sowie das Werkzeuge zum Beweis der Termination solcher Prozeduren zur Verfügung stellen würde. Das Ziel dieser Dissertation ist es, ein einheitliches Rahmenwerk zu liefern, das (1) die Beschreibung von Systemen mit unendlichem Zustandsraum erlaubt, (2) die Spezifikation ihrer Eigenschaften auf einer hohen Ebene erlaubt, (3) die Beschreibung von Prozeduren, die das Verifikationsproblem fuer Systeme mit unendlichem Zustandsraum lösen, in einer deklarativen Art und Weise erlaubt, (4) Werkzeuge zum Beweis von Terminationsbedingungen solcher Prozeduren zur Verfügung stellt, (5) die Herleitung von Abstraktionen zur Verifikation ebenso wie die einfache Einbindung von Optimierungstechniken unterstützt, (6) eine klare Trennung der logischen Seiten der Verifikation von den kombinatorischen erlaubt, (7) eine Kombination deduktiver (beweistheoretischer) Methoden mit modelltheoretischen erlaubt und (8) umsonst Datenstrukturen für die implizite Darstellung von Systemen mit unendlichem Zustandsraum bereitstellt.

Die beiden Leitmotive, die sich durch diese Dissertation ziehen, sind Constraints und Logik. Durch eine komplizierte und auserlesene Wechselwirkung zwischen Constraints und Logik liefern wir ein einheitliches constraint-basiertes Rahmenwerk zur Verifikation von Systemen mit unendlichem Zustandsraum. Dass die Verifikationsprobleme von Systemen mit unendlichem Zustandsraum auf natürliche Weise als Erfüllbarkeitsprobleme von Constraints gesehen werden können, ist der Schlüsselgedanke. Dieser Gedanke führt zu der Beobachtung, dass temporale Eigenschaften von Systemen mit unendlichem Zustandsraum als modelltheoretische Semantiken von Constraint-Datenbanken beschrieben werden können. Dieser Zusammenhang erlaubt die Beschreibung des Systems ebenso wie die Spezifikation seiner Eigenschaften auf einer hohen Ebene und in einer deklarativen Art und Weise.

Die Methoden zur Berechung (oder zum Test der Zugehörigkeit zu) modelltheoretischer Semantiken von Constraint-Datenbanken sind deduktive. Somit ersetzt unsere Methodik die herkömmlichen graphtheoretischen Verfahren zur automatischen Verifikation von Systemen mit unendlichem Zustandsraum durch einheitliche deduktive Methoden. Indem wir spezialisierte Deduktionsstrategien einsetzen, erhalten wir auf natürliche Weise optimierte lokale und symbolische Prozeduren zur automatischen Verifikation. Auf diese Weise vereinheitlichen, erweitern und erklären wir in einer einheitlichen Weise die scheinbar verschiedenen Prozeduren, die hinter dem Erfolg von mehreren existierenden Werkzeugen zur Verifikation von Systemen mit unendlichem

Zustandsraum stehen. Aufgrund der feinmaschigen Wechselwirkung zwischen Constraints und Logik konnten wir brauchbare Prozeduren entwerfen, die in reicheren Logiken spezifizierte Eigenschaften von Systemen mit unendlichem Zustandsraum verifizieren können. Da Constraints in einer logischen Landschaft benutzt werden, ist es leichter, die Korrektheit dieser Prozeduren in unserem Rahmenwerk zu beweisen. Weil Constraints potentiell unendliche Zustandsmengen repräsentieren können, erhalten wir umsonst Datenstrukturen zur impliziten Repräsentation potentiell unendlicher Zustandsmengen. Desweiteren bietet unser Rahmenwerk modulare Lösungen für Verifikationsprobleme, da der algorithmische Aspekt des Lösens von Constraints abgetrennt ist vom logischen. Theoretische Forschung über Systeme mit unendlichem Zustandsraum war bislang auf Entscheidbarkeitsresultate gerichtet; mit unserem Rahmenwerk untersuchen wir nun die spezifischen Prozeduren, die in der Praxis angewandt werden, um Verifikationsprobleme zu entscheiden. Unser Rahmenwerk stellt grundlegende Konzepte und Eigenschaften vor, die nützlich sind zur Herleitung und zum Beweis von hinreichenden Terminationskriterien für Prozeduren, die das Verifikationsproblem in der Praxis lösen. Diese Kriterien erhält man in Form syntaktischer hinreichender Bedingungen an die einzelnen Komponenten. Die zentralen Ideen hier sind Constraint-Umformer, die Folgen von Übergängen eines Systems mit unendlichem Zustandsraum zugeordnet sind, und *Constraint-Bäume*, deren Knoten mit Nachfolger-Constraints markiert sind. Wir zeigen interessante Beispiele, wo die mit unserem Rahmenwerk hergeleiteten hinreichenden Terminationsbedingungen die Termination von Prozeduren, die das Verifikationsproblem für solche System in der Praxis lösen, garantieren. Wir bieten auch ein einheitliches algebraisches Rahmenwerk, um Abstraktionen für eine grosse Klasse von Systemen mit unendlichem Zustandsraum herzuleiten und um Beweise über die Exaktheit dieser Abstraktionen zu führen. Hier sind die beteiligten zentralen Ideen *Monoide von Constraint-Umformern* und *Überdeckungen* zwischen Constraint-Umformern. Aufgrund der Wahl von Constraints als Datenstruktur zur Darstellung potentiell unendlicher Zustandsmengen, können die Abstraktionen, die meist als Widening-Regeln dargestellt werden, leicht durch constraint-basierte Operationen implementiert werden. Wir zeigen interessante Beispiele, wo die mit unserem Rahmenwerk hergeleiteten Abstraktionen die Termination andernfalls nicht-terminierender Verifikationsprozeduren erzwingen, ohne an Präzision zu verlieren.

Schliesslich zeigen wir, dass viele Verifikationsprobleme durch eine natürliche Übersetzung in unser Rahmenwerk gelöst werden können. Diese Tatsache wird untermauert durch die ermutigenden Ergebnisse, die eine auf unserem Rahmenwerk basierende Implementierung an praktischen Verifikationsproblemen lieferte. Auch haben wir hinreichend ausdrucksstarke Fragmente des aussagenlogischen μ-Kalküls bestimmt, so dass Prozeduren zur Verifikation von Eigenschaften, die in diesen Fragmenten spezifiziert sind, besonders geeignet sind für unser Rahmenwerk, das disjunktive Constraints als Datenstrukturen zur Repräsentation und Manipulation von Zustandsmengen benutzt.

Acknowledgements

The order in which different people are acknowledged in this acknowledgement does not bear any significance at all. First I start by acknowledging my advisor Andreas, without whom this dissertation would not have been possible. During the past three years, I have learnt a lot from him. He has taught me how to do research— but wait a minute — the most important thing that he has taught me is how to maintain a standard in research which I think is more important just learning to do research — it is solely my fault that I could not follow his teachings. He has always emphasized the need for good writing and clear presentation — again it is my fault that I could not pick it up (if only I could follow 0.5 percent of his advice and the examples of clear presentation that he had set before me, this dissertation would have been much better). There is only one word in the dictionary that can describe Andreas — "perfectionist" (which I hope I would become eventually though I don't know when). Any paper (possibly including this thesis) will undergo thirty revisions before getting submitted (the last revision on the table in front of the kitchen; I wonder how he does not get bored). Anyway, I hope to emulate him in future. He has allowed me to work on varied topics while taking care that I don't get drowned in the sea. The most important privilege that I think I enjoyed was that I could go to his office at any time (even without knocking his door) and could discuss with him even "half-baked" ideas. Under his direction, during the last three years, my progress was so smooth that I could start writing this dissertation even before two and half years had passed.

Next come Witold, Jean-Marc and Andreas Nonnengart. If I were to write in this acknowledgement how much I owe to these three people, probably this acknowlewdgement will become larger than the rest of this dissertation (the rest of this dissertation is 100+ pages!). All I can say in one sentence is that this dissertation simply would not have existed in the absence of any one of these three people. I thank God for bringing these three people together at Saarbrücken at the same time as me. All these three people are incredibly smart. I am really surprised by their problem solving abilities.It is really surprising how quickly they understand any new concept and then go into the heart of the matter. I would advise anybody who has a new idea to get it bounced off these three people. Any gap, if present in the idea, is guaranteed to get detected. You tell them your idea. Then you suddenly hear "Can I ask a stupid question?"; and this "stupid question" finds out a bug!

I am indebted to Harald, Viorica and the rest of the group members for encouragement. One has to see to believe how Harald has mastery of so many fields at the same time. I thank Patrick for all the nice translations and for dealing with machine related problems with a smile. I thank Georg for helping me with implementation related matters. I thank Manfred for helping me with TeX related matters.

I thank Brigitta and Christine for dealing with all my bureaucratic problems. I thank the computer support group for their hard work in keeping the system running.

Thanks to C.R. Ramakrishnan for agreeing to be my thesis reviewer. I thank Tom Henzinger for that excellent course in Computer-Aided Verification, arranging for open-problem seminars and for answering my questions promptly. Thanks to Jean-Francois Raskin for discussions. I thank Andy Gordon for arranging for my visit to Microsoft Research and for collaborating with me. I thank Luca Cardelli and Silvano Dal-Zilio for discussions, Moshe Vardi and Rajeev Alur for providing prompt answers to my questions. Also, Moshe's crash course on automata-theoretic verification was a beautiful experience. I thank our librarian Anja Becker for the patience she showed when I used to bring half the library to my office.

I thank C.R. Subramanian and Anil Kumar for keeping me sane through the days of writing this dissertation. Subramanian is, in particular, a great source of ideas in discrete mathematics. I thank my undergraduate and graduate teachers for sharing their enthusiasm for research with me.

Finally, I thank my parents for being role models to emulate. And God for making possible for this dissertation to be written. Lastly, I apologize to anybody that I have failed to acknowledge.

List of Figures

2.1 Illustrating the mapping f 26
2.2 Illustrating the mapping g 27

3.1 An Example Timed Automaton. 33
3.2 Example illustrating that the model checking procedure is possibly non-terminating. 38
3.3 Product Program corresponding to Figure 3.2. 38
3.4 Normalization of Constraints. 43
3.5 Illustrating the trim operation — A. 44
3.6 Illustrating the trim operation — B. 45
3.7 Illustrating the trim operation — C. 45
3.8 Illustrating the trim operation — D. 45
3.9 Experimental Results. 51
3.10 Illustrating the Greatest Model Resolution. 54
3.11 Greatest Model Resolution (GMR) Procedure for programs with one body predicate. 56
3.12 Non-terminating Example for Greatest Model Resolution. 56
3.13 Illustrating transience. 60

4.1 Situating the expressiveness of Sμ 67
4.2 Computation tree for Example 4.2 72
4.3 Tabled Resolution 78
4.4 Timed automata 80

5.1 Example of a timed automaton for which the breadth-first version of symbolic forward analysis terminates but the depth-first version does not, if the edge numbered 4 is followed before the edge numbered 7. 86
5.2 Gate Automaton 88
5.3 Controller Automaton 88
5.4 Train Automaton 88
5.5 Train || Gate || Controller 89
5.6 Example of a timed automaton showing that the property: "Every reachable location is reachable through a simple path" does *not* entail termination of depth-first symbolic forward analysis. 90

6.1 Illustrating Unboundedness (Boundedness) Property 98
6.2 Fragment of the pseudo-code of a program 100

6.3 Template for Model Checking for Boundedness Properties 101
6.4 Template for Model Checking for Boundedness Properties with Widening 101
6.5 Widen Function 101
6.6 Illustrating Accelarating Effect of Widening Rules 104
6.7 Local and Global Inclusion Abstraction 104
6.8 Widening Rule I 105
6.9 Widening Rule II 106
6.10 Illustrating widening rule I 107
6.11 Illustrating widening rule II 108
6.12 Widening Rule III 109
6.13 Illustrating the widening rule III 109
6.14 Experimental Results 114

7.1 Example showing composition of o-minimal hybrid systems. 125
7.2 A two-process timed mutual exclusion protocol. 127

8.1 The bakery algorithm 146
8.2 Constraints in Φ' 146

Chapter 1

Introduction

Mathematical logic can provide a uniform framework for modeling and formally verifying reactive systems. In this dissertation, we make an attempt to justify the above thesis by trying to use the power of mathematical logic in providing a uniform constraint-based framework for modeling, understanding and reasoning about reactive systems.

1.1 Perspective

Rapid progress in computer technology in the last few decades has effected a significant change in the perspectives under which computing is viewed. Computers have now become truly "global" in the sense that they make their presence felt in devices ranging from such miniature ones as mini-cameras to such monstrous ones as airplanes. This "globalization" of computer technology has been accompanied by the development of large software systems and highly integrated hardware systems with their inherent logical complexity and many layers of abstraction. This logical complexity manifests itself, in particular, in embedded systems — systems that are embedded into a natural environment that is governed by physical laws, with agents (or applications) from different problem domains interacting with each other in often unpredictable ways. This logical complexity and unpredictability renders the use of such systems in safety-critical devices like airplanes or nuclear power-plant controllers into a high-risk affair. This risk factor is justified by the number of chilling experiences that we have had in the last few decades caused by the failure of computer systems operating in life critical applications. In order to minimize this risk factor and to avert such chilling experiences, we need to structure, reason about and develop such systems more systematically. It is exactly here that methods from mathematical logic come to our aid.

A distinct feature of these "new generation" computer systems mentioned above is their power of maintaining an ongoing interaction with the environment. While for a classical sequential program, non-termination entails the presence of a possible bug in the program, for an embedded system, termination usually indicates presence of a possible "deadlock". Such embedded systems are usually called *reactive systems* as their evolution involves "reaction" in response to "stimulus" or requests from the environment.

A long promoted way of using mathematical logic for designing provably correct hardware and software is the deductive or the theorem proving approach. Here one develops a formal proof of correctness of the system along with the system itself. The proof is usually based on *invariant*

assertions — logical formulae whose truth value never changes during the possible runs of the system. The correctness of the system is expressed as a logical consequence of an invariant that is true initially. Such an approach has the drawbacks that it is not fully automatic — it requires significant assistance from a human expert. Furthermore, it does not support reusability — even if a system has been proved correct for certain properties, for proving the system correct for other properties one may need to start from scratch.

An alternative approach is automatic verification. In this approach, commonly known as model checking [CE80], one verifies whether a "model" of the system (usually a Kripke structure) satisfies a specification (given by, say, a temporal logic formula [Pnu77]) by an exhaustive search through the "state-space" of the model. More precisely, a finite state (or an infinite state) reactive system is modeled as a Kripke structure (or as a parallel composition of several Kripke structures) and the property against which it is verified is specified as a temporal logic formula; given a Kripke structure K and a temporal logic formula φ, the model checking problem is to determine whether $K \models \varphi$ i.e., whether K is a model of φ (or to compute the set of states of K that satisfy φ). This problem is solved by an exhaustive search through the state space of K (or by a reachability analysis).

While the framework of automatic verification provides a solution to the verification problem, there is an inherent non-uniformity within it. First, there is no uniform way to model systems at their own level of granularity (e.g., pushdown systems [Wal96], hierarchical systems [AY98] are modeled differently in this framework). Second, the treatment of finite state and infinite state instances of the problem within this framework are different. Third, systems over different data domains (e.g., systems operating over numeric domains like reals or integers and systems over non-numeric domains such as queues or stacks) are treated differently. Fourth, the verification problem for different specifications use seemingly different techniques (viz, reachability analysis or computing strongly connected components of a directed graph).

Not only this non-uniformity is conceptually disturbing and makes integration difficult, but the lack of uniformity prevents immediate extensions to more expressive logics (see Chapter 3 for more details) or to more succinct (or different) system models (e.g., to hierarchical systems) without changing the whole set up. Moreover, dealing with "many-sorted" systems (e.g., systems in which some variables range over numeric domains while some others range over non-numeric domains) within this framework is difficult. Further, incorporation of optimization techniques for model checking becomes difficult within this framework. Also, within the above framework, it is difficult to identify subclasses of systems that can be model checked effectively and efficiently (this also holds true for specifications). The inherent non-uniformity also presents problems in separating the logical part of the framework from its combinatorial part thereby losing the modularity of the solution provided (e.g., in model checking for systems over numeric data types, the constraint solving part is not separated from the core model checking procedure).

Mathematical logic provides a uniform framework in which to simulate (and model) such reactive systems in an essentially coding-free way as well as write down and verify properties about the behaviors of such systems. The simulation is not supposed to be performed at a lower abstraction level; it should be done on the natural abstraction level of the system. The simulation can itself be viewed as a "database" at an appropriate abstraction level and the logical formula encoding properties about the behaviors as a query so that the verification problem boils down to that of evaluating that query on the "simulation" as a database. To give the reader a taste of the uniform framework that mathematical logic provides for specifying and verifying reactive systems, we provide a few examples below.

Reachability Analysis as a Mapping between Different Abstraction Levels A (possibly infinite state) transition system (or program) $\mathcal{P}$ with n data variables $x_1, \ldots, x_n$ ranging over a domain $\mathcal{D}$ and locations $\ell_1, \ldots \ell_k$ induces a relational structure $\mathcal{T}$ over the vocabulary $\tau = \langle \ell_1, \ldots, \ell_k \rangle$ (n-ary relation symbols) in the following way

$$\mathcal{T} = \langle \mathcal{D}, \ell_1, \ldots \ell_k \rangle$$

where the relation symbols $\ell_1, \ldots, \ell_k$ are interpreted as follows: $\mathbf{v} \in \ell_i$ if and only if the location ℓ_i is reachable in $\mathcal{P}$, from the initial state, with the data variables taking the value $\mathbf{v}$. We call this relational structure $\mathcal{T}$ induced by a transition system T as the *explicit* structure induced by T. Reachability analysis for a transition system then amounts to computing the explicit structure from the "implicit representation" of the transition system. The implicit representation of a transition system as well as the explicit structure induced by it can be viewed as the same database represented at different levels of abstraction. Thus reachability analysis can be viewed as a mapping between two different levels of abstraction of the same database.

Model Checking as Constraint Satisfaction A constraint satisfaction problem [FV98] is given by a pair I (called instance) and T (called template) of relational structures over the same vocabulary (we consider here the version with fixed template). The problem is satisfied if there is a homomorphism from I to T. A model checking problem for temporal logic is given by a transition system $\mathcal{P}$ and a temporal logic (say an LTL [Pnu77]) formula φ. This problem has a 'yes' answer if $\mathcal{P} \models \varphi$. One can reduce this problem to a language inclusion problem $\mathcal{L}(A_\mathcal{P}) \subseteq \mathcal{L}(A_\varphi)$ [VW86a] i.e., checking whether all computations accepted by the automaton $A_\mathcal{P}$ corresponding to $\mathcal{P}$ are also accepted by the automaton A_φ corresponding to φ. The answer to this latter problem is 'yes' if there is a homomorphism from $A_\mathcal{P}$ to A_φ (both viewed as relational structures). Thus a 'yes' answer to the 'constraint satisfaction' problem with $A_\mathcal{P}$ as instance and A_φ as template yields a 'yes' answer to the model checking problem.

1.2 A Brief History of Computer-Aided Verification

This section makes a brief review of the research on computer-aided-verification over the last 30 years as well as the current state-of-the-art, thus placing the research described in this thesis in context. Computer-Aided-Verification started with the seminal papers of Floyd [Flo67] and Hoare [Hoa69] (though the first researcher to advocate the use of computers for verifying software was Turing himself). Floyd and Hoare provided a framework for structured, compositional deductive verification of sequential programs. Their method was extended to parallel programs by Owicki and Gries [OG76]. But as we have mentioned earlier, such methods need considerable amount of intervention from a human expert. Hence, although mathematically appealing, these methods were not so successful in practice.

In 1977, in a seminal paper, Pnueli [Pnu77] proposed temporal logic for the specification of concurrent systems. In a temporal logic, we augment a conventional logic with temporal modalities making it possible to describe the ordering of events in time. As opposed to the Floyd-Hoare framework, where the specification can only relate the initial state and final state of a system, temporal logic is well suited to describe the on-going behavior of non-terminating reactive systems.

Model checking techniques for branching time temporal logic specifications were introduced in the early 80's by Clarke and Emerson [CE80] and independently by Quielle and Sifakis [QS81]. The late '80's and the early '90's have seen a blooming period for theoretical and practical research in model checking for finite state systems. Symbolic [BCM+92] and local [SW91] model checking methods were proposed to deal with the state explosion problem, more expressive logics like the propositional mu-calculus [Koz83] were being model checked, and systems with 10^{100} states were being handled [BCM+92]. On another side, automata-theoretic methods [VW86a] were proposed to unify the various approaches to model checking that had come up so far.

With techniques like symbolic model checking providing ways of representing and manipulating (possibly infinite) sets of states, researchers started considering model checking for infinite state systems (this area also got a lot of impetus from research on Petri nets). Bradfield and Stirling [BS90] considered local model checking for systems with infinite state spaces against mu-calculus specifications. While they gave a semi-algorithm for a general class of infinite state systems, several other researchers started looking for subclasses of infinite state systems that can be model checked effectively (if not efficiently). A breakthrough in this direction was accomplished when Alur and Dill in a seminal paper [AD94] isolated the class of timed automata (finite state systems augmented with clocks that range over the non-negative reals) that admit finite bisimulation. This result led to extensive research by several other researchers who extended techniques from finite state model checking like symbolic model checking [HNSY94], local model checking [SS95] to model checking for timed systems. Buoyed by the success of model checking for timed systems, researchers started looking at more expressive models like hybrid systems [ACHH93] where semi-algorithms for symbolic model checking were obtained. In addition, subclasses of hybrid systems like initialized rectangular automata [HKPV95], o-minimal hybrid systems [LPY99], that admit finite bisimulation were identified. In another direction, several subclasses of infinite state systems in which the variables range over non-numeric domains like the pushdown processes [Wal96, BEM97] etc., were identified, for which the reachability problem turned out to be decidable.

The middle and the end of the 1990's saw the emergence of model checkers capable of model checking for industrial size systems like the Philips audio control protocol [BLL+96]. Many hardware design companies adopted model checking as part of their basic design method. To deal with the increasing complexity in the functionality of the industrial systems being considered, on one hand more succinct models like hierarchical finite state machines [AY98] came up, while on the other hand techniques like model measuring [ATEP99] were introduced to deal with more 'precise' specifications.

This period also saw several efforts to unify the various techniques available for model checking under a uniform framework. Automata-theory was proposed as a vehicle of unification [DW99, BVW94]. However, the solution provided by automata theory, though close to logic, was not entirely satisfactory — many of the non-uniformities already crept into this framework (e.g., the automata-theoretic method does not work very well for model checking timed systems; in fact entirely new models like timed alternating tree automata [DW99] needed to be introduced to deal with timed systems).

1.3 Synopsis of this Dissertation

This dissertation makes an attempt to develop a unified framework based on mathematical logic for modeling and verifying (possibly infinite state) reactive systems. The central idea is to identify a constraint-based logical formalism that can provide a uniform representation for a large class of reactive systems using logical formulae and to reduce the verification problem to computing model-theoretic semantics of logical formulae. Computing model-theoretic semantics of logical formulae is closely related to query evaluation. Hence, developing optimized algorithms for query evaluation (or query optimization techniques) yields, as a by-product, optimized algorithms for verification.

The logical formalism, developed in this work, is able to model systems at their natural abstraction level (e.g., the formalism does not need any significant extension to model a succinctly represented system). Moreover, systems represented in conventional formalisms (e.g., pushdown systems etc.) are easily translatable to this formalism (bringing more flexibility to our framework).

The logical formulae representing a system may be viewed as a database (allowing possible recursion in the database) at an appropriate abstraction level. The interpretation of the "extensional database" predicates is provided by the specification. Then model checking (or reachability analysis) amounts to computing the interpretations of "intensional database" predicates from those of the extensional database predicates, i.e., evaluating a query on the database where the interpretations of the intensional predicates are the output relations. Thus the graph-theoretic framework (i.e., reachability analysis or computing strongly connected components of a graph) in conventional verification algorithms is replaced by a model-theoretic framework in our approach.

We use the framework mentioned in the previous paragraph to treat uniformly the problems of modeling and verification of (infinite state) systems with numeric data types (like real time systems, systems with integer-valued variables). Our framework allows the logical part of the problem to be clearly separated from the combinatorial (constraint solving) part. In this way, we explain uniformly and unify the seemingly different algorithms behind the success of several existing model checking tools. Further the uniform framework provides a platform in which to identify subclasses of verification problems for which termination guarantees exist for semi-algorithms that are used to solve the model checking problem in practice as well as to develop techniques for forcing convergence of semi-algorithms (possibly losing accuracy in the process) for undecidable verification problems. We use this platform to develop a 'toolbox' consisting of basic concepts and properties that are useful for reasoning about sufficient termination conditions for symbolic model checking semi-algorithms as well as deriving abstractions to either to force termination or to accelerate the convergence of such (semi) algorithms and reason about the accuracy of such abstractions.

In stark contrast with the automata-theoretic framework which is not easily extendible for dealing with infinite state systems like timed or hybrid systems our framework can, without any extension, uniformly deal with both finite and infinite state versions of the model checking problem.

1.4 Contributions of this Dissertation

In this section, we break up the contributions of this dissertation according to chapters[1]. The dissertation is so arranged that most of the chapters can be read independently of the rest of the dissertation. Each chapter starts with an Introduction and contains comparisons with related work that places the research described in that chapter in context.

In Chapter 2, we argue that the framework of constraint query languages can provide a uniform platform for modeling and verifying reactive systems. To this end, we show how finite state systems and pushdown systems can be uniformly captured by constraint query language programs (propositional horn formulae for finite state systems and Herbrand domain for pushdown systems) and their verification problem reduces to computing model theoretic semantics of constraint query language programs (horn formulae). These results are inspired by [CP98a, SIR96] and can be viewed as extending and unifying their work. We use Dowling-Gallier graphs [DG84] as advocated by [SIR96] for computing the model theoretic semantics of (propositional) horn formulae. We show how the pebbling algorithm of Dowling and Gallier [DG84] can be modified to deal with the problem of computing the greatest model semantics of propositional horn formulae.

The uniform framework of constraint query languages mentioned in the previous paragraph encompasses the automata-theoretic framework of Bernholtz, Vardi and Wolper [BVW94]. We show how both word and tree automata (non-deterministic, deterministic or alternating) can be captured uniformly by our framework and connect the emptiness problem for these automata to computing model-theoretic semantics of constraint query language programs. This connection allows us to capture the automata-theoretic model checking methodology of [BVW94] uniformly within our framework. Since the automata-theoretic framework already unifies the various approach to finite-state model checking [BVW94] (where the system is specified as a finite Kripke structure), capturing the automata-theoretic framework already provides some evidence of the uniformity of our framework. This part of the work is inspired by [CMN+98]. Finally, we prove some topological properties of the constraint domain of infinite trees. All these results along with some preliminaries constitute Chapter 2. This chapter contains results some of which belong to the author while others belong to the existing literature. The results that do not belong to the author are clearly distinguished by their citations.

In Chapter 3, we show how the uniform framework that we have identified can deal with the problem of specifying and verifying timed systems[2]. As a part of our uniform framework, we introduce a fragment of constraint query languages over reals and show that programs in this fragment can model timed systems. We call the programs expressed in this fragment as *timed logic processes* (TLPs). We establish a formal connection of TLPs with the standard model of timed automata. We use this connection to show that the Uppaal model checking procedure for safety and bounded-liveness properties of timed systems is the top-down query evaluation with tabling (in the XSB style) for TLPs. This allows us to obtain an alternative way of implementing Uppaal's procedure and for extending it. This extension accommodates properties with 'full' disjunction and unbounded liveness properties. All the results in Chapter 3 were obtained by the author.

[1]Chapter 2 besides providing some preliminary concepts needed for reading this dissertation, leads the reader closer to the uniform framework to be used in the later chapters

[2]Note that the automata-theoretic framework does not have an easy extension for dealing with real time systems

In Chapter 4, we introduce the stratified μ-calculus. Some symbolic model checking procedures use disjunctive constraints (e.g. disjunctions of conjunctions of arithmetic inequalities) to represent sets of states. This motivates us to introduce a new class of temporal properties with a backward analysis and a forward analysis that are both well-suited for disjunctive constraints as the 'symbolic' data structure. The *stratified* μ-calculus Sμ is a natural generalization of STL (Safe Temporal Logic) and can be used to express e.g. convergence for timed automata. Our technical contribution is the novel 'symbolic forward analysis' method for checking Sμ formulas. This method is based on our characterization of Sμ properties as *perfect models* of constraint logic programs and on our *tabled-resolution* procedure for constraint logic programs with the perfect-model semantics.

In Chapter 5, we are concerned with the termination of the procedures that solve the model checking problem for timed systems in practice. Theoretical investigations of infinite-state systems have so far concentrated on decidability results; in the case of timed automata these results are based on region graphs. We investigate the specific procedure that is used practically in order to decide verification problems, namely symbolic forward analysis. This procedure is possibly non-terminating. We present basic concepts and properties that are useful for reasoning about sufficient termination conditions, and then derive some conditions. The central notions here are constraint transformers associated with sequences of automaton edges and *zone trees* labeled with successor constraints.

In Chapter 6, we propose a symbolic model checking procedure for timed systems that is based on operations on constraints. To accelerate the termination of the model checking procedure, we define history-dependent widening operators, again in terms of constraint-based operations. We show that these widenings are accurate, i.e., they don't lose precision even with respect to the test of boundedness properties.

In Chapter 7, we consider compositional termination analysis of symbolic forward analysis for infinite state systems. Existing model checking tools for infinite state systems, such as UPPAAL, HYTECH and KRONOS, use symbolic forward analysis, a possibly nonterminating procedure. We show termination for the special case of *o-minimal* hybrid systems. We give termination criteria for general integer-valued systems and nonlinear hybrid systems. These criteria are in the form of syntactic sufficient conditions on the individual components composed with *asynchronous* parallel composition.

In Chapter 8, we present a constraint-based framework for deriving abstract symbolic model checking procedures and also for reasoning about their accuracy. Symbolic forward analysis is a semi-algorithm that in many cases solves the model checking problem for infinite state systems in practice. This semi-algorithm is implemented in many practical model checking tools like UPPAAL [BLL+96], KRONOS [DT98] and HYTECH [HHWT97]. In most practical experiments, termination of symbolic forward analysis is achieved by employing abstractions resulting in an abstract symbolic forward analysis. This paper presents a unified algebraic framework for deriving abstract symbolic forward analysis procedures for a large class of infinite state systems with variables ranging over a numeric domain. Our framework provides sufficient conditions under which the derived abstract symbolic forward analysis procedure is always terminating or accurate or both. The class of infinite state systems that we consider here are (possibly non-linear) hybrid systems and (possibly non-linear) integer-valued systems. The central notions involved are those of *constraint transformer monoids* and *coverings* between constraint transformer monoids. We show concrete applications of our framework in deriving abstract symbolic forward analysis algorithms for timed automata and the two process bakery algorithm that are

both terminating and accurate.

Chapter 9 concludes the dissertation. In this chapter, we briefly summarize the subject matter of the thesis and also present problems left open in the dissertation and directions for future research.

Chapter 2

Preliminaries

We first present some preliminary notions. We then argue that constraint query languages can provide a uniform constraint-based framework for modeling and verifying (possibly infinite state) reactive systems. This is demonstrated by showing that a large number of seemingly unrelated formalisms have a natural translation to the framework of constraint query languages and their verification problems reduce to computing the model-theoretic semantics of constraint query language programs.

2.1 Transition Systems

We are interested in the formal verification of reactive systems. Labeled transition systems are a formalism for describing such systems.

Definition 2.1 (Labeled Transition System) *A labeled transition system is a six tuple*

$$\mathcal{L} = \langle S, \Sigma, S_0, \longrightarrow, AP, P \rangle,$$

where S is a set of states, Σ is a finite alphabet (or a set of letters), $S_0 \subseteq S$ is a set of initial states, $\longrightarrow \subseteq S \times \Sigma \times S$ is a transition relation, AP is a finite set of atomic propositions and $P : S \longrightarrow 2^{AP}$ assigns to each state a set of atomic propositions.

We call a labeled transition system in which $|\Sigma| = 1$ a *one-letter* transition system [BVW94] or simply an *unlabeled* transition system (or a Kripke structure). In case of an unlabeled transition system, we can assume the transition relation $\longrightarrow$ to be a binary relation; $\longrightarrow \subseteq S \times S$. A transition system (labeled or unlabeled) is infinite if S is infinite. In this dissertation, we are concerned with possibly infinite state transition systems that can be *finitely represented* (explained below). Most of the transition systems that we consider in this dissertation are unlabeled. For $s \in S$, a path $\pi = s_0, s_1, \ldots$ starting from s is an infinite sequence of states such that $s_0 = s$ and for all $i \geq 0$, there exists $a \in \Sigma$ such that $\langle s_i, a, s_j \rangle \in \longrightarrow$. For a path $\pi = s_0, s_1, \ldots$, we will write $\pi[i]$ for s_i.

For a set of atomic propositions AP, let $\mathcal{V} = \langle \longrightarrow, \{p \mid p \in AP\}, \Sigma, S_0 \rangle$ be a two-sorted vocabulary with $\{1, 2\}$ as the set of sorts, where the $\longrightarrow$ is a ternary relation symbol with sort $\langle 1, 2, 1 \rangle$ and all other relation symbols are monadic with sort $\langle 1 \rangle$ with the exception of Σ which is a monadic relation symbol of sort $\langle 2 \rangle$. Let L be a two-sorted first order language (with equality) such that $L \cap \mathcal{V} = \emptyset$. A labeled transition system $\mathcal{L}$ over L can be viewed as an expansion of

an L structure $\mathcal{A}$ with universe A to $\mathcal{V}$ such that the interpretation of Σ is a finite relation. The language L is called the underlying language of $\mathcal{L}$. A labeled transition system $\mathcal{L}$ is *finitely representable* over L if for each relation $R \in \mathcal{V} \setminus \{\Sigma\}$ there exists a quantifier-free L formula $\varphi(\mathbf{x})$ such that

$$\mathcal{L} \models \forall \mathbf{x}(R(\mathbf{x}) \longleftrightarrow \varphi(\mathbf{x}))$$

For example, the transition system $\mathcal{L}_{\mathcal{P}} = \langle \mathcal{N}, \{a\}, \longrightarrow, AP, P \rangle$ with the set of natural numbers $\mathcal{N}$ being the set of states and the relation $\longrightarrow$ defined as $\longrightarrow (i, a, j)$ iff j is even is not finitely representable over the language of Presburger arithmetic. The reason is that Presburger arithmetic does not admit quantifier elimination [End72] (e.g., there is no quantifier free Presburger arithmetic formula equivalent to the formula $\exists x\, y = x + x$ defining the set of even natural numbers). In the sequel, we write $s \xrightarrow{a} s'$ to denote $\langle s, a, s' \rangle \in \longrightarrow$.

2.1.1 Equivalences between Transition Systems

In this section, we briefly review some notions for comparing two transitions. The notions of simulation and bisimulation [Mil89] are the basic ways of comparing the structure of transition systems.

Definition 2.2 (Bisimulation [Mil89]) *Given labeled transition systems $\mathcal{L} = \langle S, \Sigma, S_0, \longrightarrow, AP, P \rangle$ and $\mathcal{L}' = \langle S', \Sigma, S'_0, \longrightarrow', AP', P' \rangle$, a binary relation $\approx \subseteq S \times S'$ is a bisimulation relation if for each $s_0 \in S_0$, there exists $s'_0 \in S'_0$ such that $s_0 \approx s'_0$ and vice versa and for all letters α, $s \approx t$ implies:*

- *Whenever $s \xrightarrow{\alpha} s'$ then, for some t', $t \xrightarrow{\alpha}{}' t'$ and $s' \approx t'$.*
- *Whenever $t \xrightarrow{\alpha}{}' t'$ then, for some s', $s \xrightarrow{\alpha} s'$ and $s' \approx t'$.*

A bisimilarity relation $\approx$ is a bisimulation between $\mathcal{L}$ and $\mathcal{L}'$, such that for all states $s \in S$, there is a state $t \in S'$ such that $s \approx t$ and for all states $t \in S'$ there exists $s \in S$ such that $s \approx t$. We say that $\mathcal{L}$ and $\mathcal{L}'$ are bisimilar iff a bisimilarity relation $\approx$ exists between $\mathcal{L}$ and $\mathcal{L}''$.

Definition 2.3 (Quotient Transition System) *Let $\mathcal{L} = \langle S, \Sigma, S_0, \longrightarrow, AP, P \rangle$ be a (labeled) transition system. Let $\sim$ be an equivalence relation on S that does not distinguish elements of S_0. The quotient transition system $\mathcal{L}/\sim$ is defined as follows: For all letters $\alpha \in \Sigma$*

$$\mathcal{L}/\sim = \langle S/\sim, \Sigma, [S_0], \longrightarrow^{\sim}, AP, P \rangle$$

where $S/\sim$ is the set of equivalence classes of S induced by the equivalence relation $\sim$, $[S_0]$ is the equivalence class containing S_0. The transition relation is defined as follows: $\mathcal{E}_1 \xrightarrow{\alpha}{}^{\sim} \mathcal{E}_2$, for two equivalence classes $\mathcal{E}_1$ and $\mathcal{E}_2$, if there exists $s_1 \in \mathcal{E}_1$ and $s_2 \in \mathcal{E}_2$ such that $s_1 \xrightarrow{\alpha} s_2$.

Note that if the equivalence relation $\sim$ is a bisimulation then $\mathcal{L}/\sim$ and $\mathcal{L}$ are bisimilar. The notion of quotient transition systems and bisimilarity will be used in Chapter 3

In the remaining part of this section, we review two types of state space partitions of a transition system.

Definition 2.4 (Pre-stable and Post-stable Partitions [ACD+92]) *Let* $\mathcal{L} = \langle S, \Sigma, S_0, \longrightarrow, AP, P \rangle$ *be a labeled transition system. Let* $\sim$ *be an equivalence relation on* S*. The partitioning of* S *induced by* $\sim$ *is pre-stable if for all* $a \in \Sigma$ *and for all states* s*,* s' *and* t*, if* $s \sim t$ *and* $s \xrightarrow{a} a'$ *then there is a state* t' *such that* $s' \sim t'$ *and* $t \xrightarrow{a} t'$*. The partitioning of* S *induced by* $\sim$ *is a post-stable partitioning if for all* $a \in \Sigma$ *and for all states* s*,* s' *and* t*, if* $s' \xrightarrow{a} s$ *and* $s \sim t$ *then there exists a state* t' *such that* $t' \xrightarrow{a} t$ *and* $s' \sim t'$*.*

2.2 Specification Logics

Till now we have described formalisms for describing reactive systems. In order to reason about the behaviors of reactive systems, we need a formalism to specify their properties. In this section, we review some of the logics for specifying properties of transition systems. The μ calculus [Koz83] is a modal logic augmented with least and greatest fixpoint operator. The syntax of μ calculus formulas are given as follows.

$$\Phi \ ::= \ q \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \Diamond(\Phi) \mid X \mid \nu\, X.\Phi$$

where q is an atomic proposition and for a formula of the form $\nu\, X.\Phi$, every occurrence of X in Φ occurs under an even number of negations. We will also use the following abbreviations.

$$\begin{array}{lcl} \Box(\Phi) & \equiv & \neg\Diamond(\neg\Phi) \\ \mu\, X.\Phi(X) & \equiv & \neg\nu\, X.\neg\Phi(\neg X) \end{array}$$

A variable X in the formula is guarded iff every occurrence of X in Φ occurs in the scope of a modality operator $\Diamond$ ($\Box$). A formula Φ is guarded iff every bound variable in the formula is guarded [Wal93]. We now describe the semantics of μ calculus with respect to an unlabeled transition system $\mathcal{L} = \langle S, \{a\}, \longrightarrow, AP, P \rangle$. The meaning or denotation of a formula Φ in an (unlabeled) transition system $\mathcal{L}$ under an assignment $Val : Var \longrightarrow 2^S$, where Var is the set of variables of Φ, is the set of states of $\mathcal{L}$ in which Φ is true. It is denoted by $\mathcal{L}\Phi$ and is defined inductively as follows.

- $\mathcal{L}q = \{s \in S \ : \ q \in P(s)\}$
- $\mathcal{L}(\Phi_1 \wedge \Phi_2) = \mathcal{L}\Phi_1 \cap \mathcal{L}\Phi_2$
- $\mathcal{L}\neg\Phi = S \setminus \mathcal{L}\Phi$
- $\mathcal{L}(\Diamond(\Phi)) = \{s \in S \mid \exists s' \in Ss \longrightarrow s' \wedge s' \in \mathcal{L}\Phi\}$
- $\mathcal{L}X = Val(X)$
- $\mathcal{L}\nu\, X.\Phi = \bigcup\{S' \subseteq S \mid S' \subseteq \mathcal{L}\Phi\}$.

The μ-calculus is a very expressive specification logic. In the literature, different fragments of μ-calculus have been considered for which efficient model checking procedures exist. In Chapter 4, we consider a fragment of μ-calculus for which two symbolic model checking procedures exist — one based on backward analysis and the other based on forward analysis, that are both suitable for disjunctive constraints as the data structure for representing and manipulating sets of states.

2.3 Logic Programs and Datalog

As constraint query languages and constraints are going to play a central role in our uniform constraint-based framework, we review some related concepts about logic programming [Llo87] and datalog [Ull89] in this section. The usual viewpoint of logic programming is to look at the synthesis of operational behavior from programs viewed as executable specifications. In this dissertation, we take a different viewpoint; namely, the analysis of operational behaviors of constraint query language programs obtained by direct translation of a system (program). For the definition and semantics of logic programs and datalog we refer the reader to [Llo87, Ull89]. For an introduction to tabling and OLDT resolution, we refer the reader to [TS86a, CW96]. We review here only some of the terminology that will be used in the sequel. One of the most important notions in logic programming is that of OLD resolution.

Definition 2.5 (OLD resolution [TS86a]) *Let C be a negative clause $\longleftarrow A_1 \wedge \ldots \wedge A_n$ and D be a definite clause. Let D' be of the form $A \longleftarrow B_1 \wedge \ldots \wedge B_m$ $(m \geq 0)$, be D with all variables renamed so that there is no conflict with those in C. The clauses C and D are said to be OLD resolvable if A_1 and A are unifiable, and the negative clause (or null clause when $n = 1$ and $m = 0$) $\longleftarrow (B_1 \wedge \ldots \wedge B_m \wedge A_2 \wedge \ldots \wedge A_n)\theta$ is the OLD resolvent of C and D where θ is the mgu of A_1 and A. The restriction of the substitution θ to the variables of A_1 is called the substitution of the OLD resolution.*

Definition 2.6 (OLD Tree [TS86a]) *Let P be a program and C_0 be a negative clause. Then the OLD tree for the pair $\langle P, C_0 \rangle$ is a possibly infinite tree with its nodes labeled with negative or null clauses so that the following condition is satisfied.*

- *The root is labeled with C_0.*
- *Assume a node v is labeled with C.*
  - *If C is a null clause, then v is a terminal node.*
  - *Otherwise, let $D_1, \ldots, D_n$ $(n \geq 0)$ be all the clauses in P that are resolvable with C, and $C_1, \ldots, C_n$ be the respective resolvents. Then v has n child nodes, labeled with $C_1, \ldots, C_n$. The edge from v to the node labeled with C_i is labeled with θ_i, where θ_i is the substitution of the OLD resolution of C and D_i.*

Having defined OLD trees, we now come to the definition of OLD refutation.

Definition 2.7 (OLD refutation [TS86a]) *Given a program P and a negative clause C, an OLD refutation of C by P is a path in the OLD tree of $\langle P, C \rangle$, from the root to a node labeled with the null clause. Let $\theta_1 \ldots \theta_k$ be the labels of the edges on the path. The substitution of the refutation is the composition $\theta = \theta_0 \circ \ldots \circ \theta_k$, and the solution of the refutation is $C\theta$.*

Definition 2.8 (Unit Sub-refutation [TS86a]) *For a node v in an OLD tree, we denote the number of predicates in the goal labeling v by $leng(v)$. Consider a path from a node v_1 in an OLD tree to one of its descendants v_2 such that for every node v on the path $leng(v) > leng(v_2)$ holds. Let the goal labeling v_1 be $\langle P, \varphi \rangle$ where P is a conjunction of predicates p_1 to p_n. Let $k = n - leng(v_2)$. Since this path can be viewed as the refutation of the first k predicates, we call it a sub-refutation of the first k predicates (from the left). If $k = 1$ we call it a unit subrefutation.*

Generally, when we speak of a program, we essentially deal with the Clark's completion of a program. The least model of a program is the same as that of its Clark's completion. We now recall the definition of Clark's completion of a program.

Definition 2.9 (Clark's Completion [Llo87]) *The conjunction of all clauses $p(\mathbf{t}) \longleftarrow body_i$, defining a predicate p in a program P is, in fact, a syntactic sugaring for the formula that expresses the logical meaning correctly, namely the equivalence (here the existential quantifier is over all variables but those in $\mathbf{t}$)*

$$p(\mathbf{t}) \longleftrightarrow \bigvee_i \exists \ldots body_i.$$

The program P' that defines each predicate (that is defined in P) by such a (unique) equivalence is known as the Clark's completion. The two forms are equivalent with respect to the least model. The greatest model, however, refers to the Clark's completion.

In the sequel, we make it a rule that whenever we refer to a program, we will (unless otherwise stated) refer to the Clark's completion of the program. Whenever we talk about the greatest model of a program we will actually be talking about the greatest model of the Clark's completion of the program.

Let $\mathcal{P}$ be a logic program, let $\mathcal{H}$ be the Herbrand universe of $\mathcal{P}$, let $\mathcal{M}$ be a Herbrand model of P and let p be a predicate in $\mathcal{P}$. The *denotation* of p in $\mathcal{M}$ is the set of terms $\{t \in \mathcal{H} \,|\, p(t) \in \mathcal{M}\}$. We will use this notion in the sequel in this chapter as well as in Chapter 4.

2.4 OLDT Resolution

In this section, we briefly review OLDT resolution for logic programs [TS86a]. The presentation below is adapted from [TS86a]. In Chapter 3, we will extend OLDT resolution to constraint query language programs. We first need a few definitions.

Definition 2.10 (Partial OLD Tree [TS86a]) *A partial OLD tree is a finite top segment of an OLD tree. That is, any finite tree obtained by deleting arbitrary number of trees from an OLD tree.*

We assume that some predicates in a program are designated as *table predicates*.

Definition 2.11 (OLDT Structure [TS86a]) *An OLDT structure is a forest of partial OLD trees with two tables, the solution table and the lookup table.*

A node is called a *table node* if the leftmost atom of its label is a table predicate. A table node is either a *lookup node* or a *solution node*. The solution table associates the leftmost atom of the label of each solution node with a list of instances of the atom, called the *solution table*. The lookup table associates with each lookup node with a pointer pointing to some solution list in the solution table.

We now describe the table node registration procedure.

Definition 2.12 (Table Node Registration [TS86a]) *Given an OLDT structure and a table node v in it, the table node registration procedure classifies it as a solution node or a lookup node, and does necessary table manipulation, resulting in the OLDT structure.*

According to the leftmost atom A of v's label, we distinguish between the following cases. (By definition, the predicate of A is a table predicate).

1. *Lookup Node The atom A is an instance of some key entry A' in the solution table. Put v in the lookup table with a pointer to the entire solution list of A'.*

2. *Solution Node Otherwise, put A in the solution table with an empty solution list.*

Definition 2.13 (Initial OLDT Structure [TS86a]) *Given a program $\mathcal{P}$ and a goal C_0, the initial OLDT structure for the pair $\langle \mathcal{P}, C_0 \rangle$ is the result of the following operation.*

1. *Let T_0 be the OLDT structure consisting of a forest with a single node v_0 labeled with C_0, an empty solution table and an empty lookup table.*

2. *Apply the table node registration procedure to the node v_0 in T_0.*

2.4.1 Extension of an OLDT structure

We now describe how to extend an OLDT structure. The presentation closely follows [TS86a]. Given a program $\mathcal{P}$ and an OLD structure T, the immediate extension of T by $\mathcal{P}$ is the result of either of the following operations.

1. OLD extension Select a terminal node v, that is not a lookup node, such that its label C is not a null clause (goal) and at least one clause in $\mathcal{P}$ is OLD resolvable with C.

   (a) Let $D_1, \ldots, D_n$ $(n \geq 1)$ be all the clauses in $\mathcal{P}$ that are OLD resolvable with C, and let $C_1, \ldots, C_n$ be the respective OLD resolvents. Add n child nodes v_i, labeled with $C_1, \ldots, C_n$ to v. The edge e_i from v to v_i is labeled with θ_i where θ_i is the most general unifier of C and D_i.

   (b) For each new node, register it if it is a table node.

   (c) For each unit subrefutation (if any) starting from a solution node and ending with some of the new nodes, assume that the subrefutation is of $\longleftarrow A$ and let $\longleftarrow A'$ be a solution. Add A' to the last of the solution list of A, if A' is not an instance of any entry in the solution list.

2. Lookup Node Extension Select a lookup node v, such that the pointer associated with it points to a nonempty sublist of a solution list. Advance the pointer by one to skip the head element of the sublist. If C and $A \longleftarrow \mathit{true}$ are OLD resolvable, where C labels the node v and A is the entry of the table pointed to by the pointer, then create a child node of v labeled with the resolvent and label the new edge with the corresponding most general unifier. Do the same thing as in 1c.

An OLDT structure T' is an extension of another OLDT structure T if T' is obtained from T through successive application of immediate extensions. We now define OLDT refutation.

Definition 2.14 (OLDT refutation [TS86a]) *Given a program $\mathcal{P}$ and a goal C, an OLDT refutation of C by $\mathcal{P}$ is a path in some extension of the initial OLDT structure for $\langle \mathcal{P}, C \rangle$, from the initial root to a node labeled with the null goal. Here, by initial root, we mean the root inherited from the initial OLDT structure.*

The soundness of OLDT refutation comes as an immediate consequence of that of OLD refutation [Llo87]. The completeness comes from the completeness of OLD refutation along with the fact that an OLD refutation can be "simulated" by an OLDT refutation [TS86a].

2.5 Constraint Query Languages

In this section, we review some preliminaries of constraint query languages. For further details, the reader is referred to [KKR95, JM94]. Constraint query languages [JM94, JMMS, KKR95] are a natural merger of two declarative paradigms: constraint solving and deductive databases. The paradigm of constraint query languages has progressed in several and quite different directions. Before going into the details of constraint query languages, we make a brief review of the notions of constraint domains and solution compactness.

Definition 2.15 (Constraint Domain) *For any signature Σ, let $\mathcal{D}$ be a Σ structure and $\mathcal{L}$ be a class of Σ-formulas. The pair $\langle \mathcal{D}, \mathcal{L} \rangle$ is called a constraint domain.*

Examples of constraint domains are $\mathcal{R}$, the domain of reals, T_Σ^∞, the domain of infinite trees over an alphabet Σ. In most of this dissertation, we will either deal with the constraint domain of reals or that of natural numbers. In the rest of this dissertation, whenever there is no confusion, we will use the symbol $\mathcal{D}$ to denote the constraint domain $\langle \mathcal{D}, \mathcal{L} \rangle$ as well as the structure $\mathcal{D}$.

Definition 2.16 (Solution Compactness [JM94]) *Let $\langle \mathcal{D}, \mathcal{L} \rangle$ be a constraint domain. Let φ, φ_i range over formulas of $\mathcal{L}$ and let I be a possibly infinite index set. A constraint domain $\langle \mathcal{D}, \mathcal{L} \rangle$ is solution compact if it satisfies the following conditions.*

- $\forall \varphi \exists \{\varphi_i\}_{i \in I}$ *s.t.* $\mathcal{D} \models \forall \mathbf{x} \neg \varphi(\mathbf{x}) \Longleftrightarrow \bigwedge_{i \in I} \varphi_i(\mathbf{x})$.

We assume that the constraint domains that we will deal with below are solution compact [JM94]. We next come to the definition of o-minimal structures.

O-minimal structures and their theories play a major role in Chapter 7 in defining o-minimal hybrid systems, a decidable subclass of hybrid systems, where the underlying theory is o-minimal.

Definition 2.17 (O-minimal Structures [vdD98]) *A (first order) structure $\mathcal{D} = \langle \mathcal{U}, \Sigma \rangle$ over a signature Σ (where $\mathcal{U}$ is the universe of the structure and the vocabulary Σ contains the relation symbol $<$) is o-minimal if every definable subset of $\mathcal{U}$ can be expressed as a finite union of points and open intervals $(a, b) = \{x \mid a < x < b\}$, $(-\infty, a) = \{x \mid x < a\}$, and $(a, \infty) = \{x \mid x > a\}$.*

For example, the structure $\langle \mathcal{R}, <, +, ., 0, 1 \rangle$ is an o-minimal structure.

We now take a brief look at constraint query languages. A constraint query language program over a constraint domain $\mathcal{C}$ is a finite set of rules. A rule is of the form $H \longleftarrow B$ where H, the head is an atom and B the body is a finite set of non-empty set of literals. A literal is either an atom or a constraint. We let $\triangleright$ denote the empty sequence of literals. An atom has the form $p(t_1, t_2, \ldots, t_n)$ where p is an user-defined predicate symbol and t_i are terms from the constraint domain. Note that below, we write a program of the form $p(\mathbf{t}) \longleftarrow B$, where t is a tuple of terms, in the form $p(\mathbf{x}) \longleftarrow B \wedge \mathbf{x} = \mathbf{t}$. The operational semantics are given in terms of "derivations" from goals. Below, we review some basic concepts about constraint query languages.

Definition 2.18 (Non-ground Fact or Generalized Tuple) *A non-ground fact or a generalized tuple is a clause of the form $p(\mathbf{x}) \longleftarrow \varphi$ where $p \in Pred$ (the set of predicate symbols of the program) and φ is a constraint.*

Definition 2.19 (Non-ground Goal) *A non-ground goal is a tuple of the form $\langle P, \varphi \rangle$ where P is a conjunction of predicates from Pred and φ is a constraint. We will call φ the constraint store of the goal $\langle P, \varphi \rangle$.*

Definition 2.20 (Non-ground State) *A non-ground state is a tuple of the form $\langle p(\mathbf{x}), \varphi \rangle$ where $p \in Pred$ and φ is a constraint (store).*

Note that a nonground state is also a nonground goal.

Definition 2.21 (Non-ground Transition System) *Given a constraint query language program $\mathcal{P}$, we define the non-ground transition system induced by $\mathcal{P}$ as follows. Let $\langle P, \varphi \rangle$ be a non-ground goal. Let $p(\mathbf{x})$ be a predicate in the conjunct P. Let $P' = P \setminus \{p\}$ be the conjunction of all predicates in P other than $p(\mathbf{x})$. Let C be a clause in $\mathcal{P}$ whose head unifies with $p(\mathbf{x})$. Let B be the conjunction of predicates in the body of C. Then the non-ground transition system induced by $\mathcal{P}$ is the transition system whose set of states is the set of non-ground goals and the transition relation $\longrightarrow_{ng}$ is defined by:*

$$\langle P, \varphi \rangle \longrightarrow_{ng} \langle Q, \varphi' \rangle$$

where $Q = B \wedge P'$ and $\varphi' = \exists_{-(Variables(P'), Variables(B))}(\varphi \wedge ((\exists_{-\mathbf{x}}\varphi) \wedge \Theta \wedge \psi))$ where ψ is the constraint in C, $Variables(P')$ ($Variables(B)$) are the free variables in P' (B) and Θ is the mgu of $p(\mathbf{x})$ and the head of C where the existential quantifier is over all variables but $\mathbf{x}$.

Definition 2.22 (Non-ground Derivation.) *A non-ground derivation is a (finite or infinite) sequence of non-ground goals of the form*

$$G_1 \longrightarrow_{ng} G_2 \longrightarrow_{ng} \ldots$$

A finite derivation from G is finished if the last goal cannot be reduced. The last state in a finished derivation is of the form $\langle \triangleright, \varphi \rangle$. If φ is $false$ the derivation is said to have finitely failed. Otherwise the finite derivation is said to be successful. A nonground goal is said to have finitely failed if all nonground derivations starting from it finitely fail.

Definition 2.23 (Ground Instance.) *Given a non-ground state $s = \langle p(\bar{x}), \varphi \rangle$, a ground instance of s is a ground atom of the form $p(\alpha(x_1), \ldots, \alpha(x_n))$, where $\alpha : Var_\varphi \longrightarrow \mathcal{D}$ such that $\mathcal{D}, \alpha \models \varphi$ where $\mathcal{D}$ is the constraint domain under consideration (here Vr_φ denotes the set of free variables of φ). This definition can be easily extended to ground instances of non-ground goals.*

We sometimes use the term ground state for the term ground atom. A ground goal is a conjunction of ground atoms. A ground transition system and a ground derivation can be viewed as a "grounding" of a non-ground transition system and a non-ground derivation respectively. In this case, the "states" are ground goals and the transition relation is defined in the obvious way.

A constraint query language program is called *monolithic* iff for each clause, the body contains only one predicate symbol. For a predicate p and a set of generalized tuples $\mathcal{Q}$, we denote by $\mathcal{Q}_p$ the set of generalized tuples defining p in $\mathcal{Q}$; i.e., $\mathcal{Q}_p = p(\mathbf{x}) \longleftarrow \bigvee_{i=1}^{k} \psi_i$ where for $i = 1..k$, $p(\mathbf{x}) \longleftarrow \psi_i$ are all the clauses defining p in $\mathcal{Q}$.

Given a generalized tuple $G \equiv p(\mathbf{x}) \longleftarrow \varphi$ over a constraint domain $\mathcal{D}$ we define the denotation of G, denoted by $[G]_{\mathcal{D}}$, as follows

$$[G]_{\mathcal{D}} = \{p(\mathbf{d}) \mid \mathcal{D}, \mathbf{d} \models \varphi\}$$

Given a set *Tup* of generalized tuples over a constraint domain $\mathcal{D}$, we define the denotation of *Tup*, denoted by $[Tup]_{\mathcal{D}}$, as follows.

$$[Tup]_{\mathcal{D}} = \bigcup_{G \in Tup} [G]_{\mathcal{D}}$$

We are now going to demonstrate that the various formalisms used for specifying transition systems and their properties have a natural translation to the framework of constraint query languages. The verification problem then reduces to the problem of computing model-theoretic semantics of constraint query language programs. The framework of constraint query languages can be viewed as unifying these seemingly different formalisms.

2.5.1 Finite Automata and Constraint Query Languages

This section is inspired by [Pod00]. A finite automaton $\mathcal{A}$ over a finite non-empty alphabet Σ is an edge labeled directed graph with a finite set of vertices $Q = \{q_1, \ldots, q_n\}$. A subset F of Q of vertices is designated as the set of accepting vertices and a vertex q_1 is designated as the initial vertex. We call the pair $\langle q, w \rangle$, where q is a vertex of $\mathcal{A}$ and $w \in \Sigma^*$ is a finite word over the alphabet Σ, as a state of $\mathcal{A}$. Note that according to this definition of state, a finite automaton is an infinite state system. An edge of the automaton $\mathcal{A}$ is a triple $\langle q_i, a, q_j \rangle$ where q_i, q_j are vertices of $\mathcal{A}$ and $a \in \Sigma$. We denote the set of edges of the automaton by $\mathcal{E}$. The transitions of the automaton are described as follows. The state $\langle q_i, w \rangle$ can make a transition to the state $\langle q_j, w' \rangle$ if there exists an edge $\langle q_i, a, q_j \rangle$ of the automaton and $w = a \cdot w'$ where $\cdot$ denotes concatenation. A finite automaton $\mathcal{A}$ can be described by a regular system of equations.

$$q_i = \bigcup_{\langle q_i, a, q_j \rangle \in \mathcal{E}} a.q_j \cup S_i$$

where $S_i = \{\varepsilon\}$ if $q_i \in F$ and empty otherwise. The denotation of q_1 in the least solution of the above set of equations gives the language accepted by $\mathcal{A}$ in the classical sense [HU79]. Since we are interested in the least solution of the system of equations, we can rewrite the equations replacing equality by supersethood; i.e., we can rewrite the above system of equations as follows.

$$q_i \supseteq a \cdot q_j$$

for each edge $\langle q_i, a, q_j \rangle$ of $\mathcal{A}$ and

$$q_i \supseteq \{\varepsilon\}$$

if q_i is an accepting vertex. Using some syntactic sugar, we can rewrite the above inclusions as follows.

$$q_i \supseteq \{x \in \Sigma^* \mid \exists y \in \Sigma^* (y \in q_j \wedge x = a \cdot y)\}$$

We can rewrite the above inclusion logically as a constraint query language clause over the Herbrand constraint domain as follows.

$$q_i(x) \longleftarrow q_j(y) \wedge x = a \cdot y.$$

Let us denote the program generated by the above translation of $\mathcal{A}$ as $\mathcal{P}$. The correctness of the above translation is given by the following theorem.

Theorem 2.1 (Correctness of Translation) *The language accepted by $\mathcal{A}$ is exactly the denotation of the predicate q_1 in the least model of $\mathcal{P}$.*

Proof. By straightforward induction on length of derivations. ∎

Since model checking for linear safety properties (for finite state systems) in the automata-theoretic framework reduces to (non)-emptiness problem for finite automata on finite words [VW86a, HKQ98], our framework can provide a uniform platform for dealing with such problems. Checking emptiness amounts to checking membership in the model-theoretic semantics of constraint query language programs. In Section 2.7, we will look at some methods of computing model-theoretic semantics of constraint query language programs.

2.5.2 Pushdown Processes

A pushdown process $\mathcal{A}_P$ over a finite non-empty alphabet Σ consists of a finite set of locations $Q = \{q_1, \ldots, q_n\}$; a state of the system being a pair $\langle q, w \rangle$ where $w \in \Sigma^*$ is viewed as the representation of the contents of a stack. A subset F of Q is designated as the set of accepting locations; the location $q_1 \in Q$ is designated as the initial location. In addition to the transitions described for finite automata (which are now called *'pop'* transitions), we also have here a set of push edges of the form $\langle q_i, !a, q_j \rangle$ where $a \in \Sigma$. A state $\langle q_i, w \rangle$ takes a transition through a push edge $\langle q_i, !a, q_j \rangle$ to the state $\langle q_j, w' \rangle$ if $w' = a.w$. The language accepted by a pushdown process can be defined in the same way as that of a finite automaton. Using transformations similar to those in the previous section, a push transition through the edge $\langle q_i, !a, q_j \rangle$ can be described by the clause

$$q_i(x) \longleftarrow q_j(y) \wedge y = a \cdot x$$

Thus, similar to a finite automaton, a pushdown process $\mathcal{A}_P$ can be translated to a constraint query language program $\mathcal{P}$ such that the following is preserved.

Theorem 2.2 *The language accepted by a pushdown process $\mathcal{A}_P$ is exactly the denotation of the predicate q_1 in the least model of $\mathcal{P}$.*

2.5.3 Tree Automata

The following definitions are taken from [Tho90]. Given an alphabet Σ, a k-ary Σ-labeled tree t is a mapping $t : dom(t) \longrightarrow \Sigma$ where the domain of t denoted by $dom(t)$ is a subset of $\{0, \ldots, k-1\}^*$, closed under prefixes, which satisfies

$$wj \in dom(t), i < j \Longrightarrow wi \in dom(t).$$

The tree t is finite iff $dom(t)$ is finite. The outer frontier of a tree t is given by the set $fr^+(t) = \{wi \notin dom(t) \mid w \in dom(t) \wedge i < k\}$. Let $dom^+(t) = dom(t) \cup fr^+(t)$.

Definition 2.24 (Tree Automaton) *A (non-deterministic top-down) tree automaton over* Σ *is a quadruple of the form* $\mathcal{A}_T = \langle Q, q_0, \Delta, F\rangle$*, where* Q *is a finite set of states,* $q_0 \in Q$ *is designated as the initial state,* $F \subseteq Q$ *is the set of accepting states and* $\Delta \subseteq Q \times \Sigma \times Q^k$ *is the transition relation. A run of* $\mathcal{A}_T$ *on a finite* k*-ary* Σ*-labeled tree* t *is a mapping* $r : dom^+(t) \longrightarrow Q$ *where* $r(\varepsilon) = q_0$ *and* $\langle r(w), t(w), r(w0), \ldots, r(w(k-1))\rangle \in \Delta$ *for each* $w \in dom(t)$*. It is accepting if* $r(w) \in F$ *for each* $w \in fr^+(t)$*. The tree language* $T(\mathcal{A}_T)$ *recognized by a tree automaton* $\mathcal{A}_T$ *is the set of all (finite) trees* t *for which there is an accepting run of* $\mathcal{A}_T$ *on* t*.*

A tree automaton $\mathcal{A}_T$ can be translated to a constraint query language program $\mathcal{P}$ as follows. For each tuple $\langle q, a, q_1, \ldots, q_k\rangle \in \Delta$ we have the following clause.

$$q(y) \longleftarrow q_1(x_0), \ldots, q_k(x_{k-1}) \wedge y = a(x_0, \ldots, x_{k-1})$$

In addition , for each accepting state $q \in F$, we add a fact $q(\varepsilon)$. The intuition behind the translation is that the tree automaton reads the letter a of the term $a(x_0, \ldots, x_{k-1})$ in state q and splits into k copies and moves to node x_i with state q_{i+1}. The following theorem that we state without proof shows the correctness of the above translation.

Theorem 2.3 *The language accepted by the tree automaton* $\mathcal{A}_T$ *is exactly the denotation of the predicate* q_0 *in the least model of* $\mathcal{P}$*.*

2.5.4 Alternating Automata

Definition 2.25 (Alternating Automaton [MS87]) *An alternating automaton is a tuple* $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$*, where* Σ *is a finite non-empty alphabet,* Q *is a finite non-empty set of states,* $q_0 \in Q$ *is the initial state,* $F \subseteq Q$ *is a set of accepting states, and* $\delta : Q \times \Sigma \longrightarrow \mathcal{B}^+(Q)$ *(where* $\mathcal{B}^+(Q)$ *is the set of positive boolean formulas over* Q*) is a transition function.*

A run of $\mathcal{A}$ on a finite word $a_0, \ldots, a_{n-1}$ is a finite Q-labeled tree r such that $r(\varepsilon) = q_0$ and the following holds:

- if $|x| = i < n$, $r(x) = q$, and $\delta(q, a_i) = \theta$, then x has k children $x_1, \ldots, x_k$, for some $k \leq |Q|$, and the interpretation $\{r(x_1), \ldots, r(x_k)\}$ satisfies θ.

The run tree r is accepting if all nodes in depth n are labeled by states in F.

Similar to the previous three cases, it can be shown that the emptiness problem for alternating automata can be reduced to checking membership in the model-theoretic semantics of constraint query language programs. We leave out the formal details which are easy.

2.5.5 Automata on Infinite Words: Büchi Automata

Definition 2.26 (Büchi Automaton [Tho90]) *A Büchi automaton over a finite non-empty alphabet* Σ *is of the form* $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, F\rangle$ *where* Q *is a finite set of states,* $q_0 \in Q$ *is the initial state,* $\Delta \subseteq Q \times \Sigma \times Q$ *is the transition relation and* $F \subseteq Q$ *is the set of accepting states. A run* σ *of* $\mathcal{A}$ *on an infinite word* $a_0, a_1, \ldots$ *is an (infinite) sequence of states* $q_0, q_1, \ldots$ *where for each* $i \geq 0$ $\langle q_i, a_i, q_{i+1}\rangle \in \Delta$*; the run is accepting iff* $inf(\sigma) \cap F \neq \emptyset$ *where* $inf(\sigma)$ *is the set of states occurring infinitely along* σ*. An infinite word* $w \in \Sigma^\omega$ *is accepted by* $\mathcal{A}$ *iff it has an accepting run on* w*. The language of* $\mathcal{A}$*, denoted by* $\mathcal{L}(\mathcal{A})$*, is given as follows;* $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } w\}$*.*

Without going into the details, we note as above that the emptiness problem for Büchi automata can be reduced to that of testing membership in the model-theoretic semantics of constraint query language programs. Since the decision problem for linear temporal logic (LTL) formulas can be reduced to the emptiness problem for Büchi automata [VW86a], our framework provides a methodology for deciding LTL.

We note without going into the details that the emptiness problem for various formalisms of alternating tree automata (e.g., weak alternating automata, hesitant alternating automata etc. [BVW94]) can be uniformly captured in the framework of constraint query languages.

2.6 The Logic CTL

So far we have talked about how the seemingly different formalisms for specifying reactive systems have a natural translation to the framework of constraint query languages. In this section, we show how model checking can be performed in the framework of constraint query languages. In particular, we show how the model checking problem for the branching time temporal logic CTL can be reduced to computing the model-theoretic semantics of constraint query languages over reals. The syntax of the logic CTL [CE80], which is a fragment of the μ-calculus, is given as follows.

$$\Phi ::= true \mid q \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid EX(\Phi) \mid E(\Phi_1 \mathcal{U} \Phi_2) \mid E(\Phi_1 \widetilde{\mathcal{U}} \Phi_2)$$

where q is an atomic proposition. We will use the following abbreviations. $false \equiv \neg true$, $EF(\Phi) \equiv E(true\, \mathcal{U} \Phi)$, $AG(\Phi) \equiv \neg EF(\neg\Phi)$, $EG(\Phi) \equiv E(false\, \widetilde{\mathcal{U}} \Phi)$. The semantics of CTL with respect to an (unlabeled) transition system $\mathcal{L} = \langle S, \Sigma, S_0, \longrightarrow, AP, P\rangle$ (where $\Sigma = \{a\}$) is described as follows. The satisfaction relation is inductively defined as follows.

- For all $s \in S$, $\mathcal{L}, s \models true$.
- $\mathcal{L}, s \models q$ for $q \in AP$ iff $q \in P(s)$.
- $\mathcal{L}, s \models \neg\Phi$ iff $\mathcal{L}, s \not\models \Phi$.
- $\mathcal{L}, s \models \Phi_1 \vee \Phi_2$ iff $\mathcal{L}, s \models \Phi_1$ or $\mathcal{L}, s \models \Phi_2$.
- $\mathcal{L}, s \models EX(\Phi)$ iff there exists $s' \in S$ such that $\langle s, a, s'\rangle \in \longrightarrow$ and $\mathcal{L}, s' \models \Phi$.
- $\mathcal{L}, s \models E(\Phi_1 \mathcal{U} \Phi_2)$ iff there exists a path π starting from s and a natural number i such that $\mathcal{L}, \pi[i] \models \Phi_2$ and for all $0 \leq j < i$, $\mathcal{L}, \pi[j] \models \Phi_1$.
- $\mathcal{L}, s \models E(\Phi_1 \widetilde{\mathcal{U}} \Phi_2)$ iff there exists a path π starting from s such that for all $i \geq 0$ such that $\mathcal{L}, \pi[i] \not\models \Phi_2$, there exists $0 \leq j < i$ such that $\mathcal{L}, \pi[j] \models \Phi_1$.

The denotation of a CTL formula Φ with respect to an unlabeled transition system $\mathcal{L}$, denoted by $[\Phi]_{\mathcal{L}}$, is given by $[\Phi]_{\mathcal{L}} = \{s \in S \mid \mathcal{L}, s \models \Phi\}$.

2.6.1 Model Checking for CTL: Programs with Oracles

This section is inspired by [CP98a]. Based on the techniques of [CP98a], we show that model checking for CTL can be reduced to computing model-theoretic semantics of constraint query

language programs with oracles. We show this for a fragment of CTL. The extension to full CTL is straightforward.

We consider the following fragment of CTL (we call this fragment as FCTL).

$$\Phi ::= p \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid EX(\Phi) \mid EF(\Phi) \mid EG(\Phi)$$

We first note that a finitely representable unlabeled transition system can be described by a (monolithic) constraint query language program $\mathcal{P}$. Assume that the denotation of an FCTL formula Φ with respect to $\mathcal{P}$, denoted by $[\Phi]_{\mathcal{P}}$ can be described by a finite set of generalized tuples. Then the denotation of $\neg\Phi$ with respect to $\mathcal{P}$ can be described by a finite set of generalized tuples. If the denotations of Φ_1 and Φ_2 with respect to $\mathcal{P}$ can be described by finite sets of generalized tuples then so is the denotation of $\Phi_1 \vee \Phi_2$ with respect to $\mathcal{P}$.

Now, given that the denotation of Φ can be described by a finite set of generalized tuples $\mathcal{Q}$ (i.e., $[\Phi]_{\mathcal{P}} = \mathcal{Q}$), we construct the programs $\mathcal{P} \vee \mathcal{Q}$ and $\mathcal{P} \wedge \mathcal{Q}$ as follows.

$$\mathcal{P} \vee \mathcal{Q} \equiv \{C \mid C \in \mathcal{P} \vee C \in \mathcal{Q}\}$$

$$\mathcal{P} \wedge \mathcal{Q} \equiv \{p(\mathbf{x}) \longleftarrow p'(\mathbf{x}') \wedge \varphi \wedge \psi \mid p(\mathbf{x}) \longleftarrow p'(\mathbf{x}') \wedge \varphi \in \mathcal{P} \wedge \mathcal{Q}_p = \{p(\mathbf{x}) \longleftarrow \psi\}\}$$

where $\mathcal{Q}_p$ denotes the denotation of p in $\mathcal{Q}$. We call the above programs as programs with oracles. We have the following theorem.

Theorem 2.4 *For a finitely representable unlabeled transition system described by a constraint query language program $\mathcal{P}$ and an FCTL property Φ whose denotation with respect to $\mathcal{P}$ can be described by a finite set of generalized tuples $\mathcal{Q}$, the denotations of $EF(\Phi)$ and $EG(\Phi)$ with respect to $\mathcal{P}$ are given as follows.*

$$[EF(\Phi)]_{\mathcal{P}} = lm(\mathcal{P} \vee \mathcal{Q})$$

$$[EG(\Phi)]_{\mathcal{P}} = gm(\mathcal{P} \wedge \mathcal{Q})$$

where for a program $\widetilde{\mathcal{P}}$, $lm(\widetilde{\mathcal{P}})$ and $gm(\widetilde{\mathcal{P}})$ denote respectively the least and greatest models of $\widetilde{\mathcal{P}}$.

The proof of the above theorem can be developed directly along the lines of [CP98a].

We have thus seen that many verification problems (for both finite and infinite state systems) can be uniformly reduced to the problem of computing model-theoretic semantics of constraint query language programs. Developing optimized procedures for computing model-theoretic semantics of constraint query language programs thus provides, for free, procedures for solving a large class of verification problems.

2.7 Computing Model-Theoretic Semantics

In this section, we describe some techniques for computing model-theoretic semantics of constraint query language programs. We first start with propositional horn programs. Some of techniques below are inspired by (and extends) the techniques developed in [SIR96, Llo87, JM94, Ull89, DG84].

2.7.1 Dowling-Gallier Graphs

Given a propositional horn program $\mathcal{P}$ with n zero-ary predicates, the Dowling-Gallier graph for $\mathcal{P}$ is constructed as follows. The graph $\mathcal{G}$ has $n+2$ nodes, n nodes corresponding to n zero-ary predicates and two special nodes designated $true$ and $false$. If the clause C_i is a fact of the form p, then there is a directed edge labeled i from the node labeled $true$ to the node labeled p. If the clause C_i is of the form $p \longleftarrow q_1 \wedge \ldots \wedge q_k$, then there are directed edges labeled i from each of the nodes corresponding to $q_1, \ldots, q_k$ to the node corresponding to p.

Definition 2.27 (Pebbling for Least Model [DG84]) *Let $\mathcal{G} = \langle V, E, L \rangle$ be an edge labeled directed graph. There is a pebbling from a node $p \in V$ from a set $X \subseteq V$ if either $p \in X$ or for some label i, there are pebblings from $q_1, \ldots, q_k$ from X, where $q_1, \ldots, q_k$ are the source of all incoming edges labeled i to p.*

Theorem 2.5 *[DG84] Given a propositional horn program $\mathcal{P}$ and the Dowling-Gallier graph $\mathcal{G}$ corresponding to it, the following holds.*

- *If $\mathcal{P}$ is satisfiable, the set of all zero-ary predicates p in $\mathcal{P}$, such that there is a pebbling of the node corresponding to p in $\mathcal{G}$ from $true$, is the least model of $\mathcal{P}$.*

Moreover, the least model of $\mathcal{P}$ can be computed in time linear in the size of $\mathcal{P}$.

Before describing the pebbling for greatest models, we modify the Dowling-Gallier graph as follows. For each zero-ary predicate p such that p is not defined in $\mathcal{P}$, add a directed edge labeled $n+1$ (where n is the number of clauses in $\mathcal{P}$) from the node corresponding to $false$ to that corresponding to p. We call this graph the modified Dowling-Gallier graph.

Definition 2.28 (Greatest Model Pebbling) *Let $\mathcal{G} = \langle V, E, L \rangle$ be an edge-labeled directed graph. There is a pebbling of a node $p \in V$ from a set $X \subseteq V$ if either $p \in X$ or for each label i such that there exists an incoming edge to p labeled i, there exists a node q_j such there is a pebbling of q_j and there is an edge labeled i from q_j to p.*

Theorem 2.6 *Given a propositional horn program $\mathcal{P}$ and the modified Dowling-Gallier graph $\mathcal{G}$ corresponding to it, the following holds.*

- *If $\mathcal{P}$ is satisfiable, the set of all zero-ary predicates p in $\mathcal{P}$, such that there is no pebbling of the node corresponding to p from in $\mathcal{G}$ from $false$, is the greatest model of $\mathcal{P}$.*

Moreover, the greatest model of $\mathcal{P}$ can be computed in time linear in the size of $\mathcal{P}$.

Proof. Suppose a zero-ary predicate p is in the greatest model of $\mathcal{P}$. Then, in the (SLD) derivation tree of $\mathcal{P}$ starting from p, there is a success leaf or an infinite branch (derivation). We define the maximum length of pebbling from $false$ in $\mathcal{G}$ as follows. Suppose that a node p is pebbled from $false$. Then a pebbling route from $false$ to p is defined as follows. If the predicate p is not defined in the program $\mathcal{P}$, then the pebbling route from $false$ is $[n+1]$. Otherwise, if there is an edge labeled i from q to p and q has a pebbling from $false$, then L is a pebbling route from $false$ to p where $L = concat([i], L')$ where $concat$ returns the concatenation of two lists and L' is a pebbling route from $false$ to q and i does not occur in L'. The length of a pebbling route L is the length of the list L. Clearly, the length of each pebbling route is finite (since there

are only a finite number of clauses in $\mathcal{P}$). Also, since the length of each pebbling route is finite, there are only a finite number of pebbling routes from $false$ to any node p. The maximum length of pebbling from $false$ to p is the maximum of the length of all pebbling routes from $false$ to p. Suppose p is in the greatest model of $\mathcal{P}$. Also, seeking a contradiction, suppose that there is a pebbling from $false$ to p. Let the maximum length of pebbling route from $false$ to p be k. We show by induction that if p is in the greatest model of $\mathcal{P}$, then the maximum length of pebbling route from $false$ to p cannot be any positive integer. Indeed, if p is in the greatest model of $\mathcal{P}$, then the maximum length of pebbling route cannot be 1. Assume that if p is in the greatest model of $\mathcal{P}$ then the maximum length of pebbling route from $false$ to p cannot be less than or equal to $k-1$. Suppose that p is pebbled and the maximum length of pebbling route from $false$ to p is k. Since p is in the greatest model of $\mathcal{P}$, there must exist either a successful derivation or an infinite derivation starting from p. Let the first clause in the derivation be the ith clause $p \longleftarrow q_1 \wedge \ldots \wedge q_m$ of the program $\mathcal{P}$. Now each of $q_1, \ldots, q_m$ is in the greatest model of $\mathcal{P}$. Since p is pebbled, at least one of q_j must be pebbled; without loss of generality, let it be q_l. Since, the maximum length of pebbling route from $false$ to p is k, the maximum length of pebbling from $false$ to q_l can be at most $k-1$. By induction hypothesis, the maximum length of pebbling route from $false$ to q_l cannot be less than or equal to $k-1$. This is a contradiction. Hence, there does not exist a pebbling from $false$ to p.

To prove the other way, it is easy to show that if there is no pebbling from $false$ to p, then there exists a derivation that either succeeds or is infinite. Hence, p is in the greatest model of $\mathcal{P}$. It is also easy to see that the greatest model of $\mathcal{P}$ can be computed in time linear in the size of $\mathcal{P}$. ‖

2.7.2 Immediate Consequence Operator

We briefly discuss about the immediate consequence operator. For details, the reader is refereed to [Llo87, JM94].

Definition 2.29 (Immediate Consequence Operator) *Given a constraint query language program $\mathcal{P}$ over a constraint domain $\mathcal{D}$, the immediate consequence operator $S^{\mathcal{D}}_{\mathcal{P}}$ is defined on sets of facts, that form a complete lattice under the subset ordering. The immediate consequence operator is defined as follows.*

$$\begin{aligned} S^{\mathcal{D}}_{\mathcal{P}}(I) \;=\; & \{p(\mathbf{x}) \longleftarrow \varphi \mid p(\mathbf{x}) \longleftarrow \varphi' \wedge b_1 \wedge \ldots \wedge b_n \text{ is a rule of } \mathcal{P} \\ & a_i \longleftarrow \varphi_i \in I, i = 1, \ldots, n, \text{ the rules and facts are renamed apart} \\ & \mathcal{D} \models \varphi \Longleftrightarrow \exists_{-\mathbf{x}} \varphi' \wedge \textstyle\bigwedge_{i=1}^{n} \varphi_i \wedge a_i = b_i\} \end{aligned}$$

where the existential quantifier is over all variables but $\mathbf{x}$.

Let $\mathbf{PRED}(\mathcal{P})$ be the set of predicate symbols in $\mathcal{P}$. Let $\mathcal{B}_{\mathcal{D}}$ denote the $\mathcal{D}$-base for a program $\mathcal{P}$; i.e.,

$$\mathcal{B}_{\mathcal{D}} = \{p(\mathbf{d}) \mid p \in \mathbf{PRED}(\mathcal{P}) \wedge \mathbf{d} \in \mathcal{D}^k\}.$$

Theorem 2.7 *The following holds for the immediate consequence operator.*

1. $SS(\mathcal{P}) = [lfp(S^{\mathcal{D}}_{\mathcal{P}})]_{\mathcal{D}} = [lm(\mathcal{P})]_{\mathcal{D}}$ *where* $SS(\mathcal{P})$ *represents the success set of* $\mathcal{P}$.
2. $[gm(\mathcal{P})]_{\mathcal{D}} = [gfp(S^{\mathcal{D}}_{\mathcal{P}})]_{\mathcal{D}} = \mathcal{B}_{\mathcal{D}} \setminus [FF(\mathcal{P})]_{\mathcal{D}}$, *where* $FF(\mathcal{P})$ *is the set of non-ground states that are finitely failed.*

Proof. We refer the reader to [JM94] for the proof of the first statement. For the second statement, for the proof of the equality $[gm(\mathcal{P})]_{\mathcal{D}} = [gfp(S_{\mathcal{P}}^{\mathcal{D}})]_{\mathcal{D}}$, we again refer the reader to [JM94]. We prove the equality $[gm(\mathcal{P})]_{\mathcal{D}} = \mathcal{B}_{\mathcal{D}} \setminus [FF(\mathcal{P})]_{\mathcal{D}}$. We first refer the reader to the notions of finitely failed SLD trees in [Llo87, JM94]. Now suppose that $p(\mathbf{x}) \longleftarrow \varphi \in gm(\mathcal{P})$ and let φ be satisfiable. Also, seeking a contradiction, suppose that $p(\mathbf{x}) \longleftarrow \varphi \in FF(\mathcal{P})$. We use the following equality [JM94]

$$[gm(\mathcal{P})]_{\mathcal{D}} = \mathcal{B}_{\mathcal{D}} \setminus GFF(\mathcal{P})$$

where $GFF(\mathcal{P})$ is the set of all ground atoms that finitely fail (in $\mathcal{P}$). Suppose that $\langle p(\mathbf{x}), \varphi \rangle$ finitely fails. Consider $\mathbf{d}$ such that $\mathcal{D}, \mathbf{d} \models \varphi$. Now $p(\mathbf{d})$ finitely fails (i.e., has a finitely failing SLD tree). But $p(\mathbf{d}) \in [gm(\mathcal{P})]_{\mathcal{D}}$. This is a contradiction.

The proof for the other direction is similar. ∎

2.7.3 Magic Sets Transformation

Given a program $\mathcal{P}$ and a query Q, if we are interested only in the answers to the query Q, then computing the least model of $\mathcal{P}$ using iterations of the immediate consequence operator can be wasteful. In this subsection, we describe the magic sets transformation, that is used to make query evaluation goal directed. For details, we refer the reader to [BMSU86].

Definition 2.30 (Magic Sets Transformation) *Let $\mathcal{P}$ be a program and $\langle Q(\mathbf{x}), \psi \rangle$ be a query. The magic sets transformation of $\mathcal{P}$ is a new program $\mathcal{P}'$ obtained as follows. Initially, $\mathcal{P}'$ is empty.*

- *Create a new predicate p_{in} for each predicate $p \in \mathcal{P}$. The arity of p_{in} is the same as that of p.*
- *For each rule in $\mathcal{P}$ add the modified version of the rule to $\mathcal{P}'$. If a rule has head $p(\mathbf{x})$, the modified version of the rule is obtained by adding the literal $p_{in}(\mathbf{x})$ in the body of the rule.*
- *For each rule r in $\mathcal{P}$ with head $p(\mathbf{x})$ and for each body literal $q(\mathbf{y})$ of r, add a magic rule to $\mathcal{P}'$. The head is $q_{in}(\mathbf{y})$. The body contains the constraint φ of r. In addition, it contains the literal $p_{in}(\mathbf{x})$ as well as all literals to the left of $q(\mathbf{y})$ in the body of r.*
- *Create a seed fact $Q_{in}(\mathbf{x}) \longleftarrow \psi$ from the query Q.*

Further optimizations to the magic sets transformation can be obtained by capturing classes of binding patterns. We refer the interested reader to [Ram91] for further details.

2.8 Constraint Domains

Having discussed about the different ways of computing model-theoretic semantics of constraint query language programs, we now come to describe some relevant properties of constraint domains. In this section, we discuss briefly about the properties of some constraint domains. We start with the domain of (possibly) infinite trees. We assume some familiarity with the basic notions of general topology on the part of the reader.

2.8.1 The Constraint Domain of Infinite Trees

Below, we prove some facts about the constraint domain of infinite trees. Let Σ be a finite ranked alphabet. As before, we can define (possibly infinite) trees labeled by Σ, where if a node w is labeled by f and the rank of f is k then w has exactly k children. We also assume that $|\Sigma| > 1$. We denote the set of possibly infinite Σ-labeled trees by T_Σ^∞. Before we describe some topological properties of this domain, we need a few definitions.

A *metric space* $\langle X, d\rangle$ is a set X together with a mapping $d : X \times X \longrightarrow \mathcal{R}^+ \cup \{0\}$ that satisfies the following conditions.

- $\forall x, y \in X, d(x, y) = d(y, x)$.
- $\forall x, y \in X, d(x, y) = 0 \Longleftrightarrow x = y$.
- $\forall x, y, z \in X, d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality).

For a metric space $\langle X, d\rangle$, a subset $A \subseteq X$ is *open* iff it is an union of ε-balls. A metric space is *complete* if every Cauchy-sequence is converges in it [Kur66]. A metric space is *totally bounded* if for every $\varepsilon > 0$ there exists a finite covering of X by ε-balls. A metric space is *compact* iff it is complete and totally bounded. A metric space $\langle X, d\rangle$ is *disconnected* if there exists (nonempty) open sets $A \neq X$ and $B \neq X$ such that $X = A \cup B$ and $A \cap B = \emptyset$. Otherwise it is connected.

We define a metric d on the set of infinite trees as follows.

$$\begin{aligned} d(x, y) &= 0 \text{ iff } x = y \\ &= 2^{-\alpha(x,y)} \text{ otherwise} \end{aligned}$$

where $\alpha(x, y)$ is the least depth at which x and y differ. Now $\langle T_\Sigma^\infty, d\rangle$ is a metric space. It can also be shown that $\langle T_\Sigma^\infty, d\rangle$ is a complete metric space.

Theorem 2.8 *The metric space $\langle T_\Sigma^\infty, d\rangle$ is compact.*

Proof. Let us take the discrete topology on Σ. Now Σ is obviously compact in the discrete topology. By Tychonoff's theorem, Σ^ω is compact in the product topology. The product topology is generated by the metric

$$d_p(x, y) = \sum_{i \in \omega} d^t(x(i), y(i))/2^i$$

where d^t is the trivial metric on Σ.

Now we consider the following space. We augment our alphabet with an additional symbol $\Box$ having arbitrary rank. Now we consider the space of possibly infinite trees over the alphabet $\Sigma \cup \{\Box\}$ such that the following holds.

- The root of each tree is labeled by $\Box$. The symbol $\Box$ can label only the root of a tree.
- Only the rightmost child of $\Box$ can be possibly infinite trees (the rest of the children must be finite trees).
- The root can have arbitrary number of children. The number of children of the other nodes is equal to the rank of the symbol labeling that node.

We denote the above set of possibly infinite trees by τ_Σ^∞. Now we define a metric d^τ on τ_Σ^∞ as follows.

$$d^\tau(x,y) = \sum_{each\ child\ of\ \Box} d(x(i),y(i))/2^{\alpha_i}$$

where $x(i)$ and $y(i)$ are respectively the ith children of $\Box$ in x and y, $\alpha_i = i + max(\sum_{j=0}^{i-1} depth(x(j)), \sum_{j=0}^{i-1} depth(y(j)))$ where $depth(x(j))$ $(depth(y(j)))$ is the depth of the jth child of $\Box$ in x (y), and d is the metric defined above. It can be easily checked that d^τ is a metric and hence $\langle \tau_\Sigma^\infty, d^\tau \rangle$ is a metric space.

Now we define a mapping f from the metric space $\langle \Sigma^\omega, d_p \rangle$ to $\langle \tau_\Sigma^\infty, d^\tau \rangle$ as follows. The mapping f builds a tree in τ_Σ^∞ from a string $w = f^0 \ldots$ as follows. First the root is labeled $\Box$ with one child f^0. Then the tree with root f^0 is built in a breadth-first manner. Whenever we cannot attach any more symbol to the tree corresponding to f^0 (since all leaves may be labeled with symbols of rank 0), we bring out a right child of $\Box$ and insert the next symbol. Then we continue the same procedure for this child (thus only the rightmost child of $\Box$ can be an infinite tree). Thus for the strings $aa\ldots$ and $faa\ldots$ where f is of rank 1 and a is of rank 0, the constructed trees are given in the left-half and right half of Figure 2.1 respectively.

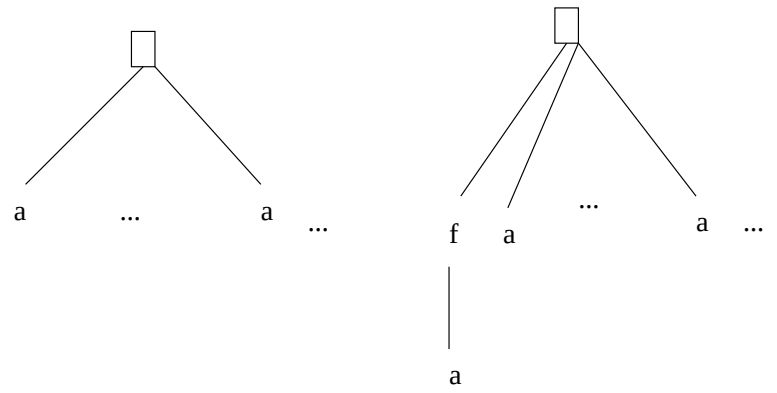


Figure 2.1: Illustrating the mapping f

It can be easily verified that f is a homeomorphism. Hence, $\langle \tau_\Sigma^\infty, d^\tau \rangle$ is a compact space. Now consider the subset A of τ_Σ^∞ in which the root has only one child. It can be easily verified that A is a closed set. Hence $\langle A, d^\tau \rangle$ is compact.

Now we define a mapping g from the the subspace $\langle A, d^\tau \rangle$ of $\langle \tau_\Sigma^\infty, d^\tau \rangle$ to the metric space $\langle T_\Sigma^\infty, d \rangle$ as shown in Figure 2.2. It can again be easily verified that g is a homeomorphism from $\langle A, d^\tau \rangle$ to $\langle T_\Sigma^\infty, d \rangle$. Hence $\langle T_\Sigma^\infty, d \rangle$ is compact. ‖

Let $Var = \{x_1, x_2, \ldots\}$ be a countably infinite set of variables. Let X be the set $\{\alpha \mid \alpha : Var \longrightarrow T_\Sigma^\infty\}$. Let us define the metric d_X on X as follows.

$$d_X(\alpha,\beta) = \sum_{i \in \omega} d(\alpha(x_i), \beta(x_i))/2^i$$

where the metric d is as defined above. It can be easily verified that $\langle X, d_X \rangle$ is a metric space.

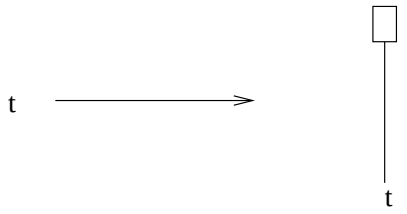


Figure 2.2: Illustrating the mapping g

Proposition 2.1 *The sets*

$$S_i = \{\alpha \in X \mid T_\Sigma^\infty, \alpha \models x = f_i(x_{i_1}, \ldots, x_{i_{n_i}})\}$$

are closed sets in the metric space $\langle X, d_X \rangle$.

Proof. Without loss of generality, let

$$A = \{\alpha \in X \mid T_\Sigma^\infty, \alpha \models x = f(x_1, \ldots, x_n)\}$$

Let l be a limit point of A. Let $x = x_k$ for some $k \in \{1, 2, \ldots\}$. Let $m > max(n, k) = p$. Therefore, we can get an $\alpha_m \in A$ such that $d_X(\alpha_m, l) \leq 2^{-m}$. Therefore,

$$\forall i \leq p, d(\alpha_m(x_i), l(x_i)) \leq 2^{-(m-p)}.$$

Obviously, the root of $l(x)$ is labeled by f. Let $l(x) = f(t_1, \ldots, t_n)$. We will prove that $l(x_j) = t_j$, $1 \leq j \leq n$. Since

$$d(\alpha_m(x), l(x)) \leq 2^{-(m-p)}$$

therefore

$$d(\alpha_m(x_i), t_i) \leq 2^{-(m-p-1)}.$$

Now,

$$d(l(x_i), t_i) \leq 2^{-(m-p)} + 2^{-(m-p-1)} < 2^{-(m-p-2)}$$

(by triangle inequality). Now we can choose m to be arbitrarily large so that $d(l(x_i), t_i)$ is less than any positive number. Therefore $l(x_i) = t_i$ and hence l is an element of A. Therefore A is closed. ‖

Proposition 2.2 *The metric space* $\langle X, d_X \rangle$ *is compact.*

Proof. The metric d_X generates the product topology on X. Now the space of infinite trees T_Σ^∞ is compact in the topology generated by d. Therefore, by Tychonoff's theorem, the metric space $\langle X, d_X \rangle$ is compact. ‖

Proposition 2.3 *The metric space* $\langle X, d_X \rangle$ *is disconnected.*

Proof. We consider a partition of the signature Σ into nonempty subsets Σ_1 and Σ_2 (this is possible as $|\Sigma| > 1$). Clearly for any f_i the set $\{\sigma \in X \,|\, T_\Sigma^\infty, \sigma \models x = f_i(x_{i_1}, \ldots, x_{i_{m_i}})\}$ is closed. Consider the two closed sets C_1 and C_2 (these two sets are closed since they are finite union of closed sets) given by

$$C_1 = \{\sigma \in X \mid T_\Sigma^\infty, \sigma \models \bigvee_{f_i \in \Sigma_1} x = f_i(x_{i_1}, \ldots, x_{i_{m_i}})\}$$

$$C_2 = \{\sigma \in X \mid T_\Sigma^\infty, \sigma \models \bigvee_{f_j \in \Sigma_2} x = f_j(x_{j_1}, \ldots, x_{j_{m_j}})\}.$$

Obviously, there cannot exist $\sigma \in X$ that satisfies both the equations. So $C_1 \cap C_2 = \emptyset$. Also $C_1 \cup C_2 = X$. So $\langle X, d_X \rangle$ is disconnected. ∎

Proposition 2.4 *The metric space $\langle T_\Sigma^\infty, d \rangle$ is disconnected.*

Proof. Suppose it is connected. Then $\langle X, d_X \rangle$ is connected which is a contradiction. ∎

The Constraint Domain of Reals In this paragraph, we state a few facts about $\mathcal{R}$, the constraint domain of reals [JMSY92]. Essentially, $\mathcal{R}$ is a two-sorted structure where one sort is the real numbers and the other sort is the set of trees over uninterpreted functors and real numbers. The constraint domain $\mathcal{R}$ is solution compact [JMSY92]. The metric space $\langle \mathcal{R}, d_e \rangle$ where $\mathcal{R}$ is the set of reals and d_e is the Euclidean metric is non-compact but is connected (we denote both the set of reals and the constraint domain of reals by $\mathcal{R}$; the meaning will be clear from the context). Below, whenever we speak of the constraint domain $\mathcal{R}$, we assume the absence of function symbols (other than constant symbols). With this assumption, the structures $\mathcal{R}_{Lin} = \langle \mathcal{R}, <, +, -, 0, 1 \rangle$, $\mathcal{R}_F = \langle \mathcal{R}, <, +, \cdot, 0, 1 \rangle$ are o-minimal structures.

2.8.2 Constraint Simplification

In this section, we describe the Fourier's algorithm [MS98, LM92], needed to simplify linear constraints over reals. Given a constraint φ involving a set of variables V, and a subset U of V, the Fourier's algorithm produces a constraint φ' obtained from φ by eliminating all variables in U. The description below assumes non-strict linear inequalities. Extension to strict linear inequalities is straightforward. Given a constraint φ as a set (conjunction) of linear inequalities, elimination of a variable y proceeds as follows: First partition φ into three subsets, the subset φ^0, consisting of inequalities which do not involve y, the subset φ^1, consisting of inequalities of the form $y \leq t$, where t does not involve y, and the subset φ^2 consisting of inequalities of the form $t \leq y$, where again t does not involve y (this is possible since we are dealing with linear constraints). Now for each pair of the form $t_1 \leq y$ in φ^2, and $y \leq t_2$ in φ^1, form an inequality of the form $t_1 \leq t_2$. This new set of inequalities along with those in φ^0 form the projection of the original constraint φ on to the original variables but y. This process is then repeated to eliminate all the variables in U.

Chapter 3

Model Checking for Timed Logic Processes

3.1 Introduction

Some software and hardware components meet the tasks for which they have been designed only if they relate properly to the passage of time. Behaviors of such computing systems operating in real time are difficult to predict by "inspection". Therefore real-time systems have become prime targets of formal methods for specification and verification methods [AD94, DT98, LPY95a, SS95]. In this chapter, we apply techniques from logic programming [TS86a, CW96] and constraint databases [KKR95, Rev90, JM94] to specify and verify real time systems.

We single out a fragment of constraint query languages [JM94, KKR95] over reals that allows us to model real-time systems operating over dense time. We call the programs expressed in this fragment as *timed logic processes* (abbreviated TLPs). We show a formal connection of TLPs with the standard model of timed automata [AD94]. We use this connection to design model checking procedures for the logic $\mathcal{L}_s$ [LPY95a] ("logic of safety and bounded liveness") and some extensions of it. Using a product construction for TLPs, we reduce the model checking problem for time logic processes against $\mathcal{L}_s$ formulas to the membership problem for the model-theoretic semantics of product timed logic processes (see Section 3.6).

To obtain a local model checker for real time systems, we extend with constraints the OLDT - resolution for logic programs [TS86a, CW96]. This way, we explain the model checking procedure of UPPAAL [LPY95a, BLL+96] based on a rewrite tree as a special case of OLDT resolution with constraints. We have implemented a prototype model checker for timed systems based on OLDT resolution with constraints using the CLP($\mathcal{R}$) [JMSY92] system of Sicstus 3.7. We have applied our prototype model checker to some standard benchmark examples and we have got reasonably good timings for these examples.

Thanks to the logical setup, we have been able to use $\mathcal{L}_s$ [LPY95a] extended with full disjunction (in contrast with $\mathcal{L}_s$ with restricted disjunction used in [LPY95a]) as the underlying logic for our model checker.

Generally, forward analysis (top-down evaluation) for timed systems, including the rewrite-tree-based model checking procedure of [LPY95a], is possibly non-terminating. To guarantee the termination of the procedure for checking membership for the least model semantics of timed

logic processes, we introduce a (new) operation on constraints (see Section 3.7). This operation, called trimming, allows us to completely avoid the computationally expensive operation of *splitting constraints* (in contrast with [SS95] where the authors construct a *region product graph* on the fly using methods similar to [ACD+92, BFH91, YL93]; such a construction, while guaranteeing termination of the model checking algorithm of [SS95], inherently involves the operation of splitting constraints; the operation of splitting constraints is known to be expensive) while still guaranteeing the termination of the procedure. Unlike many other constraint-based operations in literature (see e.g., [DT98]), the constraint-based operation that we have introduced also has a logical characterization.

We next turn our attention to model checking for unbounded liveness properties (which are not expressible in $\mathcal{L}_s$). Using the same product construction as mentioned above, we reduce the problem of model checking for timed logic processes against unbounded liveness properties (see Section 3.10) to the membership problem for the greatest model of a product TLP. To obtain a local model checker for unbounded liveness properties, we introduce a new kind of tabled resolution (for TLPs having at most one predicate in the body of any clause) to locally check if a ground atom is in the greatest model of a TLP. We call this resolution *greatest model resolution*. Greatest model resolution allows us to avoid the costly splitting operation on constraints that arises due to negation. To the best of the knowledge of the authors, tabled resolution (without using negation) has not been previously used to solve the membership problem for the greatest model of a constraint query language program. We have also been able to combine greatest model resolution with the tabled resolution mentioned above to verify receptiveness properties of timed logic processes. The model checker UPPAAL [BLL+96] is not able to model check for receptiveness properties.

Our last contribution in this chapter is to define (and present an algorithm for detecting) for the first time a notion of transience (see Section 3.13) which characterizes the transient behavior (response) of a real time system. The notion of transience is important in control theoretic applications. In control theory, underdamped (linear time-invariant) systems are known to have a transient and a steady state behavior. We capture this notion of transient (or underdamped) behavior in the context of real time systems; intuitively, a behavior is transient if it is observed "initially", but "disappears" with the passage of time. A timed logic process is transient if it has a transient behavior. We reduce the problem of deciding whether a timed logic process is transient to the non-emptiness problem for a nonground Büchi automaton induced by a (transformed) TLP (see Section 3.13.1). This reduction enables us to obtain a EXPSPACE algorithm for deciding transience.

3.2 Timed Automata

In this section, we briefly review the standard notion of timed automata. We do not view timed automata from the formal language point of view. Instead, we view them from the timed transition system point of view. A timed automaton is a finite state (location) automaton with timing elements added that take values from $\mathcal{R}$ the set of nonnegative reals. More precisely, a set of resetable clocks are added that measure progress of real time. A clock x is a variable, taking real values, such that it increases with slope 1, and the only operations that can be done on x are: a) test whether the value of x satisfies a constraint and b) reset x to zero. Let $X_n = \{x_1, x_2, \ldots, x_n\}$ be a set of variables standing for the values of the clocks. Let $guard_n$

be the set of formulae (called clock constraints) where $Var(guard_n) \subseteq X_n$ (for a formula φ, we denote its set of free variables by $Var(\varphi)$). The n in the suffix of $guard_n$ denotes the number of free variables of $guard_n$. A formula in $guard_n$ is given by:

$$\theta \ ::= \ true \mid x_i > c \mid x_i < c \mid x_i \geq c \mid x_i \leq c \mid \theta_1 \wedge \theta_2$$

where $c \in \mathcal{N}$, the set of natural numbers. We will sometimes call these formulas *clock constraints*.

Timed automata has been introduced by Alur and Dill [AD94]. The definition below stems from [HK97]:

Definition 3.1 *A Timed Automaton [HK97] is a seven-tuple*

$$U = \langle AP, X_n, L, E, P, \ell^0, inv \rangle$$

where

- *AP is a set of atomic propositions.*
- *X_n is a finite set of variables where each variable stands for a Program Clock.*
- *L is a finite set of locations.*
- *$E \subseteq L \times guard_n \times 2^{\{1,\ldots,n\}} \times L$ is a transition relation.*
- *$P : L \longrightarrow 2^{AP}$ assigns to each location a set of atomic propositions*
- *$\ell^0 \in L$ is the initial location.*
- *$inv : L \longrightarrow guard_n$ assigns to each location an invariant.*

Invariants, which are formulae of $guard_n$, can be introduced in the locations of the timed automaton to ensure that the control moves from one location to another. An edge $e \in E$ is a triple consisting of a source location (from L), a guard formula (from $guard_n$), a subset of $\{1, \ldots, n\}$ denoting the set of clocks that are reset in e and a target location (from L). In Figure 3.1 an example of a timed automaton is given. There are two locations $l0$ and $l1$ and two clocks x and y. The invariants of the locations are given at the top of each location. The clocks that are reset in each transition are shown explicitly in the transitions in the form $x := 0$ where x is a clock.

We now describe the semantics of timed automata. Informally, either the control stays at a location and let time pass (i.e., increment the clock variables) provided the invariant of that location is satisfied. Or the control jumps (instantaneously) from one location to another through an edge provided the values of the clocks satisfy the guard of that edge. Some of the clocks are reset to zero in this jump while others are kept unchanged.

A *position* of U is of the form $\langle \ell, v_1, \ v_2, \ldots, \ v_n \rangle$ where $\langle v_1, \ v_2, \ldots, \ v_n \rangle \ \in \ \mathcal{R}^n$ and $\ell \in L$. In the sequel, we will use the notations $\langle \ell, \mathbf{v} \rangle$ and $\langle \ell, v_1, v_2, \ldots, v_n \rangle$ interchangeably to denote a position. Given a position $\langle \ell, \ v_1, \ v_2, \ldots, \ v_n \rangle$, we say that the position $\langle \ell, \ v_1', \ v_2', \ldots, \ v_n' \rangle$ is a *time successor* of $\langle \ell, v_1, \ v_2, \ldots, \ v_n \rangle$ if for each i, $v_i' = v_i + \delta$, where δ is a non-negative real number, and for all $0 \leq \delta' \leq \delta$, $\mathcal{R}, \langle v_1 + \delta', v_2 + \delta', \ldots, v_n + \delta' \rangle \models inv(\ell)$. We say that a position $\langle \ell', v_1', v_2', \ldots, v_n' \rangle$ is an *edge successor* of the position $\langle \ell, v_1, v_2, \ldots, v_n \rangle$ if there exists a four-tuple $\langle \ell, \theta, Reset, \ell' \rangle \ \in E$ such that the following three conditions hold.

– $\mathcal{R}, \langle v_1, v_2, \ldots, v_n \rangle \models \theta$.

– $v_i' = \begin{cases} 0, & \text{if } i \in Reset \\ v_i, & \text{otherwise.} \end{cases}$

– $\mathcal{R}, \langle v_1', v_2', \ldots, v_n' \rangle \models inv(\ell')$.

We call an element of E an *edge*. The mapping P maps each location to the set of atomic propositions that are true for that location. As the automaton is non-deterministic, it is not a restriction to suppose that the guards are only conjunctions.

An $\langle \ell, \mathbf{v} \rangle$-*path* ρ is defined as a partial mapping from $\mathcal{N} \times \mathcal{R}$ to the set of positions of the timed automaton such that $\rho(0,0) = \langle \ell, \mathbf{v} \rangle$ and for any $(i, \varepsilon) \in \mathcal{N} \times \mathcal{R}$, $\rho(i, \varepsilon)$ (where i denotes the segment number; see below for a definition of segment) can be "reached" from $\langle \ell, \mathbf{v} \rangle$ through a sequence of time and edge transitions.

A segment of a path is a part of the path between two successive edge transitions. Initially, at the beginning of the path, the control is in the zeroth segment. If the control is in the ith segment, and it takes an instantaneous edge transition, it "enters" the $(i+1)$st segment. A point in a segment is a "snapshot" in that segment. The delay at a point p in a segment i, denoted by $delay(p)$, is the time difference between the current time (the time at that point) and the time the control entered that segment, both the times being measured from the beginning of the path. The delay of a segment is the difference between the time the control leaves that segment and the time the control enters the segment. The time at a point in a segment of a path is the sum of the delays of all the previous segments and the delay at that point in the segment. We write $time_\rho(j, \varepsilon)$ to denote the time at a point in the jth segment of ρ, having the delay ε. Note that $time_\rho(0,0) = 0$. A path maps a pair consisting of a segment number and a delay to a position.

Definition 3.2 *A* trace *of a timed automaton is an infinite sequence of snapshots of an* $\langle \ell, \mathbf{v} \rangle$*-path* ρ *of the automaton of the form given below:*

$$\langle \ell, \mathbf{v} \rangle \longrightarrow \langle \ell^1, \mathbf{v^1} \rangle \longrightarrow \ldots$$

where $\rho(0,0) = \langle \ell, \mathbf{v} \rangle$ *and for each* $i = 1, 2, \ldots,$ *if* $\rho(k, \delta) = \langle \ell^i, v_1', v_2', \ldots, v_n' \rangle$ *and* $\rho(j, \varepsilon) = \langle \ell^{i-1}, v_1'', v_2'', \ldots, v_n'' \rangle$*, then* $(j, \varepsilon) \leq (k, \delta)$ *(in the lexicographic order) and either* $j = k$ *and* $\rho(k, \delta)$ *is a time successor of* $\rho(j, \varepsilon)$ *or* $j + 1 = k$ *and* $\delta = 0$ *and* $\rho(k, \delta)$ *is an edge successor of* $\rho(j, \varepsilon)$*.*

In the above definition, we identify ℓ^0 with ℓ. We call $\langle \ell, \mathbf{v} \rangle$ the *starting element* of the trace. Given a path ρ, and a trace T of ρ, we can write T as a sequence of the form $\rho(i_0, 0), \rho(i_1, \varepsilon_1), \ldots,$ where $i_0 = 0$ and for all j either $i_j = i_{j-1}$ or $i_j = i_{j-1} + 1$ and $0 \leq \varepsilon_j \leq delay(i_j)$.

Definition 3.3 *Given an* $\langle \ell, \mathbf{v} \rangle$*-path* ρ*, and a trace* $T = \rho(i_0, 0), \rho(i_1, \varepsilon_1), \ldots$ *of* ρ*, we say that* T *is* divergent*, if the sequence* $(time_\rho(i_0, 0), time_\rho(i_1, \varepsilon_1), \ldots)$ *diverges.*

The timed automaton shown in Figure 3.1 has a non-divergent (or convergent) trace (consider the trace $\langle \ell^0, 0, 0 \rangle \xrightarrow{0.5} \langle \ell^0, 0.5, 0.5 \rangle \xrightarrow{0} \langle \ell^1, 0, 0.5 \rangle \xrightarrow{0} \langle \ell^0, 0, 0 \rangle \xrightarrow{0.25} \langle \ell^0, 0.25, 0.25 \rangle \xrightarrow{0} \langle \ell^1, 0, 0.25 \rangle \ldots$ since the sum $0.5 + 0.25 \ldots$ converges). But for the timed automaton shown in Figure 3.2 , every trace is divergent.

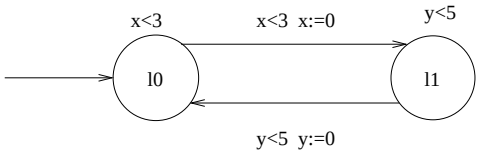


Figure 3.1: An Example Timed Automaton.

3.3 Timed Logic Processes

We identify a fragment of constraint query languages over reals (in the sense of [KKR95, Rev90, BS91]) that will allows us to model real-time systems. Furthermore, as we will see below, it allows us to express the product constructions that come up in the course of model checking for formulas in the temporal logic $\mathcal{L}_s$ [LPY95a]. We call the programs expressed in this fragment as *timed logic processes* (abbreviated TLPs). Before we define TLPs formally, we need the following notations and definitions. Let the constraint γ be defined by the grammar

$$\gamma \ ::= \ true \mid x_i > c \mid x_i < c \mid x_i \geq c \mid x_i \leq c \mid \gamma \wedge \gamma \tag{3.1}$$

where $c \in \mathcal{N}$, the set of natural numbers.

Definition 3.4 (t-clause) *A* t-clause *is a clause of one of the following four forms.*

$$\begin{aligned}
&(1)\ p(\mathbf{x}) \longleftarrow \varphi, p'(\mathbf{x}') \\
&(2)\ p(\mathbf{x}) \longleftarrow p_1(\mathbf{x}), p_2(\mathbf{x}) \\
&(3)\ p(\mathbf{x}) \longleftarrow \gamma \\
&(4)\ \mathit{init} \longleftarrow p(\mathbf{x}), \mathbf{x} = 0
\end{aligned}$$

where the constraints φ *are of the forms (here* n *is the length of the tuple* $\mathbf{x} \equiv \langle x_1, \ldots, x_n \rangle$ *of variables)*

$$\begin{aligned}
&(1.1)\ \varphi \equiv \gamma_1(\mathbf{x}) \wedge \textstyle\bigwedge_{i=1}^{n} x_i' = x_i + z \wedge z \geq 0 \wedge \gamma_2(\mathbf{x}') && \text{(“time transitions”)} \\
&(1.2)\ \varphi \equiv \gamma_1(\mathbf{x}) \wedge \textstyle\bigwedge_{i \in S} x_i' = 0 \wedge \bigwedge_{i \notin S} x_i' = x_i \wedge \gamma_2(\mathbf{x}') && \text{(“edge transitions”)}
\end{aligned}$$

where $S \subseteq \{1, \ldots, n\}$ *and the constraints* γ *are of the form defined in the grammar (3.1).*

We call the constraints γ the *guards* of the clauses. In the sequel, we call a clause of the form (1) as an *evolution clause* if the constraint φ is of the form (1.1) and as *system clause* if the constraint φ is of the form (1.2). We will also call clauses of the form (2) as *alternating clauses*, clause of the form (3) (which are facts or generalized tuples) as assertions and clauses of the form (4) as *initial clauses*.

Definition 3.5 (TLP) *An (unlabeled) TLP is a (finite) set of* t*-clauses in which at least one clause is an initial clause.*

We associate a logical formula corresponding to a TLP in the same way as in [JM94]. Note that the clauses, in which the constraint φ in the body is of the form (1.1), contain the variable

z in the body. The existentially quantified (in the logical formula associated to a TLP) variables z are called *increment variables*. In Section 3.13, we will expand TLPs with alphabets (called labeled TLPs). We now define the notion of convergent and divergent ground derivations of TLPs.

Definition 3.6 (Convergent and Divergent Ground Derivations) *Let $\mathcal{G}$ be an (infinite) ground derivation of $\mathcal{P}$ through clauses of the form* (1) *and* (4). *Let $C_1, C_2, \ldots$ be the clauses involved in $\mathcal{G}$. Let $z_1, z_2, \ldots$ be the sequence of values of the increment variables when clauses $C_1, C_2, \ldots$ are applied respectively (for a system clause or an initial clause, we assume this value to be zero). We say that an (infinite) ground derivation $\mathcal{G}$ of $\mathcal{P}$ is divergent iff the sum $\sum_i z_i$ of the increment variables in the derivation diverges. Otherwise, we say that it is* convergent.

A first motivation for TLPs is that this model subsumes the timed automata [AD94] model; i.e., we can translate timed automata to timed logic processes. These translations use only evolution clauses (clauses obtained by translating time transitions), system clauses (clauses obtained by translating edge transitions) and an initial clause (a clause specifying an initial position). Of the other types of clauses, clauses of the form (2) (i.e., alternating clauses) are used for expressing alternation (compare [DW99]). These clauses are also used to express the product constructions that come up in the course of model checking. Clauses of the form (3) (i.e., assertions) are used to rewrite an agent to a nil agent (in this respect there is a similarity with process algebras; thus $p(\mathbf{x}) \longleftarrow \gamma$ states that the agent p can rewrite to the nil agent if the values of the variables $\mathbf{x}$ satisfy the formula γ). These clauses can also be used to express assertions about processes (e.g., by rewriting an agent to the nil agent if the values of the variables $\mathbf{x}$ violate a safety property). We will see later that clauses of the form (3) can also be used for expressing $\mathcal{L}_s$ properties. Thus the TLP framework not only allows modeling a system, but also allows writing assertions about the behaviors of the system.

3.4 Translation of Timed Automata into TLPs

Since the timed automaton model is predominantly used in the literature, we show the connection of the timed logic process model with the timed automaton model. In other words, we show that timed automata can be translated to TLPs. The construction of a TLP from a timed automaton is given below.

Construction 3.1 Let

$$U = \langle AP, X_n, L, E, P, \ell^0, inv \rangle$$

be a timed automaton [AD94] with n clocks, where AP is a set of atomic propositions, X_n is a set of clocks (n clocks $x_1, \ldots, x_n$), L is a set of locations, E is a set of edges, P is a labeling function that labels each location with a set of atomic propositions, $\ell^0 \in L$ is the initial location and inv is a function that assigns to each location an invariant constraint. We translate U to a TLP $\mathcal{P}$ as follows. For each location $\ell \in L$, we introduce an n-ary predicate $\ell(\mathbf{x})$. For each location $\ell \in L$, we have an evolution clause where γ_1 and γ_2 are both the invariant of the location ℓ (i.e., $\gamma_2(\mathbf{x}')$ is obtained from $\gamma_1(\mathbf{x})$ by renaming all variables in the tuple $\mathbf{x}$ by their primed versions in the tuple $\mathbf{x}'$). Thus the evolution clause takes the form

$$\ell(\mathbf{x}) \longleftarrow \ell(\mathbf{x}') \wedge \varphi$$

where

$$\varphi \equiv inv_\ell(\mathbf{x}) \wedge \bigwedge_{i=1}^{n} x_i' = x_i + z \wedge z \geq 0 \wedge inv_\ell(\mathbf{x}')$$

(inv_{ell} is the invariant of the location ℓ). For each edge $\langle \ell, \theta, Reset, \ell' \rangle \in E$ from ℓ to ℓ', where θ is the guard of the edge and $Reset$ is the set of clocks reset in that edge, we have a clause of the form (1.2) with head predicate $\ell(\mathbf{x})$ and body predicate $\ell'(\mathbf{x})$, where $\gamma_1 \equiv \theta \wedge inv_\ell(\mathbf{x})$, $\gamma_2 \equiv inv_{\ell'}$ and $S = Reset$ (here inv_ℓ and $inv_{\ell'}$ are respectively the invariants of locations ℓ and ℓ'). Thus the system clause takes the form

$$\ell(\mathbf{x}) \longleftarrow \ell'(\mathbf{x}') \wedge \varphi$$

where

$$\varphi \equiv inv_\ell(\mathbf{x}) \wedge \bigwedge_{i \in Reset} x_i' = 0 \wedge \bigwedge_{i \notin Reset} x_i' = x_i \wedge inv_{\ell'}(\mathbf{x}').$$

We also add an initial clause $init \longleftarrow \ell^0(\mathbf{x}) \wedge \mathbf{x} = 0$. The labeling function P is extended to the predicates in the canonical way.

‖

The semantics of a timed automaton are given in terms of traces. The semantics of a TLP are given in terms of ground derivations. Identifying positions and ground atoms, we get the following.

Theorem 3.1 (Meaning of translation) *For every timed automaton U there exists a TLP $\mathcal{P}$ such that the set of ground derivations of $\mathcal{P}$ correspond exactly to the set of traces of U. In other words, for every timed automaton U there exists a TLP $\mathcal{P}$ such that U and $\mathcal{P}$ have the same semantics.*

Proof. The construction of the TLP $\mathcal{P}$ from a timed automaton U is given above. Consider any trace T of U.

$$T = \langle \ell^1, \mathbf{v}^1 \rangle \longrightarrow \langle \ell^2, \mathbf{v}^2 \rangle \longrightarrow \ldots .$$

We show by induction that $\ell^1(\mathbf{v}^1) \longrightarrow \ell^2(\mathbf{v}^2) \longrightarrow \ldots$ is a ground derivation of $\mathcal{P}$, i.e., we show by induction on i that for each i, $\ell^{i+1}(\mathbf{v}^{i+1})$ is a ground resolvent of $\ell^i(\mathbf{v}^i)$ through a clause in $\mathcal{P}$. The base case for $i = 0$ is trivial. Suppose that the result holds for all $i \leq k$. By the definition of traces, either $\langle \ell^{k+2}, \mathbf{v}^{k+2} \rangle$ is an edge successor of $\langle \ell^{k+1}, \mathbf{v}^{k+1} \rangle$ or $\langle \ell^{k+2}, \mathbf{v}^{k+2} \rangle$ is a time successor of $\langle \ell^{k+1}, \mathbf{v}^{k+1} \rangle$. In either case, by the construction of $\mathcal{P}$, there exists a clause in $\mathcal{P}$ such that $\ell^{k+2}(\mathbf{v}^{k+2})$ is a ground resolvent of $\ell^{k+1}(\mathbf{v}^{k+1})$ through that clause. By a similar induction, it can be proved that every ground derivation of $\mathcal{P}$ corresponds to a trace of U. ‖

In our definition, the semantics of a timed automaton also contain *convergent* traces. Unbounded liveness properties, however, refer only to *divergent* traces.

3.5 Logic of Safety and Bounded Liveness ($\mathcal{L}_s$)

The syntax of formulas Φ in the logic $\mathcal{L}_s$ (*Logic of safety and bounded liveness*) [LPY95a] is given as follows:

$$\Phi \ ::= \ \theta \mid q \mid q \vee \Phi \mid \theta \vee \Phi \mid \Phi_1 \wedge \Phi_2 \mid \Box\Phi \mid \forall\Phi \mid x.\Phi \mid Z$$

where θ is an atomic constraint of the form $x_i \sim c$, where $\sim \in \{=, <, >, \geq, \leq\}$, c is a natural number, q is an atomic proposition and $Z \in Id$ is an identifier (identifiers are "mu-calculus" variables). We call a variable that does not occur on the right hand side of any declaration in an $\mathcal{L}_s$ formula the root variable for that formula. An $\mathcal{L}_s$ formula is a set of declarations having a root variable. The meaning of the identifiers (or variables) Z is specified by a unique declaration $\mathcal{D}(Z) : Z = \Phi$ for each identifier assigning a formula Φ of $\mathcal{L}_s$ to that identifier Z. It can be easily shown using the techniques in [Wal93] that an $\mathcal{L}_s$ formula can be rewritten in linear time in a simple form in which each declaration is of the form $X = q \vee X'$ or $X = \theta \vee X'$ or $X = X' \wedge X''$ or $X = \Box X'$ or $X = \forall X'$ or $X = x.X'$ where X', X'' are either identifiers or atomic propositions or atomic constraints. In the following, we will always assume that $\mathcal{L}_s$ formula is given in a simple form.

The satisfaction relation $\models$ for $\mathcal{L}_s$ is the largest relation satisfying the following (where $\mathcal{P}$ is a TLP, $p(\mathbf{v})$ is a ground atom in the $\mathcal{R}$-base of $\mathcal{P}$; here $\mathcal{R}$ denotes the reals):

- $\mathcal{P}, p(\mathbf{v}) \models \theta$ implies $\mathcal{R}, \mathbf{v} \models \theta$.
- $\mathcal{P}, p(\mathbf{v}) \models q$ implies $q \in P(p)$ (where P is a function that assigns to each predicate in $\mathcal{P}$ a set of atomic propositions).
- $\mathcal{P}, p(\mathbf{v}) \models q \vee \Phi$ implies $\mathcal{P}, p(\mathbf{v}) \models q$ or $\mathcal{P}, p(\mathbf{v}) \models \Phi$.
- $\mathcal{P}, p(\mathbf{v}) \models \theta \vee \Phi$ implies $\mathcal{P}, p(\mathbf{v}) \models \theta$ or $\mathcal{P}, p(\mathbf{v}) \models \Phi$.
- $\mathcal{P}, p(\mathbf{v}) \models \Phi_1 \wedge \Phi_2$ implies $\mathcal{P}, p(\mathbf{v}) \models \Phi_1$ and $\mathcal{P}, p(\mathbf{v}) \models \Phi_2$.
- $\mathcal{P}, p(\mathbf{v}) \models \Box\Phi$ implies for all ground resolvents $p'(\mathbf{v}')$ of $p(\mathbf{v})$ through system clauses or initial clauses, $\mathcal{P}, p'(\mathbf{v}') \models \Phi$.
- $\mathcal{P}, p(\mathbf{v}) \models \forall\Phi$ implies for all ground resolvents $p'(\mathbf{v}')$ of $p(\mathbf{v})$ through evolution clauses, $\mathcal{P}, p'(\mathbf{v}') \models \Phi$.
- $\mathcal{P}, p(\mathbf{v}) \models x.\Phi$ implies $\mathcal{P}, p(\mathbf{v})[0/x] \models \Phi$ (where the ground atom $p(\mathbf{v})[0/x]$ is obtained from $p(\mathbf{v})$ by reseting the variable x to zero).
- $\mathcal{P}, p(\mathbf{v}) \models Z$ implies $\mathcal{P}, p(\mathbf{v}) \models \mathcal{D}(Z)$.
- $\mathcal{P}, p_1(\mathbf{v_1}) \wedge \ldots \wedge p_m(\mathbf{v_m}) \models \Phi$ implies $\mathcal{P}, p_1(\mathbf{v_1}) \models \Phi, \ldots, \mathcal{P}, p_m(\mathbf{v_m}) \models \Phi$ (satisfiability for goals).

It is to be noted that the logic $\mathcal{L}_s$ [LPY95b] was originally introduced for timed automata and hence does not take into account the alternating clauses and assertions of TLPs. Note that we take the greatest fixpoint of the set of declarations (viewed as a set of equations). For a TLP $\mathcal{P}$ and an $\mathcal{L}_s$ formula Φ, we say that $\mathcal{P} \models \Phi$ iff $\mathcal{P}, init \models \Phi$.

An example of a bounded liveness specification in $\mathcal{L}_s$ is as follows: let $\mathcal{C}$ be an atomic constraint. Then the formula $X = \Box(z.Z)$ where $Z = \mathcal{C} \vee (z < i \wedge \forall Z \wedge \Box Z)$ asserts that $\mathcal{C}$ should be satisfied within i time units of resolving through a system clause (for timed automata,

this amounts to the statement that $\mathcal{C}$ should be satisfied within i time units of taking an edge transition). We call the variables $\mathbf{x}$ the *real variables*.

In order to specify properties about TLPs and in order to facilitate the product construction described below, it is useful to consider the dual of the logic $\mathcal{L}_s$. So before introducing the model checking method, we first introduce the syntax of $\widetilde{\mathcal{L}_s}$ which expresses the dual of $\mathcal{L}_s$ formulas. The syntax of $\widetilde{\mathcal{L}_s}$ is given as follows:

$$\widetilde{\Phi} \ ::= \ \theta \mid q \mid q \wedge \widetilde{\Phi} \mid \theta \wedge \widetilde{\Phi} \mid \widetilde{\Phi_1} \vee \widetilde{\Phi_2} \mid \Diamond\widetilde{\Phi} \mid \exists\widetilde{\Phi} \mid x.\widetilde{\Phi} \mid \widetilde{Z}$$

where θ is an atomic constraint and q is an atomic proposition. An $\widetilde{\mathcal{L}_s}$ formula is a set of declarations with a root variable. Note that we take the least fixpoint of the set of declarations (viewed as a set of equations). For every formula Φ of $\mathcal{L}_s$, we can define a formula $\widetilde{\Phi}$ in $\widetilde{\mathcal{L}_s}$ such that for a TLP $\mathcal{P}$, $\mathcal{P} \models \widetilde{\Phi}$ iff $\mathcal{P} \not\models \Phi$. We do not provide the semantics of $\widetilde{\mathcal{L}_s}$ formulas which are easily understood from those of $\mathcal{L}_s$ formulas (dual of those of $\mathcal{L}_s$ formulas).

3.6 Product Program

In this section, we formulate the basis of our model checking methodology— a product construction of TLPs with logical formulas. Given a TLP $\mathcal{P}$, and an $\widetilde{\mathcal{L}_s}$ formula $\widetilde{\Phi}$, we construct the product TLP $\mathcal{P}^{\widetilde{\Phi}}$, in which the arity of each predicate is n (assuming that the arity of each predicate in $\mathcal{P}$ is k and the corresponding $\mathcal{L}_s$ formula Φ has $n-k$ real variables), such that $\mathcal{P} \models \widetilde{\Phi}$ iff the (new) predicate $\langle init, \widetilde{Z}\rangle$ (see below) is in the least model of $\mathcal{P}^{\widetilde{\Phi}}$. Here $\widetilde{Z}$ is the root variable of $\widetilde{\Phi}$. The construction is as follows.

Construction 3.2 For the root variable $\widetilde{Z}$ we create the (0-ary predicate) $\langle init, \widetilde{Z}\rangle$. For each predicate $\langle p, X\rangle$ created, expand (i.e., create a rule(s) defining that predicate; these rules depend on the declaration $X = \Psi$ defining X in $\widetilde{\Phi}$) using the following rules if the predicate is not already expanded:

- $X = q$: $\langle p, X\rangle(\mathbf{x}) \longleftarrow \mathit{true}$ if $q \in P(p)$ (where P is a function assigning to each predicate symbol a set of atomic propositions and p is a predicate symbol in $\mathcal{P}$).
- $X = \theta$: $\langle p, X\rangle(\mathbf{x}) \longleftarrow \theta$.
- $X = q \wedge X'$: $\langle p, X\rangle(\mathbf{x}) \longleftarrow \langle p, X'\rangle(\mathbf{x})$ if $q \in P(p)$.
- $X = \theta \wedge X'$: $\langle p, X\rangle(\mathbf{x}) \longleftarrow \langle p, X'\rangle(\mathbf{x}) \wedge \theta$.
- $X = X_1 \vee X_2$: $\langle p, X\rangle(\mathbf{x}) \longleftarrow \langle p, X_1\rangle(\mathbf{x})$ and $\langle p, X\rangle(\mathbf{x}) \longleftarrow \langle p, X_2\rangle(\mathbf{x})$.
- $X = \Diamond X'$: For each system clause C in $\mathcal{P}$ such that the predicate p stands on the head of the clause create a clause of the form $\langle p, X\rangle(\mathbf{x}) \longleftarrow \langle p', X'\rangle(\mathbf{x}') \wedge \varphi \wedge \varphi'$ where φ is the constraint in the body of the clause C and $\varphi' \equiv \bigwedge_{i=k+1}^{n} x_i' = x_i$.
- $X = \exists X'$: For each evolution clause C such that the predicate p stands on its head, create a clause of the form $\langle p, X\rangle(\mathbf{x}) \longleftarrow \langle p', X'\rangle(\mathbf{x}') \wedge \varphi \wedge \varphi'$, where φ is the constraint in C and φ' is given by $\bigwedge_{i=k+1}^{n} x_i' = x_i + z$.
- $X = x_i.X'$: $\langle p, X\rangle(\mathbf{x}) \longleftarrow \langle p, X'\rangle(\mathbf{x}') \wedge x_i = 0 \wedge \bigwedge_{j \neq i} x_j' = x_j$.

Example 3.1 Consider the TLP corresponding to the timed automaton in Figure 3.2 and the $\mathcal{L}_s$ formula $Z = \Box X$ where $X = \overline{at\_l2} \wedge \Box X \wedge \forall X$ where $\overline{at\_l2}$ is an atomic proposition satisfied at all locations but $l2$. The product program corresponding to (the dual of) this formula (i.e., the formula $\widetilde{X} = at\_l2 \vee \Diamond X \vee \forall X$ where at_l2 is an atomic proposition satisfied only at the location $l2$) is given in Figure 3.3.

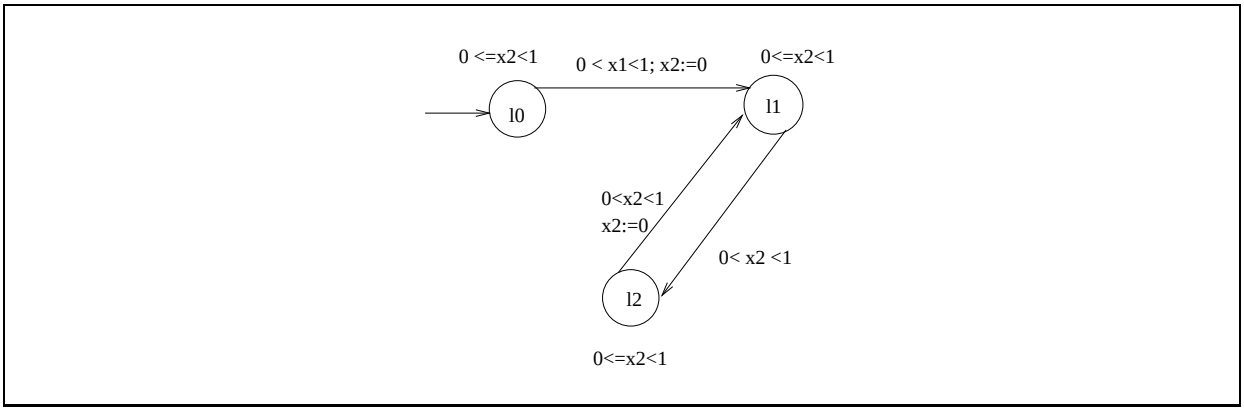


Figure 3.2: Example illustrating that the model checking procedure is possibly non-terminating.

Theorem 3.2 *Given a TLP $\mathcal{P}$ and an $\mathcal{L}_s$ formula Φ, $\mathcal{P} \models \Phi$ if and only if the atom $\langle init, \widetilde{Z}\rangle$ is not in the least model of $\mathcal{P}^{\widetilde{\Phi}}$ where $\widetilde{\Phi}$ is the dual of the $\mathcal{L}_s$ formula Φ and $\widetilde{Z}$ is the root variable of $\widetilde{\Phi}$.*

Proof. By structural induction on $\mathcal{L}_s$ formulas. We show that for any predicate symbol p and a tuple $\mathbf{v} \in \mathcal{R}^n$, $\mathcal{P}, p(\mathbf{v}) \models \Phi$ iff $\langle p, \widetilde{Z}\rangle(\mathbf{v})$ is not in the least model of $\mathcal{P}^{\widetilde{\Phi}}$.

Base Cases: For atomic propositions and atomic constraints, the proof is trivial.

Induction Step:

Most of the cases are easy. We show only a few typical ones.

Case: $Z = q \vee X$. Suppose that $\mathcal{P}, p(\mathbf{v}) \models q \vee X$. If $\mathcal{P}, p(\mathbf{v}) \models q$, no clause that defines the predicate $\langle p, \widetilde{Z}\rangle$ is created and hence $\langle p, \widetilde{Z}\rangle$ is not in the least model of $\mathcal{P}^{\widetilde{\Phi}}$. If $\mathcal{P}, p(\mathbf{v}) \models X$, then the result follows from the induction hypothesis.

$$
\begin{array}{lcl}
\langle init, \widetilde{Z}\rangle & \longleftarrow & \langle l0, X\rangle(\mathbf{x}) \wedge \mathbf{x} = 0 \\
\langle l0, X\rangle(\mathbf{x}) & \longleftarrow & \langle l0, X\rangle(\mathbf{x}') \wedge \mathbf{x}' = \mathbf{x} + z \wedge 0 \leq x2 < 1 \wedge z \geq 0. \\
\langle l0, X\rangle(\mathbf{x}) & \longleftarrow & \langle l1, X\rangle(\mathbf{x}') \wedge 0 < x1 < 1 \wedge x2' = 0 \wedge x1' = x1. \\
\langle l1, X\rangle(\mathbf{x}) & \longleftarrow & \langle l1, X\rangle(\mathbf{x}') \wedge \mathbf{x}' = \mathbf{x} + z \wedge z \geq 0 \wedge 0 \leq x2 < 1. \\
\langle l1, X\rangle(\mathbf{x}) & \longleftarrow & \langle l2, X\rangle(\mathbf{x}') \wedge \mathbf{x}' = \mathbf{x} \wedge 0 < x2 < 1. \\
\langle l2, X\rangle(\mathbf{x}) & \longleftarrow & true. \\
\langle l2, X\rangle(\mathbf{x}) & \longleftarrow & \langle l2, X\rangle(\mathbf{x}') \wedge \mathbf{x}' = \mathbf{x} + z \wedge z \geq 0 \wedge 0 \leq x2 < 1. \\
\langle l2, X\rangle(\mathbf{x}) & \longleftarrow & \langle l1, X\rangle(\mathbf{x}') \wedge x2' = 0 \wedge x1' = x1 \wedge 0 < x2 < 1.
\end{array}
$$

Figure 3.3: Product Program corresponding to Figure 3.2.

Case: $Z = X_1 \wedge X_2$. The product program in this case consists of the clauses $\langle p, \widetilde{Z}\rangle(\mathbf{x}) \longleftarrow \langle p, \widetilde{X_1}\rangle(\mathbf{x})$ and $\langle p, \widetilde{Z}\rangle(\mathbf{x}) \longleftarrow \langle p, \widetilde{X_2}\rangle(\mathbf{x})$ along with the product programs for the formulas defined by the declarations defining $\widetilde{X_1}$ and $\widetilde{X_2}$. The result then follows from the induction hypothesis.

Case: $Z = \Box X$. If $\mathcal{P}, p(\mathbf{v}) \models Z$ then for all successors $p'(\mathbf{v}')$ of $p(\mathbf{v})$ through system clauses, $\mathcal{P}, p'(\mathbf{v}') \models X$. The result now follows from the induction hypothesis.

The rest of the cases follow directly from the induction hypothesis.

The proof for the other direction follows by similar induction on $\mathcal{L}_s$ formulas. ∎

Methodology To prove $\mathcal{P} \models \Phi$, we try to prove $\mathcal{P} \not\models \widetilde{\Phi}$ where $\widetilde{\Phi}$ is the $\widetilde{\mathcal{L}_s}$ formula corresponding to Φ (i.e., the dual of Φ). This is proved by proving that $\langle init, \widetilde{Z}\rangle$ is not in the least model of $\mathcal{P}^{\widetilde{\Phi}}$ (where Z is the root variable of Φ). We can either compute the least model of $\mathcal{P}^{\widetilde{\Phi}}$ using the least fix point of the immediate consequence operator resulting in a global model checker. We will prefer top-down evaluation (backward chaining) of TLPs in contrast with the bottom-up evaluation (forward chaining) advocated in [KKR95]; top-down evaluation has the advantage that it can be goal directed, i.e., local; partial order reduction techniques can be easily incorporated into it; see [HKQ98] for a discussion of top-down vs. bottom-up evaluation. Hence we extend XSB-style tabling [CW96, TS86a, Vie87] with constraints to prove that $\langle init, \widetilde{Z}\rangle$ does not succeed in the tabled resolution using the non-ground transition system. To be precise, our method extends with constraints the OLDT resolution of [TS86a]. Extending standard results from logic programming [TS86a] we get, the state $\langle init, \widetilde{Z}\rangle$ succeeds iff it succeeds in the derivation tree obtained by using tabled resolution. Note that the tabling strategy produces a local model checker for $\widetilde{\mathcal{L}_s}$ ($\mathcal{L}_s$). To guarantee termination of the model checking procedure, we can use the trim operation on constraints, described below, along with the tabling strategy mentioned above.

Providing a Counter Example To provide a counter example, we follow the following method. With each non-ground goal we keep the following information: the constraints encountered so far (including the mgus, i.e., most general unifiers, which are also regarded as constraints), a list of the numbers of clauses encountered so far (we assume that the clauses are numbered) and a list of increment variables encountered so far (assuming that they are suitably renamed). Thus a non-ground goal will take the form of a five-tuple $\langle Q, \varphi_1, \varphi_2, L_1, L_2\rangle$ where Q is a conjunction of predicates, φ_1 is the constraint store, φ_2 is the concatenation of all the constraints of all the clauses (and the mgus) encountered thus far, L_1 is a list of the numbers of the clauses encountered so far, L_2 is the list of the increment variables of the clauses encountered so far. Now the "earliest" (with respect to time) ground counter example (i.e., a ground derivation acting as a witness to the success) can be provided in the following way. First project φ_2 on the set of variables in the list L_2. Let the constraint obtained be φ. Now minimize $\Sigma_{z_i \in L_2} z_i$ with respect to φ. The solutions of z_i obtained in this method can be used in providing a ground counter example. The counter example can now be generated from the sequence of clauses and the values of the corresponding increment variables.

3.7 The Trim Operation on Constraints

We first start with the observation the model checking procedure described above is possibly non-terminating. The counter example is provided by the translation to TLP of the timed

automaton in Figure 3.2. Before understanding, why the above procedure does not terminate for the example in Figure 3.2, we need a few definitions.

Definition 3.7 (Reachable Nonground and Ground States) *A non-ground state $\langle p(\mathbf{x}), \varphi \rangle$ is said to be* reachable *in the non-ground transition system induced by $\mathcal{P}$ iff there exists a (finite or infinite) non-ground derivation using the clauses in $\mathcal{P}$:*

$$init \longrightarrow \ldots \longrightarrow \langle \widetilde{Q}, \varphi' \rangle \longrightarrow \ldots$$

such that $p(\mathbf{x}')$ is one of the predicates in $\widetilde{Q}$ and $\langle p(\mathbf{x}), \varphi \rangle$ and $\langle p(\mathbf{x}'), \varphi'' \rangle$ are identical where $\varphi'' \equiv \exists_{-\mathbf{x}'} \varphi'$ (the existential quantifier is over all variables but $\mathbf{x}'$). A ground state $p(\mathbf{v})$ is said to be reachable in the ground transition system induced by $\mathcal{P}$ iff there exists a (finite or infinite) ground derivation of the form:

$$init \longrightarrow \ldots \longrightarrow Q \longrightarrow \ldots$$

where $p(\mathbf{v})$ is one of the conjuncts in Q and init is the initial predicate.

Note that a non-ground state is reachable in the non-ground transition system induced by $\mathcal{P}$ iff all its ground instances are reachable in the ground transition system induced by $\mathcal{P}$.

Proposition 3.1 *There exists a timed automaton such that the non-ground transition system of the TLP corresponding to that automaton has infinitely many reachable non-ground states.*

Proof. Consider the example timed automaton given in Figure 3.2. The timed automaton has three locations $l0$, $l1$ and $l2$. There are two clocks $x1$ and $x2$. The initial position is $\langle l0, 0, 0 \rangle$. There is a transition from $l0$ to $l1$ in which the clock $x2$ is reset and the guard is $0 < x_1 < 1$. There is a transition from $l1$ to $l2$ in which no clock is reset and the guard of the transition is $0 < x_2 < 1$. Also there is a transition from $l2$ to $l1$ in which the guard is $0 < x_2 < 1$ and the clock $x2$ is reset. The invariant for all these three locations is $0 \leq x_2 < 1$. We can easily model this timed automaton by an TLP $\mathcal{P}_2$ having clauses as described in the previous section. It can be easily seen that an infinitely many reachable nonground states of the form $\langle l2(\mathbf{x}), x1 - x2 \geq 0 \wedge x2 - x1 \geq -(i+1) \wedge x2 > 0 \wedge x2 < 1 \rangle$ (where $i \in \mathcal{N}$) is generated. ∎

We next start with a few definitions.

Definition 3.8 (Zones) *A* zone *is a conjunction of constraints, each of which puts a lower or upper bound on a variable or on the difference of two variables. A C-zone is a non-ground state of the form $\langle p, \varphi \rangle$ where, p is a predicate symbol and φ is a formula generated by the following grammar:*

$$\begin{aligned} \varphi \quad ::= \quad & x_i < a \mid x_i > a \mid x_i \leq a \mid x_i \geq a \mid x_i - x_j > a \\ & \mid x_i - x_j \geq a \mid x_i - x_j \leq a \mid x_i - x_j < a \mid \varphi_1 \wedge \varphi_2 \end{aligned} \tag{3.2}$$

We call the above constraints as zone constraints. *The free variables of a zone constraint are among $\{x_1, x_2, \ldots, x_n\}$.*

Definition 3.9 *A non-ground state $s = \langle p(\mathbf{x}), \varphi \rangle$ corresponds to a C-zone $\xi = \langle p, \varphi' \rangle$ if φ is equivalent to φ' with respect to $\{x_1, \ldots, x_n\}$, i.e., the set of ground instances of s is the the set $\{p(\mathbf{v}) \mid \mathcal{R}, \mathbf{v} \models \varphi'\}$.*

The following lemma holds:

Lemma 3.1 *Each reachable non-ground state corresponds to a C-zone, i.e., for each reachable non-ground state $\langle p(\mathbf{x}), \varphi \rangle$, there exists a C-zone $\xi = \langle p, \varphi' \rangle$ such that φ' is equivalent to φ with respect to $\{x_1, x_2, \ldots, x_n\}$.*

Proof. By using Fourier's Algorithm [MS98, LM92] and induction on the length of non-ground derivation. ∎

Our aim is to define an equivalence relation $\approx_M$ on the set of non-ground states of $\mathcal{P}$ (i.e., states of the form $\langle p(\mathbf{x}), \varphi \rangle$ where p is a predicate symbol and φ is the constraint store in which the free variables are $\mathbf{x}$) such that the following conditions hold:

- The quotient (in the standard sense) of the non-ground transition system of $\mathcal{P}$, induced by $\approx_M$, denoted by $\mathcal{P}/\approx_M$, has a finite index, i.e., a finite number of "states" or equivalence classes.
- The transition system induced by $\mathcal{P}/\approx_M$ (in the standard sense) bisimulates [Mil89] the non-ground transition system induced by $\mathcal{P}$.

The suffix M denotes the maximal constant occurring in the guards of the TLP $\mathcal{P}$ (this suffix is kept since the equivalence relation $\approx_M$ involves M). Before we go into the details of this equivalence relation, we start with a few definitions. Below, we identify two nonground states $\langle p(\mathbf{x}), \varphi \rangle$ and $\langle p(\mathbf{x}), \varphi' \rangle$ iff they have the same ground instances. The justification for this is that the successor relation in nonground transition systems of constraint query language programs depends only on the logical contents of the non-ground states.

Definition 3.10 (Reachable Modulo M States) *The set of ground states R_M reachable modulo M, in $\mathcal{P}$ is defined as the smallest set containing the reachable ground states which is closed under the following:*

- *if there exists a ground state $p(\mathbf{v}') \in R_M$ and for all $i \in \{1, \ldots, n\}$ either $v_i = v_i'$ or $(v_i > M \wedge v_i' > M)$ then $p(\mathbf{v}) \in R_M$.*

A non-ground state $\langle p(\mathbf{x}), \varphi \rangle$ is said to be reachable modulo M iff all its ground instances are reachable modulo M and it is identical to a state $\langle p(\mathbf{x}), \varphi' \rangle$, where φ' is given by zone constraints.

Let $\mathcal{P}$ be a TLP with M as the maximal constant occurring in the guards of the clauses. Let s be any non-ground state. Let $sol(s)$ denote the set of ground instances of s. Now we define an equivalence relation $\approx_M$ on the set of non-ground states of $\mathcal{P}$ as follows: $\approx_M$ is the smallest equivalence relation satisfying the following:

- $\langle p(\mathbf{x}), \varphi \rangle \approx_M \langle p(\mathbf{x}), \varphi' \rangle$ if for all $p(\mathbf{v}) \in sol(\langle p(\mathbf{x}), \varphi \rangle)$ there exists $p(\mathbf{w}) \in sol(\langle p(\mathbf{x}), \varphi' \rangle)$ such that $\forall i \in \{1, \ldots, n\}$ either $(v_i = w_i)$ or $(v_i > M \wedge w_i > M)$ and vice versa.

From now on, we view the non-ground transition system induced by $\mathcal{P}$ as a labeled transition system in which the clauses act as labels. We say that two nonground goals $\langle Q, \varphi \rangle$ and $\langle Q, \varphi' \rangle$ (where Q is a conjunction of predicates) are $\approx_M$-equivalent, denoted by $\langle Q, \varphi \rangle \approx_M \langle Q, \varphi' \rangle$, if for each predicate $p(\mathbf{x})$ in Q, $\langle p(\mathbf{x}), \exists_{-\mathbf{x}} \varphi \rangle \approx_M \langle p(\mathbf{x}), \exists_{-\mathbf{x}} \varphi' \rangle$. The equivalence relation $\approx_M$ can be viewed as a symbolic bisimulation relation.

Proposition 3.2 *The non-ground transition system of $\mathcal{P}$ and the quotient of the non-ground transition system of $\mathcal{P}$ induced by $\approx_M$ are bisimilar.*

Proof. All we need to prove is that if $\langle p(\mathbf{x}), \varphi\rangle \approx_M \langle p(\mathbf{x}), \varphi'\rangle$ and if one of them resolves through a clause C in $\mathcal{P}$, then the other also resolves through the same clause and the two resolvents lie in the same equivalence class induced by $\approx_M$. For this, first, given the maximal constant M occurring in the guards of the clauses of $\mathcal{P}$ we define an equivalence relation $\sim_M$ on $\mathcal{R}^n$ as follows: $\mathbf{v} \sim_M \mathbf{v}'$ iff for all i either $v_i = v'_i$ or both $v_i > M$ and $v'_i > M$. Now we observe that if $\mathbf{v} \sim_M \mathbf{v}'$ then for any atomic constraint θ, either they both satisfy θ or they both do not satisfy θ.

Now suppose $\langle p(\mathbf{x}), \varphi\rangle \approx_M \langle p(\mathbf{x}), \varphi'\rangle$. Let $\langle p(\mathbf{x}), \varphi\rangle$ resolve through a clause C of $\mathcal{P}$. If C is an alternating clause then $\langle p(\mathbf{x}), \varphi'\rangle$ also resolves through it and the resolvents in both cases are equivalent through $\approx_M$. Similarly, for initial clauses. Now we consider clauses of the form (1), i.e., system clauses and evolution clauses. Let C be an evolution clause. Let the constraint in C be ψ. Suppose that $\langle p(\mathbf{x}), \varphi\rangle$ resolves through C. Let the resolvent be $\langle p'(\mathbf{x}), \varphi''\rangle$. Now $\varphi'' \equiv \exists_{-\mathbf{x}'}\varphi \wedge \psi$. Since φ'' is satisfiable, there exists $\mathbf{v}$ that satisfies φ''. So $\mathcal{R}, \mathbf{v} \models \exists_{-\mathbf{x}'}\varphi \wedge \psi$. Hence there exists $\mathbf{w}$ and a real number δ (the value of the increment variable z) such that $\mathcal{R}, \mathbf{w} \models \varphi$, $\mathcal{R}, \mathbf{w} \models \gamma(\mathbf{x})$ and for all i, $v_i = w_i + \delta$. Since, $\langle p(\mathbf{x}), \varphi\rangle \approx_M \langle p(\mathbf{x}), \varphi'\rangle$, there exists $\mathbf{u}$ such that $\mathcal{R}, \mathbf{u} \models \varphi'$ and $\mathbf{u} \sim_M \mathbf{w}$. Therefore, $\mathcal{R}, \mathbf{u} \models \gamma(\mathbf{x})$. Let $\mathbf{v}' = \mathbf{u} + \delta$. Observe that $\mathbf{v}' \sim_M \mathbf{v}$. Also observe that $\mathcal{R}, \mathbf{u}, \mathbf{v} \models \varphi' \wedge \psi$. Hence, $\langle p(\mathbf{x}), \varphi'\rangle$ can resolve through C. Let the resolvent be $\langle p'(\mathbf{x}), \varphi'''\rangle$ where $\varphi''' \equiv \exists_{-\mathbf{x}'}\varphi' \wedge \psi$. Then $\mathcal{R}, \mathbf{v}' \models \varphi'''$. Thus for any $\mathcal{R}, \mathbf{v} \models \varphi''$, we can find a $\mathbf{v}'$ such that $\mathcal{R}, \mathbf{v}' \models \varphi'''$ and $\mathbf{v} \sim_M \mathbf{v}'$. Similarly, it can be proved that for any $\mathbf{v}'$ such that $\mathcal{R}, \mathbf{v}' \models \varphi'''$, we can find a $\mathbf{v}$ such that $\mathcal{R}, \mathbf{v} \models \varphi''$ and $\mathbf{v} \sim_M \mathbf{v}'$. Therefore, $\langle p'(\mathbf{x}), \varphi''\rangle \approx_M \langle p'(\mathbf{x}), \varphi'''\rangle$. The proof for system clauses is similar. Thus we have proved that if $\langle p(\mathbf{x}), \varphi\rangle$ resolves through a clause C, then $\langle p(\mathbf{x}), \varphi'\rangle$ also resolves through the same clause C such that the two resolvents are $\approx_M$-equivalent. Similarly, the other direction can be proved. The proof for non-ground goals can be developed on similar lines. Hence, the result follows. ‖

While the classical region equivalence [AD94] is defined on the set of positions, the equivalence relation $\approx_M$ is defined on the set of reachable nonground states. As we show below, given two reachable nonground states it is decidable whether they are $\approx_M$ equivalent. In contrast with the classical region equivalence, the connection of the equivalence relation $\approx_M$ with the trim operation established below allows us to design an on-the-fly symbolic model checking algorithm.

Now we show how to decide whether two nonground states are equivalent using the trim operation described below.

Normalization of Constraints Given a reachable nonground state $\langle p(\mathbf{x}), \varphi\rangle$, we convert it to a state $\langle p(\mathbf{x}), \varphi'\rangle$ such that $\langle p(\mathbf{x}), \varphi\rangle = \langle p(\mathbf{x}), \varphi'\rangle$, where φ' is in a normalized form, by the method given in Figure 3.4. We call the resulting constraint φ' the normalized representation of φ. In the normalized form, we allow constraints of the form $x_i \sim c$ or $x_i - x_j \; relop \; a$ where $\sim \in \{>, \geq, <, \leq\}$, $relop \in \{>, \geq\}$, c is a natural number and a is an integer.

Proposition 3.3 *For any reachable nonground state $\langle p(\mathbf{x}), \varphi\rangle$, the normalized form of the constraint φ can be generated by the following grammar:*

$$\varphi \;::=\; x_i < c \mid x_i > c \mid x_i \leq c \mid x_i \;\geq\; c \mid x_i - x_j > a \mid x_i - x_j \geq a \mid \varphi_1 \;\wedge\; \varphi_2 \qquad (3.3)$$

where c is a natural number and a is an integer.

Note that in the normalized representation, we do not allow constraints of the form $x_i - x_j \leq c$.

Note that a reachable nonground state has a unique normalized form. Since linear programming is polynomial time solvable, given a reachable nonground state, it can be converted to the

- For each variable x_i, add the conjunct $\exists_{-x_i}\varphi\ \varphi'$. For each pair of variables x_i, x_j, let $\widetilde{\varphi}$ be the constraint $(\exists_{-z}\varphi \wedge z = x_i - x_j)[x_i - x_j/z]$. Add the conjunct $\widetilde{\varphi}$ to φ'. Rewrite φ' in strong form i.e., in a form in which the bounds on a variable or the difference of two variables are as strong as possible. Strengthening of any conjunct will lead to a constraint which is not equivalent to the original constraint. Let the resulting constraint φ' be of the form $c_1 \wedge c_2 \wedge \ldots \wedge c_n$.
- Rewrite each conjunct c_i of the form $x_i + b \sim x_j + a$, where $\sim \in \{\leq, <\}$, as $x_j - x_i\, relop\, b - a$ where *relop* is $\geq$ or $>$ according as $\sim$ is $\leq$ or $<$.
- Rewrite each conjunct of the form $x_i \sim x_j$ in the form $x_i - x_j \sim 0$ if $\sim \in \{\geq, >, =\}$ or in the form $x_j - x_i\ relop\ 0$ where *relop* is $>$ or $\geq$ according as $\sim$ is $<$ or $\leq$. Similarly for the case of constraints of the form $x_i \sim x_j + a$.
- Any conjunct of the form $x_i - x_j \sim a$, where $\sim \in \{<, \leq\}$ will be rewritten as $x_j - x_i\ relop\ (-a)$ where *relop* is $>$ or $\geq$ according as $\sim$ is $<$ or $\leq$.
- Any conjunct of the form $x_i - x_j = a$ is rewritten in the form $x_i - x_j \geq a \wedge x_j - x_i \geq -a$.
- Rewrite any constant of the form $x_i = a$ in the form $x_i \geq a \wedge x_i \leq a$.

Figure 3.4: Normalization of Constraints.

normalized form in polynomial time. In what follows we deal with constraints in normalized form.

Now we are ready to introduce the trim operation. At a high level, the trim operation can be viewed as an accurate (with respect to the properties that we are concerned here) widening operation, i.e., it does not lose precision with respect to model checking for the properties that we are concerned with here. The removal and replacement of constraints in the definition of trim can be seen as constraint widening operations. The basic intuition is as follows: once the value of a real variable goes above the maximal constant, it does not matter what the value is. Hence, if a constraint has a solution in which the value of a variable is above the maximal constant, then the constraint can be widened to incorporate all "similar tuples". The relation $\approx_M$, as we show below, provides a logical characterization of the trim operation on constraints. Note that the definition of trim itself provides with an algorithm for trimming.

Definition 3.11 (Trim) *We define an operator trim, which given a satisfiable constraint φ, produces a constraint $\varphi' = trim(\varphi)$, by the method given below. The constraint $trim(\varphi)$ is obtained from the normalized form of φ by the following operations:*

- *Remove all constraints of the form $x_j - x_i > a$ or $x_j - x_i \geq a$, for each pair of variables x_i, x_j, $i \neq j$, such that $\varphi \wedge x_i > M$ is satisfiable and $\exists_{-x_j}(\varphi)$ is equivalent to $\exists_{-x_j}(\varphi \wedge x_i > M)$ and $(\varphi \wedge x_j > M)$ is not equivalent to φ, where a is an integer and the existential quantifier is over all variables but x_j.*
- *Remove all constraints of the form $x_i < c$ or $x_i \leq c$ where c is an integer and $c > M$.*
- *For each i, such that $(\varphi \wedge x_i > M)$ is equivalent to φ, replace all the constraints of the form $x_i - x_j \sim a$ or $x_i \sim c$ by the constraint $x_i > M$, where a and c are integers and $c > M$ and $\sim \in \{>, \geq\}$.*

Thus consider for example that the maximal constant $M = 4$. Let $\varphi \equiv x_1 - x_2 \geq 1 \wedge x_2 > 1 \wedge x2 < 3 \wedge x_2 - x_1 \geq -3$. Observe that for this constraint, $\varphi \wedge x_1 > M$ is satisfiable and $\exists_{-x_2}(\varphi)$ is equivalent to $\exists_{-x_2}(\varphi \wedge x_1 > 4)$ and $(\varphi \wedge x_2 > M)$ is not equivalent to φ. Hence $trim(\varphi) \equiv x_1 - x_2 \geq 1 \wedge x_2 > 1 \wedge x_2 < 3$. We illustrate the trim operation geometrically. In Figures 3.5, the solution set of a constraint φ in which the free variables are $\{x, y\}$ is shown by $ABCD$ in the left-half. The maximal constant M is indicated in the figure. None of the rules in the definition of trim apply to this constraint. Hence trimmed version of this constraint is the constraint itself. This is indicated in the right-half of the Figure 3.5. In Figure 3.6, the solution set of a constraint φ in which the free variables are $\{x, y\}$ is shown by $ABCD$ in the left-half. The maximal constant M is indicated in the figure. None of the rules in the definition of trim apply to this constraint. Hence trimmed version of this constraint is the constraint itself. This is indicated in the right-half of the Figure 3.6. In Figure 3.7, the solution set of a constraint φ in which the free variables are $\{x, y\}$ is shown by $ABCD$ in the left-half. The maximal constant M is indicated in the figure. The first rule in the definition of trim applies to this case. Hence the constraint rewrites to a constraint whose solution set is shown in the right half of Figure 3.7. In Figure 3.8, the solution set of a constraint φ in which the free variables are $\{x, y\}$ is shown by $ABCD$ in the left-half. The second and third rules in the definition of trim apply to this case. Hence the constraint is rewritten to one whose solution set is shown in the right half of Figure 3.8. Note that the trim of a constraint may not be a union of regions a' la' Alur and Dill [AD94]. Also note that the set of solutions of a constraint obtained by trimming another constraint is always convex. It is easy to see that algorithm for the trim operation completely avoids splitting of constraints.

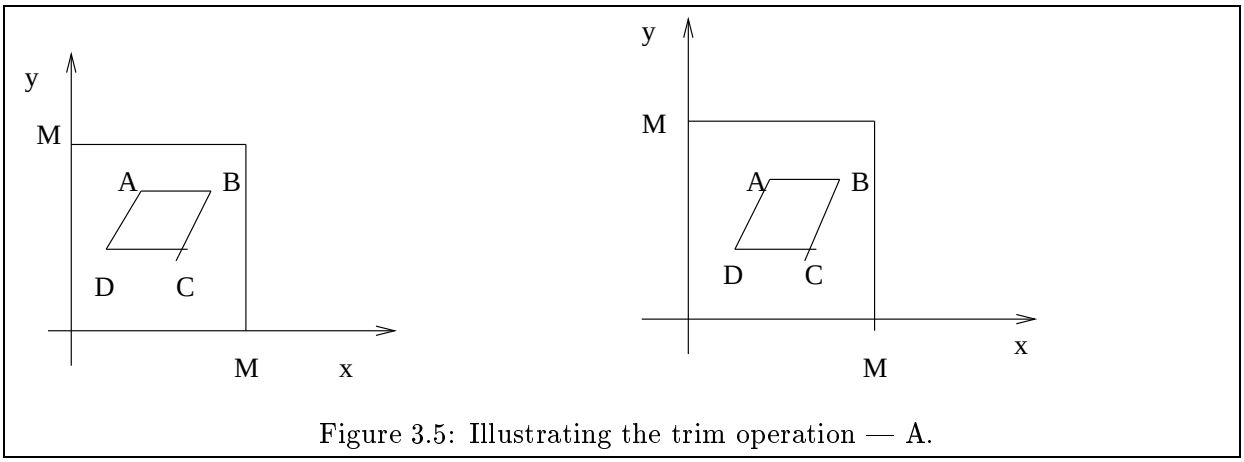


Figure 3.5: Illustrating the trim operation — A.

For the proof of the next results, the following notation is used. For a tuple $\mathbf{v}$, we define $\mathbf{v}[x_{i_1} := v_{i_1}, \ldots, x_{i_k} := v_{i_k}]$ as the tuple which agrees with $\mathbf{v}$ on all values of the variables except $x_{i_1}, \ldots, x_{i_k}$ which are set to $v_{i_1}, \ldots, v_{i_k}$ respectively.

Lemma 3.2 *The following properties hold for the trim operator:*

- *The trim operator is idempotent, i.e., for a non-ground state $\langle p(\mathbf{x}), \varphi \rangle$ reachable modulo M, $\langle p(\mathbf{x}), trim(\varphi) \rangle = \langle p(\mathbf{x}), trim(trim(\varphi)) \rangle$.*
- *For a non-ground state $\langle p(\mathbf{x}), \varphi \rangle$ reachable modulo M, we have $\langle p(\mathbf{x}), \varphi \rangle \approx_M \langle p(\mathbf{x}), trim(\varphi) \rangle$.*

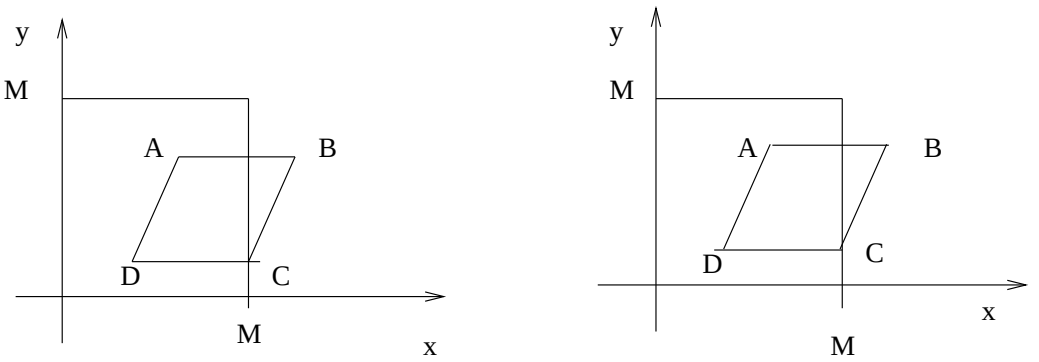


Figure 3.6: Illustrating the trim operation — B.

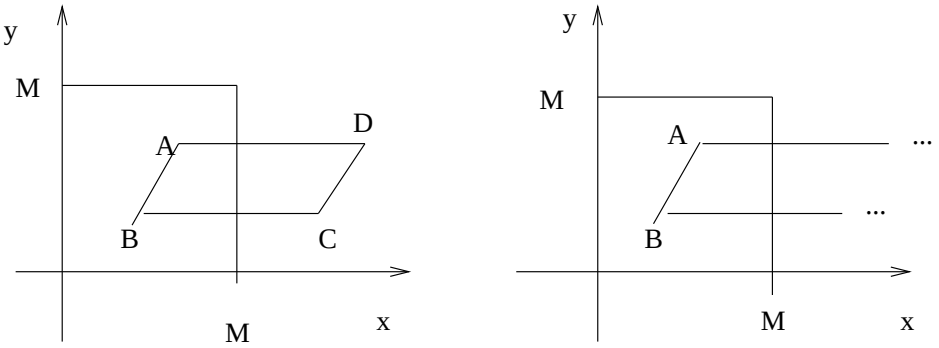


Figure 3.7: Illustrating the trim operation — C.

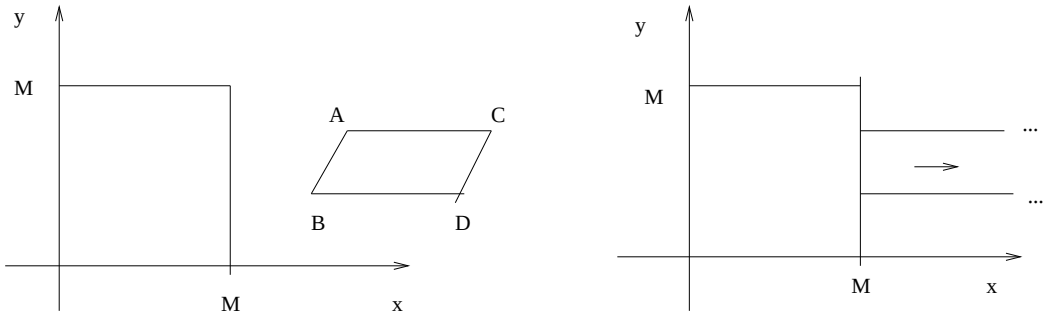


Figure 3.8: Illustrating the trim operation — D.

- *For a non-ground state $\langle p(\mathbf{x}), \varphi\rangle$ reachable modulo M and an atomic constraint θ, φ entails θ iff $trim(\varphi)$ entails θ.*

- *For a non-ground state $\langle p(\mathbf{x}), \varphi\rangle$ reachable modulo M and an atomic constraint θ, $\langle p(\mathbf{x}), trim(trim(\varphi) \wedge \theta)\rangle = \langle p(\mathbf{x}), trim(\varphi \wedge \theta)\rangle$.*

- *For a non-ground state $\langle p(\mathbf{x}), \varphi\rangle$ reachable modulo M,*

$$\langle p(\mathbf{x}), trim(\{x\}\varphi)\rangle = \langle p(\mathbf{x}), trim(\{x\}trim(\varphi))\rangle,$$

where for a constraint φ, we define $\{x\}\varphi$ to be the constraint such that $\mathcal{R}, \mathbf{v}[0/x] \models \{x\}\varphi$ if $\mathcal{R}, \mathbf{v} \models \varphi$.

Proof. The first statement is obvious since all the constraints that are to be removed or replaced by the trim operation get removed or replaced in the first operation of trim itself.

We next prove the third statement. One way is obvious. The other way is proved by using the second statement (which we prove below). Suppose $\varphi \models \theta$. Seeking a contradiction suppose that $trim(\varphi) \not\models \theta$. Then there exists a tuple $\mathbf{v}$ such that $\mathcal{R}, \mathbf{v} \models trim(\varphi)$ and $\mathcal{R}, \mathbf{v} \not\models \theta$. Then, by the second statement, there exists a $\mathcal{R}, \mathbf{v}' \models \varphi$ such that for all $i \in 1..n$ either $v_i = v_i'$ or both $v_i > M$ and $v_i' > M$. Now suppose θ is of the form $x_i > c$ where $c < M$. Then, of course, $\mathcal{R}, \mathbf{v}' \not\models \theta$. This is a contradiction. If θ is of the form $x_i > c$ with $c = M$, then we can obtain a contradiction in the same way. Similarly, for the other cases of θ, we can obtain a contradiction.

For the proof of the fourth statement, we use Lemma 3.3. We show that $\langle p(\mathbf{x}), \varphi \wedge \theta\rangle \approx_M \langle p(\mathbf{x}), trim(\varphi) \wedge \theta\rangle$. Then by the use of Lemma 3.3, the result follows. Suppose $\mathcal{R}, \mathbf{v} \models \varphi \wedge \theta$. Then $\mathcal{R}, \mathbf{v} \models \varphi$ and $\mathcal{R}, \mathbf{v} \models \theta$. Then $\mathcal{R}, \mathbf{v} \models trim(\varphi)$ and $\mathcal{R}, \mathbf{v} \models \theta$. On the other hand, suppose that $\mathcal{R}, \mathbf{v} \models trim(\varphi) \wedge \theta$. Then $\mathcal{R}, \mathbf{v} \models trim(\varphi)$ and $\mathcal{R}, \mathbf{v} \models \theta$. Then, by second statement of this Lemma, there exists $\mathcal{R}, \mathbf{v}'$ such that $\mathcal{R}, \mathbf{v}' \models trim(\varphi)$ and for all $i \in 1..n$, either $v_i = v_i'$ or both $v_i > M$ and $v_i' > M$. In both the possibilities, $\mathcal{R}, \mathbf{v}' \models \theta$. Hence $\mathcal{R}, \mathbf{v}' \models trim(\varphi) \wedge \theta$. The other direction is obvious. Therefore, $\langle p(\mathbf{x}), trim(\varphi) \wedge \theta\rangle \approx_M \langle p(\mathbf{x}), \varphi \wedge \theta\rangle$. Hence the result follows.

For the proof of the fifth statement, we reason as follows. We show that $\langle p(\mathbf{x}), \{x\}\varphi\rangle \approx_M \langle p(\mathbf{x}), \{x\}trim(\varphi)\rangle$. Then the results from Lemma 3.3. Suppose that $\mathcal{R}, \mathbf{u} \models \{x\}\varphi$ where $\mathcal{R}, \mathbf{u} = \mathbf{v}[0/x]$ where $\mathcal{R}, \mathbf{v} \models \varphi$. Then $\mathcal{R}, \mathbf{v}[0/x] \models \{x\}\varphi$. Now, by the second statement of this Lemma, we have that there exists $\mathbf{v}'$ such that $\mathcal{R}, \mathbf{v}' \models trim(\varphi)$ and for all $i \in 1..n$, either $v_i = v_i'$ or both $v_i > M$ and $v_i' > M$. Now $\mathcal{R}, \mathbf{v}'[0/x] \models \{x\}trim(\varphi)$. Now, $\mathbf{v}[0/x] \sim_M \mathbf{v}'[0/x]$ where $\sim_M$ is the equivalence relation defined in the proof of Proposition 3.2. Similarly, the other direction can be proved. Hence, $\langle p(\mathbf{x}), \{x\}\varphi\rangle \approx_M \langle p(\mathbf{x}), \{x\}trim(\varphi)\rangle$. Hence the result follows.

Finally, we prove the second statement. One direction is obvious. For the other direction, we reason as follows. Suppose that $\mathcal{R}, \mathbf{v} \models trim(\varphi)$. If $\mathcal{R}, \mathbf{v} \models \varphi$ then we are done. Otherwise, there exists some constraint θ_0 in φ such that θ_0 has been removed (or replaced by another constraint) by the trimming operation and $\mathcal{R}, \mathbf{v} \not\models \theta_0$. Now, we reason on the nature of θ_0 and the nature of its removal. Let θ_0 be of the form $x_i \leq c$ where $c > M$. Since $\mathcal{R}, \mathbf{v} \not\models x_i \leq c$, $v_i > c$. Consider the tuple $\mathbf{v}[x_i := c]$. Obviously, this tuple satisfies θ_0. If this tuple satisfies φ, we are done. Otherwise there exists a constraint θ_1 in φ that is not satisfied by this tuple. We now reason on the nature of θ_1. First observe that θ_1 cannot be a constraint of the form $x_i \leq c_1$. Second, if θ_1 is of the form $x_i \geq d$, then the following two cases arise. The first case is that of $d < M$ and θ_1 is not removed by the trimming operation. In this case the tuple satisfies θ_1. The

second case is that $d > M$ and θ_1 is removed during the trimming operation. But, then we can obtain the constraints $c \geq d$ and $c < d$ which is a contradiction. Now suppose, without loss of generality, that θ_1 is of the form $x_i - x_{i+1} \geq c_1$. We consider the case when θ_1 is removed by the trimming operation. The other case in which θ_1 is not removed by the trimming operation is easier. Suppose that $\varphi \wedge x_i > M$ is equivalent to φ and that θ_1 is removed. Then, two cases can arise. First see that $x_{i+1} \leq c - c_1$ (or a stronger constraint) is a conjunct of φ. If this constraint is not removed by the trimming operation, then, $v_j \leq c - c_1$ and hence the tuple satisfies θ_1. If $c - c_1 > M$ then this constraint is removed by the trimming operation. In this case, if $v_j \leq c - c_1$, then θ_1 is satisfied and we are done. Otherwise, consider the tuple $\mathbf{v}[x_i := c, x_{i+1} := c - c_1]$ (if there is another constraint $x_{i+1} \leq d$ stronger than $x_{i+1} \leq c - c_1$ in φ, then, either $v_j \leq c - c_1$ or consider the tuple $\mathbf{v}[x_i := c, x_{i+1} := d]$ and follow the reasoning below). Of course this tuple satisfies both θ_0 and θ_1. If this tuple satisfies φ we are done. Otherwise, there exists a constraint θ_2 that is not satisfied by this tuple. We now reason on the nature of θ_2. Observe that θ_2 cannot be of the form $x_{i+1} \leq d$. Also observe that θ_2 cannot be of the form $x_{i+1} \geq c_2$, otherwise φ is unsatisfiable. So θ_2 can only be of the form $x_{i+1} - x_{i+2} \geq c_2$. In this case, the reasoning starts again as previously. Since the number of variables is finite, we are going to show that this chain of reasoning terminates with a tuple $\mathbf{v}'$ such that $\mathcal{R}, \mathbf{v}' \models \varphi$ and $\mathbf{v} \sim_M \mathbf{v}'$. This is because, if this reasoning continues, at some point we must have that the tuple formed at that point does not satisfy θ_k where θ_k is the constraint $x_{i+k} - x_{i+k+1} \geq c_{k+1}$ such that the variable x_{i+k+1} has already been encountered in our reasoing; i.e., for some $l < k$, $x_{i+k+1} = x_{i+l}$ where x_{i+l} has been assigned to $c - c_1 - \ldots - c_l$ and x_{i+k} has already been assigned $c - c_1 - \ldots - c_k$ in our assignment process. Let the tuple formed by our reasoning method up to this point (we have been developing a tuple all through from $\mathbf{v}$ by reassigning values to $x_i, \ldots$) be $\mathbf{v}''$. We show that $-c_{l+1} - \ldots - c_k \geq c_{k+1}$ from which it follows that $\mathcal{R}, \mathbf{v}'' \models x_{i+k} - x_{i+k+1} \geq c_{k+1}$. Indeed, if φ is satisfiable then there exists a solution $\mathbf{w}$ of φ. This solution must satisfy the constraints $x_{i+l} - x_{i+l+1} \geq c_{l+1}, \ldots\ x_{i+k-1} - x_{i+k} \geq c_k$, whence it follows that $w_{i+l} - w_{i+k} \geq c_{l+1} + \ldots c_k$. Also, since $\mathcal{R}, \mathbf{w} \models x_{i+k} - x_{i+k+1} \geq c_{k+1}$ and $x_{i+k+1} = x_{i+l}$, we have $w_{i+k} - w_{i+l} \geq c_{k+1}$. From this we have $-c_{l+1} - \ldots - c_k \geq c_{k+1}$. Now all we need to show is that $\mathbf{v} \sim_M \mathbf{v}''$. Thus, we need to show that for all j such that the value of x_j has been updated from $\mathbf{v}$ to $\mathbf{v}''$, both $v_j > M$ and $v''_j > M$. Of course, this is true for x_i. For the other variables we reason as follows. Suppose for some l, in the reasoning chain above, $c - c_1 - \ldots - c_l \leq M$ and for all $m < l$ $c - c_1 - \ldots - c_m > M$. Now see that φ contains a conjunct $x_{i+1} \leq c - c_1$. Since φ contains the conjunct $x_{i+1} - x_{i+2} \geq c_2$, it must also have the conjunct $x_{i+2} \leq c - c_1 - c_2$. Going in this way, it must also contain the conjunct $x_l \leq c - c_1 - \ldots - c_l$. Now since $c - c_1 - \ldots - c_l \leq M$, this constraint is not removed by the trimming operation. Hence $v_{i+l} \leq c - c_1 - \ldots - c_l$. Hence the reasoning terminates here without updating x_{i+l}. Thus we show that $\mathbf{v}'' \sim_M \mathbf{v}$. The proofs for the remaining cases follow similar lines. ∎

Lemma 3.3 *For reachable non-ground states $\langle p(\mathbf{x}), \varphi\rangle$ and $\langle p(\mathbf{x}), \varphi'\rangle$, $\langle p(\mathbf{x}), \varphi\rangle \approx_M \langle p(\mathbf{x}), \varphi'\rangle$ iff $\langle p(\mathbf{x}), trim(\varphi)\rangle = \langle p(\mathbf{x}), trim(\varphi')\rangle$.*

Proof. $\Longleftarrow$: Suppose $\langle p(\mathbf{x}), trim(\varphi)\rangle = \langle p(\mathbf{x}), trim(\varphi')\rangle$. Since by the second statement of Lemma 3.2, $\langle p(\mathbf{x}), \varphi\rangle \approx_M \langle p(\mathbf{x}), trim(\varphi)\rangle$ and $\langle p(\mathbf{x}), \varphi'\rangle \approx_M \langle p(\mathbf{x}), trim(\varphi')\rangle$ the result follows.

$\Longrightarrow$: Seeking a contradiction suppose that $\langle p(\mathbf{x}), trim(\varphi)\rangle \neq \langle p(\mathbf{x}), trim(\varphi')\rangle$. Let $\mathcal{R}, \mathbf{v} \models trim(\varphi)$ but $\mathcal{R}, \mathbf{v} \not\models trim(\varphi')$. Wlog, suppose that $\mathcal{R}, \mathbf{v} \not\models x_i - x_j > a$ which is a constraint in $trim(\varphi')$ (wlog suppose that $a > 0$). Then $v_i - v_j \leq a$.

We reason as follows. If both v_i and v_j are less than or equal to M, we can obtain a contradiction as follows. By the second statement of Lemma 3.2 and the assumption of this Lemma, we have $\langle p(\mathbf{x}), trim(\varphi)\rangle \approx_M \langle p(\mathbf{x}), trim(\varphi')\rangle$. Now there does not exist a $\mathbf{v}'$ such that $\mathcal{R}, \mathbf{v}' \models trim(\varphi')$ and $\mathbf{v} \sim_M \mathbf{v}'$ (if $\mathbf{v}' \sim_M \mathbf{v}$, then by our assumption, $\mathcal{R}, \mathbf{v}' \not\models trim(\varphi)$).

Hence, without loss of generality, assume that $v_i > M$ and $v_j \leq M$. Now consider the tuple $\mathbf{v}[x_i := M]$. If this tuple satisfies $trim(\varphi)$ then we can again reason as previously and obtain a contradiction. If this tuple does not satisfy $trim(\varphi)$ then there must exist a constraint of the form $x_i - x_l \sim c$, where $\sim \in \{>, \geq\}$, which is not satisfied. Without loss of generality let this constraint be $x_i - x_{i+1} \geq c_i$. Consider the tuple $\mathbf{v}[x_i := M, x_{i+1} := v_{i+1} - (v_i - M)]$. Note that this tuple satisfies both the above constraints. If this does not satisfy $trim(\varphi)$, without loss of generality there must exist a constraint of the form $x_{i+1} - x_{i+2} \geq c_{i+1}$ which is not satisfied by this valuation. So consider the tuple $\mathbf{v}[x_i := M, x_{i+1} := v_{i+1} - (v_i - M), x_{i+2} := v_{i+2} - (v_i - M)]$. Since the number of variables is finite, this reasoning must terminate with a tuple $\mathbf{v}'$ satisfying $trim(\varphi)$ for which $v'_i = M$ and $v'_j = v_j$ or $v'_j = v_j - (v_i - M)$. Note that we cannot get a tuple $\mathcal{R}, \mathbf{v}'' \models trim(\varphi')$ such that $v''_i = v'_i$ and $v''_j = v'_j$ for then $\mathcal{R}, \mathbf{v}'' \models x_i - x_j \leq a$. This is a contradiction. The proof for the remaining cases proceeds by similar arguments. ‖

Definition 3.12 *A non-ground state $\langle p(\mathbf{x}), \varphi\rangle$ subsumes another non-ground state $\langle p(\mathbf{x}), \varphi'\rangle$, denoted by $\langle p(\mathbf{x}), \varphi'\rangle \prec \langle p(\mathbf{x}), \varphi\rangle$, if every ground instance of $\langle p(\mathbf{x}), \varphi'\rangle$ is also a ground instance for $\langle p(\mathbf{x}), \varphi\rangle$.*

We call the equivalence class of a reachable non-ground state in the equivalence relation $\approx_M$, a reachable equivalence class.

Lemma 3.4 *In each reachable equivalence class $\mathcal{E}$ of $\approx_M$, there exists a non-ground state reachable modulo M, called the largest non-ground state in $\mathcal{E}$ reachable modulo M, which subsumes all other non-ground states in $\mathcal{E}$ that are reachable modulo M.*

Proof. From lemma 3.2, it follows that for a non-ground state $\langle p(\mathbf{x}), \varphi\rangle$ reachable modulo M, $\langle p(\mathbf{x}), \varphi\rangle \approx_M \langle p(\mathbf{x}), trim(\varphi)\rangle$. Note that for all reachable nonground states $\langle p(\mathbf{x}), \varphi\rangle$, $trim(\varphi) \models \varphi$ (and also by Lemma 3.2, the trim operation is idempotent). By Lemma 3.3, there exists a φ' such that for all $\langle p(\mathbf{x}), \varphi\rangle \in \mathcal{E}$, $\varphi' = trim(\varphi)$. Then $\langle p(\mathbf{x}), \varphi'\rangle$ is the largest reachable (modulo M) state in $\mathcal{E}$. ‖

Lemma 3.5 *The equivalence relation $\approx_M$ produces a finite number of reachable equivalence classes (i.e., equivalence classes containing reachable nonground states).*

Proof. Each reachable equivalence class $\mathcal{E}$ can be represented by $\langle p(\mathbf{x}), trim(\varphi)\rangle$, where $\langle p(\mathbf{x}), \varphi\rangle \in \mathcal{E}$. The constraint store (for a non-ground state $s = \langle p(\mathbf{x}), \varphi\rangle$, we call φ the constraint store of s) for this state is given by grammar 3.3. Now we show that the constraint store for the representative cannot contain constraints of the form $x_i - x_j \sim a$ or $x_i \; relop \; a$ for all $i, j = 1, \ldots, n$, where $\sim \in \{>, \geq\}$, $relop \in \{>, <, \geq, \leq\}$, and $|a| > M$. Seeking a contradiction, suppose there exists a conjunct of the form $x_i - x_j > a$ where $|a| > M$. First suppose that $a > 0$. Then this conjunct is also present in φ. Suppose $\mathcal{R}, \mathbf{v} \models \varphi$. Then $v_i - v_j > a$. Therefore $v_i > a$. Therefore there exists no solution $\mathbf{v}$ of φ such that $v_i \leq M$. So $\varphi \wedge x_i > M \equiv \varphi$. So the constraint is removed by the trim operation. Similarly for the case when $a < 0$. Now we write each constraint $x_i - x_j = c$ in the form $x_i - x_j \geq c \wedge x_i - x_j \leq c$. Similar for the case

$x_i = c$. So given this representation, syntactically the number of distinct constraints is bounded by $(4M+4)^{n(n-1)} \cdot (2M+2)^{2n}$ which is $2^{O(n^2)} \cdot (2M+2)^{O(n^2)}$. This is because there are $n(n-1)$ pairs x_i, x_j. For each constraint of the form $x_i - x_j \sim c$, $\sim$ can be $>$ or $\geq$, and c can take integral values from $-M$ to M. Also for each constraint $x_i \, relop \, c$, where $relop \in \{>, \geq\}$ (i.e., constraint determining the lower bound of a variable) c can be a non-negative integer in the interval $[0, M]$. Similarly the case for the constraints determining the upper bound of a variable. ‖

Proposition 3.4 *Given two reachable non-ground states $\langle p(\mathbf{x}), \varphi \rangle$ and $\langle p(\mathbf{x}), \varphi' \rangle$, where both φ and φ' are in normalized form, it is effectively decidable whether $\langle p(\mathbf{x}), \varphi \rangle \approx_M \langle p(\mathbf{x}), \varphi' \rangle$.*

Proof. From lemma 3.3, it follows that to check $\langle p(\mathbf{x}), \varphi \rangle \approx_M \langle p(\mathbf{x}), \varphi' \rangle$, we need to check whether $\langle p(\mathbf{x}), trim(\varphi) \rangle = \langle p(\mathbf{x}), trim(\varphi') \rangle$. Now for a constraint φ, projection on a variable can be done in polynomial time. Also checking for equivalence of two constraints can be done in polynomial time. Now string searching can also be done in polynomial time. So from the definition of *trim*, it can be seen that it is decidable in polynomial time whether $\langle p(\mathbf{x}), \varphi \rangle \approx_M \langle p(\mathbf{x}), \varphi' \rangle$. ‖

The trim operation described above can be combined with the tabling strategy mentioned above to provide a termination guarantee for the model checking procedure. If $\langle p'(\mathbf{x}), \varphi' \rangle$ is the resolvent of$\langle p(\mathbf{x}), \varphi \rangle$ through a clause C, then we add the goal $\langle p'(\mathbf{x}), trim(\varphi') \rangle$ as the table entry. The detailed algorithm is described below. By lemma 3.5, termination of the algorithm is guaranteed. Before presenting the algorithm, we observe the following Lemma.

Lemma 3.6 *For every $\mathcal{L}_s$ formula Φ, the non-ground state $\langle pred(\mathbf{x}), \varphi \rangle$ succeeds in $\mathcal{P}^{\widetilde{\Phi}}$ iff the non-ground state $\langle pred(\mathbf{x}), trim(\varphi) \rangle$ succeeds in $\mathcal{P}^{\widetilde{\Phi}}$.*

Proof. By induction on structure of $\widetilde{\mathcal{L}_s}$ formulas.

Base Case The case in which *pred* is of the form $\langle p, X \rangle$ where the declaration of X is given by $X = p$, where p is an atomic proposition is obvious for this case. The second case is where *pred* is of the form $\langle p, X \rangle$ where the declaration is given by $X = \theta$ where θ is an atomic clock constraint. Suppose $\langle pred(\mathbf{x}), \varphi \rangle$ succeeds. Then there exists an instance $pred(\mathbf{v})$ such that $\mathcal{R}, \mathbf{v} \models \theta$. Since $\varphi \models trim(\varphi)$, $\mathcal{R}, \mathbf{v} \models trim(\varphi)$. On the other hand if $\langle pred(\mathbf{x}), trim(\varphi) \rangle$ succeeds then there exists a ground instance $pred(\mathbf{v})$ of $\langle pred(\mathbf{x}), trim(\varphi) \rangle$ such that $\mathcal{R}, \mathbf{v} \models \theta$. Since from the second statement of Lemma 3.2, $\langle pred(\mathbf{x}), \varphi \rangle \approx_M \langle pred(\mathbf{x}), trim(\varphi) \rangle$, there exists $\mathbf{v}'$ such that for all i, either $v_i = v'_i$ or $v_i > M \wedge v'_i > M$ and $\mathcal{R}, \mathbf{v}' \models \varphi$. Now, from the proof of Proposition 3.2 $\mathcal{R}, \mathbf{v}' \models \theta$ since $\mathcal{R}, \mathbf{v} \models \theta$.

Induction Step For the boolean connectives, the reset and the modalities, the result follows from the induction hypothesis. ‖

3.8 Extension of OLDT Resolution to Constraints

We extend the subsumption ordering defined in Definition 3.12 to a partial order $<^*$ on the set of reachable equivalence classes induced by the equivalence relation $\approx_M$ as follows: For a reachable equivalence class $\mathcal{E}$, denote its representative as $rep(\mathcal{E})$. Then for two reachable equivalence classes $\mathcal{E}$ and $\mathcal{E}'$, $\mathcal{E} <^* \mathcal{E}'$ iff $rep(\mathcal{E}) \prec rep(\mathcal{E}')$.

We extend the OLDT resolution of [TS86a] in the following way. First note that we do not have any function symbols in our program. We assume that a goal is of the form $G = \langle Q, \varphi \rangle$ where Q is a conjunction of predicates and φ is the constraint store. We also assume that the solution list associate with each entry in the solution table is a list in which each entry is a constraint. The table node registration procedure is extended as follows. First, we mark each predicate as a tabled predicate. Thus every node in an OLDT structure (which is not a success leaf) is a table node. Let G be a goal labeling a table node v in the OLDT structure. Let $G = \langle Q, \varphi \rangle$ and let $pred(\mathbf{x})$ be the leftmost predicate in Q. The following cases are distinguished:

- Lookup Node: Compute $\varphi' = \exists_{-\mathbf{x}}\varphi$. If there exists a entry $\langle pred(\mathbf{x}), \varphi'' \rangle$ such that $rep(\mathcal{E}') \prec rep(\mathcal{E}'')$, where $\mathcal{E}'$ and $\mathcal{E}''$ are respectively the equivalence classes of $\langle pred(\mathbf{x}), \varphi' \rangle$ and $\langle pred(\mathbf{x}), \varphi'' \rangle$ induced by $\approx_M$ (for a reachable non-ground state $\langle pred(\mathbf{x}), \varphi \rangle$, the representative of its equivalence class is $\langle pred(\mathbf{x}), trim(\varphi) \rangle$), then put v in the look up table with a pointer to the entire solution list of $\langle pred(\mathbf{x}), \varphi'' \rangle$.

- Solution Node: If the above case does not hold then put $\langle pred(\mathbf{x}), trim(\varphi') \rangle$ in the solution table with an empty solution list.

The initial OLDT structure is the same as in [TS86a], with a forest with a single node labeled with $\langle init, \widetilde{Z} \rangle$. The immediate extension part closely follows that of [TS86a]. Given $\mathcal{P}^{\widetilde{\Phi}}$ and an OLDT structure T, an *immediate extension* of T by $\mathcal{P}^{\widetilde{\Phi}}$ is the result of either of the following operations.

1. Select a terminal node v which is not a look-up node (the question of this node being a success node will not arise as we will see later). Let the node be labeled by the non-ground goal $\langle Q, \varphi \rangle$. Let $pred(\mathbf{x})$ be the leftmost predicate of Q and let $Q = pred(\mathbf{x}) \wedge Q'$. Also let $Variables(Q')$ be the set of variables occurring in Q'. Compute $\langle pred(\mathbf{x}), trim(\exists_{-\mathbf{x}}\varphi) \rangle$. If there exists at least one clause in $\mathcal{P}^{\widetilde{\Phi}}$ through which $\langle pred(\mathbf{x}), trim(\exists_{-\mathbf{x}}\varphi) \rangle$ resolves then

   (a) Let $C_1, \ldots, C_k$ $(k \geq 1)$ be all the clauses in $\mathcal{P}^{\widetilde{\Phi}}$ through which $\langle pred(\mathbf{x}), trim(\exists_{-\mathbf{x}}\varphi) \rangle$ resolves. Create k children $\langle Q_i, \varphi_i \rangle$ where $Q_i = B_i \wedge P$ and $\varphi_i = \exists_{-(Variables(P'), Variables(B_i))}(\varphi \wedge (trim(\exists_{-\mathbf{x}}\varphi) \wedge \psi_i \wedge \Theta_i))$ where B_i is conjunction of predicates the body of C_i and Θ_i is the mgu of the head C_i and $pred(\mathbf{x})$ and ψ_i is the constraint in clause C_i.

   (b) For each new node, register it.

   (c) For each unit subrefutation [TS86a], if there are any, starting from a solution node and ending with some of the new nodes, let the subrefutation be for the tabled non-ground state $\langle pred(\mathbf{x}), \varphi \rangle$. Add the answer constraint to the last of the solution list of the table entry $\langle pred(\mathbf{x}), trim(\varphi) \rangle$ provided the answer is not already present in that solution list (i.e., does not entail the answers present in the solution list).

2. Look-up extension: Select a look-up node v, such that the pointer associated with it points to a non-empty sublist of a solution list. Let $pred(\mathbf{x})$ be the leftmost predicate in the goal G labeling v. Advance the pointer by one to skip the head element of the sublist. If $pred(\mathbf{x}) \longleftarrow \psi$ and G are resolvable in the sense given above, where ψ is the constraint pointed to by the pointer, create a child node of v labeled with the resolvent. Do the same thing as in step 1c.

Example	*time (seconds)*
Example in figure 3.2	1.5
Fischer's Protocol (Two Processes) [LPY95a]	4.2
Rail-road Crossing	1.8
Audio Protocol [HWT95]	7.2

Figure 3.9: Experimental Results.

The rest of the details are natural extensions of those in [TS86a] which we do not repeat here. Since, our aim is to model check locally, we will terminate as soon as a success leaf is encountered. Note that $trim(\langle pred(\mathbf{x}), \varphi \rangle) = \langle pred(\mathbf{x}), trim(\varphi) \rangle$. Also note that we need the constraints to be in normalized form as our algorithm (for $trim$) works on the syntax of the constraints.

Theorem 3.3 (Soundness and Completeness) *The algorithm for model checking for $\mathcal{L}_s$ given above is sound and complete.*

Proof. The proof of soundness of the algorithm is by a simple extension of the proof of Lemma 3.17 in [TS86a] combined with Lemma 3.6. The proof of completeness is by a simple extension of the proof of Theorem 3.18 in [TS86a] along with Lemma 3.5. ∎

We have implemented a prototype local model checker based on the method given above. Even without any fine tuning, the performance of the model checker seems to be encouraging. In fact, even without any fine tuning, the timings obtained in many cases are comparable to that of UPPAAL [BLL+96] which is a highly fine tuned tool with a lot of inbuilt optimizations. We have used our model checker to verify the safety properties of several well known benchmark examples taken from literature. The experimental results are summarized in table in Figure 6.14. All the results are obtained on PC (200 MHz Pentium Pro). All the timings denote the total time needed.

3.9 Full Disjunction

In this section, we show how to model check for the logic $\mathcal{L}_s$ extended with full disjunction. Note that the logic $\mathcal{L}_s$ [LPY95a] described above allows only restricted disjunction. In this subsection, we show that in our framework we can allow for full disjunction. Note that it is stated in [LPY95a] that their model checking technique based on the rewrite tree cannot be extended to a logic with general disjunction. We call the extension of the logic $\mathcal{L}_s$ with full disjunction $\mathcal{XL}_s$. Dually, we call the the extension of the logic $\widetilde{\mathcal{L}_s}$ with full conjunction as $\widetilde{\mathcal{XL}_s}$ (i.e., the dual of $\mathcal{XL}_s$). The satisfaction relation for $\mathcal{XL}_s$ is the satisfaction relation for $\mathcal{L}_s$ augmented with the clause:

- $\mathcal{P}, p(\mathbf{v}) \models \Phi_1 \vee \Phi_2$ implies $\mathcal{P}, p(\mathbf{v}) \models \Phi_1$ or $\mathcal{P}, p(\mathbf{v}) \models \Phi_2$.

For an $\mathcal{XL}_s$ formula Φ we can obtain an $\widetilde{\mathcal{XL}_s}$ formula $\widetilde{\Phi}$ in the similar way as above ($\widetilde{\mathcal{XL}_s}$ is the corresponding extension of $\widetilde{\mathcal{L}_s}$). Given a TLP $\mathcal{P}$ and a $\widetilde{\mathcal{XL}_s}$ formula $\widetilde{\Phi}$, we can construct a product program $\mathcal{P}^{\widetilde{\Phi}}$ using an extension of the product construction given above by the following "alternating" clause.

– $X = X_1 \wedge X_2$: $\langle p, X\rangle(\mathbf{x}) \longleftarrow \langle p, X_1\rangle(\mathbf{x}) \wedge \langle p, X_2\rangle(\mathbf{x})$.

Theorem 3.4 *Given a TLP $\mathcal{P}$ and an $\mathcal{XL}_s$ formula Φ, $\mathcal{P} \models \Phi$ if and only if $\langle init, \widetilde{Z}\rangle$ is not in the least model of $\mathcal{P}^{\widetilde{\Phi}}$ where $\widetilde{\Phi}$ is the $\widetilde{\mathcal{XL}_s}$ formula corresponding to Φ and Z is the root variable of Φ.*

Proof. Similar to that of Theorem 8.1. ∎

Note that we do not have to change the methodology for the implementation for this extension – we can reuse the implementation described above.

Note that the rewrite tree based model checking procedure [LPY95a] implemented in the model checker Uppaal [BLL+96] can be viewed as a special case of our derivation tree using tabled resolution with constraints as described above. Use of tabled resolution with constraints allows us to increase the expressiveness of the underlying logic ([LPY95a] allows only restricted disjunction). Also note that the model checking procedure in [LPY95a] may not terminate (consider the timed automaton given in Figure 3.2 and the formula $X = x2 < 2 \wedge \Box X \wedge \forall X$ where $x2$ refers to the clock $x2$ of the timed automaton; this asserts that always the value of the clock $x2$ will be less than 2). In contrast our model checking procedure combined with the trim operation is guaranteed to terminate. Like the model checking procedure in [LPY95a], our model checking procedure is also local (only the reachable portion of the state space is explored and the state space is explored in a demand-driven fashion).

3.10 Unbounded Liveness Properties

In this section, we extend our methodology to deal with unbounded liveness properties of timed logic processes. Throughout this section, we consider only divergent ground derivations of TLPs. An unbounded liveness property is a declaration of the form $Z = \Box X$ where $X = \bar{q} \vee \forall \Box X$ (this is actually the dual of the property $Z = \Diamond X$ where $X = q \wedge \exists \Diamond X$, where we take the greatest fixpoint of the declaration) where $\bar{q}$ is an atomic proposition (q is an atomic proposition that is "satisfied" by all predicate symbol that do not satisfy $\bar{q}$) and we take the least fixpoint of the declaration (viewed as an equation). This asserts that "for all (infinite) ground derivations (starting from *init* using a resolution through an initial or a system clause), using resolutions through evolution clauses and system clauses in such a way that every resolution step through an initial or a system clause is immediately followed by one through an evolution clause and every resolution step through an evolution clause is immediately followed by one through a system clause, there exists a ground atom in that satisfies q". For timed automata this is the same as the assertion that for all (infinite) traces starting from the initial position using time transitions followed by edge transitions, i.e., every time transition step is immediately followed by an edge transition step and vice versa, there exists a position that satisfies q. Note that this is the dual of the specification which asserts that "there exists an (infinite) ground derivation (starting from *init* using a resolution through an initial clause or a system clause), using resolutions through evolution clauses and system clauses in such a way that every resolution step through an initial or a system clause is immediately followed by one through an evolution clause and every resolution step through an evolution clause is immediately followed by one through a system clause, such that every ground atom in the derivation satisfies the atomic proposition q".

Given an unbounded liveness specification Ψ (let $\Phi \equiv \widetilde{\Psi}$; i.e., $\widetilde{\Psi}$ is the dual of Ψ; i.e., $\mathcal{P} \models \Psi$ iff $\mathcal{P} \not\models \Phi$), and a TLP $\mathcal{P}$, we construct a TLP $\mathcal{P}^{\Phi}$ such that $\mathcal{P} \models \Phi$ iff the atom $\langle init, X\rangle$ is

in the greatest model of $\mathcal{P}^\Phi$. The construction of a product program is same as that shown in case of $\widetilde{\mathcal{L}_s}$.

Theorem 3.5 *Given a TLP $\mathcal{P}$ and an unbounded liveness specification Ψ, we have $\mathcal{P} \models \Psi$ if and only if the atom $\langle init, X\rangle$ is not in the greatest model of $\mathcal{P}^\Phi$, where $\Phi = \widetilde{\Psi}$ (the dual of Ψ) and X is the root variable of Φ.*

Proof. We first prove that if $\mathcal{P}$ does not satisfy Ψ, then $\langle init, X\rangle$ is in the greatest model of $\mathcal{P}^\Phi$. Suppose $\mathcal{P}, init \not\models \Psi$. Seeking a contradiction, suppose that $\langle init, X\rangle$ is not in the greatest model of $\mathcal{P}^\Phi$. If $\langle init, X\rangle$ fails then for all derivations starting from it, there exists a ground atom which which does not resolve through any clause in $\mathcal{P}^\Phi$. Let $\mathcal{G}$ be a ground derivation starting from $\langle init, X\rangle$ and let $\langle p, X\rangle(\mathbf{v})$ be a ground atom in it that does not resolve through any clause in $\mathcal{P}^\Phi$. Then, by the construction of $\mathcal{P}^\Phi$, either $q \notin P(p)$ or there does not exist any ground successor of $p(\mathbf{v})$. In either case, $\mathcal{P}, p(\mathbf{v}) \not\models p \wedge \exists\langle\rangle X$. Since this holds for each ground derivation $\mathcal{P}, init \not\models_{nls} \Phi$. Hence $\mathcal{P}, init \models \Psi$ which is a contradiction. Therefore $\langle init, X\rangle$ is in the greatest model of $\mathcal{P}^\Phi$.

Now we show that if $\mathcal{P} \models \Psi$, then $\langle init, X\rangle$ is not in the greatest model of $\mathcal{P}^\Phi$. Suppose that $\mathcal{P}, init \models \Psi$. Then $\mathcal{P}, init \not\models \Phi$. Seeking a contradiction suppose that $\langle init, X\rangle$ is in the greatest model of $\mathcal{P}^\Phi$. Then there is an infinite derivation starting from $\langle init, X\rangle$ through the clauses in $\mathcal{P}^\Phi$. This is because, by the construction, $\mathcal{P}^\Phi$ does not contain any assertion clause. Let this derivation be $\mathcal{G}$. For every ground atom $\langle p, X\rangle(\mathbf{v})$ in $\mathcal{G}$, there exists a clause in $\mathcal{P}^\Phi$ through which $\langle p, X\rangle$ resolves. Hence for every ground atom $\langle p, X\rangle(\mathbf{v})$ in $\mathcal{G}$, $q \in P(p)$. Now it can be shown by induction on the length of derivation that there exists an infinite ground derivation from $init$ in $\mathcal{P}$ in which the first derivation step is through an initial clause or a system clause, each derivation step through an initial clause or a system clause is followed by one through an evolution clause, each derivation step through an evolution clause is followed by one through a system clause and for each ground atom $p(\mathbf{v})$ in the derivation $q \in P(p)$. Hence $\mathcal{P}, init \models \Phi$. This is a contradiction. ∎

3.11 Implementation

Since model checking $\mathcal{P}$ for an (unbounded) liveness property Ψ reduces to checking whether $\langle init, X\rangle$ is contained in the greatest model of $\mathcal{P}^\Phi$ (as constructed above), it can be done by computing the greatest fixpoint of the immediate consequence operator for $\mathcal{P}^\Phi$. This results in a global model checker. Alternately, since the clauses in $\mathcal{P}^\Phi$ have at most one predicate in the body (from the construction of the program), we introduce a new greatest model resolution with tabling[1] prove that $\langle init, X\rangle$ is in the greatest model of $\mathcal{P}^\Phi$. To the best of the knowledge of the author, this is the first time any kind of tabling (without negation) is used for the greatest model of a constraint query language program. The greatest model resolution algorithm also allows us to avoid splitting of constraints. This is because, otherwise, in order to get a local algorithm, we had to introduce negation in the clause bodies. This would have resulted in splitting of constraints. The greatest model resolution algorithm with tabling is given in Figure 3.11. In step 3(b) of the algorithm we check whether there exists a goal $\langle pred'(\mathbf{x}), \varphi''\rangle$ in the table such that φ'' entails the constraint store φ' of the newly generated goal $\langle pred'(\mathbf{x}), \varphi'\rangle$. In this case, we

[1]Note that the tabling used here is different from that used in Section 3.6 as well as those in [CW96, TS86a].

do not need to register the solutions into the table. We will terminate at the first instance of a success leaf or the first instance when a newly generated goal contains a goal already in the table (whichever occurs earlier). Note that in the above implementation, use of negation along with tabled resolution for least model would have resulted in computing the negation of a constraint which is prohibitively expensive in practice. Note the procedure in Figure 3.11 holds only for constraint query language programs that have only one predicate in the body. The procedure can be easily extended to account for general programs (without negation). We illustrate the greatest model resolution with an example. The basic idea behind the procedure is to check if there exists a successful derivation or an infinite derivation starting from $\langle init, X \rangle$.

Example 3.2 Consider the program

$$
\begin{array}{rcl}
\langle init, X \rangle & \longleftarrow & p_1(\mathbf{x}) \wedge x_1 = x_2 \wedge x_2 \geq 0 \\
p_1(\mathbf{x}) & \longleftarrow & p_2(\mathbf{x}') \wedge \wedge x_1' = 0 \wedge x_2' = x_2 \\
p_2(\mathbf{x}) & \longleftarrow & p_3(\mathbf{x}') \wedge x_1' = x_1 + z \wedge x_2' = x_2 + z \wedge z \geq 0 \\
p_3(\mathbf{x}) & \longleftarrow & p_4(\mathbf{x}') \wedge x_2 \leq 2 \wedge x_2' = 2 \wedge x_1' = x_1 \\
p_4(\mathbf{x}) & \longleftarrow & p_5(\mathbf{x}') \wedge x_1' = x_1 + z \wedge x_2' = x_2 + z \wedge z \geq 0 \\
p_5(\mathbf{x}) & \longleftarrow & p_4(\mathbf{x}') \wedge x_2' = 0 \wedge x_1' = x_1 \wedge x_2 \leq 2
\end{array}
$$

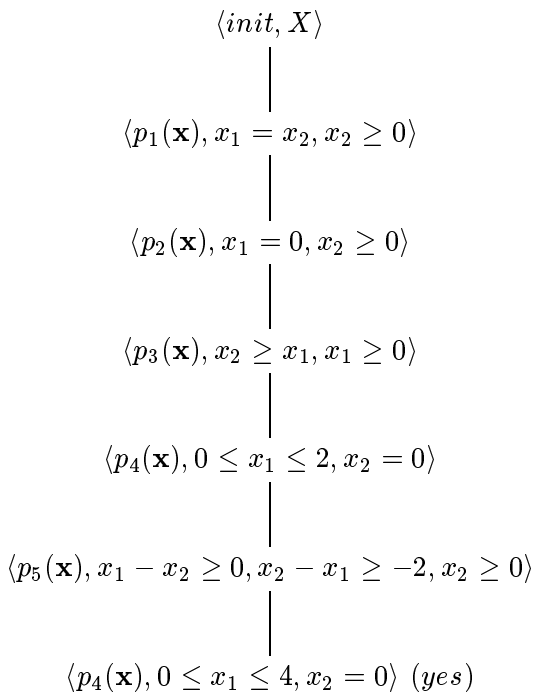


Figure 3.10: Illustrating the Greatest Model Resolution.

The derivation tree using the greatest model resolution for this example is given in Figure 3.10. The goal $\langle p(\mathbf{x}), x_1 = x_2, x_2 \geq 0 \rangle$ labeling the second node from the top is the resolvent of the goal $\langle init, X \rangle$ and the first clause. Similarly, the goal labeling the second node is the resolvent

of $\langle p(\mathbf{x}), x_1 = x_2, x_2 \geq 0\rangle$ and the second clause. Note that the constraint store of the state labeling the 5th node entails that of the state labeling the 7th node. Hence the 7th node is a 'yes' leaf (in line $4(b)(i)$ in Figure 3.11 the *Flag* is made *true*). This implies that $\langle init, X\rangle$ is in the greatest model of the program. Figure 3.10 shows the contents of *Table* for this example (the tree viewed from the bottom). Note that the algorithm in Figure 3.11 is depth-first.

Theorem 3.6 (Soundness) *If procedure in Figure 3.11 terminates then $\langle init, X\rangle$ is contained in the greatest model of $\mathcal{P}^\Phi$ if and only if it returns 'yes'.*

Proof. Consider algorithm in Figure 3.11. It returns 'yes' in the following cases:

Case 1: A non-ground descendent of $\langle init, X\rangle$ is a success leaf. In this case $\langle init, X\rangle$ is in the greatest model of P^Φ

Case 2: A non-ground descendent ng' of $\langle init, X\rangle$ is such that there exists an ancestor ng'' of ng' such that the constraint store of ng'' entails the constraint store of ng'. Now let $C_1, \ldots, C_k$ be the clause in the derivation from ng'' to ng'. Then ng' cannot fail as it goes through $C_1, \ldots, C_k$ and produces a non-ground state ng''' such that the constraint store of ng' entails that of ng'''. Hence $\langle init, X\rangle$ is in the greatest model of $\mathcal{P}^\Phi$.

To prove the other way, assume that $\langle init, X\rangle$ is in the greatest model of $\mathcal{P}^\Phi$. Assume that the procedure in Figure 3.11 terminates. Seeking a contradiction, suppose that the procedure returns 'no'. Then the procedure terminates on finding the stack *Table* empty at the end of the **repeat** – **until** loop. This means that in the depth-first tree generated by the procedure, every leaf is a failure leaf. But this contradicts the fact that $\langle init, X\rangle$ is in the greatest model of $\mathcal{P}^\Phi$. ∎

Note that procedure in Figure 3.11 may not terminate. The counter example is provided by the TLP corresponding to the timed automaton in Figure 3.12. It has two real variables x and y and one location m^0. Let the predicate $at\bar{}_{m^0}$ be an atomic proposition that does not hold at the location m^0. Consider the unbounded liveness property $Z = \Box X$ where $X = at\bar{}_{m^0} \vee \forall\Box X$ (actually consider its dual $Z = \Diamond X$ where $X = at_{m^0} \wedge \exists\Diamond X$). An infinite sequence of nonground states of the form $\langle\langle m^0, X\rangle(\mathbf{x}), y = x + i \ \wedge \ x \geq 0\rangle$ are generated where $i \in \mathcal{N}$. To ensure the termination of the model checking procedure, as in the previous section, we can combine the trim operation described above along with the procedure. The details are straightforward. The greatest model resolution procedure combined with the trim operator can also be implemented as a non-deterministic procedure requiring polynomial space (due to Lemma 3.5). A deterministic algorithm requiring polynomial space can then be obtained by using Savitch's theorem.

Using our method, we have been able to verify the unbounded liveness property $Z = \Box X$ where $X = at\_2 \vee \forall\Box X$ for the example of timed automaton shown in Figure 3.2 (TLP corresponding to that timed automaton), where the atomic proposition at_2 is satisfied only by the location 2.

The local model checking algorithm given in Section 3.6 and the model checking algorithm for unbounded liveness properties given above can be combined effectively to model check for receptiveness properties. A receptiveness property is a formula of the form $\Phi_1; \Phi_2$, where Φ_1 is a declaration of the form $X = X_1 \vee \exists\Diamond X \vee \exists X$ and Φ_2 is a declaration of the form $X_1 = q \wedge \exists\Diamond X_1$, where we take the least fixpoint for the first declaration and the greatest fixpoint for the second declaration. This asserts that there exists a reachable ground atom p such that there exists an infinite derivation (using resolutions through evolution clauses and system clauses in such a way that the first resolution step is through an evolution clause and every resolution step

Procedure Greatest Model Resolution
Input Program $\mathcal{P}^{\Phi}$ and the atom (0-ary predicate) $\langle init, X\rangle$
Output A yes/no answer whether the atom is in the greatest model of $\mathcal{P}^{\Phi}$
Data Structures
Stack *Table*
begin
Push $\langle init, X\rangle$ in *Table*.
repeat

1. Let $\langle pred(\mathbf{x}), \varphi\rangle$ be the non-ground state at the top of the stack *Table*.
2. **If** $\langle pred(\mathbf{x}), \varphi\rangle$ succeeds through a clause, return yes.
3. **else**
   (a) **If** there exists a clause C in $\mathcal{P}^{\Phi}$ such that $\langle pred(\mathbf{x}), \varphi\rangle$ has still not resolved through C then let the resolvent of C and $\langle pred(\mathbf{x}), \varphi\rangle$ be $\langle pred'(\mathbf{x}), \varphi'\rangle$.
   (b) **If** there exists $\langle pred'(\mathbf{x}), \varphi''\rangle$ in *Table* such that $\varphi'' \models \varphi'$, return yes.
   (c) **else** push $\langle pred'(\mathbf{x}), \varphi'\rangle$ to *Table*. (end **If**)
   (d) **else** pop $\langle pred(\mathbf{x}), \varphi\rangle$ from the stack *Table*. (end **if**)
4. (end **If**)

until *Table* is empty (end of **repeat until**)
return no.
end

Figure 3.11: Greatest Model Resolution (GMR) Procedure for programs with one body predicate.

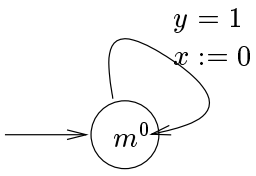


Figure 3.12: Non-terminating Example for Greatest Model Resolution.

through a system clause is immediately followed by one through an evolution clause and vice versa) starting from p in which every ground atom satisfies q (for timed automata, this amounts to the specification that there exists a reachable position $\widetilde{p}$ such that there exists an (infinite) trace starting from $\widetilde{p}$ using time transitions followed by edge transitions, i.e., first taking a time transition and then following it up by an edge transition and so on, such that every position in that trace satisfies q). Using the combination mentioned above, we have been able to falsify the receptiveness property for the example in Figure 3.2 with $q = \neg at\_2$. The model checker Uppaal [BLL+96] is not able to verify receptiveness properties.

3.12 Model Checking for TCTL formulas

In this section, we extend our methodology to deal with the model checking problem for TCTL formulas. The formulas Φ of Timed Computation Tree Logic (TCTL) are inductively defined as follows.

$$\Phi \ ::= \ q \mid x + c \leq y + d \mid \neg\varphi \mid \Phi_1 \vee \Phi_2 \mid E(\Phi_1 \mathcal{U} \Phi_2) \mid A(\Phi_1 \mathcal{U} \Phi_2) \mid z.\Phi$$

where q is an atomic proposition, $c, d \in \mathcal{N}$, x, y are real variables. Given a TLP $\mathcal{P}$ and a TCTL formula Φ, the satisfaction relation $\models$ is defined inductively as follows (here $p(\mathbf{v})$ is a ground atom).

- $\mathcal{P}, p(\mathbf{v}) \models q$ iff $q \in P(p)$ (where P is a function that labels each predicate in $\mathcal{P}$ with a set of atomic propositions).
- $\mathcal{P}, p(\mathbf{v}) \models x + c \leq y + z$ iff $\mathcal{R}, \mathbf{v} \models x + c \leq y + d$.
- $\mathcal{P}, p(\mathbf{v}) \models \neg\Phi$ iff $\mathcal{P}, p(\mathbf{v}) \not\models \Phi$.
- $\mathcal{P}, p(\mathbf{v}) \models \Phi_1 \vee \Phi_2$ iff $p(\mathbf{v}) \models \Phi_1$ or $p(\mathbf{v}) \models \Phi_2$.
- $\mathcal{P}, p(\mathbf{v}) \models E(\Phi_1 \mathcal{U} \Phi_2)$ iff there exists a ground derivation through evolution or system clauses from $p(\mathbf{v})$ to a ground atom $p'(\mathbf{v}')$ such that $\mathcal{P}, p'(\mathbf{v}') \models \Phi_2$, every other ground atom $p''(\mathbf{v}'')$ in the ground derivation satisfies $\Phi_1 \vee \Phi_2$ and if $p''(\mathbf{v}'')$ and $p''(\mathbf{v}'' + \delta)$ (where $\delta \in \mathcal{R}$) be two ground atoms in the derivation such that $p''(\mathbf{v}'' + \delta)$ is a resolvent of $p''(\mathbf{v}'')$ and a clause, then for all $0 \leq \delta' < \delta$, $\mathcal{P}, p''(\mathbf{v}'' + \delta') \models \Phi_1 \vee \Phi_2$.
- $\mathcal{P}, p(\mathbf{v}) \models A(\Phi_1 \mathcal{U} \Phi_2)$ iff for each ground derivation $\mathcal{G}$ starting from $p(\mathbf{v})$ through evolution or system clauses such that there exists a ground atom $p'(\mathbf{v}')$ in $\mathcal{G}$ such that $\mathcal{P}, p'(\mathbf{v}') \models \Phi_2$, every other ground atom $p''(\mathbf{v}'')$ in the ground derivation $\mathcal{G}$ satisfies $\Phi_1 \vee \Phi_2$ and if $p''(\mathbf{v}'')$ and $p''(\mathbf{v}'' + \delta)$ (where $\delta \in \mathcal{R}$) be two ground atoms in the derivation $\mathcal{G}$ such that such that $p''(\mathbf{v}'' + \delta)$ is a resolvent of $p''(\mathbf{v}'')$ and a clause then for all $0 \leq \delta' < \delta$, $\mathcal{P}, p''(\mathbf{v}'' + \delta') \models \Phi_1 \vee \Phi_2$.
- $\mathcal{P}, p(\mathbf{v}) \models z.\Phi$ iff $\mathcal{P}, p(\mathbf{v}[0/z]) \models \Phi$.

Let Φ_1 and Φ_2 be TCTL formulas. As usual, let $[\Phi]$ denote the denotation of Φ, i.e., $[\Phi] = \{p(\mathbf{v}) \mid \mathcal{P}, p(\mathbf{v}) \models \Phi\}$. We show that for each TCTL formula Φ and a TLP $\mathcal{P}$, the denotation of Φ, $[\Phi]$ over $\mathcal{P}$ can be represented by a finite set of generalized tuples.

Theorem 3.7 *For each TCTL formula Φ, and a TLP $\mathcal{P}$, its denotation can be described by a finite set of generalized tuples.*

Proof. We proceed by structural induction on TCTL formulas. For atomic propositions q the set of generalized tuples is given by $\{p(\mathbf{x}) \longleftarrow true \mid q \in P(p)\}$. The cases for the disjunction and conjunction are easy. We prove the theorem for the case of exists until. The remaining cases are similar.

Suppose that the formula Φ is given by $E(\Phi_1 \mathcal{U} \Phi_2)$. We construct a TLP $\mathcal{P}^\Phi$ such that the least model of $\mathcal{P}^\Phi$ is the same as $[\Phi]$. Given $[\Phi_1]$ and $[\Phi_2]$ as a finite set of generalized tuples, we first compute for each predicate p, the set S_p of all ground atoms $p(\mathbf{v})$ such that there exists a $\delta \geq 0$ such that for all $0 \leq \varepsilon \leq \delta$, $\mathcal{P}, p(\mathbf{v} + \varepsilon) \models \Phi_1 \vee \Phi_2$. Let $p(\mathbf{x}) \longleftarrow \varphi_i$ be the generalized tuple defining the predicate p in $[\Phi_i]$ $(i \in \{1, 2\})$. Then S_p is given by $p(\mathbf{u})$ such that

$$\mathbf{u} \in [\exists \delta \geq 0 \forall \varepsilon (0 \leq \varepsilon \leq \delta \Longrightarrow (\varphi_1[\mathbf{x} + \varepsilon/\mathbf{x}] \vee \varphi_2[\mathbf{x} + \varepsilon/\mathbf{x}]))].$$

I.e.,

$$\mathbf{u} \in [\exists \delta \geq 0 \neg \exists \varepsilon (0 \leq \varepsilon \leq \delta \wedge \neg \varphi_1[\mathbf{x} + \varepsilon/\mathbf{x}] \wedge \neg \varphi_2[\mathbf{x} + \varepsilon/\mathbf{x}]))].$$

I.e.,

$$\mathbf{u} \in [\exists \delta \geq 0 \neg \exists \varepsilon \xi].$$

where $\xi \equiv (0 \leq \varepsilon \leq \delta \wedge \neg \varphi_1[\mathbf{x} + \varepsilon/\mathbf{x}] \wedge \neg \varphi_2[\mathbf{x} + \varepsilon/\mathbf{x}]$. We can now convert ξ to a disjunctive normal form. Now, for each disjunct of the form ζ, we can eliminate the existential quantifier $\exists \varepsilon$ using variable elimination algorithms like Fourier's algorithm [MS98]. Let the quantifier free formula obtained be ξ'. We now negate ξ' and convert it to a disjunctive normal form. In this way, we get a constraint $\exists \delta \geq 0 \bigvee_{i=1}^{m} \eta_i$ such that $S_p = \{p(\mathbf{u}) \mid \mathcal{R}, \mathbf{u} \models \exists \delta \geq 0 \bigvee_{i=1}^{m} \eta_i\}$. Now, we construct the program $\mathcal{P}^\Phi$ as follows. For each evolution or system clause in $\mathcal{P}$, we create m clauses $p(\mathbf{x}) \longleftarrow p'(\mathbf{x}') \wedge \varphi \wedge \eta_i \wedge z = \delta$ (even though system clauses do not have the increment variable z we can add the constraint $\eta_i \wedge z = \delta$). Also, for each predicate p, we add the generalized tuple $p(\mathbf{x}) \longleftarrow \varphi_2$ (i.e., the generalized tuple defining p in $[\Phi_2]$). Now, we show that the denotation of $E(\Phi_1 \mathcal{U} \Phi_2)$ is the same as the least model of $\mathcal{P}^\Phi$. Let $p(\mathbf{v}) \in lm(\mathcal{P}^\Phi)$, where $lm(\mathcal{P}^\Phi)$ is the least model of $\mathcal{P}^\Phi$. Then there exists a ground derivation $\mathcal{G}$ through the clauses $C_1, \ldots, C_l$ starting from $p(\mathbf{v})$ that succeeds (i.e., C_l is a generalized tuple). Let $p''(\mathbf{v}'')$ be the ground atom in that ground derivation that resolves through C_l. Then $\mathcal{P}, p''(\mathbf{v}'') \models \Phi_2$. Now each ground atom in the ground derivation satisfies $\Phi_1 \vee \Phi_2$ (otherwise, it would have failed, since it would not have satisfied any η_i for $1 \leq i \leq m$ that are in the body of the clauses). Now consider two ground atoms $p_1(\mathbf{u})$ and $p_2(\mathbf{u} + \tau)$ such that the latter is a resolvent of the former through a clause in $\mathcal{P}^\Phi$ that is derived from an evolution clause in $\mathcal{P}$. Since $\mathcal{R}, \mathbf{u} \models \eta_i$ for some i (where $S_{p_1} = \{p_1(\mathbf{u}) \mid \mathcal{R}, \mathbf{u} \models \exists \delta \geq 0 \bigvee_{i=1}^{m} \eta_i\}$), therefore, for all τ' such that $0 \leq \tau' \leq \tau$, $\mathcal{P}, p_1(\mathbf{u} + \tau') \models \Phi_1 \vee \Phi_2$. Hence, $p(\mathbf{v}) \in [E(\Phi_1 \mathcal{U} \Phi_2)]$. Thus $lm(\mathcal{P}^\Phi) \subseteq [E(\Phi_1 \mathcal{U} \Phi_2)]$. Similarly, it can be shown that $[E(\Phi_1 \mathcal{U} \Phi_2)] \subseteq lm(\mathcal{P}^\Phi)$.

It can be shown that the least model of $\mathcal{P}^\Phi$ can be computed by a finite number of iterations of the immediate consequence operator. Hence the denotation of $E(\Phi_1 \mathcal{U} \Phi_2)$ can be described by a finite set of generalized tuples. ∎

Since each iteration of the immediate consequence operator requires only a finite amount of time, the computation of the least model (or the greatest model) of $\mathcal{P}^\Phi$ terminates. Hence the proof of Theorem 3.7 provides us with an algorithm for model checking for TCTL formulas. The computation of the least model can be made goal directed by using either using tabled resolution combined with the trim operator as done previously, or using magic sets transformation on the program $\mathcal{P}^\Phi$. Thus for example, if the denotations of Φ_1 and Φ_2 are given, we can check if $\mathcal{P}, init \models E(\Phi_1 \mathcal{U} \Phi_2)$ using a "local" algorithm.

3.13 Transient Behavior of Real Time Systems

In this section, we formulate a methodology for detecting transient behavior of timed logic processes. It is well known in control theory that underdamped linear time-invariant systems have both a transient and a steady-state response (see any standard textbook on control theory e.g., [Oga96]). Examples of such systems include from the mechanical mass-spring-dashpot systems to analog sensors and measuring equipments. Our aim in this section is to capture this notion of (under) damping in the context of real-time systems (modeled by timed logic processes). Detecting underdamping (or transient behavior) is useful for system identification which is an important problem in control theory. System identification involves automatically detecting the order of a given system as well as the nature of its damping. Thus if we can automatically determine that a linear time invariant system has a transient and a steady-state response, we can deduce that the system is underdamped.

We assume that real time systems are modeled as TLPs. We assume TLPs in which every clause is either an evolution clause or a system clause or an initial clause. Further we expand TLPs with alphabets to define labeled TLPs.

Definition 3.13 (Labeled TLP) *A* labeled TLP *is a TLP equipped with a (finite) alphabet* Σ *such that each system clause or initial clause is labeled by a letter (action)*[2] *from this alphabet (a letter may label several clauses).*

Let a, b, c range over Σ. In this section, whenever we speak of a TLP we will actually mean a labeled TLP.

Let $\mathcal{P}$ be a TLP. We say that a ground derivation $\mathcal{G} = init \longrightarrow g_1 \longrightarrow \ldots \longrightarrow g_m \longrightarrow \ldots$ of $\mathcal{P}$, where g_i is a ground atom, is an *advancing derivation* if g_1 is a ground resolvent of *init* through an initial clause of $\mathcal{P}$ and for each $i = 1, 2, \ldots$ g_{2i} is a ground resolvent of g_{2i-1} through an evolution clause and for each $i = 1, 2, \ldots$, g_{2i+1} is a resolvent of g_{2i} through a system clause.

Let $\mathcal{G} = init \longrightarrow g_1 \longrightarrow \ldots \longrightarrow g_i \longrightarrow \ldots$ be a (infinite) advancing ground derivation of $\mathcal{P}$ starting from *init*. We say that $\mathcal{G}$ is labeled by an omega-word $u \in \Sigma^\omega$, where $u = u_0, u_1, \ldots$ ($u_i \in \Sigma$), iff g_1 is the ground resolvent of *init* through a clause in $\mathcal{P}$ labeled u_0 and for each $i = 1, 2, \ldots$, g_{2i+1} is the ground resolvent of g_{2i} through a clause $\mathcal{P}$ labeled u_i.

We say that an omega-word $u \in \Sigma^\omega$ is accepted by a TLP $\mathcal{P}$ if there exists an infinite advancing (ground) derivation $\mathcal{G}$ of $\mathcal{P}$, starting from *init*, that is labeled by u.

Definition 3.14 (Transience) *We say that a timed logic process* $\mathcal{P}$ *is transient if there exists a word* $u \in \Sigma^\omega$ *accepted by* $\mathcal{P}$ *such that all (ground) infinite advancing derivations of* $\mathcal{P}$ *labeled by* u *converge.*

[2] We do not consider any "silent" action here.

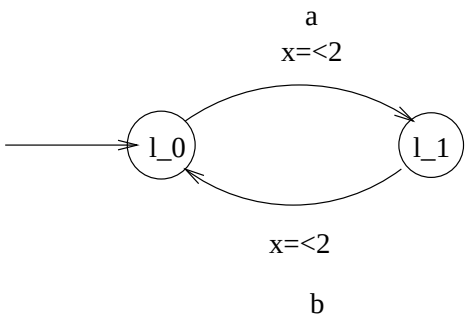


Figure 3.13: Illustrating transience.

Intuitively, a TLP is transient if it has at least one transient behavior where by transient behavior we mean a omega-word accepted by the TLP that does not label a divergent (ground) advancing derivation. So such a behavior is observed "initially" but "disappears" with the passage of time (since all advancing ground derivations labeled by it converge). Note that the TLP corresponding to the timed automaton shown in Figure 3.13 is not convergent (convergence for timed automata is defined in Chapter 4), but is transient (consider the word $(ab)^\omega$).

3.13.1 Detecting Transience

Before we delve into details of the algorithm for detecting transience, let us introduce the concept of a nonground Büchi automaton induced by a TLP. The notion of nonground Büchi automata is similar to that of concurrent constraint automata [FP93]. We assume as in the previous section that the TLPs that we consider consist only of evolution, system and initial clauses. We also assume that each TLP is equipped with a finite alphabet Σ labeling the system and initial clauses. Let $\mathcal{P}$ be a TLP. A nonground Büchi automaton induced by $\mathcal{P}$ is a five-tuple $\mathcal{P}^{\mathcal{A}} = \langle S, S_0, \Sigma, \longrightarrow, F \rangle$ where

- $S = \{\langle p(\mathbf{x}), trim(\varphi)\rangle \mid \langle p(\mathbf{x}), \varphi\rangle$ is a reachable nonground state of $\mathcal{P}\}$ is the set of states.
- $S_0 = init$ is the initial state.
- Σ is the finite alphabet associated with $\mathcal{P}$.
- $\longrightarrow = \stackrel{t}{\longrightarrow} \cup \{\stackrel{a}{\longrightarrow} \mid a \in \Sigma\}$ is the transition relation where $\stackrel{a}{\longrightarrow} \subseteq S \times S$. For $s, s' \in S$ and $a \in \Sigma$, we write $s \stackrel{a}{\longrightarrow} s'$ if s' is a resolvent of s through a clause of $\mathcal{P}$ labeled a. For $s, s' \in S$, we write $s \stackrel{t}{\longrightarrow} s'$ if s' is a resolvent of s through a evolution clause.
- $F \subseteq S$ is a set of accepting states

In addition, we associate with $\mathcal{P}^{\mathcal{A}}$ Büchi acceptance conditions (see [Saf88]). Note that due to Lemma 3.5, S is finite.

3.13.2 Construction of Nonground Büchi Automata

Given a TLP $\mathcal{P}$, we construct a TLP $\mathcal{Q}$ such that the following holds:

– There exist two nonground Büchi automata $\mathcal{Q}^{\mathcal{A}_1}$ and $\mathcal{Q}^{\mathcal{A}_2}$ such that $\mathcal{P}$ is transient if and only if the product automata $\mathcal{Q}^{\mathcal{A}_1} \times \mathcal{Q}^{\mathcal{A}_2^c}$ is nonempty, where $\mathcal{Q}^{\mathcal{A}_2^c}$ denotes the complement of $\mathcal{Q}^{\mathcal{A}_2}$.

Intuitively, (as we will see below) the automaton $\mathcal{Q}^{\mathcal{A}_1}$ accepts all behaviors (infinite words) that are accepted by $\mathcal{P}$, while the automaton $\mathcal{Q}^{\mathcal{A}_2}$ accepts only those behaviors of $\mathcal{P}$ that label a divergent advancing (ground) derivation. So the problem of detecting transience now reduces to the problem of language containment between the two automata (more precisely, whether the language of $\mathcal{Q}^{\mathcal{A}_1}$ is not contained in the language of $\mathcal{Q}^{\mathcal{A}_2}$; in fact this can be treated as a non-universality problem).

Given a TLP $\mathcal{P}$, we construct the TLP $\mathcal{Q}$ (as stated above) as follows. The set of predicates of $\mathcal{Q}$ are the same as that of $\mathcal{P}$ except for the fact that each predicate is now $n+1$-ary (if the corresponding predicate in $\mathcal{P}$ was n-ary). The initial predicate is *init* is the same as that of $\mathcal{P}$. The set of atomic propositions AP is the same as that of $\mathcal{P}$. The function λ assigning subsets of AP to predicates is the same as that of $\mathcal{P}$. The clauses of $\mathcal{Q}$ are constructed in the following way.

– The clauses in $\mathcal{Q}$ are the same as that in $\mathcal{P}$ except that the predicate in the body is now $n+1$-ary.

– For each system clause of $\mathcal{P}$ create two clauses of the form $p(\mathbf{x}) \longleftarrow p'(\mathbf{x}') \wedge \varphi \wedge x_{n+1} \geq 1 \wedge x'_{n+1} = 0$ and $p(\mathbf{x}) \longleftarrow p'(\mathbf{x}') \wedge \varphi \wedge x_{n+1} < 1 \wedge x'_{n+1} = x_{n+1}$ where φ is the constraint of the clause $\mathcal{P}$ (note that the predicates in both the clauses are $n+1$-ary.)

Now given $\mathcal{Q}$ the two nonground automata $\mathcal{Q}^{\mathcal{A}_1}$ and $\mathcal{Q}^{\mathcal{A}_2}$ induced by $\mathcal{Q}$ can be specified simply by specifying the set of accepting states (note that all the other components are same in both the automata). Let F_1 and F_2 be the set of accepting states for $\mathcal{Q}^{\mathcal{A}_1}$ and $\mathcal{Q}^{\mathcal{A}_2}$ respectively. Then $F_1 = S$ and $F_2 = \{\langle p(\mathbf{x}), \varphi\rangle \in S \mid \mathcal{R} \models \varphi \wedge x_{n+1} \geq 1\}$ where S is the (common) set of states of $\mathcal{Q}^{\mathcal{A}_1}$ and $\mathcal{Q}^{\mathcal{A}_2}$.

Theorem 3.8 *A timed logic process $\mathcal{P}$ is transient (i.e., has a transient behavior) if and only if the product automaton $\mathcal{Q}^{\mathcal{A}_1} \times \mathcal{Q}^{\mathcal{A}_2^c}$ induced by the TLP $\mathcal{Q}$ as constructed above is non-empty. Here $\mathcal{Q}^{\mathcal{A}_2^c}$ is the complement of $\mathcal{Q}^{\mathcal{A}_2}$.*

Proof. $\Longrightarrow$-part: Suppose that $\mathcal{P}$ is transient. Let $u \in \Sigma^\omega$ represent a transient behavior of $\mathcal{P}$. Let us consider a non-ground advancing derivation $\mathcal{NG}$ of $\mathcal{Q}$ starting from *init* labeled by u (A non-ground advancing derivation is defined in the same way as a ground advancing derivation; the labeling for nonground derivations is in the same way as that for the ground counterpart). Now it can be seen that after some point, all states in this nonground derivation will be of the form $\langle p(\mathbf{x}), \varphi\rangle$ for some $p \in \mathit{Pred}$ where φ is a constraint such that $\mathcal{R} \not\models \varphi \wedge x_{n+1} \geq 1$. Hence such a non-ground derivation (a trajectory of $\mathcal{Q}^{\mathcal{A}_2}$) is not accepted by $\mathcal{Q}^{\mathcal{A}_2}$. But this behavior is accepted by $\mathcal{Q}^{\mathcal{A}_1}$. Hence the non-emptiness result follows.

$\Longleftarrow$-part: Suppose that $\mathcal{P}$ is not transient. Then for each word u that it accepted by $\mathcal{P}$, there exists an infinite advancing divergent (ground) derivation of $\mathcal{P}$ labeled by u. Now any word that is accepted by $\mathcal{P}$ is also accepted by the nonground automaton $\mathcal{Q}^{\mathcal{A}_1}$. Since there exists an infinite advancing divergent ground derivation of $\mathcal{P}$ labeled by u, there exists a run of $\mathcal{Q}_2^{\mathcal{A}}$ in which a states of the form $\langle p(\mathbf{x}), \varphi\rangle$ where $\mathcal{R} \models \varphi \wedge x_{n+1} \geq 1$ are encountered infinitely often. This is because the value of x_{n+1} increases to become 1 or more after it is reset infinitely

many times. Thus u is accepted by the automaton $\mathcal{Q}^{\mathcal{A}_2}$. Hence the language of the automaton $\mathcal{Q}^{\mathcal{A}_1}$ is included in that of $\mathcal{Q}^{\mathcal{A}_2}$. So the emptiness of $\mathcal{Q}^{\mathcal{A}_1} \times \mathcal{Q}^{\mathcal{A}_2^c}$ follows. ‖

Note that both the automata $\mathcal{Q}^{\mathcal{A}_1}$ and $\mathcal{Q}^{\mathcal{A}_2}$ have exponentially (in n, where n is the number of real variables of $\mathcal{P}$) many states in the worst case. Hence $\mathcal{Q}^{\mathcal{A}_1} \times \mathcal{Q}^{\mathcal{A}_2^c}$ can have doubly exponential (in n) states in the worst case (due to the complementation for Büchi Automaton [Saf88]). Hence, using standard techniques from automata theoretic verification [VW86b], the (non) emptiness test can be done in EXPSPACE.

3.14 Related Work

Logic-based methods for specification and verification are slowly gaining popularity. In the past few years there has been a lot of work on model checking using deductive methods [RRR+97a, GGV99]. While most of these works have been focussed on finite state systems, there has also been substantial work on verification of integer-valued and parameterized systems using methods based on logic [FR96, FP93, RKR+00]. Bjorner et.al. [BBC+96] use the theorem prover STEP to verify real time systems.

The works from the logic programming, theorem proving and database community that come closest to our work are [CDD+98, GP97, Fri98, Urb96]. In [CDD+98], real time systems were translated into constraint logic programs. But no detailed model checking results based on such a translation has been provided. Gupta and Pontelli in [GP97] have been able to verify several interesting properties of real time systems. In contrast with automated model checking methods, they rely on the programmer to write a "driver" routine to identify the finite number of finite repeating patterns in the infinite strings accepted by a timed automaton. In a recent paper, Gupta and Pontelli [GP99] describe definite clause grammar for the model checker UPPAAL. In an interesting approach, they use Horn logic denotational semantics framework for specifying, implementing and automatically verifying real time systems. But in their approach, they have to make sure that the verification of properties leads to finite computations. Gupta [Gup99] extends the methods of [GP97] to more general settings.

Fribourg in [Fri98] verifies real time systems specified by logic programs with gap constraints. This work only considers reachability problems for real time systems. Termination is always guaranteed here because a backward analysis is used (industrial-scale tools like UPPAAL use forward analysis in spite of a missing termination guarantee [LPY95a]).

Du, Ramakrishnan and Smolka [DRS99] extend XSB with the POLINE constraint library to verify real time systems. But they follow the same techniques as [SS95] and hence they also ensure termination using expensive splitting of constraints.

Urbina in [Urb96] identifies a class of CLP programs as hybrid automata without, however, establishing a formal connection with the standard model for timed systems. In fact, the semantics results in [Urb96] cannot be connected with liveness properties of timed automata, in contrast to our work on TLPs.

The works from the verification community that come closest to our work are [LPY95a, DT98, SS95]. The model checking method in [LPY95a] based on the rewrite tree can be viewed as a special case of our model checking procedure based on OLDT resolution extended to constraints. We have been able to model check for a logic which is strictly more expressive than that in [LPY95a]. Also, the model checker UPPAAL [BLL+96] does not seem to be able to model check for receptiveness properties that we have been able to model check for. The model

checking procedure in [LPY95b] is possibly non-terminating (discussed above) while our model checking procedure, thanks to the trim operation, is guaranteed to terminate. In [DT98] Daws and Tripakis present a global model checking procedure for real time systems. In contrast, ours is a local one. Also, their method can be used only for model checking "reachability" properties like safety while we have given methods to deal with unbounded liveness properties. Sokolsky and Smolka [SS95] present a local model checker for real time systems. But, as mentioned in the Introduction of this chapter, their method for ensuring termination is based on an expensive "splitting" of constraints. We discuss the computational cost of splitting constraints in Chapter 4 where we consider negation. The model checking procedure of [SS95] is essentially tableau-based where sideways information passing [Ram91] cannot be used. On the other hand, utilizing the sideways information passing in the TLP clauses, we can deal with disjunction (conjunction) without splitting constraints. It is also evident from the algorithm in the definition of trim that it completely avoids splitting of constraints. We have not received any report on the performance of the model checking procedure in [SS95] on any practical example. The characterization of TCTL properties in terms of model-theoretic semantics of constraint query language programs has not been done before.

In [HHWT95], the authors describe HyTech, a model checker for hybrid systems. HyTech is based on two model checking procedures— one top-down and another bottom-up. Both procedures are global. While the bottom-up procedure is guaranteed to terminate for timed systems, the top-down procedure is possibly non-terminating even for timed systems. In contrast, in this chapter, we have provided a local top-down model checking procedure that is guaranteed to terminate for timed systems specified by TLPs.

The notion of transient (underdamped) behavior of real time systems and algorithms for detecting the same has, to the best of the knowledge of the authors, not been studied before.

In summary, we have demonstrated in this chapter how uniform framework can deal with the different types of problems arising in the process of modeling and verification of timed systems.

Chapter 4

The Stratified μ-calculus

4.1 Introduction

Symbolic model checking for systems with variables over an infinite numeric domain, e.g. for timed or hybrid systems, has become an important topic of research (see e.g. [ACD93, ACHH93, AD94, AH97, CDD+98, CJ98, CMN+98, CP98a, DP99b, DT98, Esp97, HNSY94, KMP96, GP97, MP99, MP00b, Tri99]). In this context, 'symbolic' usually refers to a representation of a set of states (i.e. of tuples of numbers) by a disjunctive constraint, e.g. a set of conjunctions of inequalities between arithmetic expressions over variables (a set standing for the disjunction of its elements). In this chapter, we single out a new class ("Sμ") of temporal properties with two symbolic model checking procedures, one based on backward analysis and the other based on forward analysis, that are both suitable for disjunctive constraints as the data structure for representing and manipulating sets of states.

We define the *stratified* μ-calculus Sμ as the subset of all μ-calculus formulas (built up with the Boolean operators, the existential predecessor operator EX and and the least fixpoint combinator μ) whose subformulas can be 'stratified'; i.e., the is-subformula relation can made a partial order that is strict for negation. This restriction excludes the expression of alternation, of the universal predecessor operator AX and of the greatest fixpoint combinator ν. The fragment of the alternation-free μ-calculus we thus obtain subsumes the so-called safety logic STL (see e.g. [AH99]) that again subsumes the EF-logic considered in [Esp97].

The Sμ properties are computable in symbolic backward analysis (essentially a least fixpoint iteration based on the existential predecessor operator EX) that uses only 'good' operations on disjunctive constraints. I.e., the application of the fixpoint operator requires the *disjunction* of two disjunctive constraints (representing each a set of states), a constant-time operation. In contrast, if it required the *conjunction* (as during a greatest fixpoint iteration) or the *complement* (e.g. for expressing the universal predecessor operator AX in terms of the existential one, EX), the corresponding implementation cost would grow as a function (quadratic resp. exponential) in the number of disjuncts in the constraints representing the state sets.

The above observation has been our original incentive to define Sμ (i.e. as a candidate for a temporal logic with symbolic model checking procedures that are well-suited for disjunctive constraints). The other motivation of Sμ is the natural generalization of STL; Sμ is defined by the same general syntactic restriction to the μ-calculus that, if applied to the fragment corresponding to CTL, yields exactly STL.

Our technical contribution is a novel 'symbolic forward analysis' method for checking Sμ formulas. This method is based on our characterization of Sμ properties as *perfect models* of constraint query language programs and on our *tabled-resolution* procedure for constraint query language programs with the perfect-model semantics.

Forward analysis is sometimes preferable to backward analysis; a thorough discussion can be found in [HKQ98]. Our procedure is a symbolic forward analysis in a sense different from the one formalized in [HKQ98] (this is already clear by the result in [HKQ98] that the Sμ property $\mathrm{EF}(p \wedge \mathrm{EF}(q) \wedge \mathrm{EF}(r))$ is not computable by 'symbolic forward analysis'). Both forms of forward analysis are essentially a least-fixpoint iteration of the direct-successor operator *post* applied to a constraint representation of a set of states. In [HKQ98], a constraint is viewed monolithically (this corresponds to a setting where a set of states is represented by a constraint in normal form, implemented e.g. by a BDD). Our procedure operates on the constraints inside of a disjunctive constraint, i.e. its disjuncts. Note that the disjuncts generally represent infinite sets of states, which makes our procedure different from an enumerative procedure (and requires the management of formulas with free variables).

Tabled resolution is originally an execution strategy for logic programs with negation; see [CW96, SI88, TS86b]. We have not, however, found a tabled-resolution procedure for non-ground constraint queries wrt. the perfect-model semantics in the (yet quite extensive) literature. In the context of verification, tabled resolution has been used in [RRR$^+$97b] for ground programs (and finite-state systems).

The connection between Sμ properties and perfect models of constraint query language programs is perhaps of intrinsic interest; its role in this chapter is a quite pragmatic one. Namely, the connection helps us to concisely formulate and to formally prove correct our forward analysis procedure.

Convergence (a.k.a. zenoness or timelock) [AH97, HNSY94, Tri99] for timed automata is an Sμ property. Therefore, one can apply either of our two general symbolic model checking methods, the backward or the forward one. The two existing specialized algorithms for checking convergence [HNSY94, Tri99] are instances thereof. Thus, our work helps to situate the two algorithms within a general model-theoretic and proof-theoretic context.

4.2 Stratification

We first recall the syntax of modal mu calculus. The syntax of (closed) formulas φ of the modal μ-calculus is given below.

$$\varphi \ ::= \ p \mid x \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \diamond\varphi \mid \mu x.\, \varphi$$

Here p is an atomic proposition, x is a variable, $\diamond$ is the next operator (written EX in CTL syntax), and in $\mu x.\varphi$, φ is monotone in x (i.e., all occurrences of the free variable x in φ lie in a scope of an even number of negation).

Sometimes we use the formulas $\nu x.\varphi(x)$ and $\Box\varphi$ informally as abbreviations for $\neg\mu x.\neg\varphi(\neg x)$ and $\neg\diamond\neg\varphi$, respectively.

The Fisher-Ladner closure $cl(\varphi)$ of a formula φ is the smallest set of formulas containing φ such that

– if $\psi \in cl(\varphi)$ and ψ' is a subformula of ψ then $\psi' \in cl(\varphi)$, and

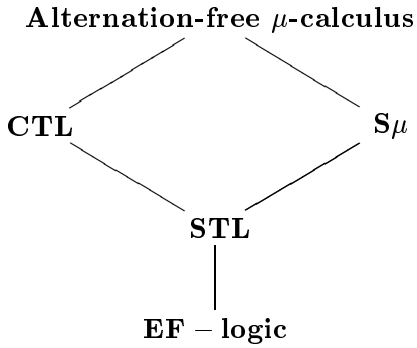


Figure 4.1: Situating the expressiveness of Sμ

– if $\mu x.\psi(x) \in cl(\varphi)$ then $\psi(\mu x.\psi(x)) \in cl(\varphi)$.

Here, for a formula $\mu x.\psi(x)$, the formula $\psi(\mu x.\psi(x))$ is obtained from $\psi(x)$ by replacing each free occurrence of x with $\mu x.\psi(x)$.

Definition 4.1 (Stratified μ-calculus Sμ) *An Sμ formula is a closed formula φ of the modal μ-calculus such that the is-subformula relation over $cl(\varphi)$ can be made a partial order $\preceq$ that is strict wrt. to negation, i.e. $\varphi' \prec \neg\varphi'$.*

The above definition is equivalent to saying that there exists a *stratification* function $\mathcal{S}$ assigning a natural number to each formula in the closure $cl(\varphi)$ such that

– if $\varphi'' \in cl(\varphi')$ then $\mathcal{S}(\varphi'') \leq \mathcal{S}(\varphi')$, and

– $\mathcal{S}(\varphi') < \mathcal{S}(\neg\varphi')$ for all $\varphi' \in cl(\varphi)$.

Thus, the stratified μ-calculus Sμ consists of all *stratifiable* formulas of the modal μ-calculus (in the syntax given above, i.e. with least fixpoints and Boolean set operators including the complement, but *without* greatest fixpoints).

Figure 4.1 situates the expressiveness of Sμ relative to the alternation-free μ-calculus, CTL and STL. We recall the definition of the syntax of STL and of its sublogic, the EF-logic.

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \mathrm{EX}(\varphi) \mid \mathrm{EF}(\varphi) \qquad \text{(EF-logic)}$$

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \mathrm{EX}(\varphi) \mid \varphi \mathrm{EU} \varphi' \qquad \text{(STL)}$$

To see that Sμ is subsumed by the alternation-free μ-calculus, observe that nesting of fixpoints μ and ν (where ν is expressed in terms of μ and negation) requires strict decreasing of the value of the stratification function; the value of the stratification for a fixpoint formula must, however, be the same for its one-step unfolding.

The example of the formula $\mu x.p \vee \neg\langle\rangle\neg x$ (abbreviated $\mu x.p \vee \Box x$, and written $\mathrm{AF}(p)$ in CTL syntax) of the alternation-free μ-calculus shows that Sμ is strictly less expressive. To see that φ is not an Sμ-formula, suppose that there exists a stratification function $\mathcal{S}$ showing that φ is an Sμ-formula. Let $\psi(x)$ be the formula $p \vee \neg\langle\rangle\neg x$. Then $\mathcal{S}(\mu x.\psi(x)) \geq \mathcal{S}(\psi(\mu x.\psi(x))) \geq \mathcal{S}(\neg\langle\rangle\neg\mu x.\psi(x)) > \mathcal{S}(\langle\rangle\neg\mu x.\psi(x)) \geq \mathcal{S}(\neg\mu x.\psi(x)) > \mathcal{S}(\mu x.\psi(x))$, which is a contradiction.

To see that Sμ is incomparable with CTL in terms of expressiveness, observe that (1) every satisfiable formula of Sμ has a model whose paths are all finite; the CTL formula $\mathrm{EG}(p)$, for

example, does not have such a model; (2) the Sμ formula $\mu x.p \vee \Diamond\Diamond x$ expresses the property "p is reachable in an even number of steps", which cannot be expressed in CTL.

Our motivation for Sμ stems from infinite-state systems with generally undecidable model checking problems. In fact, the following proposition shows that the restriction of the alternation-free μ-calculus to Sμ does not trade with a decrease of the theoretical complexity of model checking for finite-state systems. As in the alternation-free μ-calculus; the problem is hard for P even if the formula is fixed. The existing P-hardness proofs for the alternation-free μ-calculus, however, reduce the alternating graph-reachability problem, using the $\Box$ modality in an essential way, and cannot be carried over directly.

Proposition 4.1 (Finite-state systems) *The program complexity of Sμ model checking is P-complete.*

Proof. We reduce the monotone circuit value problem (MCVP), well-known P-complete problem. An instance of MCVP is a sequence variables $X_1, \ldots, X_n$ of boolean equations of the form $X_i = \mathit{true}$, $X_i = \mathit{false}$, $X_i = X_j \wedge X_k$ or $X_i = X_j \vee X_k$ where for all equation of the last two forms we have $i > \max(j,k)$, such that the value of a varaible does not depend on itself. The question we ask is whether the value of X_n is *true*.

Given such an instance I of MCVP, we construct a Kripke structure $K = \langle S, \rightarrow, L\rangle$ as follows. We define the set of atomic propositions as $\{p, p_1, p_2\}$. The set S of states consists of the variables $X_1, \ldots, X_n$ and their copies (two for each variable) $X_1^1, X_1^2, \ldots, X_n^1, X_n^2$. The labeling function $L : S \rightarrow 2^{\{p,p_1,p_2\}}$ is defined as follows. $L(X_i) = \{p\}$ iff $X_i = \mathit{true}$ is in I. For all $i = 1, \ldots, n$ we define $L(X_i^1) = \{p_1\}$ and $L(X_i^2) = \{p_2\}$. The transition relation $\rightarrow$ is defined as follows. For all equations $X_i = X_j \vee X_k$ we have $X_i \rightarrow X_j$ and $X_i \rightarrow X_k$, and for all equations $X_i = X_j \wedge X_k$ we have $X_i \rightarrow X_i^1$, $X_i \rightarrow X_i^2$ $X_i^1 \rightarrow X_j$ and $X_i^2 \rightarrow X_k$. Now it is easy to see that the state X_n in the structure K satisfies the formula

$$\mu x.p \vee \Diamond x \vee (\Diamond(p_1 \wedge \Diamond x) \wedge (\Diamond(p_2 \wedge \Diamond x)))$$

if and only if the value of the variable X_n in the circuit is *true*. ∎

4.3 Backward Analysis

In this section, we define a hierarchy of three kinds of procedures based on backward analysis that correspond to the three 'safety logics' EF-logic, STL and Sμ, respectively. These procedures are essentially least-fixpoint iterations for a fixpoint operator that is derived from the direct-predecessor operator *pre* in three different ways. We formalize the setting by defining directly the three families of sets of states that can be computed.

We fix a transition system $\mathcal{T} = \langle \Sigma, \longrightarrow\rangle$ with the set of states Σ and the transition relation $\longrightarrow$ (and the corresponding predecessor operator *pre* over sets of states).

Given a set of atomic propositions p, we fix a corresponding set of *base sets*; i.e., for every atomic proposition p there exists a base set $b \subseteq \Sigma$ that is the interpretation of p.

The *least-fixpoint closure* $F^\star(S)$ of the set S under a given operator F on sets is the least fixpoint of the function $\lambda x.(S \cup F(x))$.

We write $S \cap F$ for the operator $\lambda x.(S \cap F(x))$, $F \circ F'$ for the functional composition of the two operators F and F', and $F \cup F'$ their pointwise union.

Definition 4.2 ("computable by backward analysis") *A set of states S is lfp-computable if it is*

- *one of the given base sets,*
- *the union, intersection or complement of lfp-computable sets, or*
- *of the form $Pre(S)$ or of the form $Pre^\star(S)$*
  where S is an lfp-computable set and the operator Pre is formed in the following way (possibly using some other lfp-computable set S').

Case 1 $Pre ::= pre$

Case 2 $Pre ::= pre \mid S' \cap pre$

Case 3 $Pre ::= pre \mid S' \cap Pre \mid Pre \circ Pre' \mid Pre \cup Pre' \mid Pre \cap Pre'$

Proposition 4.2 *The sets of states expressed by the temporal properties in EF-logic, STL and $S\mu$ are exactly the lfp-computable sets in Case 1, 2 and 3, respectively.*

Proof (by structural induction). Base sets correspond to atomic propositions; union, intersection and complement of lfp-computable-sets correspond to disjunction, conjunction and negation in the corresponding logic. If a set S corresponds to a formula φ then $pre(S)$ corresponds to $\mathrm{EX}(\varphi)$ and $pre^\star(S)$ to $\mathrm{EF}(\varphi)$ in EF-logic, $true\mathrm{EU}\varphi$ in STL (where *true* can be defined as $p \vee \neg p$), and $\mu x.\varphi \vee {<>}x$ in Sμ. The set $(S' \cap pre)^\star(S)$ corresponds to $\varphi'\mathrm{EU}\varphi$ in STL and to $\mu x.\varphi \vee \varphi' \wedge {<>}x$ in Sμ, where φ' is the formula corresponding to S'. In case of $Pre^\star(S)$ where Pre is defined using composition, union or intersection, the set is translated in an obvious way to a least-fixed-point formula of Sμ using respectively composition, disjunction or conjunction of respective subformulas. The only case requiring more argumentation is the translation from least-fixed-point formulas of Sμ to lfp-computable sets.

Given a formula $\mu x.\varphi$ in Sμ we first translate it to a guarded formula (where all variables appear in a scope of the ${<>}$ modality) by rewriting all unguarded variables to *false*, and then translate the result to the disjunctive normal form. Let us call the obtained formula $\mu x.\psi$. If this formula denotes a nonempty set of states, ψ must contain a disjunct that does not contain x (otherwise it can be translated to $S \cap \sim S$ for any base set S). Let S be the set corresponding to the disjunction of all such disjuncts. Since ψ is a stratified formula, the subformula x must belong to the same stratum as ψ and thus x does not occur in a scope of negation. Now it is easy to construct an operator Pre such that $\mu x.\psi$ defines exactly the set $Pre^\star(S)$. ∎

4.4 Perfect Models

In this section, we present a translation of Sμ properties to the perfect models of stratified constraint query language programs. The translation is reminiscent of the ones in [CP98b, DP99b, GGV98, RRR$^+$97b, GP97]. Here, however, the translation is done such that it yields *stratified* programs. Roughly, a program is stratified if the dependency relation between its predicates (where $p \preceq q$ means: "the predicate p calls the predicate q", or: "p is defined using q") can be made a partial order that is strict wrt. negation; i.e., $p \prec q$ if there is a clause of the form $p(x) \leftarrow \ldots \mathsf{not}(q(x)) \ldots$. A level mapping of a program is a mapping from its set of predicates to the natural numbers. The level of a predicate p, denoted by $level(p)$, is the

value of the predicate under the mapping. A constraint query language program is stratified if it has a level mapping such that in every clause of the form $p(\mathbf{x}) \longleftarrow B \wedge \varphi$, the level of the predicate of any atom occurring positively in B is less than or equal to that of p and the level of the predicate of any atom occurring negatively in B is less than the level of p.

The original definition of a *perfect model* of a stratified constraint query language program $\mathcal{P}$ is model-theoretic [Prz88]. An equivalent definition yields a direct construction of this model; the construction uses the complementation of the least-fixpoint closure of the direct-consequence operator $T_{\mathcal{P}}$ [ABW88]. Roughly, the proof of the theorem below as well as that of Theorem 4.2 is built around this construction. In the sequel, we assume that the constraint domain $\mathcal{D}$ that we consider admits quantifier elimination. For a program $\mathcal{P}$, let $B_{\mathcal{D}}$ denote the $\mathcal{D}$-base of a program $\mathcal{P}$. The formal definition of perfect model based on direct construction, given below, is taken from [ABW88]. An equivalent definition can also be found in [Prz88]. Before we define the perfect model of a constraint query language program $\mathcal{P}$, we need the definition of the $T_{\mathcal{P}}$ operator. The operator $T_{\mathcal{P}}$ is a mapping from $2^{B_{\mathcal{D}}}$ to itself. It is defined as follows. The atom $p(\mathbf{v}) \in T_{\mathcal{P}}(I)$ (for $I \subseteq B_{\mathcal{D}}$) iff for there exists a ground instance $p(\mathbf{v}) \longleftarrow \mathit{body}$ of a clause in $\mathcal{P}$ such that $I \models \mathit{body}$.

Definition 4.3 (Perfect Model) *Let $\mathcal{P}$ be a stratified constraint query language program with maximum predicate level k. Let $M_{-1} = \emptyset$. For $0 \leq j \leq k$ do the following. Let C_j be the completed lattice $\{M_{j-1} \cup S \mid S \subseteq \{p(\mathbf{v}) \in B_{\mathcal{D}} \mid level(p) = j\}\}$ under set inclusion. Let $T^j_{\mathcal{P}}$ be the restriction of the immediate consequence operator $T_{\mathcal{P}}$ to C_j. Let $M_j = T^j_{\mathcal{P}} \uparrow \omega$. Then M_k is called the perfect model of $\mathcal{P}$.*

We assume that we can represent the transition systems of interest as constraint query language programs. That is, we are able to define a predicate $\mathsf{trans}(\mathbf{s}, \mathbf{s}')$, $\mathsf{init}(\mathbf{s})$ and $p_{ap}(\mathbf{s})$ saying, respectively, that there is a transition from the state $\mathbf{s}$ to $\mathbf{s}'$, that $\mathbf{s}$ is an initial state, and that the atomic proposition ap holds in the state $\mathbf{s}$. Such representations (together with direct syntactic translations) are known for finite systems [CP98b, GGV98, RRR$^+$97b, SIR96], push-down systems [CP98b], concurrent programs (including integer-valued protocols) [DP99b], and timed and hybrid systems [MP00b].

Given a constraint query language program $\mathcal{P}$ defining the predicate trans, we translate an Sμ formula ψ to a constraint query language program $\mathcal{P}_\psi$ as follows. For each formula φ in $cl(\psi)$ we introduce a new predicate p_φ defined as follows.

$$
\begin{array}{lrcl}
(1) & p_{\varphi_1 \wedge \varphi_2}(\mathbf{s}) & \leftarrow & p_{\varphi_1}(\mathbf{s}), p_{\varphi_2}(\mathbf{s}) \\
(2) & p_{\varphi_1 \vee \varphi_2}(\mathbf{s}) & \leftarrow & p_{\varphi_1}(\mathbf{s}) \\
 & p_{\varphi_1 \vee \varphi_2}(\mathbf{s}) & \leftarrow & p_{\varphi_2}(\mathbf{s}) \\
(3) & p_{\neg\varphi}(\mathbf{s}) & \leftarrow & \mathsf{not}(p_\varphi(\mathbf{s})) \\
(4) & p_{<>\varphi}(\mathbf{s}) & \leftarrow & \mathsf{trans}(\mathbf{s}, \mathbf{s}'), p_\varphi(\mathbf{s}') \\
(5) & p_{\mu x.\varphi(x)}(\mathbf{s}) & \leftarrow & p_{\varphi(\mu x.\varphi(x))}(\mathbf{s})
\end{array}
$$

To simplify the presentation, we abbreviate the translation of Sμ formulas that are expressed in the syntax of EF-logic or STL.

$$
\begin{array}{lrcl}
(6) & p_{\mathrm{EF}\varphi}(\mathbf{s}) & \leftarrow & p_\varphi(\mathbf{s}) \\
 & p_{\mathrm{EF}\varphi}(\mathbf{s}) & \leftarrow & \mathsf{trans}(\mathbf{s}, \mathbf{s}'), p_{\mathrm{EF}\varphi}(\mathbf{s}') \\
(7) & p_{\varphi_1 \mathrm{EU} \varphi_2}(\mathbf{s}) & \leftarrow & p_{\varphi_2}(\mathbf{s}) \\
 & p_{\varphi_1 \mathrm{EU} \varphi_2}(\mathbf{s}) & \leftarrow & p_{\varphi_1}(\mathbf{s}), \mathsf{trans}(\mathbf{s}, \mathbf{s}'), p_{\varphi_1 \mathrm{EU} \varphi_2}(\mathbf{s}')
\end{array}
$$

Theorem 4.1 *An atom $p_\varphi(\mathbf{s})$ belongs to the perfect model of the program $\mathcal{P}_\varphi$ if and only if the Sμ formula φ is true of the state $\mathbf{s}$; or: the denotation of the predicate p_φ in the perfect-model semantics of the program $\mathcal{P}_\varphi$ is exactly the denotation of the Sμ formula φ wrt. the transition system $\mathcal{P}$,*

$$[\![p_\varphi]\!]_{pm(\mathcal{P}_\varphi)} = [\![\varphi]\!]_{\mathcal{P}}.$$

Proof. By structural induction on stratified mu-calculus formulas. ∎

4.5 Tabled Resolution

In this section, we present the symbolic forward analysis for Sμ formulas. Based on Theorem 4.1, we can formally define it as a procedure for stratified constraint query language programs $\mathcal{P}$. The input is a query consisting of an atom $p(\mathbf{x})$ and a constraint φ. The output, if it terminates, is a *complete* list of answer constraints $[\varphi_1, \ldots, \varphi_n]$. This means: for every tuple $\mathbf{v}$ of values for the argument tuple $\mathbf{x}$, the atom $p(\mathbf{v})$ lies in the perfect model of $\mathcal{P}$ and $\mathbf{v}$ is a solution of the constraint φ if and only if $\mathbf{v}$ is a solution of one of the answer constraints φ_i. In short: the denotation of the predicate p in the perfect-model semantics of the program $\mathcal{P}$ intersected with the set of solutions of φ is the set of solutions of all answer constraints, i.e. of the disjunction $\varphi_1 \vee \ldots \vee \varphi_n$,

$$[\![p]\!]_{pm(\mathcal{P})} \cap [\![\varphi]\!] = [\![\varphi_1 \vee \ldots \vee \varphi_n]\!].$$

If the program $\mathcal{P}$ arises from the translation given in the previous section (i.e., it is of the form $\mathcal{P}_\psi$ for an Sμ formula ψ) and the constraint φ describes the set of initial states, then tabled resolution starting with the query $\langle p(\mathbf{x}), \varphi\rangle$ corresponds to forward analysis (depth-first or breadth-first, depending on the selection strategy of the resolution procedure; here, we have formulated a depth-first procedure). The answer constraints then specify which of the initial states satisfy the Sμ formula ψ (possibly all of them, namely if the disjunction is equivalent to φ).

The procedure is based on tabled resolution (see e.g. [TS86b, SI88, CW96]), which we have extended to handle constraints and nonground constructive negation (i.e. with nonground queries).

The central data structure is a table T of answer constraints. An index in the extendible table is a query of the form $\langle p'(x), \varphi'\rangle$ (consisting of an atom $p'(\mathbf{x})$ and a constraint φ'); the corresponding field contains a list of answers to that query that have computed to far. Initially, the table has only one index, namely the original query $\langle p(\mathbf{x}), \mathit{true}\rangle$, whose field contains the empty list.

The basic idea of the procedure is simple. We start with the initial query $\langle p(\mathbf{x}), \mathit{true}\rangle$. Iteratively, we apply resolution steps, hereby creating (disjunctions of) new queries, each of the form $\langle p_1(\mathbf{x}_1) \wedge \ldots \wedge p_k(\mathbf{x}_k), \psi\rangle$. This goes on until no more steps are applicable (either because a query is failed or because an answer constraint has been derived). We store all answers to any query encountered so far in the table T (procedure *tabulate*). We reuse the tabled information whenever possible (case (b) in the procedure *extend*); this is crucial for avoiding infinite loops. The cases (a) and (b) in the procedure *extend* give an extension of the original tabling procedure [TS86b] to handle constraints. The cases (c) and (d) are an extension to handle constructive negation. Again the idea is simple: to get an answer to a negative query

$\neg p(\mathbf{x}), \varphi$ we first run the procedure for the positive query (it is important here that we collect *all* the answers for the positive query) and then negate the answer.

If the procedure terminates (which in general cannot be guaranteed, already for decidability reasons), then the table contains all answers to all sub-queries and to the original query $p(\mathbf{x})$.

The examples below give an intuition about the procedure. Example 4.1 shows how tabling helps avoiding infinite loops. Example 4.2 shows how answers to *all* subgoals are stored in the table (for all successful subderivations). Finally, Example 4.3 shows how the negation is handled.

Example 4.1 Consider the query $p(x)$ for the following program.

$$\begin{array}{lcl} p(x) & \leftarrow & x = 0. \\ p(x) & \leftarrow & x = 1. \\ p(x) & \leftarrow & p(x). \end{array}$$

We start with the structure consisting of one active node $p(x)$, and the table T containing one entry $T[p(x)] = [\,]$. After the first extension we obtain two answers $x = 0$ and $x = 1$ and one new node, again with the query $p(x)$, hence it is classified as a lookup node. Call this node v. The table is updated with $T[p(x)] = [x = 0, x = 1]$ and the lookup mapping gives $L(v) = [x = 0, x = 1]$. The second resolution step takes the first answer from this list, creates a new node $x = 0$ and moves the lookup pointer to the tail of the list, so now $L(v) = [x = 1]$. Since the solution $x = 0$ occurs already in the table, it is not added there. After the third resolution step the new node $x = 1$ is created and the value $L(v)$ is set to the empty list. At this point no more resolution steps are possible and the procedure terminates.

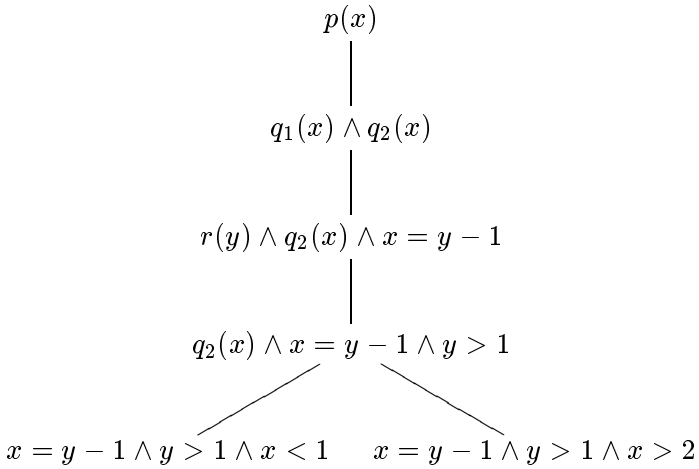


Figure 4.2: Computation tree for Example 4.2

Example 4.2 Consider the query $p(x)$ for the following program.

$$\begin{array}{rcl} p(x) & \leftarrow & q_1(x), q_2(x). \\ q_1(x) & \leftarrow & x = y - 1, r(y). \\ r(x) & \leftarrow & x > 1. \\ q_2(x) & \leftarrow & x < 1. \\ q_2(x) & \leftarrow & x > 2. \end{array}$$

The derivation tree for this query is shown on Figure 4.2. All nodes (except answers) in this tree are active. In the fourth node of this tree we obtain an answer $x = y - 1 \wedge y > 1$ to the subquery $r(y)$ and this answer is passed to the procedure *tabulate*. Before storing it, however, the formula $\exists x.\ x = y - 1 \wedge y > 1$ is normalized; in particular the existentially quantified x is eliminated and the new entry in the table is $T[r(y) \wedge x = y - 1] = [y > 1]$.

Note that the constraint $x = y - 1 \wedge y > 1$ is not only an answer to the subquery $r(y)$, but also to the subquery $q_1(x)$. In order to find it out, the procedure *tabulate* is called recursively to propagate the answer up in the tree. Then it is again normalized (this time the variable y, and not x, is existentially quantified) and stored: $T[q_1(x)] = [x > 0]$.

The same happens when we reach leaves of the tree. The two answers that we obtain have to be stored both in the entry for $q_2(x)$ and $p(x)$, thus we obtain $[x > 0 \wedge x < 1,\ \ x > 2]$ as the value for $T[p(x)]$ and $T[q_2(x) \wedge x = y - 1 \wedge y > 1]$.

Example 4.3 Consider the program from Example 4.2 together with a clause $p'(x) \leftarrow \neg p(x)$ and a query $p'(x)$. Since the tree from Figure 4.2 cannot be extended anymore (it is totally successful in the terminology of Stuckey [Stu91]), we can negate the answers collected for the query $p(x)$ and obtain correct answers for the query $\neg p(x)$ that (after transforming to the disjunctive normal form) we store in the table: $T[\neg p(x)] = [x \leq 0,\ \ x \geq 1 \wedge x \leq 2]$. (Note: since we always eliminate existential quantifiers, negating constraints does not introduce universal quantification; i.e., we do not have to treat queries with universally quantified variables.)

We will next describe the procedure in detail (see Figure 4.3). The current computation state of the procedure a triple $\langle F, T, L \rangle$. Here, F is a set of trees (intuitively, an SLDNF forest; each call to a negative goal starts a new tree). Each node of F is a goal of the form $\bigwedge_{i=1}^{n} l_i(\mathbf{x}_i) \wedge \varphi$ where $n \geq 0$. For $n = 0$ we say that $\bigwedge_{i=1}^{n} l_i(\mathbf{x}_i)$ is empty and the goal is an answer φ. For $n \geq 1$ we write such goal as $l_1(\mathbf{x}_1) \wedge R \wedge \varphi$ to indicate that $l_1(\mathbf{x}_1)$ is the selected literal and $R = \bigwedge_{i=2}^{n} l_i(\mathbf{x}_i)$ is the rest of the goal. (A literal $l(\mathbf{x})$ is either an atom $p(\mathbf{x})$ or a negated atom $\neg p(\mathbf{x})$.) An actual implementation will not keep the whole forest F but only the relevant parts of it; instead of having the procedure *tabulate* go up a path in F, one would use forward pointers in the table T; in this presentation, we will not go into such details.

The second component T of the current computation state is the table that we have discussed above. The third component L is a lookup mapping; it maps nodes of F to lists of constraints and is used to go through all answer constraints stored in the table T that are relevant for the given node.

Every node in the forest F is classified as an *active* or *lookup* node by procedure *classify*. Intuitively, active nodes are the nodes for which we have to compute answers; lookup nodes are the nodes for which the answers can be found in the table (roughly, the selected literals in lookup nodes are instances of the selected literals in active nodes). Furthermore, every node

is positive or negative depending on whether the selected literal is positive or negative; a node may be also marked as failure. The lookup mapping L is defined only for lookup nodes.

Definition 4.4 In a given current computation state $\langle F, T, L\rangle$, a node v in F is called a *extendible* node of type (a),(b), (c) or (d), respectively, if

(a) v is a positive active node, a leaf in F and not a failure node, or

(b) v is a lookup node and $L(v)$ is a nonempty list, or

(c) v is a negative active node without a companion node in F, or

(d) v is a negative active node that is not processed yet (see below) such that the computation for the companion node is done (see below).

A negative active node $\neg p(\mathbf{x}) \wedge R \wedge \varphi$ is *processed* if it is either marked as failure node or the list $T[\neg p(\mathbf{x}), \varphi]$ is nonempty. A computation for a positive node is *not* done if the tree rooted at this node contains an extendible node or it contains a lookup node v' such that the computation for the active node corresponding to v' is not done.

Definition 4.5 (Successful tabled derivation) A sequence $p(\mathbf{x}) \wedge \varphi, G_1 \wedge \varphi_1, \ldots, G_n \wedge \varphi_n$ is a *successful derivation* for the query $p(\mathbf{x}) \wedge \varphi$ wrt. the table T if: G_n is the empty goal, φ_n is a satisfiable constraint, and for all $i = 0, \ldots, n-1$ (where $G_0 = p(\mathbf{x})$)

- $G_i = q(\mathbf{x}) \wedge R$ and there exists a clause $q(\mathbf{x}) \leftarrow \mathit{body} \wedge \psi$ such that $G_{i+1} = \mathit{body} \wedge R$ and $\varphi_{i+1} = \varphi_i \wedge \psi$, or
- $G_i = q(\mathbf{x}) \wedge R$ and $G_{i+1} = R$ and there exists a constraint φ' such that $\varphi_i \models \varphi'$, $\langle q(\mathbf{x}), \varphi'\rangle$ is an index of T and there exists a member ψ of $T[\langle q(\mathbf{x}), \varphi'\rangle]$ such that $\varphi_{i+1} = \varphi_i \wedge \psi$.

A sequence $p(\mathbf{x}) \wedge R \wedge \varphi, G_1 \wedge R \wedge \varphi_1, \ldots, G_n \wedge R \wedge \varphi_n$ is a *successful subderivation* for $p(\mathbf{x}) \wedge \varphi$ wrt. T if $p(\mathbf{x}) \wedge \varphi, G_1 \wedge \varphi_1, \ldots, G_n \wedge \varphi_n$ is a successful derivation for $p(\mathbf{x}) \wedge \varphi$ wrt. T.

Since the original query appears as an index of the table T and the return statement refers to its entry, the theorem below implies that the procedure, if it terminates, returns the correct output according to its specification.

Theorem 4.2 (Correctness) *If the tabling procedure terminates then, for every query $\langle p(\mathbf{x}), \varphi\rangle$ or $\langle \neg p(\mathbf{x}), \varphi\rangle$ that occurs as an index in the table T, the entry of T at that index is a list of constraints $[\varphi_1, \ldots, \varphi_n]$ such that*

$$[\![p]\!]_{pm(\mathcal{P})} \cap [\![\varphi]\!] = [\![\varphi_1 \vee \ldots \vee \varphi_n]\!],$$

i.e., the denotation of the predicate p in the perfect-model semantics of the program $\mathcal{P}$ intersected with the set of solutions of φ is the set of solutions of all constraints $\varphi_1, \ldots, \varphi_n$.

That is, the atom $p(\mathbf{v})$ lies in the perfect model of $\mathcal{P}$ and $\mathbf{v}$ is a solution of the constraint φ if and only if $\mathbf{v}$ is a solution of one of the constraints φ_i.

Proof. The proof of the theorem is (simultaneously) by induction on the level of the query. For this, we first define the level of a query $\langle Q, \varphi\rangle$, where Q is a conjunction of literals, as

follows. For a query $\langle Q, \varphi \rangle$, its level is the larger of (a) the maximum level of the predicates in the positive literals of Q, and, (b) one more than the maximum level of the predicates of the negative literals of Q.

Base Case: The base case is when the level of query is 0. I.e., Q is a conjunction of atoms such that each predicate in Q has level 0. This case is proved by a simple extension of Lemma 3.17 and Theorem 3.18 in [TS86a].

Induction Step: Suppose that the theorem holds for queries of level $\leq k$. Suppose also that the query is $\langle Q, \varphi \rangle$ where Q is a query of level $k+1$.

(Soundness): We use induction on the length n of the refutation. Suppose, first, that $n = 1$. Then Q is a literal. Suppose also that Q is a positive literal $Q(\mathbf{x})$. Suppose that ψ is the computed answer in the table entry corresponding to $\langle Q(\mathbf{x}), \varphi \rangle$. Now there must exist a nonground fact (or a set of nonground facts) $Q(\mathbf{x}) \longleftarrow \varphi'$ through which the query succeeded. Hence, $\{Q(\mathbf{v}) | \mathcal{D}, \mathbf{v} \models \psi\}$ is contained in the perfect model of $\mathcal{P}$. Now suppose that $Q(\mathbf{x}) = \neg p(\mathbf{x})$ is a negative literal. Then the following happens. Since, the derivation is of length 1, either, the corresponding derivation starting from the positive literal fails or all clauses with $p(\mathbf{x})$ at the head are nonground facts. In the first case the answer to the query is φ itself. By induction hypothesis (for the induction on levels), for any tuple $\mathbf{v}$ such that $\mathcal{D}, \mathbf{v} \models \varphi$, the atom $p(\mathbf{v})$ is not contained in the perfect model of $\mathcal{P}$. Therefore, for each tuple $\mathbf{v}$ such that $\mathcal{D}, \mathbf{v} \models \varphi$, the negated atom $\neg p(\mathbf{v})$ is contained in the perfect model of $\mathcal{P}$. The reasoning for the success case is similar.

Suppose now that $n > 1$ and the result holds for refutations of length less than or equal to $n-1$. Suppose first that the leftmost literal of Q is a positive literal $p(\mathbf{x})$. Let $Q = p(\mathbf{x}) \wedge P'$. Assume that there exists no entry in the table for $\langle p(\mathbf{x}), \eta \rangle$ such that $\exists_{-\mathbf{x}} \varphi \models \eta$. The other case in which there is such a table entry can be proved similarly. Suppose that the first clause through which the query resolves is $p(\mathbf{x}) \longleftarrow B \wedge \varphi'$. Let the free variables in the body B be $\mathbf{y}$. Let the free variables of P' be $\mathbf{z}$. Then the resolvent is given by $\langle B \wedge P', \exists_{-(\mathbf{y},\mathbf{z})} \varphi \wedge \varphi' \rangle$. Let the answers for the table entries corresponding to the literals in B and P' be $\psi_1, \ldots, \psi_k$ (for a query $\langle l(\mathbf{y}') \wedge B', \varphi'' \rangle$, whenever a literal $l(\mathbf{y}')$ is called, then if there exists a table entry of the form $\langle l(\mathbf{y}'), \varphi''' \rangle$ such that $\exists_{-\mathbf{y}'} \varphi'' \models \varphi'''$, then this goal is made to point to the solution list corresponding to that entry in the table; otherwise a table entry for $\langle l(\mathbf{y}'), \exists_{-\mathbf{y}'} \varphi'' \rangle$ with an empty list is created). By induction hypothesis (for the induction over the length of the refutation), for each literal l_i in $B \wedge P'$, for any tuple $\mathbf{v}$, if $\mathcal{D}, \mathbf{v} \models \psi_i$ then $l_i(\mathbf{v})$ is in the perfect model of $\mathcal{P}$. Now the answer received by the entry $\langle p(\mathbf{x}), \exists_{-\mathbf{x}} \varphi \rangle$ is given by $\psi \equiv \exists_{-\mathbf{x}} \varphi \wedge \varphi' \wedge \psi_1 \wedge \ldots \wedge \psi_k$. We show that for any tuple $\mathcal{D}, \mathbf{v} \models \psi$, $\mathcal{D}, \mathbf{v} \models \varphi$ and $p(\mathbf{v})$ is in the perfect model of $\mathcal{P}$. Suppose that $\mathcal{D}, \mathbf{v} \models \psi$. Then, $\mathcal{D}, \mathbf{v} \models \varphi$. Also there exists a tuple $\mathbf{u}$ such that $\mathcal{D}, \mathbf{v}, \mathbf{u} \models \psi_1 \wedge \ldots \wedge \psi_k$. For any literal $l_i(\mathbf{y_i})$ in B, let $\mathbf{u_i}$ be the values of $\mathbf{y_i}$ in the tuple $\mathbf{u}$. Then $\mathcal{D}, \mathbf{u_i} \models \psi_i$. Therefore, for each i, $l_i(\mathbf{u_i})$ is in the perfect model of $\mathcal{P}$. Also, we have that, $p(\mathbf{v}) \longleftarrow l_1(\mathbf{u_1}) \wedge \ldots \wedge l_m(\mathbf{u_m})$ is the ground instance of a clause in $\mathcal{P}$. Hence $p(\mathbf{v})$ is in the perfect model of $\mathcal{P}$.

Now suppose that the leftmost literal of Q is a negated literal. Assume that it is $\neg p(\mathbf{x})$. Assume, without loss of generality, that there does not exist an entry in the table of the form $\langle \neg p(\mathbf{x}), \varphi' \rangle$ such that $\exists_{-\mathbf{x}} \varphi \models \varphi'$. Then a fresh entry for $\langle \neg p(\mathbf{x}), \exists_{-\mathbf{x}} \varphi \rangle$ has been put in the table with an empty list at the beginning of evaluation of this query. Assume that the answer obtained for this entry is ψ. Then we have the following. Either, the query $Q' = \langle p(\mathbf{x}), \varphi'' \rangle$, where $\exists_{-\mathbf{x}} \varphi \models \varphi''$, has failed. Or all answers to the query Q' have been obtained. In the first case, the answer $\psi \equiv \exists_{-\mathbf{x}} \varphi$. By the induction hypothesis (for the induction on the level of the query), for any tuple $\mathbf{v}$ such that $\mathcal{D}, \mathbf{v} \models \exists_{-\mathbf{x}} \varphi$, the atom $p(\mathbf{v})$ is not in the perfect model of $\mathcal{P}$.

Hence, for any tuple $\mathbf{v}$ such that $\mathcal{D}, \mathbf{v} \models \exists_{-\mathbf{x}}\varphi$, $\neg p(\mathbf{v})$ is contained in the perfect model of $\mathcal{P}$. Similarly for the second case.

(Completeness): Now suppose that for a literal l and a tuple $\mathbf{v}$, $l(\mathbf{v})$ is in the perfect model of $\mathcal{P}$. Let φ be a constraint such that $\mathcal{D}, \mathbf{v} \models \varphi$. We show that, if the procedure terminates, there exists a table entry of the form $\langle l(\mathbf{x}), \varphi' \rangle$ such that $\varphi \models \varphi'$ and the solution list corresponding to this entry contains a constraint ψ such that $\mathcal{D}, \mathbf{v} \models \psi$ (from now, we suppose that the procedure terminates). First, suppose that $l(\mathbf{x}) = p(\mathbf{x})$ is a positive literal. Then the level of the query $\langle p(\mathbf{x}), \varphi \rangle$ is $k+1$. Then, $p(\mathbf{v}) \in T_{\mathcal{P}}^{k+1} \uparrow n$, for some $n \geq 0$. We prove by induction on n that, for for any ground atom $\mathbf{v}$, such that $\mathcal{D}, \mathbf{v} \models \varphi$, if $p(\mathbf{v}) \in T_{\mathcal{P}}^{k+1} \uparrow n$, then the solution list of the table entry corresponding to $\langle p(\mathbf{x}), \varphi \rangle$ (or of $\langle p(\mathbf{x}), \varphi' \rangle$ where $\varphi \models \varphi'$) contains a constraint ψ such that $\mathcal{D}, \mathbf{v} \models \psi$.

Suppose first that $n = 1$. Then there exists a ground instance $p(\mathbf{v}) \longleftarrow D$ of a clause C such that the perfect model of $\mathcal{P}$ logically implies D. Let the clause C be of the form $p(\mathbf{x}) \longleftarrow B \wedge \varphi'''$. Let the free variables occurring in B be $\mathbf{y}$. Then the resolvent of $\langle p(\mathbf{x}), \varphi \rangle$ through C is given by $\langle B, \exists_{-\mathbf{y}}\varphi \wedge \varphi''' \rangle$. Let $\eta \equiv \exists_{-\mathbf{y}}\varphi \wedge \varphi'''$. For a literal l_i in B, the level of the query $\langle l_i(\mathbf{y_i}), \exists_{-\mathbf{y_i}}\eta \rangle$ is at most k (since $n = 1$). Let the answers received for each such query $\langle l_i(\mathbf{y_i}), \exists_{-\mathbf{y_i}}\eta \rangle$ be ψ_i. Also, let $l_i(\mathbf{u_i})$ be a conjunct in D. Then, by the main induction hypothesis, $\mathcal{D}, \mathbf{u_i} \models \psi_i$. Now, the answer to the query $\langle p(\mathbf{x}), \varphi \rangle$ is given by $\psi' \equiv \exists_{-\mathbf{x}}\varphi \wedge \varphi''' \wedge \psi_1 \wedge \ldots \wedge \psi_k$. It is now easily shown that $\mathcal{D}, \mathbf{v} \models \psi'$.

Next, suppose that $n > 1$. Then there exists a ground instance $p(\mathbf{v}) \longleftarrow D$ of a clause C such that every ground atom in D is in the perfect model of $\mathcal{P}$. Let the clause C be of the form $p(\mathbf{x}) \longleftarrow B \wedge \varphi'''$. Let the free variables in B be $\mathbf{y}$. Then the resolvent of $\langle p(\mathbf{x}), \varphi \rangle$ through the clause C is given by $\langle B, \psi \rangle$ where $\psi \equiv \exists_{-\mathbf{y}}\varphi \wedge \varphi'''$. We can write B as $B' \wedge B''$ such that for any literal l_i in B', the level of the query $\langle l_i(\mathbf{y_i}), \exists_{-\mathbf{y_i}}\psi \rangle$ is less than or equal to k and for any literal l_j in B'', the level of the query $\langle l_j(\mathbf{y_j}), \exists_{-\mathbf{y_j}}\psi \rangle$ is $k+1$. For each i, let the answers obtained for the query $\langle l_i(\mathbf{y_i}), \exists_{-\mathbf{y_i}}\psi \rangle$ be ψ_i. Then, by the main induction hypothesis on k, for each conjunct $l_i(\mathbf{u_i})$ in D such that the literal l_i is in B', $\mathcal{D}, \mathbf{u_i} \models \psi_i$. Again, by the induction hypothesis on n, for each conjunct $l_i(\mathbf{u_i})$ in D such that the literal l_i is in B'', $\mathcal{D}, \mathbf{u_i} \models \psi_i$. Now the answer to the query $\langle p(\mathbf{x}), \varphi \rangle$ is given by $\psi' \equiv \exists_{-\mathbf{x}}\varphi \wedge \varphi''' \wedge \psi_1 \wedge \ldots \wedge \psi_k$. It can now be easily shown that $\mathcal{D}, \mathbf{v} \models \psi'$.

We next come to the case when l is a negative literal $\neg p(\mathbf{x})$. Then the level of the query $\langle p(\mathbf{x}), \varphi \rangle$ is k. Let the answer to the query $\langle p(\mathbf{x}), \varphi \rangle$ be ψ'. Then, by the induction hypothesis on k, $\mathcal{D}, \mathbf{v} \not\models \psi'$. Now the answer to the query $\langle l(\mathbf{x}), \varphi \rangle$ is $\psi'' \equiv \neg\psi$. Hence $\mathcal{D}, \mathbf{v} \models \psi''$.

$\|$

4.6 Convergence in Timed Automata

The procedure given in Section 4.5 is not guaranteed to terminate even when it is applied only to logic programs $\mathcal{P}_\psi$ that arise from the translation of timed automata. However, techniques are known (e.g. extrapolation [DT98] or trimming operation described in Chapter 3) to enforce termination of forward analysis of timed automata. These techniques operate on the representation of the constraints; i.e., they are orthogonal to the control aspects and can be integrated directly into our forward analysis procedure, turning it thus into an always terminating algorithm, i.e. a decision procedure for the model checking problem of Sμ formulas for timed automata.

The stratified μ-calculus is expressive enough to capture convergence of timed au-

tomata [AH97, HNSY94, Tri99]. It is well-known that, for every timed automaton $\mathcal{A}$, one can construct a timed automaton $\mathcal{A}'$ (in linear time) such that $\mathcal{A}$ is convergent if and only if the Sμ formula EF$\neg$EF$(y > 1)$ is true for the automaton $\mathcal{A}'$.

Two specialized algorithms for detecting convergence in timed automata have been developed, one based on backward [HNSY94] and one on forward analysis [Tri99]. We can now see that both algorithms are instances of two general procedures (bottom-up computation of the perfect model using the T_P operator, and tabled resolution) to check whether a query is true in the perfect model of a given stratified logic program P.

4.7 Checking Convergence

To give some intuition for the general approach, we consider an example. The timed automaton on the left part of Figure 4.4 is supposed to model a switch with two states 'on' and 'off', staying for at least one and at most two seconds in a state and then switching to the other one. The variable x stands for a clock; the constraints $x < 2$ in both states say that the switch can stay in the state only if the value of the clock does not reach 2, the constraints $x > 1$ on both edges mean that the move to the other state can be made only if the value of the clock exceeds 1. The automaton starts in the state on with the value of the clock set to 0.

This automaton does not model the intended switch; while changing from one state to the other we do not reset the clock x. However, the automaton still does not deadlock: there still exist infinite traces of the automaton, but it has to change its state infinitely many times before the clock reaches the value 2. Such a property of an automaton is called *convergence*. More precisely, we say that an automaton $\mathcal{A}$ is convergent if there exists a reachable state s of $\mathcal{A}$ such that in every infinite computation of $\mathcal{A}$ starting in s the time of the computation is bounded by a constant. This is clearly a kind of error that one would like to be able to find automatically.

The automaton on the right part of Figure 4.4 consists of two copies of the initial automaton, where all states are connected by edges labeled $y := 0$ with their copies. The variable y is a new clock that we add here. Intuitively, we simulate the initial automaton with an additional clock which we reset only once at some nondeterministically chosen point of time. It is easy to see that the new automaton satisfies the formula EF$\neg$EF$(y > 1)$ if and only if the initial automaton is convergent.

The translation of this Sμ formula to a constraint query language program involves defining the predicate *trans*. For example, the time transition at *on* gives the clause $trans(on, x, on, x') \longleftarrow x' = x + z \wedge z \geq 0 \wedge x' < 2$. The edge transitions can be translated following the same technique as in Chapter 3.

program tabled resolution
input: program $\mathcal{P}$ and a query $\langle p(\mathbf{x}), \varphi \rangle$
output:$[\varphi_1, \ldots, \varphi_n]$ such that $[\![p]\!]_{pm(\mathcal{P})} \cap [\![\varphi]\!] = [\![\varphi_1 \vee \ldots \vee \varphi_n]\!]$
set F as the tree consisting of one node $p(\mathbf{x}) \wedge \varphi$
set T as the empty table
set L as the empty mapping
classify$(p(\mathbf{x}) \wedge \varphi, T, L)$
repeat
  choose a node v that is extendable wrt. $\langle F, T, L \rangle$
  extend$(\langle F, T, L \rangle, v)$
until there is no extendable node in F
return$(T[\langle p(\mathbf{x}), \varphi \rangle])$

procedure *classify*(v, T, L)
if v is of the form $p(\mathbf{x}) \wedge R \wedge \varphi$ % v is not an answer node
then
  if T contains an entry indexed $\langle p(\mathbf{x}), \varphi' \rangle$ such that $\varphi \models \varphi'$
  then
    mark v as a *lookup* node
    assign $L(v) = T[\langle p(\mathbf{x}), \varphi' \rangle]$
  else
    mark v as an *active* node
    assign $T[\langle p(\mathbf{x}), \varphi \rangle] = [\,]$
  endif
endif
endproc % *classify*

procedure *tabulate*(v, v', ψ)
if (v is a positive active node of the form $l(\mathbf{x}) \wedge R \wedge \varphi$
    and the path from v to v' in F is a successful subderivation for $l(\mathbf{x}) \wedge \varphi$ wrt. T
    and $T[\langle l(\mathbf{x}), \varphi \rangle]$ does not contain ψ' such that $\psi \models \psi'$)
  or (v is a negative active node of the form $l(\mathbf{x}) \wedge R \wedge \varphi$
    and v' is a child of v)
then
  assign $T[\langle p(\mathbf{x}), \varphi \rangle] = append(T[\langle p(\mathbf{x}), \varphi \rangle], [NF(\psi)])$ % NF: elimination of $\exists$'s
endif
if v is not a root in F **then** *tabulate*$(parent(v), v', T)$ **endif**
endproc % *tabulate*

Figure 4.3: Tabled Resolution

procedure $extend(\langle F, T, L\rangle, v)$
case type of v **of** % see Definition 4.4
(a): % positive active node
let $v = p(\mathbf{x}) \wedge R \wedge \varphi$
let $C_1, \ldots, C_n$ be all clauses such that
$C_i = p(\mathbf{x}) \leftarrow body_i \wedge \varphi_i$ and $\varphi \wedge \varphi_i$ satisfiable
if $n = 0$ **then** mark v as failure node
else
for all $i = 1, \ldots, n$ **do**
create new node $v_i = body_i \wedge R \wedge \varphi \wedge \varphi_i$ as a child of v in F
$classify(v_i, T, L)$
if $body_i$ is empty **then** $tabulate(v, v_i, \varphi \wedge \varphi_i)$ **endif**
endfor
endif
(b): % lookup node
let $v = l(\mathbf{x}) \wedge R \wedge \varphi$
let $\varphi' = head(L(v))$ % (the first element of the list)
assign $L(v) = tail(L(v))$ % (the remainder of the list)
if $\varphi \wedge \varphi'$ satisfiable **then**
create new node $v' = R \wedge \varphi \wedge \varphi'$
$classify(v', T, L)$
$tabulate(v, v', \varphi \wedge \varphi')$
endif
(c): % negative active node, no companion
let $v = \neg p(\mathbf{x}) \wedge R \wedge \varphi$
create a new node $v' = p(\mathbf{x}) \wedge \varphi$ as the root of a new tree in F
$classify(v', T, L)$
mark v and v' as companion nodes
(d): % negative active node, companion done
let $v = \neg p(\mathbf{x}) \wedge R \wedge \varphi$
let $v' = p(\mathbf{x}) \wedge \varphi$ be the companion of v
let $[\varphi_1, \ldots, \varphi_n]$ be the list $T[\langle p(\mathbf{x}), \varphi'\rangle]$
where $\varphi = \varphi'$ or $\varphi \models \varphi'$ depending on whether v' is an active or lookup node
if $n = 0$ **then** set $n = 1$ and $[\varphi_1, \ldots, \varphi_n] = [false]$ **endif**
if $\varphi \wedge \bigwedge_{i=1}^{n} \neg\varphi_i$ is unsatisfiable **then** mark v as a failure node
else
let $\psi_1 \vee \ldots \vee \psi_k = DNF(\varphi \wedge \bigwedge_{i=1}^{n} \neg\varphi_i)$ % Disjunctive Normal Form
assign $T[\langle \neg p(\mathbf{x}), \varphi\rangle] = [\psi_1, \ldots, \psi_k]$
for all $i = 1, \ldots, k$ **do**
create new node $v_i = R \wedge \varphi \wedge \psi_i$ as a child of v in F
$classify(v_i, T, L)$
$tabulate(v, v_i, \varphi \wedge \varphi_i)$
endfor
endif
endcase
endproc % *extend*

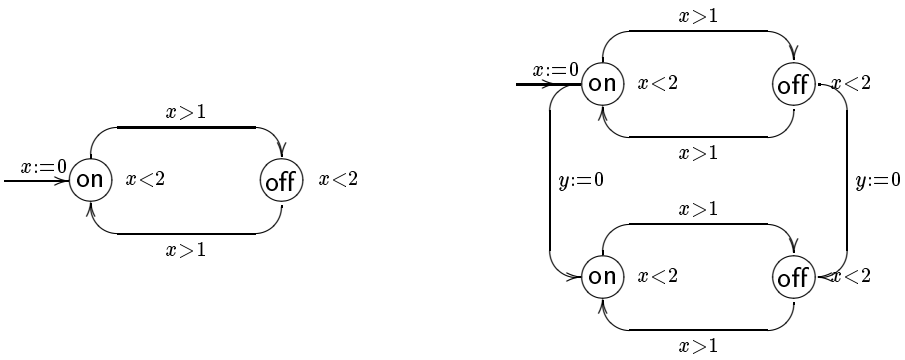


Figure 4.4: Timed automata

Chapter 5

Beyond Region Graphs: Symbolic Forward Analysis of Timed Automata

5.1 Introduction

A *timed automaton* [AD94] models a system whose transitions between finitely many control locations depend on the values of clocks. The clocks advance continuously over time; they can individually be reset to the value 0. Since the clocks take values over reals, the state space of a timed automaton is infinite.

The theoretical and the practical investigations on timed automata are recent but already quite extensive (see e.g. [AD94, HKPV95, LPY95b, Bal96, DT98]). Many decidability results are obtained by designing algorithms on the *region graph*, which is a finite quotient of the infinite state transition graph [AD94]. Practical experiments showing the feasibility of model checking for timed automata, however, employ *symbolic forward analysis*. We do not know of any practical tool that constructs the region graph. Instead, symbolic model checking is extended directly from the finite to the infinite case; logical formulas over reals are used to 'symbolically' represent infinite sets of tuples of clock values and are manipulated by applying the same logical operations that are applied to Boolean formulas in the finite state case.

If model checking is based on *backward* analysis (where one iteratively computes sets of predecessor states), termination is guaranteed [HNSY94]. In comparison, symbolic forward analysis for timed automata has the theoretical disadvantage of possible non-termination. Practically, however, it has the advantage that it is amenable to on-the-fly local model checking and to partial-order reduction techniques (see [HKQ98] for a discussion of forward vs. backward analysis).

In symbolic forward analysis applied to the timed automata arising in practical applications (see e.g. [LPY95b]), the theoretical possibility of non-terminating does not seem to play a role. Existing versions that exclude this possibility (through built-in runtime checks [DT98] or through a static preprocessing step [HKPV95]) are not used in practice.

This situation leads us to raising the question whether there exist 'interesting' sufficient conditions for the termination of symbolic model checking procedures for timed automata based

on forward analysis. Here, 'interesting' means applicable to a large class of cases in practical applications. The existence of a practically relevant class of infinite-state systems for which the practically employed procedure is actually an algorithm would be a theoretically satisfying explanation of the success of the ongoing practice of using this procedure, and it may guide us in designing practically successful verification procedures for other classes of infinite-state systems.

As a first step towards answering the question that we are raising, we build a kind of 'tool-box' consisting of basic concepts and properties that are useful for reasoning about sufficient termination conditions. The central notions here are constraint transformers associated with sequences of automaton edges and *zone trees* labeled with successor constraints. The constraint transformer associated with the sequences of edges $e_1, \ldots, e_n$ of the timed automaton assigns a constraint φ another constraint that 'symbolically' represents the set of the successor states along the edges $e_1, \ldots, e_n$ of the states in the set represented by φ. We prove properties for constraint transformers associated with edge sequences of a certain form; these properties are useful in termination proofs as we then show. The zone tree is a vehicle that can be used to investigate sufficient conditions for termination without having to go into the algorithmic details of symbolic forward analysis procedures. It captures the fact that the constraints enumerated in a symbolic forward analysis must respect a certain tree order.

We show how the zone tree can characterize termination of (various versions of) symbolic forward analysis. A combinatorial reasoning is then used to derive sufficient termination conditions for symbolic forward analysis. The reasoning essentially involves showing that certain properties of the control graph of a timed automaton are sufficient for ensuring termination of symbolic forward analysis. We prove that symbolic forward analysis terminates for three classes of timed automata. We show that the railroad-crossing example analyzed in [LS85, AD94] as well as certain fragments of the class of RQ timed automata characterized in [LB93] as a natural model for timed systems fall into these classes. Our analyses of these three classes demonstrate how the presented concepts and properties of the successor constraint function and of the zone tree can be employed to prove termination. Termination proofs can be quite tedious, as the third case shows; the proof here distinguishes and analyzes many cases (see the proof of Theorem 5.2).

5.2 The Constraint Transformer $\varphi \mapsto [\![w]\!](\varphi)$

A *timed automaton* $\mathcal{U}$ can, for the purpose of reachability analysis, be defined as a set $\mathcal{E}$ of guarded commmands e (called edges) of the form below. Here L is a variable ranging over the finite set of *locations*, and $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ are the variables standing for the clocks and ranging over nonnegative real numbers. As usual, the primed version of a variable stands for its value after the transition. The 'time delay' variable z ranges over nonnegative real numbers.

$$e \;\equiv\; L = \ell \wedge \gamma_e(\mathbf{x}) \;[\!]\; L' = \ell' \wedge \alpha_e(\mathbf{x}, \mathbf{x}', z).$$

The guard formula $\gamma_e(\mathbf{x})$ over the variables $\mathbf{x}$ is built up from conjuncts of the form $x_i \sim k$ where x_i is a clock variable, $\sim$ is a comparison operator (i.e., $\sim \in \{=, <, \leq, >, \geq\}$) and k is a natural number.

The action formula $\alpha_e(\mathbf{x}, \mathbf{x}', z)$ of e is defined by a subset Reset_e of $\{1, \ldots, n\}$ (denoting the clocks that are *reset*); it is of the form

$$\alpha_e(\mathbf{x}, \mathbf{x}', z) \;\equiv\; \bigwedge_{i \in \mathrm{Reset}_e} x_i' = z \wedge \bigwedge_{i \notin \mathrm{Reset}_e} x_i' = x_i + z.$$

We write ψ_e for the logical formula corresponding to e (with the free variables $\mathbf{x}$ and $\mathbf{x}'$; we replace the guard symbol $[\![$ with conjunction).

$$\psi_e(\mathbf{x}, \mathbf{x}') \;\equiv\; L = \ell \wedge \gamma_e(\mathbf{x}) \wedge L' = \ell' \wedge \exists z\; \alpha_e(\mathbf{x}, \mathbf{x}', z)$$

The states of $\mathcal{U}$ (called *positions*) are tuples of the form $\langle \ell, \mathbf{v} \rangle$ consisting of values for the location and for each clock. The position $\langle \ell, \mathbf{v} \rangle$ can make a *time transition* to any position $\langle \ell, \mathbf{v} + \delta \rangle$ where $\delta \geq 0$ is a real number.

The position $\langle \ell, \mathbf{v} \rangle$ can make an *edge transition* (followed by a time transition) to the position $\langle \ell', \mathbf{v}' \rangle$ using the edge e if the values ℓ for L, $\mathbf{v}$ for $\mathbf{x}$, ℓ' for L' and $\mathbf{v}'$ for $\mathbf{x}'$ define a solution for ψ_e. (An edge transition by itself is defined if we replace the variable z in the formula for α by the constant 0.)

We use *constraints* φ in order to represent certain sets of positions (called *zones*). A constraint is a conjunction of the equality $L = \ell$ with a conjunction of formulas of the form $x_i - x_j \sim c$ or $x_i \sim c$ where c is an integer (i.e. with a *zone constraint* as used in [DT98]). We identify solutions of constraints with positions $\langle \ell, \mathbf{v} \rangle$ of the timed automaton.

We single out the *initial constraint* φ^0 that denotes the time successors of the initial position $\langle \ell^0, \mathbf{0} \rangle$.

$$\varphi^0 \;\equiv\; L = \ell^0, x_1 \geq 0, x_2 = x_1, \ldots, x_n = x_1$$

Definition 5.1 (Time-closed Constraints) *A constraint φ is called* time-closed *if its set of solutions is closed under time transitions. Formally, $\varphi(\mathbf{x})$ is equivalent to $(\exists \mathbf{x} \exists z(\varphi \wedge x_1' = x_1 + z \wedge \ldots \wedge x_n' = x_n + z))[\mathbf{x}'/\mathbf{x}]$.*

For example, the initial constraint is time-closed. In the following, we will be interested only in time-closed constraints.

In the definition below, $\varphi'[\mathbf{x}'/\mathbf{x}]$ denotes the constraint obtained from φ' by α-renaming (replace each x_i' by x_i).

We write $e_1 \ldots . e_m$ for the word w obtained by concatenating the 'letters' $e_1, \ldots, e_m$; thus, w is a word over the set of edges $\mathcal{E}$, i.e. $w \in \mathcal{E}^\star$.

Definition 5.2 (Constraint Transformer $[\![w]\!]$) *The constraint transformer wrt. to an edge e is the 'successor constraint function' $[\![w]\!]$ that assigns a constraint φ the constraint*

$$[\![e]\!](\varphi) \;\equiv\; (\exists \mathbf{x}(\varphi \wedge \psi_e))[\mathbf{x}'/\mathbf{x}]$$

where ψ_e is the logical formula corresponding to e. The successor constraint function $[\![w]\!]$ wrt. a string $w = e_1 \ldots . e_m$ of length $m \geq 0$ is the functional composition of the functions wrt. the edges $e_1, \ldots, e_m$, i.e. $[\![w]\!] = [\![e_1]\!] \circ \ldots \circ [\![e_m]\!]$.

Thus, $[\![\varepsilon]\!](\varphi) = \varphi$ and $[\![w.e]\!](\varphi) = [\![e]\!]([\![w]\!](\varphi))$. The solutions of $[\![w]\!](\varphi)$ are exactly the ("edge plus time") successors of a solution of φ by taking the sequence of transitions via the edges e_1, $\ldots$, e_m (in that order).

We will next consider constraint transformers $[\![w]\!]$ for strings w of a certain form. In the next definition, the terminology 'a clock x_i is queried in the edge e' means that x_i is a variable occurring in the guard formula γ of e; 'x_i is reset in e' means that $i \in \mathrm{Reset}_e$.

Definition 5.3 (Stratified Strings) *A string $w = e_1 \ldots . e_m$ of edges is called stratified if*

– *each clock* $x_1, \ldots, x_n$ *is reset at least once in* w*, and*

– *if* x_i *is reset in* e_i *then* x_j *is not queried in* $e_1, \ldots, e_j$*.*

Proposition 5.1 *The successor constraint function wrt. a stratified string* w *is a constant function over satisfiable constraints (i.e. there exists a unique constraint* φ_w *such that* $[\![w]\!](\varphi) = \varphi_w$ *for all satisfiable constraints* φ*).*

Proof. We express the successor constraint of the constraint φ wrt. the stratified string $w = e_1 \ldots e_m$ equivalently by

$$[\![w]\!](\varphi) \;\equiv\; (\exists \mathbf{x} \exists \mathbf{x^1} \ldots \exists \mathbf{x^{m-1}} \exists z^1 \ldots \exists z^m (\varphi \wedge \psi_1 \wedge \ldots \wedge \psi_m))[\mathbf{x}/\mathbf{x^m}]$$

where ψ_k is the formula that we obtain by applying α-renaming to the (quantifier-free) conjunction of the guard formula $\gamma_{e_k}(\mathbf{x})$ and the action formula $\alpha_{e_k}(\mathbf{x}, \mathbf{x}', z)$ for the edge e_k; i.e.

$$\psi_k \;\equiv\; \gamma_{e_k}(\mathbf{x^{k-1}}) \wedge \alpha_{e_k}(\mathbf{x^{k-1}}, \mathbf{x^k}, z^k).$$

Thus, in the formula for e_k, we rename the clock variable x_i to x_i^{k-1}, its primed version x_i' to x_i^k, and the 'time delay' variable z to z^k.

We identify the variables x_i (applying in φ) with their "0-th renaming" x_i^0 (appearing in ψ_1); accordingly we can write $\mathbf{x^0}$ for the tuple of variables $\mathbf{x}$.

We will transform $\exists \mathbf{x^1} \ldots \exists \mathbf{x^{m-1}} (\psi_1 \wedge \ldots \wedge \psi_m)$ equivalently to a constraint ψ containing only conjuncts of the form $x_i^m = z^l + \ldots + z^m$ and of the form $z^l + \ldots + z^m \sim c$ where $l > 0$; i.e. ψ does not contain any of the variables x_i of φ. Thus, we can move the quantifiers $\exists \mathbf{x}$ inside; formally, $\exists \mathbf{x}(\varphi \wedge \psi)$ is equivalent to $(\exists \mathbf{x}\varphi) \wedge \psi$. Since φ is satisfiable, the conjunct $\exists \mathbf{x}\varphi$ is equivalent to *true*. Summarizing, $[\![w]\!](\varphi)$ is equivalent to a formula that does not depend on φ, which is the statement to be shown.

The variable x_i^k (the "k-th renaming of the i-th clock variable") occurs in the action formula of ψ_k, either in the form $x_i^k = z^k$ or in the form $x_i^k = x_i^{k-1} + z^k$, and it occurs in the guard formula of ψ_{k+1}, in the form $x_i^k \sim c$.

If the i-th clock is not reset in the edges $e_1, \ldots, e_{k-1}$, then we replace the conjunct $x_i^k = x_i^{k-1} + z^k$ by $x_i^k = x_i + z^1 + \ldots z^k$.

Otherwise, let l be the largest index of an edge e_l with a reset of the i-th clock. Then we replace $x_i^k = x_i^{k-1} + z^k$ by $x_i^k = z^l + \ldots + z^k$.

If $k = m$, the first case cannot arise due to the first condition on stratified strings (the i-th clock must be reset at least once in the edges $e_1, \ldots, e_m$). That is, we replace $x_i^m = x_i^{m-1} + z^k$ always by a conjunct of the form $x_i^k = z^l + \ldots + z^k$.

If the conjunct $x_i^k \sim c$ appears in ψ_{k+1}, then, by assumption on w (the second condition for stratified strings), the i-th clock is reset in an edge e_l where $l \leq k$. Therefore, we can replace the conjunct $x_i^k \sim c$ by $z_l + \ldots + z_k \sim c$.

Now, each variable x_i^k (for $0 < k < m$) has exactly one occurrence, namely in a conjunct C of the form $x_i^k = x_i + z^1 + \ldots z^k$ or $x_i^k = z^l + \ldots z^k$. Hence, the quantifier $\exists x_i^k$ can be moved inside, before the conjunct C; the formula $\exists x_i^k\, C$ can be replaced by *true*.

After the above replacements, all conjuncts are of the form $x_i^m = z^l + \ldots + z^m$ or of the form $z^l + \ldots + z^m \sim c$; as explained above, this is sufficient to show the statement. ∎

We say that an edge e is *reset-free* if $\mathrm{Reset}_e = \emptyset$, i.e., its action is of the form $\alpha_e \equiv \bigwedge_{i=1,\ldots,n} x_i' = x_i$. A string w of edges is reset-free if all its edges are.

Proposition 5.2 *If the string w is reset-free, and the successor constraint of a time-closed constraint of the form $L = \ell \wedge \varphi$ is of the form $L = \ell' \wedge \varphi'$, then φ' entails φ, formally $\varphi' \models \varphi$.*

Proof. It is sufficient to show the statement for w consisting of only one reset-free edge e. Since φ is time-closed, it is equivalent to $(\exists \mathbf{x} \exists z(\varphi \wedge \mathbf{x}' = \mathbf{x} + z))[\mathbf{x}/\mathbf{x}']$.

Then $[\![w]\!](L = \ell \wedge \varphi)$ is equivalent to $(\exists \ldots (L = \ell' \wedge \varphi \wedge \mathbf{x}' = \mathbf{x} + z' \wedge \gamma_e(\mathbf{x}') \wedge \mathbf{x}'' = \mathbf{x}' + z')[\mathbf{x}/\mathbf{x}'']$. This constraint is equivalent to $L = \ell' \wedge \varphi(\mathbf{x}) \wedge \gamma(\mathbf{x})$. This shows the statement. ∎

5.3 Zone Trees and Symbolic Forward Analysis

Definition 5.4 (Zone Tree) *The zone tree of a timed automaton $\mathcal{U}$ is an infinite tree whose domain is a subset of $\mathcal{E}^\star$ (i.e., the nodes are the strings over $\mathcal{E}$) that labels the node w by the constraint $[\![w]\!](\varphi^0)$.*

That is, the root ε is labeled by the initial constraint φ^0. For each node w labeled φ, and for each edge $e \in \mathcal{E}$ of the timed automaton, the successor node $w.e$ is labeled by the constraint $[\![e]\!](\varphi)$. Clearly, the (infinite) disjunction of all constraints labeling a node of the zone tree represents all reachable positions of $\mathcal{U}$.

We are interested in the termination of various versions of symbolic forward analysis of a timed automaton $\mathcal{U}$. All versions have in common that they traverse (a finite prefix of) its zone tree, in a particular order. The following definition of a non-deterministic procedure abstracts away from that specific order.

Definition 5.5 (Symbolic Forward Analysis) *A symbolic forward analysis of a timed automaton $\mathcal{U}$ is a procedure that enumerates constraints φ_i labeling the nodes w_i of the zone tree of $\mathcal{U}$ in a tree order such that the enumerated constraints together represent all reachable positions. Formally,*

- *$\varphi_i = [\![w_i]\!](\varphi^0)$ for $0 \leq i < B$ where the bound B is a natural number or ω,*
- *if w_i is a prefix of w_j then $i \leq j$,*
- *the disjunction $\bigvee_{0 \leq i < B} \varphi_i$ is equivalent to the disjunction $\bigvee_{0 \leq i < \omega} \varphi_i$.*

We assume that the constraint φ_i is computed by applying any of the known quantifier elimination algorithms (see e.g. [MS98]) to a conjunction of constraints.

The number i is a *leaf* of a symbolic forward analysis if the node w_i is a leaf of the tree formed by all the nodes w_i where $0 \leq i \leq B$.

We say that a symbolic forward analysis *terminates* if the bound B is finite (i.e. not ω). We define that symbolic forward analysis terminates *with local subsumption* if for all its leafs i there exists $j < i$ such that the constraint φ_i entails the constraint φ_j. In contrast, it terminates with *global* subsumption if for all its leafs i there the constraint φ_i entails the disjunction of all constraints φ_j where $j < i$. Model checking is more efficient with local subsumption than with global subsumption, both practically and theoretically [DP99a].

A depth-first symbolic forward analysis depends on a chosen order of edges. Symbolic forward analysis terminates if and only if the depth-first symbolic forward analysis of $\mathcal{U}$ terminates for *every* order chosen.

If the symbolic depth-first forward analysis of $\mathcal{U}$ terminates for at least one order of edges, then the breadth-first version also terminates. The converse need not be true, as the counterexample of Figure 6.6 shows.

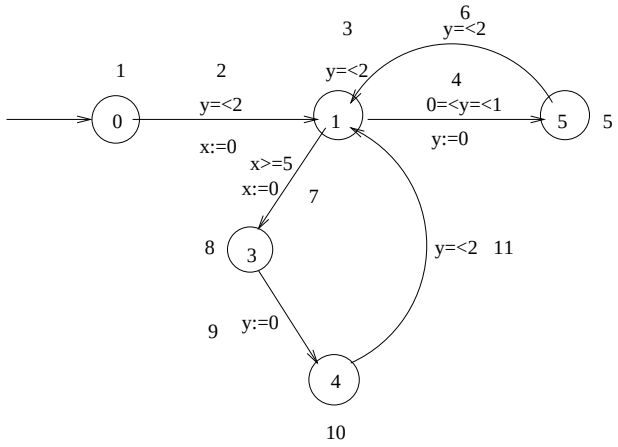


Figure 5.1: Example of a timed automaton for which the breadth-first version of symbolic forward analysis terminates but the depth-first version does not, if the edge numbered 4 is followed before the edge numbered 7.

A *path* p in a zone tree is an infinite string over $\mathcal{E}$, i.e., $p \in \mathcal{E}^\omega$; p contains a node w if the string w is a prefix of p, written $w < p$. A node v precedes a node w if v is a prefix of w, written $v < p$.

Definition 5.6 (Local finiteness) *A path p of a zone tree is locally finite if and only if it contains a node w labeled by a constraint that entails the constraint labeling some node v preceding w (formally, there exist v and w such that $v < w < p$ and $[\![w]\!](\varphi^0) \models [\![v]\!](\varphi^0)$). A zone tree is locally finite if every path is.*

The relation between the termination of symbolic forward analysis and the local finiteness of the zone tree for a timed automaton is formalized as follows.

Proposition 5.3 Every *symbolic forward analysis of a timed automaton $\mathcal{U}$ terminates with local subsumption if and only if the zone tree of $\mathcal{U}$ is locally finite.*

We will next investigate the special class of *strings* (that we call *cycles*) that correspond to cycles in the control graph of the given timed automaton. Each cycle in the graph-theoretic sense corresponds to finitely many cycles in the sense defined here (as strings), depending on the entry location.

We say that an edge e of the form $L = \ell \ldots \parallel L' = \ell' \ldots$ leads from the location ℓ to the location ℓ'. This terminology reflects the fact that there exists a directed edge from ℓ to ℓ' labeled by the corresponding guarded command in the control graph of the given timed automaton (we will not formally introduce the control graph). Semantically, all transitions using such an edge go from a position with the location ℓ to a position with the location ℓ'. We canonically extend the terminology 'leads to' from edges e to strings w of edges.

Definition 5.7 (Cycle) *The string $w = e_1 \ldots . e_m$ of length $m \geq 1$ is a cycle if the sequence of edges $e_1, \ldots, e_m$ lead from a location ℓ to the same location ℓ such that there exists a sequence of edges that leads from the initial location ℓ^0 to ℓ whose last edge is different from e_m.*

The last condition above expresses that ℓ is an entry point to the corresponding cycle in the control graph of the given timed automaton $\mathcal{U}$. The next notion is used in effective sufficient termination conditions.

Definition 5.8 (Simple Cycle) *A cycle $w = e_1 \ldots . e_m$ is called simple if it does not contain a proper subcycle; formally, no string $e_i \ldots . e_j$ where $1 \leq i < j \leq m$ is also a cycle.*

Proposition 5.4 *A locally infinite path $p \in \mathcal{E}^\omega$ in the zone tree of the timed automaton $\mathcal{U}$ contains infinitely many occurrences of a simple cycle w; formally, p is an element of the omega-language $(\mathcal{E}^\star.w)^\omega$.*

Proof. Let p be a locally infinite path. Then there exists a location ℓ such that infinitely many nodes on this path are labeled by ℓ (i.e. a constraint of the form $L = \ell \wedge \ldots$. The strings formed by the edges connecting two nodes labeled by ℓ must all contain a simple cycle. Since the number of simple cycles is finite, some simple cycles must be repeated infinitely often. ∎

A string is *stratifiable* if contains a stratified substring (a substring of a string $e_1 \ldots . e_m$ is any string of the form $e_i \ldots . e_j$ where $1 \leq i \leq j \leq m$).

Proposition 5.5 *If every simple cycle of the timed automaton $\mathcal{U}$ is either reset-free or stratifiable, the zone tree of $\mathcal{U}$ is locally finite.*

Proof. Follows from Propositions 5.1, 5.2 and 5.4. ∎

We apply the above results to obtain our first sufficient termination condition.

Theorem 5.1 *Symbolic depth-first forward analysis of a timed automaton $\mathcal{U}$ terminates if all simple cycles of $\mathcal{U}$ are either reset-free or stratifiable.*

Proof. Follows from Propositions 5.3 and 5.5. ∎

To show the applicability of our result, consider the train-gate-controller example adapted from [AD94, LS85]. This example consists of the parallel composition of three components—the gate, the controller and the train. The transition systems (timed automata) for the gate, controller and train are given in Figures 5.2, 5.3 and 5.4 respectively. The transition system corresponding to the parallel composition of the three systems is given in Figure 5.5. It can be seen that each simple cycle in the composed system is stratifiable. Hence symbolic forward analysis train-gate-controller example terminates.

5.4 RQ Automata

A timed automaton $\mathcal{U}$ is called RQ [LB93] if for each clock x, $\mathcal{U}$ contains exactly one edge with a reset of x and exactly one edge with a query of x, and moreover, for every transition sequence of $\widetilde{\mathcal{U}}$ starting from the initial position, the sequence of resets and queries of x is alternating,

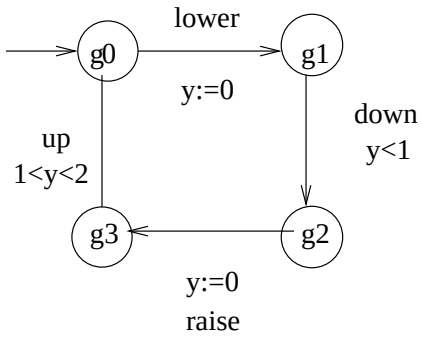


Figure 5.2: Gate Automaton

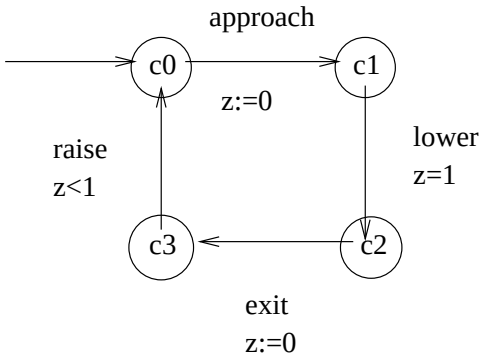


Figure 5.3: Controller Automaton

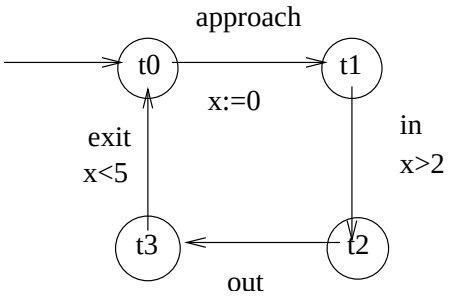


Figure 5.4: Train Automaton

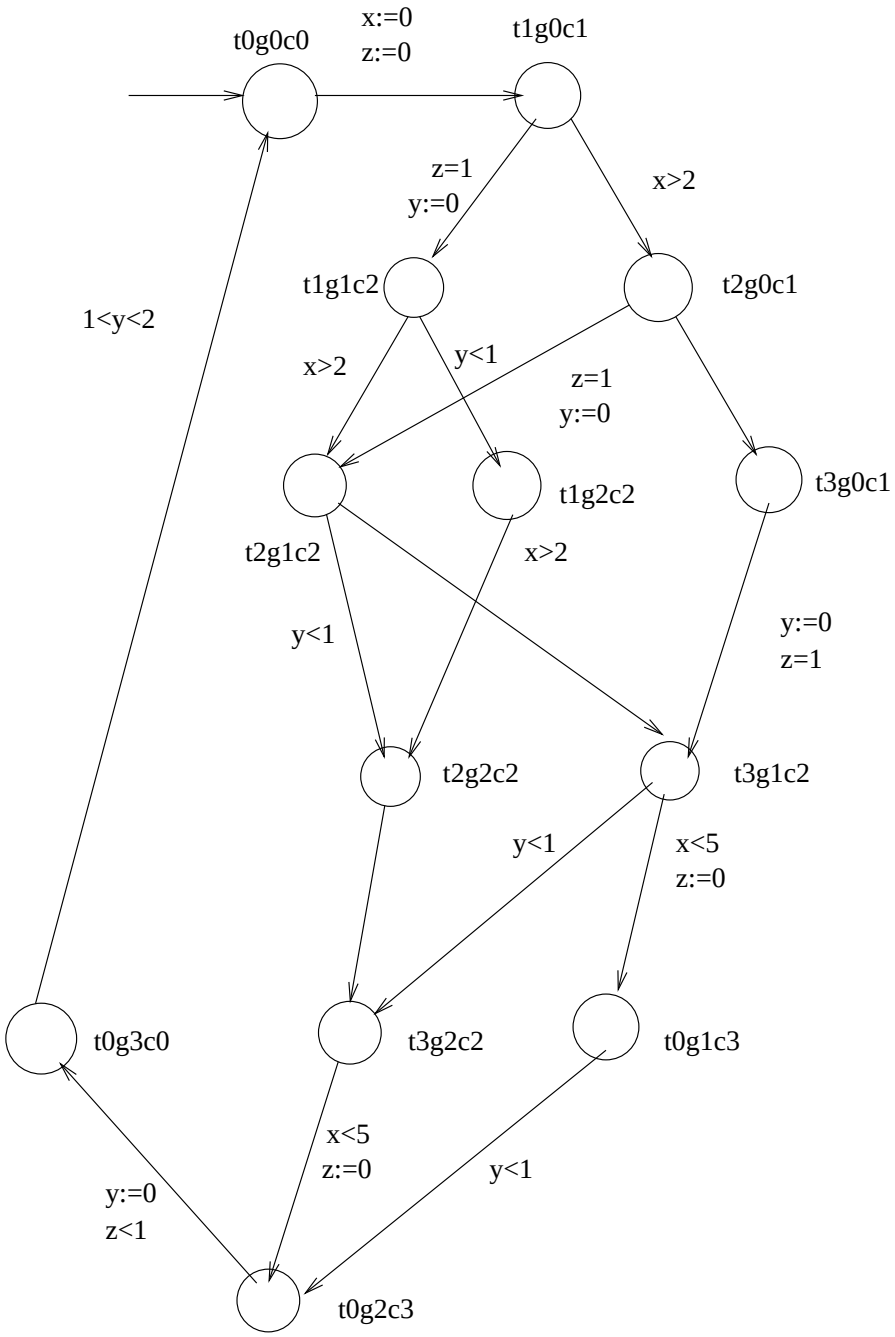


Figure 5.5: Train || Gate || Controller

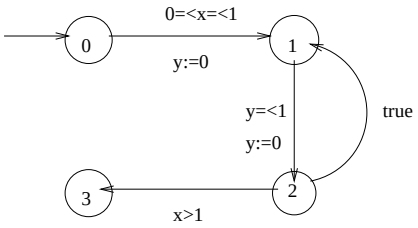


Figure 5.6: Example of a timed automaton showing that the property: "Every reachable location is reachable through a simple path" does *not* entail termination of depth-first symbolic forward analysis.

with a reset before the first query; here, $\widetilde{\mathcal{U}}$ refers to the timed automatonfrom $\mathcal{U}$ obtained by replacing all conjuncts $x \sim c$ in the guard formulas by the conjunct $x \geq 0$. We may require wlog. that no edge e of a timed automaton $\mathcal{U}$ contains both a reset of a clock and a query of a clock.

RQ automata have the following interesting property: if a location is reachable then it is reachable through a *simple path*, i.e. a sequence of edges that form a string not containing a cycle [LB93]. So it is possible to derive specialized terminating graph algorithms for reachability for RQ automata. Moreover, a cycle is traversable infinitely often if it is traversable once [LB93]. We will now investigate how a generic model checker based on symbolic forward analysis behaves on RQ automata. We do not know whether we obtain termination for this special case. We know that the distinguished property of RQ automata (that reachability is equivalent to reachability through a simple path) by itself is not sufficient for termination; Figure 5.6 gives a counterexample.

We will consider two special classes of RQ automata. The first one is characterized by the cut condition.

Definition 5.9 (Cut condition) *A timed automaton $\mathcal{U}$ satisfies the* cut condition *if any two simple cycles w and w' are either identical or their sets of edges are disjoint.*

Graph-theoretically, every simple cycle in the control graph has exactly one entry point (which is then called the 'cut vertex').

Theorem 5.2 *Symbolic depth-first forward analysis of an RQ timed automaton $\mathcal{U}$ terminates if it satisfies the cut condition and in every simple cycle, either all or no clock is reset.*

Proof. A simple cycle containing a reset for each clock in an RQ automaton satisfying the cut condition is stratified. Hence, Theorem 5.1 yields the statement. ∎

The second class of RQ automata is obtained by restricting the number of clocks to two. Many interesting timed client server protocols belong to this class. See [LB93] for examples.

Theorem 5.3 *Symbolic depth-first forward analysis of an RQ timed automaton with two clocks terminates.*

Proof. We name the two clock variables of the automaton x and y. We note R_x the unique edge of the time automaton where x is reset, and Q_y the one where x is queried; similarly we define R_y and Q_y. By our non-proper restriction, $R_x \neq Q_x$ etc..

A *segment* S of a path p in a zone tree is a sequence of nodes $n_1, \ldots, n_m$ of the zone tree. The string $w = e_1 \ldots e_{m-1}$ *labels* the segment S if n_m is reached from n_1 by following the edges $e_1, \ldots, e_m$ in the zone tree.

For a proof by contradiction, assume that p is an infinite branch of the zone tree. By Proposition 5.4, there exists a simple cycle w (leading, say, from the location ℓ to ℓ) that repeats infinitely often on p. We write S_1, S_2, ... for the segments that are labeled by w (in consecutive order). We write L_i for the segment between S_i and S_{i+1}. We note v^i the string labeling the segment L_i; each string v^i is a cycle (leading also from the location ℓ to ℓ). Below we will use the terminology 'w labels S_i' and 'v^i labels L_i'.

We first distinguish between the cases whether the edge R_x is part of the string w ("$R_x \in w$") or not.

Case 1 $R_x \in w$.

The edge Q_x must then also be an element of w (if the cycle w can be executed once then even infinitely often [LB93]; if it contained R_x but not Q_x then the RQ condition would be violated).

Case 1.1 $R_y \in w$.

Again, we must have that $Q_y \in w$.

We distinguish between the cases that the edge R_y appears strictly before the edge Q_y in the strings w ("$R_y < Q_y$") or after ("$Q_y < R_y$").

Case 1.1.1 $R_y < Q_y$.

Repeating the above reasoning for x instead of y, we distinguish between the cases "$R_y < Q_y$" and "$Q_y < R_y$".

Case 1.1.1.1 $R_x < Q_x$.

The two assumptions $R_x < Q_x$ and $R_y < Q_y$ mean that the string w is *stratified*. Hence, by Proposition 5.1, the successor constraint function wrt. w is constant. Hence, the constraint labeling the last node of S_2 entails the constraint labeling the last node of S_1. Thus, the path p is locally finite, which achieves the contradiction.

Case 1.1.1.2 $Q_x < R_x$.

We distinguish the cases whether the edge Q_x appears before the edge R_y or strictly after.

Case 1.1.1.2.1 $Q_x < R_y$.

Combining the assumptions leading to this case, namely $R_x \in w$ (and hence also $Q_x \in w$) and $R_y \in w$ (and hence also $Q_y \in w$) and $R_y < Q_y$ and $Q_x < R_x$ and $Q_x < R_y$, we know that the string w is of the form $w = w_1.Q_x.w_2$ such that w_2 contains R_x and R_y. Hence, the substring w_2 of w *stratified*. By Proposition 5.1, the successor constraint function wrt. w_2 is constant, and hence also the one wrt. w. As in the case above, we achieve a contradiction.

Case 1.1.1.2.2 $R_y < Q_x$.

Again we combine the assumptions leading to this case: namely $R_x, Q_x, R_y, Q_y \in w$ and $R_y < Q_y$ and $Q_x < R_x$ and $R_y < Q_x$.

Only using that $R_y < R_x$, we know that the string w is of the form $w = w_1.R_y.w_2.R_x.w_3$.

One of the two cases, namely $R_x \notin L_i$ or $R_x \in L_i$, will hold for infinitely many segments L_i's.

Case 1.1.1.2.2.1 $R_x \notin L_i$.

Then also $Q_x \notin L_i$ (because of the RQ-condition and since L_i is a cycle).

We then distinguish between the analogue cases for y instead of x.

Case 1.1.1.2.2.1.1 $R_y \notin L_i$.

Again, then $Q_y \notin L_i$.
We are assuming that $R_x, Q_x, R_y, Q_y \notin L_i$ for infinitely many L_i. We take two such segments, calling them L and L'. Let v and v' be the string labeling (the edge linking the nodes in) L and L'. Then, the successor constraint functions wrt. v and v' are the identity.
We form the *stratified* strings $V = R_x.w_3.v.w_1.R_y$ and $V' = R_x.w_3.v'.w_1.R_y$. Since the successor constraint functions wrt. v and v' are the identity, the successor constraint functions wrt. V and V' are the same *constant* function. The same reasoning as above leads to a contradiction.
Case 1.1.1.2.2.1.2 $R_y \in L_i$.
Then also $Q_y \in L_i$. Because of the RQ-condition and since the edge R_y precedes Q_y in S_i, the first occurrence of R_y precedes the first occurrence of Q_y in L_i. Hence, the strings v and v' (defined as above, labeling of some L_i's) is of the form $v = v_1.R_y.v_2$ or $v = v_1'.R_y.v_2'$ where v_1, v_2, v_1' and v_2' do not contain any reset or any query of a clock variable (and hence, yield the identity as the successor constraint function). We form the *stratified* substrings $V = R_x.w_3.v_1.R_y$ and $V' = R_x.w_3.v_1'.R_y$, which yield the same constant successor constraint function for the same reason as above. Again, this leads to a contradiction.
Case 1.1.1.2.2.2 $R_x \in L_i$.
Again, then $Q_x \in L_i$. Now we are assuming that $R_x, Q_x, R_y, Q_y \in L_i$ for infinitely many L_i.
As in Case 1.1.1.2.2.1.2, the first occurrence of R_y must precede the first occurrence of Q_y in L_i.
Assume that there is a reset of x in L_i before the first reset of y. We form the string $R_x.w_2.v_1, R_x$ where $w = w_1.R_x.w_2$ is such that w_2 does not contain any reset (by the assumptions for the cases 1.1.1.2 and 1.1.1.2.2) and $v^i = v_1.R_x.v_2$ (the string labeling L_i) is such that v_1 does not contain any reset. Following the lines of the proof for Proposition 5.2 one can show that for any constraint φ, $[\![R_x.w_2.v_1.R_x]\!](\varphi)$ entails $[\![R_x]\!](\varphi)$. This is a contradiction (to the fact that the path p is locally infinite).
Assume that there is no reset of x in L_i before the first reset of y. Then the string formed by the edges leading from the reset of x in S_i to the first reset of y in L_i is stratified. We can then apply the same reasoning as in Case 1.1.1.2.1 to derive a contradiction.
Case 1.1.2 $Q_y < R_y$.
Thus now $R_x \in w$ (and hence $Q_x \in w$), $R_y \in w$ (and hence $Q_y \in w$) and $Q_y < R_y$. Now we consider the following subcases of this case.
Case 1.1.2.1 $R_x < Q_x$
This case is symmetric to Case 1.1.1.2.1 where $R_x, R_y \in w$, $Q_x < R_y$ and $R_y < Q_y$.
Case 1.1.2.2 $Q_x < R_x$.
The assumption of the case is that the reset occurs after the query for both clocks. Due to the RQ condition, there cannot be any query between the two resets. Therefore, $R_x.w_1.R_y$ (or, symmetrically, $R_y.w_1.R_x$) forms a *stratified* substring of w. As before, we obtain a contradiction.
Case 1.2 $R_y \notin w$.

We distinguish between the following subcases of this case.
Case 1.2.1 $R_x < Q_x$.

One of the following subcases holds for infinitely many L_i.
Case 1.2.1.1$R_y \notin L_i$.
As in the proof for Case 1.1.1.2.2.2, we form a substring of the form $R_x.w_2.v_1.R_x$ where w_2 and v_1 don't contain any reset, and again obtain a contradiction.

Case 1.2.1.2 $R_y \in L_i$.

We show that for the case when x is reset more than once in L_i after the last reset of y in L_i, we can obtain a contradiction.

We form the string $R_x.w_1.R_x$ is such that w_1 does not contain any reset (by the assumptions for the cases 1.1.1.2 and 1.1.1.2.2) and $v^i = v_1.R_x.w_1.R_x.v_2$ (the string labeling L_i). Following the lines of the proof for Proposition 5.2 one can show that for any constraint φ, $[\![R_x.w_2.v_1.R_x]\!](\varphi)$ entails $[\![R_x]\!](\varphi)$. This is a contradiction (to the fact that the path p is locally infinite).

The remain cases are as follows, one of which repeats infinitely often.

Case 1.2.1.2.1. The last reset of y in L_i is followed by a query of y in L_i which is again followed by a reset and a query of x in L_i in that order.

Note that there cannot be any reset or query of x between R_y and Q_y above as that would violate the RQ condition. Now consider the substring $R_y.w_1.Q_y.w_2.R_x$ of v^i. This substring is *stratified* and w_1 and w_2 do not contain any reset or query. Hence using the same methods as in the previous cases. we obtain a contradiction.

Case 1.2.1.2.2 The last reset of y in L_i is followed by a reset and a query of x in L_i in that order.

Note that the substring $R_y.w_1.R_x$ of v^i is a *stratified* substring and w_1 does not contain any reset or query. Hence using techniques similar to that of the above subcases, we obtain a contradiction.

Case 1.2.1.2.3 The last reset of y in L_i is followed by a query of y in L_i which is again followed by a query, a reset and a query of x in L_i in that order (assuming that the last reset of x before the last reset of y in L_i was not followed by a query of x).

Note that the substring $R_x.w_1.R_y$ of v^i is a *stratified* substring and w_1 does not contain any reset or query. Hence using techniques similar to that of the above subcases, we obtain a contradiction.

Case 1.2.1.2.4 The last reset of y in L_i is followed by a query, a reset and a query of x in L_i in that order (assuming that the last reset of x before the last reset of y in L_i was not followed by a query of x).

Note that the substring $R_x.w_1.R_y$ of v^i is a *stratified* substring and w_1 does not contain any reset or query. Hence using techniques similar to that of the above subcases, we obtain a contradiction.

Case 1.2.1.2.5 The last reset of y in L_i is followed only by a query of y in L_i.

Note that the substring $R_y.w_1.Q_y.w_2.R_x$, of $v^i.w$ is a *stratified* substring and w_1 and w_2 do not contain any reset or query. Hence using techniques similar to that of the above subcases, we obtain a contradiction.

Case 1.2.2 $Q_x < R_x$.

We consider the case when $R_x \in w$, $R_y \notin w$ and $Q_x < R_x$. The following subcases of this case are to be considered:

Case 1.2.2.1 $R_y \notin L_i$.

Then also $Q_y \notin L_i$ (because of the RQ-condition and since L_i is a cycle).

We then distinguish between the analogue cases for x instead of y.

Case 1.2.2.1.1 $R_x \notin L_i$.

Again, then $Q_x \notin L_i$.

We are assuming that $R_y, Q_y, R_x, Q_x \notin L_i$ for infinitely many L_i. We form the substring $R_x.w_1.v^i.w_2.R_x$ where $w = w_2.R_x.w_1$. Note that there is no reset of any clock in $w_1.v^i.w_2$. Hence, reasoning as in Case 1.1.1.2.2.2, we obtain a contradiction.

Case 1.2.2.1.2 $R_x \in L_i$.

The proof for this case is the similar to that of the above case.

Case 1.2.2.2 $R_y \in L_i$.

This case assumes $R_x \in w$, $Q_x < R_x$ and for infinitely many i, $R_y \in L_i$. One of the following subcases of this case occurs infinitely often.

Case 1.2.2.2.1 $R_x \notin L_i$.

First note that there can be at most one reset of y in L_i (otherwise, reasoning as in case 1.2.1.2, we already obtain a contradiction) . Secondly, note that the query of y cannot precede its reset in L_i (otherwise the RQ condition is violated). Lastly note that the string $R_x.w_1.R_y$ is a *stratified* substring of $w.v^i$. Also w_1 does not involve any reset or query. Hence, reasoning as in the above subcases, we obtain a contradiction.

Case 1.2.2.2.2 $R_x \in L_i$.

In this case both x and y are reset in L_i. Note the following facts. First between the reset of x in S_i and the first reset of y in L_i, there cannot be any reset of x (otherwise, reasoning as in case 1.2.1.2, we already obtain a contradiction). Now first consider the case when the first reset of y in L_i precedes its first query in L_i. Notice that the substring $R_x.w_1.Q_x.w_2.R_y$ or the substring $R_x.v.R_y$ (if the first query of x precedes the first reset of y in L_i) of $w.v^i$ is a *stratified* string. Hence reasoning as in the above cases we can obtain a contradiction. The other case when the first query of y precedes the first reset of y in L_i is dealt as follows. First note that the last query of x in L_i must precede the last reset of x in L_i. Second note that the last reset of y in L_i must occur after the last query of y in L_i. Third note that after the last reset of x in L_i there can be only one reset of y in L_i and no query of y in L_i(otherwise, reasoning as in case 1.2.1.2, we alrteady obtain a contradiction). Hence we can form the *stratified* substring $R_x.w_1.R_y$ of v^i where w_1 does not contain any reset or query. Hence, reasoning as in the above cases, we obtain a contradiction.

Case 2 $R_x \notin w$. This implies that $Q_x \notin w$.

We distinguish between the following subcases of this case.

Case 2.1 $R_y \notin w$. This implies that $Q_y \notin w$.

Thus w is a reset free cycle. Hence by Proposition 5.2 we obtain a contradiction.

Case 2.2 $R_y \in w$.

This case is symmetric to case 1.2 above.

‖

5.5 Future Work

The presented work targets theoretical investigations of timed automata not at the verification problem itself but, instead, at the termination behavior of the procedure solving it in practice, namely symbolic forward analysis. This work is a potential starting point for deriving interesting sufficient termination conditions. There are, however, other open questions along these lines.

Our setup may also be used to derive *necessary* termination conditions. These are useful obviously in the cases when their test is negative. Another question is whether there exist decidable necessary and sufficient conditions.

We may also consider logical equivalence instead of local subsumption for a practically more efficient, but theoretically weaker fixpoint test (used in tools such as Uppaal [LPY95b]). We observe that Proposition 5.1 is still directly applicable in the new context, but Proposition 5.2 is not. The comparison of the different fixpoint tests (equivalence, local and global subsumption) is an interesting subject of research.

We may be able to derive natural and less restrictive sufficient termination conditions when

we consider the enhancement of symbolic forward analysis with techniques from [Boi98] to compute the effect of loops, i.e. essentially the constraint transformer $[\![w^\omega]\!]$ for simple cycles w.

The constraint transformers $[\![w]\!]$ form a 'symbolic version' of the *syntactic monoid* [Eil76] for timed automata. This notion may be of intrinsic interest and deserve further study.

Chapter 6

Accurate Widenings and Boundedness Properties

6.1 Introduction

For the last ten years, the verification problem for timed systems has received a lot of attention (see e.g., [AD94, Bal96, DT98, LPY95b, WT95]). The problem has been shown to be decidable in [AD94]. Most of the verification approaches to this problem have been based either on a region graph, which is a finite quotient of the infinite state graph, or on some variants of it (that use convex/non-convex polyhedra and avoid explicit construction of the full graph). But, as we show below, region-graph based approaches (or its variants) cannot be used for dealing with *boundedness* (unboundedness) properties. This is due to the fact that the partitioning of the state space induced by the region equivalence (or any other technique that takes into account the maximal constant in the guards) is guaranteed to be *pre-stable* but may not be *post-stable* (definitions of pre-stability and post stability are provided in Chapter 2).

A boundedness property is of the form $\exists k \geq 0\ AG(x \leq k)$ where x is a clock (read this specification as: there exists a nonnegative k such that for all paths starting from the initial position, the value of the clock x does not exceed k throughout the path). An unboundedness property is the dual of a boundedness property: $\forall k \geq 0\ EF(x > k)$. These properties are useful in verification because if the designer knows that the value of a clock should never exceed a constant, then satisfaction of an unboundedness property by the design immediately informs the designer of a possible bug in the design. Also, the implementor can use this information to save some hardware while implementing the design (in hardware).

Consider the timed automaton (see [AD94] for a definition of timed automata) given in Figure 6.1 (it has two clocks x and y and four locations 0, 1, 2 and 3; the guards and resets for the edges are indicated at the top of or beside the edges; the invariants of the locations are indicated above or below the locations). Let us try to see whether the system satisfies the property $\exists k \geq 0\ AG(x \leq k)$, where the clock x in the formula refers to the clock x in the automaton. If we use the region graph technique, we will see that the regions (the maximal constant is 2 here) $(1 < y < 2, x > 2)$, $(y = 1, x > 2)$ and $(0 < y < 1, x > 2)$ (we do not enumerate all the reachable regions) are reachable. One may now conclude, on the basis of this reachability analysis, that the automaton does not satisfy the above boundedness property (note that all the three regions given above are unbounded). Unfortunately, this is not true; the value

of the clock x never exceeds 6 (just six)! Region graphs (or its variants) cannot be directly used for model checking for boundedness properties!

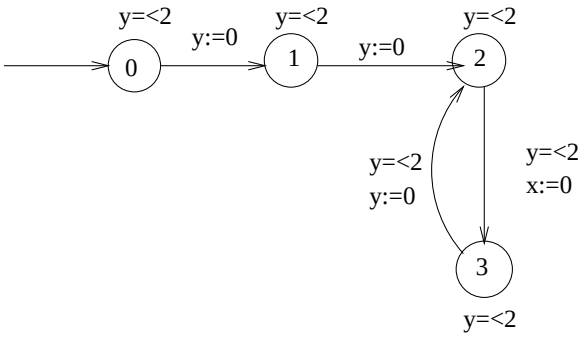


Figure 6.1: Illustrating Unboundedness (Boundedness) Property

Now consider a reachability analysis for this timed automaton using the algorithm in Figure 6.3 (this algorithm is a simple symbolic forward reachability analysis algorithm). The algorithm terminates generating the following set of reachable states: $\langle l\_0(x,y), x = 0, y = 0\rangle$, $\langle l\_0(x,y), x = y, y \geq 0, y \leq 2\rangle$, $\langle l\_1(x,y), 0 \leq x \leq 2, y = 0\rangle$, $\langle l\_1(x,y), x - y \geq 0, y - x \geq -2, y \geq 0, y \leq 2\rangle$, $\langle l\_2(x,y), 0 \leq x \leq 4, y = 0\rangle$, $\langle l\_2(x,y), x - y \geq 0, y - x \geq -4, y \geq 0, y \leq 2\rangle$, $\langle l\_3(x,y), 0 \leq y \leq 2, x = 0\rangle$ and $\langle l\_3(x,y), x - y \geq 0, y \leq 2, x \geq 0\rangle$ (the states are tuples of locations and constraint stores; we write l_i for the location i). It can be easily found out from the set of reachable states (by projecting the constraints on the x-axis) that the value of the clock x never goes beyond 6 and hence the above boundedness property is satisfied.

It can be shown that if the (symbolic) model checking algorithm in Figure 6.3 terminates, we can successfully model check for boundedness (unboundedness) properties. It is now natural to ask the question whether the procedure in Figure 6.3 is guaranteed to terminate. The answer is 'no'; consider the timed automaton in Figure 6.10 — the algorithm in Figure 6.3 will not terminate for this example (an infinite sequence of "states" which are not "included" in the "previously" generated states are produced). Of course, the procedure can be forced to terminate by including some maximal constant manipulation techniques (as the trim operation introduced in Chapter 3 or the extrapolation operation [DT98] or the preprocessing step [HKPV95]). But then, like the region graph technique, it can be shown that these techniques cannot be directly used for model checking for boundedness properties. So the natural thing now would be to develop techniques that force the termination of the procedure in Figure 6.3 (in cases where it is possible) but do not lose any information with respect to boundedness properties. It is in this context that *history-dependent constraint widenings* come into play.

Before introducing our framework of history-dependent constraint widenings (accurate widenings), let us try to see whether the already-existing *abstract interpretation* framework [CC77] can provide solutions to the problems described above. Abstraction interpretation techniques [CC77] are useful tools to force termination of the symbolic model checking procedures. Here one obtains a semi-test by introducing abstractions that yield a conservative approximation of the original property. Such methods have been successfully applied to many nontrivial examples [DT98, Bal96, WT95, HPR97]. While these abstractions force the termination of the model checking procedure, they sacrifice their accuracy in the process (note that

by accuracy, we mean not only accuracy with respect to reachability properties, but also with respect to boundedness properties). One of the most commonly used abstractions is the *convex hull* abstraction [WT95, DT98, Bal96].

The application of automated, application independent abstractions that enforce termination, as is done in program analysis, to model checking seems difficult for the reason that the abstractions are often too rough[1]. To know the accuracy of an abstraction is important both conceptually and pragmatically. As Wong-Toi observes in [WT95],

> ...The approximation algorithm proposed is clearly a heuristic. It would be of tremendous value to have analytical arguments for when it would perform well, for when it would not....

As we saw above, any symbolic model checking procedure that "loses" accuracy will not be able to model check for boundedness (unboundedness) properties. Hence, in this chapter, we propose a framework, to provide a partial answer to the question asked by Wong-Toi, viz., to determine automatically (using analytical methods) whether an abstraction performs well (does not lose accuracy) in a situation and then apply the abstraction.

We present methods that carry over the advantages of abstract interpretation techniques without losing precision. To be more specific, we apply history-dependent constraint widening techniques, as already foreseen in [CC77, CH78], to provide an application-independent abstract interpretation framework for model checking for timed systems. Basing our intuitions on techniques from Constraint Databases [JM94], we show that abstractions of the model checking fixpoint operator, through a set of widening rules, can yield an accurate model checking procedure. These abstractions are based on syntax of the constraints rather than their meaning (the solution space) in contrast with previous approaches (e.g., [Bal96, HPR97, WT95, BBR97]). As we demonstrate on examples, they can drastically reduce the number of iterations or even, in some cases, force termination of an otherwise non-terminating test. In contrast with the abstract interpretation techniques used for program analysis, they do not always force termination; instead their abstraction is accurate. That is, they do not lose information with respect to the original property; when they terminate, they provide information which is sufficient even for model checking for boundedness (unboundedness) properties; i.e., in cases where termination is achieved, the abstractions are sound and complete. Also, being based on the syntax of the constraints they can be implemented efficiently (they do not require computation of the convex hull like [WT95, Bal96, HPR97];). We first show toy examples in which our abstractions (henceforth called widening rules) either achieve termination in an otherwise non-terminating analysis or drastically accelerate the termination of symbolic forward reachability analysis.[2] We then show the performance of a prototype model checker, implemented using the techniques presented in this chapter, on some standard benchmark examples taken from literature. In the Conclusion, we discuss the generality of our approach.

[1]Note the statement of Halbwachs in [Hal93], that "Any widening operator is chosen under the assumption that the program behaves regularly .... Now the assumption of regularity is obviously abusive in one case: when a path in the loop becomes possible at step n, the effect of this path is obviously out of the scope of extrapolation before step n (since the actions performed on this path have never been taken into account) ... "

[2]Note that we consider forward analysis, instead of backward analysis, for the obvious advantages mentioned in [HKQ98] (Forward analysis is amenable to on-the-fly local model checking and also to partial order reductions. These methods ensure that only the reachable portion of the state space is explored). Moreover, backward analysis cannot be used for model checking for boundedness properties.

```
...
p: while(x<=100) do
     x:=x+1; od
q: ...
```

Figure 6.2: Fragment of the pseudo-code of a program

6.2 Intuition Behind the Accurate Widening Framework

Even before we delve into the details of timed systems, we try to give the reader a "feeling" of our accurate widening framework. Consider the fragment of a pseudo-code of an (integer-valued) program given in Figure 6.2 in which the only declared variable is x. Let us try to analyze this program fragment using constraints. More precisely, let us try to derive the constraints satisfied by the variable x at the program point q. Assume that initially the constraint on x at program point p is given by $x \leq 10$. Now let us step through the while loop once. The constraint on x at program point p is now $x \leq 11$. Stepping through the while loop again, the constraint on x at program point p becomes $x \leq 12$. From this, the abstract interpretation techniques described above [WT95, Bal96] (if we apply the same "widening" rules to this integer-valued program fragment) would derive *true* (i.e., constraint corresponding to the whole set of integers) as the constraint on x at the program point q. But unfortunately, this is not correct. The constraint on x at program point q is given by $x \leq 101$. The problem with the above techniques is that they don't seem to be able to detect that the while-loop in Figure 6.2 does not generate an "infinite sequence of constraints".

The accurate widening framework tries to "find out" from the syntax of a program loop (in cases in which it can) whether it really generates an infinite behavior (e.g., if the guard for the while loop was $x \geq 10$ instead of $x \leq 100$ it would indeed have generated an infinite behavior; thus a while loop with a guard $x \geq c$ for an integer c and an input constraint of the form $x \leq d$ where d is an integer and $d \geq c$ will indeed generate an infinite behavior). If it finds an infinite behavior, it applies widening rules to infer the "limit" (e.g., if the infinite sequence is $x \leq 1$, $x \leq 2$, ..., then the limit of the infinite sequence is *true*) of the infinite sequence. If it does not find an infinite behavior, it either "tries" to infer the "limit" of the finite sequence (based on the syntax) and speed up the computation procedure (in the program in Figure 6.2 it will infer $x \leq 101$) or (if it fails to do so) simply does not apply any widening at all. Note that such an accurate widening framework will be too restricted for integer-valued programs that are Turing complete (it may fail to find out an infinite behavior and hence may not use any widening to accelerate the convergence; it is for this reason that we cannot guarantee termination). But, as we show below, for a large class of timed systems it can be applied to great effect. Note also that if a widening is applied according to the norms specified by the accurate widening framework, it does not lose any information; it is accurate. It is this accuracy that we prove in theorem 6.1. We also provide sufficient conditions under which the accurate widenings are guaranteed to force the termination of the model checking procedure.

Procedure Symbolic-Boundedness(Φ)
Input A set of constraints Φ
Output A set of constraints representing sets of states reachable from $[\Phi]$
$\Phi_0 := \Phi$.
repeat
begin
$\Phi_{i+1} = \Phi_i \cup post(\Phi_i)$
end
until $\Phi_{i+1} \models \Phi_i$.
return Φ_i.

Figure 6.3: Template for Model Checking for Boundedness Properties

Procedure Symbolic-Boundedness-W(Φ)
Input A set of constraints Φ
Output A set of constraints representing sets of states reachable from $[\Phi]$
$\Phi_0 := \Phi$.
repeat
begin
$\Phi_{i+1} = \Phi_i \cup WIDEN(\Phi_i, post(\Phi_i))$
end
until $\Phi_{i+1} \models \Phi_i$.
return Φ_i.

Figure 6.4: Template for Model Checking for Boundedness Properties with Widening

Function WIDEN$(\Gamma, \Phi) = \{WIDEN(\gamma, \varphi) \mid \gamma \in \Gamma, \varphi \in \Phi\}$
Function $WIDEN(\gamma, \varphi)$
$\varphi_1 := WIDEN_1(\gamma, \varphi)$
If $\varphi_1 \not\equiv \varphi$ **return** φ_1
else $\{\varphi_1 := WIDEN_2(\gamma, \varphi)$
If $\varphi_1 \not\equiv \varphi$ **return** φ_1
else $\varphi_1 := WIDEN_3(\gamma, \varphi)\}$
return φ_1

Figure 6.5: Widen Function

6.3 Timed Automata, Constraints and Model Checking

For the purposes of this chapter, we model timed systems using timed automata. Recall the notion of timed automata from Chapter 3.

We now fix the formal set up of this chapter. We use lower case Greek letters for a constraint and upper case Greek letters for a set of constraints (which stands for their disjunction). The interpretation domain for our constraints is $\mathcal{R}$ the set of reals. We write $\mathbf{x}$ for the tuple of variables $x_1, \ldots, x_n$ and $\mathbf{v}$ for the tuple of values $v_1, \ldots, v_n$. As usual, $\mathcal{R}, \mathbf{v} \models \varphi$ is the validity of the formula φ under the valuation $\mathbf{v}$ of the variables $x_1, \ldots, x_n$. We formally define the relation denoted by a constraint φ as:

$$[\varphi] = \{\mathbf{v} \mid \mathcal{R}, \mathbf{v} \models \varphi\}$$

Note that $x_1, \ldots, x_n$ act as the free variables of φ and implicitly all other variables are existentially quantified. We write $\varphi[\mathbf{x}']$ for the constraint obtained by alpha-renaming from φ. We define $[\Phi]$, the relation denoted by a set of constraints Φ with respect to variables $x_1, \ldots, x_n$ in the canonical way. For a constraint φ and a set of constraints $\{\psi_1, \ldots, \psi_k\}$, we write $\varphi \models \bigvee_{i=1}^{k} \psi_i$ iff $[\varphi] \subseteq \bigcup_{i=1}^{k} [\psi_i]$. For sets of constraints Φ_1 and Φ_2 (where by a set of constraints $\Phi = \{\varphi_i\}$, we mean $\bigvee_i \varphi_i$), we write $\Phi_1 \models \Phi_2$ if for all $\varphi \in \Phi_1$ there exists $\varphi' \in \Phi_2$ such that $[\varphi] \subseteq [\varphi']$ (equivalently, in such a case, we say that there exists a *local inclusion abstraction* from Φ_1 to Φ_2 or Φ_1 is locally included in Φ_2; see below for a formal definition of local inclusion abstraction; it is this local inclusion that we will use in our symbolic model checking procedures; see below). We write an event (an edge transition or a time transition or a composition of several edge and time transitions) as **cond** ψ **action** φ, where the guard ψ is a constraint over $x_1, \ldots, x_n$ and the action φ is a constraint over the variables $x_1, \ldots, x_n$ and $x'_1, \ldots, x'_n$. The primed variable x' denotes the value of the variable x in the successor state. Note that we use interleaving semantics for our model. **We will use a set of constraints Φ to represent a set of states $\mathcal{S}$ if** $\mathcal{S} = [\Phi]$. The successor of a set of states of such a set with respect to an event e are represented by the constraints obtained by conjoining the guard ψ and the action φ of each event with each constraint φ of Φ:

$$post_{|e}(\Phi) = \{\exists_{-\mathbf{x}'} \varphi \wedge \psi \wedge \varphi \mid \varphi \in \Phi, \mathcal{R} \models \varphi \wedge \psi \wedge \varphi\}$$

where the existential quantifier is over all variables but $\mathbf{x}'$. Note that the $post_{|e}$ operation can be easily implemented using well known algorithms for variable elimination from constraint programming (eg. Fourier's algorithm [LM92, MS98] or Weispfenning's algorithm [Wei94]) and polynomial time algorithms for testing satisfiability of linear constraints over reals [MS98].

We next formulate possibly non-terminating symbolic model checking procedures for boundedness properties, in our constraint-based framework, based on local inclusion abstraction (see the definition below). The template for the algorithm is given in Figure 6.3. Here $post(\Phi) = \cup_{e \in \mathcal{E}} post_{|e}(\Phi)$ where $\mathcal{E}$ is the set of all events of the timed system (*simple* and *compound*; see below for definitions of compound (composed) events). The local inclusion abstraction (local subsumption or constraint entailment, see below for a formal definition of inclusion abstraction) in the algorithm can be implemented using standard polynomial time algorithms for (local) constraint entailment [Sri92, MS98]. The algorithm is basically a (inflationary) fixpoint computation algorithm. Note that the template Symbolic-Boundedness can be used for model checking for the logic $\mathcal{L}_s$ [LPY95b]. Also note that the algorithm is breadth

first. In the sequel, we call the algorithm Symbolic-Boundedness as the breadth first (symbolic forward) reachability analysis algorithm with (local) inclusion abstraction. Recall the notion of zones from Chapter 3. It can be easily shown (by using Fourier's Algorithm [MS98, LM92] for example) that for timed automata, the sets of reachable states in a symbolic forward reachability analysis can be represented by zones.

6.3.1 Inclusion Abstractions

We first note that the locations of a timed automaton can be encoded as finite domain constraints (in our algorithms we assume that the locations are encoded as finite domain constraints). We denote a position (simply a state) [AD94, HK97] of the timed automaton having location component ℓ as $\ell(\mathbf{v})$ where $\mathbf{v}$ denotes the values of the clocks. In general, for a set $\mathcal{S}$ of states having the location component ℓ, we write $\langle \ell, S\rangle$, or $\langle \ell(\mathbf{x}), \varphi\rangle$, where φ is a zone constraint and $S = [\varphi] = \{\mathbf{v} \mid \ell(\mathbf{v}) \in \mathcal{S}\}$. Here the free variables of φ are $\{x_1, \ldots, x_n\}$. In the sequel, we will refer to a set of states with location component ℓ and represented by $\langle \ell(\mathbf{x}), \varphi\rangle$ as a symbolic state or simply a state when it is clear from context. For a timed automaton U, we denote the set of all reachable symbolic states by $\mathcal{S}^U_{symb}$.

Definition 6.1 (Local Inclusion Abstraction.) *Given a timed automaton U, we say that $\alpha_{inc} : \mathcal{S}^U_{symb} \longrightarrow \mathcal{S}^U_{symb}$ is a local inclusion abstraction iff, for any $\langle \ell(\mathbf{x}), \varphi\rangle \in \mathcal{S}^U_{symb}$, $\langle \ell(\mathbf{x}), \varphi\rangle \prec \alpha_{inc}(\langle \ell(\mathbf{x}), \varphi\rangle)$ where for any two states $\langle \ell(\mathbf{x}), \varphi\rangle$ and $\langle \ell'(\mathbf{x}), \varphi'\rangle$, $\langle \ell(\bar{x}), \varphi\rangle \prec \langle \ell'(\mathbf{x}), \varphi'\rangle$ iff ℓ and ℓ' are identical and $[\varphi] \subseteq [\varphi']$ (equivalently $\varphi \models \varphi'$). Given $\mathcal{S}, \mathcal{S}' \subseteq \mathcal{S}^U_{symb}$, we say that α_{inc} is a local inclusion abstraction from $\mathcal{S}$ to $\mathcal{S}'$ iff for all $\langle \ell(\mathbf{x}), \varphi\rangle \in \mathcal{S}$ there exists $\langle \ell(\mathbf{x}), \varphi'\rangle \in \mathcal{S}'$ such that $\alpha_{inc}(\langle \ell(\mathbf{x}), \varphi\rangle) = \langle \ell(\mathbf{x}), \varphi'\rangle$.*

Definition 6.2 (Global Inclusion Abstraction [DT98].) *We say that $\alpha^g_{inc} : \mathcal{S}^U_{symb} \longrightarrow 2^{\mathcal{S}^U_{symb}}$ is a global inclusion abstraction iff, for any $\langle \ell(\mathbf{x}), \varphi\rangle \in \mathcal{S}^U_{symb}$, $\alpha^g_{inc}(\langle \ell(\mathbf{x}), \varphi\rangle) \sqsupseteq \langle \ell(\mathbf{x}), \varphi\rangle$, where for $S \subseteq \mathcal{S}^U_{symb}$, $\langle \ell(\mathbf{x}), \varphi\rangle \sqsubseteq S$ iff $[\varphi] \subseteq \bigcup_{\langle \ell(\mathbf{x}), \varphi'\rangle \in S} [\varphi']$ (or equivalently $\varphi \models \bigvee_{\langle \ell(\mathbf{x}), \varphi'\rangle \in S} \varphi'$).*

There is a local inclusion abstraction from the constraint $x \geq 5$ to the constraint $x \geq 4$ since $(x \geq 5) \models (x \geq 4)$ (we do not show the locations which are assumed to be the same). There is a local inclusion abstraction from the set of constraints $\{4 \leq x \leq 5, 1 \leq x \leq 2, 16 \leq x \leq 20\}$ (where by a set of constraints , we denote their disjunction) to the set of constraints $\{0 \leq x \leq 5, x \geq 16\}$ since $(4 \leq x \leq 5) \models (0 \leq x \leq 5)$, $(1 \leq x \leq 2) \models (0 \leq x \leq 5)$ and $(16 \leq x \leq 20) \models (x \geq 16)$. We write $\{4 \leq x \leq 5, 1 \leq x \leq 2, 16 \leq x \leq 20\} \models \{0 \leq x \leq 5, x \geq 16\}$. On the other hand, there is a global inclusion abstraction from φ_2 to the set of constraints $\{\varphi_1, \varphi_3\}$ where $\varphi_1 \equiv x_1 - x_2 \geq 0 \wedge x_2 - x_1 \geq -2 \wedge x_2 \geq 0 \wedge x_2 \leq 2$, $\varphi_2 \equiv x_1 - x_2 \geq 1 \wedge x_2 - x_1 \geq -3 \wedge x_2 \geq 0 \wedge x_2 \leq 2$ and $\varphi_3 \equiv x_1 - x_2 \geq 2 \wedge x_2 - x_1 \geq -4 \wedge x_2 \geq 0 \wedge x_2 \leq 2$ since $\varphi_2 \models \varphi_1 \vee \varphi_3$. Note that the existence of a local inclusion abstraction from a set of constraints Φ_1 to another set of constraints Φ_2 entails the existence of a global inclusion abstraction from Φ_1 to Φ_2. But the converse is not true unless the underlying constraint domain satisfies the independence property (see below).

Note that we have considered only local inclusion abstraction for our algorithm ($\Phi_{i+1} \models \Phi_i$ denotes that there is a local inclusion abstraction from Φ_{i+1} to Φ_i). It may happen that the breadth first forward reachability analysis terminates with the weaker condition of global inclusion but not with local inclusion. Consider the example shown in Figure 6.7. The breadth first forward reachability analysis procedure terminates for this example with global inclusion

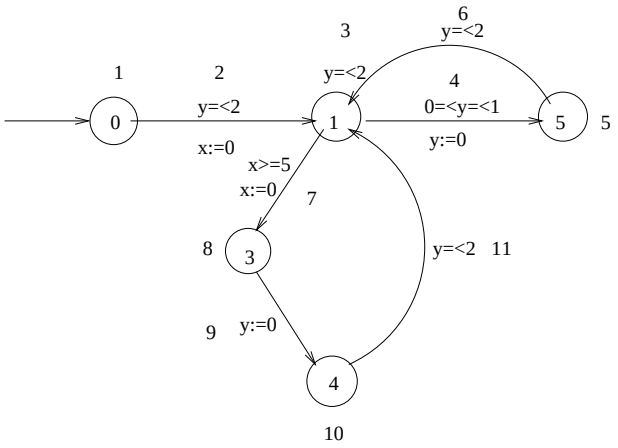


Figure 6.6: Illustrating Accelarating Effect of Widening Rules

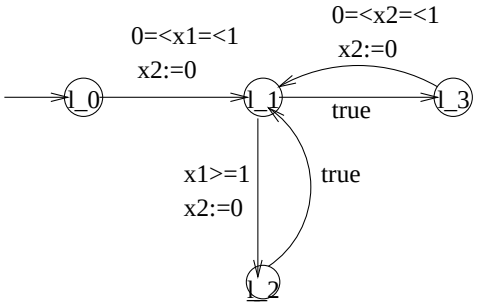


Figure 6.7: Local and Global Inclusion Abstraction

abstraction but not with local inclusion abstraction. However, for constraint domains that do not satisfy the *independence property*[3], deciding global inclusion is usually very expensive (co-NP hard [Sri92] for this particular case) where as local inclusion is decidable in polynomial time (for this particular case [Sri92]). To see that our constraint domain does not satisfy the independence property, consider the constraints $\varphi_1 \equiv x_1 - x_2 \geq 0 \wedge x_2 - x_1 \geq -2 \wedge x_2 \geq 0 \wedge x_2 \leq 2$, $\varphi_2 \equiv x_1 - x_2 \geq 1 \wedge x_2 - x_1 \geq -3 \wedge x_2 \geq 0 \wedge x_2 \leq 2$ and $\varphi_3 \equiv x_1 - x_2 \geq 2 \wedge x_2 - x_1 \geq -4 \wedge x_2 \geq 0 \wedge x_2 \leq 2$. It is clear that $\varphi_2 \models \varphi_1 \vee \varphi_3$ but $\varphi_2 \not\models \varphi_1$ and $\varphi_2 \not\models \varphi_3$.

6.4 Widening Rules

In this section, we consider how one can achieve (or just speed up) termination of the breadth first forward reachability analysis algorithms for boundedness (as well as safety) properties. We define widening rules that are accurate i.e., do not lose information with respect to the original property. We show that these widening rules can be used to achieve termination in cases where termination is not guaranteed in forward analysis with local inclusion abstraction. We also

[3]A constraint domain is said to satisfy the independence property [MS98] if for any constraint φ and a set of constraints $\{\varphi_1, \ldots, \varphi_n\}$, $\varphi \models \varphi_1 \vee \ldots \vee \varphi_n$ implies $\varphi \models \varphi_i$ for some i.

Function $\text{WIDEN}_1(\gamma, \varphi')$

$$
\begin{array}{l}
\textit{if} \left\{ \begin{array}{l}
\gamma \equiv \eta \wedge x_i - x_j \geq c_{ij} \\
\varphi \wedge \psi \equiv \theta \wedge x'_j = x_j + x'_i \wedge x_i \leq c_i \\
c_{ij} \leq 0 \\
c_i > 0 \\
\eta[\mathbf{x}'] \models (\exists_{-\mathbf{x}'}(\eta \wedge \varphi \wedge \psi)) \wedge (\exists_{-\mathbf{x}'}\theta \wedge x_i - x_j \geq c_{ij} \wedge x_i \leq c_i)
\end{array} \right. \\
\textit{or if} \left\{ \begin{array}{l}
\gamma \equiv \eta \wedge x_i - x_j \geq c_{ij} \wedge x_i \leq c_i \\
\varphi \wedge \psi \equiv \theta \wedge x'_j = x_j + x'_i \\
c_{ij} \leq 0 \\
c_i > 0 \\
\eta[\mathbf{x}'] \models (\exists_{-\mathbf{x}'}(\eta \wedge \varphi \wedge \psi)) \wedge (\exists_{-\mathbf{x}'}\theta \wedge x_i - x_j \geq c_{ij} \wedge x_i \leq c_i)
\end{array} \right. \\
\textit{return } \eta \\
\textit{else return } \varphi'
\end{array}
$$

Figure 6.8: Widening Rule I

show that for some examples for which termination of forward analysis with (local) inclusion abstraction is guaranteed, but widening can drastically accelerate the termination.

In general, the events considered here may not be an original event but is constructed as a composition of events. We write $e = \mathit{event}(\gamma, \varphi)$ when application of the event e to the constraint γ results in the constraint φ.

Definition 6.3 (Compound Events.) *Let* $e_1 \equiv$ **cond** ψ_1 **action** $\varphi_1, \ldots, e_k \equiv$ **cond** ψ_k **action** φ_k *be k original events of the timed system. Assume that the source location for the first event and the target location for the last event are the same. Assume that the target location for the jth event and the source location for the $(j+1)$st event are same ($1 \leq j \leq k-1$). Also assume that for each event e_j, each variable x_i and x'_i in the guard and action have been alpha-renamed to x_i^j and x_i^{j+1} respectively. Then the compound event (or composed event) corresponding to $e_1, \ldots, e_k$ is given by*[4] **cond** *true* **action** $\varphi \wedge \psi$ *where* $\varphi \wedge \psi$ *is given by*

$$\varphi \wedge \psi \equiv (\exists_{-\{\mathbf{x^1}, \mathbf{x^{k+1}}\}} \varphi_1 \wedge \psi_1 \wedge \ldots \wedge \varphi_k \wedge \psi_k)[\mathbf{x}, \mathbf{x}'].$$

See below for examples of compound events. Given that the theory of reals with addition and order admits quantifier elimination, $\varphi \wedge \psi$ can be expressed in a conjunctive normal form. For the timed automaton given in Figure 6.1, the event **cond** *true* **action** $x' = x + z, z \geq 0, y' = y + z, y' \leq 2$ is a simple (time) event corresponding to the time transition in location 0 while **cond** $loc = 0$ **action** $y' = 0, x' = x, loc' = 1$ is a simple or original (edge) event corresponding to the edge from location 0 to 1.

We consider only non-strict inequalities here. The strict inequalities can be dealt with similarly. The template for symbolic boundedness procedure with widening is defined in Figure

[4]Note that we can construct an event with empty guard as the events **cond** ψ **action** φ and **cond** *true* **action** $\varphi \wedge \psi$ are equivalent with respect to symbolic model checking

Function $\mathrm{WIDEN}_2(\gamma, \varphi')$

$$
\textit{if} \begin{cases}
\gamma \equiv \eta \wedge x_i - x_j \geq c_{ij} \wedge x_j - x_i \geq c_{ji} \\
\varphi \wedge \psi \equiv \theta \wedge x'_j - x'_i \leq x_j - x_i + a_{ji} \\
c_{ij} \leq 0 \\
a_{ji} > 0 \\
(\exists_{-\mathbf{x}'} \varphi \wedge \psi \wedge \eta) \wedge (\exists_{-\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij} \wedge x_j - x_i \geq c_{ji}) = \zeta \wedge x'_j - x'_i \geq c'_{ji} \\
\eta[\mathbf{x}'] \models \zeta \\
0 \leq c'_{ji} \leq -c_{ij}
\end{cases}
$$

return $\eta \wedge x_j - x_i \geq c_{ji}$

$$
\textit{else if} \begin{cases}
\gamma \equiv \eta \wedge \bigwedge_{i,j,\in I} x_i - x_j \geq c_{ij} \\
\varphi \wedge \psi \equiv \theta \wedge \bigwedge_{i,j,\in I} x_j - x_i \leq x_j - x_i + a_{ji} \\
c_{ij} \leq 0 \\
a_{ji} > 0 \\
\eta[\mathbf{x}'] \models (\exists_{-\mathbf{x}'} \varphi \wedge \psi \wedge \eta) \wedge (\exists_{-\mathbf{x}'} \theta \wedge \bigwedge_{i,j,\in I} x_i - x_j \geq c_{ij})
\end{cases}
$$

return η.

else return φ'

Figure 6.9: Widening Rule II

6.4. Note that the procedure is based on a breadth-first search. The function $WIDEN$ is defined in Figure 6.5 in the Appendix. In a call to $WIDEN(\Phi_i, post(\Phi_i))$ one of the three widening rules provided the conditions of that rule are satisfied. If the condition in the $WIDEN$ function applies to several decompositions of γ, the corresponding widenings are effectuated in several successive iterations. In the sequel, we refer to the procedure Symbolic-Boundedness-W as the breadth first forward reachability analysis procedure with widening and (local) inclusion abstraction. Note that the termination condition $\Phi_{i+1} \models \Phi_i$ means that there is a local inclustion abstraction from Φ_{i+1} to Φ_i.

We now illustrate the widening rules with examples. The intuition behind the widening rules is as follows: if we can detect from the syntax of a sequence of events $\bar{e}$ and a constraint φ, that the sequence $\varphi, post_{|\bar{e}}(\varphi), \ldots$ "grows" infinitely in a particular direction (i.e., actually leads to an infinite sequence with respect to reachability analysis), we will try to add the union of the sequence to our set of reachable states. Thus for widening rule I (for the if part), the syntax of the input constraint $(\eta \wedge x_i - x_j \geq c_{ij})$ and that of the event $(\theta \wedge x'_j = x_j + x'_i \wedge x_i \leq c_i$ which may be a composition of several simple events as described above) tells us that this constraint-event combination will generate an infinite behavior ($\eta \wedge x_i - x_j \geq c_{ij}$, $\eta \wedge x_i - x_j \geq c_{ij} - c_i$, $\ldots$; see example below) provided the other conditions are satisfied (compare the while-loop example in Section 6.2). Hence we infer the limit of this sequence which is η (since $c_{ij} \leq 0$ and $c_i > 0$) and add it to the set of states. Similar are the intuitions behind the other widening rules.

Consider the example timed automaton in Figure 6.10. Note that forward breadth-first reachability analysis with local inclusion abstraction does not terminate. Consider the events 4 and 3. Event 4 is given by $e \equiv$ **cond** $x_2 \leq 2$ **action** $x'_2 = 0, x'_1 = x_1$ (we do not show the location explicitly). Event 3 is the time event at location 1 and is given by $e' \equiv$ **cond** $true$ **action** $x'_1 =$

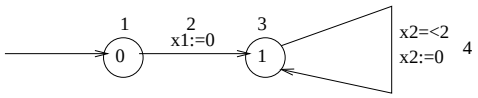


Figure 6.10: Illustrating widening rule I

$x_1 + z, x_2' = x_2 + z, z \geq 0$ (time increases by amount z). We compose transition (sometimes we will use the term 'transition' for 'event') 4 and transition 3 using the method given above. The resulting compound event is $e_1 \equiv$ **cond** *true* **action** $\varphi \wedge \psi$ where

$$\varphi \wedge \psi \equiv x_1' = x_1 + x_2', x_2 \leq 2, x_2' \geq 0.$$

Now consider the infinite sequence of states produced by a breadth-first reachability analysis for this automaton

$$\begin{aligned}
&\langle l_0(\mathbf{x}), x_1 = 0, x_2 = 0\rangle \xrightarrow{1} \langle l_0(\mathbf{x}), x_1 = x_2, x_1 \geq 0\rangle \\
&\xrightarrow{2} \langle l_1(\mathbf{x}), x_1 = 0, x_2 \geq 0\rangle \xrightarrow{3} \langle l_1(\mathbf{x}), x_2 - x_1 \geq 0, x_1 \geq 0\rangle \\
&\xrightarrow{4} \langle l_1(\mathbf{x}), 0 \leq x_1 \leq 2, x_2 = 0\rangle \xrightarrow{3} \overbrace{\langle l_1(\mathbf{x}), x_1 - x_2 \geq 0, x_2 \geq 0, x_2 - x_1 \geq -2\rangle} \\
&\xrightarrow{4} \langle l_1(\mathbf{x}), 0 \leq x_1 \leq 4, x_2 = 0\rangle \xrightarrow{3} \langle l_1(\mathbf{x}), x_1 - x_2 \geq 0, x_2 \geq 0, x_2 - x_1 \geq -4\rangle \xrightarrow{4} \ldots
\end{aligned}$$

(in the above we denote location i by l_i.) Now see that the state under the overbrace along with event e_1 satisfies the conditions of the widening rule I (the *if* part) defined in Figure 6.8 ($i = 2, j = 1$, $\gamma \equiv \eta \wedge x_2 - x_1 \geq -2$ where $\eta \equiv x_1 - x_2 \geq 0, x_2 \geq 0$, $c_{21} = -2$ and $\theta \equiv x_2' \geq 0$). Hence, applying the widening, we obtain the state $\langle l_1(\mathbf{x}), x_1 - x_2 \geq 0, x_2 \geq 0\rangle$ (the reader can easily make out that if the sequence of transition 4 and transition 3 is repeated infinitely many times to the state under the overbrace, the constraint $x_1 - x_2 \geq 0, x_2 \geq 0$ will be obtained). After this any state generated is subsumed (included) by this state. Hence the breadth first forward reachability analysis with widening and local inclusion abstraction terminates.

To see the accelerating effect of widening rule I on the convergence of reachability analysis in case of examples for which breadth-first analysis terminates with local inclusion abstraction, consider the example in Figure 6.6. If we replace the constant 5 in transition 7 by 10000 say , breadth first forward analysis with local inclusion abstraction will terminate in approximately in 10000 iterations. But, it can be easily seen that composing transitions 4, 5 and 6 gives rise to a compound event and by using widening rule I on this compound event (it is easy to verify that the conditions in the widening rule will be satisfied) breadth first forward analysis with widening and local inclusion abstraction terminates in 11 iterations.

Before defining widening rule II, let us introduce some notation. Let $\mathcal{N}_n$ denote $\{1, \ldots, n\}$. Let I denote a subset of $\mathcal{N}_n$. The widening rule II is defined in figure 6.9.

To show an example in which application of widening rule II forces termination, we look at the example in figure 6.11. Note that breadth-first forward reachability analysis with local inclusion abstraction does not terminate for this example. The following infinite sequence of

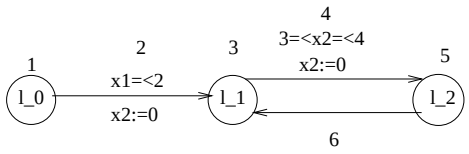


Figure 6.11: Illustrating widening rule II

states is generated in a breadth-first forward reachability analysis for this example.

$$\langle l\_0(\mathbf{x}), x_1 = 0, x_2 = 0\rangle \xrightarrow{1} \langle l\_0(\mathbf{x}), x_1 = x_2, x_2 \geq 0\rangle$$
$$\xrightarrow{2} \langle l\_1(\mathbf{x}), x_1 \geq 0, x_1 \leq 2, x_2 = 0\rangle \xrightarrow{3} \langle l\_1(\mathbf{x}), x_1 - x_2 \geq 0, x_2 - x_1 \geq -2, x_2 \geq 0\rangle$$
$$\xrightarrow{4} \langle l\_2(\mathbf{x}), x_1 \geq 3, x_1 \leq 6, x_2 = 0\rangle \xrightarrow{5} \langle l\_2(\mathbf{x}), x_1 - x_2 \geq 3, x_2 - x_1 \geq -6, x_2 \geq 0\rangle$$
$$\xrightarrow{6} \overbrace{\langle l\_1(\mathbf{x}), x_1 - x_2 \geq 3, x_2 - x_1 \geq -6, x_2 \geq 0\rangle} \xrightarrow{3} \langle l\_1(\mathbf{x}), x_1 - x_2 \geq 3, x_2 - x_1 \geq -6, x_2 \geq 0\rangle$$
$$\xrightarrow{4} \langle l\_2(\mathbf{x}), x_1 \geq 6, x_1 \leq 10, x_2 = 0\rangle \xrightarrow{5} \langle l\_2(\mathbf{x}), x_1 - x_2 \geq 6, x_2 - x_1 \geq -10, x_2 \geq 0\rangle$$
$$\xrightarrow{6} \langle l\_1(\mathbf{x}), x_1 - x_2 \geq 6, x_2 - x_1 \geq -10, x_2 \geq 0\rangle \ldots$$

Now consider the compound event $e_2 \equiv$ **cond** *true* **action** $\varphi \wedge \psi$ obtained by composing transitions 3, 4, 5 and 6. Here

$$\varphi \wedge \psi \equiv x_1' \geq x_1 - x_2 + x_2' + 2 \wedge x_1' - x_2' \leq x_1 - x_2 + 3 \wedge x_1' \geq x_1 + x_2' \wedge x_2' \geq 0.$$

See that the conditions of widening rule II (the *if* part) are satisfied for e_2 and the state under the overbrace in the sequence ($i = 2, j = 1$, $\eta \equiv x_2 \geq 0$, $c_{21} = -6 < 0$, $c_{12} = 3$ and $\theta \equiv x_1' \geq x_1 - x_2 + x_2' + 2, x_1' \geq x_1 + x_2', x_2' \geq 0$). The reader can easily convince herself that the give state and event e_2 do not satisfy the conditions of widening rule I). Applying the widening, we obtain the state $\langle l\_1(\mathbf{x}), x_1 - x_2 \geq 3, x_2 \geq 0\rangle$ (viewing the constraint solving involved geometrically may provide better intuitions). The states which are further generated are subsumed by this state. So breadth-first forward reachability analysis with widening and inclusion abstraction terminates after this. Note that in this case, application of abstract interpretation with the convex hull operator as is done in [WT95, Bal96, HPR97] would produce the state $\langle l\_1(\mathbf{x}), x_1 - x_2 \geq 0, x_2 \geq 0\rangle$. This can lead to 'don't know' answers to certain reachability questions (e.g., consider the reachability question whether the location l_1 can be reached with the values of the clocks satisfying the constraint $x_1 - x_2 > 2, x_2 - x_1 > -3, x_2 \geq 0$). As for the extrapolation abstraction [DT98], we have already stated in the Introduction that it is unsuitable for model checking for boundedness properties.

In widening rule III we use periodic sets following Boigelot and Wolper [BW94].

Definition 6.4 (Periodic Sets [BW94].) *A periodic vector set or simply a periodic set is a set of vectors* $\mathbf{x} \in \mathcal{R}^n$ *such that*

$$\exists \mathbf{k} \in \mathcal{N}^m \ : \ \mathbf{x} = C\mathbf{k} + \mathbf{d} \wedge P\mathbf{k} \leq \mathbf{q}$$

where C *and* P *integer matrices.*

The widening rule III is defined in Figure 6.12, where the predicate $int(x)$ is true if and only if x is a nonnegative integer. Consider the example in Figure 6.13. Note that breadth-first

Function $\text{WIDEN}_3(\gamma, \varphi')$

$$if \begin{cases} \gamma \equiv \eta \wedge x_i - x_j \geq c_{ij} \wedge x_j - x_i \geq c_{ji} \\ \varphi \wedge \psi \equiv \theta \wedge x_i' = x_i + x_j' \wedge x_j = c_j \\ c_{ij} \leq 0 \\ c_i > 0 \\ c_{ji} \geq c_{ij} \\ \eta[\mathbf{x}'] \models (\exists_{-\mathbf{x}'}(\eta \wedge \varphi \wedge \psi)) \wedge (\exists_{-\mathbf{x}'}\theta \wedge x_i - x_j \geq c_{ij} \wedge x_j - x_i \geq c_{ji} \wedge x_j = c_j) \end{cases}$$

$return\ \eta \wedge \exists k \geq 0 \wedge int(k) \wedge x_i - x_j \geq c_{ji} + k * c_j \wedge x_j - x_i \geq c_{ji} - k * c_j$

$else\ return\ \varphi'$

Figure 6.12: Widening Rule III

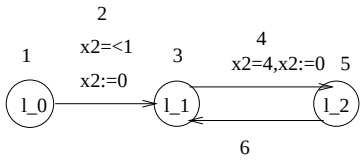


Figure 6.13: Illustrating the widening rule III

forward reachability analysis with inclusion abstraction does not terminate for this example. The following infinite sequence of states is generated in course of a forward (breadth-first) reachability analysis for this example:

$$\begin{aligned} &\langle l\_0(\mathbf{x}), x_1 = 0, x_2 = 0\rangle \xrightarrow{1} \langle l\_0(\mathbf{x}), x_1 = x_2, x_2 \geq 0\rangle \\ &\xrightarrow{2} \langle l\_1(\mathbf{x}), x_2 = 0, x_1 \geq 0, x_1 \leq 1\rangle \xrightarrow{3} \overbrace{\langle l\_1(\mathbf{x}), x_1 - x_2 \geq 0, x_2 - x_1 \geq -1, x2 \geq 0\rangle} \\ &\xrightarrow{4} \langle l\_2(\mathbf{x}), x_1 \geq 4, x_1 \leq 5, x_2 = 0\rangle \xrightarrow{5} \langle l\_2(\mathbf{x}), x_1 - x_2 \geq 4, x_2 - x_1 \geq -5, x_2 \geq 0\rangle \\ &\xrightarrow{6} \langle l\_1(\mathbf{x}), x_1 - x_2 \geq 4, x_2 - x_1 \geq -5, x_2 \geq 0\rangle \ldots \end{aligned}$$

Now we compose transitions 4, 5 and 6. The compound event is $e_3 \equiv$ **cond** $true$ **action** $\varphi \wedge \psi$ where

$$\varphi \wedge \psi \equiv x_1' = x_1 + x_2' \wedge x_2' \geq 0 \wedge x_2 = 4.$$

It is easy to see that the state under the overbrace in the infinite sequence along with event e_3 satisfies the conditions of widening rule III ($i = 2, j = 1$, $\eta \equiv x_2 \geq 0$, $c_{12} = 0$, $c_{21} = -1 < 0$). Hence, applying widening rule III we get the state $\langle l\_1(\mathbf{x}), \exists k \geq 0, int(k), x_1 - x_2 \geq k * 4, x2 - x_1 \geq -1 - k * 4\rangle$. The states further generated are subsumed by this state. So (breadth-first) forward reachability analysis with local inclusion abstraction terminates after applying the widening rule. Note that application of abstract interpretation with the convex hull operator [HPR97, Bal96, WT95] will produce the state $\langle l\_1(\mathbf{x}), x_1 - x_2 \geq 0, x_2 \geq 0\rangle$. Hence for certain reachability questions we can get a 'don't know' answer.

Now we show that the widening rules are accurate with respect to boundedness properties.

Theorem 6.1 (Soundness and Completeness) *The procedure Symbolic-Boundedness-W obtained by abstracting the forward breadth first reachability analysis procedure with widening defined by the widening rules I,II and III yields (if terminating) a full test of boundedness (unboundedness) properties for timed systems (modeled by timed automata).*

Proof. The proof works by showing that $[WIDEN_j(\gamma,\varphi)] \subseteq [\bigcup_{i\geq 0}\Phi_i]$ for $j = 1,2,3$ and $\gamma \in \Phi_i$ and $\varphi \in post(\Phi_i)$ where $\Phi_0 = \Phi$ and for $i > 0$, $\Phi_{i+1} = \Phi_i \vee post(\Phi_i)$. Before proving the theorem, we consider the following properties. Let e be an event with guard ψ and action φ from location ℓ to itself. Then

$$post_e(S) = \{\ell(\mathbf{v}') \mid \mathcal{R}, \mathbf{v}, \mathbf{v}' \models \varphi \wedge \psi[\mathbf{x},\mathbf{x}'], \ell(\mathbf{v}) \in S\}$$

where S is a set of states $\ell(\mathbf{v})$ of the timed automaton consisting of a location and valuations to all the clocks. Now

$$post_{|e}(\Phi) = \{\exists_{-\mathbf{x}'}\varphi \wedge \psi[\mathbf{x},\mathbf{x}'] \wedge \varphi \mid \varphi \in \Phi, \mathcal{R} \models \varphi \wedge \psi[\mathbf{x},\mathbf{x}'] \wedge \varphi\}$$

where Φ is a set of constraints and $S = [\Phi]$. Notice that $post_e$ and $post_{|e}$ are monotonic with respect to inclusion and continuous with respect to set union.

$$\bigcup_{i\geq 0} post_e^i([\Phi]) = \bigcup_{i\geq 0}[post_{|e}^i(\Phi)] = [\bigcup_{i\geq 0} post_{|e}^i(\Phi)]$$

$$post_e([\bigcup_{\varphi\in\Phi}\varphi]) = \bigcup_{\varphi\in\Phi} post_e([\varphi]) = [\bigcup_{\varphi\in\Phi} post_{|e}(\varphi)]$$

Also

$$post_e([\bigcup_{i\geq 0}\Phi_i]) = [\bigcup_{i\geq 0}\Phi_i]$$

Now we prove the following cases.

Case I. Widening rule I is used. If $e \equiv event(\gamma,\varphi)$ and φ and γ are as defined in the conditions of the widening rule, we show that

$$[\bigvee_{q\geq 0}\eta \wedge x_i - x_j \geq c_{ij} - q * c_i] \subseteq [\bigcup_{i\geq 0}\Phi_i]$$

The proof is by induction on q. The base case follows from the assumption. To prove the induction step observe that:

$$[post_{|e}(\bigvee_{q\geq 0}\eta \wedge x_i - x_j \geq c_{ij} - q * c_i)] \equiv post_e([\bigvee_{q\geq 0}\eta \wedge x_i - x_j \geq c_{ij} - q * c_i]) \subseteq post_e([\bigcup_{i\geq 0}\Phi_i]) \equiv [\bigcup_{i\geq 0}\Phi_i]$$

Induction Step:

$$\begin{array}{ll} & post_{|e}(\eta \wedge x_i - x_j \geq c_{ij} - q * c_i) \\ \equiv & \exists_{-\mathbf{x}'}\eta \wedge x_i - x_j \geq c_{ij} - q * c_i \wedge \theta \wedge x_j' = x_j + x_i' \wedge x_i \leq c_i \\ \equiv & \exists_{-\mathbf{x}'}(\eta \wedge \varphi \wedge \psi) \wedge (x_i - x_j \geq c_{ij} - q * c_j \wedge \theta) \wedge x_i - x_j \geq c_{ij} - q * c_i \wedge x_j' = x_j + x_i' \wedge x_i \leq c_i \\ \equiv & (\exists_{-\mathbf{x}'}\varphi \wedge \psi \wedge \eta) \wedge (\exists_{-\mathbf{x}'} x_i - x_j \geq c_{ij} - q * c_i \wedge \theta) \wedge x_i' - x_j' \geq c_{ij} - (q+1) * c_i \end{array}$$

Since $\eta[\mathbf{x}'] \models (\exists_{\mathbf{x}'}\varphi \wedge \psi \wedge \gamma) \wedge (\exists_{-\mathbf{x}'} x_i - x_j \geq c_{ij} - q * c_i \wedge \theta)$ therefore $[\eta \wedge x_i' - x_j' \geq c_{ij} - (q+1) * c_i] \subseteq [\bigcup_{i\geq 0} \Phi_i]$. Therefore $[\bigvee_{q\geq 0} \eta \wedge x_i - x_j \geq c_{ij} - q * c_i] \subseteq [\bigcup_{i\geq 0} \Phi_i]$. But

$$\bigvee_{q\geq 0} \eta \wedge x_i - x_j \geq c_{ij} - q * c_i = \eta \wedge \bigvee_{q\geq 0} x_i - x_j \geq c_{ij} - q * c_i = \eta \wedge true = \eta.$$

So $[\eta] \subseteq [\bigcup_{i\geq 0} \Phi_i]$. The case where the *or if* condition holds can be proven similarly.

Case II. Widening rule II is used. We now show by induction on q that

$$[\bigvee_{q\geq 0} \eta \wedge x_i - x_j \geq c_{ij} - q * a_{ji} \wedge x_j - x_i \geq c_{ji}] \subseteq [\bigcup_{i\geq 0} \Phi_i]$$

The base case follows by assumption.
Induction Step:
Observe that

$$\begin{array}{ll} & post_{|e}(\eta \wedge x_i - x_j \geq c_{ij} - q * a_{ji} \wedge x_j - x_i \geq c_{ji}) \\ \equiv & \exists_{-\mathbf{x}'} \eta \wedge x_i - x_j \geq c_{ij} - q * a_{ji} \wedge x_j - x_i \geq c_{ji} \wedge x_j' - x_i' \leq x_j - x_i + a_{ji} \wedge \theta \\ \equiv & (\exists_{-\mathbf{x}'} \varphi \wedge \psi \wedge \eta) \\ \wedge & (\exists_{-\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij} - q * a_{ji} \wedge x_j - x_i \geq c_{ji}) \\ \wedge & (\exists_{-\mathbf{x}'} x_i - x_j \geq c_{ij} \wedge x_j' - x_i' \leq x_j - x_i + a_{ji} \wedge x_i - x_j \geq c_{ij} - q * a_{ji}) \\ \equiv & (\exists_{-\mathbf{x}'} \eta \wedge \varphi \wedge \psi) \wedge (\exists_{-\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij} - q * a_{ji} \wedge x_j - x_i \geq c_{ji}) \wedge x_i' - x_j' \geq c_{ji} - (q+1) * a_{ji} \end{array}$$

Now we show that

$$\begin{array}{ll} \mathcal{R} \models & (\exists_{-\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij} \wedge x_j - x_i \geq c_{ji}) \\ \iff & (\exists_{\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij} - q * a_{ji} \wedge x_j - x_i \geq c_{ji}) \wedge (\exists_{-\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij}) \end{array}$$

Indeed

$$\begin{array}{ll} \mathcal{R} \models & (\exists_{-\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij} \wedge x_j - x_i \geq c_{ji}) \\ \iff & (\exists_{-\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij} - q * a_{ji} \wedge x_i - x_j \geq c_{ij} \wedge x_j - x_i \geq c_{ji}) \text{ (Since } a_{ji} > 0) \\ \iff & (\exists_{\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij} - q * a_{ji} \wedge x_j - x_i \geq c_{ji}) \wedge (\exists_{-\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij}) \end{array}$$

Now it is easy to see that we can write $(\exists_{-\mathbf{x}'} \varphi \wedge \psi \wedge \eta) \wedge (\exists_{-\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ji} - q * a_{ji} \wedge x_j - x_i \geq c_{ji})$ as $\zeta' \wedge x_j' - x_i' \geq c_{ji}''$ where $\eta[\mathbf{x}'] \models \zeta'$ and $c_{ji}'' \leq c_{ji}$. Now we prove that

$$\begin{array}{ll} & \eta[\mathbf{x}'] \wedge x_i' - x_j' \geq c_{ij} - (q+1) * a_{ji} \wedge x_j' - x_i' \geq c_{ji} \\ \models & (\eta[\mathbf{x}'] \wedge x_j' - x_i' \geq c_{ji} \wedge x_i' - x_j' \geq c_{ij} - q * a_{ji}) \\ \vee & ((\exists_{-\mathbf{x}'} \eta \wedge \varphi \wedge \psi) \wedge (\exists_{-\mathbf{x}'} x_i - x_j \geq c_{ij} - q * a_{ji} \wedge x_j' - x_i' \geq c_{ji} \wedge \theta)) \end{array} \quad (6.1)$$

Indeed, suppose $\mathcal{R}, \mathbf{v} \models$ left-hand-side. Then $\mathcal{R}, \mathbf{v} \models \eta[\mathbf{x}']$. Also $\mathcal{R}, \mathbf{v} \models x_j' - x_i' \geq c_{ji}$. Suppose $\mathbf{v}$ does not satisfy the first disjunct on the right hand side. Then $\mathbf{v} \not\models x_i' - x_j' \geq c_{ij} - q * a_{ji}$. Therefore $\mathcal{R}, \mathbf{v} \models x_j' - x_i' > q * a_{ji} - c_{ij}$. Therefore $\mathcal{R}, \mathbf{v} \models x_j' - x_i' \geq q * a_{ji} - c_{ij}$. Now

$$(\exists_{-\mathbf{x}'} \varphi \wedge \psi \wedge \eta) \wedge (\exists_{-\mathbf{x}'} \theta \wedge x_i - x_j \geq c_{ij} - q * a_{ji} \wedge x_j - x_i \geq c_{ji}) \equiv \zeta' \wedge x_j' - x_i' \geq c_{ji}''.$$

Then $\mathcal{R}, \mathbf{v} \models x'_j - x'_i \geq c''_{ji}$ since $c''_{ji} \leq c_{ji}$. Since $\mathcal{R}, \mathbf{v} \models \eta[\mathbf{x}']$, therefore $\mathbf{v} \models \zeta'$. Also $\mathcal{R}, \mathbf{v} \models x'_i - x'_j \geq c_{ij} - q * a_{ji}$. Since the (solution of) right hand side of (6.1) is included in $[\bigcup_{i\geq 0} \Phi_i]$ hence $[\eta \wedge x_j - x_i \geq c_{ji} \wedge x_i - x_j \geq c_{ij} - (q+1) * a_{ji}] \subseteq [\bigcup_{i\geq 0} \Phi_i]$. Hence by a reasoning similar to that in the previous case, we get, $[\eta \wedge x_j - x_i \geq c_{ji}] \subseteq [\bigcup_{i\geq 0} \Phi_i]$. The proof for the *else if* part is easy.

Case III. Widening rule III is used. We show by induction on q that

$$[\bigvee_{q\geq 0} \eta \wedge x_i - x_j \geq q * c_j + c_{ij} \wedge x_j - x_i \geq -q * c_j + c_{ji}] \subseteq [\bigcup_{i\geq 0} \Phi_i].$$

The base case follows from the assumption.
Induction Step:

$$\begin{array}{ll} & post_{|e}(\eta \wedge x_i - x_j \geq q * c_j + c_{ij} \wedge x_j - x_i \geq -q * c_j + c_{ji}) \\ \equiv & \exists_{-\mathbf{x}'} \eta \wedge x_i - x_j \geq q * c_j + c_{ij} \wedge x_j - x_i \geq -q * c_j + c_{ji} \wedge \varphi \wedge \psi \\ \equiv & (\exists_{-\mathbf{x}'} \eta \wedge \varphi \wedge \psi) \wedge (\exists_{-\mathbf{x}'} x_i - x_j \geq q * c_j + c_{ij} \wedge x_j - x_i \geq c_{ji} - q * c_j \wedge \theta) \\ \wedge & (\exists_{-\mathbf{x}'} x_i - x_j \geq c_{ij} + q * c_j \wedge x_j - x_i \geq -q * c_j + ac_{ji} \wedge x'_i = x_i + x'_j \wedge x_j = c_j) \\ \equiv & (\exists_{-\mathbf{x}} \eta \wedge \varphi \wedge \psi) \wedge (\exists_{-\mathbf{x}'} x_i - x_j \geq c_j * q + c_{ij} \wedge x_j - x_i \geq -c_j * q + c_{ji} \wedge \theta) \\ \wedge & x'_i - x'_j \geq c_{ij} + (q+1) * c_j \wedge x'_j - x'_i \geq -(q+1) * c_j + c_{ji} \end{array}$$

Hence, by reasons similar to the above cases, $[\eta \wedge x_i - x_j \geq c_{ij} + (q+1) * c_j \wedge x_j - x_i \geq c_{ji} - (q+1) * c_j] \subseteq [\bigcup_{i\geq 0} \Phi_i]$. The rest of the proof is similar to that of the previous cases. ‖

Note that the above theorem also implies that if the procedure Symbolic-Boundedness-W terminates, then one can get a full test of safety properties as well. Below we provide effective sufficient conditions for termination of Symbolic-Boundedness-W. By a simple path in a timed automaton $\mathcal{U}$, we mean a sequence of events $e_1 \ldots e_m$ where each e_i is an original event of $\mathcal{U}$ and

- the source location of e_{i+1} is the same as the target location of e_i for $1 \leq i \leq m-1$,
- any event e_i with same source and target locations is a time event,
- for any two edge events e_i and e_j, $1 \leq i < j < m$, the target locations of e_i and e_j are different,
- and if e_i is an (original) time event, then e_{i-1} and e_{i+1} are edge events.

With this definition, there are only a finite number of such simple paths in a timed automaton. The simple path $p = e_1 \ldots e_m$ leads from location ℓ^1 to the location ℓ^2 if there is a the source location of e_1 is ℓ^1 and the target location of e_m is ℓ^2. The simple path $e_1 \ldots e_m$ is a simple cycle if the source location of e_1 is the same as the target location of e_m. Note that there are only a finite number of such simple cycles in a timed automaton.

Theorem 6.2 (Sufficient Conditions for Termination) *Let $\mathcal{U}$ be a timed automaton and let ℓ be a location in $\mathcal{U}$ such that there is a simple cycle C from ℓ to itself and the following three conditions are satisfied.*

- *There is a simple path in $\mathcal{U}$ of the form $e \equiv$ **cond** φ **action** ψ leading from the initial location ℓ^0 to ℓ such with the cycle C along with the the constraint $(\exists_{-\mathbf{x}'} \varphi^0 \wedge \varphi \wedge \psi)[\mathbf{x}]$ that satisfies the conditions of the widening rules I, II or III where φ^0 is the initial constraint.*

- *For each original event* $e' \equiv$ **cond** φ' **action** ψ' *with target location* ℓ *that lies on a cycle in the control graph of* $\mathcal{U}$, $(\exists_{-\mathbf{x}'} \varphi' \wedge \psi')[\mathbf{x}] \models post_{|t}(\eta)$ *if widening rule I or II is satisfied in the previous condition and* $(\exists_{-\mathbf{x}'} \varphi' \wedge \psi')[\mathbf{x}] \models post_{|t}(\eta \wedge \exists k \geq 0 \wedge int(k) \wedge x_i - x_j \geq c_{ji} + k * c_j \wedge x_j - x_i \geq c_{ji} - k * c_j)$ *if widening rule III is satisfied in the previous condition, where* η, c_{ji} *are as in the definition of the widening rules and* t *is the time event at location* ℓ.

- *The control graph of* $\mathcal{U}$ *satisfies the temporal formula* $AG(true \Longrightarrow AF(at\_\ell))$ *where* at_ℓ *is an atomic proposition satisfied only by location* ℓ.

Then the procedure Symbolic-Boundedness-W terminates for $\mathcal{U}$.

Proof. The proof follows from the observation that along the breadth-first tree generated by the procedure Symbolic-Boundedness-W there is a branch that starts from the initial location ℓ^0 follows the simple path e and then follows the cycle C. Since, the constraint γ at the end of this simple path along with the guard and action of C satisfies one of the three widening rules (by the first condition), we get the constraint η (if the widening rules I or II are satisfied) or $\eta \wedge \exists k \geq 0 \wedge int(k) \wedge x_i - x_j \geq c_{ji} + k * c_j \wedge x_j - x_i \geq c_{ji} - k * c_j$ (if widening rule III is satisfied). (Here η and the other constraints are as in the definition of the widening rules.) Hence, there exists a finite i such that Φ_i contains η (or $\eta \wedge \exists k \geq 0 \wedge int(k) \wedge x_i - x_j \geq c_{ji} + k * c_j \wedge x_j - x_i \geq c_{ji} - k * c_j$). Now since the control graph of $\mathcal{U}$ satisfies $AG(true \Longrightarrow AF(at\_\ell))$, therefore along any branch on the breadth-first tree, the location ℓ will be reached in some iteration greater than i through an original event e' that lies in a cycle. Suppose the constraint generated at this point be $\langle \ell(\mathbf{x}), \chi \rangle$. We show that $\chi \models \eta$ (or $\chi \models \eta \wedge \exists k \geq 0 \wedge int(k) \wedge x_i - x_j \geq c_{ji} + k * c_j \wedge x_j - x_i \geq c_{ji} - k * c_j$ if widening rule III was satisfied in the first condition of the theorem. We prove this in the case of widening rule I; the remaining cases are similar. Suppose that $\mathbf{v} \models \chi$. Suppose $\langle \ell(\mathbf{x}), \chi \rangle$ was generated by the original event $e' \equiv$ **cond** φ' **action** ψ'. Then $\mathcal{R}, \mathbf{v} \models (\exists_{-\mathbf{x}'} \varphi' \wedge \psi')[\mathbf{x}]$. Then $\mathcal{R}, \mathbf{v} \models \eta$. Hence, each branch along the breadth-first tree is finite. Therefore the procedure Symbolic-Boundedness-W terminates. ∎

It can be seen that the example in Figure 6.10 satisfies the sufficient conditions stated above.

We have implemented a prototype based on the approach (in the CLP($\mathcal{R}$) system of Sicstus Prolog 3.7). The performance shown, so far, by our approach has been quite encouraging. We have used our implementation to verify the safety and boundedness properties of several well-known benchmark examples taken from literature. The experimental results are summarized in the table in Figure 6.14. All results are obtained on a PC (200 MHz Pentium Pro). The experiments show a marked improvement over the timings obtained without using the accurate widening rules in Chapter 3. The timings obtained for Fischer's protocol (two processes), Rail-Road Crossing, and Audio Protocol without using the widening rules are $4.2s$, $1.8s$ and $7.2s$ respectively. All the timings in Figure 6.14 denote the total time taken for reachability analysis.

6.5 Related Work

In this chapter, we have presented a constraint based framework for symbolic model checking of timed systems against boundedness properties. We have shown that it is possible to achieve (or just accelerate) termination of our symbolic model checking procedure with abstractions by widening that are, as we prove, accurate. Our approach allows us to do a full test of the safety and boundedness (unboundedness) properties without going into the complications of

Example	*time (seconds)*
Fischer's Protocol (Two Processes) [LPY95b]	2.1
Rail-road Crossing	0.8
Audio Protocol [HWT95]	2.3

Figure 6.14: Experimental Results

region construction. Regarding the generality of our approach, we do not claim that the three widening rules described in this chapter encompass (i.e., achieves termination and/or speed-up of model checking procedure) the full class of timed automata. We have provided sufficient conditions under which the procedure Symbolic-Unboundedness-W is guaranteed to terminate. However, for several examples the procedure terminates even though the sufficient conditions do not apply..

Note that there has been a few attempts at verification of timed and hybrid systems based on constraint logic programming [GP97, Fri98, CDD+98, DRS99, GP99]. Our work differs from these approaches in that we exploit the constraint-based setting for defining acceleration techniques based on abstract interpretation. Note that the model checking procedure for Uppaal [BLL+96] is also based on semantics of constraints but their algorithms are based on graph-theoretic techniques rather than techniques from constraint programming. We believe that incorporation of accurate widening framework in UPPAAL and the other approaches mentioned above can significantly speed-up model checking procedures based on those approaches.

Our widening operator is closely related to Boigelot and Wolper's loop-first technique [BW94] for deriving periodic sets as representations of infinite sets of integer valued states for reachability analysis. As a difference, Boigelot and Wolper analyze cycles and nested cycles in the control graph to detect meta-transitions *before* and independently of their forward model checking procedure, whereas we construct new events *during* our model checking procedure and consider them only if we detect that they possibly lead to an infinite loop. Berard' and Fribourg [BF99] use a constraint-based framework for reachability analysis for timed Petri nets. They have been able to verify several interesting examples using their approach based on meta transitions. Our approach, rooted in the abstract interpretation framework, is different from theirs in that we accelerate the model checking procedure using widening rules based on syntax.

The application of widening techniques to the verification of systems with huge or infinite state spaces has proven useful in several examples. Halbwachs et.al. [HPR97], using linear relational analysis to prove properties for *linear hybrid systems*, defines a widening operator over convex polyhedra: unions of convex polyhedra are approximated by their convex hull before the widening step. Approximation techniques for more general classes of hybrid systems are studied in [HHWT97, HH95]. Specifically, Henzinger and Ho [HH95] apply an extrapolation operator which gives better approximations than Halbwachs et. als' convex widening operator in their examples. For integer valued systems, abstract interpretation has been used effectively in [BGP97]. In [BGP98], it was explicitly mentioned that one main difficulty with the approximate approach is that the abstraction is often too rough. We have shown in Section 6.4 that our widening techniques will give full test of reachability properties for timed systems where the approximate methods [Bal96, WT95, HPR97] would produce a 'don't know' answer. Also, in contrast with our accurate widenings, the widening techniques proposed in [Bal96, WT95, HPR97] cannot be used for model checking for boundedness properties. Note that it is not possible to find out

in most cases, using semantics-based techniques, whether a program loop really generates an infinite behavior with respect to reachability analysis. Hence, application of widening combined with semantics-based techniques may result in loss of accuracy that will render these techniques unsuitable for model checking for boundedness properties. It would be interesting to look at how the techniques described in this chapter extend to more general classes of hybrid systems. The general goal will be a whole library of accurate widening rules for a variety of verification problems.

Chapter 7

Compositional Termination Analysis of Symbolic Forward Analysis for Infinite-State Systems

7.1 Introduction

Over the last few years, there has been an increasing research effort directed towards automatic verification of infinite state systems. Research on decidability issues (e.g., [ACJT96, ACHH93, Boi98, LPY99, HKPV95, CJ98]) has resulted in highly nontrivial algorithms for the verification of different subclasses of infinite state systems. These results do not, of course, imply the termination of the semi-algorithms on which practical tools are based (for example, the decidability of the model checking problem for timed automata does not entail termination for the symbolic forward analysis of timed automata which is possibly non-terminating). This chapter addresses the termination for such a procedure, namely symbolic forward analysis; we show termination for the subclass of *o-minimal* hybrid systems (for which backward analysis is known to be terminating [LPY99]), and we give *compositional* syntactic sufficient conditions for integer-valued systems and for nonlinear hybrid systems; i.e., the syntactic sufficient conditions are on the individual components rather than on the composed system. The conditions roughly express that, in each loop, the variables are initialized before they are used.

Sufficient termination conditions for symbolic forward analysis seem interesting for several reasons. First, since they apply to concrete examples such as practical mutual exclusion protocols, they may shed a new light on the practical success of symbolic model checking for infinite-state systems (see e.g. [BGP97, DP99a, DT98, LPY95b]). Second, for a concrete verification problem in a practical setting, the model to be checked can possibly be adapted to meet the sufficient termination conditions (e.g. by adding semantically redundant initializations of variables).

Moreover, our results suggest a potential optimization of the symbolic forward analysis procedure. Namely, the termination guarantee continues to hold even when the fixpoint test is made more efficient by weakening it to *local entailment* (explained below; e.g. for linear arithmetic constraints over reals, the complexity of fixpoint test reduces from co-NP hard to polynomial).

7.2 Infinite State Systems

We use guarded-command programs to specify (possibly infinite-state) transition systems. A guarded-command program consists of a set $\mathcal{E}$ of guarded commands e (called edges) of the form

$$e \;\equiv\; L = \ell \wedge \gamma_e(\mathbf{x}) \;[\!]\; L' = \ell' \wedge \alpha_e(\mathbf{x}, \mathbf{x}')$$

where L is a variable ranging over a finite set of program locations, $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ is the tuple of program variables (ranging over a possibly infinite data domain); $\gamma_e(\mathbf{x})$ is a formula (the guard) whose free variables are among $\mathbf{x}$; $\alpha_e(\mathbf{x}, \mathbf{x}')$ is a formula (the action) whose free variables are among $\mathbf{x}, \mathbf{x}'$ of e. Intuitively, the primed version of a variable stands for its value in the successor state after taking a transition through a guarded command. We translate a guarded command e to the logical formula ψ_e simply by by replacing the guard $[\!]$ with conjunction.

$$\psi_e \equiv L = \ell \wedge \gamma_e(\mathbf{x}) \wedge L' = \ell' \wedge \alpha_e(\mathbf{x}, \mathbf{x}')$$

A state of the system is a pair $\langle \ell, \mathbf{v} \rangle$ consisting of the values for the location variable and for each program variable. The state $\langle \ell, \mathbf{v} \rangle$ can make a transition to the state $\langle \ell', \mathbf{v}' \rangle$ through the edge e provided that the values of ℓ for L, ℓ' for L', $\mathbf{v}$ for $\mathbf{x}$ and $\mathbf{v}'$ for $\mathbf{x}'$ define a solution for ψ_e. A run of the system is a sequence $\langle \ell^1, \mathbf{v^1} \rangle \longrightarrow \langle \ell^2, \mathbf{v^2} \rangle \longrightarrow \ldots$ such that for each $i = 1, 2, \ldots$ there exists an edge e such that the state $\langle \ell^i, \mathbf{v^i} \rangle$ can make a transition to the state $\langle \ell^{i+1}, \mathbf{v^{i+1}} \rangle$ through the edge e.

In this chapter, we consider two basic classes of infinite state systems. In the first, the program variables range over the set of natural numbers $\mathcal{N}$, and the guard and the action formulas are arithmetic constraints. Examples of such systems above include the bakery algorithm, the bounded buffer producer-consumer problem etc.. In the second, we deal with the so-called hybrid systems in which the program variables range over the set of reals $\mathcal{R}$.

Systems with Integer-valued Variables. We write $Arith(\mathcal{N})$ for the theory of natural numbers with addition multiplication and order; it is interpreted over the structure $\langle \mathcal{N}, <, +, \cdot, 0, 1 \rangle$. A possibly nonlinear system with integer-valued variables can be defined as a set of guarded commands as above where the variables $\mathbf{x}, \mathbf{x}'$ are interpreted over the set of natural numbers $\mathcal{N}$. The guard formula $\gamma_e(\mathbf{x})$ is an $Arith(\mathcal{N})$ formula with free variables among $\mathbf{x}$. The action formula $\alpha_e(\mathbf{x}, \mathbf{x}')$ of e, with free variables among $\{\mathbf{x}, \mathbf{x}'\}$, is also an $Arith(\mathcal{N})$ formula.

Hybrid Systems We write $OF(\mathcal{R})$ for the theory of the ordered field of reals; it is interpreted over the structure $\langle \mathcal{R}, <, +, \cdot, 0, 1 \rangle$.

A (possibly non-linear) hybrid system can be defined as a set of guarded commands as above where the guard $\gamma_e(\mathbf{x})$ is an $OF(\mathcal{R})$ formula, and the action $\alpha_e(\mathbf{x}, \mathbf{x}')$ is an $OF(\mathcal{R})$ formula given by

$$\alpha_e(\mathbf{x}, \mathbf{x}') \;\equiv\; \exists z \geq 0 \; \exists \mathbf{x}'' \; \delta_e(\mathbf{x}, \mathbf{x}'') \wedge \beta_e(\mathbf{x}'', \mathbf{x}', z).$$

Here, δ_e is an $OF(\mathcal{R})$ formula defining the "update" in e, and β_e is the $OF(\mathcal{R})$ formula defining the continuous evolution at the target location ℓ'.

A transition according to a guarded command e represents an instantaneous 'jump' followed by a continuous evolution over time at the target location ℓ'. Namely, a state $\langle \ell, \mathbf{v} \rangle$ can make

a transition through e to the state $\langle \ell', \mathbf{v}' \rangle$ if the values ℓ for the location variable L and $\mathbf{v}$ for the tuple of data variables $\mathbf{x}$ satisfy the guard $L = \ell \wedge \gamma_e(\mathbf{x})$ of e and there exists a $\mathbf{v}''$ such that $\mathbf{v}, \mathbf{v}''$ satisfies the update δ_e of e and there exists a real value d of the delay variable z such that $\mathbf{v}'$ is obtained from $\mathbf{v}''$ through continuous evolution over the delay d at the location ℓ'. Similarly the *time transition* (continuous evolution over time at a location) from the state $\langle \ell, \mathbf{v} \rangle$ to the state $\langle \ell, \mathbf{v}' \rangle$ can be defined. For a location ℓ, let β_ℓ be the $(OF(\mathcal{R}))$ formula denoting the continuous evolution of time at ℓ where z is the time delay variable. Thus in the above formula defining the guard $\alpha(\mathbf{x}, \mathbf{x}')$, $\beta_e = \beta_{\ell'}$.

O-minimal hybrid systems In this paragraph, we define o-minimal hybrid systems. The definition below is adapted from [LPY99]. In o-minimal hybrid systems, the action formula $\alpha_e(\mathbf{x}, \mathbf{x}')$ of e with free variables among $\{\mathbf{x}, \mathbf{x}'\}$ is defined as follows.

$$\alpha_e(\mathbf{x}, \mathbf{x}') \equiv \exists z \geq 0 \exists \mathbf{x}''(\delta_e(\mathbf{x}'') \wedge \mathbf{x}' = exp^{zA_{\ell'}}\mathbf{x}'')$$

where the free variables in the "update" formula δ_e are among $\mathbf{x}''$, exp is the base of the natural logarithms, $A_{\ell'}$ is an $n \times n$ rational matrix that is either nilpotent or is diagonalizable with rational eigenvalues ($\mathbf{x}' = exp^{zA_{\ell'}}\mathbf{x}''$ represents the continuous evolution at the target location ℓ'). It can be shown [LPY99] that in these cases, $\alpha_e(\mathbf{x}, \mathbf{x}')$ is definable in $OF(\mathcal{R})$.

7.3 Parallel Composition

We consider asynchronous parallel composition of infinite state systems. We assume that the component programs do not share variables (except for the synchronizing labels). For the purpose of parallel composition, we assign to each guarded command a synchronizing label. Thus with each guarded command program $\mathcal{S}$ we associate a (finite) set Σ of synchronizing labels and a mapping $lab : \mathcal{E} \longrightarrow \Sigma$ that assigns to each guarded command (or edge) a synchronizing label from Σ.

Given two guarded command programs $\mathcal{S}_1$ and $\mathcal{S}_2$ with label sets Σ_1 and Σ_2 and labeling functions lab_1 and lab_2 respectively, their parallel composition $\mathcal{S} = \mathcal{S}_1||\mathcal{S}_2$ with set of synchronizing labels $\Sigma_1 \cup \Sigma_2$ and labeling function lab is defined as follows. Intuitively, in an edge in the composed program $\mathcal{S}_1||\mathcal{S}_2$, either only $\mathcal{S}_1$ "moves" (i.e., takes a transition through an edge) while $\mathcal{S}_2$ undergoes continuous evolution at the same location (if the synchronizing label σ is in Σ_1 but not in Σ_2) or $\mathcal{S}_2$ "moves" while $\mathcal{S}_1$ undergoes continuous evolution (in case of hybrid systems; stays unchanged in case of integer-valued systems) at the same location (provided the synchronizing label σ is in Σ_2 but not in Σ_1) or both "move" (if the synchronizing label $\sigma \in \Sigma_1 \cap \Sigma_2$) with the same label $lab(e_1) = lab(e_2) = \sigma$. The composed program $\mathcal{S}$ consists of all guarded commands of the form

$$e \equiv L_1 = \ell^1 \wedge L_2 = \ell^2 \wedge \gamma_e(\mathbf{x}, \mathbf{y}) \| L_1' = \ell^{1\prime} \wedge L_2' = \ell^{2\prime} \wedge \alpha_e(\mathbf{x}, \mathbf{y}, \mathbf{x}', \mathbf{y}')$$

with $lab(e) = \sigma$ such that either

- (First component "moves")
  - there is an edge $e_1 \equiv L = \ell^1 \wedge \gamma_{e_1}(\mathbf{x}) \| L' = \ell^{1\prime} \wedge \alpha_{e_1}(\mathbf{x}, \mathbf{x}')$ in $\mathcal{S}_1$ with $lab_1(e_1) = \sigma$, where $\sigma \in \Sigma_1$ and $\sigma \notin \Sigma_2$, ℓ^2 is a location in $\mathcal{S}_2$ and $\ell^{2\prime} = \ell^2$

- $\gamma_e(\mathbf{x},\mathbf{y}) \equiv \gamma_{e_1}(\mathbf{x})$
- $\alpha_e(\mathbf{x},\mathbf{y},\mathbf{x}',\mathbf{y}') \equiv \alpha_{e_1}(\mathbf{x},\mathbf{x}') \wedge \mathbf{y}' = \mathbf{y}$ for systems with integer-valued variables and $\alpha_e(\mathbf{x},\mathbf{y},\mathbf{x}',\mathbf{y}') \equiv \exists z \geq 0\ \exists t \geq 0\ \varphi_{e_1}(\mathbf{x},\mathbf{x}',z) \wedge \beta_{\ell^2}(\mathbf{y},\mathbf{y}',t) \wedge z = t$ for hybrid systems where $\alpha_{e_1}(\mathbf{x},\mathbf{x}') \equiv \exists z \geq 0\ \varphi_{e_1}(\mathbf{x},\mathbf{x}',z)$.

- Or (Second component "moves") Same as the previous point but with the roles of $\mathcal{S}_1$ and $\mathcal{S}_2$ reversed.

- Or

  (Both components "move")there is an edge $e_1 \equiv L = \ell^1 \wedge \gamma_{e_1}(\mathbf{x}) [\![L' = \ell^{1\prime} \wedge \alpha_{e_1}(\mathbf{x},\mathbf{x}')$ in $\mathcal{S}_1$ and an edge $e_2 \equiv L = \ell^2 \wedge \gamma_{e_2}(\mathbf{y}) [\![L' = \ell^{2\prime} \wedge \alpha_{e_2}(\mathbf{y},\mathbf{y}')$ in $\mathcal{S}_2$ such that $\sigma \in \Sigma_1 \cap \Sigma_2$, $lab_1(e_1) = \sigma$ and $lab_2(e_2) = \sigma$ $\gamma_e(\mathbf{x},\mathbf{y}) \equiv \gamma_{e_1}(\mathbf{x}) \wedge \gamma_{e_2}(\mathbf{y})$ $\alpha_e(\mathbf{x},\mathbf{y},\mathbf{x}',\mathbf{y}') \equiv \alpha_{e_1}(\mathbf{x},\mathbf{x}') \wedge \alpha_{e_2}(\mathbf{y},\mathbf{y}')$ for systems with integer-valued variables and $\alpha_e(\mathbf{x},\mathbf{y},\mathbf{x}',\mathbf{y}') \equiv \exists z \geq 0\ \exists t \geq 0\ \varphi_{e_1}(\mathbf{x},\mathbf{x}',z) \wedge \varphi_{e_2}(\mathbf{y},\mathbf{y}',t) \wedge z = t$ for hybrid systems where $\alpha_{e_1}(\mathbf{x},\mathbf{x}') \equiv \exists z \geq 0\ \varphi_{e_1}(\mathbf{x},\mathbf{x}',z)$ and $\alpha_{e_2}(\mathbf{y},\mathbf{y}') \equiv \exists t \geq 0\ \varphi_{e_2}(\mathbf{y},\mathbf{y}',t)$.

A state of the composed program is a tuple $\langle \ell, \ell', \mathbf{v}, \mathbf{w} \rangle$ consisting of values of the locations and each variable. The semantics of the composed program is defined in the usual way. The parallel composition operation defined above is commutative and associative. For guarded command programs $\mathcal{S}_1, \ldots, \mathcal{S}_k$, we write $\mathcal{S}_1 || \ldots || \mathcal{S}_k$ to denote $(\ldots(\mathcal{S}_1||\mathcal{S}_2)||\mathcal{S}_3)|| \ldots)||\mathcal{S}_k)$. Tools like UPPAAL [BLL+96], HYTECH [HHWT95] use the kind of parallel composition described above (they also use urgent transitions; the framework described below can be easily made to take into account such urgent transitions).

7.4 Constraints Representing Sets of States

In this chapter, by constraints we will mean $Arith(\mathcal{N})$ or $OF(\mathcal{R})$ formulas. We use constraints φ to represent certain sets of positions. We will consider only conjunctive constraints. A constraint φ is a conjunction of atomic constraints of the form $t\, relop\, c$ where t is a term, $c \in \mathcal{N}$ and $relop \in \{>, <, \geq, \leq\}$. We identify solutions of the constraints with states of the system. We write $\mathcal{D}, \langle \ell, \mathbf{v} \rangle \models \varphi$ to denote that the state $\langle \ell, \mathbf{v} \rangle$ is a solution of the constraint φ where $\mathcal{D}$ is the structure under consideration, i.e., either $\langle \mathcal{N}, <, +, \cdot, 0, 1 \rangle$ or $\langle \mathcal{R}, <, +, \cdot, 0, 1 \rangle$. For a constraint φ, we define the denotation of φ, denoted by $[\varphi]$ as

$$[\varphi] = \{\langle \ell, \mathbf{v} \rangle \mid \mathcal{D}, \langle \ell, \mathbf{v} \rangle \models \varphi\}.$$

By a set of constraints we mean their disjunction; i.e., if Φ is a set of constraints then $[\Phi] = \bigcup_{\varphi \in \Phi} [\varphi]$. For two constraints φ and φ', we say that φ entails φ', denoted by $\varphi \models \varphi'$, iff $[\varphi] \subseteq [\varphi']$. We assume that given two constraints φ and φ', it is decidable whether $\varphi \models \varphi'$ (though this is not true for arbitrary $Arith(\mathcal{N})$ constraints, still we assume that for the constraints that we will deal with, the problem of checking whether a constraint entails another is decidable). For a constraint φ with free variables $\mathbf{x}$, we denote by $\varphi(\mathbf{x}')$, the constraint obtained by replacing the free variables $\mathbf{x}$ by $\mathbf{x}'$ (renaming).

A constraint φ is *time closed* if its set of solutions (i.e., its denotation) is closed under time transitions, i.e., if the constraint φ is of the form $L = \ell \wedge \psi$ and if $\beta_\ell(\mathbf{x}'', \mathbf{x}', z)$ is the $OF(\mathcal{R})$ formula representng continuous evolution at the location ℓ (for hybrid systems), then φ is time-closed iff $\mathcal{R} \models \varphi \iff (\exists z \geq 0\ \exists \mathbf{x}\, (\varphi \wedge \beta_\ell(\mathbf{x}, \mathbf{x}', z)))(\mathbf{x})$. We denote by φ^0 the formula defining

the time closure of the set of initial states and call it the initial constraint. In the following, whenever we talk of constraints in the context of hybrid systems, we will refer to time-closed constraints. In case of o-minimal hybrid systems we also assume that the initial constraint φ^0 is definable in $OF(\mathcal{R})$.

We identify two constraints φ and φ' iff they have the same denotations; i.e., $[\varphi] = [\varphi']$. We recall the definition of constraint transformer from Chapter5. This notion is inspired by the notion of syntactic transformation monoids in classical automata theory [Eil76].

7.5 Bound Variables and Initialized Strings

We consider bindings of variables in infinite state systems. Roughly, a (data) variable is bound at a location of an infinite state system if its value at that location does not "depend" on its previous values. Let $w = e_1 \dots e_m$ be a string of edges of an infinite state system with integer-valued variables.

- **Definition 7.1 (Bound Variables)** *We say that a subset $X \subseteq \{x_1, \dots, x_n\}$ of variables is bound at the edge e_i $(1 \leq i \leq m)$ in the string w if there exists $S \subseteq \{x_1, \dots, x_n\}$ such that (1) the action $\alpha_{e_i}(\mathbf{x}, \mathbf{x}')$ can be written as a quantifier free formula $\theta_1 \wedge \theta_2$ where the variables in θ_1 are among $X' \cup S$ (where $X' = \{x' \mid x \in X\}$) and the variables in $X' \cup S$ do not occur in θ_2, (2) if $i = 1$ then $S = \emptyset$ and (3) if $i > 1$ then*

  - *the variables in S are bound in e_{i-1} (in w) and*
  - *the guard constraint $\gamma_{e_i}(\mathbf{x})$ can be written as a quantifier free formula $\gamma^1_{e_i} \wedge \gamma^2_{e_i}$ where the variables in $\gamma^1_{e_i}$ are bound in e_{i-1} (in w) and the variables in $\gamma^2_{e_i}$ are not bound in e_{i-1} (in w).*

  Thus, consider the guarded commands e_1, e_2 and e_3 given by

$$e_1 \equiv L = \ell \wedge x > y \parallel L' = \ell' \wedge x' = x \wedge y' = y \wedge z' \geq 2,$$

$$e_2 \equiv L = \ell' \wedge z \leq 4 \wedge x < y \parallel L' = \ell'' \wedge z' = z \wedge x' \geq z + 2 \wedge y' \geq y + 4$$

and

$$e_3 \equiv L = \ell'' \wedge x \geq 6 \parallel L' = \ell''' \wedge z' = z \wedge x' = x \wedge y' = x + 2.$$

According to the above definition, in the string $w = e_1.e_2.e_3$, only z is bound in e_1, and $\{x, z\}$ are bound in e_2 and $\{x, y, z\}$ are bound in e_3 in w. This is because, at e_1 the action α_{e_1} can be written as $\theta^1_{e_1} \wedge \theta^2_{e_1}$ where $\theta^2_{e_1} \equiv x' = x \wedge y' = y$ (where z' does not occur) and $\theta^1_{e_1} \equiv z' \geq 2$ (where the free variables are among z'). Hence z is bound at e_1 in w. At the edge e_2, the guard γ_{e_2} can be written as $\gamma^1_{e_2} \wedge \gamma^2_{e_2}$ where $\gamma^1_{e_2} \equiv z \leq 4$ (where the free variable z is bound at e_1 in w) and $\gamma^2_{e_2} \equiv x < y$ (where the free variables are not bound at e_1 in w). Now the action α_{e_2} of e_2 can be written in the form $\theta^1_{e_2} \wedge \theta^2_{e_2}$ where $\theta^1_{e_2} \equiv z' = z \wedge x' \geq z + 2$ (where z is bound at e_1 in w) and $\theta^2_{e_2} \equiv y' \geq y + 4$ (where $\{x', z', z\}$ do not occur free). Hence $\{x, z\}$ are bound at e_2 in w. At the edge e_3, the guard γ_{e_3} can be written in the form $\gamma^1_{e_3} \wedge \gamma^2_{e_3}$ where $\gamma^1_{e_3} \equiv x \geq 6$ (where the free variable x is bound at e_2 in w) and $\gamma^2_{e_3} \equiv true$. Also the action α_{e_3} can be written in the form $\theta^1_{e_3} \wedge \theta^2_{e_3}$ where $\theta^1_{e_3} \equiv z' = z \wedge x' = x \wedge y' = x + 2$ (where $\{x, z\}$ are bound at e_2 in w). Hence $\{x, y, z\}$ are bound at e_3 in w.

We next come to the definition of initialized strings.

Definition 7.2 (Initialized Strings) *A string $w = e_1 \dots e_m$ of edges of an infinite state system is initialized if for each variable x_i, there exists a k ($1 \leq k \leq m$) such that x_i is bound (in w) in every edge in $e_k \dots e_m$.*

For the cases of non-linear hybrid systems (with the underlying theory being the theory of real closed fields) as well as integer valued systems in which the underlying theory is the Presburger arithmetic extended with all relations $x = y(mod\ k)$, $k > 1$, it can be effectively decided using the methods presented in [Lib00] whether a string is initialized.

7.6 Constraint Trees and Symbolic Forward Analysis

Given an infinite state system $\mathcal{S}$ with set of edges $\mathcal{E}$, we define the constraint tree for $\mathcal{S}$ as follows.

Definition 7.3 (Constraint Tree) *The constraint tree for $\mathcal{S}$ is an infinite tree with domain $\mathcal{E}^*$ (i.e., the nodes are strings over $\mathcal{E}$) that labels the node w by the constraint $[\![w]\!](\varphi^0)$ where φ^0 is the initial constraint.*

Clearly, the (infinite) disjunction of all constraints labeling a node of the constraint tree represents all reachable states of $\mathcal{S}$. We now define symbolic forward analysis formally. A symbolic forward analysis is a traversal of (a finite prefix of) a constraint tree in a particular order. The following definition of a non-deterministic procedure abstracts away from that specific order.

Definition 7.4 (Symbolic Forward Analysis) *A symbolic forward analysis of an infinite state system $\mathcal{S}$ is a procedure that enumerates constraints φ_i labeling the nodes w_i of the constraint tree of $\mathcal{S}$ in a tree order such that the disjunction of the enumerated constraints represents all reachable states of $\mathcal{S}$. Formally,*

- *$\varphi_i = [\![w_i]\!](\varphi^0)$ for $0 \leq i < B$ where the bound B is either a natural number or ω,*
- *if w_i is a prefix of w_j then $i \leq j$,*
- *the disjunction $\bigvee_{0 \leq i < B} \varphi_i$ is equivalent to the disjunction $\bigvee_{0 \leq i < \omega} \varphi_i$.*

The number i is a leaf of a symbolic forward analysis if the node w_i is a leaf of the tree formed by all the nodes w_i where $0 \leq i \leq B$. We say that a symbolic forward analysis terminates if its bound B is finite. We define that a symbolic forward analysis terminates with local entailment if for all its leaves i there exists a $j < i$ such that the constraint φ_i entails the constraint φ_j (as a passing remark, we note that by changing the notion of local entailment, we can get a model checking procedure for liveness properties; we can change the notion of local entailment by requiring that for all leaves i, there exists a $j < i$ such that such that the constraint φ_j entails the constraint φ_i). In contrast, a symbolic forward analysis terminates with global entailment if for all its leaves i, the constraint φ_i entails the disjunction of the constraints φ_j where $j < i$. As discussed in the Introduction, model checking is more efficient with local entailment than with global entailment, both theoretically and practically. Many model checking tools for infinite state systems use local entailment (e.g., UPPAAL [BLL+96], which uses identity; the model checker for infinite state systems with integer-valued variables described in [DP99a] also uses local entailment).

We say that a location ℓ is a *part of a cycle* $w = e_1 \dots e_m$ if it is the source of an edge e_i of the cycle; i.e., an edge e_i of the cycle is of the form $\dots L = \ell[\![\dots$.

A string is *initializable* if it contains an initialized substring.

Proposition 7.1 *If every simple cycle of an infinite state system $\mathcal{S}$ is initializable, symbolic forward analysis for the system terminates with local entailment.*

Proof. We first show that the constraint transformer function associated with each initialized string w is either a constant function or unsatisfiable. Let $w = e_1 \dots e_m$ be an initialized string. Now, by definition, for each variable x there exists a j $(1 \leq j \leq m)$ such that x is bound (in w) in every edge in $e_j \dots e_m$. Let j_x be the least such j for x. Let $l = min\{j_x \mid x \in \{x_1, \dots, x_n\}\}$. We now introduce some terminology that will be needed in the rest of the proof. Let $bound(e_i) \subseteq \{x_1, \dots, x_n\}$ be the set of variables that are bound (in the word under consideration)) at an edge e_i. Then, by definition, there is a partition of the set $\{x_1, \dots, x_n\}$ into two subsets S_{e_i} and S'_{e_i} such that for $1 < i \leq m$, the guard γ_{e_i} are can be written as a quantifier free formula of the form $\gamma^1_{e_i} \wedge \gamma^2_{e_i}$ where the free variables in $\gamma^1_{e_i}$ are bound (in w) in e_{i-1} and the free variables in $\gamma^2_{e_i}$ are not bound (in w) in e_{i-1}, the variables in S_{e_i} are bound in e_{i-1} (in w) and the action $\alpha_{e_i}(\mathbf{x}, \mathbf{x}')$ can be written as a quantifier free formula of the form $\theta_1 \wedge \theta_2$ such that the variables occurring free in θ_1 are among $bound(e_i)' \cup S_{e_i}$ (where $bound(e_i)' = \{x' \mid x \in bound(e_i)\}$) and the variables in $bound(e_i)' \cup S_{e_i}$ do not occur free in θ_2. We call S_{e_i} as the past of e_i and write $past(e_i) = S_{e_i}$. Now consider the edge e_l. If $past(e_l) = S_{e_l}$ then the variables in S_{e_l} are bound (in w) in e_{l-1}. If $past(e_{l-1}) = S_{e_{l-1}}$ then the variables in $S_{e_{l-1}}$ are bound (in w) in e_{l-2}. We can continue this reasoning only a finitely many times after which we will get an edge e_p in w such that $past(e_p) = \emptyset$. We will show that the constraint transformer associated with the string $w' = e_p \dots e_m$ is either a constant function or unsatisfiable. The constraint transformer function associated with w' is given by

$$[\![w']\!](\varphi) \equiv (\exists \mathbf{x} \exists \mathbf{x^p} \dots \exists \mathbf{x^{m-1}} (\varphi \wedge \psi_p \wedge \dots \wedge \psi_m))(\mathbf{x})$$

where ψ_k is the formula obtained by applying α-renaming to the conjunction of the guard formula $\gamma_{e_k}(\mathbf{x})$ and the formula $\exists z \geq 0 \exists \mathbf{x''} \delta_{e_k}(\mathbf{x}, \mathbf{x''}) \wedge \wedge \beta_{e_k}(\mathbf{x''}, \mathbf{x'}, z)$. That is

$$\psi_k \equiv \gamma_{e_k}(\mathbf{x^{k-1}}) \wedge \exists z^k \geq 0 \; \exists x^{k\prime\prime} \delta_{e_k}(\mathbf{x^{k-1}}, \mathbf{x^{k\prime\prime}}) \wedge \beta_{e_k}(\mathbf{x^{k\prime\prime}}, \mathbf{x^k}, z^k)$$

We identify the variable x_i with its 0th renaming; accordingly we can write $\mathbf{x^0}$ for $\mathbf{x}$.

Now by definition, we can write ψ_p as $\gamma_{e_p}(\mathbf{x}) \wedge \theta^1_p(bound(e_p)^p) \wedge \theta^2_p((\{x^p_1, \dots, x^p_n\} \backslash bound(e_p)^p), \mathbf{x})$ where for any subset $X \subseteq \{x_1, \dots, x_n\}$ we denote by X^j the set $\{x^j \mid x \in X\}$. For each i $(p < i \leq m)$, we can write ψ_i as $\gamma^1_{e_i}(bound(e_{i-1})^{i-1}) \wedge \gamma^2_{e_i}(\{x_1, \dots, x_n\} \backslash bound(e_{i-1})^{i-1}) \wedge \theta^1_i(bound(e_i)^i, past(e_i)^{i-1}) \wedge \theta^2_i((\{x_1, \dots, x_n\} \backslash bound(e_i))^i, (\{x_1, \dots, x_n\} \backslash past(e_i))^{i-1})$. Observe that under this rewriting, ψ_m rewrites to $\gamma^1_{e_m}(bound(e^{m-1}_{m-1}) \wedge \gamma^2_{e_m}(\{x_1, \dots, x_n\} \backslash bound(e_{m-1})^{m-1} \wedge \theta^1_m(\mathbf{x}^m, past(e_{m-1})^{m-1}) \wedge \theta^2_m((\{x_1, \dots, x_n\} \backslash past(e_m))^{m-1})$. Now we can transform the constraint

$$(\exists \mathbf{x} \exists \mathbf{x^p} \dots \exists \mathbf{x^{m-1}} (\varphi \wedge \psi_p \wedge \dots \wedge \psi_m))$$

to a constraint of the form

$$(\exists \mathbf{x} \exists \mathbf{x^p} \dots \exists \mathbf{x^{m-1}} (\varphi \wedge \psi \wedge \psi'))$$

such that $\mathbf{x}$ is not free in ψ' and the variables that occur free in ψ do not occur free in ψ'; in particular, the variables $\mathbf{x^m}$ occur free only in ψ' (this will hold since for each $p < i \leq m$, $past(e_i) \subseteq bound(e_{i-1})$). In this case, we can move the corresponding existential quantifiers inside; i.e., we can write the above constraint as

$$(\exists \mathbf{y}(\varphi \wedge \psi)) \wedge \exists \mathbf{y}' \psi'$$

where $\mathbf{y}$ occur free in $\varphi \wedge \psi$, and $\mathbf{y}' \backslash \mathbf{x^m}$ occur free in ψ'. If $\varphi \wedge \psi$ is unsatisfiable, then $[\![w]\!](\varphi)$ is unsatisfiable. Otherwise, if $\varphi \wedge \psi$ is satisfiable (which we have assumed), therefore the conjunct $\exists \mathbf{y} \varphi \wedge \psi$ is equivalent to $true$. Thus, $[\![w']\!](\varphi)$ is equivalent to a formula that does not depend on φ, and hence a constant function.

Let e_q be the least q such that $bound(e_q) = \{x_1, \ldots, x_n\}$ (such a q exists since $bound(e_m) = \{x_1, \ldots, x_n\}$). We rewrite the above constraint as follows.

$$\exists \mathbf{x} \exists \mathbf{x^p} \ldots \exists \mathbf{x^{m-1}} (\varphi \wedge \psi) \wedge \psi'$$

where

$$\begin{array}{rcl}
\varphi \wedge \psi & \equiv & (\varphi \wedge \gamma_{e_p}(\mathbf{x}) \\
& \wedge & \theta^2_p((\{x^p_1, \ldots, x^p_n\} \backslash bound(e_p)^p), \mathbf{x}) \\
& \wedge & \gamma^2_{p+1}(\{x^p_1, \ldots, x^p_n\} \backslash bound(e_{i-1})^p) \\
& \wedge & \theta^2_{p+1}((\{x^{p+1}_1, \ldots, x^{p+1}_n\} \backslash bound(e_{p+1})^{p+1}), ((\{x_1, \ldots, x_n\} \backslash past(e_{p+1}))^p)) \\
& \wedge & \ldots \\
& \wedge & \theta^2_q((\{x_1, \ldots, x_n\} \backslash past(e_q))^{q-1}) \\
& \wedge & \ldots \\
& \wedge & \theta^2_m((\{x_1, \ldots, x_n\} \backslash past(e_m))^{m-1}))
\end{array}$$

and

$$\begin{array}{rcl}
\psi' & \equiv & \theta^1_p(bound(e_p)^p) \wedge \gamma^1_{e_{p+1}}((bound(e_p))^p) \\
& \wedge & \theta^1_{p+1}(bound(e_{p+1})^{p+1}, past(e_{p+1})^p) \\
& \wedge & \ldots \\
& \wedge & \gamma^1_{e_m}(bound(e_{m-1})^{m-1}) \\
& \wedge & \theta^1_m(\mathbf{x}^m, past(e_{m-1})^{m-1})
\end{array}$$

Now let $w'' = e_1 \ldots e_{p-1}$. Then $w = w''.w'$. Hence the constraint transformer $[\![w]\!]$ associated with w is given by $[\![w]\!] = [\![w'']\!] \circ [\![w']\!]$. Since $[\![w']\!]$ is either unsatisfiable or constant function, $[\![w]\!]$ is also either unsatisfiable or a constant function.

Now seeking a contradiction, assume that symbolic forward analysis for $\mathcal{S}$ does not terminate with local entailment. Hence, there must be an infinite path p along the constraint tree. Following an argument in the proof of Theorem 5.1, p contains infinitely many occurrences of a simple cycle w; i.e., p is an element of the language $(\mathcal{E}^*.w)^\omega$. Now consider any two nodes $s_1 = w_1.w$ and $s_2 = w_2.w$ of p such that $s_1 < s_2$. Since the constraint transformer function labeling w is a constant function, the constraints labeling s_1 and s_2 are the same. Since this happens for every path p in the constraint tree, following the argument in Theorem 5.1, we can obtain a contradiction. Hence symbolic forward analysis for $\mathcal{S}$ terminates with local entailment. $[\![$

Corollary 7.1 *Symbolic forward analysis of an o-minimal hybrid system terminates with local entailment.*

Proof. It is easy to see that each simple cycle of an o-minimal hybrid system is initializable. Hence the result follows from an application of Proposition 7.1. $[\![$

7.7 Compositional Reasoning about Termination

In this section, we show how to reason compositionally about sufficient termination conditions in our framework. In order to motivate that just proving termination for individual components is not enough, consider Figure 7.1. The figure shows two hybrid systems $\mathcal{S}_1$ and $\mathcal{S}_2$. Each system is o-minimal and hence symbolic forward analysis for each terminates. The first system $\mathcal{S}_1$ consists of two locations ℓ^0 and ℓ^1 and one (program) variable x which increases with derivative 1 in each location. There is an edge from ℓ^0 to ℓ^1 labeled a. The second system $\mathcal{S}_2$ consists of a single location m^0 and an edge from m^0 to itself labeled b. The variable y is the only program variable. It increases with derivative 1 at the location m^0. The initial states (constraints) for $\mathcal{S}_1$ and $\mathcal{S}_2$ are respectively $L = \ell^0 \wedge x = 0$ and $L = m^0 \wedge y = 0$. The asynchronous parallel composition of $\mathcal{S}_1$ and $\mathcal{S}_2$ is not o-minimal. In fact, symbolic forward analysis for their asynchronous parallel composition does not terminate.

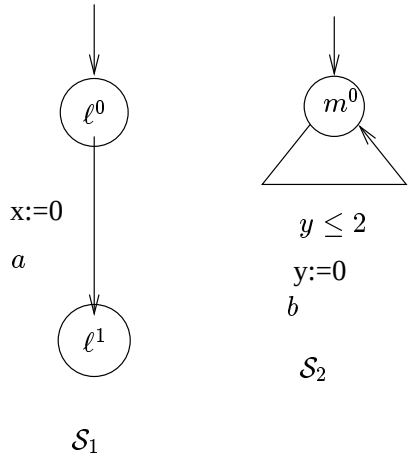


Figure 7.1: Example showing composition of o-minimal hybrid systems.

The above example illustrates the need to develop sophisticated compositional techniques to infer the termination of symbolic forward analysis of the composed system based on certatain sufficient criteria in the component systems. In the rest of this section, we provide sufficient conditions under which symbolic forward analysis of the parallel composition of n infinite state systems $\mathcal{S}_1, \ldots, \mathcal{S}_n$ is terminating. To this end, we first define the notion of an initialized edge.

Definition 7.5 (Initialized Edge) *An edge e of an infinite state system is said to be initialized if the free variables in the action α_e are among $\mathbf{x}'$.*

Let $\mathcal{S}_1, \ldots, \mathcal{S}_k$ be k infinite state systems with synchronizing alphabet sets $\Sigma_1, \ldots, \Sigma_k$. Below, for a finite set I, we write $\prod_{i \in I} \mathcal{S}_i$ for the parallel composition of infinite state systems $\mathcal{S}_i$ where $i \in I$.

Theorem 7.1 *If each simple cycle $w = e_1 \ldots e_m$ $(m \geq 1)$ of each $\mathcal{S}_i$*

- *contains an e_j $(1 \leq j \leq m)$ such that $lab(e_j) \in \Sigma_1 \cap \ldots \cap \Sigma_k$*
- *and for each $e \in w$ such that $lab(e) \in \Sigma_1 \cap \ldots \cap \Sigma_k$, e is an initialized edge*

then symbolic forward analysis for $\mathcal{S} = \mathcal{S}_1 || \ldots || \mathcal{S}_k$ terminates with local entailment.

Proof. We show that the constraint transformer function associated with each simple cycle in the composed is either a constant functionor unsatisfiable . The proof requires complicated combinatorial arguments. Before formally proving this, we state the basic intuition behind our proof method: since each simple cycle of each component $\mathcal{S}_j$ contains at least one edge e_i such that $lab(e_i) \in \bigcap_{l=1}^{k} \Sigma_l$, therefore in each simple cycle w of the composed system, each component "moves"; i.e., for each component there exists at least one edge in w such that the projection of that edge on that component is an edge in that component. We now formally state our proof. Formally, we first show that for any nonempty subset $I \subseteq \{1, \ldots, k\}$, in each simple cycle w of the composed system $\prod_{i \in I} \mathcal{S}_i$ there exists an edge e such that each component $\mathcal{S}_i$ $(i \in I)$ "moves" on that edge and $lab(e) \in \bigcap_{i=1}^{k} \Sigma_i$. We prove this by induction on the cardinality of I. The base case when I is a singleton is trivial. Let the statement hold for all subsets of $\{1, \ldots, k\}$ of size less than or equal to l. Let $I \subseteq \{1, \ldots, k\}$ be such that $|I| = l + 1$ and there exists a simple cycle w in the composed system $\prod_{i \in I} \mathcal{S}_i$ such that for each edge in w there exists a component $\mathcal{S}_i$ $(i \in I)$ such that $\mathcal{S}_i$ does not "move" on that edge. Now consider the simple cycle w. There exists a component $\mathcal{S}_j$ $(j \in I)$ such that the projection of w on $\mathcal{S}_j$ is a cycle in $\mathcal{S}_j$. Now pick up any $\alpha \in I$. Consider the composed system $\prod_{i \in I \backslash \{\alpha\}} \mathcal{S}_i$. The projection of w on this system contains a cycle w' in it. Let w'' be a simple cycle within w'. By induction hypothesis, there exists an edge e'' in w'' such that every component $\mathcal{S}_i$ $(i \in I \backslash \{\alpha\})$ "moves" on e'' and $lab(e'') \in \bigcap_{i=1}^{k} \Sigma_i$. Consider the edge e in w such that its projection on w' is e''. By our assumption, $\mathcal{S}_\alpha$ does not "move" on that e. But since $lab(e) \in \bigcap_{i=1}^{k} \Sigma_i$, therefore, by the definition of parallel composition, the existence of this edge e in the composed system $\prod_{i \in I} \mathcal{S}_i$ is impossible. Hence, we have shown that for any nonempty subset $I \subseteq \{1, \ldots, k\}$, in each simple cycle w of the composed system $\prod_{i \in I} \mathcal{S}_i$ there exists an edge e such that each component $\mathcal{S}_i$ $(i \in I)$ "moves" on that component and $lab(e) \in \bigcap_{i=1}^{k} \Sigma_i$. Thus, in every simple cycle $w = e_1 \ldots . e_m$ the composed system $\prod_{i=1}^{k} \mathcal{S}_i$, there exists an edge e such that every component $\mathcal{S}_i$ "moves" on that edge and $lab(e) \in \bigcap_{i=1}^{k} \Sigma_i$. We now show that the constraint transformer function $[\![w]\!]$ associated with each simple cycle w of $\mathcal{S}$ is either a constant function or unsatisfiable. Indeed, let $w = e_1 \ldots . e_m$ be any simple cycle of $\mathcal{S}$. Then, there exists an edge e such that every component $\mathcal{S}_i$ "moves" on that edge and $lab(e) \in \bigcap_{i=1}^{k} \Sigma_i$. Let $e = e_j$. Consider the projection of e_j on any component $\mathcal{S}_i$. The projection will be an edge e' in this component and also $lab(e') \in \bigcap_{i=1}^{k} \Sigma_i$. Hence, by the assumption of the theorem, only the primed variables are free in $\alpha_{e'}$. Hence, in $\mathcal{S}$, only the primed variables are free in α_e. Hence α_e is an initialized edge. It can be easily seen that the constraint transformer function associated with an initialized edge is either a a constant function or unsatisfiable. Let $w' = e_1 \ldots e_{j-1}$ and $w'' = e_{j+1} \ldots e_j$. Then the constraint transformer function associated with w is given by $[\![w]\!] = [\![w']\!] \circ [\![e_j]\!] \circ [\![w'']\!]$. Hence $[\![w]\!]$ is either a constant function or unsatisfiable. Now we can argue as in Proposition 7.1 and prove termination of symbolic forward analysis with local entailment. ∎

To see the applicability of our results, consider the two-process real time mutual exclusion protocol given in Figure 7.2. The critical section is denoted by *cs*. Here, the processes do not share real variables — the communication is through the synchronization labels. The set of synchronization labels Σ_1 of process $P1$ is the set $\{a, b, g, p, t1\}$ and that for process $P2$ $\Sigma_2 = \{a, b, g, q, t2\}$. Each process Pi has only one clock xi. It can be seen that this protocol satisfies the conditions of theorem 7.1. Hence, symbolic forward analysis for the protocol (i.e., symbolic forward analysis of the composed system) terminates.

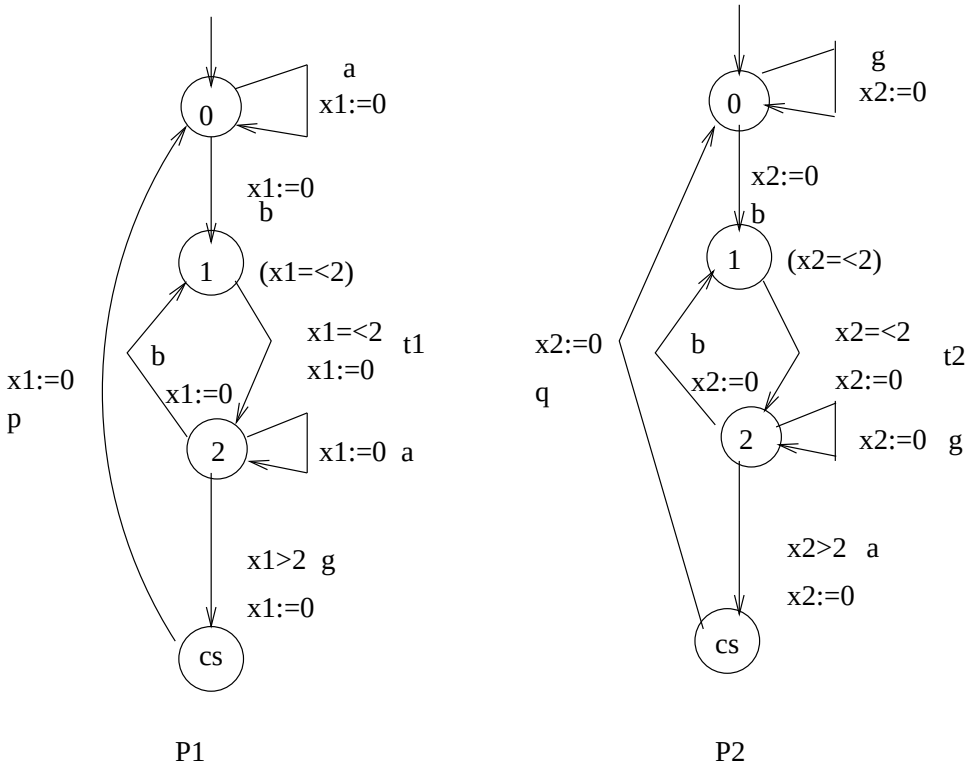


Figure 7.2: A two-process timed mutual exclusion protocol.

Our next two theorems are concerned with infinite state systems with integer-valued variables. Let $\mathcal{S}_1, \ldots, \mathcal{S}_k$ be k infinite state systems with integer-valued variables with synchronizing alphabet sets $\Sigma_1, \ldots, \Sigma_k$.

Theorem 7.2 *If each simple cycle $w = e_1 \ldots e_m$ $(m \geq 1)$ of each $\mathcal{S}_i$ is*

- *an initialized string and*
- *contains an e_i $(1 \leq i \leq m)$ such that $lab(e_i) \in \Sigma_1 \cap \ldots \cap \Sigma_k$*

then symbolic forward analysis for $\mathcal{S}_1 || \ldots || \mathcal{S}_k$ terminates with local entailment.

Proof. By following the same line of reasoning as in the proof of Theorem 7.1, we can show that in every simple cycle $w = e_1 \ldots . e_m$ the composed system $\prod_{i=1}^{k} \mathcal{S}_i$, there exists an edge such that every component $\mathcal{S}_i$ "moves" on that edge. This means that the projection of w on any component $\mathcal{S}_i$ contains a cycle w' in $\mathcal{S}_i$.

We now show that w is an initialized string. We show that for any variable x there exists a j such that x is bound (in w) in every edge in $e_j \ldots . e_m$. Precisely, we show that if $w' = \widetilde{e_1} \ldots . \widetilde{e_p}$, then there exists a t $(1 \leq t \leq p)$ such that x is bound (in w') in every edge in $\widetilde{e_t} \ldots . \widetilde{e_p}$ where the variable x belongs to $\mathcal{S}_i$. We prove this by induction on the nesting depth of the cycle w' contained in the projection of w on the component $\mathcal{S}_i$ that x belongs to. The base case is when the nesting depth is 0, i.e., when (the control graph of) w' is a simple cycle; i.e., w' is of the form as below where $w' = \widetilde{e_1} \ldots . \widetilde{e_p}$. Without loss of generality, we are assuming that the portion of the projection $\widetilde{w}$ of w on $\mathcal{S}_i$ from the end of w' to the end of $\widetilde{w}$ does not contain any cycle of $\mathcal{S}_i$.

$$\ell \ldots \overbrace{\ell \xrightarrow{\widetilde{e_1}} \ldots \xrightarrow{\widetilde{e_p}} \ell}^{w'} \ldots \ell$$

In this case, due to the fact that the components do not share variables and the assumption of the theorem, we can show that there exists a t such that x is bound (in w') in every edge in $\widetilde{e_t} \ldots . \widetilde{e_p}$. From this it follows that there exists a j such that x is bound (in w) in every edge in $e_j \ldots . e_m$. Now assume that the result holds for all w' such that the nesting depth of w' is less than or equal to q. Let w' be of nesting depth $q+1$. Then w' must contain a cycle w'' which is of nesting depth less than or equal to q. We can chose w'' such that the portion of w' from the end of w'' to the end of w' does not contain any nested cycle. This situation is depicted below.

$$\ell \ldots \overbrace{\ell \xrightarrow{\widetilde{e_1}} \ldots \xrightarrow{\widetilde{e_r}} \ell' \xrightarrow{\hat{e_1}} \ldots \xrightarrow{\hat{e_p}} \ell' \xrightarrow{\widetilde{e_t}} \ldots \xrightarrow{\widetilde{e_u}} \ell}^{w'} \ldots \ell$$
$$\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{w''}$$

Now there can be two cases. The first case is that there exists an edge e between $\widetilde{e_t}$ and $\widetilde{e_u}$ such that x is bound (in w') in every edge in $e \ldots . \widetilde{e_u}$. In this case, since the components do not share variables, we can easily show that there exists a j such that x is bound (in w) in every edge in $e_j \ldots . e_m$. For the other case in which such an edge does not exist we appeal to the induction hypothesis. By the induction hypothesis, there exists a t' such that x is bound in every edge in $\hat{e_{t'}} \ldots . \hat{e_p}$ in w''. Also, by the induction hypothesis, every variable in $\mathcal{S}_i$ is bound in $\hat{e_p}$ (in w''). The binding of x in $\hat{e_p}$ can be extended to $\widetilde{e_t} \ldots . \widetilde{e_u}$ (in w'). This is done as follows. First

observe that the string $\widetilde{e_t}.\ldots.\widetilde{e_u}$ is a part of a simple cycle $v = \check{e}_1.\ldots.\check{e}_p.\widetilde{e_t}.\ldots.\widetilde{e_u}$ of $\mathcal{S}_i$. Now we show that the following is an invariant. If a variable y is bound in any edge $\widetilde{e_a}$ between $\widetilde{e_t}$ and $\widetilde{e_u}$ in v, then it is also bound in $\widetilde{e_a}$ in the string $v' = \hat{e_{t''}}.\ldots.\widetilde{e_t}.\ldots.\widetilde{u}$ where t'' is the greatest t such that $\hat{e_t}\ldots\hat{e_p}$ is an initialized string (from the induction hypothesis such a string exists). Indeed, first consider any variable y that is bound in $\widetilde{e_t}$ in v. Then there exists a partition of $\{x_1,\ldots,x_n\}$ into two sets S and S' such that the guard constraint $\gamma_{\widetilde{e_t}}(\mathbf{x})$ can be written as a quantifier free formula of the form $\gamma^1_{\widetilde{e_t}} \wedge \gamma^2_{\widetilde{e_t}}$ where the free variables in $\gamma^1_{\widetilde{e_t}}$ are bound (in v) in $\check{e_p}$ and the free variables in $\gamma^2_{\widetilde{e_t}}$ are not bound in $\check{e_p}$ (in v) and the action $\alpha_{\widetilde{e_t}}(\mathbf{x},\mathbf{x}')$ can be written as a quantifier free formula of the form $\theta_1 \wedge \theta_2$ where

- the variables occurring free in θ_1 are among $bound(\widetilde{e_t})' \cup S$ (where we use the same notation as in the proof of Proposition 7.1.
- The variables in $bound(\widetilde{e_t})' \cup S$ do not occur free either in β_2 or in θ_2.
- Each variable in S is bound in $\check{e_p}$ (in v).

But each variable in S is bound in $\hat{e_p}$ (in v'). Hence y is bound in $\widetilde{e_t}$ in v'. Now suppose that this holds for all edges in $\widetilde{e_t}.\ldots.\widetilde{e_a}$. Then it can be easily shown that this holds for $\widetilde{e_t}.\ldots.\widetilde{e_{a+1}}$. Now according to the assumption of the theorem x is bound in every edge $\widetilde{e_a}$ between $\widetilde{e_t}$ and $\widetilde{e_u}$ in v. Hence it is bound in every edge bound in every edge $\widetilde{e_a}$ between $\widetilde{e_t}$ and $\widetilde{e_u}$ in $\hat{e_t}\ldots\widetilde{e_u}$. Thus x is bound in every edge in the string $\hat{e_t}\ldots\widetilde{e_u}$. From this it easily follows that there exists a j such that x is bound in every edge in $e_j.\ldots.e_m$. Thus we have proved that every simple cycle in the composed system $\mathcal{S}_1||\ldots||\mathcal{S}_k$ is an initialized string. The result of the theorem then follows from Proposition 7.1. ∎

Our next theorem also considers infinite state systems with integer-valued variables. The sufficient conditions provided are more graph-theoretic. We first define an exit point for a simple cycle $w = e_1\ldots e_m$.

Definition 7.6 (Exit Point) *An edge e_i of a simple cycle $w = e_1\ldots e_m$ is an exit point of w if the source location of e_i is a part of a cycle $w' \neq w$; i.e., it is also the source location of some edge in w'.*

For a simple cycle $w = e_1\ldots e_m$, we call the edge e_i the *last* exit point of w if e_i is an exit point of w and for all j with $i < j \leq m$, e_j is not an exit point of w; i.e., the source location of e_i is the last location in w from which one can leave w. If e_i is the last exit point of a simple cycle $w = e_1\ldots e_m$, we call the substring $w' = e_i\ldots e_m$ the *remainder section* of w.

Theorem 7.3 *Assume that each $\mathcal{S}_i$ is an infinite state system with integer-valued variables. Suppose that*

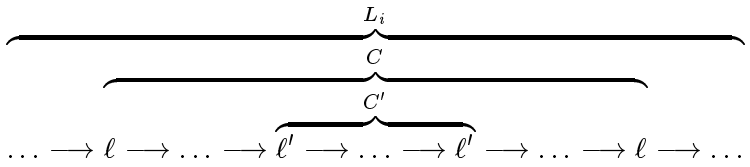
- *each simple cycle in each $\mathcal{S}_i$ is an initialized string and*
- *the remainder section of each simple cycle in each $\mathcal{S}_i$ contains an initialized string.*

Then symbolic forward analysis for $\mathcal{S}_1||\ldots||\mathcal{S}_k$ terminates with local entailment.

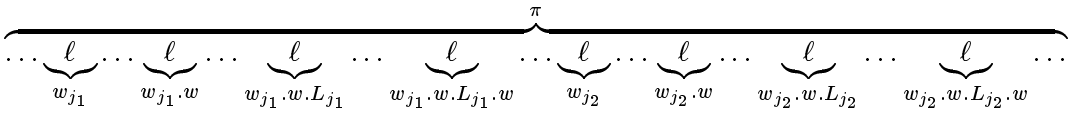
Proof. Seeking a contradiction, suppose that the constraint tree for the composed system contains an infinite branch π. Then some simple cycle w must repeat infinitely often along that branch. We now reason on the type of this simple cycle w.

Case 1. The first case is that each component $\mathcal{S}_i$ "moves" on w; i.e., for each component $\mathcal{S}_i$ there exists an edge e in w such that the projection of e on $\mathcal{S}_i$ is an edge in $\mathcal{S}_i$. In this case we can reason as in Theorem 7.2 and show that w is an initialized string. Then we can follow the reasoning of Proposition 7.1 and obtain a contradiction.

Case 2. This case is the negation of the first one. I.e., there exists at least one component that does not "move" on $w = e_1 \dots e_m$. Hence w may not be initialized. However for each component $\mathcal{S}_i$ that "moves" on w, for each variable x that belongs to $\mathcal{S}_i$, there exists a j such that x is reset in any edge in $e_j \dots e_m$. This can be proved by using reasoning similar to that in Theorem 7.2. Now since the branch π belongs to the language $(\mathcal{E}^* w)^\omega$, it is of the form $T_0.w.T_1.w.\dots$. Consider the components that do not "move" on w. Without loss of generality, the variables that belong to these components be $\{x_q, \dots, x_n\}$. Among these, let $\{x_q, \dots, x_s\}$ be the variables that belong to components that do not "move" after some point in the branch π. So the variables $\{x_{s+1}, \dots, x_n\}$ belong to components such that for every node τ on π, for each of these components there exists a descendant $\tau' = w'.e$ of τ such that the component "moves" on e. We go down the branch π beyond the point after which the components to which the variables $\{x_q, \dots, x_s\}$ belong do not "move" any more. Since the cycle w repeats infinitely often, we can find below this point two nodes labeled w_1 and $w_1.w$. From the latter node, we can still go down until we can find a stretch in which each of the components to which $\{x_{s+1}, \dots, x_n\}$ belong "move" at least once along this stretch. We can find two nodes beyond this "point" labeled w_2 and $w_2.w$. Let the word between $w_1.w$ and w_2 be denoted by L_1; i.e., $w_2 = w_1.w.L_1$. Since w repeats infinitely often along π, we can find nodes w_3, $w_3.w$, w_4 and $w_4.w$ such that $w_2.w < w_3$ and $w_4 = w_3.w.L_2$ where each component that contains variables among $\{x_{s+1}, \dots, x_n\}$ "moves" at least once in L_2. In this way we can get an infinite sequence of nodes w_i, $w_i.w$, w_{i+1}, $w_{i+1}.w$ where $w_{i+1} = w_i.w.L_i$ and each component that contains variables in $\{x_{s+1}, \dots, x_n\}$ "moves" at least once in L_i. Now notice that the edges in L_i on which the components, that contain variables $\{x_{s+1}, \dots, x_n\}$, "move" must lie within a cycle of $\mathcal{S}$. The situation for a component $\mathcal{S}_j$ that "moves" in L_i is shown below.



Note that by "unpumping" (where if $w = e_1 \dots e_m$ is a word and $C = e_k \dots e_l$ $(1 \leq k \leq l \leq m)$ is a cycle contained in w, then the word obtained from w by "unpumping" C is given by $e_1 \dots e_{k-1}.e_{l+1} \dots e_m$) is all the cycles that are inside C we can get a simple cycle. We say that a component "moves" within a simple cycle $\widetilde{C}$ if the component "moves" in C and $\widetilde{C}$ is obtained by unpumping all the cycles contained in C and the component "moves" in $\widetilde{C}$. Now consider the simple cycles in which each component that contains the variables in $\{x_{s+1}, \dots, x_n\}$ "moves" for the last time in L_i. That is, there is no "movement" of a component in L_i after its corresponding simple cycle (i.e., the simple cycle in which it "moves" for the last time in L_i). Since there are infinitely many L_is but finitely many simple cycles, there must exist infinitely many indices j_i such that in each L_{j_i}, for each component that contains variables in $\{x_{s+1}, \dots, x_n\}$, its last "movement" in L_{j_i} is contained in the same simple cycle. Consider L_{j_1} and L_{j_2}. Consider the following situation.



Let the constraints labeling the nodes w_{j_1}, $w_{j_1}.w$, $w_{j_1}.w.L_{j_1}$, $w_{j_1}.w.L_{j_1}.w$, w_{j_2}, $w_{j_2}.w$, $w_{j_2}.w.L_{j_2}$ and $w_{j_2}.w.L_{j_2}.w$ be φ_1, φ_2, φ_3, φ_4, φ_5, φ_6, φ_7 and φ_8 respectively. We will show that $\varphi_8 \models \varphi_4$. Then reasoning as in Proposition 7.1, we can obtain a contradiction.

Suppose that $\mathcal{N}, \mathbf{v} \models \varphi_8$. Then there must be a run from a solution $\mathbf{v}'$ of $\varphi_1$to $\mathbf{v}$. Let us denote this run by R. Thus

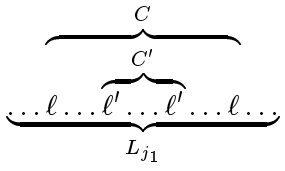
$$\langle v'_1, \ldots, v'_{q-1}, v_q, \ldots, v_s, v'_{s+1}, \ldots, v'_n \rangle \xrightarrow{R}{}^{*} \langle v_1, \ldots v_n \rangle$$

We will show that $\mathcal{N}, \mathbf{v} \models \varphi_4$. Since φ_3 is satisfiable, there must exist a solution $\langle v''_1, \ldots, v''_{q-1}, v_q, \ldots, v_s, v''_{s+1}, \ldots, v''_n \rangle$ of φ_3. Hence there must exist a run

$$\langle v''''_1, \ldots, v''''_{q-1}, v_q, \ldots, v_s, v''''_{s+1}, \ldots, v''''_n \rangle \longrightarrow^* \langle v''_1, \ldots, v''_{q-1}, v_q, \ldots, v_s, v''_{s+1}, \ldots, v''_n \rangle$$

from the node w_{j_1} to the node $w_{j_1}.w.L_{j_1}$ and $\mathcal{N}, \langle v''''_1, \ldots, v''''_{q-1}, v_q, \ldots, v_s, v''''_{s+1}, \ldots, v''''_n \rangle \models \varphi_1$. Let us call this run R'. Now we will construct a run from $\langle v''''_1, \ldots, v''''_{q-1}, v_q, \ldots, v_s, v''''_{s+1}, \ldots, v''''_n \rangle \longrightarrow^* \mathbf{v}$ from the node w_{j_1} to the node $w_{j_1}.w.L_{j_1}.w$. This will prove that $\mathcal{N}, \mathbf{v} \models \varphi_4$.

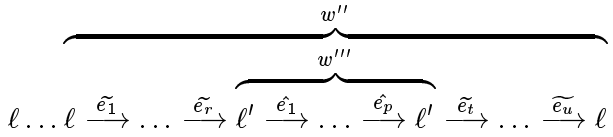
This run is constructed as follows. From the node w_{j_1} to the node $w_{j_1}.w$ we follow the run R'. Without loss of generality let $\mathcal{S}_1, \ldots, \mathcal{S}_l$ be the components that contain the variables $\{x_{s+1}, \ldots, x_n\}$. From the node $w_{j_1}.w$ to the node $w_{j_1}.w.L_{j_1}$, we follow the following strategy. For each of the edges in this stretch other than those in the simple cycles in which some components in $\{\mathcal{S}_1, \ldots, \mathcal{S}_l\}$ "moves" for the last time in L_{j_1}, we update the variables according to the run R'. For the simple cycles in which at least one of the components in $\{\mathcal{S}_1, \ldots, \mathcal{S}_l\}$ "moves" for the last time in L_{j_1}, we reason as follows. We first notice the following. Consider a cycle C nested in L_{j_1} in which the component $\mathcal{S}_i$ "moves" for the last time in L_{j_1}. It is of the form



where C' is a cycle nested inside C. Now there are two cases.

Case 2.1 The component $\mathcal{S}_i$ does not move in the stretch from the end of C' to the end of C. Then it must have "moved" in the stretch from the beginning of C to the beginning of C'. In that case, it cannot have "moved" in the cycle C'. Now the projection w' of the stretch from the beginning of C to that of C' on $\mathcal{S}_i$ must contain a cycle w'' of $\mathcal{S}_i$. We choose this w'' in such a way that the stretch from the end of w'' to the end of w' does not contain any cycle of $\mathcal{S}_i$. Suppose that $w'' = \widetilde{e_1} \ldots . \widetilde{e_p}$. Now we prove that for each variable x that belongs to $\mathcal{S}_i$, there exists a j such that x is bound (in w'') in every edge in $\widetilde{e_j} \ldots \widetilde{e_p}$. We prove this by induction on the nesting depth of w''. The base case when the nesting depth of w'' is zero, i.e., w'' is a simple cycle, is trivial. Now assume that the result holds for all w'' such that the nesting depth of w'' is less than or equal to q. Let w'' be of nesting depth $q+1$. Then w'' must contain a cycle

w''' which is of nesting depth less than or equal to q. We can chose w''' such that the portion of w'' from the end of w''' to the end of w'' does not contain any nested cycle. This situation is depicted below.



Now there can be two cases. The first case is that there exists an edge e between $\widetilde{e_t}$ and $\widetilde{e_u}$ such that x is bound (in w'') in every edge in $e.\ldots.\widetilde{e_u}$. In this case we have nothing to prove. The second case is when there does not exist such an edge. In this case we appeal to the induction hypothesis. By the induction hypothesis, there exists a t' such that x is bound in every edge in $\hat{e_{t'}}.\ldots.\hat{e_p}$ in w''. Also, by the induction hypothesis, every variable in $\mathcal{S}_i$ is bound in $\hat{e_p}$. The binding of x in e_p can be extended to $\widetilde{e_t}.\ldots.\widetilde{e_u}$. This is done as follows.

First observe that the string $\widetilde{e_t}.\ldots.\widetilde{e_u}$ is a part of a simple cycle $v = \check{e_1}.\ldots.\check{e_p}.\widetilde{e_t}.\ldots.\widetilde{e_u}$ of $\mathcal{S}_i$. Now we show that the following is an invariant. If a variable y is bound in any edge $\widetilde{e_a}$ between $\widetilde{e_t}$ and $\widetilde{e_u}$ in v, then it is also bound in $\widetilde{e_a}$ in the string $v' = \hat{e_{t''}}.\ldots.\widetilde{e_t}.\ldots.\widetilde{u}$ where t'' is the greatest t such that for the word $\widetilde{u} = \hat{e_t} \ldots \hat{e_p}$ every variable y in $\mathcal{S}_i$, there exists a j $(t \leq j \leq p)$ such that y is bound (in $\widetilde{u}$) in every edge in $\hat{e_j} \ldots \hat{e_p}$ (from the induction hypothesis). Indeed, first consider any variable y that is bound in $\widetilde{e_t}$ in v. Then there exists a partition of $\{x_1, \ldots, x_n\}$ into two sets S and S' such that the guard constraint $\gamma_{\widetilde{e_t}}(\mathbf{x})$ can be written as a quantifier free formula of the form $\gamma^1_{\widetilde{e_t}} \wedge \gamma^2_{\widetilde{e_t}}$ where the free variables in $\gamma^1_{\widetilde{e_t}}$ are bound in $\check{e_p}$ (in v) and the free variables in $\gamma^2_{\widetilde{e_t}}$ are not bound in $\check{e_p}$ (in v) and the action $\alpha_{\widetilde{e_t}}(\mathbf{x}, \mathbf{x}')$ can be written as a quantifier free formula of the form $\theta_1 \wedge \theta_2$ where

- the variables occurring free in θ_1 are among $bound(\widetilde{e_t})' \cup S$ (where we use the same notation as in the proof of Proposition 7.1).
- The variables in $bound(\widetilde{e_t})' \cup S$ do not occur free either in β_2 or in θ_2.
- Each variable in S is bound in $\check{e_p}$ (in v).

But each variable in S is bound in $\hat{e_p}$ (in v'). Hence y is bound in $\widetilde{e_t}$ in v'. Now suppose that this holds for all edges in $\widetilde{e_t}.\ldots.\widetilde{e_a}$. Then it can be easily shown that this holds for $\widetilde{e_t}.\ldots.\widetilde{e_{a+1}}$. Now according to the assumption of the theorem x is bound in every edge $\widetilde{e_a}$ between $\widetilde{e_t}$ and $\widetilde{e_u}$ in v. Hence it is bound in every edge bound in every edge $\widetilde{e_a}$ between $\widetilde{e_t}$ and $\widetilde{e_u}$ in $\hat{e_t} \ldots \widetilde{e_u}$. Thus x is bound in every edge in the string $\hat{e_t} \ldots \widetilde{e_u}$. Hence, we have shown that for every variable x that belongs to $\mathcal{S}_i$, there exists a j such that x is bound (in w'') in every edge in $\widetilde{e_j} \ldots \widetilde{e_p}$.

In this case, for the variables that belong to $\mathcal{S}_i$, we update them in the edges that corresponding to $\widetilde{e_j} \ldots \widetilde{e_p}$ in the same way as is done in the run R in L_{j_2}. Note that since the simple cycles in which $\mathcal{S}_i$ "moves" for the last time are same in L_{j_1} and L_{j_2}, we can do this kind of update (action).

Case 2.2. The component $\mathcal{S}_i$ "moves" in the stretch from the end of C' to the of C. We now show that projection u of the stretch from the end of C' to the end of C on the component $\mathcal{S}_i$ contains an initialized string. Indeed, let $\widetilde{C}$ be the simple cycle obtained by unpumping all the cycles that are contained in C. Consider the projection of $\widetilde{C}$ on the component $\mathcal{S}_i$. Since $\mathcal{S}_i$ "moves" in the stretch from the end of C' to the end of C, either the projection of this stretch

on the component $\mathcal{S}_i$ itself contains a simple cycle of $\mathcal{S}_i$, or there is a (projection of a) simple cycle of $\mathcal{S}_i$ whose starting point corresponds to an edge in the stretch from the beginning of C to the beginning of C' and it ends in the stretch from the end of C' to the end of C, after which the projection does not contain any cycle. In the former case, consider the last simple cycle contained in the projection of the stretch from the end of C' to the end of C. By the assumption of the theorem, it contains an initialized string. For the edges in L_{j_1} that corresponding to this initialized string, update the the variables belonging to $\mathcal{S}_i$ in the same way as in L_{j_2}.

In the latter case, there can be two subcases.

Case 2.2.1 The first subcase is that the component $\mathcal{S}_i$ does not "move" in C'. In this case, we reason as follows. If the component $\mathcal{S}_i$ does not "move" in the stretch from the end of C' to the end of C, then the projection of the stretch from the beginning of C to the beginning of C' on $\mathcal{S}_i$ must contain a simple cycle of $\mathcal{S}_i$. Hence on the projection u of the stretch from the beginning of C to that of C', we choose a simple cycle w' such that the stretch from the end of w' to the end of u does not contain any cycle. In this case, for the variables that belong to $\mathcal{S}_i$, we update them in the edges that correspond to w' in the same way as is done in the run R in L_{j_2}. If the component $\mathcal{S}_i$ does "move" in the stretch from the end of C' to the end of C, then if the projection of this stretch on $\mathcal{S}_i$ contains a simple cycle of $\mathcal{S}_i$, we can reason as in Case 2.1. Otherwise, the projection of C on $\mathcal{S}_i$ will contain a simple cycle $u_1.u_2$ of $\mathcal{S}_i$ such that u_1 belongs to the projection of the stretch from the beginning of C to the beginning of C' while u_2 belongs to the projection of the stretch from the end of C' to the end of C. In this case, for the variables that belong to $\mathcal{S}_i$, we update them in the edges that correspond to $u_1.u_2$ in the same way as is done in the run R in L_{j_2}.

Case 2.2.2 The second subcase is that the component $\mathcal{S}_i$ "moves" in the cycle C'. In this case, if the component $\mathcal{S}_i$ must "move" in the stretch from the end of C' to the end of C (otherwise $\widetilde{C}$ is not the simple cycle in which $\mathcal{S}_i$ "moves" for the last time in L_{j_1}). If the projection of the stretch from the end of C' to the end of C on $\mathcal{S}_i$ contains a simple cycle of $\mathcal{S}_i$, then we can reason as in Case 2.2.1. In the other case, the projection of C on $\mathcal{S}_i$ will contain a simple cycle $u_1.u_2$ of $\mathcal{S}_i$ such that u_1 belongs to the projection of the stretch from the beginning of C to the beginning of C' while u_2 belongs to the projection of the stretch from the end of C' to the end of C. Now the end of u_1 is an exit point of the simple cycle $u_1.u_2$. Hence from the second condition of this Theorem, u_2 must contain a initialized string. Let w' be the last initialized string contained in u_2 (i.e., if we let $u_2 = u.w'.u'$, then, for any u'' such that $u_2 = u''.w''.u'''$ and u is a prefix of u'', w'' is not a initialized string). We now show that $w'.u'$ is an initialized string. Indeed, consider any variable x tat belongs to $\mathcal{S}_i$. Of course x is bound in the edge e in $w'.u'$ where $w' = w''.e$. Either this binding extends all the way through u'. Or we must get an edge e' in u' such that $u' = u'''.e'.w'''$ and x is bound in $w'.u'$ in every edge in w'''. In this case, for the variables that belong to $\mathcal{S}_i$, we update them in the same way as is done in the run R in L_{j_2}.

Finally, we note that in all these cases, in the run created, we get a tuple $\langle v''_1, \ldots, v''_{q-1}, v_q, \ldots, v_s, v_{s+1}, \ldots, v_n \rangle$. Now we can easily construct a run from this tuple to the tuple $\mathbf{v}$ at the node $w_{j_1}.w.L_{j_1}.w$. This is done by the following method. In every edge in the stretch from tthe node $w_{j_1}.w.L_{j_1}$ to the node $w_{j_1}.w.L_{j_1}.w$, we update the variables in $\{x_1, \ldots, x_{q-1}\}$ in the same way as is done in R in the stretch from the node $w_{j_2}.w.L_{j_2}$ to the node $w_{j_2}.w.L_{j_2}.w$. Hence $\mathcal{N}, \mathbf{v} \models \varphi_4$. ‖

7.8 Related Work

Reachability analysis for infinite state systems with integer valued variables has been considered by Berard and Fribourg [BF99] as well as by Fribourg and Olsen [FO97]. Berard and Fribourg [BF99] did not identify any (interesting) subclass of such systems for which their reachability analysis procedure terminates. They relaxed the reachability analysis procedure over integers to reals with the observation that over the class of constraints that they have considered, elimination of variables over $\mathcal{R}$ (the domain of reals) using Fourier-Motzkin procedure is exact, i.e., produces the same result as the elimination of variables over $\mathcal{N}$ (the domain of natural numbers). However, it can be easily shown that over the class of constraints that they have considered, the real and integer solving algorithms perform exactly in the same way. Hence, the relaxation to reals does not provide any advantage with respect to computational complexity contrary to the claim in [BF99]. Like the work of Berard and Fribourg [BF99], Fribourg and Olsen [FO97] also do not provide any sufficient conditions for termination of their model checking procedure.

Abdulla, Cerans, Jonsson and Tsay [ACJT96] as well as Finkel and Schnoebelen [FS98] gave a unifying framework for deriving decidability results for model checking for infinite state systems. However, their framework requires finding a well quasi-ordering on the states. In many practical situations, finding such a well quasi-ordering on the states is not feasible. Besides, their method of deriving sufficient termination conditions for reachability analysis is monolithic; one has to consider the state-space of the composed system to show the termination of reachability analysis.

Comon and Jurski [CJ98] obtained decidability results for reachability analysis for a fragment of the class of multiple counter automata. They showed that the fixpoint of iterating transitions for this subclass of multiple counter automata is expressible in Presburger arithmetic. Again, their framework does not provide any means of reasoning about sufficient termination conditions compositionally.

Boigelot [Boi98] obtained sufficient conditions for termination of reachability analysis for infinite state systems with integer-valued variables based on graph-theoretic properties of the underlying control graphs. However, like the works mentioned above, his work does not provide a compositional way of reasoning about sufficient termination conditions.

Bultan, Gerber and Pugh [BGP97] presented a model checker for infinite state systems with integer-valued variables based on the Presburger solver from the Omega library [Pug92]. While [BGP97] provided model checking procedures for both safety and liveness properties, no sufficient conditions for termination of the procedures were provided.

Wong-Toi [WT95] has identified a subclass of linear hybrid systems called skewed clock automata that can be translated to timed safety automata. The subclass of skewed clock automata is closed under parallel composition. While symbolic backward analysis is guaranteed to terminate for skewed clock automata, symbolic forward analysis is possibly non-terminating for this subclass. However, as discussed in the previous chapters, symbolic forward analysis is widely used in practical experiments. It is also not clear how the methods of [WT95] can be extended to nonlinear hybrid systems.

Non-linear hybrid systems have been considered by Lafferriere, Pappas and Yovine [LPY99]. For the class of o-minimal hybrid systems, they proved the termination of symbolic backward analysis by showing that this class admits finite bisimulations. Using our toolbox, we have given a simple proof of the termination of symbolic forward analysis for o-minimal hybrid systems. In

fact this result has been obtained as a corollary of a more general theorem. While the reasoning about termination of symbolic backward analysis in [LPY99] is not compositional, our toolbox also allows compositional reasoning about termination of symbolic forward analysis for the class of hybrid systems considered in [LPY99].

Henzinger, Kopke, Puri and Varaiya [HKPV95] considered initialized rectangular automata, a subclass of linear hybrid systems, for which symbolic backward analysis is guaranteed to terminate. Henzinger [Hen95] considered hybrid automata with finite bisimulations for which symbolic backward analysis is guaranteed to terminate. But none of these works addressed the issue of compositional reasoning about sufficient termination conditions.

Lam and Brayton [LB93] considered alternating RQ timed automata which were closed under I/O composition. The class of alternating RQ automata is restrictive in the sense that it allows exactly one reset and exactly one query for each clock in an entire automaton. Moreover the notion of I/O composition that they used is much more restrictive than the notion of parallel composition used in this chapter. It is also not known whether symbolic forward analysis for alternating RQ timed automata is guaranteed to terminate.

Namjoshi [Nam98] considered model checking for parameterized systems in which each process is finite state. In contrast, in this chapter, we considered finite families of possibly infinite state systems.

In Chapter 5, we provided a framework for reasoning about sufficient termination conditions for symbolic forward analysis of timed automata. The present chapter is an extension of that framework to the more general context of infinite state systems with integer-valued variables and (nonlinear) hybrid systems as well as augmenting the framework with compositional reasoning.

Chapter 8

Constraint Transformer Monoids: A Unified Algebraic Framework for Abstract Symbolic Forward Analysis of Infinite State Systems

8.1 Introduction

Over the last few years, there has been an increasing research effort directed towards automatic verification of infinite state systems. Research on decidability issues (e.g., [ACJT96, ACHH93, Boi98, LPY99, HKPV95, CJ98]) has resulted in highly non-trivial algorithms for the verification of different subclasses of infinite state systems. These results do not, of course, imply termination guarantees for semi-algorithms on which practical tools are based (e.g., the decidability of the model checking problem for timed automata does not entail a termination guarantee for symbolic forward analysis of timed automata; symbolic forward analysis for timed automata is possibly non-terminating).

Practical tools generally use abstractions to guarantee (or speed-up) the termination of these semi-algorithms. The abstract semi-algorithms resulting from such abstractions may be always terminating but approximate (i.e., they always terminate but can produce don't know answers; for example the semi-algorithm with widening used in [HPR97]), or both terminating and accurate (e.g., the algorithm with the extrapolation operator in [DT98] and used in KRONOS) or possibly non-terminating and accurate (such abstract semi-algorithms are possibly non-terminating; but when they terminate they produce a yes/no answer; examples are the semi-algorithm with the cycle-step abstraction in [BBR97] and the semi-algorithm with accurate widening in [MP00a]). Many of these abstractions are inspired by the abstract interpretation framework of Cousot and Cousot [CC77].

Symbolic forward analysis is a semi-algorithm that in many cases solves the model checking problem for infinite state systems in practice. This semi-algorithm is implemented in many practical model checking tools like UPPAAL [BLL+96], KRONOS [DT98] and HYTECH [HHWT97]. This chapter presents a uniform algebraic framework for deriving abstract symbolic forward analysis procedures for a large class of infinite state systems with variables ranging over a numeric domain. We obtain the framework by lifting notions from classical algebraic theory of automata

to constraints representing sets of states. Our framework provides sufficient conditions under which the derived abstract symbolic forward analysis procedure is always terminating or accurate or both. The class of infinite state systems that we consider here are (possibly non-linear) hybrid systems and (possibly non-linear) integer-valued systems. The central notions involved are those of *constraint transformer monoids* and *coverings* between constraint transformer monoids. We show concrete applications of our framework in deriving abstract symbolic forward analysis algorithms for timed automata and the two process bakery algorithm that are both terminating and accurate.

Our results suggest a potential optimization of the (abstract) symbolic forward analysis procedures. Namely, the termination guarantees continue to hold even when the fixpoint test is made more efficient by weakening it to *local entailment* (explained below; e.g., for linear arithmetic constraints over reals, the complexity of fixpoint test reduces from co-NP hard to polynomial).

8.2 Infinite State Systems

We recall the notion of infinite states systems from Chapter 7. We assume that the program variables range over the set of natural numbers $\mathcal{N}$ or the set of reals $\mathcal{R}$, and the guard and the action formulas are $Arith(\mathcal{N})$ (the theory of natural numbers with addition, multiplication and order; it is interpreted over the structure $\langle \mathcal{N}, <, +, \cdot, 0, 1 \rangle$) or $OF(\mathcal{R})$ (the theory of the ordered field of reals; it is interpreted over the structure $\langle \mathcal{R}, <, +, \cdot, 0, 1 \rangle$) formulas. Below, we will refer to $OF(\mathcal{R})$ or $Arith(\mathcal{N})$ formulas as constraints. For a formula φ with free variables $\mathbf{x}$, we denote by $\varphi(\mathbf{x}')$, the formula obtained by replacing the free variables $\mathbf{x}$ of φ by $\mathbf{x}'$. Similar to Chapter 7, we will use constraints φ to represent certain sets of states of the system.In the sequel, we assume only *conjunctive* constraints; i.e., constraints that are conjunctions of atomic constraints of the form $t\, relop\, c$ where t is a term, $c \in \mathcal{N}$ and $relop \in \{>, <, \geq, \leq\}$. Examples of systems as described above include the bakery algorithm, the bounded buffer producer-consumer problem etc. as well as the so-called hybrid systems.

8.3 Constraint Transformer Monoids

Our definition of constraint transformer monoids is inspired by the definition of (syntactic) transformation monoids in [Eil76]. Let Φ be a (possibly infinite) set of satisfiable constraints (i.e., each constraint in Φ is satisfiable). We denote the set of all partial functions $\Phi \longrightarrow \Phi$ by $\mathcal{SF}(\Phi)$. Let 1_Φ denote the identity function. The set $\mathcal{SF}(\Phi)$ forms a monoid with functional composition as the multiplication and 1_Φ as the identity element. A *constraint transformer semigroup* is a pair $\langle \Phi, S \rangle$ where S is a subsemigroup of $\mathcal{SF}(\Phi)$. The constraint transformer semigroup $\langle \Phi, S \rangle$ is a constraint transformer monoid if the identity function 1_Φ is in S. The elements of Φ are called *symbolic states*. The elements of S are called *constraint transformers*. A constraint transformer monoid $X = \langle \Phi, S \rangle$ is a constraint transformer submonoid of a constraint transformer $Y = \langle \Phi', S' \rangle$ if $\Phi \subseteq \Phi'$ and S is a submonoid of S'.

By the denotation of set of constraints Φ, we represent the denotation of their disjunction; i.e., $[\Phi] = \bigcup_{\varphi \in \Phi} [\varphi]$.

We next define a *syntactic order* $\sqsubseteq^\varphi$ on a constraint transformer monoid $X = \langle \Phi, S \rangle$ with respect to a constraint $\varphi \in \Phi$ as follows.

Syntactic Order. For $w, w' \in S$, $w \sqsubseteq^{\varphi} w'$ iff $w(\varphi) \models w'(\varphi)$. Given a set of constraints $\Psi \subseteq \Phi$, we say that an infinite sequence $w_0, w_1, \ldots$, where $w_i \in S$, is *syntactically increasing* with respect to Ψ if for all $i \geq 1$, there exists $\varphi \in \Psi$ such that $w_i \not\sqsubseteq^{\varphi} w_j$ for all $j < i$.

Finitary Constraint Transformer Monoids. We say that a constraint transformer monoid X is *finitary* with respect to a set $\Psi \subseteq \Phi$ of constraints if there does not exist any syntactically increasing infinite sequence with respect to Ψ. Note that X is finitary does not mean that Φ is finite.

Reachability. For a constraint transformer monoid $X = \langle \Phi, S \rangle$, a reachability question is of the form: given $\varphi^1, \varphi^2 \in \Phi$, does there exist a $w \in S$ such that $\varphi^2 = w(\varphi^1)$?

Constraint transformer monoids generated by infinite state systems: We now show how an infinite state system generates a constraint transformer monoid. *We identify two constraints φ and φ' iff they have the same denotations; i.e.,* $[\varphi] = [\varphi']$. We recall the notion of constraint transformers from Chapter 5. The constraint transformer monoid generated by an infinite state system $\mathcal{S}$ is given by $CT(\mathcal{S}) = \langle \Phi, S \rangle$ where $\Phi = \{\varphi \mid \exists w \in \mathcal{E}^* [\![w]\!](\varphi^0) = \varphi\}$ and $S = \{[\![w]\!] \mid w \in \mathcal{E}^*\}$ with functional composition as the multiplication in S and $[\![\varepsilon]\!]$ as the unit element.

8.4 Coverings of Constraint Transformer Monoids

Our definition of covering between constraint transformer monoids is inspired by that of covering between (syntactic) transformer monoids in [Eil76]. Let $X = \langle \Phi, S \rangle$ and $Y = \langle \Phi', S' \rangle$ be two constraint transformer monoids. Let f be a total (binary) relation from Φ to Φ'. For $w \in S$ and $v \in S'$, we consider the following diagram.

$$\begin{array}{ccc} \Phi & \xrightarrow{w} & \Phi \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f} \\ \Phi' & \xrightarrow[v]{} & \Phi' \end{array}$$

If the above diagram commutes, i.e., for all $\varphi \in \Phi$, $v(\{\psi \mid f(\varphi, \psi)\}) = \{\psi \mid f(w(\varphi), \psi)\}$, then we say that v *covers* w with respect to f where for a set $\tilde{\Phi}$, $v(\tilde{\Phi}) = \{v(\varphi) \mid \varphi \in \tilde{\Phi}\}$. If for each $w \in S$ there exists a $v \in S'$ such that v covers w we say that the relation f is a *covering* between X and Y. We say that the constraint transformer monoid Y covers the constraint transformer monoid X if a covering f exists between X and Y and we write $X \prec Y$. We are now going to define a quotient of X with respect to f; we call such a quotient an f-quotient of X.

f quotient. In order to define an f-quotient of X, we first define an equivalence relation $\sim_f$ on Φ as follows. For $\varphi, \varphi' \in \Phi$,

$$\varphi \sim_f \varphi' \iff \{\psi \mid f(\varphi, \psi)\} = \{\psi' \mid f(\varphi', \psi')\}.$$

Next we define a representant function $rep : \Phi/\sim_f \longrightarrow \Phi$ as $rep([\varphi]) = \varphi$, where $[\varphi]$ is the equivalence class of φ with respect to the equivalence relation $\sim_f$. Given a covering relation f and

a representant function rep as above, we call the constraint transformer monoid $X' = \langle rep(\Phi/\sim_f), \hat{S}\rangle$ an f-quotient of X where $\hat{S} = \{\widetilde{w} \mid w \in S\}$ and for any constraint $\psi \in rep(\Phi/\sim_f)$, $\widetilde{w}(\psi) = rep([\psi'])$ iff $w(\psi) = \psi'$.

Canonicity and Saturation. We say that a constraint $\varphi \in \Phi$ is *canonical* with respect to f if for all $\varphi' \in \Phi$ with $\varphi \neq \varphi'$, $\{\psi \mid f(\varphi, \psi)\} \neq \{\psi' \mid f(\varphi', \psi')\}$. We say that the relation f *saturates* a constraint $\varphi \in \Phi$ if there exists a constraint $\psi \in \Phi'$ such that $f(\varphi, \psi)$ and for all $\varphi' \in \Phi'$ with $f(\varphi', \psi)$, we have $\varphi = \varphi'$. The notions of canonicity and saturation indicate the "local" distinguishing power of f.

Definition 8.1 (Homeocovering) *We say that f is a* homeocovering *from X to Y with respect to constraints φ^1 and φ^2 if f is a covering from X to Y and one of the following conditions hold.*

- *either f^{-1} is a covering from Y to a constraint transformer submonoid $X' = \langle \Phi'', S''\rangle$ of X (i.e., $Y \prec X'$ and f^{-1} witnesses the covering) and $\varphi^1, \varphi^2 \in \Phi''$,*
- *or f^{-1} is a covering from Y to an f-quotient X' of X (i.e., $Y \prec X'$ and f^{-1} is a witness to this covering) and φ^1 and φ^2 are both canonical with respect to f*

Definition 8.2 (Finitary Covering) *We say that a covering f is a* finitary covering *from X to Y with respect to a set of constraints $\Psi \subseteq \Phi$, if f is a covering from X to Y and Y is finitary with respect to $\{\psi \mid f(\varphi, \psi), \varphi \in \Psi\}$.*

Note that even if f is a finitary covering from X to $Y = \langle \Phi', S'\rangle$, it does not mean that Φ' is finite. We will use the notion of finitary coverings to provide sufficient conditions for termination of abstract symbolic forward analysis in Theorem 8.1.

Proposition 8.1 *Let $\mathcal{S}$ be an infinite state system. Let $X = \langle \Phi, S\rangle$ be the constraint transformer monoid generated by $\mathcal{S}$. Let $Y = \langle \Phi', S'\rangle$ be a constraint transformer such that $X \prec Y$ with f being a covering between X and Y. Suppose that a constraint φ^2 is reachable from the initial constraint φ^1 in $\mathcal{S}$. Then there exists $v \in S'$ such that*

$$\{\psi \mid f(\varphi^2, \psi)\} = v(\{\psi^1 \mid f(\varphi^1, \psi^1)\}).$$

If, in addition, f saturates φ^1 and f is homeocovering from X to Y with respect to φ^1 and φ^2, then the converse also holds.

Proof. The equality follows directly from the definition of covering between constraint transformer monoids. Indeed, if φ^2 is reachable from φ^1, then there exists a $w \in \mathcal{E}^*$ such that $[\![w]\!](\varphi^1) = \varphi^2$. Since f is a covering between X and Y, there exists $v \in S'$ that covers $[\![w]\!]$. Hence, the equality follows from the definition.

Now assume the equality. If f saturates φ^1 and one of the two conditions for homeocovering holds, then we show that there exists $w \in \mathcal{E}^*$ such that $[\![w]\!](\varphi^1) = \varphi^2$. Suppose that the first condition holds. Since f saturates φ^1, there must exist a constraint ψ^1 in Φ' such that $f(\varphi^1, \psi^1)$ and for all φ' such that $f(\varphi', \psi^1)$, $\varphi^1 = \varphi'$, i.e., $\varphi^1 = \{\varphi' \mid f(\varphi', \psi^1)\}$. Also, by the assumed equality, $f(\varphi^2, v(\psi^1)$. Since f^{-1} is a covering between Y and X', there exists a $w \in \mathcal{E}^*$ such

that $[\![w]\!]$ covers v. Therefore $\{[\![w]\!](\varphi^1)\} = \{\varphi' \mid f(\varphi', v(\psi^1)\}$. Therefore $\varphi^2 = [\![w]\!](\varphi^1)$. Hence, φ^2 is reachable from φ^1.

Suppose now that the second condition holds. Let $X' = \langle \Phi'', S'' \rangle$ be an f-quotient of X with rep as the chosen representant function. Since f saturates φ^1, there exists ψ^1 such that $f(\varphi^1, \psi^1)$ and for all φ' such that $f(\varphi', \psi^1)$, we have $\varphi' = \varphi^1$. Since φ^1 and φ^2 are canonical with respect to f, we have $rep([\varphi^1]) = \varphi^1$ and $rep([\varphi^2]) = \varphi^2$. Hence, we have, $f(rep([\varphi^1], \psi^1)$. By the assumed equality, there exists $v \in S'$ such that $f(\varphi^2, v(\psi^1))$. Since, f^{-1} is a covering between X and X', there exists $\widetilde{[\![w]\!]} \in S''$, such that $\{\widetilde{[\![w]\!]}(rep([\varphi^1]))\} = \{rep([\varphi]) \mid f(rep([\varphi]), v(psi^1))\}$. Since, $rep([\varphi^2])$ is in the right hand side of this equality, therefore, $\widetilde{[\![w]\!]}(rep([\varphi^1])) = rep([\varphi^2])$. By canonicity of φ^1 and φ^2 with respect to f, $\varphi^2 = [\![w]\!](\varphi^1)$.

∎

8.5 Constraint Trees and Symbolic Forward Analysis

Given a constraint transformer monoid $X = \langle \Phi, S \rangle$ with a finite set of generators $\widetilde{S}$ (i.e., $\widetilde{S}$ generates S), we define the constraint tree for X as follows. Let S^{free} be the free monoid generated by $\widetilde{S}$. For $\widetilde{w} \in S^{free}$, we say that $w \in S$ is the *companion* of $\widetilde{w}$ iff w is obtained by replacing concatenation in $\widetilde{w}$ with multiplication in S. Thus, for example, $w \in S$ is a companion of $g_1.g_2$ iff $w = g_1 \circ g_2$ where $\circ$ is the multiplication in S.

Definition 8.3 (Constraint Tree) *The constraint tree for $X = \langle \Phi, S \rangle$ with respect to a constraint $\varphi^0 \in \Phi$ and a finite set of generators $\widetilde{S}$ of S is an infinite tree with domain S^{free} that labels the node $\widetilde{w}$ by the constraint $w(\varphi^0)$ where w is the companion of $\widetilde{w}$.*

That is, the root ε is labeled with φ^0. For a node $\widetilde{w}$ labeled φ, for each $g \in \widetilde{S}$, the successor node $\widetilde{w}.g$ is labeled by $g(\varphi)$. We are now in a position to define symbolic forward analysis of a finitely generated constraint transformer monoid with respect to a constraint formally. A symbolic forward analysis is a traversal of (a finite prefix of) a constraint tree in a particular order. The following definition of a non-deterministic procedure abstracts away from that specific order.

Definition 8.4 (Symbolic Forward Analysis) *A symbolic forward analysis of a finitely generated constraint transformer monoid X with respect to a constraint φ^0 and a finite set of generators $\widetilde{S}$ is a procedure that enumerates constraints φ_i labeling the nodes $\widetilde{w}_i$ of the constraint tree of X with respect to φ^0 and $\widetilde{S}$ in a tree order such that the following holds.*

- *$\varphi_i = w_i(\varphi^0)$ for $0 \leq i < B$ where the bound B is either a natural number or ω and w_i is the companion of the word $\widetilde{w}_i \in S^{free}$,*
- *if $\widetilde{w}_i$ is a prefix of $\widetilde{w}_j$ then $i \leq j$,*
- *the disjunction $\bigvee_{0 \leq i < B} \varphi_i$ is equivalent to the disjunction $\bigvee_{0 \leq i < \omega} \varphi_i$.*

The number i is a leaf of a symbolic forward analysis if the node $\widetilde{w}_i$ is a leaf of the tree formed by all the nodes $\widetilde{w}_i$ where $0 \leq i \leq B$. We say that a symbolic forward analysis terminates if its bound B is finite. We define that a symbolic forward analysis terminates with local entailment if for all its leaves i there exists a $j < i$ such that the constraint φ_i entails the constraint φ_j (remember that each φ_i is a conjunctive constraint). In contrast, a symbolic forward analysis terminates with global entailment if for all its leaves i, the constraint φ_i entails the disjunction

of the constraints φ_j where $j < i$. For constraint domains that do not satisfy the independence property[1], checking for global entailment is usually more expensive than checking for local entailment. Many model checkers use local entailment for their fixpoint test (e.g., UPPAAL [LPY95b] uses identity; the model checker for infinite state systems described in [DP99a] uses local entailment).

Remark 8.1 *A symbolic forward analysis for an infinite state system $\mathcal{S}$ with respect to the initial constraint φ^0 is a symbolic forward analysis of the constraint transformer monoid generated by $\mathcal{S}$ with respect to φ^0. If terminating, the constraint $\bigvee_{0 \leq i < B} \varphi_i$ represents the set of all reachable states in $\mathcal{S}$. For an infinite state system $\mathcal{S}$, a constraint φ^2 is reachable from the constraint φ^1 if there exists a node $\widetilde{w}$ labeled by φ^2 in the constraint tree with respect to φ^1 of the constraint transformer monoid generated by $\mathcal{S}$.*

8.6 Abstract Constraint Trees and Abstract Symbolic Forward Analysis

Let $X = \langle \Phi, S \rangle$ be a constraint transformer monoid with a finite set of generators $\widetilde{S}$. Let S^{free} be the free monoid generated by $\widetilde{S}$. Let $Y = \langle \Phi', S' \rangle$ be a constraint transformer monoid covering X and let f be a covering relation witnessing the covering. We define an abstract constraint tree of $\mathcal{S}$ with respect to Y, f, $\widetilde{S}$ and a constraint φ^0 as follows.

Definition 8.5 (Abstract Constraint Tree) *An abstract constraint tree for X with respect to the constraint transformer monoid Y, a constraint φ^0, a finite set of generators $\widetilde{S}$ and a covering relation f is an infinite tree with domain S^{free} that labels the node $\widetilde{w} \in S^{free}$ by the set of constraints $\Psi = \{v(\psi^0) \mid f(\varphi^0, \psi^0)\}$ where $v \in S'$ covers w (the companion of $\widetilde{w}$).*

In the above definition we assume that there is a *finite representation* for each Ψ labeling $\widetilde{w} \in S^{free}$ in the abstract constraint tree. Note that the constraint tree for X with respect to φ^0 is an abstract constraint tree for X with respect to the constraint transformer monoid X and the identity function as the covering. Also note that the constraint transformer monoid Y may be arbitrary; i.e., it need not be finitely generated. If for each $w \in S$, we fix a $v \in S'$ covering w, we call the resulting abstract constraint tree a fixed-cover abstract constraint tree. Below, whenever we talk about abstract constraint tree, we assume a fixed cover $\mathcal{C} \subseteq S'$, i.e., for each $w \in S$ there exists a unique $v \in \mathcal{C}$ such that v covers w. We denote by $T_{\mathcal{C}}$ be the abstract constraint tree of X with respect to Y, f and $\mathcal{C}$, i.e., a node $\widetilde{w}$ is labeled by $\{v(\psi^0) \mid f(\varphi^0, \psi^0)$ where v is the unique element of $\mathcal{C}$ covering w (the companion of $\widetilde{w}$) and $\{\psi^0 \mid f(\varphi^0, \psi^0)\}$ labels the root. We are now in a position to define formally abstract symbolic forward analysis. An abstract symbolic forward analysis of X with respect to a constraint transformer monoid Y is a traversal of (a finite prefix of) the (fixed cover) abstract constraint tree of X with respect to Y in a particular order. The following definition of a non-deterministic procedure abstracts away from that specific order.

Definition 8.6 (Abstract Symbolic Forward Analysis) *An abstract symbolic forward analysis of a constraint transformer monoid X with respect to a constraint φ^0 and a fixed cover*

[1] A constraint domain is said to satisfy the independence property if for any constraint ψ and a set of constraints Φ, $\psi \models \bigvee_{\varphi \in \Phi} \varphi$ iff there exists $\varphi \in \Phi$ such that $\psi \models \varphi$

$\mathcal{C}$ is a procedure that enumerates the sets of constraints Ψ_i labeling the nodes $\widetilde{w_i}$ of the abstract constraint tree $T_{\mathcal{C}}$ with respect to φ^0 (and $\mathcal{C}$) in a tree order such that the following holds.

- *$\Psi_i = \{v_i(\psi^0) \mid f(\varphi^0, \psi^0)\}$ where $v_i \in \mathcal{C}$ covers $w_i \in S$ (the companion of $\widetilde{w_i}$) and the bound B is either a natural number or ω,*
- *if $\widetilde{w_i}$ is a prefix of $\widetilde{w_j}$ then $i \leq j$,*
- *the disjunction $\bigvee_{0 \leq i < B} \bigvee \Psi_i$ is equivalent to the disjunction $\bigvee_{0 \leq i < \omega} \bigvee \Psi_i$ where $\bigvee \Psi_i \equiv \bigvee_{\varphi \in \Psi_i} \varphi$.*

Similar to symbolic forward analysis, we say that an abstract symbolic forward analysis terminates if the bound B is finite; the concept of a leaf is defined similarly. We say that an abstract symbolic forward analysis terminates with local entailment if for all its leaves i, for each constraint $\varphi \in \Psi_i$, there exists a $j < i$, and a constraint $\varphi' \in \Psi_j$ such that $\varphi \models \varphi'$. The notion of termination with global entailment is defined in the obvious way.

We now present sufficient conditions under which an abstract symbolic forward analysis is possibly non-terminating and accurate, terminating and possibly inaccurate or both terminating and accurate with respect to a reachability question.

Theorem 8.1 *Let $X = \langle \Phi, S \rangle$ be a constraint transformer monoid having a finite set of generators $\widetilde{S}$. Let $\varphi^1, \varphi^2 \in \Phi$. Let $Y = \langle \Phi', S' \rangle$ be a constraint transformer monoid covering X with f witnessing the covering and let $\mathcal{C} \subseteq S'$ be a fixed cover. Then the following hold.*

1. *Suppose that for all i, $\Psi_i \neq \{\psi \mid f(\varphi^2, \psi)\}$ where Ψ_i is the set of constraints labeling the node $\widetilde{w_i}$ of the abstract constraint tree of X with respect to φ^1, Y, f, and $\mathcal{C}$. Then the constraint φ^2 is not reachable from φ^1 in X.*
   (a) *If, in addition, f is a finitary covering with respect to $\{\varphi^1\}$, then each abstract symbolic forward analysis of X with respect to φ^1, Y, f and $\mathcal{C}$ terminates with local entailment. In this case, abstract symbolic forward analysis always terminates with local entailment but may produce a 'don't know' answer the reachability question.*
2. *If f saturates φ^1 and f is a homeocovering from X to Y with respect to constraints φ^1 and φ^2 then φ^2 is reachable from φ^1 in X iff there exists an i such that $\Psi_i = \{\psi \mid f(\varphi^2, \psi)\}$ where Ψ_i labels the node $\widetilde{w_i}$ in the abstract constraint tree of X with respect to Y, f, φ^1 and $\mathcal{C}$. In this case, abstract symbolic forward analysis is possibly non-terminating; but when it terminates, it produces a yes/no answer for the reachability question.*
   (a) *In particular, if f is a function then φ^2 is reachable from φ^1 in X iff ψ is not reachable from ψ^0 in the abstract symbolic forward analysis of X with respect to Y, f, φ^1 and $\mathcal{C}$ where $f(\varphi^1, \psi^0)$ and $f(\varphi^2, \psi)$.*
   (b) *If, in addition, f is a finitary covering with respect to $\{\varphi^1\}$ then each abstract symbolic forward analysis of X with respect to φ^1, Y, f and $\mathcal{C}$ terminates with local entailment. In this case, abstract symbolic forward analysis always terminates with local entailment and is accurate.*

Proof. The first statement follows from Proposition 8.1. Suppose that the inequality in the first statement of the theorem holds for all i. Seeking a contradiction, suppose that φ^2 is

reachable from φ^1. Then, there exists $w_i \in S$ such that $\varphi^2 = w_i(\varphi^1)$. Now consider $\widetilde{w_i} \in S^{free}$ such that w_i is the companion of $\widetilde{w_i}$. The node $\widetilde{w_i}$ in the abstract constraint tree $T_{\mathcal{C}}$ is labeled by $\Psi_i = \{v_i(\psi^0) \mid f(\varphi^1, \psi^0)\}$ where $v_i \in \mathcal{C}$ covers w_i. By Proposition 8.1, $\Psi_i = \{\psi \mid f(\varphi^2, \psi)\}$. Hence a contradiction.

Suppose, f is a finitary covering with respect to φ^1. Consider, first, the case when Y is finite. Then, along every branch of the abstract con the abstract constraint tree, there exists two nodes $\widetilde{w_i}$ and $\widetilde{w_j}$, where $j < i$, labeled by the same set of constraints Ψ. Hence, any abstract symbolic forward analysis terminates with local entailment. Suppose now that Y is finitary with respect to $\{\psi^0 \mid f(\varphi^1, \psi^0)\}$. Then, along any branch of the constraint tree there exists a node $\widetilde{w_i}$ labeled by Ψ_i such that for each constraint $\psi \in \Psi_i$, there exists a $j < i$ and a constraint $\psi' \in \Psi_j$ such that $\psi \models \psi'$. The statement 2 in the theorem follows from a direct application of Proposition 8.1. ‖

8.7 Applications

In this section, we show concrete applications of the framework developed above to timed automata and the two-process bakery algorithm.

8.7.1 Timed Automata

We recall the notion of timed automata from Chapter 5. Symbolic forward analysis of timed automata is possibly non-terminating [MP99]. In order to define an abstract symbolic forward analysis for timed automata, we recall the trim operation on constraints from Chapter 3.

Let $\mathcal{T}$ be a timed automaton and let $X = \langle \Phi, S \rangle$ be the constraint transformer monoid generated by $\mathcal{T}$. We define the constraint transformer monoid Y obtained by trimming as follows.

Definition 8.7 (Constraint transformer monoid obtained by trimming) *Given a timed automaton $\mathcal{T}$, the constraint transformer monoid Y obtained by trimming is defined as $Y = \langle \Phi', S' \rangle$, where $\Phi' = \{trim(\varphi) \mid \varphi \in \Phi\}$ and $S' = \{\widetilde{w} \mid w \in S\}$ and $\widetilde{w}(trim(\varphi)) = trim(\varphi')$ if $w(\varphi) = \varphi'$.*

It can be easily verified that each $\widetilde{w}$ is a function from Φ' to Φ' and that S' is a monoid with the identity function as the unit element.

Proposition 8.2 *For a timed automaton $\mathcal{T}$ with the generated constraint transformer monoid $X = \langle \Phi, S \rangle$, the constraint transformer monoid Y obtained by trimming covers X with the function $f : \varphi \mapsto trim(\varphi)$ (note that the trim operation is a function) witnessing the covering.*

Proof. Follows from Proposition 3.2. ‖

Intuitively, each w is covered with respect to f by $\widetilde{w}$.

Proposition 8.3 *The constraint transformer monoid Y obtained by trimming is finite.*

Proof. Follows from Lemma 3.5. ‖

Proposition 8.4 *Any f-quotient of X' of X covers the constraint transformer monoid Y obtained by trimming with f^{-1} witnessing the covering.*

Proof. The proof is similar to that of Proposition 3.2. ‖

We call a constraint φ *bounded* if $\varphi \wedge \bigwedge_{i=1}^{n} x_i \leq M = \varphi$. It can be easily verified that any bounded constraint $\varphi \in \Phi$ is canonical with respect to f. Also for each bounded constraint $\varphi \in \Phi$, f saturates φ.

Theorem 8.2 *For any constraint φ, abstract symbolic forward analysis of a timed automaton $\mathcal{T}$ with respect to the constraint transformer monoid Y obtained by trimming, f and φ terminates. Moreover, if φ, φ' is are bounded constraints, then φ' is reachable from φ in $\mathcal{T}$ iff $f(\varphi')$ is reachable in the abstract symbolic forward analysis of $\mathcal{T}$ with respect Y, f and φ.*

Proof. Follows from Propositions 8.1, 8.2, 8.3, 8.4 and Theorem 8.1. ‖

Note that the constraint transformer monoid Y above is never constructed explicitly. Rather, it is constructed on-the-fly.

8.7.2 The Two-process Bakery Algorithm

The bakery algorithm implements a mutual exclusion protocol. The guarded commands for the two-process bakery algorithm are given in Figure 8.1. We say that the two process bakery algorithm is safe if no state of the form $L = \langle use, use \rangle \wedge \psi$ is reachable from the initial state. Let $X = \langle \Phi, S \rangle$ be the constraint transformer monoid generated by the two-process bakery algorithm. We define the covering monoid, called the abstract target monoid, as follows.

Definition 8.8 (Abstract target monoid) *Given the two-process bakery algorithm, the abstract target monoid Y is defined as $Y = \langle \Phi', S' \rangle$ where $\Phi' = \{\varphi_1, \ldots, \varphi_{10}\}$[2] and $S' = \{\widetilde{w} \,|\, [\![w]\!] \in S\}$ where the constraints $\varphi_1, \ldots, \varphi_{10}$ are defined in Figure 8.2.*

Here $\widetilde{w}(\varphi_i) = \varphi_j$ if there exists $\psi, \psi' \in \Phi$ such that $[\![w]\!](\psi) = \psi'$ and $\psi \models \varphi_i$ and $\psi' \models \varphi_j$. It can be easily verified that each $\widetilde{w} \in S'$ is a function from Φ' to Φ'. Define the relation f from Φ to Φ' as $f(\varphi, \varphi')$ iff $\varphi \models \varphi'$. Note that f is a function in this case.

Proposition 8.5 *The abstract target monoid Y covers the constraint transformer monoid X (generated by the two-process bakery algorithm) with the mapping f witnessing the covering.*

Proof. Follows from the definitions of S' and f. ‖

Each w is covered with respect to f by $\widetilde{w}$.

Proposition 8.6 *Any f-quotient of X covers the abstract target monoid Y with f^{-1} witnessing the covering.*

Proof. Consider any $\varphi_i \in \Phi'$. Let $\psi = rep(\{\varphi \in \Phi \,|\, \varphi \models \varphi_i\})$ where rep is a chosen representant function for a quotient. Consider $\widetilde{w} \in S'$. We claim that $[\![w]\!] \in S$ covers $\widetilde{w}$ with respect to f^{-1}. Suppose that $\widetilde{w}(\varphi_i) = \varphi_j$. It can be verified that $[\![w]\!](\psi) \models \varphi_j$. Therefore, in the f-quotient with the representant function rep, $\widetilde{[\![w]\!]}(\psi) = rep([[\![w]\!](\psi)]) = \psi'$. Therefore, by definition, $\psi' \models \varphi_j$. Therefore $f(\psi', \varphi_j)$. ‖

[2]These constraints are obtained by a simple inspection of the guards and the actions of the composed transition system.

Control variables: p_1, p_2 varying on $\{think, wait, use\}$
Data variables: a_1, $a_2 \geq 0$.
Initial condition: $p_1 = think \wedge p_2 = think \wedge a_1 = a_2 = 0$
Transitions for $i, j : 1, 2$, $i \neq j$**:**

τ_{t_i} :	:	$p_i = think$	$\|\|$	$p_i' = wait \wedge a_i' = a_j + 1$
τ_{w_i} :	:	$p_i = wait \wedge a_i < a_j$	$\|\|$	$p_i' = use$
$\tau_{w_i'}$:	:	$p_i = wait \wedge a_j = 0$	$\|\|$	$p_i' = use$
τ_{u_i} :	:	$p_i = use$	$\|\|$	$p_i' = wait \wedge a_i' = 0$

Figure 8.1: The bakery algorithm

Theorem 8.3 *For any constraint φ, any abstract symbolic forward analysis of the two-process bakery algorithm with respect to φ,the abstract monoid Y and f terminates. Moreover, for any two constraints φ and φ' such that f saturates both φ and φ', φ' is reachable from φ, iff φ_j, such that $f(\varphi', \varphi_j)$, is reachable from φ_i, such that $f(\varphi, \varphi_i)$, in an abstract symbolic forward analysis wrt φ, the abstract target monoid Y and f. In particular, the two-process bakery algorithm is safe iff the constraint $L = \langle use, use\rangle \wedge a_1 \geq 0 \wedge a_2 \geq 0$ is reachable in the abstract symbolic forward analysis with respect to $L = \langle think, think\rangle \wedge a_1 = 0 \wedge a_2 = 0$, f and Y.*

Proof. Follows from Propositions 8.1, 8.5, 8.6 and Theorem 8.1. $\|$

Symbolic forward analysis for the bakery algorithm with respect to the initial constraint $L = \langle think, think\rangle \wedge a_1 = 0, a_2 = 0$ is (possibly) nonterminating. To the best of the knowledge of the authors, this is the first time that reachability properties for the two-process bakery algorithm have been shown to be verifiable using a terminating abstract symbolic forward analysis. Previous approaches were based either on symbolic backward analysis [BGP97, DP99a] or on deductive methods [BBM97, KPV99]. While model checking using symbolic backward analysis is inherently global model checking [HKQ98], model checking by symbolic forward analysis can be made local.

$$
\begin{array}{lcl}
\varphi_1 & \equiv & L = \langle think, think\rangle \wedge a_1 = 0 \wedge a_2 = 0 \\
\varphi_2 & \equiv & L = \langle wait, think\rangle \wedge a_1 \geq 0 \wedge a_2 = 0 \\
\varphi_3 & \equiv & L = \langle think, use\rangle \wedge a_1 = 0 \wedge a_2 \geq 0 \\
\varphi_4 & \equiv & L = \langle use, think\rangle \wedge a_1 \geq 0 \wedge a_2 = 0 \\
\varphi_5 & \equiv & L = \langle wait, wait\rangle \wedge a_1 = a_2 + 1 \wedge a_2 \geq 1 \\
\varphi_6 & \equiv & L = \langle wait, wait\rangle \wedge a_2 = a_1 + 1 \wedge a_1 \geq 1 \\
\varphi_7 & \equiv & L = \langle use, wait\rangle \wedge a_2 = a_1 + 1 \wedge a_2 \geq 1 \\
\varphi_8 & \equiv & L = \langle think, wait\rangle \wedge a_1 = 0 \wedge a_2 \geq 0 \\
\varphi_9 & \equiv & L = \langle wait, use\rangle \wedge a_1 \geq 1 \wedge a_1 = a_2 + 1 \\
\varphi_{10} & \equiv & L = \langle use, use\rangle \wedge a_1 \geq 0 \wedge a_2 \geq 0
\end{array}
$$

Figure 8.2: Constraints in Φ'

8.8 Summary and Related Work

We have presented a new algebraic theory for abstract symbolic forward analysis. Our framework is well suited to constraint based symbolic model checking of infinite state systems. Our framework provides sufficient conditions under which the abstract symbolic forward analysis is always terminating or accurate or both. As in the classical abstract interpretation framework [CC77] one has to establish a Galois connection from the concrete lattice to the abstract lattice more or less manually, in our framework one has to establish a covering manually. Note that the covering constraint transformer monoid can be arbitrary (i.e., may not be finitely generated). Also note that the sufficient termination conditions in our framework do not require the covering constraint transformer to be finite. Also the termination guarantees continue to hold even when the fixpoint test is weakened to local entailment.

Colon and Uribe [CU98] present an algorithm that uses decision procedures to generate finite state abstractions of possibly infinite state systems. Our work is different from theirs; the denotation of the covering transformer monoid $Y = \langle \Phi', S' \rangle$ (i.e., $[\Phi']$) may be infinite; moreover Φ' may itself be infinite. In [CC98], Cousot and Cousot describe improvements to abstract model checking by combining forwards and backwards abstract fixpoint computations. It would be interesting to see how their techniques can be adapted to a constraint-based setting as ours. Cleaveland, Iyer and Yankelevich [CIY95] develop a framework in which they can establish optimality results by showing that a particular system abstraction is the most precise one possible among a class of safe abstractions. It is not clear how to apply their techniques in a constraint-based setting. An automata-theoretic framework for verification by finitary abstraction has been developed in [KPV99]. There, the authors reduce the verification problem to the infeasibility problem for Büchi discrete systems. They then provide a general proof method called WELL to establish the infeasibility of a Büchi discrete system. In contrast, our technique uses abstract symbolic forward analysis for verification after a covering has been established.

Chapter 9

Conclusions

9.1 Summary

In this dissertation, we have described a uniform constraint-based framework for the verification of possibly infinite state reactive systems. Constraint query languages provide a framework for representing reactive systems as well as for specifying their properties. Many of the seemingly different formalisms for representing reactive systems have a natural translation into this framework. The model checking problem reduces to computing (or checking membership in) the model-theoretic semantics of constraint query languages. We have provided several optimized methods for computing model theoretic semantics of constraint query languages. The product construction for timed logic processes introduced in this dissertation allowed us to extend the methodology to deal with more expressive logics. Several existing model checking procedures can be obtained as special cases of the model checking procedures that we obtained in our constraint-based framework. A prototype implementation based on the methodology developed in this dissertation has shown encouraging results. We have also been able to identify a logic that can be model checked efficiently in practice within our framework. Our framework has also been used to solve control-theoretic problems e.g., detection of transient behavior in linear time-invariant systems.

The two main currents that have run through this dissertation are logic and constraints. The constraint-based setting has enabled us to reason about the termination of the symbolic model checking procedures that solve the verification problem for infinite state systems in practice. We have obtained sufficient termination conditions for these procedures even with a weaker but more efficient fixpoint test. We have shown several examples for which the termination of symbolic forward analysis can be explained by using our sufficient termination conditions. Much of this reasoning has also been compositional. Since the combinatorial (constraint solving) part is clearly separated from the logical part, we could easily extend our methodology to deal with nonlinear systems. Moreover, we have been able to reason about the accuracy of constraint-based abstractions introduced to solve the verification problem in practice.

9.2 Future Work

We end our discussion by addressing some of the future research issues. One obvious research issue is to try to use our framework to verify larger examples. More experimentation is needed

in this direction. In this dissertation, the focus was mainly on infinite state systems in which the variables range over a (possibly infinite) numeric data domain. But our methodology can be easily adapted to model and verify out-of-order execution in the design of microprocessors. In this case, the relevant constraint domain is the Herbrand one. A similar line of work would be to consider many-sorted systems in which in which some variables range a numeric data-domain while others range over the domain of possibly infinite trees. Another line of research is to use our framework for analysis of programs written in programming languages like C or Java. It would also be interesting to see how our framework can be extended to deal with mobility of processes.

We state below some of the other problems left open in this dissertation.

- Extend the product construction defined in Chapter 3 to decide whether two timed logic proceseses are timed bisimilar.
- Is the following problem decidable— given a timed automaton, does symbolic forward analysis for it terminate?
- Can one design an algorithm for deciding whether a timed logic process has a transient behavior that is more efficient than the one presented in Chapter 3?
- Can one come up with a tableau based model checking procedure for timed systems in the style of Bradfield and Stirling [BS90]?

Bibliography

[ABW88] K. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–214. Morgan Kaufmann, 1988.

[ACD+92] R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. Minimization of timed transition systems. In R. Cleaveland, editor, *CONCUR: Concurrency Theory*, volume 630 of *LNCS*, pages 340–354. Springer-Verlag, 1992.

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real time. *Information and Computation*, 104(2):2–34, 1993.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems I*, LNCS 736, pages 209–229. Springer-Verlag, 1993.

[ACJT96] P. Abdulla, K. Cerans, B. Jonsson, and T. K. Tsay. General decidability theorems for infinite state systems. In *LICS*, pages 313–321, 1996.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–236, 1994.

[AH97] R. Alur and T. A. Henzinger. Modularity for timed and hybrid systems. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR'97: Concurrency Theory*, volume 1243 of *LNCS*, pages 74–88. Springer-Verlag, 1997.

[AH99] R. Alur and T. A. Henzinger. Computer-aided verification: An introduction to model building and model checking for concurrent systems, 1999. Book in preparation.

[ATEP99] R. Alur, S. La Torre, K. Etessami, and D. Peled. Parameteric temporal logic model measuring. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP: Automata, Languages and Programming*, volume 1644 of *LNCS*, pages 159–168. Springer, 1999.

[AY98] R. Alur and M. Yannakakis. Model checking hierarchical state machines. In *ACM Symposium on the Foundations of Software Engineering*, 1998.

[Bal96] F. Balarin. Approximate reachability analysis of timed automata. In *17th IEEE Real-Time Systems Symposium*, pages 52–61. IEEE Computer Society Press, 1996.

[BBC+96] N. Bjorner, A. Browne, E. Chang, M. Colon, A. Kapur, Z. Manna, H. Sipma, and T. Uribe. Step: Deductive-algorithmic verification of reactive and real-time systems. In R. Alur and T. A. Henzinger, editors, *CAV'96: Computer Aided Verification*, volume 1102 of *LNCS*, pages 415–418. Springer-Verlag, 1996.

[BBM97] N. Bjorner, A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, 173(1):49–87, 1997.

[BBR97] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems. In O. Grumberg, editor, *CAV'97: Computer Aided Verification*, volume 1254 of *LNCS*, pages 167–178. Springer-Verlag, 1997.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *CONCUR'97*. LNCS 1243, 1997.

[BF99] B. Berard' and L. Fribourg. Reachability analysis of (timed) petri nets using real arithmetic. In J. C. M. Baeten and S. Mauw, editors, *CONCUR: Concurrency Theory*, volume 1664 of *LNCS*, pages 178–193. Springer-Verlag, 1999.

[BFH91] A. Bouajjani, J.C. Fernandez, and N. Halbwachs. Minimal model generation. In *Computer-Aided Verification '90*, volume 3 of *DIMACS*, 1991.

[BGP97] T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using presburger arithmetics. In Orna Grumberg, editor, *the 9th International Conference on Computer Aided Verification (CAV'97)*, LNCS 1254, pages 400–411. Springer, Haifa, Israel, July 1997.

[BGP98] T. Bultan, R. Gerber, and W. Pugh. Model Checking Concurrent Systems with Unbounded Integer Variables: Symbolic Representations, Approximations and Experimental Results, february 1998.

[BLL+96] Johan Bengtsson, Kim. G. Larsen, Fredrik Larsson, Paul Petersson, and Wang Yi. Uppaal in 1995. In T. Margaria and B. Steffen, editors, *TACAS*, LNCS 1055, pages 431–434. Springer-Verlag, 1996.

[BMSU86] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *PODS: ACM Symposium on Principles of Database Systems*, pages 1–16. ACM, 1986.

[Boi98] Bernard Boigelot. *Symbolic Methods for Exploring Infinite State Spaces.* PhD thesis, Universite De Liege, Montefiore, Belgium, 1998.

[BS90] J. Bradfield and C. Stirling. Verifying temporal properties of processes. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR: Concurrency Theory*, volume 458 of *LNCS*, pages 115–125. Springer, 1990.

[BS91] A. Brodsky and Y. Sagiv. Inference of inequality constraints in logic programs. In *PODS: Principles of Database Systems*, pages 227–240. ACM Press, 1991.

[BVW94] Orna Bernholtz, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, volume 818 of *LNCS*, pages 142–155, Stanford, California, June 1994. Springer-Verlag. Full version available from authors.

[BW94] Bernard Boigelot and Pierre Wolper. Symbolic verification with periodic sets. In David Dill, editor, *6th International Conference on Computer-Aided Verification*, volume 818 of *LNCS*, pages 55–67. Springer-Verlag, June 1994.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *the 4th ACM Symposium on Principles of Programming Languages*, 1977.

[CC98] P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Automated Software Engineering*, 6:69–95, 1998.

[CDD+98] B. Cui, Y. Dong, X. Du, K. N. Kumar, C. R. Ramakrishnan, I. V. Ramakrishnan, A. Roychoudhury, S. A. Smolka, and D. S. Warren. Logic programming and model checking. In *PLAP/ALP98*, volume 1490 of *LNCS*, pages 1–20. Springer-Verlag, 1998.

[CE80] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1980.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *the Fifth Annual ACM Symposium on Principles of Programming Languages*. ACM Press, 1978.

[CIY95] R. Cleaveland, P. Iyer, and D. Yankelevich. Optimality in abstractions of model checking. In A. Mycroft, editor, *SAS: Static Analysis Symposium*, volume 983 of *LNCS*, pages 51–63. Springer, 1995.

[CJ98] H. Comon and Y. Jurski. Multiple Counters Automata, Safety Analysis, and Presburger Arithmetics. In Alan J. Hu and M. Y. Vardi, editors, *CAV'98: Computer Aided Verification*, volume 1427 of *LNCS*, pages 268–279. Springer-Verlag, 1998.

[CMN+98] W. Charatonik, D. McAllester, D. Niwinski, A.Podelski, and I. Walukiewicz. The Horn mu-calculus. In Vaughan Pratt, editor, *The 13th IEEE Annual Symposium on Logic in Computer Science*, 1998.

[CP98a] W. Charatonik and A. Podelski. Set-based analysis of reactive infinite-state systems. In Bernhard Steffen, editor, *the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 358–375. Springer-Verlag, March-April 1998.

[CP98b] Witold Charatonik and Andreas Podelski. Set-based analysis of reactive infinite-state systems. In Bernhard Steffen, editor, *Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *LNCS*, pages 358–375, Lisbon, Portugal, March-April 1998. Springer-Verlag.

[CU98] M. A. Colon and T. E. Uribe. Generating finite-state abstractions of reactive systems using decision procedures. In A. Hu and M. Y. Vardi, editors, *CAV: Computer-Aided Verification*, volume 1427 of *LNCS*, pages 293–304. Springer, 1998.

[CW96] W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *JACM*, 43(1):20–74, 1996.

[DG84] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of horn formulas. *The Journal of Logic Programming*, 1(3):267–284, 1984.

[DP99a] G. Delzanno and A. Podelski. Model Checking in CLP. In R. Cleaveland, editor, *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 223–239. Springer-Verlag, March 1999.

[DP99b] Giorgio Delzanno and Andreas Podelski. Model Checking in CLP. In Rance Cleaveland, editor, *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 223–239, Amsterdam, The Netherlands, January 1999. Springer-Verlag.

[DRS99] X. Du, C. R. Ramakrishnan, and S. A. Smolka. Tabled resolution + constraints: A recipe for model checking real-time systems, 1999. Submitted.

[DT98] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In Bernhard Steffen, editor, *TACAS98: Tools and Algorithms for the Construction of Systems*, LNCS 1384, pages 313–329. Springer-Verlag, March/April 1998.

[DW99] M. Dickhöfer and T. Wilke. Timed alternating tree automata: The automata-theoretic solution to the tctl model checking problem. In J. Widermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP: Automata, Languages and Programming*, volume 1644 of *LNCS*, pages 281–290. Springer-Verlag, 1999.

[Eil76] S. Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[Flo67] R. W. Floyd. Assigning meaning to programs. In *Proceedings Symposium on Applied Mathematics*, volume 19, 1967.

[FO97] L. Fribourg and H. Olsen. A Decompositional Approach for Computing Least Fixed Point of Datalog Programs with Z-counters. *Journal of Constraints*, 2(3-4):305–336, 1997.

[FP93] Laurent Fribourg and Marcos Veloso Peixoto. Concurrent constraint automata. Technical Report LIENS 93-10, ENS Paris, 1993.

[FR96] L. Fribourg and J. Richardson. Symbolic verification with gap-order constraints. In J. P. Gallagher, editor, *LOPSTR'96: Logic Based Program Synthesis and Transformation*, volume 1207 of *LNCS*, pages 20–37. Springer-Verlag, 1996.

[Fri98] Laurent Fribourg. A closed-form evaluation for extended timed automata. Technical report, ENS Cachan, 1998.

[FS98] A. Finkel and P. Schnoebelen. Well-structured Transition Systems Everywhere! Technical Report LSV-98-4, Laboratoire Spécification et Vérification, Ecole Normale Supérieure de Cachan, 1998.

[FV98] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal of Computing*, 28(1):57–104, 1998.

[GGV98] Georg Gottlob, Erich Grädel, and Helmut Veith. Datalog LITE: Temporal versus deductive reasoning in verification. Technical Report DBAI-TR-98-22, Institut für Informationssysteme, Technische Universität Wien, December 1998.

[GGV99] G. Gottlob, E. Grädel, and H. Veith. Datalog lite: A deductive approach to verification. Technical report, Technische Universität Wien, 1999.

[GP97] G. Gupta and E. Pontelli. A constraint-based approach for the specification and verification of real-time systems. In Kwei-Jay Lin, editor, *IEEE Real-Time Systems Symposium*, pages 230–239. IEEE Press, 1997.

[GP99] Gopal Gupta and Enrico Pontelli. A horn logic denotational framework for specification, implementation, and verification of domain specific languages, March 1999.

[Gup99] Gopal Gupta. Horn logic denotations and their applications. In *The Logic Programming Paradigm: A 25 year perspective*. Springer-Verlag, 1999.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In C. Courcoubetis, editor, *the International Conference on Computer-Aided-Verification*, volume 697 of *LNCS*, pages 333–346. Springer-Verlag, 1993.

[Hen95] T.A. Henzinger. Hybrid automata with finite bisimulations. In Z. Fülöp and F. Gécseg, editors, *ICALP 95: Automata, Languages, and Programming*, LNCS 944, pages 324–335. Springer-Verlag, 1995.

[HH95] T. A. Henzinger and P.-H. Ho. A note on abstract-interpretation strategies for hybrid automata. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid Systems II*, LNCS 999, pages 252–264. Springer-Verlag, 1995.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1019, pages 41–71. Springer-Verlag, 1995.

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. In O. Grumberg, editor, *CAV97: Computer-aided Verification*, LNCS 1254, pages 460–463. Springer-Verlag, 1997.

[HK97] Thomas. A. Henzinger and Orna Kupferman. From quantity to quality. In Oded Maler, editor, *Hybrid and Real-Time Systems International Workshop,Hart '97*, volume 1201 of *LNCS*, pages 48–62, Grenoble, France, March 1997. Springer-Verlag.

[HKPV95] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *the 27th Annual Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995.

[HKQ98] T. A. Henzinger, O. Kupferman, and S. Qadeer. From pre-historic to post-modern symbolic model checking. In A. J. Hu and M. Y. Vardi, editors, *CAV'98: Computer-aided Verification*, LNCS 1427, pages 195–206. Springer-Verlag, 1998.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994. Special issue for LICS 92.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 10:576–580, 1969.

[HPR97] N. Halbwachs, Y-E. Proy, and P. Romanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.

[HU79] J. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

[HWT95] Pei-Hsin Ho and Howard Wong-Toi. Automated analysis of an audio control protocol. In P. Wolper, editor, *the Seventh Conference on Computer-Aided Verification*, pages 381–394, Liege, Belgium, 1995. Springer-Verlag. LNCS 939.

[JM94] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *The Journal of Logic Programming*, 19/20:503–582, May-July 1994.

[JMMS] J. Jaffar, M. Maher, K. Marriot, and P. Stuckey. The semantics of constraint logic programs. *J. Logic Programming.* To appear.

[JMSY92] J. Jaffar, S. Michaylov, P.J. Stuckey, and R.H.C. Yap. The clp(r) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[KKR95] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51:26–52, 1995. (Preliminary version in *Proc. 9th ACM PODS*, 299–313, 1990.).

[KMP96] Y. Kesten, Z. Manna, and A. Pnueli. Verifying clocked transition systems. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *LNCS*, pages 13–40. Springer-Verlag, 1996.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[KPV99] Y. Kesten, A. Pnueli, and M. Y. Vardi. Verification by augmented abstraction: The automata-theoretic view. In J. Flum and M. R. Artalejo, editors, *CSL: Computer Science Logic*, volume 1683 of *LNCS*, pages 141–156. Springer, 1999.

[Kur66] K. Kuratowski. *Topology.* Academic Press, 1966.

[LB93] W. K. C. Lam and R. K. Brayton. Alternating RQ timed automata. In Costas Courcoubetis, editor, *the 5th International Conference on Computer-Aided Verification*, LNCS 697, pages 236–252. Springer-Verlag, June/July 1993.

[Lib00] L. Libkin. Variable independence, quantifier elimination, and constraint representations. In *ICALP: International Colloqium on Automata Languages and Programming*, 2000.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Symbolic Computation. second, extended edition, 1987.

[LM92] J-L. Lassez and M. J. Maher. On fourier's algorithm for linear arithmetic constraints. *Journal of Automated Reasoning*, 9(3), December 1992.

[LPY95a] K. G. Larsen, P. Peterson, and W. Yi. Model-checking for real-time systems. In Horst Reichel, editor, *Fundamentals of Computation Theory*, volume 965 of *LNCS*, pages 62–88. Springer-Verlag, 1995.

[LPY95b] K.G. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model checking of real-time systems. In *Proceedings of the 16th Annual Real-time Systems Symposium*, pages 76–87. IEEE Computer Society Press, 1995.

[LPY99] G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In F. W. Vaandrager and J. H. van Schuppen, editors, *Hybrid Systems, Computation and Control*, volume 1569 of *LNCS*, pages 137–151, 1999.

[LS85] N. G. Leveson and J. L. Stolzy. Analyzing safety and fault tolerance using time petri nets. In H. Uhrig, C. Floyd, M. Nivat, and J. W. Thatcher, editors, *TAPSOFT: Theory and Practice of Software*, volume 186 of *LNCS*, pages 339–355. Springer, 1985.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[MP99] S. Mukhopadhyay and A. Podelski. Beyond region graphs: Symbolic forward analysis of timed automata. In C. Pandurangan, V. Raman, and R. Ramanujam, editors, *19th International Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *LNCS*, pages 233–245, December 1999.

[MP00a] S. Mukhopadhyay and A. Podelski. Accurate widenings and boundedness properties, 2000.

[MP00b] S. Mukhopadhyay and A. Podelski. Model checking for timed logic processes. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, editors, *CL: Computational Logic*, LNCS, pages 598–612. Springer, 2000. Available at http://www.mpi-sb.mpg.de/~supratik/.

[MS87] D.E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[MS98] K. Marriott and P. J. Stuckey. *Programming with Constraints: An Introduction.* MIT Press, 1998.

[Nam98] K. S. Namjoshi. *Ameliorating the State Explosion Problem.* PhD thesis, The Graduate School of the University of Texas at Austin, 1998.

[OG76] S. S. Owicki and D. Gries. An axiomatic proof technique for parallel programs i. *Acta Informatica*, 6:319–340, 1976.

[Oga96] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall, 1996.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[Pod00] Andreas Podelski. Model checking as constraint solving. In Jens Palsberg, editor, *Proceedings of SAS'2000: Static Analysis Symposium*, LNCS, pages 22–37, Berlin, Germany, 2000. Springer-Verlag.

[Prz88] T. Przymusinski. On the semantics of stratified deductive databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1988.

[Pug92] W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:102–104, 1992.

[QS81] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the fifth International Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer, 1981.

[Ram91] R. Ramakrishnan. Magic templates: A spellbinding approach to logic programs. *Journal of Logic Programming*, 11(34):189–216, 1991.

[Rev90] Peter Revesz. A closed form for datalog queries with integer order. In S. Abiteboul and P. C‘. Kanellakis, editors, *ICDT: the International Conference on Database Theory*, volume 470 of *LNCS*, pages 187–201. Springer-Verlag, 1990.

[RKR+00] A. Roychoudhury, K. N. Kumar, C. R. Ramakrishnan, I. V. Ramakrishnan, and S. A. Smolka. Verification of parameterized systems using logic program transformations. In *TACAS'00: Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *LNCS*, pages 172–187. Springer, 2000.

[RRR+97a] Y. S. Ramakrishna, C. R. Ramakrishnan, I. V Ramakrishnan, S. A. Smolka, T. W. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In O. Grumberg, editor, *the 9th International Conference on Computer-Aided-Verification*, pages 143–154. Springer-Verlag, July 1997.

[RRR+97b] Y.S. Ramakrishna, C.R. Ramakrishnan, I.V. Ramakrishnan, S.A. Smolka, T. Swift, and D.S. Warren. Efficient model checking using tabled resolution. In *Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*. Springer-Verlag, June 1997.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 319–327. IEEE Computer Society Press, 1988.

[SI88] H. Seki and H. Ito. A query evaluation method for stratified programs under the extended cwa. In *Proceedings of the fifth International Conference and Symposium on Logic Programming*, 1988.

[SIR96] S. K. Shukla, H. B. Hunt III, and D. J. Rosenkrantz. Hornsat, model checking, verification and games. In R. Alur and T. A. Henzinger, editors, *CAV'96: Computer Aided Verification*, LNCS 1102, pages 99–110. Springer-Verlag, 1996.

[Sri92] D. Srivistava. Subsumption and indexing in constraint query languages with linear arithmetic constraints. In *2nd International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale*, 1992.

[SS95] Oleg Sokolsky and Scott. A. Smolka. Local model checking for real-time systems. In Pierre Wolper, editor, *7th International Conference on Computer-Aided Verification*, volume 939 of *LNCS*, pages 211–224. Springer-Verlag, July 1995.

[Stu91] Peter J. Stuckey. Constructive negation for constraint logic programming. In *Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 328–339. IEEE Computer Society Press, 1991.

[SW91] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers (North-Holland), 1990.

[Tri99] S. Tripakis. Verifying progress in timed systems. In J. P. Katoen, editor, *Formal Methods in Real time and Probablistic Systems: 5th International AMAST Workshop (ARTS99)*, volume 1601 of *LNCS*, pages 299–314. Springer-Verlag, 1999.

[TS86a] H. Tamaki and T. Sato. Old resolution with tabulation. In *International Conference on Logic Programming*, LNCS, pages 84–98. Springer-Verlag, 1986.

[TS86b] Hisao Tamaki and Taisuke Sato. OLD resolution with tabulation. In Ehud Shapiro, editor, *Proceedings of the Third International Conference on Logic Programming*, volume 225 of *LNCS*, pages 84–98, London, 1986. Springer-Verlag.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-base Systems*, volume II. Computer Science Press, 1989.

[Urb96] L. Urbina. Analysis of hybrid systems in clp(r). In *Principles and Practice of Constraint Programming, CP96*, Lectures Notes in Computer Science 1118, pages 451–467. Springer-Verlag, 1996.

[vdD98] L. van den Dries. *Tame Topology and o-minimal structures.* Cambridge University Press, 1998.

[Vie87] L. Vielle. A database-complete proof procedure based on sld-resolution. In *Fourth International Conference on Logic Programming.* MIT Press, 1987.

[VW86a] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *LICS: Logic in Computer Science*, pages 332–344, Cambridge, Massachusetts, 1986. IEEE Computer Society.

[VW86b] Moshe. Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *the first IEEE Symposium on Logic in Computer Science*, 1986.

[Wal93] I. Walukiewicz. *A Complete Deductive System for the μ-Calculus.* PhD thesis, Warsaw University, 1993.

[Wal96] I. Walukiewicz. Pushdown processes: Games and model checking. In R. Alur and T. A. Henzinger, editors, *CAV: Computer Aided Verification, 8th International Conference*, volume 1102 of *LNCS*, pages 62–74. Springer, 1996.

[Wei94] V. Weispfenning. Parametric linear and quadratic optimization by elimination. Technical report, Universität Passau, 1994.

[WT95] H. Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems.* PhD thesis, Stanford University, 1995.

[YL93] Mihalis Yannakakis and David Lee. An efficient algorithm for minimizing real-time transition systems. In Costas Courcoubetis, editor, *Computer-Aided Verification– The 5th International Conference*, LNCS, pages 210–224. Springer, 1993.

Index

t-clause, 33
'pop', 18

advancing derivation, 60

active, 78
all, 72
alternating clauses, 33

backward, 81
base sets, 68
bounded, 145

canonical, 140
classify, 73, 78, 79
clock constraints, 31
compact, 25
companion, 141
complement, 65
complete, 25, 71
compositional, 117
conjunction, 65
conjunctive, 138
constant, 92
constraint transformer monoids, 138
constraint transformer semigroup, 138
constraint transformers, 138
constraints, 83
Convergence, 66
convergence, 77
convergent, 34, 35
covering, 139
coverings, 138
covers, 139
cut condition, 90
cycles, 86

denotation, 13
disconnected, 25
disjunction, 65
divergent, 35

edge transition, 83
Every, 86
every, 85
evolution clause, 33
Example, 52
extend, 71, 78, 79
extendible, 74

finitary, 139
finitary covering, 140
finite representation, 142
finitely representable, 10
finitely represented, 9
Forward analysis, 66

global, 85
greatest model resolution, 30
guards, 33

homeocovering, 140

increment variables, 33
independence property, 104
initial clauses, 33
initial constraint, 83
initializable, 123

labeled TLP, 60
labels, 91
last, 130
leaf, 85
least-fixpoint closure, 68
local entailment, 117, 138
local inclusion abstraction, 102
locations, 82
Logic of safety and bounded liveness, 35
lookup, 78

lookup node, 13

metric space, 25
monolithic, 16

necessary, 94
not, 90

o-minimal, 117
open, 25

part of a cycle, 123
path, 86
perfect model, 70
perfect models, 66
positions, 83
Proof, 69

real variables, 36
region graph, 81
region product graph, 30
remainder section, 130
reset, 82
reset-free, 84

saturates, 140
segment, 91
simple path, 90
solution node, 13
solution table, 13
splitting constraints, 30
stratifiable, 67, 87
stratification, 67
stratified, 65, 69, 91–94
strings, 86
symbolic forward analysis, 81
symbolic states, 138
syntactic monoid, 95
syntactic order, 138
syntactically increasing, 139
system clause, 33

table node, 13
table predicates, 13
Tabled resolution, 66
tabled-resolution, 66
tabulate, 71, 73, 78, 79
terminates, 85
The connection, 66
time (seconds), 52
time closed, 120
time transition, 83, 119
time-closed, 83
timed automaton, 81, 82
timed logic processes, 29, 33
totally bounded, 25

unlabeled, 9

with local subsumption, 85
without, 67

zone constraint, 83
zone constraints, 41
zone trees, 82
zones, 83