LfSA 2012
Logics for System Analysis

Workshop affiliated with
CAV, Berkeley, 7. July 2012

André Platzer and Philipp Rümmer (Chairs)
Preface

Safety-critical systems occur in various different forms like real-time systems, embedded systems, hybrid systems, distributed systems, and cyber-physical systems. They are becoming more and more important in application domains, including aviation, automotive, railway, robotic, or medical applications, where both safety and security are relevant aspects. To ensure the correct functioning of safety-critical systems, it is necessary to model and verify aspects of hardware (including physical properties or movement), software, communication, and qualitative and quantitative aspects of the system environment.

Logics for system analysis, system modeling, and specification, are primary tools to analyze system behavior. Logic is equally important for understanding the theoretical foundations of system analysis and verification and serves as the basis for practical analysis tools that establish correct functioning of systems or find bugs in their designs. Depending on the nature of the system, modeling languages that are amenable to logical analysis and the study of correctness properties could include logical representations, automata, state charts, Petri nets, dataflow models, or systems of differential equations. Several system models can be analyzed rigorously with the help of techniques like logical calculi, decision procedures, model checking, and abstraction.

In light of these developments it is high time to start a forum for exchange of ideas and collaboration on logic for system analysis. The workshop LfSA is devoted to the systematic theoretical study, practical development, and applied use of logics for system analysis and verification. The purpose of the LfSA workshop is to bring together researchers and practitioners interested in studying practically relevant systems or in developing the logical foundations and analysis tools for their study.

This volume contains the research papers and abstracts of invited and contributed talks presented at the Second Workshop on Logics for System Analysis (LfSA’12) held at July 7, 2012 in Berkeley, USA. LfSA’12 was held in affiliation with CAV 2012. Each paper submitted the workshop was reviewed by four referees (two for contributed presentation-only talks), and a discussion on the papers was held during the Programme Committee meeting periods. We would like to thank the Programme Committee members for their effort and professional work in the reviewing process and the paper selection.
LfSA’12 had a rich program with two invited talks, two talks presenting contributed papers, and three presentation-only talks. The LfSA workshop is proud to feature special invited talks by Radu Grosu from the Vienna University of Technology on “Time-Frequency Logic For Signal Processing,” and by Ashish Tiwari from SRI International on “Verifying Safety of Hybrid Systems.” We are grateful to Radu Grosu and Ashish Tiwari for accepting the invitation to address the second LfSA workshop.

July 2012

André Platzer and Philipp Rümmer
Programme Chairs
LfSA’12
Organization

LfSA’12 is the second workshop on Logics for System Analysis.
LfSA is affiliated with CAV 2012.

Programme Chairs

André Platzer Carnegie Mellon University (Pittsburgh, USA)
Philipp Rümmer Uppsala University (Sweden)

Programme Committee

Anindya Banerjee IMDEA (Madrid, Spain)
Raoul Barbosa University of Coimbra (Portugal)
Frank S. de Boer CWI (Amsterdam, Netherlands)
Alessandro Cimatti FBK/IRST (Trento, Italy)
Mads Dam KTH (Stockholm, Sweden)
Stéphane Demri CNRS (Cachan, France)
Martin Giese University of Oslo (Norway)
Ichiro Hasuo University of Tokyo (Japan)
Franjo Ivančić NEC Laboratories (Princeton, USA)
Einar Broch Johnsen University of Oslo (Norway)
Viorica Sofronie-Stokkermans University of Koblenz-Landau (Germany)
Uwe Waldmann MPI Saarbrücken (Germany)
# Table of Contents

**9:00–10:00 Invited Talk**

Time-Frequency Logic For Signal Processing ............................. 1  
*Radu Grosu*

**10:30–12:00 Morning Session**

HydLa: A High-Level Language for Hybrid Systems ....................... 3  
*Kazunori Ueda, Shota Matsumoto, Akira Takeyachi, Hiroshi Hosobe,  
Daisuke Ishii*

Interpolation-based Function Summaries in Bounded Model Checking ... 19  
*Ondrej Sery, Grigory Fedyukovich, and Natasha Sharygina*

Abstract Conflict Driven Clause Learning .............................. 21  
*Vijay D’Silva, Leopold Haller, and Daniel Kroening*

**14:00–15:00 Invited Talk**

Verifying Safety of Hybrid Systems .................................. 23  
*Ashish Tiwari*

**15:30–16:30 Afternoon Session**

HySon: Precise Simulation of Hybrid Systems with Imprecise Inputs.... 25  
*Olivier Bouissou, Samuel Mimram, and Alexandre Chapoutot*

Decidability and Completeness of PDL through Canonical Model ........ 27  
*Xinxin Liu, Bingtian Xue*
Invited Talk:
Time-Frequency Logic For Signal Processing

Radu Grosu
Vienna University of Technology, Austria

Abstract. In this talk I will first introduce Time-Frequency Logic (TFL), a new specification formalism for real-valued signals that combines temporal logic properties in the time domain with frequency-domain properties. I will then present a property checking framework for this formalism and demonstrate its expressive power to the recognition of musical pieces. Like hybrid automata and their analysis techniques, the TFL formalism is a contribution to a unified systems theory for hybrid systems. This is joint work with Alexandre Donze, Oded Maler, Dejan Nickovic, Ezio Bartocci and Scott Smolka.
HydLa: A High-Level Language for Hybrid Systems

Kazunori Ueda¹, Shota Matsumoto¹, Akira Takeguchi¹, Hiroshi Hosobe², and Daisuke Ishii²

¹ Department of Computer Science and Engineering, Waseda University
  3-4-1, Okubo, Shinjuku-ku, Tokyo 169-8555, Japan
  {ueda,matsusho,takeguchi}@ueda.info.waseda.ac.jp
² National Institute for Informatics
  2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
  hosobe@nii.ac.jp, dsksh@acm.org

1 HydLa, A Hybrid Constraint Language

We have been working on the design and implementation of HydLa, a modeling language for hybrid systems [5]³. The principal feature of HydLa is that it employs constraint-based formalisms both in the modeling and reliable simulation of hybrid systems. We take this approach for two reasons: one is that a constraint-based formalism is non-procedural but yet provides the language with control structures including synchronization and conditionals that are expressive enough to model hybrid systems, and the other is it allows us to handle uncertainties or partial information in a smooth way. Rather few tools for hybrid systems fully exploit constraint-based formalisms. The closest previous work was Hybrid cc [2][3], but HydLa differs in that its implementation ensures the correctness of simulation results. Another constraint-based approach was CLP(F), constraint logic programming over real-valued functions [4]. Both CLP(F) and HydLa aim at rigorous simulation and handle intervals, but they have very different control structures.

HydLa programs are sets of constraint modules that describe static and/or dynamic properties of systems using (among others) ordinary differential equations, implication, and a temporal operator. Constraint modules form constraint hierarchies [1] that define priorities between constraints. In determining the set of trajectories by constraint satisfaction, a maximal consistent subset of the set of effective constraints is taken that satisfies the requirements of HydLa’s declarative semantics [5]. Implication and constraint hierarchy govern the change of the set of effective constraints over time. Constraint-based modeling allows high-level description but can easily cause over- and under-constrainedness, but constraint hierarchy provides us with a concise mechanism that makes trajectories well-defined.

³ The English version of [5] appears in Appendix of this paper.
INIT <=> h=10 /\ h'=0 /\ timer=0.
PARAMS <=> exT=3 /\ volume>3 /\ volume<10 /\ \[\] (exT'=0 /\ volume'=0).
TIME <=> \[\] (timer'=1).
RESET <=> \[\] (timer- => volume+exT => timer=0).
BURN <=> \[\] (timer- <volume => h''=1).
FALL <=> \[\] (timer- => volume => h''=-2).

ASSERT(h>=0).

INIT, PARAMS, BURN, FALL, TIME<<RESET.

Fig. 1. Hot-air balloon model in HydLa

2 An Example Model

Figure 1 describes a model of a hot-air balloon going up by using multiple fuel tanks. Each fuel tank lasts volume time units and changing it takes exT time units. Uppercase names stand for constraint modules, x' stands for the time derivative of x, \[\] stands for the always temporal operator, and the postfix minus sign of x- stands for the left limit of x, where each variable is interpreted as a function of time. The first six lines are module definitions: INIT defines initial values of h and timer; PARAMS defines the values of the two parameters exT and volume; TIME and RESET define the continuous and discrete changes of the variable timer, respectively; BURN and FALL define the two modes of operations. TIME<<RESET means TIME is superseded by RESET when they contradict. Other modules are not superseded by any other modules and are always in effect. Note that the initial value of volume is given as an interval constraint. Figure 2 shows possible trajectories of the height h, where volume = 3.0, 3.1, ..., 10.0. The actual output from HydLa represents an infinite number of trajectories by using the symbolic parameter pvolume (see Section 4), and the trajectories of Fig. 2 were sampled for the purpose of drawing.

Although HydLa is a language for reliable simulation, it comes with an assertion construct as shown in Fig. 1 that can be used for checking simple global properties.

3 Nondeterministic Simulation Algorithm

We have been developing Hyrose, an implementation of HydLa’s nondeterministic simulation algorithm given in [6]. The principles of Hyrose are (i) to guarantee the accuracy of answers and (ii) to be able to compute all possible trajectories so that it can be used for reasoning about hybrid systems. Simulation proceeds by successive constraint satisfaction of alternating point phases (PP, a.k.a. jump) and interval phases (IP, a.k.a. flow), where phase change is triggered either by the discharging of constraints from implicational constraints or the change of
maximal consistent set of modules. An important feature of the HydLa’s simulation algorithm is that it allows models containing symbolic parameters whose values are possibly specified as interval constraints. Uncertainty expressed this way may cause nondeterminism in the truth/falsity of the antecedent of an implicational constraint, in which case the simulation algorithm splits the interval into subintervals that make the antecedent uniformly true and those that make the antecedent uniformly false, and subsequent simulation may pursue all those alternatives. In this way, the algorithm automatically performs case analysis and classifies possible trajectories into qualitatively equivalent groups.

4 Simulating the Hot-Air Balloon Model

Hyrose is currently based on symbolic computation, though it also employs interval computation to be able to compare two concrete or parametric values rigorously. Figure 3 shows a fragment of the execution result (for 30 time units) of the hot-air balloon model without the ASSERT check. It shows the third point phase at time $3 + p_{volume}$ and the third interval phase of time $(3 + p_{volume}, 3 + 2 \cdot p_{volume})$ of the case $p_{volume} \in [9/2, 21/4]$, where $t$ is the current time and $p_{volume}$ is a symbolic parameter introduced by Hyrose to represent the initial values of $volume$. For this model, Hyrose returned a total of six cases which differed only in the number of phases within the simulation time. Hyrose’s automatic case analysis can handle multiple symbolic parameters for this example, while the power of automatic case analysis depends on the underlying constraint solver (which can be chosen from Mathematica and REDUCE currently). Figure 4 shows five qualitatively different cases that may happen in 5 time units of simulation with $volume \in (1, 3)$ and $exT \in (2, 4)$. Zones marked as “assertion failed” violate the constraint $h \geq 0$. “PP$n$” means that $n$ point phases have been encountered in the simulation.
#---------3---------
---------PP---------
time : 3+pvolume
eT : 3
h : 1/2*(2+6*pvolume+pvolume^2)
timer : 0
volume : pvolume
eT' : 0
h' : -6+pvolume
timer' : UNDEF
volume' : 0
h'' : -2

---------IP---------
time : 3+pvolume -> 3+2*pvolume
eT : 3
h : 1/2*(47+18*pvolume+(-18)*t+t^2)
timer : -3+(-1)*pvolume+t
volume : pvolume
eT' : 0
h' : -9+t
timer' : 1
volume' : 0
h'' : 1

#---------parameter condition---------
volume : [9/2, 21/4]

Fig. 3. Output from Hyrose (fragment)

Fig. 4. Classifying two-dimensional parameter space
Other applications of hybrid systems with parameters or uncertainties include analysis of systems with singular points and sensitivity analysis. Hyrose is still in its initial stage and the size of the systems it can handle is limited by the underlying constraint solver, but it is beginning to show the viability of HydLa’s constraint-based simulation algorithm and is gaining a role complementary to other tools aiming at simulating and analyzing large hybrid systems.

Acknowledgments We are indebted to the other members of the HydLa group for their contribution to the implementation and daily discussions. This research is partially supported by Grant-In-Aid for Scientific Research ((B) 23300011), JSPS, Japan.

References

Appendix:

Declarative Semantics of the Hybrid Constraint Language HydLa *

Kazunori Ueda¹, Hiroshi Hosobe², and Daisuke Ishii³

¹ Department of Computer Science and Engineering, Waseda University
² National Institute of Informatics
³ University of Nantes

Abstract. Hybrid systems are dynamical systems with continuous evolution of states and discrete evolution of states and governing equations. We have been working on the design and implementation of HydLa, a constraint-based modeling language for hybrid systems, with a view to the proper handling of uncertainties and the integration of simulation and verification. HydLa’s constraint hierarchies facilitate the description of constraints with adequate strength, but its semantical foundations are not obvious due to the interaction of various language constructs. This paper gives the declarative semantics of HydLa and discusses its properties and consequences by means of examples.

A.1 Introduction

Hybrid systems are dynamical systems with continuous evolution of states and discrete evolution of states and governing equations. We have been developing a modeling framework of hybrid systems based on the notion of Constraint Programming. Our goal is to establish a constraint-based paradigm in which (i) to describe diverse phenomena found in physical, cyber-physical, and biological systems using logical formulae involving equations and inequations and (ii) to solve or verify them using search techniques represented by constraint propagation.

Our motivation has been to establish, in the field of hybrid systems, a declarative programming paradigm that directly handles as source programs high-level description of problems in mathematical and logical formulas, as opposed to traditional formalisms based on automata and Petri Nets [4][1]. A similar approach was first taken by Hybrid CC [3], and we have made a lot of experiments on Hybrid CC programming. However, we found that it was not necessarily straightforward to specify constraints that a system consisting of alternate discrete and continuous phases should satisfy, and this lead us to design a new language that enables a concise description of hybrid systems.

* This is an English translation of the paper that appeared in Computer Software, Vol. 28, No. 3 (2011), pp.167–172, available online at https://www.jstage.jst.go.jp/article/jssst/28/1/28_1_1_306/_article/.
Appendix: Declarative Semantics of HydLa

\[ \text{INIT} \iff ht=10 \land ht'=0. \]
\[ \text{PARAMS} \iff \square(g=9.8 \land c=0.5). \]
\[ \text{FALL} \iff \square(ht''=-g). \]
\[ \text{BOUNCE} \iff \square(|ht-|=0 \Rightarrow ht'= -c*(ht'-)). \]

\[ \text{INIT, PARAMS, (FALL} \ll \text{BOUNCE)}. \]

Fig. A.1. A bouncing ball

Since the basic design of HydLa was established in 2008 [9], we studied the details of the language through the description of a number of examples [5], developed a simulation algorithm [8] and a prototype implementation, and explored technologies for implementing discrete changes with guaranteed accuracy [6]. All those studies contributed to the clarification of the essence and subtle points of the HydLa language specification. Based on those experiences, this paper formulates the declarative semantics of the core of HydLa, and discusses its descriptive power and properties by means of examples.

A.2 Overview of HydLa

HydLa is a declarative language for hybrid systems. Its objective is to allow one to provide the mathematical formulation of a given problem with minimal modification and to simulate or analyze them. For the design principles and related work of HydLa, the readers are referred to [9].

Dynamical systems that HydLa aims to handle are in general represented as a countable number of real-valued functions \( x_1(t), x_2(t), \ldots (t \geq 0) \) that include integer-valued functions as a special case. A HydLa program imposes constraints on the behavior of those functions (hereafter called trajectories) that may cause continuous or discrete changes over time. The declarative semantics of a HydLa program \( P \) is defined as a satisfaction relation between trajectories \( \pi(t) = \{x_i(t)\}_{i \geq 1} \) and \( P \), or equivalently, the set of all \( \pi(t) \)'s that satisfy \( P \).

In order to describe hybrid systems in a concise manner, the use of hierarchies to represent defaults and exceptions will play an important role exactly as in knowledge representation and object-oriented design. Consider a ball bouncing on a floor. The change of the velocity of the ball is determined by the gravity most of the time (default), while it is determined by the collision equation when the ball hits the floor (exception). A mathematically concise way to describe solution trajectories of such systems in a well-defined matter would be to introduce partial order between candidate sets of equations that the system should satisfy and to take a maximally consistent element of the partially ordered set (poset) of sets of constraints. HydLa's design principle is exactly based on this idea.

Figure A.1 shows the description of a bouncing ball in HydLa. The first four lines are the definition of constraint modules. Constraint modules are program
units which are combined to form a set of constraints and to which priorities may be given. In the right-hand side constraints, ‘′ stands for a time derivative, the postfix minus sign stands for the left-side limit of a trajectory, and □ stands for an always temporal operator. All the constraints stand for constraints at time 0. However, since the constraints other than INIT start with □, they hold at all time points on and after time 0. A constraint with an implication (such as BOUNCE) is called a conditional constraint. A conditional constraint prefixed by an □ imposes its consequent exactly when its antecedent (guard) holds.

The final line combines the four constraint modules. A comma stands for composition without priorities, while << gives a higher priority to BOUNCE than to FALL. In this example, all the four constraints are taken when the ball is in the air, while {INIT, PARAMS, BOUNCE} will be taken as the maximally consistent set when the ball hits the floor because FALL and BOUNCE become inconsistent.

This example is known to exhibit a Zeno behavior, an infinite number of discrete changes within a finite amount of time, beyond which the simulation normally does not proceed.

A.3 Basic HydLa

We consider the semantics of the Basic HydLa whose syntax is shown in Fig. A.2. Basic HydLa simplifies HydLa [9] as follows:

1. For each time point, HydLa chooses a consistent set of constraint modules that satisfies the priority constraint and that is maximal with respect to the set inclusion relation between constraint modules. More specifically, from a relative priority relation between constraint modules, HydLa first derives a poset whose elements are admissible (with regard to constraint priorities) of all the subsets of constraint modules [5], and then chooses a maximal consistent element. Basic HydLa does not handle this derivation but assumes that the “(irreflexive) poset of sets of constraint modules” is directly given in a program together with the definitions of constraint modules. Default constraints such as the continuity of trajectories (frame axioms, see Section A.6.1) are to be explicitly specified within this poset. The constraints

Fig. A.2. Syntax of the Basic HydLa.
at the top of a constraint hierarchy should often be treated as required constraints that must be adopted, and whether to do so can be expressed explicitly within the poset.

2. Basic HydLa does not support the time shift (i.e. delay) operator \( \hat{\cdot} \). We can use the feature explained in the next item instead.

3. To enable dynamic creation of trajectories, Basic HydLa introduces an existential quantifier \( \exists \) for local variable creation. This enables us to dynamically create a timer with which to represent a delay between the detection of some condition and the issue of a new constraint.

4. Basic HydLa does not support program definitions since they can be simply inlined.

5. For the same reason, Basic HydLa does not support the operator \( \forall \) to generate a family of trajectories.

We assume that a Basic HydLa program \((MS, DS)\) satisfies \( \bigcup MS \subseteq \text{dom}(DS) \), where \( \bigcup MS \) is the set of modules appearing in \( MS \) and \( \text{dom}(DS) \) is the set of left-hand sides of \( DS \). In the following, we consider a set \( DS \) of constraint module definitions as a function from module names to constraints.

As shown in Fig. A.2, we restrict the guard constraints to atomic constraints and their conjunctions. HydLa does not specify the class of constraints that can be described in a program. In this sense, HydLa is a language scheme that parameterizes constraint systems. The reason why we allow only \( \Box \) as a temporal operator is that our syntax is targeted at the modeling of systems. Other temporal operators such as \( \Diamond \) will be included in the specification language when we construct a verification system that use HydLa as a modeling language.

### A.4 Declarative Semantics of Basic HydLa

As shown in Section A.2, the declarative semantics of HydLa is defined as a relation meaning that a given trajectory (or interpretation) satisfies a program (or specification). The information to be maintained by the declarative semantics depends on design criteria such as what class of programs it deals with and what degree of compositionality (i.e., the ability to compose the overall semantics from the semantics of components) it aims at. The semantics in [9] dealt with programs containing no \( \Box \) operators in the consequents of conditional constraints. Parameters and behaviors of systems with a finite number of components and no delays can be described by constraints with \( \Box \)'s only in their prenex positions. When those programs contain conditional constraints, their consequents hold exactly when the antecedents hold, which means that a maximal consistent set of constraints can be chosen at that time.

However, a constraint whose consequent includes an \( \Box \hat{\cdot} \) leaves the consequent as a candidate for choice even after the corresponding antecedent ceases to hold. If we have to judge which consequents of constraints should be chosen in the future when the corresponding antecedents held, it would be a lookahead of the future. Thus the choice of a maximal consistent set must be performed not
when constraints are discharged but when the constraints are actually applied. Therefore we further refine our semantics in the following way.

First, we identify a conjunction of constraints with a set of constraints; i.e., we view the syntax of a constraint in Fig. A.2 as
\[ C := \{A\} | C \cup C | \{G \Rightarrow C\} | \{\Box C\} | \{\exists x. C\} \]
and also allow an empty set. By Skolemization, we recursively eliminate existential quantifiers \( \exists \) except for those occurring in the consequents of conditional constraints.

Next, we consider constraint sets as functions of time. For example, a constraint \( C \) that occurs in a program is regarded as a function \( C(0) = C, C(t) = \{\} (t > 0) \).

For a constraint \( C(t) \) that is a function of time, the \( \Box \)-closure \( C^*(t) \) is defined as a function that satisfies the following properties:

- (Extension) \( \forall t (C(t) \subseteq C^*(t)) \);
- (\( \Box \)-closure) \( \forall t (\Box a \in C^*(t) \Rightarrow \forall t' \geq t (a \subseteq C^*(t'))) \);
- (Minimality) For each \( t \), \( C^*(t) \) is the minimum set that satisfies the above two conditions.

For \( C = \{f=0, \Box \{f'=1\}\} \) for example, we have \( C^*(0) = \{f=0, f'=1, \Box \{f'=1\}\}, C^*(t) = \{f'=1\} (t > 0) \).

The constraint set that determines a solution trajectory of a HydLa program may change over time for two reasons: one is that a maximal consistent set may change; the other is that the consequent of a conditional constraint is newly added when its antecedent holds. The choice of a maximal consistent set in the former case is performed independently at each time point. By contrast, when the program has a constraint whose consequent begins with \( \Box \), whether the constraint is active or not depends on whether its antecedent has been activated in the past; hence the state of a system should maintain the activation history of the antecedents. Therefore it is appropriate to consider a satisfaction relation stating that a program \( P = (MS, DS) \) is satisfied by a pair \((\pi, Q)\) of a solution trajectory \( \pi = \pi(t) \) and the constraint module definition \( Q = Q(M)(t) (M \in \text{dom}(DS)) \) recording the activation of antecedents. We define this relation as shown in Fig. A.3.

The principle of the declarative semantics in Fig. A.3 is the consistency-based adoption of constraints. It requires that, at each time point, a consistent set of constraint modules with a maximal preference must be adopted and satisfied.

Condition (i) requires \( Q(M) \) to satisfy the \( \Box \)-closure property, and Condition (ii) requires \( Q(M) = Q(M)^* \) to be an extension of \( DS(M)^* \). Now we look into Condition (iii) closely. The order of the quantifiers at Line (s0) allows \( \pi \) to choose, at each time point, a different set of candidate modules from the constraint hierarchy. Line (s1) means that, at time \( t \), \( \pi \) satisfies some set of candidate modules in the constraint hierarchy. Lines (s2) mean that there is no trajectory \( \pi' \) that behaves exactly as \( \pi \) before \( t \) and satisfies a better candidate module set than \( \pi \) at \( t \). Lines (s3) mean that, when the antecedent of a chosen conditional
Appendix: Declarative Semantics of HydLa

\[
\langle x, Q \rangle \models (MS, DS) \iff (i) \land (ii) \land (iii) \land (iv), \text{ where }
\]

(i) \forall M \ (Q(M) = Q(M)^*);
(ii) \forall M \ (DS(M)^* \subseteq Q(M));
(iii) \forall t \exists E \in MS( (s0)
\begin{align*}
& (\pi(t) \Rightarrow \{Q(M)(t) \mid M \in E\}) \\
& \land \neg \exists \exists' \exists E' \in MS( \\
& \forall t' < t \ (\pi(t') = \pi(t'))) \\
& \land E \prec E' \\
& \land \pi'(t) \Rightarrow \{Q(M)(t) \mid M \in E'\})
\end{align*}

(s2)

(iv) For each \( M \) and \( t \), \( Q(M)(t) \) is the minimum set that satisfies (i)–(iii).

Fig. A.3. Definition of \( \langle x, Q \rangle \models P \)

constraint holds, \( Q \) is extended by expanding its consequent into the definition of the corresponding module \( M \) in \( Q \). If a member of the consequent (regarded as a set of constraints) begins with \( \Box \), it is further expanded by the \( \Box \)-closure condition (i). Also, if it begins with \( \exists \), the quantifier is eliminated by using a Skolem function. Condition (iv) requires the minimality.

A.5 Examples

Using simple examples, we explain how the declarative semantics actually defines solution trajectories and the constraint sets used to determine them.

Example 1: The first example shows how the arrival of a monotonically increasing function at a certain threshold is reflected to another function with a delay.

\[
P_1 = (MS_1, DS_1)
\]

\[
MS_1 = \{\{A, C\}, \{A, B, C\}\}, \{\{A, C\} \prec \{A, B, C\}\}
\]

\[
DS_1 = \{A \iff f=0 \land \Box(f'=1),
B \iff \Box(g=0),
C \iff \Box(f=5) \land \exists a. (a=0 \land \Box(a'=1) \land \Box(a=2 \Rightarrow g=1))\}
\]

Here, \( f \) is a function that expresses the current time, \( a \) is a timer invoked by \( f=5 \) as the trigger, and \( g \) is a pulse function that is usually 0 but momentarily becomes 1 two seconds after the invocation of the timer. The solution trajectory \( \pi \) expresses those behaviors of \( f, a \) (whose Skolem function is also called \( a \) here), and
g. Now we see all the constraints \( Q(\ast)(t) = \bigcup\{ Q(M)(t) \mid M \in \text{dom}(DS_1)\} \) that are stored in \( Q \). At \( 0 < t < 5 \), \( Q(\ast)(t) \) consists of \( f'=1 \), \( g=0 \), and the constraint \( C \) with the leftmost \( \Box \) removed. At \( t = 5 \), \( a=0 \), \( \Box(a'=1) \), \( a'=1 \), \( \Box(a=2 \Rightarrow g=1) \), and \( a=2 \Rightarrow g=1 \) are added to them. At \( 5 < t < 7 \), \( a=0 \), \( \Box(a'=1) \), and \( \Box(a=2 \Rightarrow g=1) \) are removed. At \( t = 7 \), \( g=1 \) replaces \( g=0 \). At \( t > 7 \), \( g=1 \) is replaced by \( g=0 \) again; the other constraints that remain are \( f'=1 \), \( a'=1 \) and the two conditional constraints.

**Example 2:** The declarative semantics presented in the previous section disallows the propagation of constraints to the past. This may be obvious from the construction of the semantics, but we confirm it by using an example since it is an important property.

\[
P_2 = ((P(\{D,E,F\}), \subseteq), DS_2)
\]
\[
DS_2 = \{ D \Leftrightarrow y=0, \\
E \Leftrightarrow \Box(y'=1 \land x'=0), \\
F \Leftrightarrow \Box(y=5 \Rightarrow x=1) \}
\]

\( P_2 \) leaves the initial value of \( x \) undefined. We check whether the constraint \( x=1 \) imposed by \( F \) at \( t = 5 \) propagates to the past by the effect of \( x'=0 \) in \( E \). We consider the following three cases as candidates for solution trajectories.

1. \( y(t) = t \ (t \geq 0) \) and \( x(t) = 1 \ (t \geq 0) \) satisfy all the constraints \( D, E, \) and \( F \) at all times.
2. \( y(t) = t \ (t \geq 0) \) and \( x(t) = 2 \ (t \geq 0) \) satisfy all the constraints except at \( t = 5 \) and satisfy \( D \) and \( E \) at \( t = 5 \).
3. \( y(t) = t \ (t \geq 0), x(t) = 2 \ (t < 5), \) and \( x(t) = 1 \ (t \geq 5) \) satisfy all the constraints except at \( t = 5 \) and satisfy \( D \) and \( F \) at \( t = 5 \).

Case 1 is a solution since it obviously satisfies the maximality. Cases 2 and 3 obviously satisfy the maximality except at \( t = 5 \), and there are no better solutions than these. Neither of them is worse than the other at \( t = 5 \), and there are no other solutions that satisfy all the constraints; hence both of them are maximal. In other words, any of Cases 1 to 3 is a result of “extending a solution along the time axis so the maximality is satisfied,” and is therefore a solution to \( P_2 \).

### A.6 Discussions on the Specification and the Semantics of the Language

#### A.6.1 Differential Constraints

The basic principle of HydLa to utilize existing mathematical and logical notations as much as possible suggests that the precise meaning of the notations should also conform to mathematical conventions. For example, at the time point
where a piecewise continuous trajectory causes a discrete change, we do not consider the trajectory differentiable even if it is differentiable both from the left and the right, and we do not deactivate the differential constraints at that time point. We also assume only the right continuity and right differentiability at the initial time.

For the reasons above, the priority of the differential constraints of a piecewise continuous function should in general be lower than that of the constraints describing discrete changes. On the other hand, for a continuous trajectory after a discrete change to be well-defined as an initial value problem of an ordinary differential equation, we need to assume the right continuity at the time of the discrete change. Since the differential constraints are deactivated when a discrete change occurs, we also require left continuity to be able to decide the value of a trajectory. Accordingly, HydLa assumes both the right and the left continuity of trajectories described by differential constraints. Since these two continuity constraints are automatically entailed whenever a trajectory is differentiable, we assume them separately with a priority higher than differential constraints.

**A.6.2 Expressive Power of HydLa**

Although the primary purpose of HydLa is to describe piecewise continuous trajectories, we can define various trajectories or sets of trajectories using HydLa’s constraints and constraint hierarchies.

**Trajectories defined without using differential equations.** HydLa allows us to describe trajectories without using differential constraints. For example, a drifting parameter can be described by a constraint $\square(0.9 < a \land a < 1.1)$, which represents the set of all trajectories whose range is $(0.9, 1.1)$.

Note that a trajectory defined by the above constraint may not be continuous. Hence, a trajectory defined by $f=0 \land \square(f'=1)$ is not guaranteed to satisfy $f=a$ between time 0.9 and 1.1. By adding a constraint $\square(a'=b)$ (we do not add any constraint for $b$), $a$ stands for a set of all continuous and differentiable trajectories whose range is $(0.9, 1.1)$, and is guaranteed to intersect with $f$.

A pulse function is another example defined without differential constraints. An example of a pulse function is $g$ of Example 1 (Section A.5). Pulse functions play a significant role in representing the occurrences of events. Since pulse functions are not right-continuous at the time of discrete changes, we conjecture that a trajectory after the discrete change cannot be defined directly by a differential equation. The following example shows that our attempt to define a pulse function $b$ fails:

$$P_3 = (MS_3, DS_3)$$
$$DS_3 = \{ G \leftrightarrow a=0 \land b=0 \land \square(a'=1),
\quad H \leftrightarrow \square(b'=0),
\quad J \leftrightarrow \square(a=1 \Rightarrow b=1) \land \square(b=1 \Rightarrow b=0)\}$$
Based on the discussion in Section A.6.1, between two sets of constraint modules in MS3, there exist several sets with additional constraints on the continuity including the right continuity of $b$. At $t = 1$, the set $\{G, H, J\}$ is not satisfiable but $\{G, J\}$ with the right continuity of $b$ is satisfiable, and $b(1) = 1$ holds from the first constraint of $J$. However, then, the greatest lower bound of the time when the guard of the second constraint of $J$ holds is $t = 1$. The consequent of the constraint $b(t) = 0$ is thus activated at $t > 1$ and contradicts the right continuity. Now suppose we drop the assumption of the right continuity. Then it turns that $b(t) = c (t > 1)$ is consistent for all $c \neq 1$. Therefore, although there exists a solution trajectory, HydLa fails to guarantee its uniqueness.

**Zeno behaviors** Let us reinvestigate the bouncing ball example in Section A.2 based on the declarative semantics of Section A.4. Although the program in Fig. A.1 specifies a unique solution trajectory until the Zeno time, after that, it allows a trajectory that falls through the floor. We need some additional rules to specify the behavior after the Zeno time [10]. In HydLa, we can define it as $\square(\text{ht} = 0 \land \text{ht}' = 0 \Rightarrow \square(\text{ht} = 0))$, though checking the guard condition would need a special simulation method, e.g., in [7].

The following program shows another method for detecting the Zeno time. It checks the convergence of a function $v_{\text{max}}$ that holds the velocity at the last bounce.

\[
\square(v_{\text{max}}' = 0) \iff \\
\square(\text{ht}' = \text{ht}' \Rightarrow v_{\text{max}} = \text{ht}') \\
\land \square(v_{\text{max}} = 0 \Rightarrow \square(\text{ht} = 0))
\]

This example shows that the left limit operator $-$ is also useful for a function that only causes discrete changes.

**A.7 Conclusions and Future Work**

This paper gave the declarative semantics of HydLa, a hybrid constraint language with hierarchical structure, described its mechanisms and consequences by means of examples, and discussed the language features and expressive power.

The semantics given in this paper regards trajectories as functions over time. On the other hand, the theory of hybrid systems often adopts hybrid time that allows more than one discrete change at a single time point [2]. One of the motivations of hybrid time is to model computation involving multiple steps at the time of a single discrete change. However, because HydLa is constraint-based, such evolution can be represented as constraint propagation rather than state changes. Another motivation of hybrid time is to deal with the stability and convergence of trajectories in a declarative framework. This would require the extension of our semantics with hybrid time, which is a topic of future work.

We are currently working on the formulation and its implementation of a simulation algorithm corresponding to our declarative semantics. The resulting
system is planned to exploit the flexibility of constraint programming and the affinity to interval computation.

Acknowledgments The design of HydLa has received constant feedback from the theoretical and implementation work and daily discussions of the past and present members of the HydLa group. This research is partially supported by Grant-In-Aid for Scientific Research ((B) 20300013), JSPS, Japan.

References

Interpolation-based Function Summaries in Bounded Model Checking

Ondrej Sery, Grigory Fedyukovich, and Natasha Sharygina

Formal Verification Lab, University of Lugano, Switzerland

Abstract. During model checking of software against various specifications, it is often the case that the same parts of the program have to be modeled/verified multiple times. To reduce the overall verification effort, this paper proposes a new technique that extracts function summaries after the initial successful verification run, and then uses them for more efficient subsequent analysis of the other specifications. Function summaries are computed as over-approximations using Craig interpolation, a mechanism which is well-known to preserve the most relevant information, and thus tend to be a good substitute for the functions that were examined in the previous verification runs. In our summarization-based verification approach, the spurious behaviors introduced as a side effect of the over-approximation, are ruled out automatically by means of the counter-example guided refinement of the function summaries. We implemented interpolation-based summarization in our FunFrog tool, and compared it with several state-of-the-art software model checking tools. Our experiments demonstrate the feasibility of the new technique and confirm its advantages on the large programs.
Ondrej Sery, Grigory Fedyukovich, and Natasha Sharygina
Abstract Conflict Driven Clause Learning

Vijay D’Silva, Leopold Haller, and Daniel Kroening
Department of Computer Science, Oxford University
firstname.surname@cs.ox.ac.uk

The performance of solvers for the propositional satisfiability problem (SAT) has improved at an exponential rate in the last decade. Underlying these improvements is the Conflict Driven Clause Learning (CDCL) algorithm, efficient data structures, and heuristics that exploit the non-adversarial nature of practical problems. Modern satisfiability solvers are an important cornerstone of modern decision-procedure based program verification, which is sometimes seen as an alternative to classic approaches based on abstract interpretation.

Transferring the success of CDCL to new domains is an open research problem. One approach is to use propositional CDCL solvers as components of more complex decision procedures. The DPLL(T) framework, which has been studied extensively in Satisfiability Modulo Theory (SMT) research, provides a mathematical and algorithmic recipe to implement and reason about these types of algorithm. More recently, so-called natural domain SMT procedures have been proposed, which attempt to emulate the success of CDCL by lifting the algorithm to operate directly over richer logics or program verification problems.

We present Abstract CDCL (ACDCL), a systematic framework for deriving natural domain SMT procedures. Our work is based on the simple insight that existing CDCL solvers can be viewed as abstract interpreters for logical formulae [1]. This view allows one to clearly discern an abstract, domain independent algorithm at the core of CDCL.

Modern CDCL is based on the interleaved execution of model search, which heuristically searches for solutions, and conflict analysis, which heuristically learns explanations for failed model search runs. We show that model search and conflict analysis can be understood as fixed point computations over abstract domains. The result of this characterisation is a mathematical and algorithmic framework for building natural domain SMT procedures for satisfiability and program verification. The problem of lifting CDCL to a new domain is reduced to the well-understood problem of designing abstract domains. Furthermore, many domains proposed for program analysis can be directly integrated into CDCL.

We have instantiated our algorithm over the abstract domain of intervals to yield program analysers [2] and SMT solvers that outperform the state of the art in terms of precision and efficiency.

References

Invited Talk:
Verifying Safety of Hybrid Systems

Ashish Tiwari
SRI International, USA

Abstract. The safety verification problem for hybrid systems asks if a hybrid system can ever reach an undesirable state. There are at least three different approaches that have been studied for solving this problem: (a) reach set computation by applying abstract interpretation on the hybrid dynamics (b) abstraction of the hybrid system and model checking the abstract system (c) direct proof of safety by demonstrating existence of an inductive invariant In this talk, we argue that ideas from each of the three approaches can be used to improve the other approaches. We present three concrete safety verification techniques: qualitative abstraction, inductive invariants, and relational abstraction, and show how ideas from one influence the other techniques.
HySon: Precise Simulation of Hybrid Systems with Imprecise Inputs

Olivier Bouissou\textsuperscript{1}, Samuel Mimram\textsuperscript{2}, and Alexandre Chapoutot\textsuperscript{3}

\textsuperscript{1} Commissariat à l’Energie Atomique, France
\textsuperscript{2} CEA, LIST, France
\textsuperscript{3} ENSTA ParisTech, France

Abstract. Hybrid systems are a widely used model to represent and reason about control-command systems. Most of the work in this domain is devoted to compute reachable sets of hybrid automata or equivalent models. However, in an industrial context, control-command systems are often implemented in Simulink and their validity is checked using numerical simulation. In this article, we present a tool named HySon that performs set-based simulation of hybrid systems with uncertain parameters, expressed in Simulink. Our tool handles advanced features such as non-linear operations, zero-crossing events or discrete sampling. It is based on well-known, efficient numerical algorithms that were adapted to handle set-based domains. We demonstrate the performance of our method on various examples.
Decidability and Completeness of PDL through Canonical Model *

Xinxin Liu a and Bingtian Xue a,b

a State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
and b Graduate School of Chinese Academy of Sciences
P.O.Box 8718, 100190 Beijing, China
xinxin,xuebt@ios.ac.cn

Abstract. Propositional dynamic logic (PDL) is a logic aimed at program verification. Canonical model is a powerful notion which is often used in proving completeness of logical systems. We propose a method to construct canonical models for PDL formulas. Using this method we can obtain a procedure for checking satisfiability of formulas. For a given formula \( \varphi \), the procedure will end with a canonical model for it if \( \varphi \) is satisfiable, and will end with a proof of its negation \( \neg \varphi \) if \( \varphi \) is not satisfiable. This also gives a constructive proof of the completeness of Segerberg’s axiomatization for PDL.

1 Introduction

Propositional Dynamic Logic (PDL) is a logic aimed at program verification. It was introduced by Fischer and Ladner [1] in the late 1970s as a formalism for reasoning about programs. It uses regular expressions for programs in which iterative patterns of traces can be described. Soon afterwards the logic was outdated for that purpose through the introduction of the modal \( \mu \)-calculus - a much more expressive logic with a little higher complexity. However, there has been a resurgence of interest in PDL in recent years. PDL has by now become a standard logic that is far from being outdated. It can be used in program verification, to describe the dynamic evolution of agent-based systems, for planning or knowledge engineering, it has links to epistemic logics, it is closely related to description logics, etc. In [2] Lange studied model checking problem for PDL extended with some operators on programs such as repeat and loop. In [3], instead of introducing recursive definitions for propositions, Leivant proposed PDL with recursive procedures. The resulting logic \( \mu \)PDL is strictly more powerful than the modal \( \mu \)-calculus. In [4] Löding, Lutz, and Serre studied satisfiability problem for certain non-regular extension of PDL and showed that the problem is still decidable.

In this paper we study the satisfiability checking problem in PDL, that is, for an arbitrary given PDL formula, decide if there is a model in some state of which

* Research supported by ANR-NSFC 61161130530 and CAS Innovation Program
the given formula is satisfied. The classical algorithm for checking satisfiability of PDL formulas is starting from the maximum sets of the given formula’s Fischer-ladner closure, delete those sets formulas whenever inconsistency is found, and finally obtain a canonical model. For a given formula, the algorithm will output a canonical model for it if there exists one.

In this work, we propose an improved method for canonical model construction within PDL. Using this method we can obtain not only a canonical model for a given formula \( \psi \) when the formula is satisfiable, but also a proof of \( \neg \psi \) when the formula is not satisfiable. This also gives a constructive proof of the completeness of Segerberg’s axiomatization for PDL. A more detailed comparison of our method with the existing methods is provided in the conclusion.

In the following section we review the syntax and semantics of PDL, and the Segerberg’s axiomatization. In section 3 we describe our canonical model construction method, and state the related main results about deciding satisfiability and the completeness of Segerberg’s axiomatization. In section 4 we present the proof of the main theorem. In the last section we conclude our work, together with some remarks on future and related works.

2 Preliminary

The presentation of PDL here follows closely from that in [5].

The language of PDL has expressions of two sorts: propositions or formulas \( \varphi, \psi, \ldots \) and programs \( \alpha, \beta, \gamma, \ldots \). There are countably many atomic symbols of each sort. Atomic programs are denoted \( a, b, c, \ldots \) and the set of all atomic programs is denoted \( \Pi_0 \). Atomic propositions are denoted \( p, q, r, \ldots \) and the set of all atomic propositions is denoted \( \Phi_0 \). The set of all programs is denoted \( \Pi \) and the set of all propositions is denoted \( \Phi \). Programs and propositions are built inductively according to the following abstract syntax:

\[
\varphi ::= p \mid 0 \mid \varphi \rightarrow \psi \mid [\alpha] \varphi
\]

\[
\alpha ::= a \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^* \mid \varphi?
\]

Apart from the above basic logic constructs, we also use the following common propositional constructs such as \( 1, \lor, \land, \neg, \leftrightarrow \), and these are derived constructs with their common definitions. For a set of formulas \( S \subseteq \Phi \), we often write \( \bigwedge S \) for \( \bigwedge_{\psi \in S} \psi \).

The semantics of regular PDL is interpreted on Kripke frames \( K = (K, m_K) \), where \( K \) is a set of states \( u, v, w, \ldots \) and \( m_K \) is a meaning function assigning a subset of \( K \) to each proposition and a binary relation on \( K \) to each program. That is:

\[
m_K(\varphi) \subseteq K, \quad \text{for } \varphi \in \Phi;
\]

\[
m_K(\alpha) \subseteq K \times K, \quad \text{for } \alpha \in \Pi.
\]

Formally the meaning of \( m_K(\varphi) \) of \( \varphi \in \Phi \) and \( m_K(\alpha) \) of \( \alpha \in \Pi \) are defined by mutual induction on the structure of \( \varphi \) and \( \alpha \). The basis of the induction, which
specifies the meaning of the atomic propositions and programs, is already given in the specification of $\mathcal{R}$. The meaning of compound propositions and programs are defined as follows:

$$ m_{\mathcal{R}}(\emptyset) \overset{\text{def}}{=} \emptyset $$

$$ m_{\mathcal{R}}(\varphi \rightarrow \psi) \overset{\text{def}}{=} (K - m_{\mathcal{R}}(\varphi)) \cup m_{\mathcal{R}}(\psi) $$

$$ m_{\mathcal{R}}([\alpha] \varphi) \overset{\text{def}}{=} \{ u \mid \forall v \in K $$

$$ \text{if } (u, v) \in m_{\mathcal{R}}(\alpha) \text{ then } v \in m_{\mathcal{R}}(\varphi) \} $$

$$ m_{\mathcal{R}}(\alpha; \beta) \overset{\text{def}}{=} m_{\mathcal{R}}(\alpha) \circ m_{\mathcal{R}}(\beta) $$

$$ = \{ (u, w) \mid u, w \in K $$

$$ (u, w) \in m_{\mathcal{R}}(\alpha) \text{ and } (w, v) \in m_{\mathcal{R}}(\beta) \} $$

$$ m_{\mathcal{R}}(\alpha \cup \beta) \overset{\text{def}}{=} m_{\mathcal{R}}(\alpha) \cup m_{\mathcal{R}}(\beta) $$

$$ m_{\mathcal{R}}(\alpha^*) \overset{\text{def}}{=} \bigcup_{n \geq 0} m_{\mathcal{R}}(\alpha)^n $$

$$ m_{\mathcal{R}}(\varphi?) \overset{\text{def}}{=} \{ (u, u) \mid u \in m_{\mathcal{R}}(\varphi) \} $$

We write $\mathcal{R}, u \models \varphi$ and $u \in m_{\mathcal{R}}(\varphi)$ interchangeably, and say that $u$ satisfies $\varphi$ in $\mathcal{R}$, or that $\varphi$ is true at state $u$ in $\mathcal{R}$. We may omit $\mathcal{R}$ and write $u \models \varphi$ when $\mathcal{R}$ is understood. We say $\varphi$ is valid, written $\models \varphi$, if $\mathcal{R}, u \models \varphi$ holds for any $\mathcal{R}$ and any $u$ in $\mathcal{R}$.

The following list of axioms and rules, proposed by Segerberg, constitutes a sound and complete Hilbert-style deductive system for PDL:

(i) Axioms for propositional logic

(ii) $[\alpha](\varphi \rightarrow \psi) \rightarrow [\alpha] \varphi \rightarrow [\alpha] \psi$

(iii) $[\alpha](\varphi \land \psi) \leftrightarrow [\alpha] \varphi \land [\alpha] \psi$

(iv) $[\alpha \cup \beta] \varphi \leftrightarrow [\alpha] \varphi \land [\beta] \varphi$

(v) $[\alpha; \beta] \varphi \leftrightarrow [\alpha] [\beta] \varphi$

(vi) $[\psi?] \varphi \leftrightarrow \neg (\psi \land \neg \varphi)$

(vii) $\varphi \land [\alpha] [\alpha^*] \varphi \leftrightarrow [\alpha^*] \varphi$

(viii) $\varphi \land [\alpha^*] ([\varphi \rightarrow [\alpha] \varphi] \rightarrow [\alpha^*] \varphi)$

(MP) $\varphi, \varphi \rightarrow \psi$ \quad $\frac{\varphi}{\psi}$

(GEN) $\frac{\varphi}{[\alpha] \varphi}$

We write $\vdash \varphi$ if the proposition $\varphi$ is a theorem of this system.

**Theorem 1.** For all $\psi \in \Phi$, $\vdash \psi$ if and only if $\models \psi$.

This is the soundness and completeness statement of the axiom system. The soundness direction can be easily established by checking that all the axioms
are valid and all the rules are validity preserving. The completeness direction is much harder to establish, and there are many existing proofs. In this paper we will present a new proof by using canonical model construction.

3 Canonical model, decidability, and completeness

This section describes the main results of the paper.

The Fischer-Ladner closure of a formula \( \varphi_0 \), written \( FL(\varphi_0) \), is the smallest set which contains \( \varphi_0 \) and is closed in the following sense:

1. if \( \varphi \rightarrow \psi \in FL(\varphi_0) \) then \( \varphi, \psi \in FL(\varphi_0) \);
2. if \([\alpha]\varphi \in FL(\varphi_0)\) then \( \varphi \in FL(\varphi_0) \);
3. if \([\alpha; \beta]\varphi \in FL(\varphi_0)\) then \([\alpha][\beta]\varphi \in FL(\varphi_0)\);
4. if \([\alpha \cup \beta]\varphi \in FL(\varphi_0)\) then \([\alpha]\varphi, [\beta]\varphi \in FL(\varphi_0)\);
5. if \([\alpha^*]\varphi \in FL(\varphi_0)\) then \([\alpha]\alpha^*\varphi \in FL(\varphi_0)\);
6. if \([\psi]^\ast\varphi \in FL(\varphi_0)\) then \( \psi \in FL(\varphi_0) \).

For a given \( \varphi_0 \in \Phi \), let \( \Omega(\varphi_0) = 2^{FL(\varphi_0)} \), i.e. \( \Omega(\varphi_0) \) is the set of all subsets of \( FL(\varphi_0) \). Then every \( N \subseteq \Omega(\varphi_0) \) induces the following Kripke frame:

\[ \mathcal{R} = (N, m_{\mathcal{R}}) \]

where for all \( p \in \Phi_0 \) and \( a \in \Pi_0 \):

\[ m_{\mathcal{R}}(p) \overset{\text{def}}{=} \{ \Gamma \in N \mid p \in \Gamma \} \]

\[ m_{\mathcal{R}}(a) \overset{\text{def}}{=} \{ \{ (\Gamma, \Gamma') \in N^2 \mid \text{ whenever } [a] \psi \in \Gamma, \text{ then } \psi \in \Gamma' \} \} \]

We say \( \mathcal{R} = (N, m_{\mathcal{R}}) \) is canonical if for all \( \psi \in FL(\varphi_0) \) the following holds:

\[ m_{\mathcal{R}}(\psi) = \{ \Gamma \in N \mid \psi \in \Gamma \} \]

We say \( \psi \in FL(\varphi_0) \) is a witness in \( \mathcal{R} = (N, m_{\mathcal{R}}) \) if \( m_{\mathcal{R}}(\psi) \neq \{ \Gamma \in N \mid \psi \in \Gamma \} \).

If \( \psi \in FL(\varphi_0) \) is a witness in \( \mathcal{R} = (N, m_{\mathcal{R}}) \), we say \( \psi \) is a minimal witness if all other witnesses are not a sub-formula of \( \psi \). We say \( \Gamma \in N \) is a key witness in \( \mathcal{R} = (N, m_{\mathcal{R}}) \) if there exists a minimal witness \( \psi \) in \( \mathcal{R} = (N, m_{\mathcal{R}}) \) such that either \( \Gamma \in m_{\mathcal{R}}(\psi) \) and \( \psi \notin \Gamma \), or \( \Gamma \notin m_{\mathcal{R}}(\psi) \) and \( \psi \in \Gamma \).

**Proposition 1.** \( \mathcal{R} = (N, m_{\mathcal{R}}) \) is not canonical if and only if there exists \( \Gamma \in N \) s.t. \( \Gamma \) is a key witness in \( \mathcal{R} = (N, m_{\mathcal{R}}) \).

The proof of this proposition is obvious according to the above definitions. Moreover, since \( FL(\varphi_0), \{ \Gamma \mid \psi \in \Gamma \} \) and \( m_{\mathcal{R}}(\psi) \) are all finite sets, if key witnesses exist, then a key witness can be found in finite steps.

We can now construct a sequence of structures \( \mathfrak{M}_i = (M_i, m_{\mathfrak{M}_i}) \) as follows: take \( M_0 = \Omega(\varphi_0) \); for \( i \geq 0 \), if \( \mathfrak{M}_i = (M_i, m_{\mathfrak{M}_i}) \) is canonical then it is the last in
the sequence, otherwise choose a key witness $\Gamma \in M_i$ and take $M_{i+1} = M_i - \{\Gamma\}$.

Obviously we obtain a decreasing chain

$$M_0 \supseteq M_1 \supseteq M_2 \supseteq \ldots,$$

and since each $M_i$ is a finite set, the chain must end with some canonical $M_n = (M_n, m_{M_n})$.

In order to discuss further properties of $M_0, \ldots, M_n$, we need the following notion:

For $N \subseteq \Omega(\varphi_0)$, we say $N$ is rich (with respect to $\varphi_0$) if for all $\Gamma \in \Omega(\varphi_0)$ either $\Gamma \in N$ or $\vdash (\bigwedge \Gamma \land \bigwedge T) \rightarrow 0$, where $T = \{\neg \varphi | \varphi \notin \Gamma\}$.

Intuitively, $N$ is rich w.r.t. $\varphi_0$ expresses the condition that for all $\Gamma \in \Omega(\varphi_0)$, if $\Gamma \notin N$ then $\Gamma$ is not consistent.

**Theorem 2. (Main theorem)** If $N \subseteq \Omega(\varphi_0)$ is rich w.r.t. $\varphi_0$, and $\Gamma$ is a key witness in $N = (N, m_N)$, then $\vdash (\bigwedge \Gamma \land \bigwedge T) \rightarrow 0$.

We will delay the proof but first look at how to use the main theorem to decide satisfiability of arbitrary formula and to obtain completeness of the axiomatization.

Let us come back to discuss properties of the sequence $M_0, \ldots, M_n$. It is easy to see that $M_0$ is rich w.r.t. $\varphi_0$, and moreover according to the main theorem every $M_i$ in the sequence above is rich w.r.t. $\varphi_0$. The following lemma also shows that every $M_i$ is a non-empty set together with other properties.

**Lemma 1.** Let $N \subseteq \Omega(\varphi_0)$. If $N$ is rich w.r.t. $\varphi_0$, then $N \neq \emptyset$ and

$$\vdash 1 \leftrightarrow \bigvee \{\bigwedge \Gamma \land \bigwedge T | \Gamma \in N\}.$$

Moreover if $S \subseteq FL(\varphi_0)$, then

$$\vdash \bigwedge S \leftrightarrow \bigvee \{\bigwedge \Gamma \land \bigwedge T | \Gamma \in N, S \subseteq \Gamma\}.$$

**Proof:** First note that

$$\vdash 1 \leftrightarrow \bigwedge \{\varphi \lor \neg \varphi | \varphi \in FL(\varphi_0)\},$$

and also note the following sequence of bi-implications:

$$\vdash \bigwedge \{\varphi \lor \neg \varphi | \varphi \in FL(\varphi_0)\} \leftrightarrow \bigvee \{\bigwedge \Gamma \land \bigwedge T | \Gamma \in \Omega(\varphi_0)\},$$

$$\vdash \bigvee \{\bigwedge \Gamma \land \bigwedge T | \Gamma \in \Omega(\varphi_0)\} \leftrightarrow \bigvee \{\bigwedge \Gamma \land \bigwedge T | \Gamma \in \Omega(\varphi_0), \Gamma \in N\} \lor \bigvee \{\bigwedge \Gamma \land \bigwedge T | \Gamma \in \Omega(\varphi_0), \Gamma \notin N\}.$$ 

Since $N$ is rich w.r.t. $\varphi_0$, we have $\vdash \bigwedge \Gamma \land \bigwedge T \rightarrow 0$ for those $\Gamma \notin N$. Thus

$$\bigvee \{\bigwedge \Gamma \land \bigwedge T | \Gamma \in \Omega(\varphi_0), \Gamma \notin N\} \rightarrow 0,$$
and from the bi-implication above we have
\[ \vdash \bigvee \{ \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in \Omega(\varphi_0) \} \leftrightarrow \bigvee \{ \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in \Omega(\varphi_0), \Gamma \in N \}. \]

Now take the bi-implications together we obtain:
\[ \vdash 1 \leftrightarrow \bigvee \{ \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in \Omega(\varphi_0), \Gamma \in N \}. \]

\( N \) cannot be empty, since otherwise the above bi-implication would give \( \vdash 1 \leftrightarrow 0 \), which is impossible.

From the last bi-implication it is easy to see that
\[ \vdash (\bigwedge S) \leftrightarrow (\bigwedge S \land (\bigvee \{ \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N \})). \]

Note the following sequence of bi-implications:
\[ \vdash (\bigwedge S \land (\bigvee \{ \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N \})) \leftrightarrow (\bigvee \{ \bigwedge S \land \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N \}), \]
\[ \vdash (\bigvee \{ \bigwedge S \land \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N \}) \leftrightarrow (\bigvee \{ \bigwedge S \land \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N, S \subseteq \Gamma \}) \]
\[ \lor \bigvee \{ \bigwedge S \land \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N, S \not\subseteq \Gamma \}. \]

Now note that if \( S \subseteq \Gamma \) then \( \bigwedge S \land \bigwedge \Gamma \leftrightarrow \bigwedge \Gamma \), and if \( S \not\subseteq \Gamma \) then there exists \( \psi \in S \) with \( -\psi \in T \), thus
\[ \vdash \bigwedge S \land \bigwedge T \leftrightarrow 0, \]

thus
\[ \vdash (\bigvee \{ \bigwedge S \land \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N, S \subseteq \Gamma \}) \leftrightarrow (\bigvee \{ \bigwedge S \land \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N, S \not\subseteq \Gamma \}), \]

and
\[ \vdash (\bigvee \{ \bigwedge S \land \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N, S \not\subseteq \Gamma \}) \to 0, \]

so
\[ \vdash (\bigvee \{ \bigwedge S \land \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N, S \subseteq \Gamma \}) \]
\[ \lor \bigvee \{ \bigwedge S \land \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N, S \not\subseteq \Gamma \} \leftrightarrow (\bigvee \{ \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N, S \subseteq \Gamma \})). \]

Now with all the bi-implications we obtain:
\[ \vdash S \leftrightarrow \bigvee \{ \bigwedge \Gamma \land \bigwedge T \mid \Gamma \in N, S \subseteq \Gamma \}. \]

\[ \square \]

The following property of \( M_n \) easily leads to the wanted decidability and completeness results.

**Theorem 3.** Let \( \psi \in FL(\varphi_0) \), then the following conditions are equivalent:

1. \( \psi \) is not satisfiable;
2. \( \vdash \neg \psi \);
3. whenever \( \Gamma \in M_n \) then \( \psi \notin \Gamma \).

\( \psi \) is satisfiable if and only if there exists \( \Gamma \in M_n \) such that \( \psi \in \Gamma \).

**Proof:** The implication from 2 to 1 follows from the soundness of the axiomatization.

1 \( \Rightarrow \) 3: Suppose that there exists \( \Gamma \in M_n \) such that \( \psi \in \Gamma \). Since \( \mathfrak{M}_n = (M_n, m_{\mathfrak{M}_n}) \) is canonical, thus in this case \( \psi \in m_{\mathfrak{M}_n}(\psi) \), in other words \( \mathfrak{M}_n, \Gamma \models \psi \), so \( \psi \) is satisfiable.

3 \( \Rightarrow \) 2: Suppose whenever \( \Gamma \in M_n \) then \( \psi \notin \Gamma \). According to lemma 1

\[ \vdash \psi \leftrightarrow \bigvee \{ \bigwedge \Gamma \wedge \bigwedge \mathcal{T} \mid \Gamma \in M_n, \psi \in \Gamma \}, \]

and in this case the right hand side of the above bi-implication is an empty disjunction, thus \( \vdash \psi \leftrightarrow 0 \), and \( \vdash \neg \psi \).

This theorem easily gives us a procedure to decide if a given formula \( \psi \) is satisfiable: Construct the sequence \( M_0, \ldots, M_n \) with \( \varphi_0 = \psi \), and then check if there exists \( \Gamma \in M_n \) s.t. \( \psi \in \Gamma \).

Now we can obtain the completeness of the axiomatization.

**Theorem 4.** Let \( \psi \in \Phi \). If \( \models \psi \) then \( \vdash \psi \).

**Proof:** Suppose \( \models \psi \). Take \( \varphi_0 \equiv \neg \psi \) and construct the sequence \( M_0, \ldots, M_n \).

Obviously \( \neg \psi \in FL(\varphi_0) \) and \( \neg \psi \) is not satisfiable, thus by the last theorem \( \vdash \neg (\neg \psi) \), so \( \vdash \psi \).

\[ \square \]

## 4 Proof of main theorem

In this section we prove theorem 2, i.e. if \( N \subseteq O(\varphi_0) \) is rich w.r.t. \( \varphi_0 \), and \( \Delta \) is a key witness in \( \mathfrak{M} = (N, m_{\mathfrak{M}}) \), then \( \vdash (\wedge \Delta \wedge \bigwedge \overline{\Delta}) \rightarrow 0 \).

According to the definition of key witness there must exist a minimal witness \( \psi \in FL(\varphi_0) \) satisfies the following:

1. either \( \Delta \in m_{\mathfrak{M}}(\psi) \) and \( \psi \notin \Delta \), or \( \Delta \notin m_{\mathfrak{M}}(\psi) \) and \( \psi \in \Delta \);
2. for \( \varphi \in FL(\varphi_0) \), if \( \varphi \) is a sub-formula of \( \psi \) and \( \varphi \neq \psi \), then \( m_{\mathfrak{M}}(\varphi) = \{ \Gamma \mid \varphi \in \Gamma \} \).

Here we will establish \( \vdash (\wedge \Delta \wedge \bigwedge \overline{\Delta}) \rightarrow 0 \) under all possible structures of \( \psi \).

First note that \( \psi \) cannot be an atomic proposition \( p \), since according to the construction of \( \mathfrak{M} = (N, m_{\mathfrak{M}}) \), \( m_{\mathfrak{M}}(p) = \{ \Gamma \mid p \in \Gamma \} \), which contradicts condition 1.

When \( \psi \equiv 0 \), since \( m_{\mathfrak{M}}(0) = \emptyset \), condition 1 implies \( 0 \in \Delta \), so in this case \( \vdash \wedge \Delta \rightarrow 0 \), thus \( \vdash (\wedge \Delta \wedge \bigwedge \overline{\Delta}) \rightarrow 0 \).

When \( \psi \equiv \varphi_1 \rightarrow \varphi_2 \), according to condition 2, \( m_{\mathfrak{M}}(\varphi_1) = \{ \Gamma \mid \varphi_1 \in \Gamma \} \) and \( m_{\mathfrak{M}}(\varphi_2) = \{ \Gamma \mid \varphi_2 \in \Gamma \} \). We have two cases to discuss, i.e. \( \Delta \notin m_{\mathfrak{M}}(\varphi_1 \rightarrow \varphi_2) \) with \( \varphi_1 \rightarrow \varphi_2 \notin \Delta \), and \( \Delta \in m_{\mathfrak{M}}(\varphi_1 \rightarrow \varphi_2) \) with \( \varphi_1 \rightarrow \varphi_2 \notin \Delta \).
Case $\Delta \not\in m_\Pi(\varphi_1 \to \varphi_2)$ and $\varphi_1 \to \varphi_2 \in \Delta$. In this case it must be that $\Delta \in m_\Pi(\varphi_1)$ and $\Delta \not\in m_\Pi(\varphi_2)$, which implies $\varphi_1 \in \Delta, \varphi_2 \not\in \Delta$. Thus $\vdash (\wedge \Delta) \to (\varphi_1 \to \varphi_2)$, $\vdash (\wedge \Delta) \to \varphi_1$, and $\vdash (\wedge \Delta) \to \neg \varphi_2$, and then it is easy to see that $\vdash (\wedge \Gamma \land \wedge \Delta) \to 0$.

Case $\Delta \in m_\Pi(\varphi_1 \to \varphi_2)$ and $\varphi_1 \to \varphi_2 \not\in \Delta$. In this case, from $\varphi_1 \to \varphi_2 \not\in \Delta$ follows $\vdash (\wedge \Delta) \to \neg (\varphi_1 \to \varphi_2)$. Here we discuss two subcases for $\varphi_1 \not\in \Delta$ and $\varphi_1 \in \Delta$. If $\varphi_1 \not\in \Delta$, then $\vdash (\wedge \Delta) \to \neg \varphi_1$, together with $\vdash (\wedge \Delta) \to 0$, so $\vdash (\wedge \Gamma \land \wedge \Delta) \to 0$. If $\varphi_1 \in \Delta$, then $\Delta \in m_\Pi(\varphi_1)$, thus $\Delta \in m_\Pi(\varphi_2)$ since $\Delta \in m_\Pi(\varphi_1 \to \varphi_2)$, thus $\varphi_2 \in \Delta$, so $\vdash (\wedge \Delta) \to (\varphi_1 \wedge \varphi_2)$, together with $\vdash (\wedge \Delta) \to \neg (\varphi_1 \to \varphi_2)$ we also obtain $\vdash (\wedge \Delta \land \wedge \Delta) \to 0$.

When $\psi \equiv [\alpha] \varphi$, according to condition 2, $m_\Pi(\varphi) = \{\Gamma \mid \varphi \in \Gamma\}$. Here we also have two cases to discuss, i.e. $[\alpha] \varphi \not\in \Delta$ with $\Delta \in m_\Pi([\alpha] \varphi)$, and $[\alpha] \varphi \in \Delta$ with $\Delta \not\in m_\Pi([\alpha] \varphi)$.

Case $[\alpha] \varphi \not\in \Delta$ and $\Delta \in m_\Pi([\alpha] \varphi)$. We prove the following fact in this case:

**Fact A:** for all $\beta \in \Pi, \Gamma \in N$, if $\beta$ is a sub-program of $\alpha$ then

$$\vdash \wedge \Gamma \to [\beta] \bigvee \{\wedge \Gamma' \mid (\Gamma, \Gamma') \in m_\Pi(\beta)\}.$$ 

If Fact A is proved, then apply it to $\alpha$ and $\Delta$ we have

$$\vdash \wedge \Delta \to [\alpha] \bigvee \{\wedge \Gamma' \mid (\Delta, \Gamma') \in m_\Pi(\alpha)\}.$$ 

Since $\Delta \in m_\Pi([\alpha] \varphi)$, whenever $(\Delta, \Gamma') \in m_\Pi(\alpha)$ then $\Gamma' \in m_\Pi(\varphi)$, so $\varphi \in \Gamma'$ and $\vdash \wedge \Gamma' \to \varphi$. Thus

$$\vdash \bigvee \{\wedge \Gamma' \mid (\Delta, \Gamma') \in m_\Pi(\alpha)\} \to \varphi,$$

and

$$\vdash [\alpha] \bigvee \{\wedge \Gamma' \mid (\Delta, \Gamma') \in m_\Pi(\alpha)\} \to [\alpha] \varphi,$$

and

$$\vdash \wedge \Delta \to [\alpha] \varphi.$$ 

On the other hand, with $[\alpha] \varphi \not\in \Delta$ we have

$$\vdash \wedge \Delta \to \neg [\alpha] \varphi,$$

thus

$$\vdash (\wedge \Delta \land \wedge \Delta) \to 0.$$ 

Now we prove Fact A by induction on the structure of $\beta$. Here we show some key cases.

In the case that $\beta$ is an atomic program $a$, we have to show

$$\vdash \wedge \Gamma \to [a] \bigvee \{\wedge \Gamma' \mid (\Gamma, \Gamma') \in m_\Pi(\alpha)\},$$
which can be done as follows. Clearly
\[ \vdash \bigwedge \Gamma \rightarrow \bigwedge \{ [a] \phi \mid [a] \phi \in \Gamma \}, \]
and by Axiom (iii)
\[ \vdash \bigwedge \{ [a] \phi \mid [a] \phi \in \Gamma \} \rightarrow [a] \bigwedge \{ \phi \mid [a] \phi \in \Gamma \}. \]
Then by Lemma 1
\[ \vdash \bigwedge \{ \phi \mid [a] \phi \in \Gamma \} \leftrightarrow \bigvee \{ \bigwedge \Gamma' \mid \{ \phi \mid [a] \phi \in \Gamma \} \subseteq \Gamma', \Gamma' \in N \}. \]
Note that by the definition of \( m_\Omega(a) \), \( \{ \phi \mid [a] \phi \in \Gamma \} \subseteq \Gamma' \) if and only if \( (\Gamma, \Gamma') \in m_\Omega(a) \), we obtain
\[ \vdash \bigwedge \Gamma \rightarrow [a] \bigvee \{ \bigwedge \Gamma' \mid (\Gamma, \Gamma') \in m_\Omega(a) \}. \]
In the case for \( \beta_1; \beta_2 \), we have to show
\[ \vdash \bigwedge \Gamma \rightarrow [\beta_1; \beta_2] \bigvee \{ \bigwedge \Gamma' \mid (\Gamma, \Gamma') \in m_\Omega(\beta_1; \beta_2) \}, \]
which can be done as follows. By the induction hypothesis,
\[ \vdash \bigwedge \Gamma \rightarrow [\beta_1] \bigvee \{ \bigwedge \Gamma'' \mid (\Gamma, \Gamma'') \in m_\Omega(\beta_1) \} \quad (1) \]
and for each \( \Gamma'' \) such that \( (\Gamma, \Gamma'') \in m_\Omega(\beta_1) \), by the induction hypothesis
\[ \vdash \bigwedge \Gamma'' \rightarrow [\beta_2] \bigvee \{ \bigwedge \Gamma' \mid (\Gamma'', \Gamma') \in m_\Omega(\beta_2) \} \quad (2) \]
Since \( (\Gamma, \Gamma'') \in m_\Omega(\beta_1) \), so \( (\Gamma'', \Gamma') \in m_\Omega(\beta_2) \) implies \( (\Gamma, \Gamma') \in m_\Omega(\beta_1; \beta_2) \), thus
\[ \vdash [\beta_2] \bigvee \{ \bigwedge \Gamma' \mid (\Gamma'', \Gamma') \in m_\Omega(\beta_2) \} \rightarrow [\beta_2] \bigvee \{ \bigwedge \Gamma' \mid (\Gamma, \Gamma') \in m_\Omega(\beta_1; \beta_2) \} \quad (3) \]
take (2), (3) together, so for each \( \Gamma'' \) with \( (\Gamma, \Gamma'') \in m_\Omega(\beta_1) \)
\[ \vdash \bigwedge \Gamma'' \rightarrow [\beta_2] \bigvee \{ \bigwedge \Gamma' \mid (\Gamma, \Gamma') \in m_\Omega(\beta_1; \beta_2) \} \]
so
\[ \vdash \bigvee \{ \bigwedge \Gamma'' \mid (\Gamma, \Gamma'') \in m_\Omega(\beta_1) \} \rightarrow [\beta_2] \bigvee \{ \bigwedge \Gamma' \mid (\Gamma, \Gamma') \in m_\Omega(\beta_1; \beta_2) \}. \]
Then first apply rule (GEN) then use axiom (ii) obtain
\[ \vdash [\beta_1] \bigvee \{ \bigwedge \Gamma'' \mid (\Gamma, \Gamma'') \in m_\Omega(\beta_1) \} \rightarrow [\beta_1][\beta_2] \bigvee \{ \bigwedge \Gamma' \mid (\Gamma, \Gamma') \in m_\Omega(\beta_1; \beta_2) \} \quad (4) \]
and by axiom (v)

\[ \vdash [\beta_1][\beta_2] \bigvee \{ \bigwedge \Gamma'' \mid (\Gamma, \Gamma'') \in m_N(\beta_1; \beta_2) \} \equiv [\beta_1; \beta_2] \bigvee \{ \bigwedge \Gamma'' \mid (\Gamma, \Gamma'') \in m_N(\beta_1; \beta_2) \}. \]

Take this with (1) and (4) we obtain what we want.

In the case of \( \beta^* \), we have to show

\[ \vdash \bigwedge \Gamma' \rightarrow [\beta^*] \bigvee \{ \bigwedge \Gamma'' \mid (\Gamma', \Gamma'') \in m_N(\beta^*) \}, \]

which can be done as follows. Let \( S = \{ \Gamma'' \mid (\Gamma', \Gamma'') \in m_N(\beta^*) \} \), then for all \( \Gamma'' \in S \) by the induction hypothesis,

\[ \vdash \bigwedge \Gamma' \rightarrow [\beta] \bigvee \{ \bigwedge \Gamma'' \mid (\Gamma', \Gamma'') \in m_N(\beta) \} \quad (5) \]

since \( \Gamma' \in S \), it is easy to see that \( (\Gamma', \Gamma'') \in m_N(\beta) \) implies \( \Gamma'' \in S \), so

\[ \vdash [\beta] \bigvee \{ \bigwedge \Gamma'' \mid (\Gamma', \Gamma'') \in m_N(\beta) \} \rightarrow [\beta] \bigvee \{ \bigwedge \Gamma'' \mid \Gamma'' \in S \} \quad (6) \]

thus for all \( \Gamma' \in S \), take (5),(6) we obtain

\[ \vdash \bigwedge \Gamma' \rightarrow [\beta] \bigvee \{ \bigwedge \Gamma'' \mid \Gamma'' \in S \}. \]

So

\[ \vdash \bigvee \{ \bigwedge \Gamma'' \mid \Gamma'' \in S \} \rightarrow [\beta] \bigvee \{ \bigwedge \Gamma'' \mid \Gamma'' \in S \}. \]

Let \( \psi \equiv \bigvee \{ \bigwedge \Gamma'' \mid \Gamma'' \in S \} \), so far we have shown \( \vdash \psi \rightarrow [\beta] \psi \), using (GEN) we obtain

\[ \vdash [\beta^*](\psi \rightarrow [\beta] \psi). \]

Obviously \( \vdash \bigwedge \Gamma \rightarrow \psi \), using axiom (viii) \( \vdash \psi \land [\beta^*](\psi \rightarrow [\beta] \psi) \rightarrow [\beta^*] \psi \) we obtain

\[ \vdash \bigwedge \Gamma \rightarrow [\beta^*] \psi, \]

which is what we want to show in this case.

**Case** \( [\alpha] \varphi \in \Delta \) and \( \Delta \not\in m_N([\alpha] \varphi) \). In this case we prove the following fact:

**Fact B**: for all \( \beta \in \Pi \), for all \( \rho \in \Phi \), if \( \beta \) is a sub-formula of \( \alpha \), and \( [\beta] \rho \in \Delta \), and \( \Delta \not\in m_N([\beta] \rho) \) then

\[ \vdash (\bigwedge \Delta \land \bigwedge \Box) \rightarrow 0. \]

When this is done then let \( \beta = \alpha \) we obtain what we want.

We prove **Fact B** by induction on the structure of \( \beta \). The case for \( \alpha \) is easy.

In the case of \( \beta_1 \cup \beta_2 \), \( \Delta \not\in m_N([\beta_1 \cup \beta_2] \rho) \) implies either (1) \( \Delta \not\in m_N([\beta_1] \rho) \) or (2) \( \Delta \not\in m_N([\beta_2] \rho) \).

In case (1), since \( [\beta_1 \cup \beta_2] \rho \in \Delta \) implies

\[ \vdash \bigwedge \Delta \rightarrow [\beta_1 \cup \beta_2] \rho, \]
thus using axiom (iv) to obtain \( \vdash \bigwedge \Delta \rightarrow [\beta_1]\rho \). Now if \([\beta_1]\rho \notin \Delta \) then

\[ \vdash \bigwedge \Delta \rightarrow -[\beta_1]\rho, \]

thus

\[ \vdash (\bigwedge \Delta \land \bigwedge \overline{\Delta}) \rightarrow 0. \]

If \([\beta_1]\rho \in \Delta \), since \(\beta_1\) is a sub-program of \(\beta_1; \beta_2\), with \(\Delta \notin m_{\beta_1}([\beta_1]\rho)\) by the induction hypothesis we also have

\[ \vdash (\bigwedge \Delta \land \bigwedge \overline{\Delta}) \rightarrow 0. \]

In case (2) it can be proved in the same way.

For the case \(\beta_1; \beta_2\), \(\Delta \notin m_{\beta_1}([\beta_1; \beta_2]\rho)\) implies \(\Delta \notin m_{\beta_1}([\beta_1] | [\beta_2] \rho)\). Here we also have two sub-cases to discuss, i.e. (1) \([\beta_1][[\beta_2] \rho \in \Delta \) and (2) \([\beta_1][[\beta_2] \rho \notin \Delta \).

In case (1), since \(\beta_1\) is a sub-program of \(\beta_1; \beta_2\), by the induction hypothesis of Fact B,

\[ \vdash (\bigwedge \Delta \land \bigwedge \overline{\Delta}) \rightarrow 0. \]

In case (2), \([\beta_1][[\beta_2] \rho \notin \Delta \), so

\[ \vdash \bigwedge \overline{\Delta} \rightarrow -[\beta_1][[\beta_2] \rho. \]

On the other hand \([\beta_1; \beta_2] \rho \in \Delta \), so

\[ \vdash \bigwedge \Delta \rightarrow [\beta_1; \beta_2] \rho, \]

thus

\[ \vdash \bigwedge \Delta \rightarrow [\beta_1][[\beta_2] \rho \]

by axiom (v). Thus we also obtain

\[ \vdash (\bigwedge \Delta \land \bigwedge \overline{\Delta}) \rightarrow 0. \]

For the case \(\beta^*\), \(\Delta \notin m_{\beta_1}([\beta^*] \rho)\) implies \(\Delta \notin m_{\beta_1}(\rho)\), or \(\Delta \notin m_{\beta_1}([\beta][\beta^*] \rho)\). In the first case \(\rho \notin \Delta \),

\[ \vdash \bigwedge \overline{\Delta} \rightarrow -\rho. \]

From \([\beta^*] \rho \in \Delta \) we obtain

\[ \vdash \bigwedge \Delta \rightarrow [\beta^*] \rho, \]

and by axiom (vii)

\[ \vdash \bigwedge \Delta \rightarrow (\rho \land [\beta][\beta^*] \rho), \]

so

\[ \vdash (\bigwedge \Delta \land \bigwedge \overline{\Delta}) \rightarrow 0. \]
In the second case $\Delta \not\in m_N(\[\beta\][\beta^*]?\rho)$, the proof is similar.
For the case $\varphi'?\Delta \not\in m_N(\[\varphi'?\]?!\rho)$ implies $\Delta \in m_N(\varphi')$ and $\Delta \not\in m_N(\rho)$, thus $\varphi' \in \Delta$, $\rho \not\in \Delta$, and

$$\vdash \wedge \Delta \rightarrow \varphi', \vdash \wedge \neg \Delta \rightarrow \neg \rho.$$  

On the other hand, from $[\varphi'?]?\rho \in \Delta$ it follows

$$\vdash \wedge \Delta \rightarrow [\varphi'?]?\rho,$$

together with $\vdash \wedge \Delta \rightarrow \varphi'$ and $\vdash \wedge \neg \Delta \rightarrow \neg \rho$, by axiom (vi) it is easy to get

$$\vdash (\wedge \Delta \land \wedge \neg \Delta) \rightarrow 0.$$  

\[\Box\]

5 Conclusion and related works

A well known algorithm for the satisfiability checking problem in PDL is Pratt’s deterministic single exponential algorithm [6], which is further described in page 403 of [7] and in page 213 of [5]. The algorithm works as follows. Starting from the maximum sub-sets of the given formula’s Fischer-Ladner closure, first delete those sets which contradict any of the propositional axioms or any of the axioms about program structures to obtain the so called Hintikka sets. Then iteratively performing elimination of Hintikka sets, that is to delete those sets which are not ”demand-satisfied”, (these sets are likely to be unsatisfiable because the violation of demand-satisfied criteria is a semantic evidence of inconsistency) and finally obtain a canonical model. For a given formula, the algorithm will output a canonical model for it if there exists one.

In this work, we propose an improved method for canonical model construction within PDL. Roughly speaking, the proposed algorithm also starts from the maximum sub-sets of the given formula’s Fischer-Ladner closure, and then delete those sets which are found inconsistent, and in the end to obtain a canonical model. The difference with the classical algorithm is that here, each time when we delete a set, we are able to find inconsistency in the form of a syntactic proof of contradiction in the proof system, not just a kind of semantic inconsistency in the sense that the set is not demand-satisfied. With that we are able to obtain the following two extra benefits than merely deciding whether a formula is satisfiable or not. The first benefit is that using this method we can obtain not only a canonical model for a given formula $\psi$ when the formula is satisfiable, but also a proof of $\vdash \neg \psi$ when the formula is not satisfiable. The second benefit is that we can use this decision procedure to obtain a constructive proof of the completeness of Segerberg’s axiomatization for PDL. Furthermore, since the criteria of demand-satisfied used in the classical algorithm is quite subtle in the sense that it is not immediately obvious that those not demand-satisfied sets are not satisfiable, which leads to a somewhat involving justification in the correctness proof of the algorithm (see Theorem 6.54 in [7]). In our case, by using the
criteria of syntactic inconsistency, the correctness of the new algorithm follows immediately from the main theorem.

Other related works include the completeness proofs of Segerberg’s axiomatization of PDL in [5] and [8]. In [5], the completeness proof of the axiomatization is obtained by filtration of the so-called non-standard model of maximal consistency sets of the entire language. Non-standard model of PDL is a frame where iteration is not guaranteed to be a transitive closure of the argument but only satisfies the induction rule. The construction of non-standard model is only conceptual and not feasible because it deals with infinite numbers of infinite maximal consistency sets and it cannot provide a method to actually construct a proof. The completeness proof in [8] uses maximal consistency sets on Fischer-Ladner closure, obtains a finite domain of discourse. However, it does not suggest a method as how to construct such finite maximal consistency sets, only prove that if maximal consistency set exists for a formula then it has a model. Compared with these two proof, the completeness proof presented here is completely constructive in that for every unsatisfiable formula $\psi$, a proof of $\vdash \neg \psi$ can be constructed in finite steps. A more recent work about satisfiability and completeness of PDL is [9]. In this work Lange and Stirling use a very different approach of focus games to solve the satisfiability problem of PDL. Particularly interesting is that a complete axiom system, different from the Segerberg’s, can easily be read off the game rules.

In [10], we propose a language by extending PDL with simple recursive propositions, which is strictly more expressive than PDL and not more expressive than the single alternation fragment of the modal $\mu$-calculus. Whether we can check the satisfiability of the extended language by constructing canonical model is under consideration. Moreover, we also want to provide a tool for deciding satisfiability for formulas in this extended language.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable suggestions.

References

10. X. Liu and B. Xue, “Specification in PDL with recursion,” in The Proceedings of