CARL VON OSSIETZKY universität OLDENBURG

Fakultät II- Informatik, Wirtschafts- und Rechswissenschaften
Department für Informatik

# Advancing Software Model-Checking by SMT Interpolation Beyond Decidable Arithmetic Theories

An approach to verify safety properties in embedded and hybrid system models

Dissertation zur Erlangung des Grades eines Doktors der Ingenieurwissenschaften

vorgelegt von

M.Sc. Ahmed Mahdi

**Gutachter:**

Prof. Dr. Martin Fränzle
Prof. Dr. Bernd Becker (Albert-Ludwigs Universität Freiburg)

**weitere Mitglieder der Prüfungskommission:**

Prof. Dr. Oliver Theel (Vorsitz)
Dr. Ingo Stierand

Tag der Einreichung: 06.02.2017
Tag der Disputation: 09.08.2017

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

«وَعَلَّمَكَ مَا لَمْ تَكُنْ تَعْلَمُ وَكَانَ فَضْلُ اللّهِ عَلَيْكَ عَظِيمًا»

سورة النساء ـ آية رقم ١١٣

In the Name of Allah, the Most Beneficent, the Most Merciful

*"Allah (The God) has taught you what you did not know before,
and great is Allah's grace upon you."*

The Holy Quran, Chapter 4, Verse 113.

# Dedication

*Dedicated to Yousra,*

...

*Abdul Raouf,*

...

*Ayah,*

...

*Selma and Osama*

# Abstract

Envisage a world where *embedded and hybrid system models* are analysed with scrutiny by algorithms that automatically, efficiently, and unhesitatingly can answer questions about *reachability* and *stability* analyses as well as asserting *safety* of these models. In such a world, embedded and hybrid systems are significantly more reliable than what we currently see in our life; software has fewer bugs and is easily certifiable like other engineering artifacts; software-induced disasters are effortlessly absent; and the billions of euros that are normally spent – over previous decades – on testing and maintenance activities, are instead rushed into more productive enterprises. Alas, such a world is a *fictitious utopia*, since the verification questions mostly translate into *undecidable problems*. Nevertheless, one can still invent algorithms and build tools that can answer some questions about safety most of the time, and this is exactly what is introduced in this dissertation. We advance safety property verification in several directions.

First, a particular variant of safety question asks for methods to accurately and safely detect unreachable code fragments, a.k.a. *dead code*, in arithmetic programs. The reason for doing so is that detecting dead code permits a more rigorous interpretation of coverage based-criteria in testing-based code validation and certification, as recommended in various standards for embedded system development, as well as meeting the demand for absence of dead code imposed by *pertinent standards*, like DO-178C. To do so, we integrate several techniques; namely *Craig interpolation* (CI), *counterexample guided abstraction refinement* (CEGAR), *interval constraint propagation* (ICP), and *conflict-driven clause learning over theories* (CDCL(T)) in one framework. In this framework, CI is the workhorse for abstraction refinement by using stepwise interpolants in lazy abstraction technique as well as for dis-/proving the reachabiity of bad states representing the violation of the safety property. CEGAR is used to ward off *the state space explosion problem* while handling large models. ICP and CDCL(T) are employed to reduce the generally non-linear problem to a satisfiability modulo theory (SMT) problem in the order theory of the reals and to solve extremely large Boolean and arithmetic constraint systems respectively. In order to implement this framework in the iSAT3 model checker, we extend the latter in a way such that it encodes an embedded arithmetic program, inducing a transition system, in its iSAT3 corresponding control flow automaton (CFA) such that the CFA nodes represent the program control points and the CFA edges represent the conditions and changes between the program control points. Then, iSAT3 will generate an *adequate abstraction* of this CFA model and verify the unreachability of unsafe property representing the unsafe states by finding a *safe invariant* that overapproximates the reachable states of the verified model and on the same time does not intersect with unsafe states. We verify several benchmarks by using our new implementation, where benchmarks indicate superior performance of the new CEGAR approach on non-linear benchmarks, even in floating points dominated C-programs where IEEE 754 standard is supported in the proposed framework.

Second, motivated by the practical need for verifying probabilistic hybrid systems involving

linear, polynomial, and transcendental arithmetic, we go beyond *stochastic boolean satisfiability problems* (SSAT) by defining a notion of Generalized Craig Interpolant (GCI) for the *stochastic satisfiability modulo theories* (SSMT), and introduce a mechanism to compute such stochastic interpolants for non-polynomial SSMT problems based on *a sound and relatively complete resolution calculus.* The new notion of Craig interpolation can handle unbounded probabilistic reachability and stability problems in a *probabilistic hybrid automaton.*

Finally, optimising the verification process in terms of improving the time consumption and memory usage while verifying *assumption-commitment specifications*, is an industrial quest. An assumption-commitment specification is a *contract* where the commitment is required to hold if the assumption holds as well. Our approach improves the verification process by pruning the state space of the model where *the assumption is violated.* This exclusion is performed by admissible *transformation functions* which are defined based on a *new notion of edges supporting* a property. Our approach applies to *computational models* which range from finite automata to hybrid ones. This technique was evaluated by verifying several case studies in Uppaal.

# Zusammenfassung[1]

Man stelle sich eine Welt vor, in der Modelle eingebetteter und hybrider Systeme genauestens durch Algorithmen untersucht werden, welche automatisch und effizient Fragen bezüglich ihrer Erreichbarkeit, Stabilität und Sicherheit beantworten können. In solch einer Welt wären eingebettete und hybride Systeme signifikant zuverlässiger als in unserer gegenwärtigen Situation; Software würde weniger Programmierfehler aufweisen und wäre - wie andere technische Artefakte - leichter zu zertifizieren; softwarebedingte Unfälle ließen sich mühelos vermeiden; und die bisher hohen Ausgaben für Tests und Wartung könnten gespart und für produktivere Aktivitäten verwendet werden. Leider ist solch eine Welt eine utopische Vorstellung, denn die meisten Verifikationsaufgaben stellen unentscheidbare Probleme dar. Dennoch kann man Algorithmen und Werkzeuge einführen, welche bestimmte Sicherheitsfragen in den meisten Fällen beantworten, und genau das ist Gegenstand dieser Dissertation.

Als erstes erfordert eine bestimmte Variante von Sicherheitsfragen Methoden zur Entdeckung sogenannten toten Codes (engl. dead code) in arithmetischen Programmen. Die Entdeckung toten Codes ermöglicht die präzisere Interpretation von Codeabdeckungskriterien in der testbasierten Softwarevalidierung und -zertifizierung, wie sie in verschiedenen Standards für die Entwicklung eingebetteter Systeme gefordert wird. Zudem erfült dies die Anforderung für Abwesenheit von totem Code in einschlägigen Standards wie DO-178C. Dazu werden verschiedene Techniken in einem Framework kombiniert, und zwar: (Craig Interpolation) CI, (counterexample guided abstraction refinement) CEGAR, (interval constraint propagation) ICP und (conflict-driven clause learning over theories) CDCL(T). In dem Framework fungiert CI als das Zugpferd der Abstraktionsverfeinerung und setzt dazu schrittweise Interpolanten zur verzögerten Abstraktion und zum Auffinden sog. bad states, welche eine Verletzung der Sicherheitseigenschaft darstellen, ein. CEGAR dient der Vermeidung der Explosion des Zustandsraumes im Umgang mit großen Modellen. ICP und CDCL(T) werden zur Reduzierung des nicht-linearen Problems zu einem Erfüllbarkeits-modulo-Theorie-Problem in der order theory of the reals und zur Lösung extrem großer boolescher bzw. arithmetischer Constraint Systeme benutzt. Zur Umsetzung des Frameworks wird der model checker iSAT3 erweitert, sodass er ein eingebettetes arithmetisches Program, welches ein Transitionssystem induziert, als (control flow automaton) CFA kodiert, in dem die Knoten die Kontrollpunkte des Programms und die Kanten die Bedingungen und Änderungen zwischen den Kontrollpunkten repräsentieren. So generiert iSAT3 eine adäquate Abstraktion des CFA Modells und verifiziert die Unerreichbarkeit der Unsicherheitseigenschaft, indem unsichere Zustände mittels einer sicheren Invariante repräsentiert werden, welche die erreichbaren Zustände des zu verifizierenden Modelles ohne Überschneidung mit unsicheren Zuständen überapproximiert. Mithilfe der neuen Implementierung werden verschiedene Benchmarks verifiziert, in denen eine Performanceverbesserung der neuen CEGAR Heuristik in nichtlinearen Benchmarks – auch in dem von

[1] "Abstract" in German

Gleitkomma dominierten C-Programen entsprechend dem Standard IEEE 754 unterstützt wird – verzeichnet wird.

Zweitens, motiviert durch die praktische Notwendigkeit der Verifikation von probabilistischen hybriden Systemen welche lineare, polynomielle und transzendentale Arithmetik enthalten, gehen wir über (*stochastic boolean satisfiability problems*) SSAT hinaus, indem wir einen Begriff von (Generalized Craig Interpolant) GCI für den (*stochastic satisfiability modulo theories*) SSMT definieren und führen einen Mechanismus ein, um solche stochastischen Interpolanten für nicht-polynomische SSMT-Probleme auf der Grundlage von einem korrekt und relativ zu der darunterliegenden Theorie vollsändigen resolution calculus zu berechnen. Der neue Begriff der Craigschen Interpolation kann unbeschränkte probabilistische Erreichbarkeits- und Stabilitätsprobleme in einem *probabilistischen hybriden Automaten* behandeln.

Schließlich hat die Optimierung des Verifikationsprozesses hinsichtlich Performanz und Speicherplatzbedarf bei der Verifikation von Assumption-Commitment Spezifikationen eine hohe industrielle Relevanz. Eine Assumption-Commitment Spezifikation ist ein Vertrag, bei dem eine Verpflichtung (commitment) erfüllt sein muss, sofern die Annahme (assumption) gegeben ist. Unser Ansatz verbessert den Verifikationsprozess durch Beschneidung des Zustandsraumes an der Stelle, wo eine Annahme verletzt wird. Der Ausschluss erfolgt mittels zulässiger Transformationsfunktionen, welche auf einem neuen Begriff von *eigenschafstunterstützenden* Kanten basieren. Der vorgestellte Ansatz ist anwendbar bei verschiedenen Rechenmodellen, von endlichen Automaten bis hin zu hybriden Modellen. Die Technik wurde durch Verifikation mehrerer Fallstudien in Uppaal evaluiert.

# Acknowledgements

All Praise and thanks be to Allah for the strengths and His blessing in completing this thesis after all the challenges and difficulties. I praise and thank Him, ask Him for His help and forgiveness, and we seek refuge in Allah from the evils of our souls and the mischiefs of our deeds. He whom Allah guides will not be misled, and he whom Allah misleads will never have a guide.

This thesis has been kept on track and been seen through to completion with the support and encouragement of numerous people including my well wishers, my friends, colleagues and various institutions. At this point, I would like to thank all those people who made this thesis possible and an unforgettable experience for me.

I'd like to express my sincere appreciation to my PhD-thesis main supervisor, *Martin Fränzle*, for his valuable assistance, inspiration, and guidance he has dedicated to me all through this thesis especially in recognition of his patience for answering all questions and engaging into long discussions.

I am furthermore very grateful to *Bernd Becker* for the valuable collaboration during AVACS project as well as for his willingness of being my co-examiner and for the friendly atmosphere he brought to my thesis defence.

I would like to thank *Oliver Theel* and *Ingo Stierand* for serving on my thesis committee and for taking the time to get involved with the ideas presented in this thesis.

I'm also grateful to *Karsten Scheibler* for too long discussions, and the numerous instances of assistance given to me especially in dealing with the iSAT3 tool. Also, I'd like to thank *Felix Neubauer* for assistance given to me especially in dealing with the SMI2ISAT tool.

Nearly last, but by no means least, I'd like to thank my master-study supervisor *Bernd Westphal* who was one of the main reasons for me to put me on the road to study formal methods and verification techniques. He supported, encouraged and led me from 2009 till 2012. Additionally, he recommended me to do my PhD under Martin's supervision.

Most of all, there are five persons to whom my gratefulness is never-ending: my parents, *Yousra* and *Abdul Raouf*, for taking care of me in so many different aspects of life and for their limitless encouragement and patience. They have been there for me with all they have got in every moment of the past thirty-two years. This humble dissertation is dedicated to you. Finally, my heartfelt gratitude to my beloved wife *Ayah* and my children *Selma* and *Osama*. No words can describe the love and emotions I have for you. Your support and love mean the world to me.

# Contents


**Dedication** v

**Abstract** vii

**Zusammenfassung** x

**Acknowledgements** xiii

**Contents** xiv

**List of Figures** xvii

**List of Tables** xx

**List of Abbreviations** xxii

**List of Symbols** xxiv

**1 Introduction** 1
- 1.1 Motivation . . . 1
- 1.2 (Partial) History of embedded and (probabilistic) hybrid systems verification 3
  - 1.2.1 Verification of embedded systems . . . 4
  - 1.2.2 Verification of (probabilistic) hybrid systems . . . 5
- 1.3 Challenges and contributions . . . 5
- 1.4 Organization of this dissertation . . . 9

**2 Reachability Analysis** 11
- 2.1 Preface . . . 11
- 2.2 Different terminologies for reachability analysis . . . 13
- 2.3 Classical vs. probabilistic reachability . . . 14

**3 Model Slicing** 16
- 3.1 Problem statement . . . 17
  - 3.1.1 Motivation . . . 17
  - 3.1.2 Related work: . . . 18
- 3.2 Preliminaries . . . 20
- 3.3 Assumption-commitment specifications . . . 23
- 3.4 Model element-based slicing technique . . . 24
- 3.5 Transformation functions . . . 28
  - 3.5.1 Admissible transformations . . . 28
  - 3.5.2 Semi-admissible transformations . . . 31

3.6 New reachability concept: supporting edges . . . 34
3.6.1 Supporting edges . . . 34
3.6.2 Supporting edges and transformation functions . . . 36
3.6.3 Verification based on support-notion . . . 37
3.7 Compositional verification . . . 40
3.8 Case studies . . . 42
3.8.1 Wireless sensor network: Alarm system . . . 42
3.8.2 Fischer's mutual exclusion protocol . . . 49

**4 Dead Code Detection** **58**
4.1 Problem statement . . . 59
4.1.1 Motivation . . . 59
4.1.2 Related work . . . 61
4.1.3 Example . . . 67
4.2 Preliminaries . . . 69
4.2.1 Control flow automaton . . . 69
4.2.2 Craig interpolation: theory and application . . . 72
4.2.3 Interpolation-based model checking (ITP) . . . 73
4.2.4 Counterexample guided abstraction refinement: theory and application . . . 76
4.3 The iSAT3 model checker . . . 77
4.3.1 Syntax and semantics . . . 77
4.3.2 iSAT3 architecture and engines . . . 78
4.3.3 iSAT3 interpolants . . . 80
4.3.4 BMC problems in iSAT3 . . . 94
4.3.5 CFA problems in iSAT3 . . . 96
4.4 Interpolation-based CEGAR technique . . . 97
4.4.1 Interpolation-based refinement procedure in iSAT3: algorithm . . . 97
4.4.2 Example . . . 106
4.4.3 Case studies . . . 107
4.5 Handling floating points dominated C-programs – experiments in industrial-scale . . . 111
4.5.1 Floating point arithmetic due to IEEE 754 . . . 111
4.5.2 Floating points in iSAT3 . . . 112
4.5.3 Floating point arithmetic in iSAT3 with CEGAR . . . 112
4.5.4 Industrial case studies . . . 114
4.5.5 Converting SMI code to iSAT3-CFG input language . . . 115
4.5.6 BTC-ES benchmarks . . . 117

**5 Generalized Craig Interpolation for SSMT** **122**
5.1 Introduction . . . 123
5.1.1 Motivation . . . 123
5.1.2 Related work . . . 124
5.2 Stochastic Satisfiability Modulo theories (SSMT) . . . 124
5.2.1 SSMT: syntax . . . 125
5.2.2 SSMT: semantics . . . 125
5.2.3 SSMT: illustrative example . . . 126

5.2.4 Complexity of SSMT . . . 127
5.2.5 Structure of SSMT formula . . . 127
5.3 Resolution Calculus for SSMT . . . 129
5.3.1 Resolution rules for SSMT . . . 129
5.3.2 Soundness and completeness of SSMT-resolution . . . 131
5.3.3 Example of applying SSMT-resolution . . . 133
5.4 Generalized Craig interpolation for SSMT . . . 134
5.4.1 Generalized Craig Interpolants . . . 135
5.4.2 Computation of Generalized Craig Interpolants – Púdlak's rules extension . . . 136
5.5 Interpolation-based probabilistic bounded model checking . . . 142
5.5.1 Probabilistic bounded reachability – probabilistic safety analysis . . . 143
5.5.2 SSMT encoding scheme for PHAs . . . 144
5.5.3 PBMC solving by means of generalized Craig interpolation . . . 144
5.5.4 Interpolation-based approach for reachability . . . 146
5.5.5 Generalized Craig interpolation for Stability analysis . . . 151

**6 Conclusion** **155**
6.1 Achievements of this dissertation . . . 155
6.2 Outlook . . . 158
6.2.1 Applying transformation for models admitting system modes . . . 158
6.2.2 Extending iSAT3-CFG with interprocedural calls . . . 158
6.2.3 Computing loop summaries – maximum number of while-loop unwindings . . . 159
6.2.4 Integrating generalized Craig interpolation with DPLL-based SSMT solving . . . 159

**Appendix A** **162**

**Bibliography** **171**

**Index** **197**

# List of Figures


1.1 The major contributions of this dissertation and the dependencies between them. The cut in the right upper corner separates stochastic reachability from classical one. . . . . . 6

2.1 Forward and backwards reachability analyses. . . . . . 12

3.1 A satisfaction relation between an automaton and specification $\Box x = 0$. . . . 24
3.2 List of interesting cases for Theorem 3.1. . . . . . 27
3.3 Support notions in timed automaton. . . . . . 34
3.4 Transformed timed automata models after considering different notions of supporting. . . . . . 38
3.5 Example of wireless fire alarm system topology. . . . . . 43
3.5 Uppaal model of WFAS as in [AWD+14], however sensor model in Figure 3.5g extended by Call-messages behaviour. The thick edges represent Call-message scenarios. . . . . . 46
3.6 Sensor automaton after applying transformation function. The other automata remain the same. . . . . . 47
3.7 Results of verifying well-functioning property in WFAS model. . . . . . 49
3.8 Uppaal model of the Fischer's protocol with **direct-fault** detection. . . . . 50
3.9 Uppaal model of the Fischer's protocol with **delayed-fault** detection. . . . 51
3.10 Uppaal model of Fischer's protocol **after applying redirecting** transformation function for the model with direct fault detection in Figure 3.8. . . . 52
3.11 Uppaal model of Fischer's protocol **after applying removing** transformation function for the model with direct fault detection in Figure 3.8. . . . . 52
3.12 Uppaal model of Fischer's protocol **after applying redirecting** transformation function for the model with delayed fault detection in Figure 3.9. . . 53
3.13 Uppaal model of Fischer's protocol **after applying removing** transformation function for the model with delayed fault detection in Figure 3.9. . . . 53
3.14 Results of verifying mutual exclusion in Fisher's protocol with direct detection. 54
3.15 Results of verifying mutual exclusion in Fisher's protocol with delayed detection. . . . . . 55

4.1 Left: An arithmetic program, middle: corresponding control flow graph, right: encoding in iSAT3-CFG format. . . . . . 68
4.2 Bounded model checking and computing post-image by interpolation. . . . 73
4.3 Different interpolant computing approaches. . . . . . 75
4.4 Guiding decide step in iSAT3 affects the resolution tree. . . . . . 82
4.5 Guiding deduce step in iSAT3 affects the resolution tree. . . . . . 83
4.6 Possible influences between deduce and decide steps in iSAT3. . . . . . 84
4.7 Two disjoint circles and two different interpolants with sufficient slackness. . 87

4.8 Two disjoint spheres and two different interpolants with sufficient slackness. 88
4.9 Two disjoint connected-circles and two different interpolants with sufficient slackness. . . . . 89
4.10 Two disjoint tori and two different interpolants with sufficient slackness. . . 90
4.11 Two disjoint tori and two different interpolants with *semi* slackness. . . . . 91
4.12 Example of integrating iSAT3 with downsizing interpolants method where blue area represents $A$ formula and green area represents $B$ formula. . . . . 93
4.13 iSAT3 bounded model checking problem format. Left: a transition system representing logistic map problem [KB11], right: the corresponding encoding in iSAT3 format. . . . . 95
4.13 CEGAR procedure to solve Example 4.1, where bold **paths** and cyan predicates represent the current counterexample and added constraints in each iteration after refinement respectively. . . . . 102
4.14 An example of useless refinement since none of four checks holds. Image 2 represents the abstract model with the marked spurious counterexample and computed interpolants. Image 3 represents the abstraction after first refinement, where none of checks holds between $\mathcal{I}_0$ and $\mathcal{I}_1$. . . . . 105
4.15 Accumulated verification times for the first $n$ benchmarks. . . . . 109
4.16 Memory usage (#benchmarks processed within given memory limit). . . . . 109
4.17 Accumulated verification times for the first $n$ benchmarks. . . . . 110
4.18 Memory usage (#benchmarks processed within given memory limit). . . . . 110
4.19 Accumulated verification times for the first $n$ benchmarks. . . . . 113
4.20 Memory usage (#benchmarks processed within given memory limit). . . . . 114
4.21 State-Chart of resulting analysis by using CI-based CEAGR (adjusted from [FB13]). . . . . 115
4.22 Left: An *smi*-program with *symtab*-table, middle: corresponding control flow graph, right: encoding in iSAT3-CFG format with FP new syntax according to [SNM+16a]. . . . . 116
4.23 Accumulated verification times for the first $n$ benchmarks. . . . . 119
4.24 Memory usage (#benchmarks processed within given memory limit). . . . . 119

5.1 $1\frac{1}{2}$ player game semantics of an SSMT formula. In recursive solvers, traversal of the dashed part of the quantifier tree will be skipped due to pruning [Tei12]. . . . . 126
5.2 On the right side, an architecture of SSMT solver, e.g. SiSAT. On the left side, an example of solving SSMT formula and how this will be mapped to the architecture of an SSMT solver. . . . . 128
5.3 Example of SSMT-resolution and computing the satisfaction probability 0.12. Red lines identify the pivots. . . . . 134
5.4 Generalized Craig interpolant for Example 5.1. The green part is $A$ and the blue one is $B$. The red part represents $\neg S_{A,B}$ with a don't-care interpolant. 141
5.5 Thermostat case-study discussed in [ZSR+10, FHH+11]. Blue expressions represent the assignments, green ones represent the guards and the magenta ones represent the invariants at each location. . . . . 147
5.6 Illustration of computed backward reachable sets together with generalized Craig interpolants to compute the maximum probability of reaching Error state over number $k$ of transition steps. . . . . 148

5.7 Probability of reaching Error within 5 time units once by using PBMC and once by using GCI. . . . 149
5.8 PHA model represents action planning of a robot, where fail state represents unwanted behaviour. . . . 150
5.9 Probability of reaching fail once by using PBMC and once by using GCI. . . . 151
5.10 Probability of avoiding fail $\wedge\, x \leq 7$ by using GCI. . . . 153

# List of Tables


3.1 Summary of supporting edges results in Example 3.2. . . . . . . . . . . . . . . . . 34
3.2 Non-supporting edges in WFAS model. . . . . . . . . . . . . . . . . . . . . . . . 48
3.3 Figures for verifying well functioning property in WFAS model[2]. Detecting potentially non-supporting edges needs about 0.58 s and 6632 KB. . . . . . 48
3.4 Non-supporting edges in Fischer's protocol with direct fault detection. . . . 51
3.5 Non-supporting edges in Fischer's protocol for the model with delayed fault detection in Figure 3.9. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
3.6 Figures for verifying mutual exclusion. The latter property was satisfied in all verified models. Detecting potentially non-supporting edges needs about 0.17 s and 5856 KB. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
3.7 Fischer's protocol with delayed fault detection. Redirecting edges technique is applied here only, as removing edges cannot be applied since the premise of over-approximating-$P$-rule of Theorem 3.1.2 is broken. Detecting potentially non-supporting edges needs about 0.17 s and 5916 KB. . . . . . . . 56

4.1 Verification results of linear/non-linear hybrid models. Bold lines refer to best results w.r.t. best verification time. Red lines refer to false alarms reported by the solver and blue lines refer to inability to solve the problem due to unsupported functions. . . . . . . . . . . . . . . . . . . . . . . . . . . . 108
4.2 Verification results of (non)-linear hybrid models while comparing abstraction techniques. Bold lines refer to best results w.r.t. best verification time. 110
4.3 Verification results of (non)-linear hybrid models while supporting IEEE 754 standard. **Bold** lines refer to best results w.r.t. best verification time. . 113
4.4 Verification results of linear/non-linear BTC models while supporting IEEE 754 standard for floating points. These models are converted to iSAT-CFG syntax then verified. All benchmarks contain loops and polynomials, but no transcendental functions. In case of bounded model checking techniques as in BMC or preprocessing, if the result is SAFE, it means till depth 250. Generally, if the result is MODEL ERROR, it means the model is SAFE independent of problem-depth. These results were obtained while running tests on AMD Opteron(tm) Processor 6328@2.0 GHZ with 505 GB RAM. . 118

5.1 Results of interpolation-based approach of Example 5.3, where $j$ represents the number of the transitions considered by the interpolation, $\mathcal{I}$ represents the interpolant computed at $j$-th step, and $\mathcal{B}$ represents the backward reachable states. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 147
5.2 Results of interpolation-based approach of Example 5.4, where $j$ represents the number of the transitions considered by the interpolation to increase the preciseness, $\mathcal{I}$ represents the interpolant computed at $j$-th step, and $\mathcal{B}$ represents the backward reachable states. . . . . . . . . . . . . . . . . . . . . . 150

5.3 Results of interpolation-based approach of Example 5.5, where $j$ represents the number of the transitions considered by the interpolation to increase the preciseness, $\mathcal{I}$ represents the interpolant computed at $j$-th step, and $\mathcal{R}$ represents an overapproximation of possible reachable set of states in $\mathcal{M}$. . 152

# List of Abbreviations

**1UIP** First Unique Implication Point

**A400M** Airbus 400 Military Transport

**ACDCL(T)** Abstract Conflict Driven Clause Learning

**ACTL** CTL with only universal path quantifiers

**AI** Abstract Interpretation

**AVACS** Automatic Verification and Analysis of Complex System

**BCP** Boolean Constraint Propagation

**BDD** Binary Decision Diagram

**BMC** Bounded Model Checking

**BNF** Backus Normal Form

**CBMC** C Bounded Model Checking

**CDCL(T)** Conflict Driven Clause Learning

**CEGAR** Counter-Example Guided Abstraction Refinement

**CEX** Counter-Example

**CFA** Control Flow Automaton

**CFG** Control Flow Graph

**CI** Craig Interpolation

**CNF** Conjunctive Normal Form

**CPU** Control Process Unit

**DNF** Disjunctive Normal Form

**DO-178C** Software Considerations in Airborne Systems and Equipment Certification

**DPLL** Davis Putnam Logemann Loveland Algorithm

**DUV** Design Under Verification

**ECA** Event Condition Action

**FMEA** Failure Mode and Effects Analysis

**FP** Floating Point

**GCI** Generalized Craig Interpolation

**GR** Generalized interpolation Rule

**HA** Hybrid Automaton

**ICP** Interval Constraint Propagation

**IEEE** Institute of Electrical and Electronics Engineers

**ILP** Integer Linear Programming

**ISO/IEC PDTR 24772** standard that specifies software programming language vulnerabilities to be avoided in the development of systems

**ITP** Interpolation based Model Checking

**LTL** Linear Temporal Logic

**LZ** Lebens-Zeichen

**MDP** Markov Decision Process

**NaN** Not a Number

**OD** Overapproxmation Driven

**ODE** Ordinary Differential Equation

**PBMC** Probabilistic Bounded Model Checking

**PHA** Probabilistic Hybrid Automaton

**PSPACE** set of all decision problems that can be solved by a Turing machine using a polynomial amount of space

**QBF** Quantified Boolean Formula

**QSAT** Quantified Satisfiability Problems

**RR** Resolution Rule

**SAT** Satisfiability Boolean

**SB** Simple Bounds

**SMC** Symbolic Model Checking

**SMT** Satisfiability Modulo Theories

**SSAT** Stochastic Boolean Satisfiability

**SSMT** Stochastic Satisfiability Modulo Theories

**TA** Timed Automaton

**TCTL** Timed Computation Tree Logic

**TDMA** Time Division Multiple Access

**UD** Underapproxmation Driven

**WFAS** Wireless Fire Alarm System

# List of Symbols

$\alpha$ abstraction function in CFA

$Act$ a set of interpreted actions of automaton where $\eta \in Act$

$\Psi(V, B)$ set of assignments over (non-) boolean variables where $\psi \in \Psi(V, B)$

$\tau$ function assigns to a variable a value from its domain

$Aut$ a set of automata where $\mathcal{A} \in Aut$

$B$ set of boolean variables where $b \in B$

$C$ set of constants over rationals where $c \in C$

$\Sigma$ set of control flow automaton paths where $\sigma \in \Sigma$

$\Gamma$ set of control flow automata where $\gamma \in \Gamma$

$cl$ a clause in a formula

$\Xi$ a set of computation paths of automaton where $\xi \in \Xi$

$\kappa$ concretize function in CFA

$Conf$ a set of configurations of automaton where $c \in Conf$

`DC` do not care area/situation

$EDGES$ edges part in iSAT3-CFG file

$\exists$ existential quantifier

$ff_{\mathbf{sp}}$ function falsifies a simple bound

$falsify_{cl}$ function falsifies a clause

$\varphi$ SMT formula

$\Phi(V, B)$ set of guards over (non-) boolean variables where $\phi \in \Phi(V, B)$

$INIT$ initial configuration in iSAT3-BMC file

$\mathbb{Z}$ a set of integer numbers

$\mathcal{I}$ interpolant

$\mu$ function which maps variables to intervals

$\Lambda$ a set of labels of transitions where $\lambda \in \Lambda$

$Loc$ **or** $\mathcal{L}$ a set of finite locations of automaton where $\ell \in Loc$

$\mathcal{M}$ a probabilistic hybrid automaton model

$N$ set of finite nodes where $n \in N$

$\mathcal{O}_{\mathcal{T}}$ an observable behaviour of automaton

$\mathcal{T}$ operational semantics of automata

$P$ specification in LTL (often assumption)

$p$ probability of holding

$Pr$ probability of satisfiability of a formula

$Q$ specification in LTL (often commitment)

$\mathcal{Q}$ prefix of randomized and existential quantifiers where $Q \in \mathcal{Q}$

ꓤ randomized quantifier

$\Theta(N, \Phi(V, B))$ set of reachability properties where $\theta \in \Theta(N, \Phi(V, B))$

$\mathbb{R}$ a set of real numbers

$S$ specification in LTL

$S_{A,B}$ set of satisfiable assignments between $A$ and $B$ formulae

$F$ some edges of automaton ($F \subseteq E$) which are redirected or removed

$SPECIFICATION$ specification part in iSAT3-CFG file

$\sigma_{sp}$ spurious cfa path during using CEGAR technique

$\delta$ stochastic satisfiability modulo theories formula

$\mathcal{S}$ a set of states of automaton where $s \in \mathcal{S}$

$TARGET$ target part in iSAT3-BMC file

$\mathcal{F}$ a transformation function of automaton

$\mathcal{F}^F_{\mathsf{rd}}$ a transformation function of automaton by redirecting

$\mathcal{F}^F_{\mathsf{rm}}$ a transformation function of automaton by removing

$TRANS$ transitions part in iSAT3-BMC file

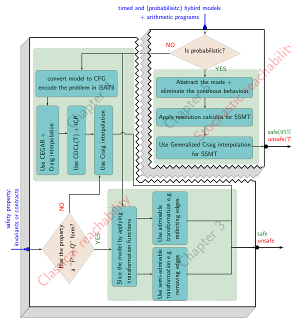
$V$ set of integer and real variables where $v \in V$

$val$ a value in a discrete domain assigned to variables in randomized quantifiers

$\mathcal{I}_{weak}$ weak interpolant

# 1

# Introduction




For, usually and fitly, the presence of an introduction is held to imply that there is something of consequence and importance to be introduced.

*(Arthur Machen)*

## Contents


**1.1 Motivation** . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1**
**1.2 (Partial) History of embedded and (probabilistic) hybrid systems verification** . . . . . . . . . . . . . . . . . . . . . . . . . . . **3**
1.2.1 Verification of embedded systems . . . . . . . . . . . . . . . . . . . 4
1.2.2 Verification of (probabilistic) hybrid systems . . . . . . . . . . . 5
**1.3 Challenges and contributions** . . . . . . . . . . . . . . . . . . . . . . **5**
**1.4 Organization of this dissertation** . . . . . . . . . . . . . . . . . . **9**


## 1.1 Motivation

Software engineering is a discipline that provides methods and techniques to support the software development process and *ensure its quality*. It is composed of several phases: requirement analysis, specification, design, implementation, testing and maintenance. However, testing of reactive and control-oriented programs to *assure quality* or to *assess safety* are becoming more and more complex and highly needed nowadays. For example, we get annoyed when the smart phones or laptops react unexpectedly and wrongly to the issued commands although these software and hardware errors do not threaten our lives. With our increased reliance on software, both at the personal and organizational level, the consequences of software failure can transcend mere annoyance and have profound negative effects on our lives. Think about airbags, braking, cruise control, fuel injection and communication systems where a failure costs not only money, but also people life. Therefore, over the past few decades a very attractive approach toward the correctness of computer-based control systems strongly imposed itself, which is a *model checking*. Model checking requires (1) *a model of the system* under consideration and (2) *a desired property* and systematically checks whether or not the given model satisfies this property [BK08]. Verification by model checking was defined in the late 1970's [Pnu77, CE81], where the

fathers of it have won two Turing awards (Pnueli in 1996 for "introducing temporal logics in computing science", especially in model checking; Clarke, Emerson and Sifakis in 2007 for making model checking "a highly effective verification technology").

Model checking has been used successfully in verifying *hardware problems* by using binary decision diagrams (BDDs) [Ake78] to mitigate the notorious state explosion problem. However, software verification is more complicated and thus is a research track which received a lot of attention in the last decades. In software, we would face arithmetic operations including polynomials and transcendental functions over reals, floating points and integers where its rich theories is beyond what is concisely representable by finite automata. Moreover, software contains complex control structures of programs. To appeal to Edsger Dijkstra's famous quote [Dij72], "Program testing can be used to show the presence of bugs, but never to show their absence!" This means that testing is not sufficient to guarantee the reliability of software if the criterion of quality is zero fault. Also, it is clear that the explicit exploration of all program valuations by computing the possible reachable states of the arithmetic program is mostly impossible. At this point, *abstraction* techniques enter the scene. An abstraction tends to overapproxiamte a model with an abstract system model with finite-state that has all behaviours of the program (and generally more). Hence, if the model checker proves that the abstract system is safe, then so is the original program. The opposite is not true, if the abstract system is not safe, it does not necessarily mean that the original system is not safe too. Building automatically such an abstraction is not a trivial task and will affect the verification process obviously. Predicate abstraction[GS97] was introduced in the middle 1990's by Graf et al.; it uses predicates to encode program states. Given a finite set of predicates, one is able to build a finite system that abstracts the original (infinite) model. Another question arises here is how one would find adequate predicates; interpolants can achieve that feasibly as they are sufficient assertions generated by the infeasibility proof for the error-traces that belong to the abstraction rather than the original model [HHP09]. Several model checkers use interpolants as necessary predicates to eliminate the discovered spurious counterexamples; e.g., IMPACT [McM06], WHALE [AGC12], FunFrog [SFS12], Ultimate Automizer [HCD+13], CPAchecker [BK09] and iSAT3-CFG [SKB13, MSN+16][1].

In this dissertation, the safety aspect is discussed while verifying a wide spectrum of models. In other words, the objects under investigation range from programs to (abstract) models of embedded or hybrid control systems without, however, necessarily attacking the latter at the level of implementable program code. Moreover, these verified models do not only involve linear, polynomial and transcendental arithmetics, but also they may admit probabilistic behaviour, which turn our verification task to be a sophisticated one. We would like to draw the attention of the reader that a *safety case* in probabilistic safety-critical systems which is under investigation relates to situation where the health of people might be jeopardized in the aviation, automotive, and railroad industry. Additionally, it belongs to the medical engineering instead of applications that contribute to the quality of life such as the use of smart phones, washing machines and fridges.

The typical properties that can be verified in our situation, are (probabilistic) *safety* properties. An example of a safety property is to avoid train collision (accident) under any

[1] iSAT3 is the third stable version of Hysat [FH07] which is developed between Carl von Ossietzky Universität Oldenburg and Albert-Ludwigs Universität Freiburg.

scenario of arriving and departing. An example of a probabilistic safety property is to assure that a probability of airplane crash is less than or equal $10^{-9}$ per year [Int96]. Verifying safety properties is very important since it is claimed that hundreds of aircraft crashes in the last decades occurred due to software and hardware failures more than human errors [GF15]. Additionally, it is suitable at this point to highlight on an interesting class of safety properties that has a special form called assumption-commitment [Bro98] statement a.k.a. *contract* [Mey92]. This kind of properties is widely used in component-based design, where under a certain *assumption* on the environment, we guarantee a particular behaviour of a component. Although a contract-based component scheme is successful in specifying functional, safety, and real-time requirements of components, it does not succeed always in verification. For example, while preforming virtual integration testing [DHJ+11] to get a safety case in contract-based component design, the verifiers often suffer from scalability problem since we speak here about very large models. A key step in achieving scalability in the verification of large software systems while verifying safety contracts is to "divide and conquer"; that is, to break up the verification of a system into smaller tasks that involve the verification of its components [CGP03]. Decomposing the verification task can be achieved by many approaches and on different levels. A promising approach is preforming what is so-called *models slicing*, which tends to highlight the relevant parts of the verified model that affect the safety property. Thus, the verification will consider only these highlighted parts while checking the validity of the verified property. Also, slicing technique can be applied at the component-level or at the parallel compositions of components level.

This chapter draws a wide (but incomplete) picture of embedded and hybrid system verification research over the past few decades beside a detailed view of modern automated safety verification tools. We then state the main challenges and contributions of this dissertation.

## 1.2 (Partial) History of embedded and (probabilistic) hybrid systems verification

An *embedded system* is a microprocessor-based system that is built to control a function or range of functions and is not designed to be programmed by the end user in the same way that a PC is [Hea02]. Sometimes, it is defined as a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints [Hea02]. The word "embedded" comes from the fact that this kind of systems is *planted* as a part of a complete device often including hardware and mechanical parts [GB03]. One can consider that embedded systems concept imposed itself – rather than appeared – when the microprocessors were born as a replacement for discrete logic-based circuits in the 1970's since embedded systems concept provides functional upgrades and easy maintenance upgrades as well as improving mechanical performance. Inside such an embedded system, there are usually a processor, memory, peripherals and software. (Expanded) Microcontrollers are proper examples of embedded systems.

The appropriate mathematical model for design of embedded control systems is *hybrid systems models* that can capture both the controller – the system under design and the plant – the environment with continuously evolving physical activities in which the system

operates [Alu11]. A hybrid system is a dynamic system with combined discrete (with a countable number of states) and continuous (a continuous behaviour and a real-valued state space) behaviour [Hen96]. Typical examples are physical systems controlled by a discrete controller. Think about continuous motion that may be interrupted by collisions (mechanical engineering), continuous charging of capacitors being interrupted by switches opening and closing (electrical engineering), the continuous evolution of chemical reactions controlled by valves and pumps (chemical process control), a program behind the autopilot of an aeroplane, which is running on a computer and acting with the physical environment (avionics engineering) [ACH+95]. Sometimes the hybrid systems admit probabilistic behaviour, therefore they need suitable computational models as an extension of classical hybrid systems. For example, probabilistic hybrid automata [Spr01, Spr00] where they only admit discrete probabilistic choices within state transitions, piecewise deterministic Markov processes [Dav84] whose behaviour is governed by random jumps at points in time, but whose evolution is deterministically governed by an ordinary differential equation between those times.

### 1.2.1 Verification of embedded systems

Verification of embedded systems is correlated with early hardware verification experiments in the middle of eighties of the last century. However, the oldest verification was much more likely testing than a formal verification; build the system, run the software and hope for the best. If by chance it does not work, try to do what you can to modify the software and hardware to get the system to work at the end. This practice is called *testing* which is not as comprehensive as formal verification. Several verification techniques have been introduced in the last decades which are mainly as follows:

- *Simulation-based verification*: It has been, and continues to be, the primary method for functional verification of hardware and system-level designs. It consists of providing input stimuli to the design under verification (DUV), and checking the correctness of the output response [Ber00].
- *Formal verification*: In contrast to simulation approach, formal verification methods do not rely upon the dynamic response of a DUV to certain testcases [CW96]. We have two main techniques in formal verification; namely model checking and theorem proving, where model checking techniques have found better acceptance in the industry so far [McM92] due to the easiness of automating the verification and providing counterexamples which are useful for debugging. In model checking [CGP01], the DUV is typically modelled as a finite-state transition system, the property is specified as a temporal logic formula, and verification consists of checking whether the formula is true in that model. In theorem proving [BS92], both the DUV and the specification are modelled as logic formulas, and the satisfaction relation between them is proved as a theorem, using the deductive proof calculus [Hun73] of a theorem prover.
- *Assertion-based verification*: E.g., SystemVerilog assertions [Spe10]; e.g., are considered as a systematic means of enhancing the benefits of simulation and formal verification, and for combining them effectively. Mainly, it is used to capture the designer intent at all steps of the design. Desired properties are used as assertions,

to check for violations of correct behaviour or functionality. The checking can be done dynamically during simulation, statically using formal verification techniques, or by a combination of the two.

### 1.2.2 Verification of (probabilistic) hybrid systems

The verification of hybrid systems as a standalone concept appears at the beginnings of 1990's where the hybrid automaton was proposed as a characteristic model for embedded control systems [ACHH92]. The idea of verifying hybrid system models depends on the possibility of computing an over- or under-approximation of reachable states[2] of the hybrid automaton model and then verifying the desired property within the approximated model. Proving that such a hybrid model is unsafe, requires us to prove that the undesired behaviour is feasible in the underapproximated model or to be able to validate the counterexample if the latter is found in the overapproximated model [CGJ+00]. However, proving such a hybrid model is safe, requires us to prove that the undesired behaviour is infeasible in the overapproximated model [CGJ+00].

Several tools and model checkers support hybrid system verification; e.g, (sorted in order of their appearances) HyTech [AHH96], its follower: HyperTech [HHMW00], HSolver [RS07], PHAver [Fre08], Hysat [FH07] with its ODE-extension; i.e. iSAT2-ODE [ERNF11], KeYmaera [PQ08], SpaceEx [FGD+11], PowerDEVS [BK11], HyEQ [SCN13], and dREAL [GKC13].

Model checking of probabilistic finite-state models is also a very active research topic and has sparked efficient probabilistic model checking tools. For example, PRISM [KNP02] verifies Markov decision processes models, MRMC [KKZ05] verifies continuous-time Markov chains models, SiSAT [FTE10] and ProHVer [ZSR+10] verify probabilistic hybrid automata with discrete time steps and ProbReach [SZ14] verifies probabilistic hybrid automata with continuous random parameters.

## 1.3 Challenges and contributions

In the previous section, a concise overview of embedded and (probabilistic) hybrid systems and their verification tools and techniques was introduced. In this thesis, we make three contributions to automatic verification of embedded and (probabilistic) hybrid systems beside several novel implementations of solving techniques. The high level contribution of this dissertation is new verification algorithms that push the frontiers of interpolation-based verification in stochastic direction and while incorporating ideas from abstraction-based techniques. This allows us to perform unbounded model checking technique while verifying (probabilistic) safety properties in (probabilistic) hybrid and embedded models, such that we can assess the safety in the verified models at any point of time. Furthermore, applying compositional verification while verifying rely-guarantee properties in real time and hybrid system models. These contributions are elaborated on in Chapters 3 to 5 which are published by the author of this thesis together with his

[2]These terms will be explained in Chapter 2.

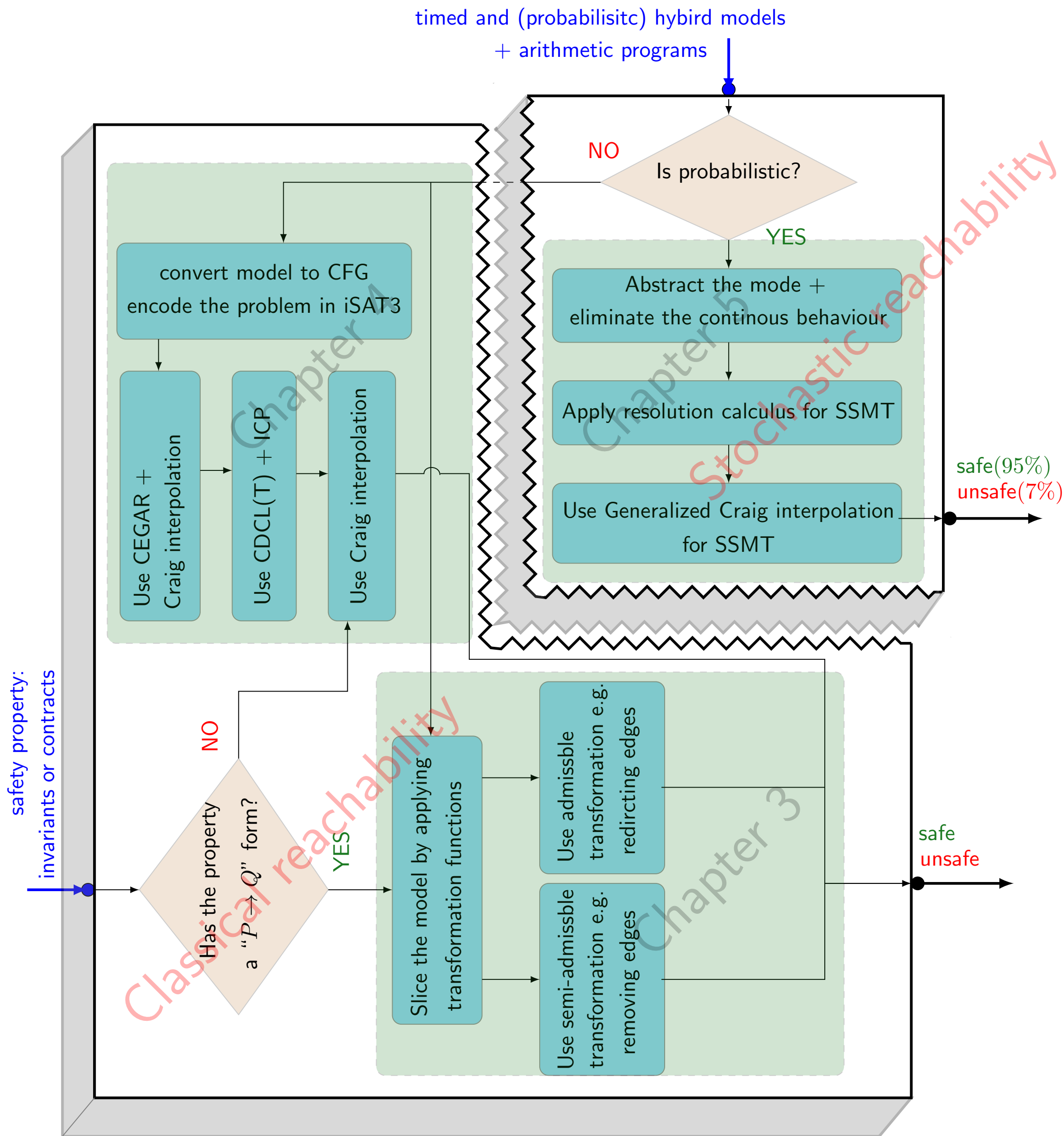



Figure 1.1: The major contributions of this dissertation and the dependencies between them. The cut in the right upper corner separates stochastic reachability from classical one.

co-authors [MF14, MWF14, MSN+16, SNM+16b, SNM+16a]. An overview about reachability analysis within its deterministic and stochastic settings is shown in Chapter 2. Finally, Chapter 6, finalizes this thesis with a summary of the achievements and sheds some light on promising directions for future research.

In the remainder of this section, we outline the three major contributions which are depicted in Figure 1.1.

**1: Verification of assumption-commitment specifications in timed and hybrid models (Chapter 3)** Assumption-commitment forms or contracts fulfil the industrial needs in component-based specification schemes and help in verification as well. However, the scalability of testing and model checking of the industrial models becomes critical due to the size of verified models. Thus, compositional verification is proposed to attack the state space explosion problem which appears often in our situation. Chapter 3 defines the set of models that can be compositionally verified by our approach; namely any computational model where its operational semantics induces a transition system semantics. That is, timed, hybrid, finite automata and programs are under investigation in our approach. Additionally, in Chapter 3 we introduce a general concept of assumption-based (semi-) admissible transformation functions which allows us to eliminate irrelevant traces from the state space of the verified model in a way such that the resultant model is conservative with respect to those traces that violate the commitment only. Moreover, our transformation is an edge-based procedure; it syntactically removes the transitions that always lead to the violation of the assumption. This removal depends on a new concept called *an edge supports a specification*. In addition to that, our proposed technique can be forthrightly integrated with other *slicing or abstraction* techniques and model checkers since it acts as a sound preprocessing approach. Although sometimes just a low number of edges is removed, we observe a speedup of up to ten orders of magnitude relative to direct verification without our compositional procedure.

**2: Verification of reachability in embedded systems involving non-linear arithmetics (Chapter 4)** Detecting dead code (unreachable code fragments) in embedded system C-programs is a challenging task of practical relevance. It is required by several embedded software standards; e.g., DO-178C to avoid critical problems due to possible hidden bugs. In Chapter 4 we will relate the dead code detection problem to the classical reachability analysis in finding a safe invariant of a model. Finding a safe inductive invariant of a model requires a formal verification procedure; e.g., interpolation-based model checking through McMillan's seminal work on hardware model checking [McM03]. McMillan demonstrated how to exploit the resolution proof produced by a SAT solver for a BMC problem [BCCZ99] to over-approximate the reachable states of a finite unrolling of a transition relation. The final interpolant that acts as a guess of a safe inductive invariant is extracted from the resolution proof by rules defined by Púdlak [Pud97] and McMillan [McM03]. Kupfershmid et al. [KB11] succeeded to extend the previous work in the iSAT2 model checker by solving non-linear problems involving transcendental functions. But this extended work did not address a solution for complex generated interpolants.

In this chapter, an incomplete but promising approach is introduced to control the strength and the size of interpolants a.k.a. *the slackness of interpolants*. While Kupferschmid's ap-

proach addressed a feasible solution for non-linear problems, it fails to provide summaries for loops in the control flow and does not scale enough to cover the full branching structure of a complex program in just few sweeps. Therefore, we introduce an extension of iSAT3 – the latest implementatiom of iSAT – in two directions. The first direction introduces a well defined syntax and semantics of a control flow automaton to encode the semantics of programs in iSAT3. The second direction presents a tightly integrated framework that combines iSAT3 as a backend, conflict driven clause learning (CDCL(T)) [ZM02] with interval constraint propagation (ICP) [Ben96] and Craig interpolation (CI) [Cra57], with counterexample guided abstraction refinement [CGJ+00] as a frontend. This allows us to verify reachability in embedded software program without, however, regularly attacking the latter at the level of implementable program code even if these programs are floating points dominated C-programs which may admit non-linear behaviours. The latter problem was spotted by supporting the IEEE 754 standard for floating points.

Finally, Chapter 4 shows a toolchain integration which deals with real case studies from BTC-ES AG, where simulink models are translated into their proprietary intermediate language; i.e. SMI [WBBL02] and consequently these SMI programs are encoded into the new iSAT3 control flow automaton-based language to be verified by using our framework.

**3: Verification of reachability in probabilistic hybrid automata (Chapter 5)** Most of the aforementioned tools [KNP02, KKZ05, ZSR+10, SZ14] and techniques introduced in the last Subsection 1.2.2 are only able to cope with asserting safety in probabilistic models by considering only a fixed number of model unrollings a.k.a. *probabilistic bounded system behaviour*. However, Teige et al. in [TF12a, FTE10] proposed an approach which verifies probabilistic *unbounded* reachability and stability based on a stochastic satisfiability problem. They built a resolution calculus for SSAT problems by extending the classical SAT-resolution rule in order to derive resolvent clauses annotating with probabilities. After that, they extend the classical symmetric rules for systematically computing interpolants. This enables them to encode probabilistic finite-state models; e.g., MDPs as SSAT formulae, whose quantitative interpretations yield upper bounds on the worst-case probability of reaching the unsafe states. However, in Chapter 5, we advance a symbolic approach that goes beyond probabilistic unbounded reachability in the stochastic satisfiability problem by introducing a generalized Craig interpolation for stochastic satisfiability modulo theories (SSMT) [FHT08], where richer fragments of theories are supported. This generalized interpolation is computed over a sound and relatively complete resolution calculus for SSMT, where it provides an opportunity to compute a symbolic overapproximation of the (backward) reachable state set of probabilistic (in)finite-state systems. At this point, whenever the interpolant that overapproximates the (backward) reachable state set reaches a fixed point, we construct an SSMT formula whose quantitative interpretations yield upper bounds on the worst-case probability of reaching the unsafe states.

As an example, the safety property with the following shape: "*the probability that the temperature of the thermostat of the oven exceeds* 220° *Celsius is at most 1%*" will be verified by using the latter approach to compute the upper bound of reaching the unsafe states. Whenever an upper bound of at most 1% is computed then above probabilistic safety property is verified.

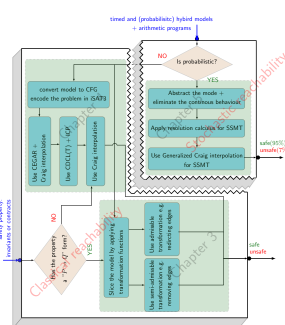
## 1.4 Organization of this dissertation

The rest of this dissertation is organized as follows:

- In Chapter 2, we illustrate the reachability problem with its common terminologies; in particular qualitative and quantitative settings of reachability are introduced.
- In Chapter 3, we introduce a compositional verification technique while verifying assumption-commitments properties in computational models inducing a consistent operational semantics.
- In Chapter 4, we present a novel integration of conflict driven clause learning, interval constraint propagation, Craig interpolation, and counterexample guided abstraction refinement to detect dead codes in mostly non-linear hybrid models where basic floating point arithmetic operations are supported as in IEEE 754. Also, a tool-chain representing our framework with other preprocessing steps are provided to real case studies given by BTC-ES AG.
- In Chapter 5, we develop a resolution calculus for SSMT problems together with generalized Craig interpolation to verify unbounded probabilistic reachability in models admitting stochastic behaviour.
- Chapter 6 summarizes the contributions and discusses open problems and future research directions.
- Appendix A states explicitly all the steps to compute GCI for Thermostat case study in Chapter 5.

# Reachability Analysis

# 2




When you reach for the stars, you are reaching for the farthest thing out there. When you reach deep into yourself, it is the same thing, but in the opposite direction. If you reach in both directions, you will have spanned the universe.

*(Vera Nazarian)*

## Contents


**2.1 Preface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11**
**2.2 Different terminologies for reachability analysis . . . . . . . . . 13**
**2.3 Classical vs. probabilistic reachability . . . . . . . . . . . . . . . 14**


## 2.1 Preface

The reachability problem has received a lot of attention over the past years; precisely since the seventies of last century [Hac74]. The reachability problem was first defined and used in graph theory [BLW86]; e.g., the ability to get from one vertex to another within a graph[1].

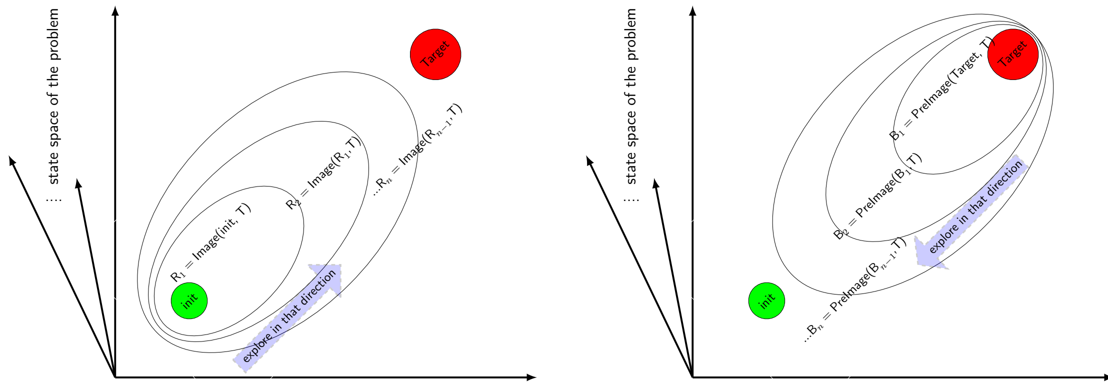
Since several problems are presented in graph notion, reachability analysis is deemed as a feasible approach to address solution for wide class of problems in (in)finite-state systems, rewriting systems [SFG14], dynamical and hybrid systems [GNRR93].

In general terms, a reachability problem consists of deciding whether a given system configuration a.k.a. *target* can ever be reached from a given initial configuration [Hac74] We begin from the initial configuration of the system and find the possible reachable states of the model by exploring the post image of the current reachable states via following the trajectories forwards. The analysis which applies the previous procedure is known as *a forward reachability analysis* (cf. Figure 2.1a). If the target is found to be reachable within our exploration, then it is reachable; otherwise we need to explore more, unless we reach the fixed point where no more states would be inserted in the current image of

[1] A very early allusion for graph notion was indirectly mentioned in a letter written by W. Leibniz in 1679 to C. Huygens [BLW86]. However, the paper written by Leonhard Euler on the Seven Bridges of Königsberg and published in 1736 is considered the first paper in the history of graph theory.

the reachable states. In the latter case, we can surely say that the target is *unreachable*. On the other hand, deciding whether we are able to reach the initial configuration if we begin from the target and follow the trajectories backwards, is called *a backwards reachability analysis* [GD98] (cf. Figure 2.1b). Several recent work combine both forward and backward analyses in one paradigm [Mas01, SS04] which shows impressive results (in terms of time) in proving or disproving reachability, however by paying the cost of extra needed memory.

Moreover, reachability analysis can be seen in several computational models such as timed and hybrid automata, Petri nets and programs as well as predictability in iterative maps. Looking at reachability from another perspective, we will observe that several specification




(a) The idea of forward reachability analysis.

(b) The idea of backward reachability analysis.

Figure 2.1: Forward and backwards reachability analyses.

problems were defined in terms of reachability. For example:

- A part of *Functional correctness of a system*, is to assure that a given system does what it is supposed to do without necessary leading to unsafe behaviour. In terms of reachability problem, it is defined as whether a given system globally remains in a region (e.g. a set of states) representing the desired functional behaviour of the system.
- *Safety*, which means something bad (Hazard) never happens, is defined as reaching bad states of the desired system never happens.

The most crucial reachability property is the one which necessitates that the system must be kept outside of the bad region of the state space; the prototypic *safety property*. Several aircraft in last decades has been plagued by technical faults and software failures which miserably led to crashes. For example, in 2015 Airbus has issued software bug alert after fatal crash of the A400M military transport plane in Spain [GF15]. In the summary of the annual report [Air16] published by Boeing in 2015, 19 from 28 – ranging from minor to fatal – problems occurred due to mechanical or software failures. This may justify the claim that major plane crashes are often technology failures, rather than human errors.

Normally, system failures refer to bad/unexpected behaviours. In other words, the system *reaches* a bad state that violates the safety properties. Thereby, in this dissertation, we will draw a special attention to reachability problem relating to safety aspect in hybrid and embedded systems. Specifically, we address the problem of automatically verifying safety invariance properties of computational models that induce a transition system semantics; e.g., timed and hybrid automata and programs. Safety properties encompass a wide spectrum of desirable correctness property (e.g., no assertions are violated, memory safety, secure information flow, etc.). To prove that a model satisfies such a safety property, we need to find *a safe inductive invariant*[2]. A safe inductive invariant portrays an over-approximation of reachable model states that does not intersect with unsafe states specified by the property. We advance safety property verification in classical and probabilistic fields as shown in Chapter 4 and Chapter 5.

## 2.2 Different terminologies for reachability analysis

Many research works discuss reachability problems in different fields[3], however with divergent terminologies in some cases. The most common terminology or keyword is reachability problem or reachability analysis, while other keywords are used in certain communities. For example dead code detection [CGK98, CGK97], dead code elimination/removing [Kno96, DP96], unreachble code detection [PT15, CCK11], infeasible code detection [CHS12, AS12, DZT14], finding safe invariants [Alb15], region stability [PW07] which involves a combination of liveness and invariance properties. All previous terminologies are correlated and some of them can be mostly mapped together.

In the following we will add some highlights on dead code detection/elimination which will be discussed in detail in Chapter 4. First of all, (partial) dead code elimination or removal is a subsequent step beyond detecting it. In the literature, there are two inconsistent definitions of dead code:

- Dead code is a section in the source code of a program which is executed but whose result is never used in any other computation [DEMS00].
- Dead code is a section in the source code of a program which as a result of a design error is neither reached by the logics of the program flows nor executed a.k.a. *unreachable code* [EH10]. This definition conforms with standards definition like DO178C, therefore it will be used in the sequel of this thesis.

Given that safety-critical control functions, like in automated driving, would ultimately have to be certified at the code level, there obviously is a pronounced industrial quest for verification methods directly addressing the level of program code, and to do so even if the program is a controller implementation heavily depending on (potentially non-linear) arithmetic. In addition to that, DO-178B/ED-12B essentially requires that any dead code has to be removed in particularly for embedded systems in avionic domain. Identifying

[2]This is needed in unbounded model checking settings. In bounded model checking, one ought to prove the system does not reach bad states till certain depth while iteratively exploring the state space of the problem.

[3]For example, searching for published works in `http://dblp.org/` with title including “reachability” returns with more than 1800 hits.

dead code is also a good development practice irrespective of certification requirements because studies have shown that dead code is a source of hidden defects and run-time errors. In this thesis, we use *dead code* and *unreachable codes* interchangeably as proposed in the latter standard which conforms with [EH10]. Furthermore as shown in Chapter 4, if we identify such a code-segment as a dead one or a state representing a bad (unwanted) behaviour as an unreachable state in a program or a model respectively, this means that the invariant that overapproximates the reachable states of the program or the model is safe.

## 2.3 Classical vs. probabilistic reachability

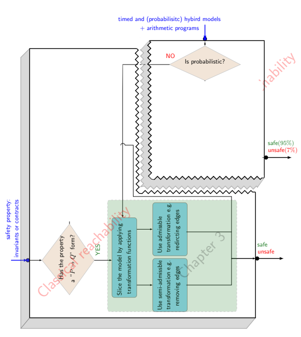
In a qualitative setting, reachability is a yes/no problem, where one evaluates whether starting from a given set of initial states the system will reach a certain set or not; this kind of analysis refers to classical reachability where a verification task aims at obtaining a definite verdict; e.g., that the code-segment is reachable.

However engineering systems like communication networks [HA07] or automotive [Kru07] and air traffic control systems [LH10], financial and industrial processes like manufacturing [BP01] and market models [vdBKvdB04] , and natural systems like biological [Alt95] and ecological environments [SHBV03] exhibit probabilistic behaviour arising from the compositions and interactions between their (heterogeneous) components. Thus probability is necessary in order to:

- quantify arrival and waiting times as well as time between failures while analysing system performance and dependability.
- quantify environmental factors in decision support, unpredictable delays and express soft deadlines while modelling uncertainty in the environment.
- implement randomized algorithms while building protocols for networked embedded systems.

Since we refer to a stochastic setting, one has to refer to a well-known mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision maker, Markov decision process (MDP) [Bel57]. Informally, MDP is a finite-state stochastic process in which state changes are subject to non-deterministic selection among available actions followed by a probabilistic choice among potential successor states, where the probability distribution of the latter choice depends on the selected action. The main problem of MDPs is how one would find an adequate policy for the decision maker: a function that determines the action which will be chosen by the decision maker in the current state. In a stochastic setting, the different trajectories originating from each initial state have a different likelihood and we are interested in the maximum probability of reaching a given set of target states under an arbitrary policy (adversary) (within a given number of transition steps in case of bounded model checking problems). The latter analysis is supposed to handle probabilistic safety properties of the shape “the worst-case probability of reaching the unsafe states is at most 2%”.

# 3 Model Slicing




No matter how tough the meat may be, it's going to be tender if you slice it thin enough.

*(Guy Fieri)*

## Contents


**3.1 Problem statement . . . 17**

3.1.1 Motivation . . . 17

3.1.2 Related work: . . . 18

**3.2 Preliminaries . . . 20**

**3.3 Assumption-commitment specifications . . . 23**

**3.4 Model element-based slicing technique . . . 24**

**3.5 Transformation functions . . . 28**

3.5.1 Admissible transformations . . . 28

3.5.2 Semi-admissible transformations . . . 31

**3.6 New reachability concept: supporting edges . . . 34**

3.6.1 Supporting edges . . . 34

3.6.2 Supporting edges and transformation functions . . . 36

3.6.3 Verification based on support-notion . . . 37

**3.7 Compositional verification . . . 40**

**3.8 Case studies . . . 42**

3.8.1 Wireless sensor network: Alarm system . . . 42

3.8.2 Fischer's mutual exclusion protocol . . . 49

## 3.1 Problem statement

### 3.1.1 Motivation

Embedded systems are nowadays expected to provide increasingly many functions and different modes as mentioned in Chapter 1. Some of these modes are operational ones, and others are related to possible failures. However, in order to determine correct failure modes, at all system levels, failure mode and effects analysis (FMEA) [CCC+93] was developed by reliability engineers in the late 1950's to study problems that might arise from malfunctions of military systems [MIL]. In the real-world, faults[1] cannot be avoided in general: wires may break, radio frequencies may continuously be blocked, a random hardware bug, a memory bit stuck, and physical sensors and actors may fail. One way to deal with these situations is to analyse the system during development process, identify their effects on the operation of the product, define a mechanism to detect and display faults in order to, e.g., inform users to take countermeasures against the fault. At this point, one can assert the correctness of the system under design if it delivers regular functionality *unless* a fault is detected/displayed or unexpected behaviour outside the frame of functions under consideration occurs. As in this thesis, we deal with formal verification methods rather than simulation-based techniques, our main safety requirement – to assure correctness of the design – is to verify that under the assumption of the absence of faults, the system functions properly. For example, given the brake systems of an aeroplane as in [SAE96], if the command units have no failures, then our model has to guarantee that the brakes work properly. This kind of requirements is widely used in industrial fields under the name *contract* or assumption-commitment specification [MC81, Dam08, DHJ+11, SVD+12]. An assumption-commitment specification consists of an assumption and a commitment, where a commitment is required to hold (by a component) if the assumption holds as well (by an environment). Now, we can generalize our safety verification task to assure that under the given assumption; e.g., fault absence, the commitment has to hold in our model. In this thesis, *assumption* and *rely* are used interchangeably and so are *commitment*, *promise* and *guarantee* used reciprocally.

Since we spoke about contract-based component specification, one has to imagine the complexity and the difficulty of verifying properties in the models with industrial scale: there are several layers of abstractions and thousands of components and subcomponents from heterogeneous environments. Therefore, the verification process is really challenging due to several reasons. Among others, system models are increasingly complex and hardly traceable, and verifier tools face a combinatorial blow up of the state-space, commonly known as the state explosion problem.

In order to overcome the latter problem, we introduce a new compositional verification technique that on one hand optimises the verification time and memory by a fair margin in comparison to other techniques. On the other hand, it conforms with other slicing and abstraction techniques, as it applies sound and conservative model transformations

[1] A system *failure* is an event that occurs when the delivered service deviates from correct service. A system may fail either because it does not comply with the specification, or because the specification did not adequately describe its function. *An error* is that part of the system state that may cause a subsequent failure: a failure occurs when an error reaches the service interface and alters the service. *A fault* is the adjudged or hypothesized cause of an error [UAcLR01].

without affecting the validity of the verified property.

Our approach is based on the new notion of model elements *supporting* a specification. Intuitively, a specification is supported by a model element if there is a computation path in the model's semantics which satisfies the specification and uses that model element. That is, a specification is supported by a model element if the model element is *reachable* by a computation path which satisfies the specification. Instead of verifying an assumption-commitment property on the model, we apply *a source-to-source* transformation to the desired model, where those model elements which *do not support* the assumption are effectively disabled. This transformation excludes computation paths from the verification process which are irrelevant for the overall property because they violate the assumption. Thereby, our approach decreases the complexity of the desired model already before running a model checking tool on the transformed model. Furthermore, our approach is independent from particular model checking tools as we transform the model and leave the model checking procedure unscathed. We develop our approach for a generalized notion of *automata* consisting of directed, interpreted action-labelled edges between locations in order to uniformly treat computational models such as finite and Büchi automata, timed and hybrid automata, and even programs. A **necessary assumption** of our approach is that the operational semantics by which an automaton induces a transition system is *consistent* for the syntactical transformations. This consistency assumption is typically satisfied by the standard semantics. Our approach is particularly well-suited for systems which provide many functions and operation modes, e.g. a plane's brake system may offer landing and taxiing modes. For validation purposes, it is useful to have only a single system model including all features but verification may practically be infeasible on such a model. Given an assumption-commitment specification, where the assumption limits the focus to only some features, our approach allows to mechanically create a smaller verification model by excluding irrelevant transitions but still guarantee to reflect the relevant behaviour of the original model. Thereby, there is no more need to create specially tailored verification models manually.

### 3.1.2 Related work:

Clever verification and testing engineers often apply several preprocessing steps in order to optimise the verification and testing tasks, where our approach would be seen as one of these feasible preprocessings.

In this subsection, we mention the major related works to our theory, however without neglecting the fundamental differences in comparison to our approach.

**Abstractions.** There is substantial previous work [BGD11, H$^+$13] on excluding irrelevant computation paths from the verification process by abstracting the original model. These works mainly apply more or less counterexample-guided abstraction refinement [CGJ$^+$00], where the abstract model is refined upon request: an erroneous counterexample is discovered and necessary predicates enrich the abstract model to exclude the discovered spurious counterexample[2]. In contrast, our work is a source-to-source transformation, hence abstractions can still be applied after our approach.

[2]More details can be found in Chapter 4.

**Model reduction vs. model slicing.** Our approach shares the same idea with model slicing or reduction techniques. However the main difference is that slicing models is focusing on the main parts of the model that affect the verified property, where our approach applies targeted slicing rules; it excludes traces of the model that are irrelevant to the verification process even if they are considered to be under investigation while slicing the models.

For example, the exclusion of model behaviour by a source-to-source transformation proposed in [MW+12] only considers networks of timed automata with disjoint activities. Thus, instead of taking the parallel compositions of automata, a concatenation of automata is sufficient. They showed that the complexity of verification in Uppaal-like [BLL+95] tools reduces from quadratic to linear time. Slicing of timed automata [JJ04] removes locations and clock and data variables on which a given property does not depend, thus it also keeps variables on which an assumption depends while our approach may remove the corresponding behaviour. Also, the path slicing [JM05] technique determines which subset of the edges along a given control flow path to a particular target location are relevant towards demonstrating the (un)reachability of the target location along the given path, however slicing is done on-fly and locally for each infeasible path to a target location. In contrast, our approach applies a source-to-source transformation to reduce the entire size of the model, but both approaches can be forthrightly integrated.

Reduction of concurrent models via classical slicing in object-oriented programs [DHH+06] shows good results, where static slicer tools are used such as Bandrea [CDH+00] and Indus [JRH05]. Moreover, (safety) slicing of Petri nets [Rak11, Rak12] applies slicing technique but on Petri nets models where the resultant models may still contain computation paths where the assumption is violated. This is not the case in our approach where these paths will be removed or disabled in our technique.

**Partial model checking.** With partial model checking [And95], verification problems are modularized by computing weakest context specifications through quotienting, which allows to successively remove components completely from the verification problem. We are instead trying to pragmatically reduce the size of components before composition by exploiting the specification. Both approaches could go well together.

**Static contract checking.** Static contract checking for functional programs [XJC09] is dealing with a very different class of computational objects and relies heavily on assumptions local to the individual functions, while our approach is meant to also “massage” the global specification into the components.

**Structural transformations.** This work is close to our work in applying a source-to-source transformations in networks of timed automata. They remove some transitions by what is so-called *flattening* [OS15]. In addition to that, all transformations have an algebraic flavour, obtained by rewriting the system’s composition using a library of composition operators. Each of the proposed transformations, despite being local – which is desirable in the context of a network of parallel components, as the costly computation of the parallel product can then be avoided – preserves certain properties in parallel contexts [OS13].

Also, the verified property must be affected/changed only by memoryless assignments. However, our work deals with general models without necessarily being timed automata models on one hand. On the other hand, our approach is performed in an assumption-commitment setting, where our transformation or slicing approach is an assumption-based one and no model restriction applies for the operations that affect the assumption part.

## 3.2 Preliminaries

In this section, we introduce the main definitions and concepts that will be used in this chapter and in the later chapters, where a generalized notion of automata is considered which allows us to treat, among others, timed and hybrid automata uniformly as well as arithmetic programs.

**Definition 3.1: Automaton**

An automaton $\mathcal{A}$ is a structure

$$(Loc, Act, E, L_{ini})$$

where:

- $Loc$ is a finite set of **locations** with typical element $\ell$.
- $Act$ is a set of **interpreted actions** with typical element $\eta$.
- $E \subseteq Loc \times Act \times Loc$ is a finite set of **directed edges**.
  An element $(\ell, \eta, \ell') \in E$ describes an edge from location $\ell$ to $\ell'$ with action $\eta$.
- $L_{ini} \in Loc$ is the **initial** location.

We use $Aut$ to denote the set of automata.

Finite, Büchi [Bue62], timed [AD94], and hybrid automata [Hen96] can be represented as automata in the sense of Definition 3.1. For example, for timed automata, we can consider pairs of locations and invariants as locations, and triples consisting of synchronization, guard, and update vector as action. Thereby, the alphabet of a timed automaton is represented in the set of interpreted actions. Formally, it is mapped as following:

**Definition 3.2: Timed automaton**

A timed automaton

$$TA = (L^{TA}, B^{TA}, X^{TA}, V^{TA}, I^{TA}, E^{TA}, \ell_{ini}^{TA})$$

where

- $L^{TA}$ is a finite set of locations,
- $B^{TA}$ is an alphabet,
- $X^{TA}$ is a set of clocks,
- $V^{TA}$ is a set of variables,
- $I^{TA} : L^{TA} \to \Phi(X^{TA})$ is a function mapping locations to constraints, where $\Phi(X^{TA})$ is a set of clock constraints,

- $E^{TA} \subseteq L^{TA} \times B^{TA}_{!?} \times \Phi(X^{TA}, V^{TA}) \times R(X^{TA}, V^{TA})^* \times L^{TA}$ is a finite set of edges, where $B^{TA}_{!?}$ is a set of channels $\Phi(X^{TA}, V^{TA})$ is a set of constraints and $R(X^{TA}, V^{TA})$ is a set of reset operations. An element $(\ell^{TA}, b_{!?}, \varphi^{TA}, \vec{r}, \ell^{TA'}) \in E^{TA}$ describes an edge from location $\ell^{TA}$ to $\ell^{TA'}$ with channel $b_{!?}$, guard $\varphi^{TA}$ and assignments $\vec{r}$, and
- $\ell^{TA}_{ini} \in L^{TA}$ is the initial location.

is an automaton $\mathcal{A}_{TA}$ as in Definition 3.1 where:
- $Loc := \{(\ell^{TA}, I^{TA}(\ell^{TA})) \mid \ell^{TA} \in L^{TA}\}$, for $\ell = (\ell^{TA}, I^{TA}(\ell^{TA})) \in Loc$, we use $\ell_I$ to denote the unique (!) $I^{TA}(\ell^{TA})$,
- $Act := B^{TA}_{!?} \times \Phi(X^{TA}) \times R(X^{TA}, V^{TA})$,
- $E := \{(\ell, (b, \varphi, \vec{r}), \ell') \mid (\ell, ,\varphi, \vec{r}, \ell') \in E^{TA}\}$,
- $L_{ini} := (\ell^{TA}_{ini}, I^{TA}(\ell^{TA}_{ini}))$.

The hybrid automaton definition holds analogously. Moreover, *programs* are automata as follows: the nodes in the control flow graph (CFG) become automaton locations and the edges in the CFG become automaton edges labelled with statements.

In the following definition, we introduce an *edge-centric notion* of operational semantics for generalized automata; i.e. there is one transition relation per edge and one dedicated additional transition relation. This allows for a simple definition of *support* in Section 3.6. Later we will characterise those operational semantics for which our approach applies as consistent operational semantics.

**Definition 3.3: Operational semantics of automaton**

Let $\mathcal{S}$ a set of **states**. An *operational semantics* of *Aut* (over $\mathcal{S}$) is a function $\mathcal{T}$ which assigns to each automaton $\mathcal{A} = (Loc, Act, E, L_{ini}) \in Aut$ a labelled transition system $\mathcal{T}(\mathcal{A}) = (Conf, \Lambda, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda\}, C_{ini})$ where:
- $Conf(\mathcal{A}) \subseteq Loc \times \mathcal{S}$ is the set of **configurations**, and
- $\Lambda = E \cup \{\bot\}$, where $\bot \notin E$,
- $\xrightarrow{\lambda} \subseteq Conf(\mathcal{A}) \times Conf(\mathcal{A})$,
- $C_{ini} \subseteq (\{L_{ini}\} \times \mathcal{S}) \cap Conf(\mathcal{A})$ is the set of initial configurations.

The previous definition states the framework for operational semantics of automata since $Conf(\mathcal{A})$ and $\xrightarrow{\lambda}$ have to be specified according to each kind of automata. Thus, it can be applied to *finite* and *Büchi*, *timed* and *hybrid* automata as long as the configurations and transition relation are defined appropriately. For example, in finite and Büchi automata, the label transitions with edges are mapped to symbols and no need to define states and $\bot$-transitions. The formal mapping of operational semantics of timed automata to the generalized automata is defined as following:

**Definition 3.4: The operational semantics of timed automata induces an automata semantics**

Let $TA$ be a timed automaton. Its operational semantics

$$\mathcal{T}(TA) = (Conf(TA), B_{!?} \cup \mathbb{R}^+_0, \{\xrightarrow{\lambda} \mid \lambda \in B_{!?} \cup \mathbb{R}^+_0\}, C_{ini})$$

with

- $Conf(TA) := \{\langle \ell^{TA}, \nu \rangle \mid \ell^{TA} \in L^{TA}, \nu \in \mathcal{V}, \nu \models I^{TA}(\ell^{TA})\}$,
- $\mathcal{V}$ is the set of valuations of clocks and variables, i.e. $\mathcal{V} := ((X^{TA} \cup V^{TA}) \to \mathsf{Time} \cup \mathcal{D}(V^{TA}))$ where $\mathsf{Time} := \mathbb{R}_0^+$ and $\mathcal{D}(V^{TA})$ is the domain of the variables,
- there is a satisfaction relation $\models$ defined on $\mathcal{V} \times \Phi(X^{TA}, V^{TA})$,
- there is an update function $\cdot[\cdot] : \mathcal{V} \times R(X^{TA}, V^{TA})^* \to \mathcal{V}$, and a elapse-time function $\cdot + \cdot : \mathcal{V} \times \mathbb{R}_0^+ \to \mathcal{V}$,
- there is an action transition $\langle \ell^{TA}, \nu \rangle \xrightarrow{\alpha} \langle \ell^{TA\prime}, \nu' \rangle$ if there is an edge $e = (\ell^{TA}, \alpha, \varphi, r, \ell^{TA\prime}) \in E^{TA}$ such that
  - $\nu \models \varphi$,
  - $\nu' = \nu[r]$,
  - $\nu' \models I^{TA}(\ell^{TA\prime})$.

  We then say that $e$ *justifies* this action transition.
- there is a delay transition $\langle \ell^{TA}, \nu \rangle \xrightarrow{d} \langle \ell^{TA}, \nu' \rangle$ if
  - $d \in \mathbb{R}_0^+$,
  - $\nu' = \nu + d$,
  - $\forall 0 \leq t \leq d \bullet \nu + t \models I^{TA}(\ell^{TA})$,
- $C_{ini} = \{\langle \ell_{ini}^{TA}, \nu_{ini} \rangle\} \cap Conf(TA)$ where $\nu_{ini}(v) = 0$ for all $x \in X^{TA} \cup V^{TA}$,

induces an operational semantics of $\mathcal{A}_{TA}$ as in Definition 3.3 where:

- $\mathcal{S} := \mathcal{V}$,
- there is an (edge-justified) transition $\langle \ell, s \rangle \xrightarrow{e} \langle \ell', s' \rangle$ if and only if
  - $e = ((\ell^{TA}, I^{TA}(\ell^{TA})), (\alpha, \psi, r), (\ell^{TA\prime}, I^{TA}(\ell^{TA\prime}))) \in E$,
  - and $(\ell^{TA}, \alpha, \psi, r, \ell^{TA\prime})$ justifies $\langle \ell^{TA}, \nu \rangle \xrightarrow{\alpha} \langle \ell^{TA\prime}, \nu' \rangle$.
- there is an (other) transition $\langle \ell, s \rangle \xrightarrow{\perp} \langle \ell, s' \rangle$ if and only if
  - $\ell = (\ell^{TA}, I^{TA}(\ell^{TA}))$,
  - $\langle \ell^{TA}, \nu \rangle \xrightarrow{d} \langle \ell^{TA}, \nu' \rangle$ for some $d \in \mathbb{R}_0^+$.

The situation for hybrid automata is similar to timed automata except that during delay transitions, variables are updated according to the flows. Finally, the *semantics of programs* are mapped as follows: program states are variable valuations, labelled-transitions with edges are statements, no $\perp$-transitions, initial states are given by programming language semantics.

An operational semantics induces computation paths as usual. In addition, we distinguish computation paths based on the occurring labels. Depending on computation path semantics, we introduce the definition of the observable behaviour of the automaton, i.e., the sequence of configurations obtained by disregarding the labelled transitions.

#### Definition 3.5: Computation Path

A *computation path* of automaton $\mathcal{A} \in Aut$ under operational semantics $\mathcal{T}(\mathcal{A}) = (Conf, \Lambda, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda\}, C_{ini})$ is an initial and consecutive, infinite or maximally finite sequence $c_0 \xrightarrow{\lambda_1} c_1 \xrightarrow{\lambda_2} \cdots$ where $c_0 \in C_{ini}$ (initiation) and for each $i \in \mathbb{N}_0$, $(c_i, c_{i+1}) \in \xrightarrow{\lambda_{i+1}}$ (consecution).
Let $E$ be the set of edges of $\mathcal{A}$. We use $\Xi_{\mathcal{T}}(\mathcal{A}, F)$ to denote the set of computation paths of $\mathcal{A}$ where only label $\perp$ and labels from $F \subseteq E$ occur. $\Xi_{\mathcal{T}}(\mathcal{A}) := \Xi_{\mathcal{T}}(\mathcal{A}, E)$

denotes the set of all computation paths of $\mathcal{A}$ (under $\mathcal{T}$).

**Definition 3.6: Observable Behaviour**

Let $\xi = c_0 \xrightarrow{\lambda_1} c_1 \xrightarrow{\lambda_2} \cdots \in \Xi_{\mathcal{T}}(\mathcal{A})$ be a computation path of automaton $\mathcal{A}$ under operational semantics $\mathcal{T}$. The *observable behaviour* of $\xi$ is the sequence $\downarrow\xi = c_0, c_1, \ldots$ We use $\mathcal{O}_{\mathcal{T}}(\mathcal{A})$ to denote the set of observable behaviours of the computation paths of $\mathcal{A}$ under $\mathcal{T}$, i.e. $\mathcal{O}_{\mathcal{T}}(\mathcal{A}) = \{\downarrow\xi \mid \xi \in \Xi_{\mathcal{T}}(\mathcal{A})\}$.

## 3.3 Assumption-commitment specifications

As introduced in the beginning of this chapter we are interested in verifying what are so-called assumption-commitment properties or contract-based specifications. Note that a *semantical* characterisation of specifications is used for sake of simplicity. A specification is a set of sequences, i.e. we consider path specifications. Specifications can *syntactically* be described by, e.g., LTL [Pnu77].

**Definition 3.7: Assumption-Commitment Specification**

A *specification* over *alphabet* $\Sigma$ is a set $S \subseteq \Sigma^* \cup \Sigma^\omega$ of finite or infinite sequences over $\Sigma$.
A specification is called *assumption-commitment* specification, a.k.a. *contract*, if there are two prefix closed specifications $P$ and $Q$ such that $S = \overline{P} \cup Q$, where $\overline{P}$ denotes the complement of $P$ in $\Sigma^* \cup \Sigma^\omega$, i.e. the set $(\Sigma^* \cup \Sigma^\omega) \setminus P$. We write $P \to Q$ to denote the assumption-commitment specification $\overline{P} \cup Q$. A set $p \subseteq \Sigma$ is called *atomic proposition*, and the specification $\Box p := p^* \cup p^\omega$ is called an *invariant*.

Now, we establish the *satisfaction relation* between automata and specifications based on the observable behaviour obtained from computation paths of the automata. One remarks that we use the notions an *automaton implements a specification* and an *automaton satisfies a specification* interchangeably.

**Definition 3.8: Satisfying a Specification**

Automaton $\mathcal{A}$ is said to *satisfy* or *implement* the specification $S$ (under $\mathcal{T}$), denoted by $\mathcal{A} \models_{\mathcal{T}} S$, if and only if the set of observable behaviours of $\mathcal{A}$ (under $\mathcal{T}$) is a subset of $S$, i.e. if $\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \subseteq S$.

**Example 3.1: Satisfaction example**

Consider the finite automata depicted in Figure 3.1. It has four locations; namely $q_0, q_1, q_2$ and $q_3$, where $q_0$ is the initial location. Also, each edge is labelled with a guard and an assignment. Now, we have a specification $\Box x = 0$ which informally means that always $x$ equals 0. Firstly a computation path $\xi_1 : \langle q_0, x = 0\rangle \to \langle q_1, x = 0\rangle \to \langle q_2, x = 0\rangle \to \langle q_3, x = 0\rangle \to \langle q_3, x = 0\rangle$... obviously satisfies the specification.

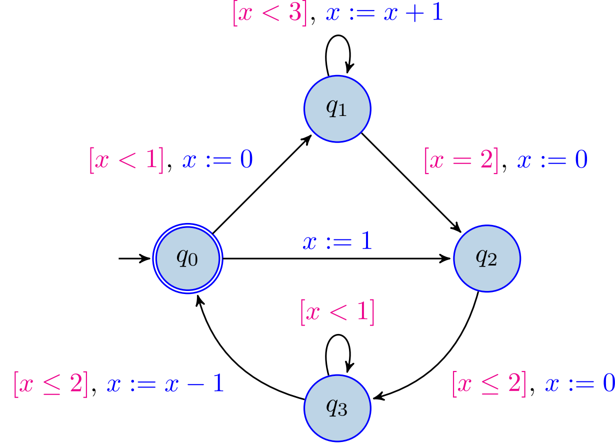



Figure 3.1: A satisfaction relation between an automaton and specification $\Box x = 0$.

However, all other computation paths don't satisfy this specification. Thus, the automaton does not implement $\Box x = 0$.

## 3.4 Model element-based slicing technique

In this section, the idea of model slicing will be presented in a general fashion, where in the next section a valid instance of that procedure is introduced. Our slicing model technique is applying a sound compositional verification on the desired model checking task as shown in the following theorem.

**Theorem 3.1: Compositional Verification**

Let $\mathcal{A}_1 \in Aut_1$ and $\mathcal{A}_2 \in Aut_2$ be automata and $\mathcal{T}_1$ and $\mathcal{T}_2$ operational semantics for $Aut_1$ and $Aut_2$, respectively. Let $P \to Q$ be an assumption-commitment specification.

1. The *common-P*-hypothesis: whenever the set of observable behaviours of $\mathcal{A}_1$ that satisfy $P$ is equal to the set of observable behaviours of $\mathcal{A}_2$ that satisfy $P$, then $\mathcal{A}_1$ satisfies $P \to Q$ if and only if $\mathcal{A}_2$ satisfies $P \to Q$, i.e.,
$$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P = \mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \cap P \implies (\mathcal{A}_1 \models_{\mathcal{T}_1} P \to Q \iff \mathcal{A}_2 \models_{\mathcal{T}_2} P \to Q).$$
2. The *over-approximating-P*-hypothesis: whenever the set of observable behaviours of $\mathcal{A}_1$ that satisfy $P$ is a subset of the set of observable behaviours of $\mathcal{A}_2$, then $\mathcal{A}_1$ satisfies $P \to Q$ if $\mathcal{A}_2$ satisfies $Q$, i.e.,
$$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P \subseteq \mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \implies (\mathcal{A}_2 \models_{\mathcal{T}_2} Q \implies \mathcal{A}_1 \models_{\mathcal{T}_1} P \to Q).$$
In general, the second implication does not hold in the other direction.

**Proof of Compositional Verification**

We will begin with the first part of the theorem:

1. Let $\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P = \mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \cap P$ as given in the premise of *common-P*-rule.
   By using Definition 3.3, we know that: the automaton $\mathcal{A}_1$ under $\mathcal{T}_1$ satisfies the specification $P \to Q$ if and only if the set of observable behaviour of $\mathcal{A}_1$ under $\mathcal{T}_1$ is a subset of the $P \to Q$.
   That is,
   $$\mathcal{A}_1 \models_{\mathcal{T}_1} P \to Q \iff \mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \subseteq P \to Q$$
   Consequently be using Definition 3.7 we get:
   $$\mathcal{A}_1 \models_{\mathcal{T}_1} P \to Q \iff \mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \subseteq \overline{P} \cup Q(*)$$
   Now, if we consider the whole set of observable behaviour i.e. $P \cup \overline{P}$ together with the result obtained in $(*)$; i.e. $\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \subseteq \overline{P} \cup Q$ we get:
   $$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap (P \cup \overline{P}) \subseteq (\overline{P} \cup Q) \cap (P \cup \overline{P})$$
   $$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap (P \cup \overline{P}) \subseteq (\overline{P} \cup Q)(**)$$
   Now we split $(**)$ into two expressions; namely:
   $$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P \subseteq (\overline{P} \cup Q) \text{ and } \mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap \overline{P} \subseteq (\overline{P} \cup Q)$$
   We know that $\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P \nsubseteq \overline{P}$, thus:
   $$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P \subseteq (\overline{P} \cup Q) \equiv \mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P \subseteq Q$$
   We know also that $\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap \overline{P} \subseteq (\overline{P} \cup Q)$ is a tautology. Additionally, $\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P \subseteq Q$ holds if and only if the premise of the rule holds i.e. $\mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \cap P \subseteq Q$ (as given in the rule).
   With the same procedure of previous analysis, we can say:
   $$\mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \cap \overline{P} \subseteq \overline{P}$$
   By combining $\mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \cap P \subseteq Q$ with $\mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \cap \overline{P} \subseteq \overline{P}$, we get:
   $$(\mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \cap P) \cup (\mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \cap \overline{P}) \subseteq \overline{P} \cup Q$$
   So
   $$\mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \subseteq \overline{P} \cup Q$$
   By using Definition 3.7, we get
   $$\mathcal{A}_2 \models_{\mathcal{T}_2} P \to Q$$
2. For the second part of the theorem, let $\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P \subseteq \mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2)$ as given in the premise of the rule.
   We are given that the second automaton $\mathcal{A}_2$ satisfies the specification $Q$

i.e. $\mathcal{A}_2 \models_{\mathcal{T}_2} Q$. By using Definition 3.3 we get:

$$\mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \subseteq Q$$

By using the premise of *over-approximating-P*-rule we get:

$$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P \subseteq \mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \subseteq Q$$

We know that any operation over observable behaviour follows *set-semantics*. Thus

$$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap \overline{P} \subseteq \overline{P}.$$

By combining $\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P \subseteq Q$ with $\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap \overline{P} \subseteq \overline{P}$, we get:

$$(\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P) \cup (\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap \overline{P}) \subseteq (\overline{P} \cup Q) \cup \overline{P})$$

So

$$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \subseteq \overline{P} \cup Q$$

By using Definitions 3.7, we get:

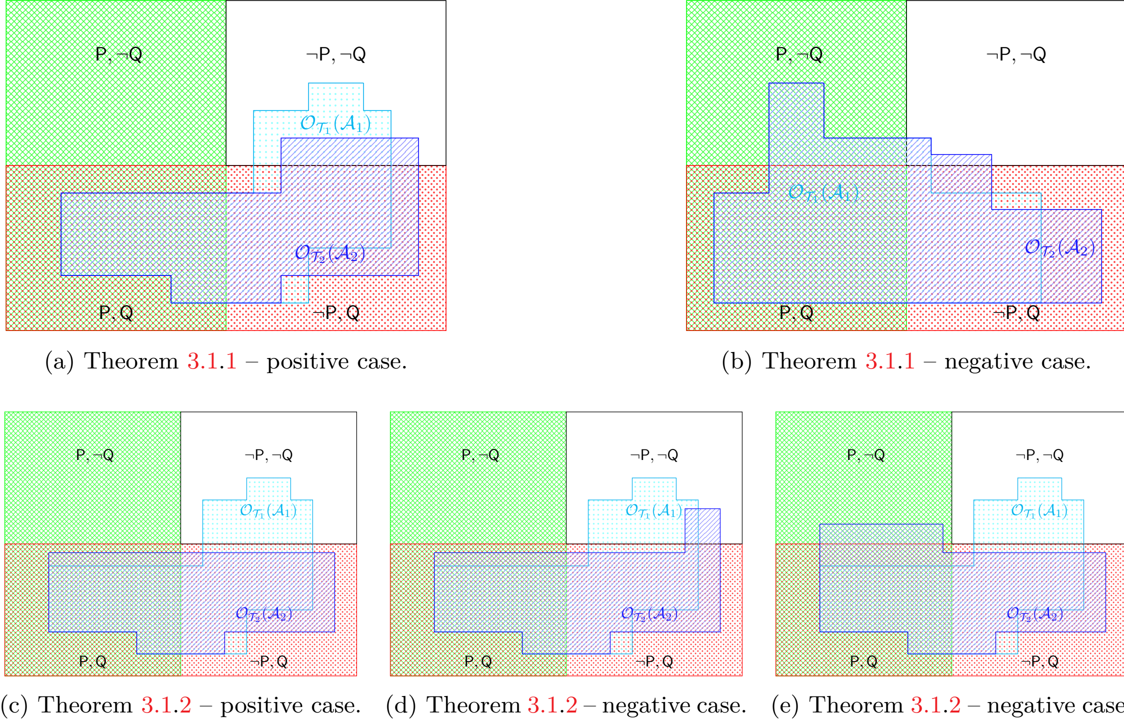
$$\mathcal{A}_1 \models_{\mathcal{T}_1} P \rightarrow Q$$

The previous theorem states two observations for assumption-commitment specifications $S$ of the form $P \rightarrow Q$. Firstly, whether an automaton satisfies $S$ depends exactly on the observable behaviours satisfying $P$. That is, in order to check an automaton $\mathcal{A}_1$ against $S$, we may as well check $\mathcal{A}_2$ (even under a different operational semantics) as long as $\mathcal{A}_1$ and $\mathcal{A}_2$ (under the considered semantics) agree on the observable behaviours satisfying $P$. Secondly, it is possible to verify satisfaction of $S$ by an automaton through checking only $Q$ in an overapproximation of the automaton's observable behaviour.

Moreover, in order to understand the idea behind Theorem 3.1, let us consider the cases depicted in Figure 3.2. We have five cases, where each case represents either a holding situation of the specification or a violation situation. For each figure, the **green area** represents the area where the specification $P$ holds and the specification $Q$ doesn't hold. The **red area** represents the situation where the specification $Q$ holds only. The last area represents the situation where neither $P$ nor $Q$ holds. The **mixed area** (green and read) represents the situation where the both specifications $P$ and $Q$ hold. The **cyan polygon area** and **blue polygon area** represent the observable behaviours of automaton $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively.

Figure 3.2a represents the situation where the set of observable behaviours of $\mathcal{A}_1$ that satisfy $P$ is equivalent to the set of observable behaviours of $\mathcal{A}_2$ that satisfy $P$. On the same time the set of observable behaviours of $\mathcal{A}_1$ satisfy the specification $P \rightarrow Q$ since the green area is not touched at all. At this point, we conclude that the set of observable behaviours of $\mathcal{A}_2$ satisfy also $P \rightarrow Q$ by using Theorem 1.

Figure 3.2b represents the situation where the set of observable behaviours of $\mathcal{A}_1$ that satisfy $P$ is equivalent to the set of observable behaviours of $\mathcal{A}_2$ that satisfy $P$. On the same time the set of observable behaviours of $\mathcal{A}_1$ do not satisfy the specification $P \rightarrow Q$




(a) Theorem 3.1.1 – positive case.

(b) Theorem 3.1.1 – negative case.

(c) Theorem 3.1.2 – positive case.

(d) Theorem 3.1.2 – negative case.

(e) Theorem 3.1.2 – negative case.

Figure 3.2: List of interesting cases for Theorem 3.1.

since the green area is intersected with $\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1)$. At this point, we conclude that the set of observable behaviours of $\mathcal{A}_2$ do not satisfy also $P \rightarrow Q$ by using Theorem 1.

Figure 3.2c represents the situation where the set of observable behaviours of $\mathcal{A}_1$ that satisfy $P$ is a subset of the set of observable behaviours of $\mathcal{A}_2$ and on the same time the set of observable behaviours of $\mathcal{A}_2$ satisfy the specification $Q$, since neither the green area nor the white one is touched. At this point, we conclude that the set of observable behaviours of $\mathcal{A}_1$ satisfy also $P \rightarrow Q$ by using Theorem 2.

Figure 3.2d represents the situation where the set of observable behaviours of $\mathcal{A}_1$ that satisfy $P$ is a subset of the set of observable behaviours of $\mathcal{A}_2$ and on the same time the set of observable behaviours of $\mathcal{A}_2$ do not satisfy the specification $Q$ since the white area is touched. At this point, we cannot conclude any result for $\mathcal{A}_1$, despite the fact that the automata $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfy the specification $P \rightarrow Q$.

Figure 3.2e represents the situation where the set of observable behaviours of $\mathcal{A}_1$ that satisfy $P$ is a subset of the set of observable behaviours of $\mathcal{A}_2$. On the same time the set of observable behaviours of $\mathcal{A}_2$ do not satisfy the specification $Q$, since parts of white and the green areas are included. At this point, we cannot conclude any result for $\mathcal{A}_1$, despite the fact that $\mathcal{A}_1$ does not satisfy the specification $P \rightarrow Q$. The latter two cases address the limitation of over-approximating $P$-rule usage.

The next section introduces two kinds of *source-to-source* transformation functions that entail the premises of the over-approximating-$P$- and the common-$P$-rules.

## 3.5 Transformation functions

In this section, a general concept of transformations for automata is defined where this transformation is based on a certain specification. Whenever we perform a transformation based on a desired specification, we call this transformation *admissible* if satisfaction of the desired specification is preserved after transformation.

However if the observable behaviour of the resultant automaton after applying the transformation function admits an overapproximation of the desired specification, this transformation is called *semi-admissible*. After that, the two transformations *redirecting edges* and *removing edges* are presented in details.

**Definition 3.9: Transformation**

Let $Aut$ be a set of automata. A *transformation* is a function $\mathcal{F} : Aut \to Aut$ which assigns to each *original automaton* $\mathcal{A} \in Aut$ a *transformed automaton* $\mathcal{F}(\mathcal{A}) \in Aut$.

### 3.5.1 Admissible transformations

**Definition 3.10: Admissible Transformation**

Let $\mathcal{T}$ be an operational semantics of $Aut$ and $S$ a specification. Transformation $\mathcal{F}$ on $Aut$ is called *admissible* for $S$ (under $\mathcal{T}$) if and only if for each automaton $\mathcal{A} \in Aut$, the observable behaviours of $\mathcal{A}$ and $\mathcal{F}(\mathcal{A})$ under $\mathcal{T}$ coincide on $S$, i.e. if

$$\forall \mathcal{A} \in Aut \bullet \mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \cap S = \mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap S.$$

The following lemma states the benefit of transformations which are admissible for the assumption of assumption-commitment specifications: the original and the transformed automaton obtained by an admissible transformation satisfy the premise of the common-$P$-rule. Thus, if one can prove such a transformation is admissible, one gains the benefits of Theorem 3.1.1.

**Lemma 3.1: Admissible Transformation in verification**

Let $\mathcal{T}$ be an operational semantics of $Aut$, $S = P \to Q$ an assumption-commitment specification, and $\mathcal{F}$ a transformation on $Aut$. If $\mathcal{F}$ is admissible for $P$, then for $\mathcal{A} \in Aut$,

$$\mathcal{F}(\mathcal{A}) \models S \text{ if and only if } \mathcal{A} \models S.$$

**Proof of Admissible Transformation in verification**

In order to prove this lemma, we will consider the both directions of the bi-implication; namely:

- The first direction, $\mathcal{A} \in Aut, \mathcal{F}(\mathcal{A}) \models S$ if $\mathcal{A} \models S$

  Since $\mathcal{F}(\mathcal{A}) \models S$ and by using Definition 3.3, we get:

$$\mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \subseteq S, \text{ i.e. } \mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \subseteq \overline{P} \cup Q$$

If we take the intersection of both sides with $P$, we get:

$$\mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \cap P \subseteq (\overline{P} \cup Q) \cap P (*)$$

Also, we know that $\mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \cap \overline{P} \subseteq \overline{P} (**)$.
By combining the the latter two facts in $(*), (**)$, we get:

$$\mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \subseteq \overline{P} \cup Q$$

Now by considering the previous chain of relations, we get:

$$\mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \cap P \subseteq \mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \subseteq \overline{P} \cup Q$$

Since $\mathcal{F}$ is admissible transformation function with respect to $S$ and by using Definition 3.10, we get:

$$\mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \cap P = \mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap P$$

Finally, from previous two facts, we get:

$$\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap P \subseteq \overline{P} \cup Q$$

Thus by using Definition 3.3 and Theorem 3.1,

$$\mathcal{A} \models S$$

- The second direction, $\mathcal{A} \in Aut, \mathcal{A} \models S$ if $\mathcal{F}(\mathcal{A}) \models S$ holds analogously.

### Redirecting edges

The first proposed admissible transformation function *redirects* a set of edges in a given automaton to a *new location*. It is defined as follows.

**Definition 3.11: Redirecting Edges**

Let $\mathcal{A} = (Loc, Act, E, L_{ini})$ be an automaton, $F \subseteq E$ a set of edges, and $\dashv \notin Loc$ a fresh location. We use $\mathcal{A}[F \curvearrowright \dashv]$ to denote the automaton $(Loc \cup \{\dashv\}, Act, E', L_{ini})$ where

$$E' = (E \setminus F) \cup \{(\ell, \eta, \dashv) \mid (\ell, \eta, \ell') \in F\}.$$

We say $\mathcal{A}[F \curvearrowright \dashv]$ is obtained from $\mathcal{A}$ by *redirecting* the edges in $F$ (to $\dashv$).

A transformation is a *syntactic* operation, thus the observable behaviour of a transformed automaton may in general, given a sufficiently pathological operational semantics, not resemble the behaviour of the original automaton at all. The following definition of consistency states minimal sanity requirements on operational semantics which we need in order to effectively use the redirection transformation. These requirements are directly

satisfied by the standard semantics of, e.g., timed and hybrid automata

**Definition 3.12: Consistent for Redirection**

An operational semantics $\mathcal{T}$ for *Aut* over states $\mathcal{S}$ is called *consistent* (for redirection) if and only if for each automaton $\mathcal{A} = (Loc, Act, E, L_{ini}) \in Aut$, there is a location $\dashv$ such that $\mathcal{A}[F \curvearrowright \dashv] \in Aut$ and $\mathcal{T}(\mathcal{A}[F \curvearrowright \dashv]) = (Conf', \Lambda', \{\xrightarrow{\lambda}' \mid \lambda \in \Lambda'\}, C'_{ini})$ where

1. the set of configurations over the old locations and the transition relations for $\bot$ and the unchanged edges do not change, i.e.

$$Conf' \cap (Loc \times \mathcal{S}) = Conf, \quad \forall e \in E \setminus F \bullet \xrightarrow{e} = \xrightarrow{e}',$$
$$\text{and} \quad \xrightarrow{\bot}' \cap (Conf \times Conf) = \xrightarrow{\bot},$$

2. $\mathcal{T}(\mathcal{A}[F \curvearrowright \dashv])$ simulates transitions induced by edges from $F$ and vice versa, and the $\bot$-transition relation does not leave $\dashv$, i.e.

$$\xrightarrow{(\ell,\eta,\dashv)}' = \{(c, \langle \dashv, s' \rangle) \mid \exists e = (\ell, \eta, \ell') \in F \bullet (c, \langle \ell', s' \rangle) \in \xrightarrow{e}\},$$
$$\text{and} \quad \forall s, s' \in \mathcal{S}, \ell' \in Loc' \bullet ((\dashv, s), (\ell', s')) \in \xrightarrow{\bot}' \implies \ell' = \dashv$$

3. the fresh location $\dashv$ is not initial, i.e. $C'_{ini} = C_{ini}$.

The following lemma states that for consistent semantics, the redirection transformation affects only behaviours where redirected edges are used.

**Lemma 3.2: Consistent for Redirection**

Let $\mathcal{T}$ be an operational semantics of *Aut* which is consistent for redirection. Let $\mathcal{A} \in Aut$ be an automaton with edges $E$ and $F \subseteq E$. Then there is a location $\dashv$ such that $\Xi_{\mathcal{T}}(\mathcal{A}[F \curvearrowright \dashv], E \setminus F) = \Xi_{\mathcal{T}}(\mathcal{A}, E \setminus F)$.

**Proof of Consistent for Redirection**

In order to prove this lemma, we will consider both directions of the bi-implication; namely:

- The first direction, $\forall \xi \in \Xi_{\mathcal{T}}(\mathcal{A}[F \curvearrowright \dashv], E \setminus F)$ implies $\xi \in \Xi_{\mathcal{T}}(\mathcal{A}, E \setminus F)$.
  Let $\mathcal{T}$ be an operational semantics of *Aut* which is consistent for redirection, $\mathcal{A} = (Loc, Act, E, L_{ini}) \in Aut$ an automaton, and $F \subseteq E$ a set of edges of $\mathcal{A}$.
  As $\mathcal{T}$ is consistent for redirection, there is, by Definition 3.12, a location $\dashv$ such that $\mathcal{A}[F \curvearrowright \dashv] \in Aut$ and conditions 1 to 3 hold.
  Let $\xi = c_0 \xrightarrow{\lambda_1} c_1 \xrightarrow{\lambda_2} \cdots \in \Xi_{\mathcal{T}}(\mathcal{A}, E \setminus F)$ be a computation path.
  Then $c_0 \in C_{ini}$ and $c_i \xrightarrow{\lambda_i} c_{i+1}$ by Definition 3.5, and $\lambda_i \in (E \setminus F) \cup \{\bot\}$ also by Definition 3.5.
  As $\dashv$ is a fresh location and only the edges in $F$ are modified, none of the edges occurring in $\xi$ has destination $\dashv$, thus all configurations $c_i$ in $\xi$ are for locations from *Loc*. $(*)$
  By Definition 3.12.3, $c_0 \in C'_{ini}$, by $(*)$ and 3.12.1, $c_i \in Conf'$, and by

Definition 3.12.1 and 3.12.1, for all labels $\lambda$ occurring in $\xi$, i.e. $\perp$ and edges from $E$, $\xrightarrow{\lambda} = \xrightarrow{\lambda}\!\!\prime$, thus $c_i \xrightarrow{\lambda_i}\!\!\prime c_{i+1}$.
Hence, by Definition 3.5, $\xi$ is a computation path of $\mathcal{A}[F \curvearrowright \dashv]$, which uses only labels from $(E \setminus F) \cup \{\perp\}$, thus $\xi \in \Xi_{\mathcal{T}}(\mathcal{A}[F \curvearrowright \dashv], E \setminus F)$.

- The second direction, $\forall \xi \in \Xi_{\mathcal{T}}(\mathcal{A}, E \setminus F)$ implies $\xi \in \Xi_{\mathcal{T}}(\mathcal{A}[F \curvearrowright \dashv], E \setminus F)$ is shown analogously.

#### Redirecting edges in computational models

Applying redirecting edges can be preformed in all following computational models, since their operational semantics is consistent after redirecting as in the aforenamed lemma:

**Finite and Büchi automata:** the standard semantics is consistent for redirection, thus we need to define/choose a fresh location and redirect desired edges to this new location. The fresh location must not be accepting one, as we are normally want to exclude/restrict some behaviours of the model by redirecting edges to a new sink location.

**Timed automata:** the standard semantics is consistent for redirection, thus we just choose a fresh location with invariant *true* – to accept all incoming edges – and redirect edges to this new location.

**Hybrid automata:** it is similar to timed automata, the *flows* in the fresh location can be chosen freely.

**Programs:** e.g., the Boogie semantics [BCD+05] is consistent for redirection; we just append a `goto` statement with a fresh label after the statements which should be redirected in order to allow jumps to a unique fresh label $\dashv$.

More details about applying redirecting edges on automata will be spelled out in the next two sections.

### 3.5.2 Semi-admissible transformations

**Definition 3.13: Semi-admissible Transformation**

Let $\mathcal{T}$ be an operational semantics of $Aut$ and $S$ a specification. Transformation $\mathcal{F}$ on $Aut$ is called *semi-admissible* for $S$ (under $\mathcal{T}$) if and only if for each automaton $\mathcal{A} \in Aut$, the observable behaviour of $\mathcal{F}(\mathcal{A})$ under $\mathcal{T}$ over-approximates the observable behaviour of $\mathcal{A}$ in $S$, i.e. if

$$\forall \mathcal{A} \in Aut \bullet \mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap S \subseteq \mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})).$$

The following lemma states the benefit of transformations which are semi-admissible for the assumption of assumption-commitment specifications: the original and the transformed automaton obtained by a semi admissible transformation satisfy the premise of the over-

approximating-$P$-rule. Thus, if we can prove such a transformation is semi-admissible, we gain the benefits of Theorem 3.1.2.

**Lemma 3.3: Admissible Transformation**

Let $\mathcal{T}$ be an operational semantics of $Aut$, $S = P \to Q$ an assumption-commitment specification, and $\mathcal{F}$ a transformation on $Aut$. If $\mathcal{F}$ is semi-admissible for $P$, then for $\mathcal{A} \in Aut$, $\mathcal{F}(\mathcal{A}) \models Q$ implies $\mathcal{A} \models S$.

**Proof of Semi-admissible Transformation**

By using Definition 3.13 and as $\mathcal{F}$ is semi-admissible for $P$, we get:

$$\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap P \subseteq \mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A}))$$

Since $\mathcal{F}(\mathcal{A}) \models Q$, by using Definition 3.3, we get:

$$\mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \subseteq Q$$

Thus, from previous facts; namely $\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap P \subseteq \mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \subseteq Q$, we get:

$$\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap P \subseteq Q$$

We know by the definition of specification and the operations over sets that $\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap \overline{P} \subseteq \overline{P}$.
By combining the previous two facts we get:

$$(\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap \overline{P}) \cup (\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap P) \subseteq \overline{P} \cup Q$$

Finally,

$$\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \subseteq \overline{P} \cup Q$$

By using Definition 3.3, we get:

$$\mathcal{A} \models S$$

### Removing edges

The first proposed semi-admissible transformation function *removes* a set of edges in a given automaton. It is formally defined as follows.

**Definition 3.14: Removing edges**

Let $\mathcal{A} = (Loc, Act, E, L_{ini})$ be an automaton and $F \subseteq E$ a set of edges. We use $\mathcal{A} \setminus F$ to denote the automaton $(Loc, Act, E \setminus F, L_{ini})$. We say $\mathcal{A} \setminus F$ is obtained from $\mathcal{A}$ by *removing* $F$.

As for redirection, we want that all computation paths of the original automaton that take only non-removed edges are preserved in the new automaton. A sufficient criterion is the

following notion of consistency *for removal*.

**Definition 3.15: Consistent Operational Semantics for Removal**

An operational semantics $\mathcal{T}$ for $Aut$ is called consistent (for removal) if and only if for each automaton $\mathcal{A} \in Aut$, $\mathcal{A} \setminus F \in Aut$ and

$$\mathcal{T}(\mathcal{A} \setminus F) = (Conf, \Lambda \setminus F, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda \setminus F\}, C_{ini}),$$

given $\mathcal{T}(\mathcal{A}) = (Conf, \Lambda, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda\}, C_{ini})$. That is, if the operational semantics of $\mathcal{A} \setminus F$ is obtained from the operational semantics of $\mathcal{A}$ by removing some transition relations and leaving everything else unchanged.

**Lemma 3.4: Consistent Operational Semantics for Removal**

Let $\mathcal{T}$ be an operational semantics of $Aut$ which is consistent for removal. Let $\mathcal{A} = (Loc, Act, E, L_{ini}) \in Aut$ be an automaton and $F \subseteq E$ a set of edges. Then $\Xi_{\mathcal{T}}(\mathcal{A} \setminus F, E \setminus F) = \Xi_{\mathcal{T}}(\mathcal{A}, E \setminus F)$.

**Proof of Consistent Operational Semantics for Removal**

We will prove this lemma by considering the both directions:

- First direction: $\forall \xi \in \Xi_{\mathcal{T}}(\mathcal{A} \setminus F, E \setminus F)$ implies $\xi \in \Xi_{\mathcal{T}}(\mathcal{A}, E \setminus F)$
  Let $\xi \in \Xi_{\mathcal{T}}(\mathcal{A} \setminus F, E \setminus F)$.
  Then only labels $e \in E \setminus F$ and $\perp$ occur in $\xi$ as in Definition 3.14.
  By using Definition 3.15, the occurring configurations are also configurations of $\mathcal{T}(\mathcal{A})$ and the transition relations for these labels are equal in $\mathcal{T}(\mathcal{A})$ and $\mathcal{T}(\mathcal{A} \setminus F)$.
  Thus $\xi \in \Xi_{\mathcal{T}}(\mathcal{A})$.
- Second direction: $\forall \xi \in \Xi_{\mathcal{T}}(\mathcal{A}, E \setminus F)$ implies $\xi \in \Xi_{\mathcal{T}}(\mathcal{A} \setminus F, E \setminus F)$ is shown analogously.

#### Removing edges in computational models

Applying transformation by removing edges can be preformed in all following computational models, since their operational semantics is consistent after removing:

**Finite, Büchi, timed and hybrid automata:** the standard semantics of previous models is consistent for removing, thus it is needed to identify the desired edges to that has to be removed.

**Programs:** e.g., the Boogie semantics is consistent for removing; we only append a `goto` statement with a fresh label before the statements which should be removed in order to preform jumps over this removed block. An alternative solution is to use `assume`(*false*) before the statements that should be removed in order to make all computation paths through this block infeasible.

More details about applying removing edges on automata will be spelled out in the next two sections.

## 3.6 New reachability concept: supporting edges

### 3.6.1 Supporting edges

In this section, we introduce the novel concept of supporting edges, based on *edge reachability*. This concept identifies a relation between a specification and edges. Informally, an edge supports a specification if and only if there is a computation path which satisfies the specification and where that edge is taken.

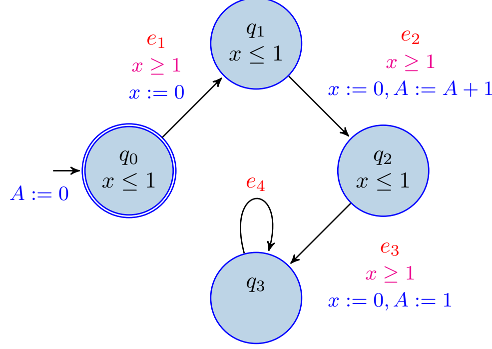



Figure 3.3: Support notions in timed automaton.

| edges / Support notions | $e_1$ | $e_2$ | $e_3$ | $e_4$ |
|---|---|---|---|---|
| edge supports specification $\Box A = 0$ | ✗ | ✗ | ✗ | ✗ |
| edge supports proposition $A = 0$ | ✓ | ✗ | ✗ | ✗ |
| edge potentially supports proposition $A = 0$ | ✓ | ✓ | ✗ | ✓ |

Table 3.1: Summary of supporting edges results in Example 3.2.

**Definition 3.16: Supporting Edges**

Let $\mathcal{T}$ be an operational semantics of $Aut$ and $\mathcal{A} \in Aut$ an automaton with edges $E$. An edge $e \in E$

1. *supports specification* $S$ (under $\mathcal{T}$) if and only if there is a computation path where label $e$ occurs and whose observable behaviour is in $S$, i.e. if
$$\exists \xi = c_0 \xrightarrow{\lambda_1} c_1 \xrightarrow{\lambda_2} \cdots \in \Xi_{\mathcal{T}}(\mathcal{A})\ \exists i \in \mathbb{N} \bullet \lambda_i = e \wedge \downarrow\xi \in S.$$
2. *supports atomic proposition* $p$ (under $\mathcal{T}$) if and only if there is a computation path where label $e$ occurs between two configurations that are in $p$, i.e. if
$$\exists \xi = c_0 \xrightarrow{\lambda_1} c_1 \xrightarrow{\lambda_2} \cdots \in \Xi_{\mathcal{T}}(\mathcal{A})\ \exists i \in \mathbb{N} \bullet \lambda_i = e \wedge \{c_{i-1}, c_i\} \subseteq p.$$
3. *potentially supports atomic proposition* $p$ (under $\mathcal{T}$) if and only if there are two configurations of $\mathcal{T}(\mathcal{A}) = (Conf, \Lambda, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda\}, C_{ini})$ which are in $p$ and in $\xrightarrow{e}$-relation, i.e. if $\exists c, c' \in Conf \cap p \bullet (c, c') \in \xrightarrow{e}$.

**Example 3.2: Supporting Edges example**

In order to shed some light on supporting definition with its three variants, let us consider the timed automata depicted in Figure 3.3 and the supporting results shown in Table 3.1. This automata consists of four locations where each location except $q_3$ has an invariant $x \leq 1$. Also, each edge has a guard $x \geq 1$ which means that any feasible transition in the operational semantics of the automata will be executed exactly at the point of time when $x = 1$. For simplicity issue, we label the edges from $q_0$ to $q_1$, $q_1$ to $q_2$, $q_2$ to $q_3$ and $q_3$ to $q_3$ with $e_1, e_2, e_3$ and $e_4$ respectively. We want to consider the specification $\Box A = 0$ (which mean always $A$ equals 0) and

the atomic proposition $A = 0$ (cf. Definition 3.7). By considering the strongest definition of support; i.e., support $\Box A = 0$ (cf. Definition 3.16.1), it is observed that none of the previous edges support this specification, since there exists no computation path such that the edge is traversed and the specification is satisfied. By considering the second definition of support (cf. Definition 3.16.2), it is observed that only edge $e_1$ supports the atomic proposition $A = 0$ since we have the following feasible computation path:

$$\langle q_0, \{x = 0, A = 0\}\rangle \xrightarrow{\tau} \underbrace{\langle q_0, \{x = 1, A = 0\}\rangle}_{c_1} \xrightarrow{e_1} \underbrace{\langle q_1, \{x = 0, A = 0\}\rangle}_{c_2} \ldots$$

where $\{c_1, c_2\} \subseteq \{A = 0\}$. The other edges do not support the atomic proposition $A = 0$.
By considering the third definition of support (cf. Definition 3.16.3), the weakest definition of support, it is observed that all edges except $e_3$ potentially support the proposition $A = 0$ as follows:

- an evidence that $e_1$ potentially supports $A = 0$ is

$$\langle q_0, \{x = 1, A = 0\}\rangle \xrightarrow{e_1} \langle q_1, \{x = 0, A = 0\}\rangle$$

- an evidence that $e_2$ potentially supports $A = 0$ is

$$\langle q_1, \{x = 1, A = -1\}\rangle \xrightarrow{e_2} \langle q_2, \{x = 0, A = 0\}\rangle$$

- an evidence that $e_4$ potentially supports $A = 0$ is

$$\langle q_3, \{x = 1, A = 0\}\rangle \xrightarrow{e_4} \langle q_3, \{x = 0, A = 0\}\rangle$$

Table 3.1 summaries the results of supporting edges for the automaton depicted in Figure 3.3. Table 3.1 consists of five columns where the first one defines the type of support notion according to Definition 3.16, the other columns characterize the edges of the automaton in Figure 3.3. From this table, it is observed that $e_3$ was detected as non-support edge for all support notions, since it has a restricted assignment that violates the atomic proposition $A := 0$.

**Corollary 3.1: Supporting Edges relations**

Let $\mathcal{A} \in Aut$ be an automaton and $p$ an atomic proposition.

1. If an edge $e$ of $\mathcal{A}$ supports the invariant $\Box p$ (under $\mathcal{T}$), then $e$ supports the proposition $p$ (under $\mathcal{T}$), but in general not vice versa.
2. If $e$ supports $p$, then $e$ potentially supports $p$, but in general not vice versa.

**Proof of Supporting Edges relations**

The proof of this corollary is straightforward, namely:

- If an edge $e$ of $\mathcal{A}$ supports the invariant $\Box p$, then there exists at least one computation path where its configurations sequence is an element of the invariant. Thus, the direct predecessor and direct successor configurations while

traversing $e$ are in $p$.

- If $e$ supports $p$, it means that we can find a feasible and direct predecessor and successor configurations while traversing $e$. Thus these feasible configurations are in $Conf$.

### 3.6.2 Supporting edges and transformation functions

In order to combine previous ideas and concepts together to portrait a complete picture, we will show that transformation functions by using redirecting and removing edges which *do not support* a specification $S$ are admissible and semi-admissible for $S$, respectively.

**Theorem 3.2: Admissibility and redirection**

Let $\mathcal{T}$ be an operational semantics of $Aut$ with states $\mathcal{S}$ and let $S$ be a specification over $\Sigma$. Let $F$ be a set of edges of automaton $\mathcal{A} \in Aut$ which do not support $S$ under $\mathcal{T}$.

1. $\mathcal{F}^F_{\mathsf{rd}} : \mathcal{A} \mapsto \mathcal{A}[F \curvearrowright \dashv]$ is admissible for $S$ if $\mathcal{T}$ is consistent for redirection and if $S$ *does not refer to* the fresh location $\dashv$, i.e. if $(\{\dashv\} \times \mathcal{S}) \cap \Sigma = \emptyset$.
2. $\mathcal{F}^F_{\mathsf{rm}} : \mathcal{A} \mapsto \mathcal{A} \setminus F$ is semi-admissible for $S$ if $\mathcal{T}$ is consistent for removal.

**Proof of Admissibility and redirection**

Let $\mathcal{T}$ be an operational semantics with states $\mathcal{S}$ for $Aut$ which is consistent for redirection, let $S$ be a specification over $\Sigma$, and $F$ a set of edges of automaton $\mathcal{A} = (Loc, Act, E, L_{ini}) \in Aut$ which do not support $S$.

1. In order to prove the first item of the theorem, assume that $S$ does not refer to the fresh location $\dashv$.
   We have to show $\mathcal{O}_\mathcal{T}(\mathcal{A}[F \curvearrowright \dashv]) \cap S = \mathcal{O}_\mathcal{T}(\mathcal{A}) \cap S$. As the observable behaviour is defined point-wise by Definition 3.6, we can consider computation paths.
   "$\subseteq$": Let $\xi$ be a computation path of $\mathcal{A}[F \curvearrowright \dashv]$ under $\mathcal{T}$ whose observable behaviour is in $S$.
   As $S$ does not refer to $\dashv$ and as by Definition 3.12.2, only edges obtained by redirecting the edges in $F$ lead to $\dashv$, none of these edges occurs in $\xi$. Furthermore, none of the edges in $F$ occurs in $\xi$ because they have been redirected, i.e. they are not edges of $\mathcal{A}[F \curvearrowright \dashv]$, thus $\xi \in \Xi_\mathcal{T}(\mathcal{A}[F \curvearrowright \dashv], E \setminus F)$.
   As $\mathcal{T}$ is consistent for redirection, we have $\xi \in \Xi_\mathcal{T}(\mathcal{A}, E \setminus F)$ by Lemma 3.2.
   Thus $\xi$ is also a computation path of $\mathcal{A}$ under $\mathcal{T}$, its observable behaviour is still in $S$.
   "$\supseteq$": Let $\xi$ be a computation path of $\mathcal{A}$ under $\mathcal{T}$ whose observable behaviour is in $S$.
   As the edges in $F$ do not support specification $S$, we have, by using Definition 3.16.1, $\xi \in \Xi_\mathcal{T}(\mathcal{A}, E \setminus F)$.
   As $\mathcal{T}$ is consistent for redirection, we have $\xi \in \Xi_\mathcal{T}(\mathcal{A}[F \curvearrowright \dashv], E \setminus F)$ by Lemma 3.5.1.

Thus $\xi$ is also a computation path of $\mathcal{A}[F \curvearrowright \dashv]$ under $\mathcal{T}$, its observable behaviour is still in $S$.

2. In order to prove the second item of the theorem, we have to show $\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap S \supseteq \mathcal{O}_{\mathcal{T}}(\mathcal{A} \setminus F) \cap S$. As the observable behaviour is defined point-wise by Definition 3.6, we can consider computation paths.

   "$\supseteq$": Let $\xi$ be a computation path of $\mathcal{A}$ under $\mathcal{T}$ whose observable behaviour is in $S$.

   As the edges in $F$ do not support specification $S$, we have, by Definition 3.16.1, $\xi \in \Xi_{\mathcal{T}}(\mathcal{A}, E \setminus F)$.

   As $\mathcal{T}$ is consistent for redirection, we have $\xi \in \Xi_{\mathcal{T}}(\mathcal{A} \setminus F, E \setminus F)$ by using Lemma 3.4.

   Thus $\xi$ is also a computation path of $\mathcal{A}[F \curvearrowright \dashv]$ under $\mathcal{T}$, its observable behaviour is still in $S$.

To apply Theorem 3.7, a set of edges which do not support the given specification has to be identified. In the following example, we will apply transformation functions by redirecting and removing on the timed automaton model depicted in Figure 3.3 through considering the supporting edges results depicted in Table 3.1.

**Continuing with Example 3.2: Supporting dependent transformation**

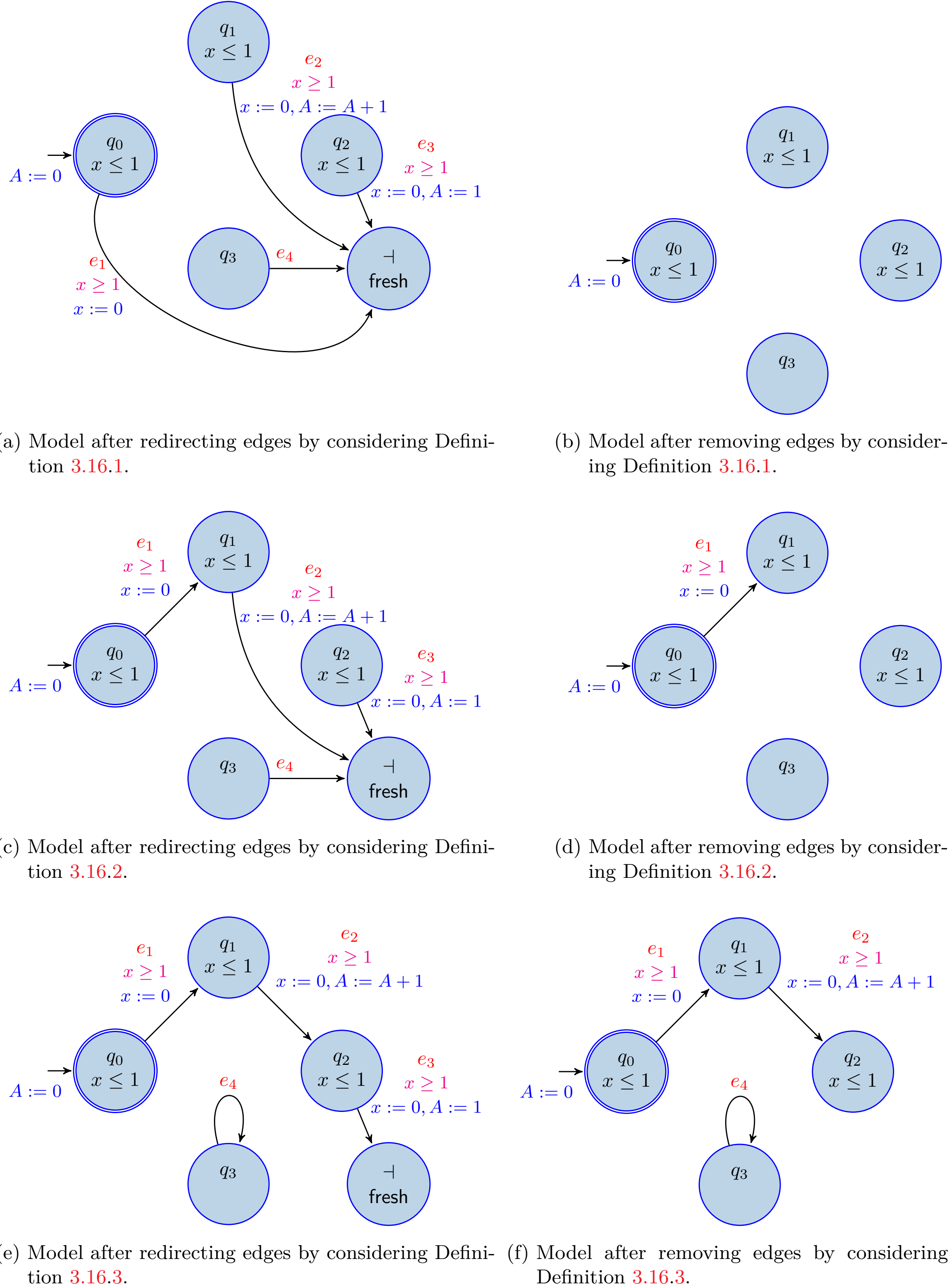
Consider the timed automaton in Example 3.2, Figure 3.3 and Table 3.1. We will apply transformations however depending on the edges that do not support a specification or an atomic proposition. Unexpectedly, we begin to speak about the non-supporting edges instead of supporting ones since in this example, the transformation will exclude/deactivate the paths where the specification or the atomic proposition will be violated. Case 1: **transformations based on edges do not support the specification $\Box A = 0$.** If we transform the timed automaton model by redirecting or removing the set of edges that don't support the specification $\Box A = 0$, we will get the timed automaton models in Figure 3.4a and Figure 3.4b respectively.

Case 2: **transformations based on edges do not support the proposition $A = 0$.** If we transform the timed automaton model by redirecting or removing the set of edges that don't support the proposition $A = 0$, we will get the timed automaton models in Figure 3.4c and Figure 3.4d respectively.

Case 3: **transformations based on edges do not potentially support the proposition $A = 0$.** If we transform the timed automaton model by redirecting or removing the set of edges that don't support the proposition $A = 0$, we will get the timed automaton models in Figure 3.4e and Figure 3.4f respectively.

### 3.6.3 Verification based on support-notion

In general, detecting edges which do not support a specification (cf. Definition 3.16.1 is as expensive as *reachability checking*. Though if the specification is an invariant, the contrapositions of the implications in Corollary 3.1 are particularly useful: if an edge *does not potentially support a proposition*, then it does not support the proposition, and if an




(a) Model after redirecting edges by considering Definition 3.16.1.

(b) Model after removing edges by considering Definition 3.16.1.

(c) Model after redirecting edges by considering Definition 3.16.2.

(d) Model after removing edges by considering Definition 3.16.2.

(e) Model after redirecting edges by considering Definition 3.16.3.

(f) Model after removing edges by considering Definition 3.16.3.

Figure 3.4: Transformed timed automata models after considering different notions of supporting.

edge does not support a proposition $p$, then it does not support the invariant $\Box p$. A sufficient criterion for an edge not (potentially) supporting a proposition $p$ is to be a *cause* or a *witness*. An edge $e$ is a *cause* of a violation of $p$ if the $p$ is always violated after taking this edge, i.e., if the interpreted action of $e$ causes $p$ not to hold. Similarly, an edge is a *witness* of a violation of $p$ if $p$ is necessarily violated when $e$ is taken, i.e., if the guard of $e$ may imply $\neg p$. Removal of witnesses is even admissible. There are sufficient syntactical criteria to detect causes and witnesses. Furthermore, detection of potential support can be reduced to an SMT problem for the formula given by Definition 3.16.3 and attacked by SMT solvers like iSAT3 [SKB13] or SMTInterpol [CHN12] or Z3 [dMB08].

For the special case of timed automata and bounded-integer propositions, a procedure based on the well-known reaching definitions analysis detects all edges which support an atomic proposition [Mah12]. Considering *all* edges which do not support a given specification is optimal in the sense that removing or redirecting any more edges breaks (semi-)admissibility. But it is not necessary to determine *all* non-supporting edges in order to obtain an optimal reduction of behaviour. It is sufficient to determine all *points of no return* (PNR) for a given specification, i.e., edges which are the first on a computation path which do not support the specification. Causes and witnesses are often PNRs.

**Remark 3.1: Networks of automata**

All previous discussions consider a single automaton in the syntax and semantics. However, most practical models are networks of automata. In the following, we discuss briefly how our approach is applied to networks of automata.
*For timed automata*, each network has an equivalent timed automaton, the *parallel composition*. Edges in the parallel composition are constructed from internal transitions of automata in the network, or (with broadcast) from synchronisation edges of one or more automata in the network over a channel. An edge $e$ in the network *supports a specification* if and only if there is an edge in the parallel composition which supports the specification and which is constructed from $e$. Edges not supporting a specification in this sense can safely be disabled by applying redirection or removal to the automata in the network.
The same approach applies to *hybrid automata* and as neither redirection nor removal changes the set of interpreted actions, the sets of labels are preserved and thus no new computation paths emerge.

**Remark 3.2: Automaton Templates**

In several tools; e.g., Uppaal, networks of automata are composed of automaton **template instances**. An edge in a template supports a specification if and only if there is an edge instance which supports the specification. That is, an edge can only be safely redirected or removed in the template, if all instances of this edge in the network do not support the specification. That means we can apply our transformation on template (component) level .

[2] In [FW16] a feasible implementation for detecting non-supporting edges was integrated with Z3 solver and used in this thesis.

**Remark 3.3: Computation Paths vs. Runs**

The standard semantics of timed and hybrid automata distinguish between computation paths and *runs*, where the latter are computation paths with the *progress* property [OD08]. Interestingly, for timed automata, removing and redirecting edges have the same semantical effect if only *runs* are considered. However, in practice that will not give an obvious benefit, because verification tools – for sake of verifying invariants – typically check computation paths, not only runs.

## 3.7 Compositional verification

**Algorithm 1** Verification procedure of $S : P \to Q$ in automata models.

*Input*: automaton model $\mathcal{A}$ with edges $E$ and specification $S : P \to Q$.
*Output*: either *valid* or *invalid*.

```
 1: procedure VERIFY(S, A)
 2:     DETECT EDGES F ⊆ E NOT SUPPORTING P
 3:     if A \ F ⊨ Q then
 4:         RETURN valid
 5:     else
 6:         if A[F ↷ ⊣] ⊨ S then
 7:             RETURN valid
 8:         else
 9:             RETURN invalid
10:         end if
11:     end if
12: end procedure
```

In previous sections, we introduced the concept of transformation by either redirecting or removing edges and different notions of edges supporting a specification or an atomic proposition are explained as well. Firstly, removal and redirection of all edges which do not support a given specification is optimal in the following sense: they yield always smaller models with comparison to the original one. Smaller means; e.g., in removal case that we obviously decrease the syntactical size of a given automaton model. Consequently, we deal with smaller state space of the problem. Additionally, although redirection adds a fresh location, it also decreases the size of a given automaton in many practical examples if we also remove all locations and edges which are, after redirecting non-supporting edges, are no longer graph-reachable in the automaton. This section proposes an approach to use the previous theory in practice with any model checker. Later, our procedure will be integrated with the Uppaal tool.

To use redirection and removal, we propose to apply the procedure shown in Algorithm 1 to all assumption-commitment properties $P \to Q$.

Algorithm 1 depends generally on two steps; (1) *the technique to detect the non-supporting edges* and (2) *the chosen transformation function*. Firstly, we need to detect the set of edges that do not support the assumption $P$ as in Line 2 in Algorithm 1. At this point, we

can control the cost of this step by several options. For example, if we tend to detect all edges that do not support the invariant $P$ as described in Definition 3.16.1, one has to take into account, as we have pointed out before, that this detection is at least **as complex as solving the location-reachability** problem, the following theorem emphasizes this observation.

**Theorem 3.3: Complexity of the non-support problem**

Solving the non-support problem for all edges in a given automaton $\mathcal{A}$ is at least as hard as solving the location-reachability problem.

**Proof of Complexity of the non-support problem**

In order to prove this theorem, we need to reduce the location-reachability problem onto the problem of solving the non-support problem for all edges in a given automaton.
We are given an automaton $\mathcal{A} = (Loc, Act, E, L_{ini})$ with its operational semantics $\mathcal{T}(\mathcal{A})$ and one of its locations $\ell \in Loc$. The location-reachability problem asks whether $\ell$ is reachable in $\mathcal{A}$ under $\mathcal{T}(\mathcal{A})$ i.e. whether there exists a computation path such that it starts from the initial configurations and location $\ell$ is reached by this computation path configuration as stated in Definition 3.5.
Let us consider the automaton $\mathcal{A}$ and the invariant (specification) $\Box true$. We solve the non-support problem for all edges in $\mathcal{A}$ and maintain/update a set $E_{supp}$ which contains all edges that support the invariant $\Box true$ i.e. all non-reachable edges will be excluded. Thereby, the set $E_{supp}$ contains exactly all edges which can occur as labels on any computation path of $\mathcal{A}$.
Now, in order to determine whether location $\ell$ is reachable or not, we have to check whether there exists an edge which its source or destination locations is $\ell$ and on the same time this edge is reachable on any computation path of $\mathcal{A}$.
This is equivalent to checking whether there exists any edge $e$ in the set of supporting edges $E_{supp}$. If such an edge exists, then location $\ell$ is reachable, otherwise location $\ell$ is not reachable.

This means that the exact detection of non-supporting edges is quite expensive and provides no efficiency benefits if it is used as preprocessing step for verification of $P \to Q$. However, knowing the exact number of non-supporting edges in a given automaton for a given invariant $P$ enables us to assess the precision of the approximative detection procedures that depends on Definitions 3.16.2 and 3.16.3. Thus the approximative detection procedures are reasonable and low-priced detection, in particular for automata models where the variables occurring in $P$ are appearing always in memoryless operations; e.g., $v := z$ where $z \in \mathbb{Z}$. In these cases, one can identify almost the same edges that are detected with the exact procedure as in Definition 3.16.1 and they can be safely removed or redirected. The second reasonable option to consider supporting a proposition which is still less expensive than supporting a specification.

After detecting the non-supporting edges of $P$, we apply a suitable transformation as in Line 3. The first transformation is to *remove edges* which do not support the assumption and check whether the resulting model satisfies the commitment $Q$. If $Q$ is satisfied, we deduce that the original model satisfies the property by the over-approximating-$P$-rule

from Theorem 3.1 as in Line 4. Otherwise, we need to *redirect the edges* that do not support $P$ as in Line 6. Then checking whether the resulting model satisfies $P \rightarrow Q$ yields the final verification result by the common-$P$-rule from Theorem 3.1 as in Lines 7 and 9. The reason for using removal before redirection is that removing edges leads to a smaller state space than redirecting, and consequently less time and memory consumption.

## 3.8 Case studies

As mentioned in Section 3.2, our approach can be applied to automata models inducing consistent operational semantics. One valid instance of these models are timed automaton models. We explain in this section in more detail how the verification process would be optimised in terms of time consumption and memory usage while applying our proposed compositional verification procedure in timed automata models. In each case, we will explain the *use-case* and describe its network of timed automata model. After that, depending on the the assumption of the verified desired property, we will detect the edges which do not support the assumption (under the means of potentially supporting), and then apply the transformation function by redirecting or removing edges. It is obvious that the latter procedure follows Algorithm 1. Finally, we will compare the results of verifying the contract property in the original timed automata model by Uppaal [LSW95] with the results while verifying the same contract in the resultant timed automata model after transforming.

### 3.8.1 Wireless sensor network: Alarm system

We consider a wireless fire alarm system modelled, improved and verified in [AWD$^+$14]. The wireless fire alarm system (WFAS) model in [AWD$^+$14] follows the European standard EN-54, part 25 [DIN97], which regulates the main obligations for commercially available WFAS. Wireless fire alarm systems are increasingly preferred over wired ones due to advantages such as spatial flexibility. The main purpose of a fire alarm system is to reliably and timely notify occupants about the presence of indications for fire, such as smoke or high temperature. Because of possible failure in system components like physical damages or disconnection, we cannot guarantee that the fire alarm system fulfils this requirement. Hence, the modelled fire alarm system does not only need to employ self-monitoring mechanisms, but also notify maintainers if it is not able to fulfil its main purpose. Moreover, *false alarms* and maintainer notifications should be avoided as they induce unnecessary costs, in particular when we speak about large scalable WFAS models.

Most aspects of the WFAS protocol are modelled using timed automata [AD94], which were subsequently verified by using Uppaal. However, many difficulties and challenges appear even by using state-of-the-art verification technique for many reasons: First, the number of components is very large, e.g., 120 sensors and 30 repeaters. Second, the standard explicitly specifies complex environment assumptions which need to be taken into consideration while modelling the system. Third, the relevant properties are real-time safety properties.

#### Description of the system model

The WFAS can be briefly described as follows. It consists of one central unit, repeaters and sensors, where a word *component* and *node* are used interchangeably, and they refer to either the central unit, a repeater or a sensor. In the WFAS topology, each component is assigned a unique master. The master-slave relation forms a tree with the central unit as root. Only the central unit or a repeater can act as a master in this topology, and a sensor or a repeater can act as a slave in this relation.

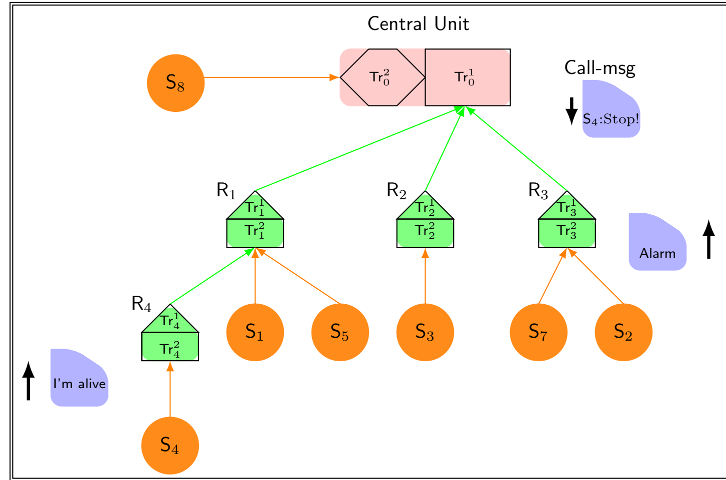



Figure 3.5: Example of wireless fire alarm system topology.

Figure 3.5 shows a valid instance of WFAS topology, where $C$ refers to the central unit, $R_1, ..., R_4$ are repeaters and $S_1, ..., S_8$ are sensors. Each master component has two transceivers as shown in Figure 3.5. A Transceiver realizes three functions: the slave-role towards another repeater or the central unit, the master-role towards other repeaters, and the forwarding of events.

If we want to classify the exchange of messages in WFAS, they will be as follows:

- "create and send message" would be carried out in the following situations:
  - Any slave component must periodically send an LZ-message which denotes that this component still works properly.
  - Any sensor must send Alarm-message when it detects fire or smoke.
  - The central unit can send a Call-message to any sensor in order to give it one command to be executed; namely stop sending LZ-messages.
- "forward incoming message" would be carried out in the following situations:
  - Any repeater must forward the incoming messages, e.g., Alarm, Call and LZ-messages.
- "replying to incoming messages" would be carried out in the following situations:
  - Any master component must send acknowledgement-message for LZ-messages where the former contains a time stamp which allows the slave to synchronize

its clock with the master's clock.

- Any sensor must acknowledge the received Call-message.

Looking at the timing schedule of WFAS, the Time Division Multiple Access (TDMA) Protocol [Stu96] is used. TDMA protocol is used in networks where several nodes share the same frequency channel, by dividing the time into time intervals, such that in each interval, one node at most is allowed to use the frequency channel. In WFAS, the time is considered to start from a fixed point in time, at which the system starts to evolve. Starting from this point, the time is divided into an infinite sequence of equal *time cycles* (which are called *time-frames* in some literature). Within the time cycles, messages between central unit and sensors take place frequently. If we look deeper into the time cycle, each time cycle is divided into equal intervals, called time windows. Moreover, each time window is subdivided into intervals called timeslots which are assigned to different protocol functions. Each node measures the time using its clock, and the clock presents a count of **tics** starting from a fixed point in time. In this sense, one can represent the length of each time interval by a fixed number of tics [MJ11]. Finally, every sensor and repeater is assigned a unique window.

As aforementioned, this case study follows the European standard EN-54 which requires the following:

R-1 The loss of the ability to transmit a signal from a component to the central unit is detected in less than 300 seconds and displayed at the central unit within 100 seconds thereafter.

R-2 A single alarm event is displayed at the central unit within 10 seconds.

R-3 Two alarm events occurring within 2 seconds of each other are each displayed at the central unit within 10 seconds after their occurrences.

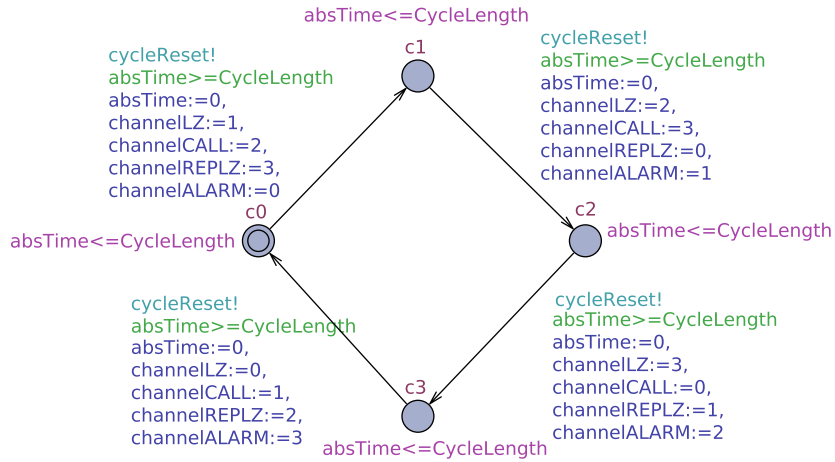
R-4 Out of exactly ten alarms occurring simultaneously, the first should be displayed at the central unit within 10 seconds and all others within 100 seconds.

R-5 There must be no erroneous displays of events at the central unit.

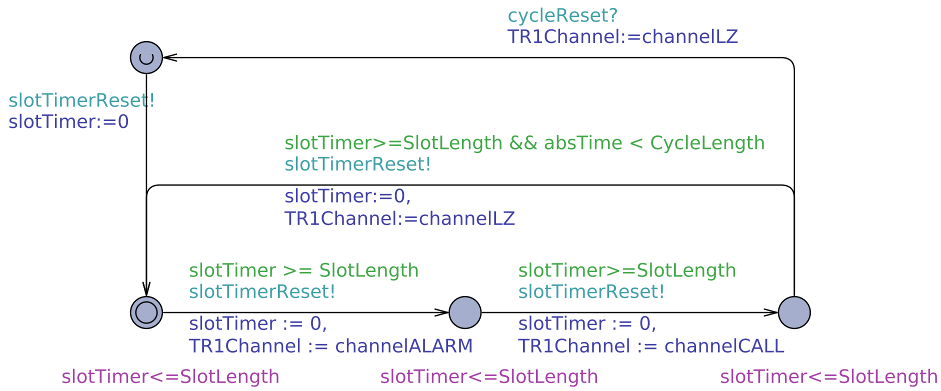
Requirements 1 to 5 must hold in all situations; e.g. including the case where we have radio interference by other users of the frequency band. Radio interference by other users of the frequency band is simulated by a jamming device specified in the standard.

#### Problem statement

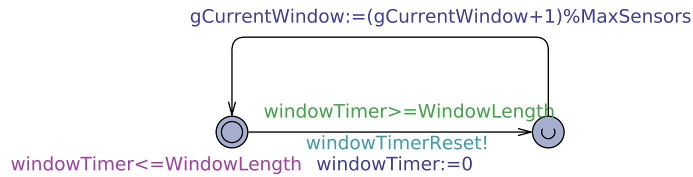
In order to perform failure detection, repeaters in the slave-role and sensors use the same functionality. Slaves periodically send LZ-message to their master in the corresponding slot of their assigned window. If no acknowledgement message is received from the master, a second and third LZ message are transmitted in the subsequent slots using a different channel. Masters listen (this follows listen before talk (LBT) procedure [LMM06]) on the corresponding channel during the slots of their assigned slaves. A master enters its error detection status when a specified number of LZ-messages from one slave have consecutively been lost. The master then initiates the forwarding of the failure detection event. Event forwarding takes place without regarding slot assignments, using the transceivers.



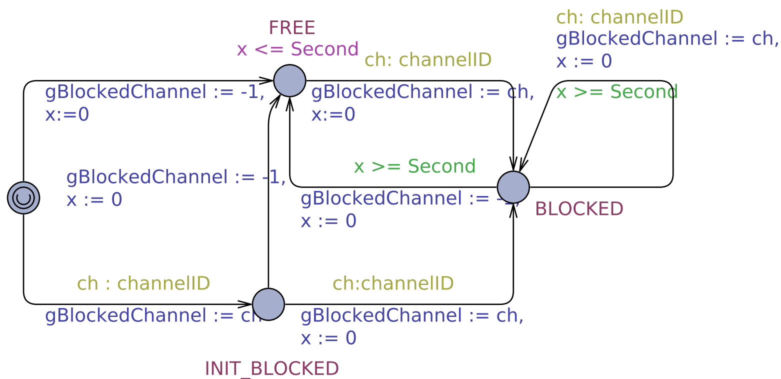

(a) MasterClock.



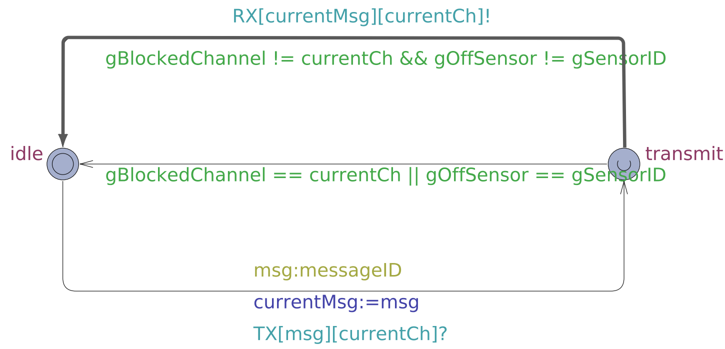

(b) SlotClock.



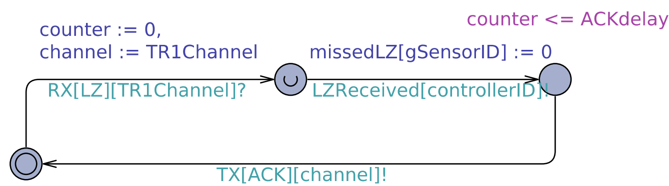

(c) WindowClock.



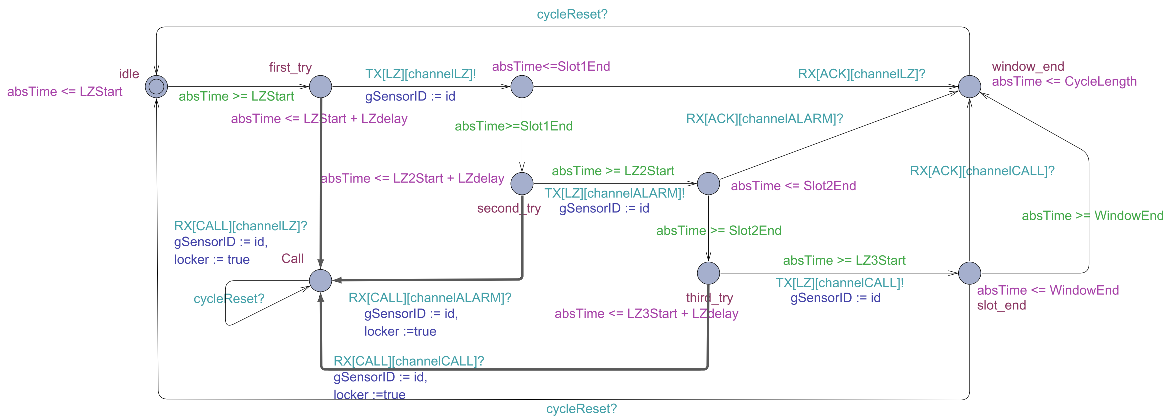

(d) ChannelBlocker.



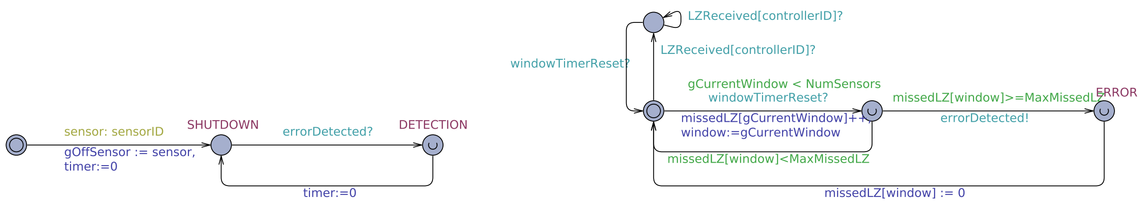

(e) Medium.




(f) Transceiver.

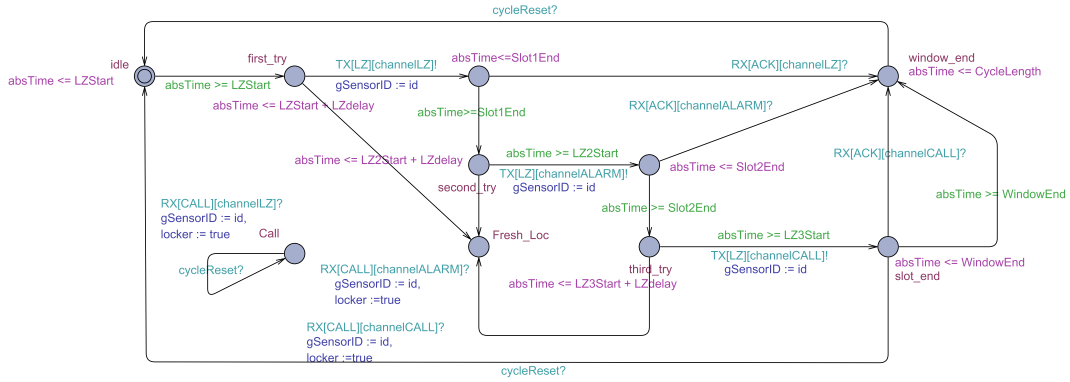



(g) Sensor.

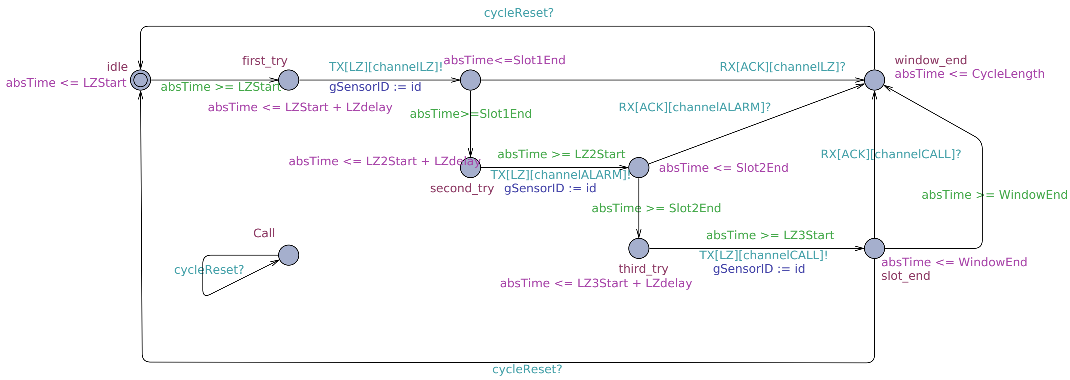



(h) Fault. (i) Controller.

Figure 3.5: Uppaal model of WFAS as in [AWD+14], however sensor model in Figure 3.5g extended by Call-messages behaviour. The thick edges represent Call-message scenarios.




(a) Sensor after redirecting non-supporting edges.




(b) Sensor after removing non-supporting edges.

Figure 3.6: Sensor automaton after applying transformation function. The other automata remain the same.

### Uppaal model

Figure 3.5 represents the wireless fire alarm system model as in [AWD+14], however it is extended by adding call messages mechanism, where a call message prevents a specific sensor from sending LZ messages as aforesaid.

In this model, we have a MasterClock as in Figure 3.6a, SlotClock as in Figure 3.6b and WindowClock as in Figure 3.6c timed automata to organise the time-based scheduling in the TDMA protocol. In addition to that, Channelblocker as in Figure 3.5i and Medium as in Figure 3.5e timed automata represents the channel and the medium behaviour respectively. Figure 3.5g represents Sensor timed automaton model and Figure 3.5f represents transceiver timed automaton that organises sending and receiving LZ messages. Finally, Figure 3.5i and 3.5h represents the Controller and the Fault timed automata, whereas the error is detected whenever the number of missed LZ exceeds 2.

In the previous model, we are interested to verify the following well-functioning property: in case that the central unit has never sent any **Call** message and the sensor has a failure, then the central unit has to detect this failure in at most 300 s.

It is formally formalized as $(\Box p) \rightarrow (\Box q)$ with $p = (\forall i : \mathit{Sensor} \bullet \neg i.\mathsf{Call})$ which represents no call messages, and $q = (\mathit{Switcher}.\mathsf{DETECTION} \rightarrow \mathit{Switcher}.\mathsf{timer} \leq 300 * \mathsf{Second})$ which

represents that the detection is achieved in at most 300 seconds.

### Transformation process

| edges \ Support notions | does not potentially support ¬Call | does not support ¬Call | does not support specification □(¬Call) |
|---|---|---|---|
| $e_1$ := ⟨first_try, RX[CALL][channelLZ]?, *true*, *locker:=true, gSensorID:=id*, Call⟩ | ✓ | ✓ | ✓ |
| $e_2$ := ⟨second_try, RX[CALL][channelALARM]?, *true*, *locker:=true, gSensorID:=id*, Call⟩ | ✓ | ✓ | ✓ |
| $e_3$ := ⟨third_try, RX[CALL][channelCALL]?, *true*, *locker:=true, gSensorID:=id*, Call⟩ | ✓ | ✓ | ✓ |
| $e_4$ := ⟨Call, cycleReset?, *true*, −−, Call⟩ | ✗ | ✓ | ✓ |

Table 3.2: Non-supporting edges in WFAS model.

In order to verify the well-functioning property that has the assumption-commitment form in the WFAS model depicted in Figures 3.5, the edges that do not support the absence of the call messages will be identified. This identification will be achieved by detecting the edges that do not potentially support the assumption (which is a low-priced task). Only for comparing the effectiveness and the full coverage of the latter detection, we will also detect the edges that do not support the assumption specification (which is the most expensive support detection). The results are presented in Table 3.2 where potentially-support technique behaves almost the same as specification-support except in one edge.

### Verification results

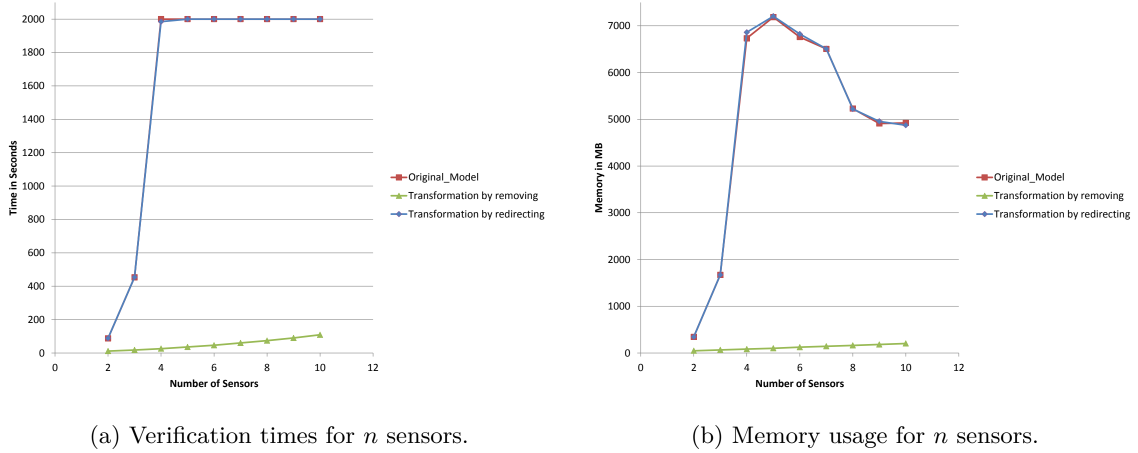
| # | original model, query: $\mathsf{A}\Box(p \to q)$ seconds | MB | kStates | non-supporting removed, query: $\mathsf{A}\Box q$ seconds | MB | kStates | non-supporting redirected, query: $\mathsf{A}\Box(p \to q)$ seconds | MB | kStates |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 87.38 | 344.2 | 4573.06 | 11.47 | 47.1 | 601.87 | 88.17 | 344.2 | 4573.06 |
| 3 | 453.67 | 1671.0 | 21597.64 | 17.81 | 66.1 | 891.14 | 446.79 | 1653.0 | 21582.34 |
| 4 | >2,000.00 | – | – | 26.10 | 83.8 | 1218.34 | 1984.8 | 6859.9 | 89493.30 |
| 5 | >2,000.00 | – | – | 35.98 | 101.3 | 1583.60 | >2,000.00 | – | – |
| 6 | >2,000.00 | – | – | 46.37 | 124.7 | 1986.99 | >2,000.00 | – | – |
| 7 | >2,000.00 | – | – | 60.13 | 144.1 | 2428.59 | >2,000.00 | – | – |
| 8 | >2,000.00 | – | – | 74.07 | 163.0 | 2908.50 | >2,000.00 | – | – |
| 9 | >2,000.00 | – | – | 90.02 | 182.4 | 3426.79 | >2,000.00 | – | – |
| 10 | >2,000.00 | – | – | 109.18 | 203.2 | 3983.55 | >2,000.00 | – | – |

Table 3.3: Figures for verifying well functioning property in WFAS model[3]. Detecting potentially non-supporting edges needs about 0.58 s and 6632 KB.

Table 3.3 summarize the verification results (time and memory) while verifying the WFAS model in Uppaal. It comprises four groups of columns. The first one represents the number of sensors which ranges[4] from two to ten. The second group shows the result of verifying the benchmarks when using Uppaal in the original model, thereby stating the verification time in seconds, memory usage in megabytes, and the number of explored states. The

[3] All results: Linux x64, 16 Quad-Core Opteron 8378, 132 GB, Uppaal 4.1.13. All recent Uppaal versions after 4.1.13 have an **index** bug while instantiating the sensor model. This bug was reported to the Uppaal developers.

[4] In the original model specification as in [AWD+14], it is allowed to have maximally 126 sensors in the wireless alarm model.




(a) Verification times for $n$ sensors.

(b) Memory usage for $n$ sensors.

Figure 3.7: Results of verifying well-functioning property in WFAS model.

third group has the same structure, yet reports results for using Uppaal after removing the edges that do not support the assumption. Finally, the fourth group has the same structure, yet reports results for using Uppaal after redirecting the edges that do not support the assumption.

All results are summarized in Figures 3.7a and 3.7b. The first observation is that verifying the model after removing non-supporting edges scores the best results in terms of time and memory usage with a big difference with other verification techniques. In the original model as shown in Table 3.3, Uppaal does not succeed to verify a system with 4 sensors in 2,000 seconds, a system with 3 sensors takes about 8 min. to be successfully verified. However, while redirecting edges that do not support the absence of call messages, Uppaal succeeds to verify the WFAS model with 5 sensors in 1984 seconds. Finally, while removing edges that do not support the absence of call messages, we can verify all listed benchmarks in a record time as well as verify the same model while the number of sensors is 39 in 2000 seconds. In addition to that, both transformation functions (redirecting and removing) achieve better result than the classical verification of original model without any preprocessing.

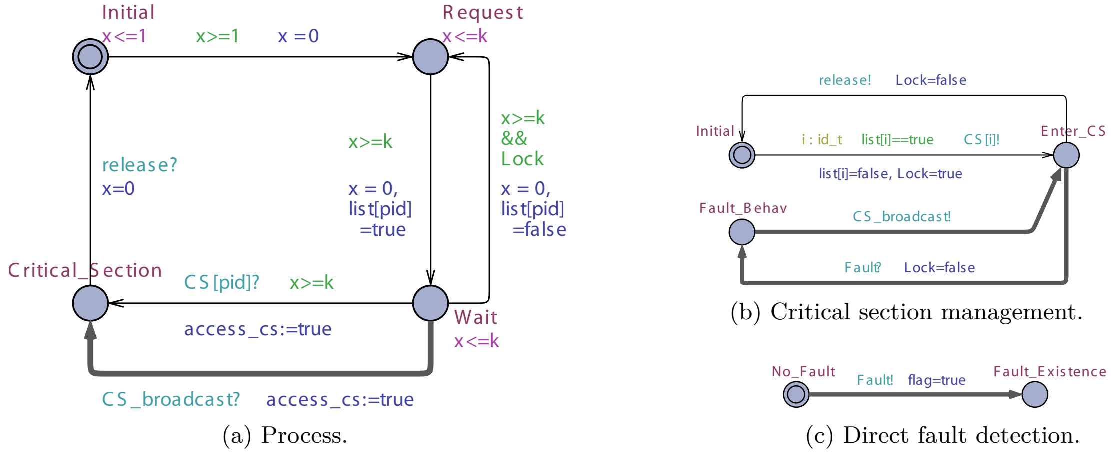
### 3.8.2 Fischer's mutual exclusion protocol

In this section, we explain another case study, which is the Fischer's protocol, a time-controlled mutual exclusion protocol. It is discussed by many others in the literature [Lam87, BDL04, LSW95]. Briefly, Fischer's protocol is a timed protocol where the concurrent processes check for both a delay and their turn to enter the critical section using a shared variable [BDL04]. We take this example from the Uppaal repository of case studies, and extend it by allowing the possibility of failure occurrence which violates the mutual exclusion condition.

### Problem statement

Consider that we have $n \in \mathbb{N}^+$ processes, the main property of this protocol is to assure that any time at most one process can access the critical section. The desired property which is under investigation is that *whenever there is not any error in the model, any time at most one process can access the critical section.* This property is called a *mutual exclusion* property. We formalise this property as a contract with the form $P \rightarrow Q$.

### Description of the model




(a) Process.

(b) Critical section management.

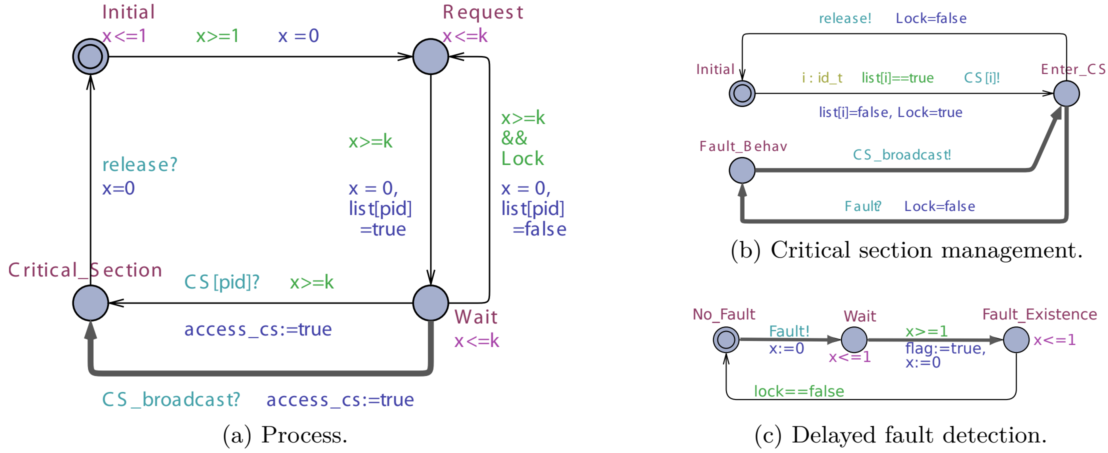
(c) Direct fault detection.

Figure 3.8: Uppaal model of the Fischer's protocol with **direct-fault** detection.

Figure 3.8 presents the Uppaal model of extended Fischer's protocol where a fault may occur in the critical section management as in Figure 3.8b where the latter is expected to monitor the accessibility of the critical section.

If no fault occurs, processes (cf. Figure 3.8a) pose a request to enter their critical section by the shared variable list. The critical section manager (cf. Figure 3.8b) grants access to the critical section by a synchronisation on CS and expects notification of leaving the critical section on channel release. Our extensions are indicated by thick edges: if a fault occurs, processes may enter the critical section bypassing the manager and thereby violate the mutual exclusion property. For simplicity, we merged the environment model which triggers faults and fault detection in Figure 3.8c. Location Fault_Existence models the display of a fault occurrence.

Note that Figure 3.8 is a special case, because fault detection is immediate (no delay between occurrence and detection) and persistent. For real-world systems, fault detection often needs time so there may be small durations of time where the system cannot guarantee proper operation, but where the fault is not yet displayed. An Uppaal model of delayed fault detection for Fischer's protocol is shown in Figure 3.9. This case – under certain sense – simulates the real life situation where such a delay is expected between fault detection and its display due to the nature of hardware communication. Of course, the Uppaal model which is depicted in Figure 3.9 has more transitions and larger state space compared with Figure 3.8.




(a) Process.

(b) Critical section management.

(c) Delayed fault detection.

Figure 3.9: Uppaal model of the Fischer's protocol with **delayed-fault** detection.

### Transformation process

In order to verify the mutual exclusion property that has the assumption-commitment form in the Fischer's protocol depicted in Figures 3.8 and 3.9, the same procedure described in the previous case study is applied. Namely, we will identify the edges that do not support the absence of the faults. This identification will be achieved by detecting the edges that do not potentially support the assumption (which is a low-priced task). Only for comparing the effectiveness and the full coverage of the latter detection, we will also detect the edges that do not support the assumption specification (which is the most expensive support detection).

| Support notions / edges | does not potentially support ¬Fault_Existence | does not support ¬Fault_Existence | does not support specification □(¬Fault_Existence) |
|---|---|---|---|
| $e_1 :=$ ⟨Enter_CS, Fault?, *true*, *lock:=false*, Fault_Behav⟩ | ✗ | ✓ | ✓ |
| $e_2 :=$ ⟨No_Fault, Fault!, *true*, *flag:=true*, Fault_Existence⟩ | ✓ | ✓ | ✓ |

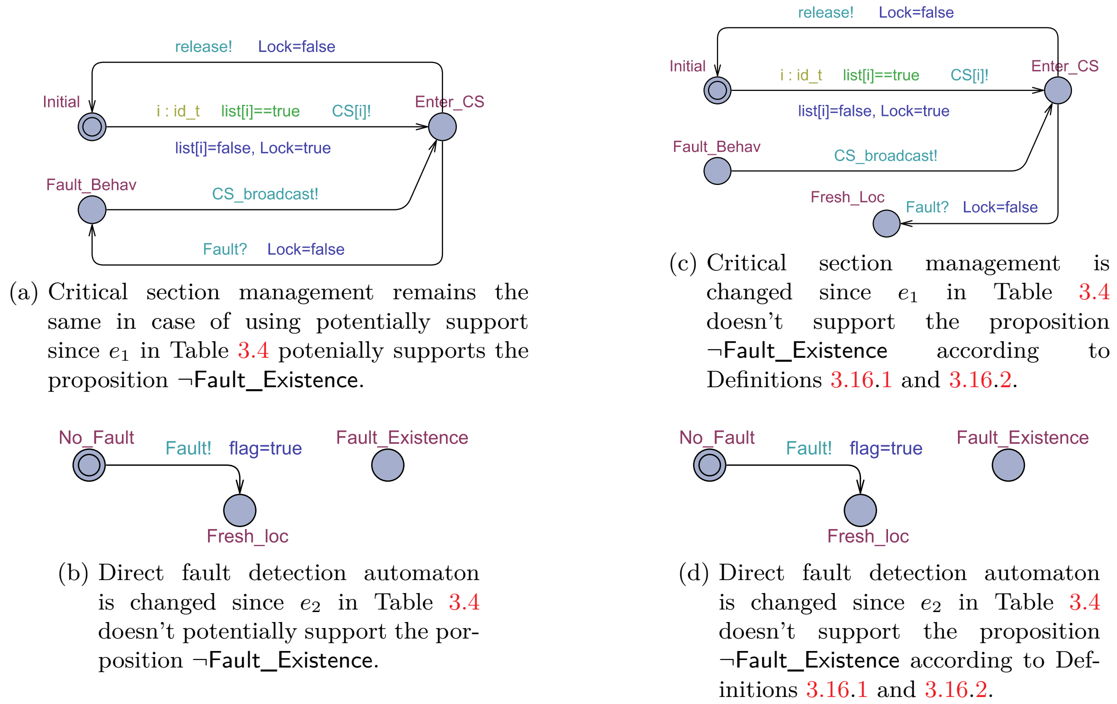
Table 3.4: Non-supporting edges in Fischer's protocol with direct fault detection.

Table 3.4 shows the edges that do not support the assumption i.e., ¬Fault_Existence in the Fischer's protocol with direct fault detection depicted in Figure 3.8. It is observed that detecting edges that do not support the proposition ¬Fault_Existence will give us the real exact number of non-supporting edges in this model. Moreover, using the weakest definition of support; i.e, *potentially support* lacks the precise in detecting edge $e_1$ as non-supporting one, but still feasible as seen in the verification results.

| Support notions / edges | does not potentially support ¬Fault_Existence | does not support ¬Fault_Existence | does not support specification □(¬Fault_Existence) |
|---|---|---|---|
| $e_1 :=$ ⟨Enter_CS, Fault?, *true*, *lock:=false*, Fault_Behav⟩ | ✗ | ✓ | ✓ |
| $e_2 :=$ ⟨No_Fault, Fault!, *true*, *x:=0*, Wait⟩ | ✓ | ✓ | ✓ |
| $e_3 :=$ ⟨Wait, −, *true*, *flag:=true, x:=0*, Fault_Existence⟩ | ✗ | ✓ | ✓ |
| $e_4 :=$ ⟨Fault_Existence, −, *lock == false*, −, No_Fault⟩ | ✗ | ✓ | ✓ |

Table 3.5: Non-supporting edges in Fischer's protocol for the model with delayed fault detection in Figure 3.9.

Now, in order to apply transformations either by removing or redirecting together with



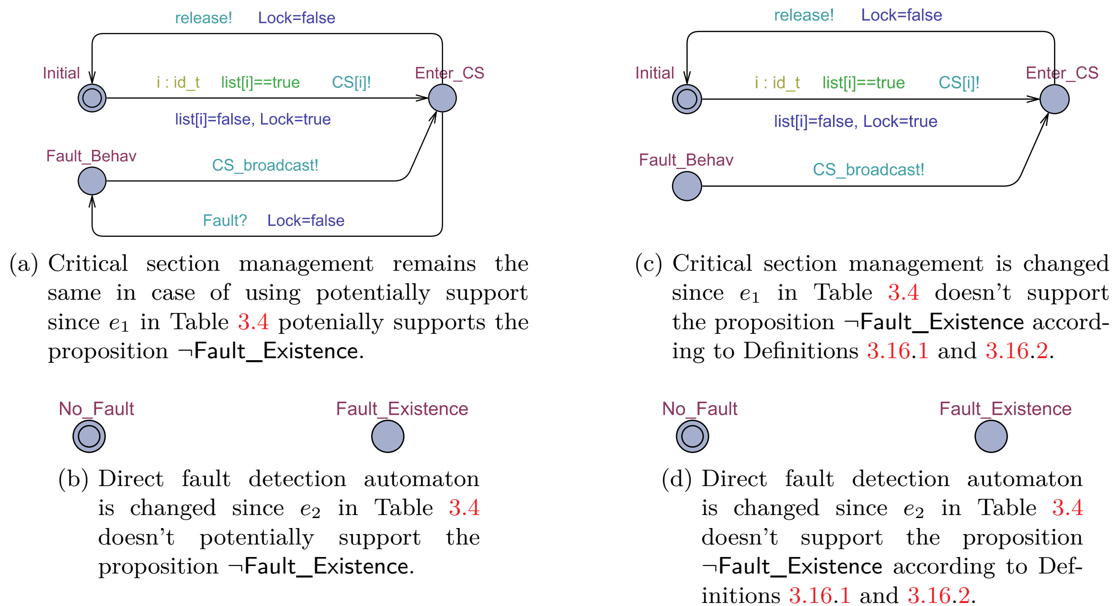

(a) Critical section management remains the same in case of using potentially support since $e_1$ in Table 3.4 potenially supports the proposition ¬Fault_Existence.

(b) Direct fault detection automaton is changed since $e_2$ in Table 3.4 doesn't potentially support the porposition ¬Fault_Existence.

(c) Critical section management is changed since $e_1$ in Table 3.4 doesn't support the proposition ¬Fault_Existence according to Definitions 3.16.1 and 3.16.2.

(d) Direct fault detection automaton is changed since $e_2$ in Table 3.4 doesn't support the proposition ¬Fault_Existence according to Definitions 3.16.1 and 3.16.2.

Figure 3.10: Uppaal model of Fischer's protocol **after applying redirecting** transformation function for the model with direct fault detection in Figure 3.8.



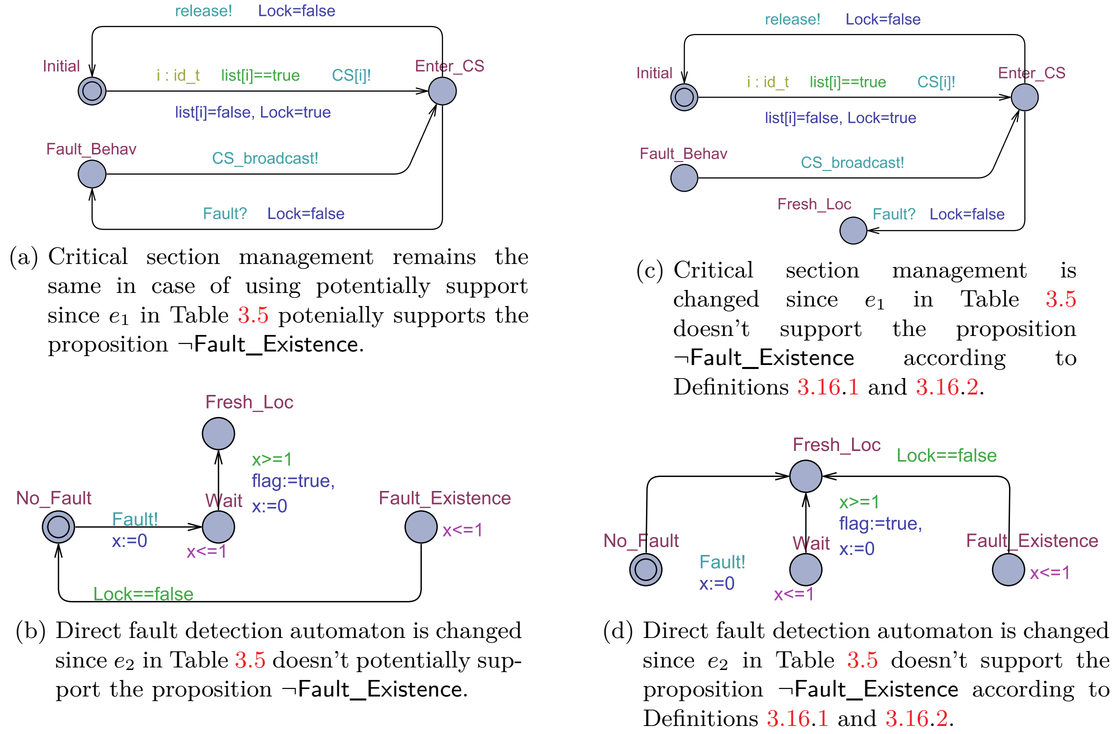

(a) Critical section management remains the same in case of using potentially support since $e_1$ in Table 3.4 potenially supports the proposition ¬Fault_Existence.

(b) Direct fault detection automaton is changed since $e_2$ in Table 3.4 doesn't potentially support the proposition ¬Fault_Existence.

(c) Critical section management is changed since $e_1$ in Table 3.4 doesn't support the proposition ¬Fault_Existence according to Definitions 3.16.1 and 3.16.2.

(d) Direct fault detection automaton is changed since $e_2$ in Table 3.4 doesn't support the proposition ¬Fault_Existence according to Definitions 3.16.1 and 3.16.2.

Figure 3.11: Uppaal model of Fischer's protocol **after applying removing** transformation function for the model with direct fault detection in Figure 3.8.



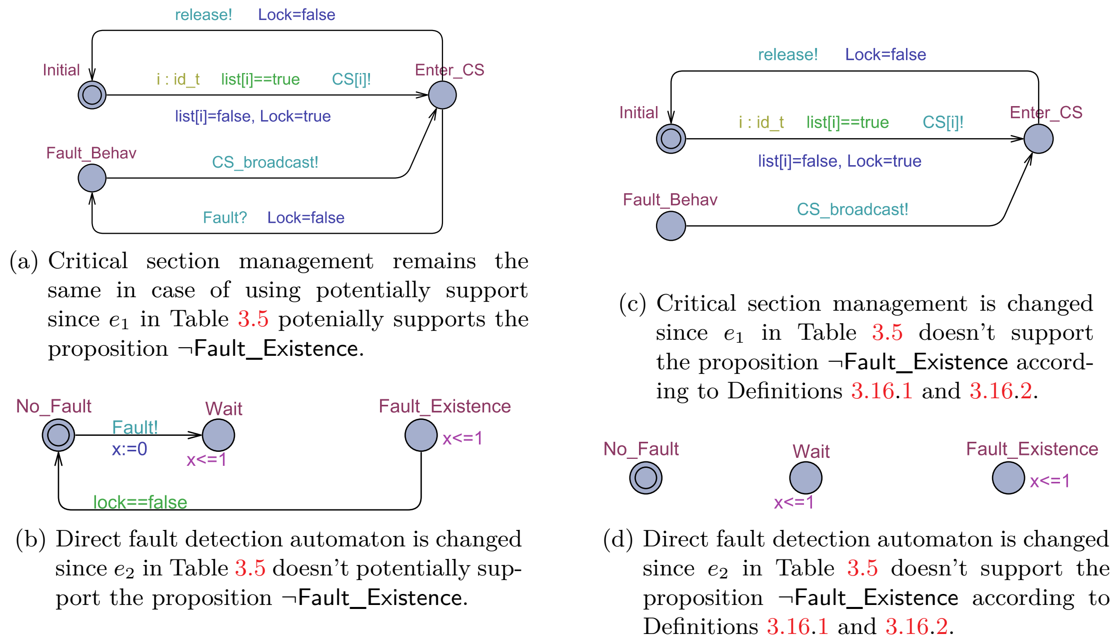

(a) Critical section management remains the same in case of using potentially support since $e_1$ in Table 3.5 potenially supports the proposition ¬Fault_Existence.

(b) Direct fault detection automaton is changed since $e_2$ in Table 3.5 doesn't potentially support the proposition ¬Fault_Existence.

(c) Critical section management is changed since $e_1$ in Table 3.5 doesn't support the proposition ¬Fault_Existence according to Definitions 3.16.1 and 3.16.2.

(d) Direct fault detection automaton is changed since $e_2$ in Table 3.5 doesn't support the proposition ¬Fault_Existence according to Definitions 3.16.1 and 3.16.2.

Figure 3.12: Uppaal model of Fischer's protocol **after applying redirecting** transformation function for the model with delayed fault detection in Figure 3.9.




(a) Critical section management remains the same in case of using potentially support since $e_1$ in Table 3.5 potenially supports the proposition ¬Fault_Existence.

(b) Direct fault detection automaton is changed since $e_2$ in Table 3.5 doesn't potentially support the proposition ¬Fault_Existence.

(c) Critical section management is changed since $e_1$ in Table 3.5 doesn't support the proposition ¬Fault_Existence according to Definitions 3.16.1 and 3.16.2.

(d) Direct fault detection automaton is changed since $e_2$ in Table 3.5 doesn't support the proposition ¬Fault_Existence according to Definitions 3.16.1 and 3.16.2.

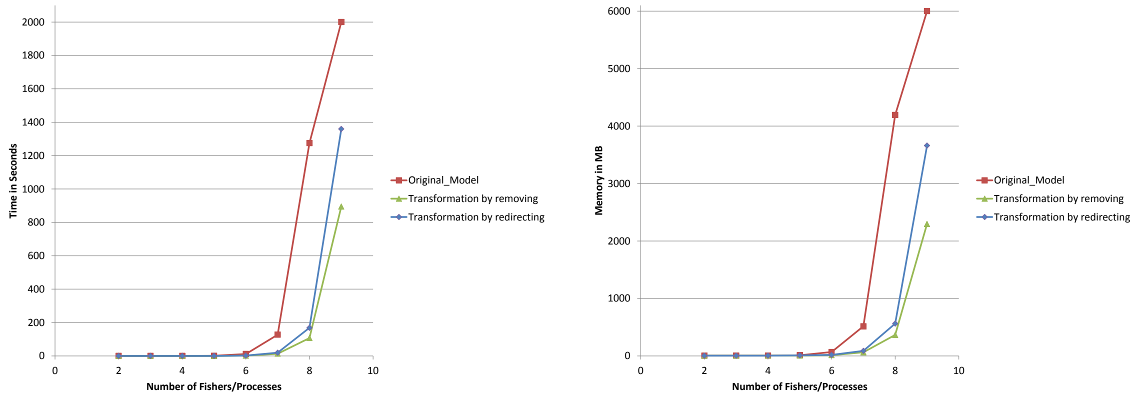
Figure 3.13: Uppaal model of Fischer's protocol **after applying removing** transformation function for the model with delayed fault detection in Figure 3.9.

considering all definitions of supporting, we will get the following results depicted in Figure 3.10 and Figure 3.11 respectively. Figure 3.10 represents the transformation results after redirecting the edges that do not potentially support the proposition ¬Fault_Existence as in Figures 3.10a and 3.10b. On the other hand, as in Table 3.4, supporting the proposition and the specification have the same behaviour, we combine its resultant transformed automata after redirecting in Figures 3.10c and 3.10d. We draw the attention of the reader that for simplicity issue, Figure 3.10 does not consider the process automaton, since it is not affected in any transformation functions. Figure 3.11 has the same description of Figure 3.10, however after applying removing transformation.

Table 3.5 shows the edges that do not support the proposition ¬Fault_Existence in the Fischer's protocol with delayed fault detection, depicted in Figure 3.9. It is observed that detecting edges that do not support the proposition ¬Fault_Existence will give us the real exact number of non-supporting edges in this model. Moreover, using the weakest definition of support; i.e, *potentially support* lacks the precise in detecting edges $e_2, e_3$ and $e_4$ as non-supporting ones[5], but still feasible as seen in the verification results.

Figure 3.12 has the same description as Figure 3.10, however while using delayed fault detection. Also, Figure 3.13 has the same description as Figure 3.11, however while using delayed fault detection.

### Verification results




(a) Verification times for $n$ processes.

(b) Memory usage for $n$ processes.

Figure 3.14: Results of verifying mutual exclusion in Fisher's protocol with direct detection.

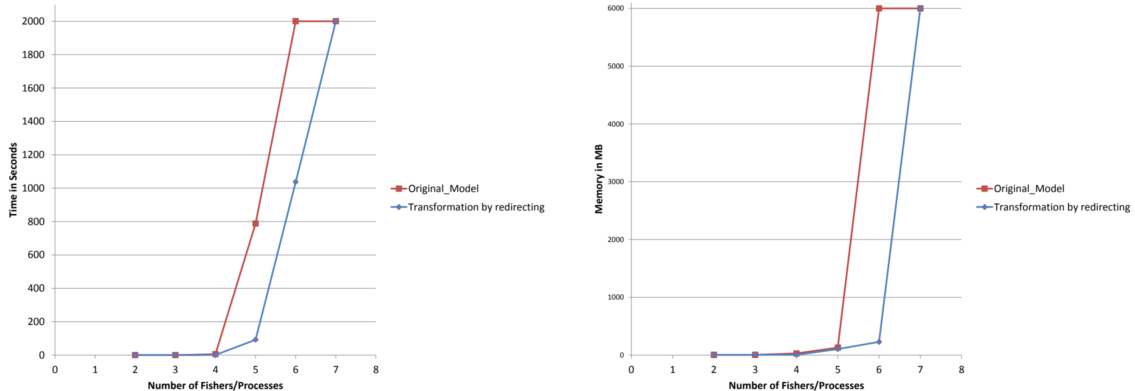
Table 3.6[6] shows results from attempts to verify mutual exclusion given no faults occur, formally, $(\Box p) \rightarrow (\Box q)$ with $p = \neg\mathsf{Fault_Existence}$ and $q = (\forall i \neq j : \mathit{Process} \bullet \neg(i.\mathsf{Crit_Sec} \wedge j.\mathsf{Crit_Sec}))$. Note that it is sufficient to check the Uppaal query $\mathsf{A}\Box(p \rightarrow q)$ for the considered model due to the immediate detection. Table 3.6 comprises four groups of columns. The first one represents the number of concurrent processes which ranges from

[5]Although edges $e_4$ is not detected, but it is disconnected after transformation thus unreachable.
[6]All results: Linux x64, 16 Quad-Core Opteron 8378, 132 GB, Uppaal 4.1.18

| # | original model query: $\mathsf{A}\Box(p \to q)$ | | | non-supporting removed query: $\mathsf{A}\Box q$ | | | non-supporting redirected query: $\mathsf{A}\Box(p \to q)$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | **seconds** | **MB** | **kStates** | **seconds** | **MB** | **kStates** | **seconds** | **MB** | **kStates** |
| 2 | 0.02 | 4.7 | 0.14 | 0.01 | 4.6 | 0.06 | 0.02 | 4.7 | 0.07 |
| 3 | 0.04 | 4.9 | 4.06 | 0.02 | 4.8 | 1.57 | 0.02 | 4.8 | 1.95 |
| 4 | 0.14 | 5.1 | 9.90 | 0.08 | 4.9 | 3.40 | 0.10 | 5.0 | 3.80 |
| 5 | 1.02 | 12.2 | 82.90 | 0.22 | 6.3 | 20.70 | 0.23 | 9.3 | 25.70 |
| 6 | 11.46 | 67.9 | 683.90 | 1.65 | 13.7 | 120.00 | 2.18 | 17.7 | 140.00 |
| 7 | 127.33 | 516.0 | 5,610.70 | 13.18 | 62.8 | 735.80 | 19.40 | 88.2 | 933.10 |
| 8 | 1,274.64 | 4,193.8 | 47,630.30 | 107.18 | 365.3 | 5,142.10 | 168.37 | 562.7 | 6,158.60 |
| 9 | >2,000.00 | – | – | 894.83 | 2,297.0 | 35,614.60 | 1359.34 | 3,659.0 | 40,310.80 |

Table 3.6: Figures for verifying mutual exclusion. The latter property was satisfied in all verified models. Detecting potentially non-supporting edges needs about 0.17 s and 5856 KB.

two to nine. The second group shows the result of verifying the benchmarks when using Uppaal in the original model, thereby stating the verification time in seconds, memory usage in megabytes, and the number of explored states. The third group has the same structure, yet reports results for using Uppaal after removing the edges that do not support the assumption. The fourth group has the same structure, yet reports results for using Uppaal after redirecting the edges that do not support the assumption. All results are summarized in Figures 3.14a and 3.14b. The first observation is that verifying the model after removing non-supporting edges scores the best results in terms of time and memory usage. In the original model as shown in Table 3.6, Uppaal does not succeed to verify a system with 9 processes in 2,000 seconds, a system with 8 processes takes about 20 min. to be successfully verified. In addition to that, both transformation functions (redirecting and removing) achieve better result than the classical verification of original model without any preprocessing.




(a) Verification times for $n$ processes.

(b) Memory usage for $n$ processes.

Figure 3.15: Results of verifying mutual exclusion in Fisher's protocol with delayed detection.

An Uppaal model of delayed fault detection for Fischer's protocol is shown in Figure 3.9.

| # | original model query: $\neg q \leadsto \neg p$ seconds | MB | kStates | non-supporting redirected query: $\neg q \leadsto \neg p$ seconds | MB | kStates |
|---|---|---|---|---|---|---|
| 2 | 0.10 | 5.2 | 0.4 | 0.04 | 5.2 | 0.2 |
| 3 | 0.140 | 5.3 | 12.8 | 0.070 | 5.3 | 7.1 |
| 4 | 5.90 | 29.7 | 109.6 | 0.87 | 5.5 | 37.0 |
| 5 | 788.20 | 130.1 | 2216.9 | 92.25 | 104.1 | 1250.0 |
| 6 | >2,000.00 | – | – | 1037.24 | 228.7 | 3853.4 |
| 7 | >2,000.00 | – | – | >2,000.00 | – | – |

Table 3.7: Fischer's protocol with delayed fault detection. Redirecting edges technique is applied here only, as removing edges cannot be applied since the premise of over-approximating-$P$-rule of Theorem 3.1.2 is broken. Detecting potentially non-supporting edges needs about 0.17 s and 5916 KB.

Here, applying removing edges transformation function as in Figure 3.13 and then verify the commitment in the resultant model does not work, since the premise of Theorem 3.1.2 does not hold after transformation. Instead, one needs to check the commitment "globally mutual exclusion" under the assumption "globally no fault". Still, one can effectively exclude fault scenarios from the verification procedure by *redirecting* the right edge to a fresh sink location Fresh_Loc (cf. Figure 3.12). According to our approach proposed above, we proceed as follows: the edges synchronizing on channel Fault in Figures 3.8c and 3.8b do not support the atomic proposition ¬Fault_Existence. As the automaton resulting from removing these edges satisfies the mutual exclusion property, As a conclusion, the original model satisfies the assumption-commitment property by using Theorem 3.1.2. The verification results are stated in Table 3.6. Figures 3.15a and 3.15b summarize these results. For example, in case of having 7 processes, Uppaal exceeds the limit of 2000$s$ min without giving final verdict. However, after using redirecting edges transformation function, the same problem can be verified in less than 12 min. If the fault detection is delayed as described in Subsubsection 3.8.2 in Figure 3.9, then the automata model obtained from edge removal does not satisfy the mutual exclusion property (the commitment). So no beneficial results would be concluded. Therefore, the check whether the resulting automata model after redirecting satisfies the assumption-commitment property (as in Line 4 in Algorithm 1) comes on scene.

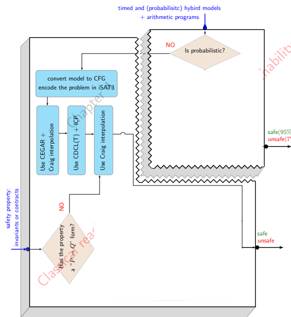
Note that Tables 3.6 and 3.7 only report verification time. For the case study, identifying potentially non-supporting edges [FW16] by using an SMT-solver; i.e. Z3 [dMB08] takes less than one second and the time needed for the subsequent simple source-to-source transformation is negligible.

**Remark 3.4: LTL vs. TCTL**

Interestingly, using non-supporting edges enables us to reduce the linear temporal logic (LTL) property $(\Box p) \rightarrow (\Box q)$, which is not directly supported by the timed computation tree logic (TCTL) fragment of Uppaal, to the leads-to query $\neg q \leadsto f$ where $f$ is a fresh observer for non-supporting edges.

# 4

# Dead Code Detection




Failure should be our teacher, not our undertaker. Failure is delay, not defeat. It is a temporary detour, not a dead end. Failure is something we can avoid only by saying nothing, doing nothing, and being nothing.

*(Denis Waitley)*

## Contents


**4.1 Problem statement** . . . . . . . . . . **59**

4.1.1 Motivation . . . . . . . . . . 59

4.1.2 Related work . . . . . . . . . . 61

4.1.3 Example . . . . . . . . . . 67

**4.2 Preliminaries** . . . . . . . . . . **69**

4.2.1 Control flow automaton . . . . . . . . . . 69

4.2.2 Craig interpolation: theory and application . . . . . . . . . . 72

4.2.3 Interpolation-based model checking (ITP) . . . . . . . . . . 73

4.2.4 Counterexample guided abstraction refinement: theory and application . . . . . . . . . . 76

**4.3 The iSAT3 model checker** . . . . . . . . . . **77**

4.3.1 Syntax and semantics . . . . . . . . . . 77

4.3.2 iSAT3 architecture and engines . . . . . . . . . . 78

4.3.3 iSAT3 interpolants . . . . . . . . . . 80

4.3.4 BMC problems in iSAT3 . . . . . . . . . . 94

4.3.5 CFA problems in iSAT3 . . . . . . . . . . 96

**4.4 Interpolation-based CEGAR technique** . . . . . . . . . . **97**

4.4.1 Interpolation-based refinement procedure in iSAT3: algorithm . . 97

4.4.2 Example . . . . . . . . . . 106

4.4.3 Case studies . . . . . . . . . . 107

**4.5 Handling floating points dominated C-programs – experiments in industrial-scale** . . . . . . . . . . **111**

4.5.1 Floating point arithmetic due to IEEE 754 . . . . . . . . . . 111

4.5.2 Floating points in iSAT3 . . . . . . . . . . 112

4.5.3 Floating point arithmetic in iSAT3 with CEGAR . . . . . . . . . . 112

4.5.4 Industrial case studies . . . . . . . . . . 114

4.5.5 Converting SMI code to iSAT3-CFG input language . . . . . . . 115
4.5.6 BTC-ES benchmarks . . . . . . . . . . . . . . . . . . . . . . . . . 117

## 4.1 Problem statement

### 4.1.1 Motivation

The wide-spread use of embedded control programs involving linear, polynomial, and transcendental arithmetic provokes a quest for corresponding verification methods. A crucial technique here is the automatic verification of reachability properties in such programs, as many problems can be reduced to it and as it in particular provides a method for detecting unreachable code fragments, a.k.a. *dead code*, in such programs. The latter is an industrial requirement, as various pertinent standards for embedded system development either demand adequate handling of dead code during testing or even bar it altogether, like DO-178C [EH10], DO-278A [Che09], or ISO/IEC PDTR 24772 [TRn09]. Most existing program verification techniques are either confined to verify reachability in programs featuring just linear arithmetic or they address non-linear hybrid systems if they feature just a small control skeleton. They are thus inapt of verifying industrial-scale arithmetic programs with their non-trivial control flow and non-linear arithmetic. Such programs, on the one hand may admit non-linear arithmetic involving transcendental functions, like *sin* and *exp*, and on the other hand they exhibit high impact of data on control flow. The latter is particularly true for well-known schemes like Simulink-Stateflow auto-generated programs [KEB+14], where control flow is – to quite some extent – coded by manipulating data items.

One has to undoubtedly observe that traditional satisfiability-modulo-theory (SMT) solving as one of the workhorses of automatic program analysis, would not help in the latter situation, since it mostly addresses decidable fragments of arithmetic only, like linear or polynomial[1]; e.g., C Bounded Model Checking (CBMC) which is one of the leading approaches to automatic software analysis [KT14], can handle C programs admitting only linear and polynomial constraints, but not transcendental functions, while richer fragments are covered by only few tools [GKC13, SKB13].

Similar problems apply to the predominant program analysis techniques, like abstract interpretation [CC77, CC92, BCC+11, Cou12, CC14], which lack exactness when going beyond linear arithmetic, since the geometry of sets of numbers representable by its usual lattices and the graphs of the monotonic functions over these can only provide coarse overapproximations.

To overcome the aforementioned problems, this chapter aims at combining several techniques to verify code reachability in floating-point dominated programs admitting non-linear constraints. Namely, we integrate counterexample guided abstraction refinement (CEGAR) [CGJ+00], Craig interpolation (CI) [Cra57], conflict-driven clause learning

[1] We avoid the ambiguous and thus misleading term "non-linear" here, whose use for denoting polynomial arithmetic unfortunately has become popular.

CDCL(T) [ZM02] in its lattice-based variant [DHK13], and interval constraint propagation (ICP) [Ben96] in one framework.

First, CEGAR is used as a frontend of our framework due to its use of abstractions to efficiently handle large programs and circumvent the state-space explosion problem. In our approach, it starts form *a conservative initial abstract model* which simulates only the control flow of the original program. This abstract model is *progressively refined* based on model checking the abstraction and analysing the counterexamples generated by the model checker. In each refinement step, either *a spurious counterexample* will be excluded by enriching the abstract model with sufficient predicates obtained from the stepwise interpolants as proposed in lazy abstraction technique [HJMS02], or a real counterexample is found. This refinement procedure is feasible despite non-polynomial arithmetic as the floating points numbers are from a (large but) finite domain. CEGAR has been applied successfully in the context of programs [BHJM07], real time systems [NOK10] and hybrid systems [DT13]. What sets our method apart is that it applies CI to learn concise reasons for a counterexample being spurious despite its rich, non-polynomial arithmetic domain. While CI-based CEGAR is standard in the domain of software model-checking, current approaches tend to address linear arithmetic only.

Craig interpolation thus is our workaholic for abstraction refinement in the CEGAR loop. CI is a technique from logics that for two contradicting formulas yields an *interpolant formula* only containing shared variables, being implied by the first formula, and still contradicting the second formula. By using the concept of CI with SAT-based as well as SMT-based bounded model checking [McM03], we are able to prove that certain target states or rather code fragments are unreachable as CI permits the computation of invariants in arithmetic programs, even for non-polynomial constraints systems as Kupferschmidt et al. demonstrated in the iSAT tool [KB11], which we build upon. In conjunction with CEGAR, in software verification with lazy abstraction technique [HJMS02], stepwise interpolants are used to extract meaningful predicates from infeasible error paths, where the resulting interpolants are used to refine the abstract model. This is needed in order to ensure that the interpolants at the different locations achieve the goal of providing a precision that eliminates the infeasible error path from further explorations.

In order to tackle the problem of non-polynomiality in our programs, interval analysis for floating-point computations is augmented with constraint narrowing for floating-point intervals (including NaNs), yielding interval constraint propagation (ICP) as a sound deduction approach. The interval consistency notions of a set of non-linear arithmetic constraints are used to relax the general non-polynomial problem to an SMT problem over the linear order of the reals. As a result of using ICP as an arithmetic reasoner in the core of our framework, each computed interpolant obtained from unsatisfiability is a formula involving Boolean and simple arithmetic bounds only, i.e., just order-theoretic statements.[2]

The same applies for conflict clauses thus constructed in the conflict-driven clause learning (CDCL) procedures or our iSAT-based solver, as in iSAT's [FH07, SKB13] historic implementation of abstract CDCL [DHK13].

Together, this provides an unprecedented integration of technologies. All previous ap-

[2]iSAT3 rarely returns a non-linear constraint as a subexpression of an interpolant, in case this non-linear constraint occurs as a shared expression having two contradicting bounds.

proaches individually cannot provide a feasible solution to our problem due to either scalability issues or too confined fragments of arithmetic addressed. To the authors' best of knowledge, this is the first attempt to combine CEGAR, CI, ACDCL, and ICP in one platform in order to attack the state space explosion problem in the model checking of arithmetic software, find invariants of the verified non-polynomial program, solve very large complex Boolean formulae, and capture the arithmetic reasoning over non-polynomial constraints respectively. This combination facilitates precisely checking reachability in arithmetic programs which may involve transcendental functions, like *sin*, *cos* and *exp*.

The closest work to ours is [KB11] that verifies reachability in the presence of non-linear constraints by using CDCL(T), ICP and CI, yet it fails to provide summaries for loops. Moreover, it does not scale enough to cover the full branching structure of complex programs in just few sweeps, in particular if the verified programs contain nested loops. Many previous works employed CEGAR with Craig interpolation, however confined to linear programs. Our approach is implemented within the iSAT3 solver [SKB13], where this new combination gives impressive results while verifying (as of yet moderately small) non-linear programs. Concisely reflecting IEEE 754 [IEE85] floating-point arithmetics will be discussed in the last section of this chapter, where our approach is integrated with [SNM+16b] to preform tests on BTC-ES AG benchmarks.

### 4.1.2 Related work

Looking up the literature covering related work, one can find a large number of previous works discussing the same problem from different perspectives, addressing slightly different fragments. For example, our work can be compared with other approaches that are exhaustively used in static program analysis to detect and eliminate dead code. Moreover, it can be equiponderated with verification techniques that address the same problem, but are confined to other arithmetics theories. Furthermore, our work draws an analogy to some previous works using mostly the same technologies, however outperforms them as shown in this chapter.

**Abstract interpretation (static code analyses).** Many static program analyses were introduced to help compilers and programmers to optimize code, by analysing and verifying its properties. For example, *reaching definitions* is a data-flow analysis which statically determines which definitions may reach a given control point in the code. Because of its simplicity, it is known as the canonical example of a data-flow analysis in textbooks. *Very busy expressions analysis* is a variant of available expression analysis. An expression is very busy at a point if it is guaranteed that the expression will be computed at some time in the future. Thus starting at the point in question, the expression must be reached before its value changes. *Available expression analysis* is a backward flow analysis, since it propagates information about future evaluations backward to "earlier" points in the computation. *Live variable analysis* is a classic data-flow analysis performed by compilers to calculate for each program point the variables that may be potentially read before their next write, that is, the variables that are live at the exit from each program point [NNH05].

Normally, compilers detect dead code in a program in a sense that *they find only the*

*segment of codes that have no effect in the program.* E.g., dead variables or the code that can never be executed. The aforementioned mechanism in compilers tries to detect as much as possible of dead code, but at the same time without exhausting the resources and without much time consumption. Therefore they are roughly incomplete in that sense and fail to flag substantial amounts of dead code in programs. Consequently, it means that compilers lack the full coverage and the precision of detecting dead code in programs which some standards for embedded system development demand as aforestated. Moreover, detecting dead code by using compilers techniques can not be properly achieved if we speak about complex programs that combine data and control flows in one code without a clear distinction (*even if they are distinguished, we need to adapt two different algorithms to analyse the code* [THR82, CF87, ZTM11, SGL+11]). In addition to that, detecting dead code becomes more difficult if we have *non-linear system constraints* or some code blocks that depend on complex computations involving numerous floating-point and discrete data entities.

Abstract interpretation (AI) is a theory of semantics approximation which is used for the construction of semantics-based program analysis algorithms (sometimes called "data-flow analysis"), the comparison of formal semantics (e.g., construction of a denotational semantics from an operational one), the design of proof methods, etc [CC77]. The origin of AI is in static program analysis [CC76].

In [BCC+11], Bertrane et al. showed how sound semantic static analyses based on abstract interpretation would be used to check properties at various levels of a software design: from high level models to low level binary code. In [Cou12], Cousot addressed how AI provides scaling solutions to achieving assurance in safety critical systems through *verification* by fully automatic, semantically sound and precise static program analysis.

**Counterexample guided abstraction refinement (CEGAR).** In [CGJ+00], Clarke et al. presented an automatic *iterative abstraction-refinement* methodology in which the initial abstract model is generated by an automatic analysis of the control structures in the program to be verified. Abstract models may admit erroneous counterexamples. They devised symbolic techniques which analyse such counterexamples and refine the abstract model correspondingly. The *refinement algorithm keeps the size of the abstract state space small* due to the use of abstraction functions which distinguish many degrees of abstraction for each program variable. In [CFH+03], Clarke et al. presented a procedure to perform refinement operation for abstractions of hybrid systems. Following the previous approach [CFH+03], the refinement procedure constructs a *new abstraction* that eliminates a counterexample generated by the model checker. For hybrid systems, analysis of the counterexample requires the *computation of sets of reachable states* in the continuous state space. They showed how such reachability computations with varying degrees of complexity can be used to refine hybrid system abstractions efficiently. In [BK04], Bjesse et al. presented a method for finding failure traces for safety properties that are out of reach for traditional approaches to counterexample generation. They achieved this by *guiding bounded model checking* (BMC) with information gathered from counterexample guided abstraction refinement. Unlike approaches based on reconstructing abstract counterexamples on the concrete machines, they did not search only for failures of the same length as the current abstract counterexample. But they also described a *combination of several methods for choosing registers* to include in the abstraction. In [MFH+06], Manevich et al.

formalized *CEGAR for general powerset domains*. If a spurious abstract counterexample needs to be removed through abstraction refinement, there are often several choices, such as which program location(s) to refine, which abstract domain(s) to use at different locations, and which abstract value to compute. They introduced *several plausible preference orderings* on abstraction refinements, such as refining as “late” as possible and as “coarse” as possible. Finally, they presented generic algorithms for finding refinements that are optimal w.r.t. different preference orderings. In [SH13], Seipp et al. showed how counterexample guided abstraction refinement can be used to *derive informative heuristics* for optimal classical planning. Additionally, they introduced an algorithm for building additive abstractions. In [TD13], Tian et al. developed a new approach to refine the spurious counterexamples in CEGAR with the first failure set they found. That procedure was feasible as it eliminates many late bad states that can appear in other counterexamples. Thus they can handle large industrial case studies with impressive time. However, this approach is restricted to linear arithmetic programs.

**Using Binary decision diagrams (BDD).** BDD is a well-defined data structure which is used to represent a Boolean function. It was firstly introduced by Bryant et al. [Bry86] as a compressed representation of sets or relations. Then it was developed further by McMillan. There is an enormous number of previous works that use BDD in analysing, optimizing and verifying tasks.

In [CNQ03] Cabodi et al. combined BDD and SAT-based methods to increase the efficiency of BMC by exploiting affordable BDD-based symbolic approximate reachability analysis to gather information on the state space. Then, they used the collected reachable state sets to guide the search space of a SAT-based BMC. This is doable by feeding the SAT solver with a description that is the combination of the original BMC problem with the extra information obtained from BDD-based symbolic analysis. In [GGW$^+$03] Gupta et al. explored the use of learning from BDDs, where learned clauses generated by BDD-based analysis are added to the SAT solver, to supplement its other learning mechanisms. In addition to that, they introduced several heuristics for guiding this process, aimed at increasing the usefulness of the learned clauses, while reducing the overheads. This approach was effectively scaled to several industrial designs, where BMC performance is improved and the design can be searched up to a greater depth by use of BDD-based learning. In [SB06] Sinz et al. presented a method to convert the construction of binary decision diagrams (BDDs) into extended resolution proofs by conjoining BDDs in SAT solving. This approach enables the usage of BDDs for this purpose instead – or in combination with – other verification and proving methods based on DPLL with clause learning. In [BB06] Bartzis et al. showed how to construct linear-sized BDDs for linear integer arithmetic constraints. They presented basic constructions for atomic equality and inequality constraints and extend them to handle arbitrary linear arithmetic formulas. Also, they presented three alternative ways of handling out-of-bounds transitions, and discuss multiple bounds on integer variables. This approach can be used to improve the behaviour of other BDD-based symbolic model checkers. In [RIS13] Ribeiro et al. proposed a new algorithm that learns a logic program from interpretation transitions. However, the run time of this algorithm is exponential where a memory space is optimized by using an efficient data structure based on zero-suppressed binary decision diagrams. In [BS12, BS14] Beyer et al. evaluated the use of a pure BDD representation of integer values in a particular

class of programs: event-condition-action (ECA) programs with limited operations. They configured a program analysis based on BDDs and experimentally compare it to four approaches to verify reachability properties of ECA programs: an explicit-value analysis, a symbolic bounded-loops analysis, a predicate abstraction analysis, and a predicate impact analysis. The results showed clearly that BDDs are efficient for a restricted class of programs, which yields the insight that BDDs could be used selectively for variables that are restricted to certain program operations (according to the variable's domain type), even in general software model checking.

**Using Conflict-driven clause learning (CDCL).** CDCL is an algorithm for solving the Boolean satisfiability problem (SAT). The internal workings of CDCL-SAT solvers were inspired by DPLL solvers. The main differences between CDCL and DPLL are that CDCL's back jumping is non-chronological and a conflict-clause learning mechanism. The first proposal of CDCL was by Bayardo and Schrag [JS97]. Then it was completely presented by Marques-Silva and Sakallah [SS99]. Since 2001, many solvers and researches turn to use/improve CDCL in industrial designs [BCKS08], quantified Boolean Satisfiability solvers [ZM02], test pattern generation [WRP+02], numeric bounds analysis [DHKT12] and in first version of iSAT; i.e. HySAT II [FH07].

One of closest approaches to my work is [BDG+13], introduced by Brain et al. In this work, a proof generation and interpolation techniques for the family of abstract CDCL solvers are presented, in which all reasoning is performed within an abstract domain. They build upon these techniques to implement the interpolation-based verifiers for programs with floating-point variables. However, the latter work is confined to linear and polynomial arithmetic theories.

**Using Craig interpolation (CI).** CI is a mathematical concept that enables us to generalize a reason of unsatisfiability between two conflicting formula. It was proposed by William Craig in 1957 [Cra57]. Then, it was systematically computed by Huang [Hua95] and Púdlak [Pud97].

In [McM03], another way of computing Craig interpolation was introduced by McMillan. Moreover, in [McM05], McMillan succeeded to extend bounded model checking problems to unbounded ones by using Craig interpolation. He presented an interpolating prover for a quantifier-free theory that includes linear inequalities and equality with uninterpreted function symbols. In [BKRW10], Brillout et al. presented an interpolating sequent calculus that can compute interpolants for the combination of uninterpreted functions and linear integer arithmetic. The interpolants computed using their method might contain quantifier since they do not use divisibility predicates. Furthermore their method limits the generation of Gomory cuts [Gom58, Gom10] (find integer solutions to mixed integer linear programming problems) in the integer solver to prevent some mixed cuts. In [YM05], Yorsh et al. showed how to combine interpolants generated by an SMT solver based on Nelson-Open combination [NO79]. They defined the concept of equality-interpolating theories. These are theories that can provide a shared term $t$ for a mixed literal $a = b$ that is derivable from an interpolation problem. The annoying mixed interface equality $a = b$ is rewritten into the conjunction $a = t \wedge t = b$. They show that both, the theory of uninterpreted functions and the theory of linear rational arithmetic are equality-interpolating.

In [CGS08], Cimatti et al. presented a method to compute interpolants for linear rational arithmetic and difference logic. In [GKT09], a way to compute interpolants for a generalization of equality-interpolating theories is introduced by Goel et al. where a class of almost-colorable proofs and an algorithm to generate interpolants from such proofs are presented. Furthermore they describe a restricted DPLL system to generate almost-colorable proofs. This system does not restrict the search if convex theories are used. Their procedure is incomplete for non-convex theories like linear arithmetic over integers, since it prohibits the generation of mixed branches and cuts. In [KB11], Kupferschmid et al. succeeded to compute the interpolants for integer, rational and real theories even for non-linear complex constraints by using interval constraint propagation, however the approach itself is incomplete as non-linear problems (including sine, cosine, and exponential functions) are undecidable. In [CHN13], Christ et al. computed Craig interpolants in the presence of mixed literals. Contrary to most existing approaches, this scheme neither limits the inferences done by the SMT solver, nor does it transform the proof tree before extracting interpolants. This approach works for the combination of uninterpreted functions and linear arithmetic, but is extendible to other theories. In [AM13b], Albarghouthi et al. described *a compositional approach* to Craig interpolation based on the heuristic that simpler proofs of special cases are more likely to generalize. They presented a method for finding such simple facts in the *theory of linear rational arithmetic*. This makes it possible to use interpolation to discover inductive invariants for *numerical programs that are challenging* for existing techniques. However this approach deals only with linear arithmetic formulae with DNF structure. In [GZ16], Gao et al. implemented the equivalent techniques for computing interpolants for non-linear constraint as Kupferschmid [KB11]. It was shown that this approach implements the framework of $\delta$-complete decision procedures in dreal.[3]

**Combining Craig interpolation with abstract interpretation.** In [HJMM04], Henzinger et al. succeeded to use Craig interpolants as summaries of reasons why a path found on an abstract model has no concrete counterpart and exploit these summaries for refining the abstract state-space in a CEGAR loop as well. In [BL12], Beyer et al. integrated *abstraction* and *interpolation-based refinement* into an *explicit-value analysis*, i.e., a program analysis that tracks explicit values for a specified set of variables (the precision). The algorithm uses *an abstract reachability graph* as central data structure and a path sensitive dynamic approach for precision adjustment.

**Combining BDD and CI with abstraction.** In [EKS08], Esparza et al. investigated CEGAR in the context of sequential (possibly recursive) programs whose statements are given as binary decision diagrams (BDDs). Additionally, they succeeded to treat multiple counterexamples in one refinement cycle by using Craig interpolation, where the latter is computed efficiently. In [VG09], Vizel at al. presented a new *SAT-based approach* such that it can perform *full verification*. The approach combined BMC with *interpolation sequence* in order to imitate BDD-based symbolic model checking (SMC). In [AGC12, ALGC12], Albarghouthi et al. presented what is so-called "Ufo", an algorithm that unifies OD (overapproximation) and UD (underapproximation) driven approaches in order to leverage both of their advantages. Ufo is parametrized by the degree to which over- and under-approximations drive the analysis. This framework is used for verifying

[3]You can access this tool under the following link: http://dreal.github.io/

(and finding bugs in) C-programs. It allows definition of different abstract post operators, refinement strategies and exploration strategies. They have built three instantiations of the framework: a *predicate-abstraction based* version, an *interpolation based* version, and a combined version which uses a novel and powerful combination of interpolation based and predicate-abstraction based algorithms. In [McM10], McMillan presented an application of the IMPACT principle to testing and similarly proposed computing predicate abstractions in order to speed up the convergence of the algorithm. In [HHP10], Heizmann et al. succeeded to extend the IMPACT approach by supporting recursive programs. This method averts the costly construction of the abstract transformer by constructing a nested *word automaton* from an inductive sequence of "nested interpolants" (i.e., interpolants for a nested word which represents an infeasible error trace). In [EHP12], Ermis et al. introduced a software model checking approach that uses the concept of *path insensitive interpolation* to compute loop invariants. In contrast to other approaches, path insensitive interpolation *summarizes several paths through a program location instead of one.* As a consequence, it takes the abstraction refinement considerably less effort to obtain an adequate interpolant. In [AGC12], Albarghouthi et al. proposed an interpolation-based software verification algorithm for checking safety properties of (possibly *recursive*) sequential programs. The latter algorithm, called Whale, produces interprocedural proofs of safety by exploiting interpolation for guessing function summaries by generalizing underapproximations (i.e., finite traces) of functions. In [BW12], Beyer et al. compared two of the most important algorithms that are *based on CEGAR and Craig interpolation techniques*: *lazy predicate abstraction* (as in BLAST [BHJM07]) and *lazy abstraction with interpolants* (as in IMPACT or Wolverine [WKM12]). But they unified the algorithms formally (by expressing both in the CPA framework [BK09]) as well as in practice (by implementing them in the same tool). This allows to flexibly experiment with new configurations and gain new insights about their performance characteristics. They showed that the essential contribution of the IMPACT algorithm is the *reduction of the number of refinements*, and compare this to another approach for reducing refinement effort: adjustable block encoding (ABE).

**Combining trace abstraction with Craig interpolation.** In [Seg07], Segelken introduced unprecedented idea of applying CEGAR loop based on learning reasons of spurious counterexamples in an $\omega$-automaton, where this technique exhibits relatively few refinement iterations to prove or disprove safety properties in quite large models. In [HHP09], Heizmann et al., inspired by [Seg07], presented a new counterexample guided abstraction refinement scheme. The scheme *refines an overapproximation of the set of possible traces.* Each refinement step introduces a finite automaton that recognizes *a set of infeasible traces.* A central idea is to use interpolants in order to automatically construct such an automaton. A database of interpolant automata has an interesting potential for reuse of theorem proving work.

**Combining CEGAR with SAT-based techniques.** In [CGS04], Clarke et al. described a technique for model checking in the counterexample guided abstraction refinement framework. The abstraction phase "hides" the logic of various variables, hence considering them as inputs. This type of abstraction may lead to spurious counterexamples, i.e. *traces that cannot be simulated on the original (concrete) machine.* They checked whether a coun-

terexample is real or spurious with a SAT checker. Then, they used a combination of 0-1 *Integer Linear Programming* (ILP) and machine learning techniques for refining the abstraction based on the counterexample. The process is repeated until either a real counterexample is found or the property is verified.

**Combining abstraction with CEGAR in concurrent programs.** In [DKK+12], Donaldson et al. tried to fill the gap of using the combination of *predicate abstraction* and *counterexample guided abstraction refinement in shared-variable concurrent software*. They attributed this gap to the lack of abstraction strategies that permit a scalable analysis of the resulting multi-threaded Boolean programs and have developed a symmetry-aware CEGAR technique.

**Combining abstraction, BDD, SMT-based solvers.** In [CFG+10], Cimatti et al. addressed the problem of computing the exact abstraction of a program with respect to a given set of predicates, a key computation step in counterexample guided abstraction refinement. To do so a recently proposed approach that integrates *BDD-based quantification techniques* with *SMT-based constraint solving* is employed to compute the abstraction.

**Combining abstraction with static analysis.** In [WKG07], Wang et al. proposed a *hybrid abstraction method* that allows both visible variables and predicates to take advantages of their relative strengths. They used *refinement based on weakest preconditions* to add new predicates, and under certain conditions trade in the predicates for visible variables in the abstract model. They presented heuristics for improving the overall performance, based *on static analysis* to identify useful candidates for *visible* variables, and use of *lazy constraints* to find more meaningful/effective unsatisfiable cores for refinement.

Using Craig interpolation or counterexample-guided abstraction refinement or conflict driven clause learning or interval constraint propagation alone cannot address a feasible solution to our problem as will be shown in Subsections 4.4.3 and 4.5.6. Moreover, most previous related works combined several techniques together, but they are inapt to highlight substantial parts of dead code in programs admitting non-linear constraints. Since they are either confined to linear arithmetic theories or handling non-linear ones, however with a small control skeleton.

Abstract interpretation and static code analysers lack the exactness, thus a sufficient coverage of detecting dead code is not guaranteed.

### 4.1.3 Example

We will begin this chapter with a simple motivational example, which is a small one but still interesting, since according to author's best knowledge, all existing approaches cannot handle it directly without preprocessing by linearisation, or without simplifying the problem by just asserting bounded safety using BMC.

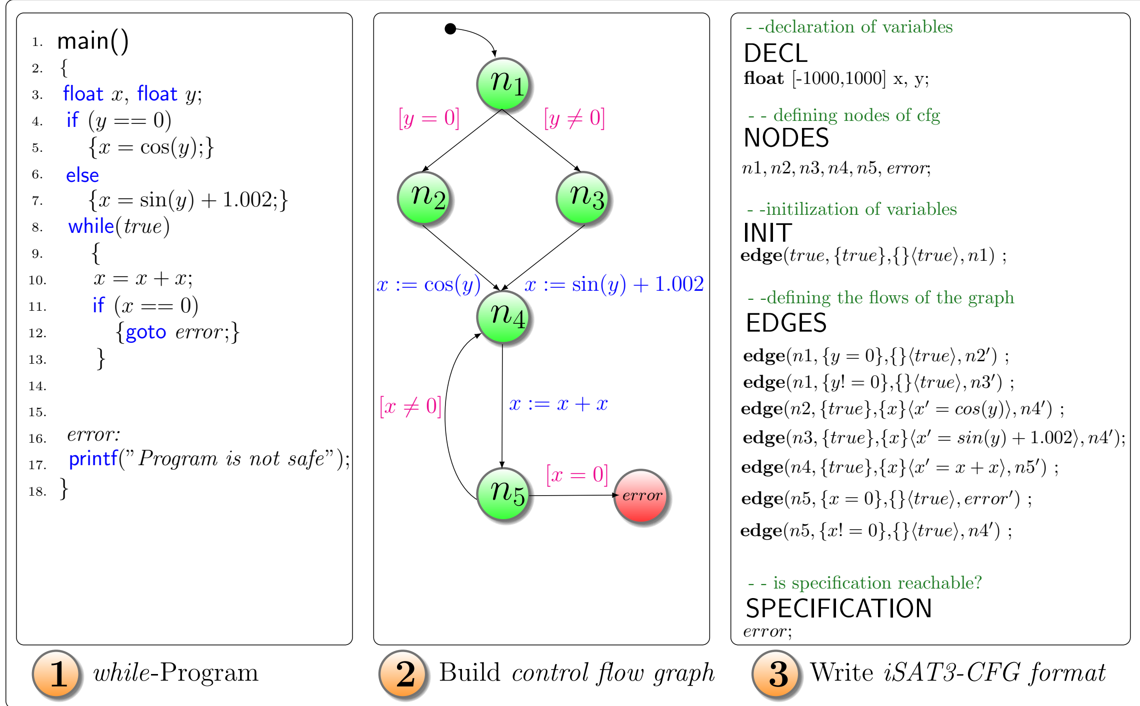



Figure 4.1: Left: An arithmetic program, middle: corresponding control flow graph, right: encoding in iSAT3-CFG format.

**Example 4.1: Example of a non-polynomial program**

This example admits a small control flow automaton (CFA) of an arithmetic program, however involving a rich fragment of arithmetic; namely transcendental functions. The formal semantics of the control flow automaton will be introduced in the next section where the operation semantics of CFA induces a formal operational semantics of general automata (cf. Section 3.2 and Definition 3.3). Consider the arithmetic C-program depicted in Figure 4.1. Its corresponding CFA has six nodes and seven transitions where node error represents the bad state that must be ascribed to the unreachable set of states in order warrant safety. If one checks all possible evolutions of the program model, one can (spontaneously) say that error is unreachable since:

- the first kind of CFA paths has a prefix condition which restricts $y$ not to be zero and on the same time restricts $x$ to be $sin(y) + 1.002$. At the end, $x$ in any case will be larger than or equal 0.002. Considering the $x$-valuations at $n_4$, one can informally represent the reachable set as $r_1 := \{(x = i) \mid i \in [0.002, 2.002]\} = \{(x = 0.002), ..., (x = 2.002)\}$
- the second kind of paths has a prefix condition which restricts $y$ to be zero and on the same time restricts $x$ to be $cos(y)$. At the end, $x$ in any case will equal 1. Considering the reachable set of $x$-valuations at $n_4$, it represents the following set $r_2 := \{(x = 1)\}$.

Thus, one can say that the reachable set of $x$-valuations at $n_4$ is the join between $r_1$

and $r_2$ which is equal to $r_1$. Consequently, the infinite loop from node $n_4$ to $n_5$ will be executed forever, and node error will not be reachable. Although it seems as a straightforward explanation, many techniques like ICP, CDCL(T) with CI or with k-induction are inapt to prove safety of this example. That is, the solver continues splitting the intervals of $x$ and $y$ and cannot find a suitable safe inductive invariant. However, by using CEGAR, we address a feasible safety proof of that example as will be discussed in more details in Subsection 4.4.2.

## 4.2 Preliminaries

We use and suitably adapt several existing concepts to fit our purpose. A *control flow automaton* (CFA) is a cyclic graph representation of all paths that might be traversed during program execution.

### 4.2.1 Control flow automaton

In our context, we attach code effect to edges rather than nodes of the CFA. i.e., each edge comes with a set of constraints and assignments pertaining to execution of the edge. Formally, constraints and assignments are defined as follows:

**Definition 4.1: Assignments and constraints**

Let $V$ be a set of integer and real variables, with typical element $v$, $B$ be a set of Boolean variables, with typical element $b$, and $C$ be a set of constants over rationals, with typical element $c$.

- The set $\Psi(V, B)$ of assignments over integer, real, and Boolean variables with typical element $\psi$ is defined by the following syntax:

$$\begin{aligned}
\psi &::= v := aterm \mid b := bterm \\
aterm &::= uaop\ v \mid v\ baop\ v \mid v\ baop\ c \mid c \mid v \\
comp &::= aterm\ lop\ c \mid aterm\ lop\ v \\
bterm &::= ubop\ b \mid b\ bbop\ b \mid b \mid comp \\
uaop &::= - \mid sin \mid cos \mid exp \mid abs \mid ... \\
baop &::= + \mid - \mid \cdot \mid ... \\
ubop &::= \neg \\
bbop &::= \wedge \mid \vee \mid \oplus \mid ... \\
lop &::= < \mid \leq \mid = \mid > \mid \geq
\end{aligned}$$

  By $\vec{\psi}$ we denote a finite list of assignments on integer, real, and Boolean variables, $\vec{\psi} = \langle \psi_1, ..., \psi_n \rangle$ where $n \in \mathbb{N}_{\geq 0}$. We use $\Psi(V, B)^*$ to denote the set of lists of assignments and $\langle\ \rangle$ to denote the empty list of assignments.
- The set $\Phi(V, B)$ of constraints over integer, real, and Boolean variables with

typical element $\phi$ is defined by the following syntax:

$$\begin{aligned} \phi &::= atom \mid ubop\ atom \mid atom\ bbop\ atom \\ atom &::= theory_atom \mid bool \\ theory_atom &::= comp \mid simple_bound \\ simple_bound &::= v\ lop\ c \\ bool &::= b \mid ubop\ b \mid b\ bbop\ b \end{aligned}$$

where *uaop*, *baop*, *ubop*, *bbop*, *lop* and *comp* are defined above.

We assume that there is a well-defined valuation that assigns to each assigned variable a value from its associated domain. Also, we assume that there is a satisfaction relation between the valuations of variables and guards. In addition to that, the modification of a valuation after executing a finite list of assignment will consider the new reset-value, or propagate a previous value of the variable. It is formally introduced as follows:

**Definition 4.2: Valuation**

A *valuation* $\nu$ of integer and real variables $V$ and Boolean variables $B$ is a mapping

$$\nu : V \cup B \rightarrow \mathcal{D}_V \cup \mathcal{D}_B$$

assigning to each variable $v \in V$ a value in $\mathcal{D}(v)$ and assigning to each variable $b \in B$ a value in $\mathcal{D}(b)$. We assume that there is a satisfaction relation $\models \subseteq (V \rightarrow \mathcal{D}(V)) \times \Phi(V)$ and in case of non-Boolean constraints we write

$$\nu \models \phi \text{ iff } \nu|_V \models \phi$$

Analogously, we assume that there is a satisfaction relation $\models \subseteq (B \rightarrow \mathcal{D}(B)) \times \Phi(B)$ and in case of Boolean constraints we write

$$\nu \models \phi \text{ iff } \nu|_B \models \phi$$

The modification of a valuation $\nu$ under a finite list of assignment $\vec{\psi} = \langle \psi_1, ..., \psi_n \rangle$ denoted by $\nu[\vec{\psi}] = \nu[\psi_1]...\nu[\psi_n]$ and defined as follows:

$$\nu[v := aterm](v') = \begin{cases} \nu[aterm] & if \quad v' := v \\ \nu(v') & otherwise, \end{cases}$$

$$\nu[b := bterm](b') = \begin{cases} \nu[bterm] & if \quad b' := b \\ \nu(b') & otherwise, \end{cases}$$

**Definition 4.3: Control Flow Automaton (CFA): syntax**

A control flow automaton

$$\gamma = (N, E^{CFA}, i)$$

where
- $N$ is a finite set of nodes,
- $E^{CFA} \subseteq N \times \Phi(V, B) \times \Psi(V, B) \times N$ is a finite set of directed edges, where $\Phi(V, B)$ is a set of constraints and $\Psi(V, B)$ is a set of assignments. Each edge $(n, \phi, \vec{\psi}, n') \in E^{CFA}$ has a source node $n$, a constraint $\phi$, a list $\vec{\psi}$ of assignments and a destination node $n'$,
- $i \in N$ is an initial node which has no incoming edges.

is an automaton $\mathcal{A}_{CFA}$ as in Definition 3.1 where:
- $Loc := N$,
- $Act := \Phi(V, B) \times \Psi(V, B)$,
- $E := \{(\ell, (\phi, \vec{\psi}), \ell') \mid (n, \phi, \vec{\psi}, n') \in E^{CFA}\}$
- $L_{ini} := i$.

For simplicity reason, we will omit the "*CFA*" superscript from the edges of the control flow automaton. Additionally, we use "*control flow automaton*" and "*control flow graph*" interchangeably.

CFA's operational semantics interprets the edge constraints and assignments and induces the operational semantics of automaton introduced in Definition 3.3:

**Definition 4.4: Operational semantics of control flow automaton induces an automaton semantics**

The operational semantics $\mathcal{T}$ assigns to each control flow automaton $\gamma = (N, E, i)$ a labelled transition system

$$\mathcal{T}(\gamma) = (Conf(\gamma), \{\xrightarrow{e} \mid e \in E\}, C_{ini})$$

where:
- $Conf(\gamma) = \{\langle n, \nu \rangle \mid n \in N \wedge \nu : V \cup B \to \mathcal{D}_V \cup \mathcal{D}_B\}$ is the set of configurations of $\gamma$,
- $\xrightarrow{e} \subseteq Conf(\gamma) \times Conf(\gamma)$ are transition relations where $\langle n, \nu \rangle \xrightarrow{e} \langle n', \nu' \rangle$ occurs iff there is an edge $e = (n, \phi, \vec{\psi}, n')$, $\nu \models \phi$ and $\nu' = \nu[\vec{\psi}]$,
- and $C_{ini} = \{\langle i, \nu_{init} \rangle\} \cap Conf(\gamma)$ is the set of initial configurations of $\gamma$.

In contrast to common usage of CFA, we do not define an *exit* node of CFA as we allow infinite paths. We assume that a control flow automaton represents possibly finite or infinite paths (traces) of a program. Formally, it is introduced as follows:

**Definition 4.5: Path**

*A path* $\sigma$ of control flow automaton $\gamma = (N, E, i)$ under operational semantics $\mathcal{T}(\gamma)$ is an infinite or finite sequence $\langle n_0, \nu_0 \rangle \xrightarrow{e_1} \langle node_1, \nu_1 \rangle \xrightarrow{e_2} \langle n_2, \nu_2 \rangle \ldots$ of consecutive transitions in the transition system $\mathcal{T}(\gamma)$, which furthermore has to be anchored in the sense of starting in an initial state $\langle i, \nu_0 \rangle \in C_{ini}$. We denote by $\Sigma(\gamma)$ the set of paths of $\gamma$ and by $\downarrow \sigma$ the set $\{\langle i, \nu_0 \rangle, \langle n_1, \nu_1 \rangle, ...\}$ of configurations visited along a path $\sigma$.

We often convert programs to CFAs and then we verify some properties in CFAs. One of the well-known and important properties to be verified is *reachability*. Reachability is a graph property useful in verification (cf. Chapter 2). In verification task, given certain nodes (even one node) of a control flow automaton are representing bad states/behaviours of a program, if we verify that these parts are *disconnected from the other parts w.r.t. program semantics*; i.e. unreachable, then we prove the safety of the model. That is, verification of safety property is carried out in terms of verifying the reachability of nodes in a control flow automaton.

Also, if we consider that a CFA models a program behaviour, then proving or disproving reachability can be seen as detecting *dead code of a program*. Whenever some code segments are not reachable in the program, they are called *dead* codes. Detecting/eliminating [Kno96, DP96, CGK98, CGK97] dead code (or unreachable states/parts of a model) is considered to be very important and challenging nowadays. The reachability property is formally defined as follows.

**Definition 4.6: Reachability Property:syntax**

The set $\Theta(N, \Phi(V, B))$ of reachability properties (RPs) over nodes and constraints in a control flow graph $\gamma = (N, E, i)$, is given by the following syntax:

$$\theta ::= n \mid \phi$$

where $n \in N$ and $\phi \in \Phi(V, B)$.

Any reachability property of "$\theta = \phi$" form can be represented by the other form namely, $\theta = n$ via introducing a new node $n_{new}$ where each node in $N \setminus n_{new}$ has an outgoing edge with the guard $\phi$ to $n_{new}$. However, we aimed at making Definition 4.6 complete and self-contained by introducing both forms.

**Definition 4.7: Satisfaction relation for reachability property**

Given a control flow graph $\gamma = (N, E, i)$ and an RP property $\theta$, we say that $\sigma \in \Sigma(\gamma, \theta)$ satisfies $\theta$ and write $\sigma \models \theta$ iff

- $\sigma$ traverses a configuration $\langle n, \nu \rangle$ that satisfies $\theta$, i.e., $\sigma \models \theta$ iff $\exists \nu : \langle n, \nu \rangle \in\downarrow \sigma$ and $\langle n, \nu \rangle \models \theta$.

We say that $\gamma$ satisfies a reachability property $\theta$ iff

- some path $\sigma \in \Sigma(\gamma)$ satisfies $\theta$.

By $\Sigma(\gamma, \theta)$, we denote the set of all paths of $\gamma$ that satisfy $\theta$.

### 4.2.2 Craig interpolation: theory and application

Craig's interpolation [Cra57] establishes a relationship between different logical formulae in the same theory. Roughly stated: if a formula $A$ implies a formula $\neg B$ then one can find a third formula $\mathcal{I}$, called an *interpolant*, such that every symbol in $\mathcal{I}$ occurs both in $A$ and $B$, $A$ implies $\mathcal{I}$, and $\mathcal{I}$ implies $\neg B$. That is, an interpolant $\mathcal{I}$ is a generalization of unsatisfiability in case $A \wedge B \models$ *false*. The theorem was first proved for first-order logic by William Craig in 1957. Variants of the theorem hold for other logics, such as propositional

logic. A stronger form of Craig's theorem for first-order logic was proved by Roger Lyndon in 1959; the overall result is sometimes called the *Craig-Lyndon theorem* [Obe68]. Formally Craig interpolation is defined as follows.

**Definition 4.8: Craig Interpolation**

Given two propositional logic formulae $A$ and $B$ in a logics $\mathcal{L}$ such that $\models_{\mathcal{L}} A \rightarrow \neg B$, a *Craig interpolant for* $(A, B)$ is a quantifier-free $\mathcal{L}$-formula $\mathcal{I}$ such that $\models_{\mathcal{L}} A \rightarrow \mathcal{I}$, $\models_{\mathcal{L}} \mathcal{I} \rightarrow \neg B$, and the set of the variables of $\mathcal{I}$ are subset of set of the shared (and thus free) variables between $A$ and $B$, i.e., $Var(\mathcal{I}) \subseteq Var(A) \cap Var(B)$.

**Example 4.2: Example of Craig interpolation**

Assume that we have propositional formulae $A = (p \wedge r) \vee (p \wedge \neg q)$ and $B = (\neg p \wedge z)$.
- ✓ $A \rightarrow p$, i.e. $(p \wedge r) \vee (p \wedge \neg q) \rightarrow p$
- ✓ $p \rightarrow \neg B$, i.e. $p \rightarrow \neg(\neg p \wedge z)$
- ✓ $p \in V_{A,B}$.

Thus $p$ is a valid interpolant that justifies the unsatisfaiblity of $A \wedge B$. Later, we will show different mechanisms to compute the interpolants systematically.

### 4.2.3 Interpolation-based model checking (ITP)

In 2003 and subsequently, Craig interpolation was used as a novel technique to verify reachability in unbounded model checking problems [McM03, EKS08, BL12]. The idea of using interpolants in unbounded model checking is as follows:

- bounded model checking and interpolation can be combined to produce an overapproximate image operator that can be used in symbolic model checking.
- a bounded model checking problem consists of the conjunction of initial constraints; i.e., *INIT*, transition assignments and constraints; i.e., *TRANS* and final constraints (target); i.e. *TARGET* where the BMC is as follows:

$$INIT \wedge \bigwedge_{i=0}^{n} TRANS[i] \wedge \bigvee_{i=0}^{n} TARGET[i]$$

These constraints are instantiated for each depth from 0 to $k$, where $k \in \mathbb{N}_{>0}$ as in Figure 4.2.

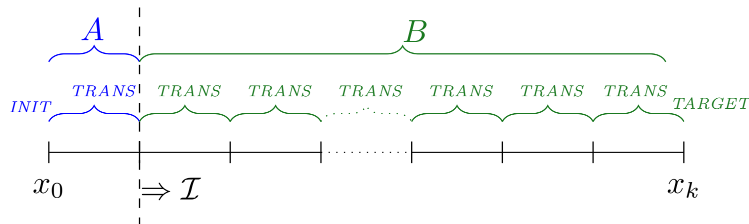



Figure 4.2: Bounded model checking and computing post-image by interpolation.

- if we partition the CNF of the problem such that the initial constraint together with the first instance of the transition are considered to be $A$ formula and the rest of the CNF is considered to be $B$ formula, then the shared variables between $A$ and $B$ will be the instantiated variables at depth 1 in $x_1$.
- by using a SAT solver, if we get that $A \wedge B$ is unsatisfiable, then we can generate an interpolant $\mathcal{I}$ that overapproximates the forward image of initial constraints.
- this procedure can be iterated to compute an overapproximation of the reachable states by splitting the CNF formula to prefix and suffix subformulae as follows:

$$PREFIX_l = INIT(x_0) \wedge \bigwedge_{i=0}^{l-1} TRANS(x_i, x_{i+1})$$

$$SUFFIX_l^k = \bigwedge_{i=l}^{k-1} TRANS(x_i, x_{i+1}) \wedge \bigvee_{i=k-l}^{k} TARGET(i_k)$$

  where $l$ is a parameter which controls the number of overapproximated steps. We continue as before until we end with one of the following situations:

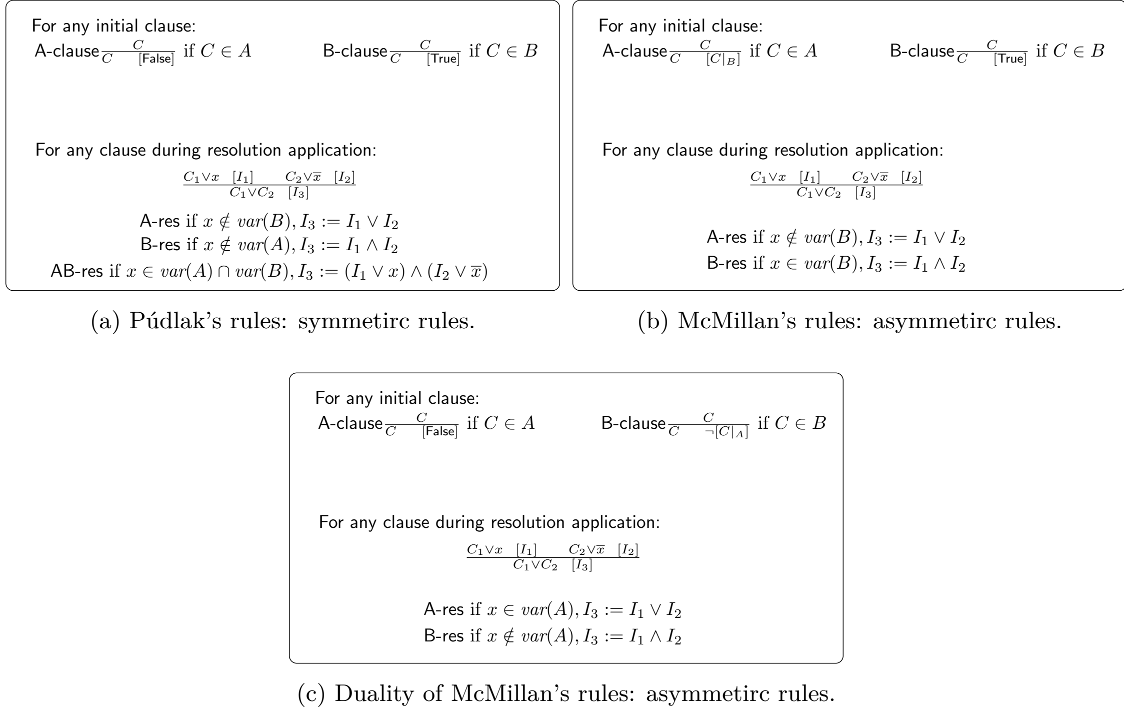
  – the interpolant stabilizes, the computed interpolant implies the initial state, thus, a safe inductive invariant is found. This implication can be achieved after stripping the variable names in the computed interpolants of their step-dependent renaming. In this case we prove that the target is not reachable, because the overapproximation of reachable states did not intersect with the target.

  – if the initial state is not implied by the interpolant, we need to increase the depth of the formula, however by considering the current interpolant as the initial state of the BMC problem. By setting the current interpolant as initial state, the number of added unwindings has been increased by $l$. That means, we may get a counterexample that the target is reachable which is not necessary to be true, since our initial state is an overapproximation. To validate the discovered counterexample we need to unfold the problem till depth $k$ and solve the normal BMC problem as a CNF formula. If the counterexample is a real one, then we know that the target is reachable. Otherwise, we need to discard the previously calculated interpolants, and start a new ITP run at the current unroll (For more details, cf. [McM03, KB11]).

Additionally, Craig interpolation is used also in counterexample-guided abstraction refinements loops, in particular, for adequate predicate extraction in refinement step [HJMM04], which will be generally discussed in that chapter and particularized in Section 4.4.

### Systematic computing approaches

Depending on the logics $\mathcal{L}$ as in Definition 4.8, such a Craig interpolant which provides a reason why $A$ is not satisfiable together with $B$, can be computed by various mechanisms. If $\mathcal{L}$ admits quantifier elimination, then this can in principle be used; various more efficient




(a) Púdlak's rules: symmetirc rules.

(b) McMillan's rules: asymmetirc rules.



(c) Duality of McMillan's rules: asymmetirc rules.

Figure 4.3: Different interpolant computing approaches.

schemes have been devised for propositional logic and SAT-modulo theory by exploiting the connection between resolution and variable elimination [Pud97, EKS08].

The first trial to construct the Craig interpolation from proof refutation for propositional logic was by Huang [Hua95]. Then, he was followed by Pudlak [Pud97], where both of them compute the interpolants in terms of restricted quantifier elimination, however Huang method was introduced for a general calculus. Then in 2003, McMillan [McM03] proposed another way of computing Craig interpolation which in a certain sense computes stronger interpolants in comparison to Púdlak's ones. After that, many works appeared to compute the interpolants for SMT problems e.g., integer, rational, linear real and non-linear arithmetic. Most of these approaches [McM03, CHN13, BKW08, GLS11, GKT09, CGS10, LT08] depend on reducing SMT constraints to Boolean literals in SAT problem and then using extended version of Gaussian elimination for linear inequations; i.e., Fourier-Motzkin elimination [DE73, Kha09]).

Figure 4.3 shows the well-known systematic mechanisms of computing Craig interpolants, however with different strengths. The first one is called a symmetric method or Púdlak's approach as in Figure 4.3a. The other mechanisms are both asymmetric rules methods, however in duality position, McMillan's approach computes stronger interpolants (more closely to $A$-formula) as in Figure 4.3b and its duality computes weaker interpolants (more closely to $B$-formula) as in Figure 4.3c.

One should observe that the resultant interpolants by using any of the previous approaches depends on the resolution tree extracted from the unsatisfiability proof. Thus, one cannot generally guarantee the interpolant structures and strengths. This point will be investi-

gated more in Subsubsection 4.3.3, where slackness of interpolants will be under scrutiny.

### 4.2.4 Counterexample guided abstraction refinement: theory and application

The CEGAR technique was proposed to attack the state space explosion problem; particularly in large models. It was briefly introduced in [Kur94]. However it was completely defined in [CGJ+00, Cla03] by Clarke et al.

The idea of CEGAR depends on abstracting the original control flow automaton model $\gamma$ to an abstract one; i.e., obtaining $\alpha(\gamma)$ by a well-defined abstraction function $\alpha$, such that concrete behaviour model is contained by the abstract one. After abstracting a

**Algorithm 2** Counterexample guided abstraction refinement

*Input:* A concrete model $\gamma$ and a bad state $\theta$
*output:* A Boolean variable *result* denotes that reachability of bad states

```
 1: procedure CEGAR(γ, θ)
 2:     do
 3:         abstract the concrete model by α i.e. α(γ).
 4:         if α(γ) ⊨ θ. then
 5:             extract a counterexample (CEX); i.e. σ from the abstract model α(γ).
 6:             concretize σ in the original model γ.
 7:             if σ is a real CEX then
 8:                 result := unsafe.
 9:                 break;
10:             else
11:                 extract sufficient predicates to exclude σ from α(γ).
12:                 α(γ) := refine(α(γ)).
13:             end if
14:         else
15:             result := safe.
16:         end if
17:     while true
18:     return result;
19: end procedure
```

model, one can verify a safety property $\theta$ (it is almost unanimously that $\theta \in \mathsf{ACTL}^*$) in the abstract one. That is, we want to verify whether the bad states that violate the safety property, are reachable or not. The description of how CEGAR is applied is shown in Algorithm 2, where abstracting, extracting a counterexample and concretizing it are stated from Line 3 to Line 6.

Refinement is depicted in this algorithm from Line 7 to Line 15, where we have two cases. *The first case* happens if the bad states are not reachable in the abstract model, then we guarantee that the original model is safe (Line 15), since the abstract model behaviour is an overapproximation of the concrete model behaviour. *The second case*, if we found a counterexample in the abstract model such that the bad states are reachable, then we need to *validate* (concretize) this counterexample in the original model by a well-defined

concretizing function; i.e., $\kappa$ where the latter collects the path conditions and assignments of the abstract path. If we have found that the current counterexample is *a real one* (e.g., has a corresponding concrete trace in the original model), we know the safety property *is violated* (broken) in the original model (Line 8). However, if we have found that the latter counterexample is not a real counterexample (e.g., *spurious*, *erroneous*, *bogus*), we need to *refine* this counterexample in the abstract model. Refining is achieved by removing the causes of this spurious counterexample in the abstract model such that it will be excluded from further exploration. In general, termination of this procedure is not guaranteed always, since we cannot expect to obtain a sound and complete verification algorithm, as the reachability problem is undecidable in general even if programs are confined to only linear arithmetics [KSU11]. However CEGAR is found to be feasible in practice. For more details about the recent developments of CEGAR confer Section 4.1.2. Later in Section 4.4, we will show in details how CEGAR loop is applied in programs that may admit non-linear constraints where a novel refinement approach is preformed in the iSAT3 model checker.

## 4.3 The iSAT3 model checker

### 4.3.1 Syntax and semantics

We build our CEGAR loop on the iSAT3 solver, which is an SMT solver accepting formulas containing arbitrary Boolean combinations of theory atoms involving linear, polynomial and transcendental functions as follows:

**Definition 4.9: Syntax of iSAT3 SMT-formulae in CNF**

Given $b \in B$, $v \in V$ and $c \in C$, an SMT constraint formula in iSAT3 $\varphi$ (in CNF form) is defined inductively as follows:

$$\begin{aligned}
\varphi &::= \{clause \wedge\}^* clause \\
clause &::= (\{atom \vee\}^* atom) \\
atom &::= comp \mid bound \mid bool \\
comp &::= v \ lop \ c \mid v \ lop \ v \\
bound &::= v \ lop \ term \\
bool &::= b \mid ubop \ b \mid b \ bbop \ b \\
term &::= uaop \ v \mid v \ baop \ v \mid v \ baop \ c \\
lop &::= < \ \mid \ \leq \ \mid \ = \ \mid \ > \ \mid \ \geq \\
uaop &::= - \mid sin \mid cos \mid exp \mid abs \mid ... \\
baop &::= + \mid - \mid \ \cdot \ \mid ... \\
ubop &::= \neg \\
bbop &::= \wedge \mid \ \vee \ \mid \ \oplus \ \mid ...
\end{aligned}$$

*We informally define the underlying semantics of iSAT3.* A constraint formula $\varphi$ is satisfied by a valuation of its variables if and only if all its clauses are satisfied, that is, if

and only if *at least one atom* is satisfied in any clause. An atom is satisfied according to the standard valuation of the Boolean and arithmetic constraints such that the *underlying theory semantics are respected*. A formula $\varphi$ is *satisfiable* if and only if there exists a *satisfying valuation of each variable occurring* in the formula such that at least one atom in each clause of the formula is evaluated to `true`. Otherwise, $\varphi$ is unsatisfiable. We remark that by definition of satisfiability a formula $\varphi$ including or implying the empty clause, denoted by $\bot$ or $\emptyset$, cannot be satisfied at all; i.e., if $\bot \in \varphi$ or $\varphi \to \bot$ then $\varphi$ is unsatisfiable.

In addition to that, iSAT3 assigns to each non-Boolean variable[4] an interval in contrast to classical CDCL(T) or DPLL(T); i.e. $\mu : V \to \mathbb{I}$ where $V$ represents the set of all integer and real variables and $\mathbb{I}$ represents the set of all valid intervals defined over (a finite set of) reals. This kind of assigning agrees with the aforementioned semantics of control flow graph variables valuations, since one can map that valuation of variables introduced in Definition 4.2 to iSAT3 semantics. iSAT3 uses the *hull-consistency* concept (cf. [BMH94, BG06]) in order to safely contract the initial interval of the variables according to the new deduced bounds [Her11]. During solving a problem, iSAT3 tries to detect safe and more precise bounds of the variables until we end with two cases. *First case*, iSAT stops making decisions when every problem variable has reached a current interval width that is less or equal to a given splitting minimal width. In this case, if there exists a satisfiable interpretation that respects the variable assigned intervals and the structure of the formula, then the formula is *satisfiable*. Otherwise, the result will be referred to as a *candidate solution*, since the current $\mu$ may contain a solution. *Second case* the solver finds a conflict that cannot be resolved; e.g., an assigned interval of a variable collapses to *empty*. In this case, the formula is *unsatisfiable*.

### 4.3.2 iSAT3 architecture and engines

In classical SMT solving a given SMT formula is split into a Boolean skeleton and a set of theory atoms. The Boolean skeleton (which represents the truth values of the theory atoms) is processed by a SAT solver in order to search for a satisfying assignment. If such an assignment is found, a separate theory solver is used to check the consistency of the theory atoms under the truth values determined by the SAT solver. In case of an inconsistency the theory solver determines an infeasible sub-set of the theory-atoms which is then encoded into a clause and added to the Boolean skeleton. This scheme is called CDCL(T).

In contrast to CDCL(T), there is no such separation between the SAT and the theory part in the family of iSAT solvers [FHT+07]; instead interval constraint propagation (ICP) [BG96] is tightly integrated into the CDCL framework in order to dynamically build the Boolean abstraction by deriving new facts from theory atoms. Similarly to SAT solvers, which usually operate on a *conjunctive normal form* (CNF), iSAT3 operates on a CNF as well, but a CNF additionally containing the decomposed theory atoms (so-called *primitive constraints*). We apply a definitional translation akin to the Tseitin-transformation [Tse68] in order to rewrite a given formula into a CNF with primitive constraints. The basic idea

[4]Boolean variables are assigned to point intervals either $[0, 0]$ or $[1, 1]$ representing false and true respectively.

of this transformation is to introduce new auxiliary variables for sub-expressions. This technique is well-known for Boolean formulas – we use it also to decompose theory atoms. The formula

$$(b_1 \wedge b_2) \oplus ((v_1 + sin(v_2)) \cdot v_3 < 7)$$

would thus be rewritten to the following equisatisfiable CNF by introducing fresh auxiliary variables $b_3, v_4, v_5, v_6$ and $b_4$:

$$\begin{gathered}(b_1 \vee \neg b_3) \wedge (b_2 \vee \neg b_3) \wedge (\neg b_1 \vee \neg b_2 \vee b_3) \wedge \\ (v_4 = sin(v_2)) \wedge (v_5 = v_1 + v_4) \wedge (v_6 = v_5 \cdot v_3) \wedge \\ (b_3 \vee (v_6 < 7) \vee \neg b_4) \wedge (b_3 \vee \neg(v_6 < 7) \vee b_4) \wedge \\ (\neg b_3 \vee (v_6 < 7) \vee b_4) \wedge (\neg b_3 \vee \neg(v_6 < 7) \vee \neg b_4) \wedge (b_4)\end{gathered}$$

Besides plain Boolean variables $b_1, b_2, b_3$ and $b_4$, the simple bound $(v_6 < 7)$ is also handled as a Boolean literal. The theory atom $((v_1 + sin(v_2)) \cdot v_3 < 7)$ is decomposed into three primitive constraints, each containing two or three variables and one arithmetic operation in order to facilitate simple ICP contractors for each operation.

iSAT3 solves the resulting CNF through a tight integration of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [DLL62] in its conflict-driven clause learning (CDCL) variant and interval constraint propagation [BG96]. Details of the algorithm, which operates on interval valuations for both the Boolean and the numeric variables and alternates between choice steps splitting such intervals and deduction steps narrowing them based on logical deductions computed through ICP or Boolean constraint propagation (BCP), can be found in [FHT+07]. Implementing branch-and-prune search in interval lattices and conflict driven clause learning of clauses comprising irreducible atoms in those lattices, it can be classified as an early implementation of abstract conflict driven clause learning (ACDCL) [BDG+13].

Additionally, for every integer- or real-valued variable $v$ an initial interval has to be provided. During the search process these intervals will be shrunk in order to find a solution – the two parameters *minimal splitting width* and *minimal progress* control when to stop the interval narrowing. We extend the three basic building blocks of CDCL in the following manner:

**Decisions:** besides deciding Boolean literals (and simple bounds) an integer or real-valued variable $v$ could be subject to a decision as well. This is done by splitting its interval and introducing a new simple bound which is then decided.

**Propagation:** Boolean Constraint Propagation (BCP) is applied to clauses containing Boolean literals and simple bounds. Additionally, ICP is applied to the primitive constraints. Each deduction performed by ICP which leads to a stronger lower or upper bound for a variable, generates a new clause consisting of simple bounds. E.g. for the primitive constraint $(v_5 = v_1 + v_4)$ and the interval valuations $v_1 \in [0, 10], v_4 \in [0, 10]$ and $v_5 \in [1, 100]$, ICP would deduce $((v_1 \leq 10) \wedge (v_4 \leq 10)) \rightarrow (v_5 \leq 20)$ This deduction is encoded in the clause $\neg(v_1 \leq 10) \vee \neg(v_4 \leq 10) \vee (v_5 \leq 20)$ which is then attached to the implication graph. Additionally, the implication graph also contains clauses which encode the linear ordering between existing simple bounds of a variable $v$ (e.g. $(v_1 < 5) \rightarrow (v_1 \leq 10)$). These clauses are generated lazily.

**Conflict resolution:** is similar to what a SAT-solver is doing, except that the implication

graph might contain clauses with simple bounds. The conflict clause is created according to the first unique implication point (1UIP) scheme.

Besides the results *satisfiable* and *unsatisfiable* the solver might also terminate with a *candidate solution*, because ICP cannot guarantee to reach point intervals for real-valued variables. Nonetheless, a candidate solution provides useful information, because the solver has proven that all primitive constraints contain non-conflicting interval valuations. If small perturbations of variables and constants are allowed, this is an indication for a possible solution under these perturbations. When verifying hybrid systems, one usually assumes some kind of robustness of the hybrid system against such small perturbations. If this is not the case, the system might show unwanted behaviour by being very fragile and sensitive to small changes in its environment.

### 4.3.3 iSAT3 interpolants

iSAT3 is also able to generate Craig interpolants. Here we exploit the similarities between iSAT3 and a CDCL SAT solver with respect to the conflict resolution. As atoms occurring as pivot variables in resolution steps are always simple bounds mentioning a single variable only, we are able to *straightforwardly generalize the technique employed in propositional SAT solvers* to generate partial interpolants [KB11]. depending on two rules; namely deduction and resolution rules.

**Deduction rule:** In order to apply deduction rule, there must be a clause such that this clause contains at most one unassigned literal that is not evaluated to false under the current status of the variable assignments of the solver. Then iSAT3 will deduce a new interval of the remaining literal with the help of the current interval assignments of dependent variables; e.g.; if we are given the following clause $(x < 0 \vee (x+1)^2 = y)$, where the current valid intervals of $x$ and $y$ are $\mu(x) = [2, 3]$ and $\mu(y) = [0, 10]$ respectively, then we deduce a new bounds of $y$ by ICP to be $\mu(y) = [9, 10]$.

**Resolution rule:** It is applied implicitly or explicitly in SAT solvers, however in iSAT3, it is only allowed to apply a resolution between clauses that contain only simple bounds; i.e. a variable, a logical operator and a well-defined value; e.g. integer or real. If two simple bounds together are found to be unsatisfiable, then iSAT3 will consider this situation exactly as a Boolean literal and its negation. Thus, one can apply the resolution step safely between the clauses that contain these simple bounds. However it is required that the resolvent will not be a tautology, e.g., if we are given the following clauses $(x < 3 \vee y \geq 12), (x \geq 5 \vee z \leq 2)$, then by resolving over $x$ variable since $x < 3 \wedge x \geq 5 \rightarrow \bot$, we get $(y \geq 12 \vee z \leq 2)$ as a resolvent clause. Both of deduction and resolution rules are formally presented in [KB11, KBTF11].

#### Slackness in iSAT3 interpolants

*slack.ness* *noun* /ˈslæk-nəs/
*slack:* a depression between hills, in a hillside, or in the land surface [Sla16].

As a valid interpolant for the pair $(A, B)$ is obtained since $A \wedge B$ is unsatisfiable, we know that such a valid interpolant must overapproximate $A$ formula and on the same time does not intersect with $B$. That is, a valid interpolant $\mathcal{I}$ would be computed in the area between $A$ and $B$ and extracted from so-called *local proof* [KV09]. Roughly, in local proofs some symbols are coloured in the red or blue colors and others are uncoloured. Uncoloured symbols are said to be grey. A *local proof* cannot contain an inference that uses both red and blue symbols. In other words, colors cannot be mixed within the same inference. In [HKV12], they control the size and the structure of the interpolants based on transformations of "grey area" of local proofs. The main metrics to judge the quality of resultant interpolants are:

- the size of the interpolants. As long as we get small interpolants, we mostly reduce the burden of solving complex formulae.
- the structure of the interpolants. It is found that CNF interpolants are more useful and practical in verifying unbounded model checking problems [VRN13].

The strength of the interpolants would be measured w.r.t. how close the interpolant is to $A$ or $B$, we say that interpolant $\mathcal{I}_1$ is stronger than interpolant $\mathcal{I}_2$ for the pair $(A, B)$, if and only if $\mathcal{I}_1$ and $\mathcal{I}_2$ are valid interpolants for $(A, B)$ and $\mathcal{I}_1 \rightarrow \mathcal{I}_2$. However, one cannot identify in advance whether the strong interpolants prove the safety in unbounded model checking problems faster than weaker ones or vice verse. Thus, we cannot attribute the quality of interpolants depending on their strengths.

In iSAT3, we can control the slackness of the interpolants on two levels. First, control the local proof by guiding the decide and deduce steps in a way such that we give the solver some preferences before any decide and deduce steps. Second, we choose appropriate interpolants computing rules e.g. Púdlak, McMillan and dual of McMillan upon request.

#### Biasing the heuristics in iSAT3

Guiding or controlling the heuristic of iSAT3 solver during decide or deduce steps means to guide the heuristic of iSAT3 where the proof tree would be changed according to this guidance. Additionally, this mechanism should be applied or considered in decide or deduce steps only as long as no other higher preferences have to be taken in our consideration. That is, if the solver finds a unit clause, then regardless of any heuristic, iSAT3 has to decide the remaining literal/atom to be true; otherwise the clause will be false, and consequently the formula is unsatisfiable.

**Decide step:** Prioritizing the decide queue which contains bunch of literals without prior preferences, can help us to decide on $A$-literals (related to $A$-clauses) or $B$-literals (related to $B$-clauses). This prioritizing would occur in following situations:

1. at the beginning of solving the problem in case of non-persistent unit-clauses,
2. after k-decisions where $k > 0$, we still do not get a conflict, and we can still decide on some literals,
3. and after non-chronological backtracking in CDCL(T), where iSAT3 may get on a situation where it has to decide as well.

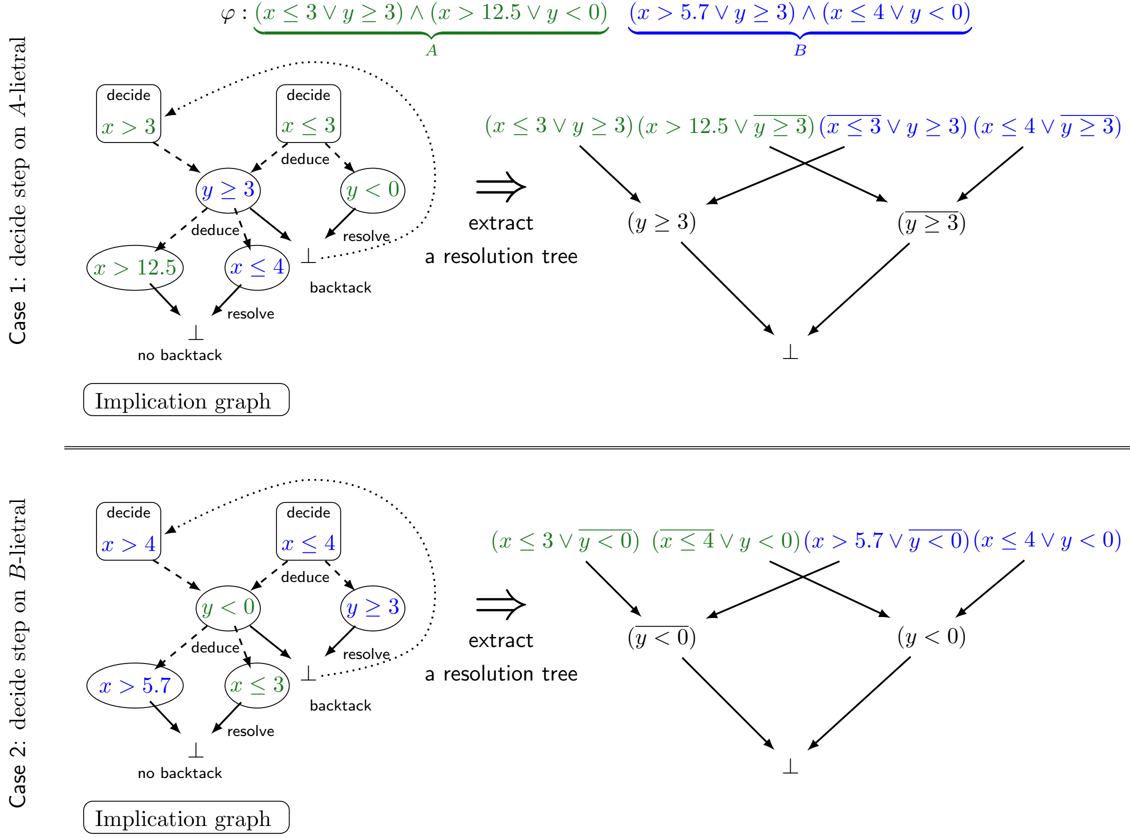



Figure 4.4: Guiding decide step in iSAT3 affects the resolution tree.

## Example 4.3: Example of decide step effect on proof

Consider the following formula as in Figure 4.4:

$$\varphi : A \wedge B$$

where $A : (x \leq 3 \vee y \geq 3) \wedge (x > 12.5 \vee y < 0)$ and $B : (x > 5.7 \vee y \geq 3) \wedge (y \leq 4 \vee y < 0)$.

Firstly, we do not have unit clauses to be considered. Thus, we have the liberty to decide on any literal at decide-level 0. In Case 1, we decide on a literal $l_0 : x \leq 3$ which belongs to $A$-formula. Thus, by the implication graph, we know that $l_1 : y \geq 3$ and $l_2 : y < 0$ have to be true, however $l_1 \wedge l_2 \models \mathit{false}$, we get a conflict. Therefore, we backtrack and decide that $l_0$ has to be false. Consequently, by the implication graph, $l_1 : y \geq 3$ has to be true. As a result of last deduction, $l_3 : x > 12.5$ and $l_4 : x \leq 4$ have to be true, however $l_3 \wedge l_4 \models \mathit{false}$. At this point, we get a non-resolvable conflict. On the right side of Figure 4.4, we have a corresponding extracted resolution tree.

In Case 2, we do the same procedure as before, but we decide on a literal which belongs to $B$ formula. At the end, we get another resolution tree as in the right bottom of Figure 4.4.

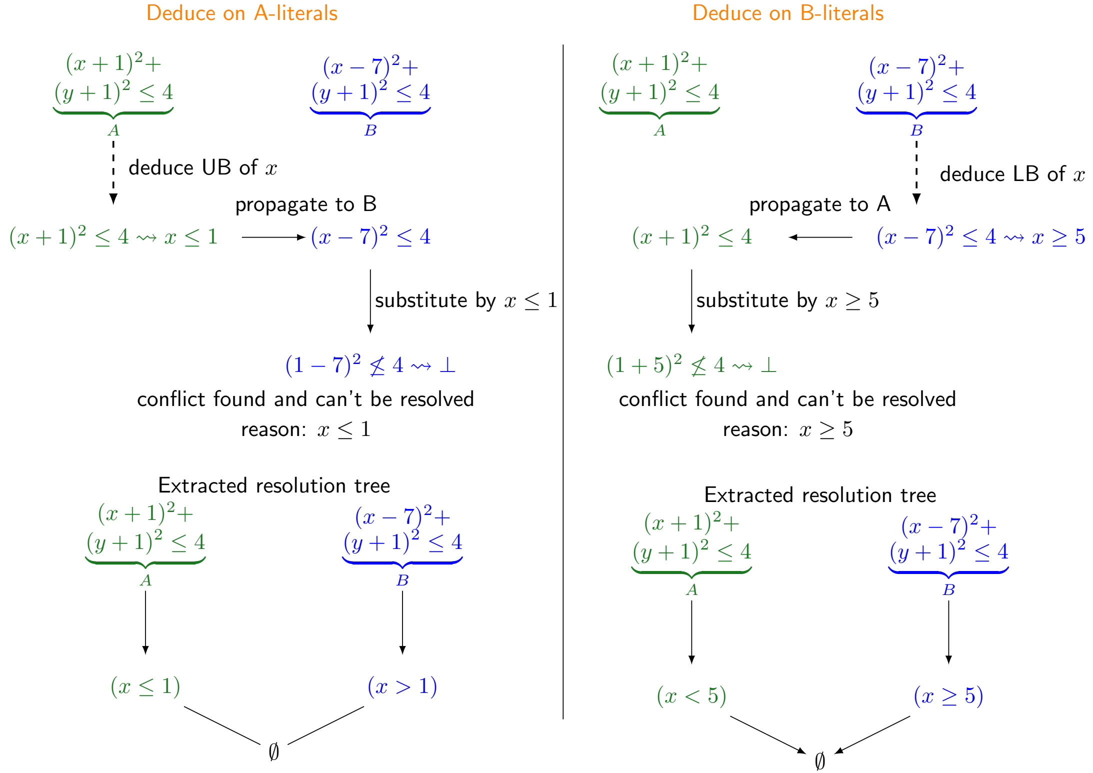



Figure 4.5: Guiding deduce step in iSAT3 affects the resolution tree.

**Deduce step:** In non-linear cases or complex constraints, iSAT3 will deduce upper and lower bounds of variables by using interval constraint propagation (ICP) as aforementioned. This early deduction application may trigger other literals on the formula and lead to a conflict clause. Thus, one needs to prioritize the deduction queue of the formula $A \wedge B$ such that one can choose which constraints yielding upper or lower bounds must be deduced earlier.

**EXAMPLE 4.4: EXAMPLE OF DEDUCE STEP EFFECT ON PROOF**

Consider the following formula as in Figure 4.5:

$$\varphi : A \wedge B$$

where $A : (x^2+y^2 \leq 4)$ and $B : (x-7)^2+(y+1)^2 \leq 4$. We are in the situation where we have only two atomic conjunctions $(A \wedge B)$, where both have to be satisfiable; otherwise the formula is unstisfaible. Thus, iSAT3 has to deduce safe bounds of each formula in order either to finds a common solution, or ends with a conflict.
In Case 1, iSAT3 deduces bounds on $A$ first, only then one gets an upper bound of $x$ which in our case is 1. We propagate this fact to $B$ part as in the left side of Figure 4.5, where we get a non-resolvable conflict.

In Case 2, iSAT3 deduces bounds on $B$ first, then one gets a lower bound of $x$ which in our case is 5. We propagate this fact to $A$ part as in the right side of Figure 4.5, where we get a non-resolvable conflict.
One can observe the differences in the extracted resolution trees from Case 1 and Case 2 depending on the deduce step.

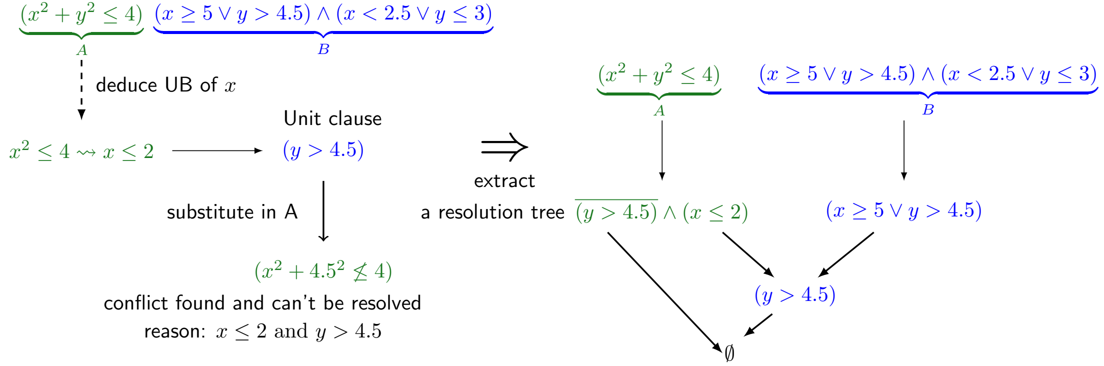



Figure 4.6: Possible influences between deduce and decide steps in iSAT3.

Deduce step sometimes affects the decide step; e.g., deducing an upper bound or a lower bound of a variable in one clause, triggers another literal(s)/atom(s) in binary clauses. Then we get a situation that we have a unit clause; i.e. we are forced to choose the remaining literal to be true as in Figure 4.6 where $(x^2 + y^2 \leq 4)$ has to be true.

Then we deduce a safe upper bound of $x$; namely 2. After that we get a unit clause $y > 4.5$. So deducing that $x^2 \leq 4$ forces us to decide that $y > 4.5$ is true. Sometimes, this situation is considered to be an effect of *arithmetic deduction* on *Boolean deduction*. Also, if iSAT3 decides a literal to be true, this may lead to deduce an upper or a lower bound of a variable in that literal. The latter situation happens probably in non-trivial constraints; e.g. $(x^2 + y^2 \leq 4)$ as in Figure 4.6.

We propose to extend the preferences of iSAT3 solver by two parameters either a-biasing or b-biasing. In case of $a$-biasing, iSAT3 is forced to deduce on $A$-literals first, then on $B$-literals. This behaviour holds at any point of time during solving as long as we do not have other preferences that have higher priority; e.g. unit clause. Also, in a-biasing, iSAT3 is forced to decide on any literal on $A$-formula part as long as we do not have unit clause on $B$ part and vice versa in case of $b$-biasing.

By this procedure, we guide the decide and deduce operations in implication graph either to use $A$ literals (in case of $a$-biasing) or $B$-literals (in case of $b$-biasing).

**Remark 4.1: Relation between biasing in iSAT3 and abduction rule in [BDG+14]**

In [BDG+14], the authors defined a so-called *abduction* rule in abstract conflict driven clause learning (ACDCL). This rule aims at finding a general reason by

relaxing bounds iteratively. It is found in practice to be very efficient and important. Whenever they get a reason; e.g. $x < 4.1$ of unsatisfiability, they step backwards through the deduction queue and attempt to weaken the current element by using most recent deduced bounds say $x < 7.3$ (more relaxed and more general). This abduction rule is more relevant to *backwards propagation* in ICP. It has some affinity with our biasing technique in a way that biasing technique in some cases relaxes the bounds to obtain at the end a general reason which is still sufficient to deduce the same fact or to lead to the conflict. However, our biasing technique is $A$ or $B$ driven one which does not intend to weaken or strengthen bounds.

**Lemma 4.1: Strong and weak interpolants**

Let $A$ and $B$ be SMT formulae such that $A \wedge B$ is unsatisfiable. The interpolant $\mathcal{I}_1$ obtained by using McMillan rules with a-biasing scheme is stronger than the interpolant $\mathcal{I}_2$ obtained by using duality of McMillan rules with b-biasing scheme.

**Proof of Strong and weak interpolants**

We prove this lemma considering two situations:

1. applying McMillan's rules and its duality to the same resolution proof tree.
   It is proven in [DKPW10] (Page 11, Lemma 2, Theorem 3, and Corollary 1) that the interpolant obtained by McMillan's rules is stronger than the one obtained by Púdlak's rules for the same resolution tree. Additionally, both are stronger than the interpolant obtained by the duality of McMillan's rules.
   The idea of the proof follows the fact that at any vertex $v$ in the resolution tree:
   $$\mathcal{I}_{mcmillan} \rightarrow \mathcal{I}_{du_mcmillan} \vee (C \downarrow_A \cap C \downarrow_B)$$
   where $\mathcal{I}_{mcmillan}$ is the McMillan's interpolant, $\mathcal{I}_{du_mcmillan}$ is the duality of McMillan's interpolant, and $C$ is the clause obtained at $v$ (resolvent).
2. applying the same computation rules to the resolution proof trees obtained after using biasing (say for example using McMillan's rules).
   The idea of the proof also follows the fact that at any vertex $v$ in the resolution tree:
   $$\mathcal{I}_{a-biased} \rightarrow \mathcal{I}_{b-biased} \vee (C \downarrow_A \cap C \downarrow_B)$$
   where $\mathcal{I}_{a-biased}$ is the interpolant obtained by $a$-biasing, $\mathcal{I}_{b-biased}$ is the interpolant obtained by $b$-biasing, and $C$ is the clause obtained at $v$ (resolvent). For more information, see [DKPW10].

By following 1 and 2, the lemma holds.

**Definition 4.10: Slackness of interpolants**

Given unsatisfiable formula $\varphi : A \wedge B$, we say that the resultant interpolants $\mathcal{I}_{strong}$ and $\mathcal{I}_{weak}$ have slackness if and only if:

- $\mathcal{I}_{strong}$ does not equal $\mathcal{I}_{weak}$; i.e., $\mathcal{I}_{strong} \not\leftrightarrow \mathcal{I}_{weak}$ and
- $\mathcal{I}_{strong}$ is stronger than $\mathcal{I}_{weak}$; i.e., $\mathcal{I}_{strong} \rightarrow \mathcal{I}_{weak}$.

### Examples

In order to show the effectiveness of using biasing together with (duality of) McMillan's rules, we will show some small but interesting cases, where our technique is obviously affecting the computed interpolants. Later, we will show how one can use this interesting results in order to obtain simple interpolant at the end for non-linear problems where the latter is still challenging in most non-linear unbounded model checking problems. In all following examples, $A$-formula is represented in blue color, $B$-formula is represented in green color, $\mathcal{I}_{strong}$-interpolant is represented in orange color and $\mathcal{I}_{weak}$-interpolant is represented in red color.

**Example 4.5: Two disjoint circles**

Consider the following two disjoint circles depicted in Figure 4.7a:

$$A : (x + 1)^2 + (y + 1)^2 \leq 4$$

$$B : (x - 7)^2 + (y - 7)^2 \leq 4$$

In Figure 4.7b we get the first interpolant $\mathcal{I}_{strong} : x \leq 1$, computed by McMillan's rules with $a$-biasing. In addition to that, in Figure 4.7c we get the second interpolant $\mathcal{I}_{weak} : x < 5$, computed by duality of McMillan's rules with $b$-biasing. In this example, we get a sufficient slackness between $\mathcal{I}_{strong}$ and $\mathcal{I}_{weak}$ such that $\mathcal{I}_{strong} \rightarrow \mathcal{I}_{weak}$ as shown in Figure 4.7d.

**Example 4.6: Two disjoint spheres**

Consider the following two disjoint spheres depicted in Figure 4.8a:

$$A : (x + 1)^2 + (y + 1)^2 + (z + 1)^2 \leq 4$$

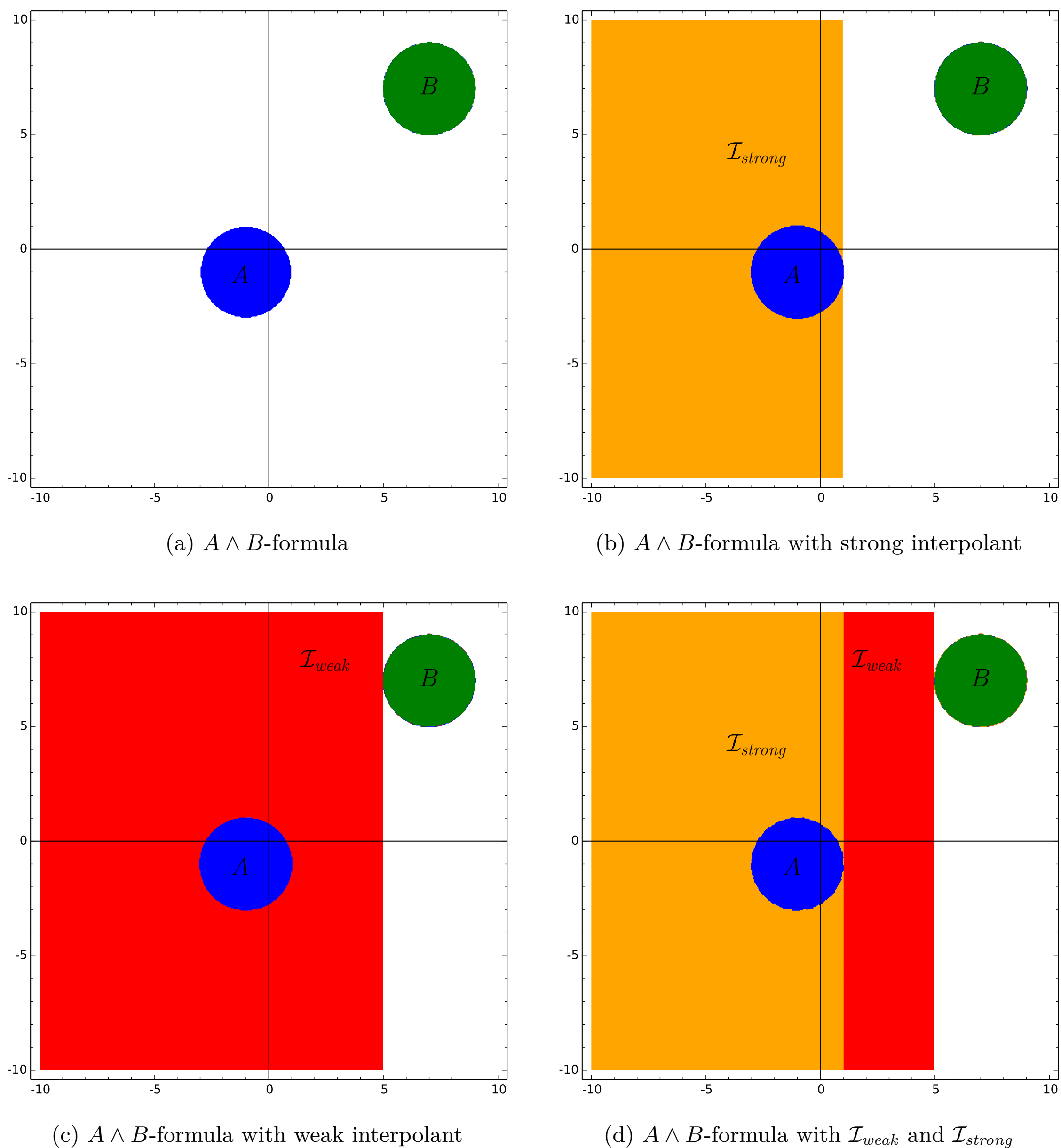
$$B : (x - 7)^2 + (y - 7)^2 + (z + 7)^2 \leq 4$$

In Figure 4.8b we get the first interpolant $\mathcal{I}_{strong} : x \leq 1 \wedge z \leq 1$, computed by McMillan's rules with $a$-biasing. In addition to that, in Figure 4.8c we get the second interpolant $\mathcal{I}_{weak} : x < 5 \vee z < 5$, computed by duality of McMillan's rules with $b$-biasing. In this example, we get a sufficient slackness between $\mathcal{I}_{strong}$ and $\mathcal{I}_{weak}$ such that $\mathcal{I}_{strong} \rightarrow \mathcal{I}_{weak}$ as shown in Figure 4.8d.

**Example 4.7: Two disjoint connected circles**

Consider the following two disjoint connected circles depicted in Figure 4.9a:

$$A : (x + 1)^2 + (y + 1)^2 \leq 1 \vee (x + 2.5)^2 + (y + 2)^2 \leq 1$$




(a) $A \wedge B$-formula

(b) $A \wedge B$-formula with strong interpolant

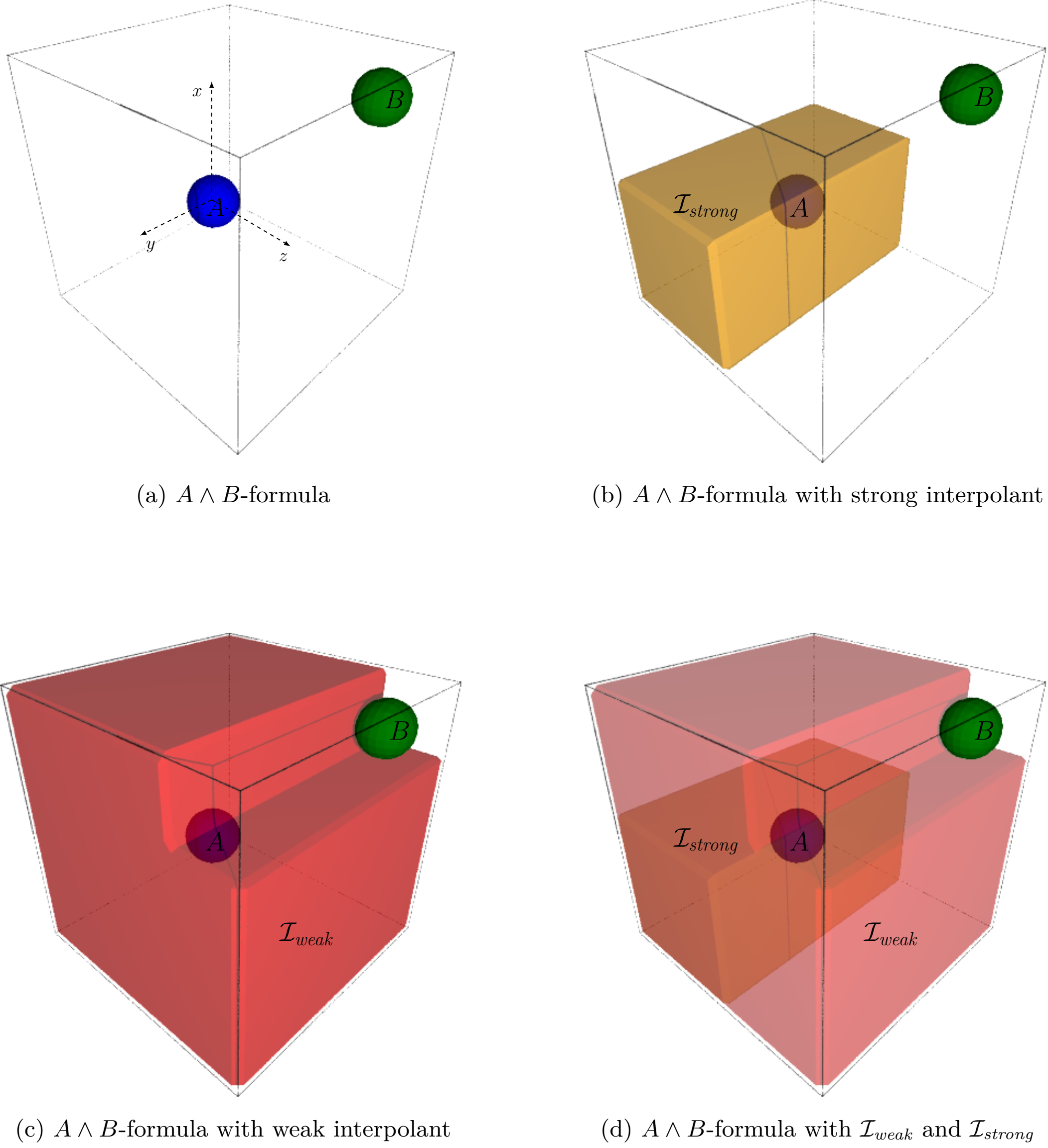
(c) $A \wedge B$-formula with weak interpolant

(d) $A \wedge B$-formula with $\mathcal{I}_{weak}$ and $\mathcal{I}_{strong}$

Figure 4.7: Two disjoint circles and two different interpolants with sufficient slackness.

$$B : (x + 4.5)^2 + (y - 0.5)^2 \leq 1 \vee (x + 3)^2 + (y - 1.5)^2 \leq 1$$

In Figure 4.9b we get the first interpolant $\mathcal{I}_{strong} : ((y \leq -1 \wedge x \geq -3.5) \vee (y \leq 0 \wedge x \geq -2))$, computed by McMillan's rules with $a$-biasing. In addition to that, in Figure 4.9c we get the second interpolant $\mathcal{I}_{weak} : ((y < -0.5 \vee x > -3.5) \wedge (y < 0.5 \vee x > -2))$, computed by duality of McMillan's rules with $b$-biasing. In this example, we get a sufficient slackness between $\mathcal{I}_{strong}$ and $\mathcal{I}_{weak}$ such that $\mathcal{I}_{strong} \rightarrow \mathcal{I}_{weak}$ as shown in Figure 4.9d.




(a) $A \wedge B$-formula

(b) $A \wedge B$-formula with strong interpolant

(c) $A \wedge B$-formula with weak interpolant

(d) $A \wedge B$-formula with $\mathcal{I}_{weak}$ and $\mathcal{I}_{strong}$

Figure 4.8: Two disjoint spheres and two different interpolants with sufficient slackness.

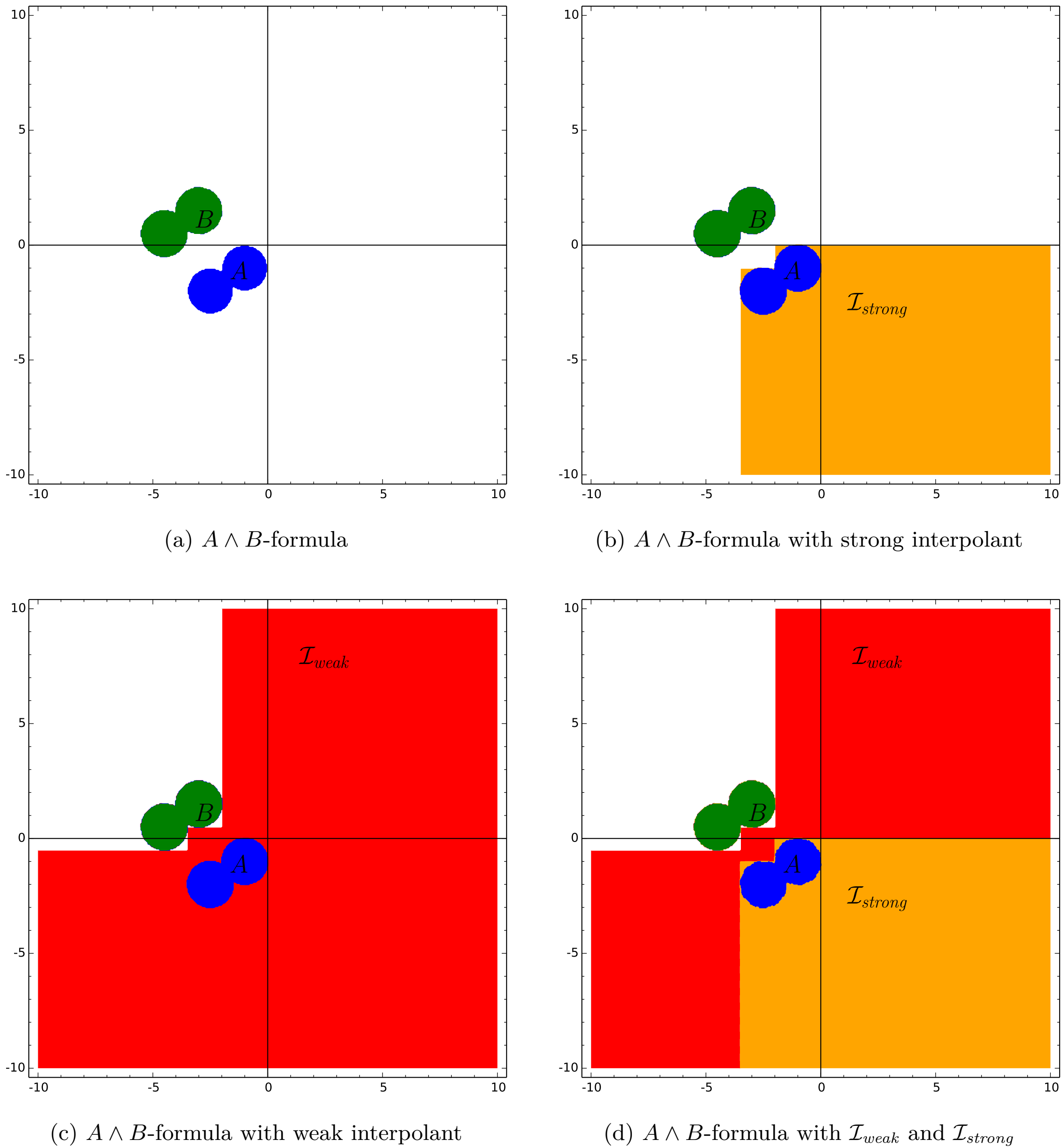
**Example 4.8: Two tori**

Consider the following two disjoint tori depicted in Figure 4.10a:

$$A : (\sqrt{x^2 + y^2} - 4)^2 + z^2 = 0.4$$

$$B : (\sqrt{x^2 + y^2} - 2)^2 + z^2 = 0.4$$

In Figure 4.10b we get the first interpolant $\mathcal{I}_{strong} : \sqrt{(x^2 + y^2)} \geq 3.368$, computed by McMillan's rules with $a$-biasing. In addition to that, in Figure 4.10c we get the second interpolant $\mathcal{I}_{weak} : \sqrt{(x^2 + y^2)} \geq 2.632$, computed by duality of McMillan's




(a) $A \wedge B$-formula

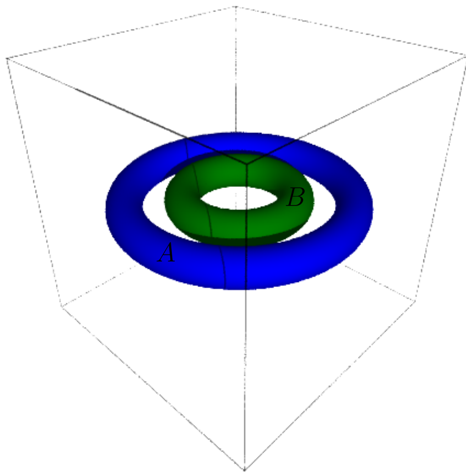
(b) $A \wedge B$-formula with strong interpolant

(c) $A \wedge B$-formula with weak interpolant

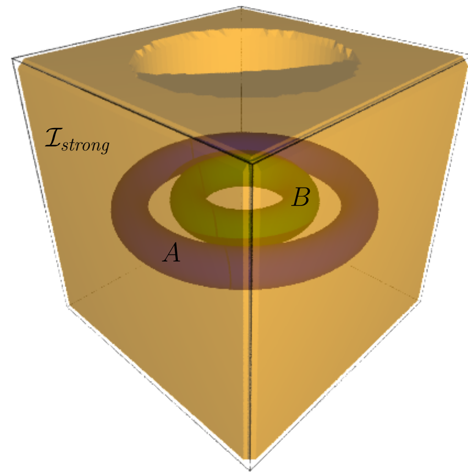
(d) $A \wedge B$-formula with $\mathcal{I}_{weak}$ and $\mathcal{I}_{strong}$

Figure 4.9: Two disjoint connected-circles and two different interpolants with sufficient slackness.

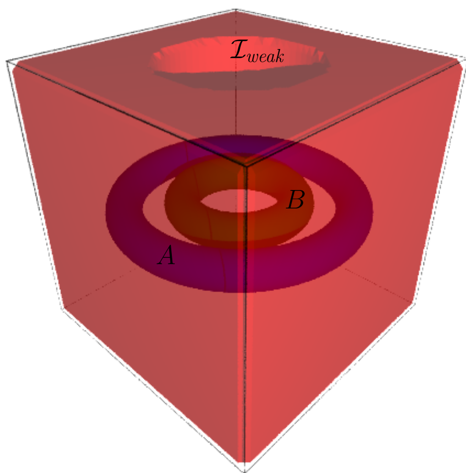
rules with $b$-biasing. In this example, we get a sufficient slackness between $\mathcal{I}_{strong}$ and $\mathcal{I}_{weak}$ such that $\mathcal{I}_{strong} \rightarrow \mathcal{I}_{weak}$ as shown in Figure 4.10d.
Although we get here a sufficient slackness, one should remark that the resultants interpolants in this example are non-linear formulae. The reason behind this verdict comes from the fact that iSAT3 finds a non-linear constraint during interval splitting that is sufficient to capture the reasoning of unsatisfiability. One can see it as an advantage since we get an interpolant earlier without deducing more the intervals.



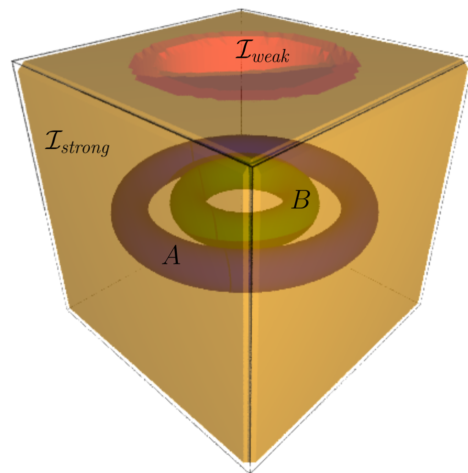

(a) $A \wedge B$-formula




(b) $A \wedge B$-formula with strong interpolant



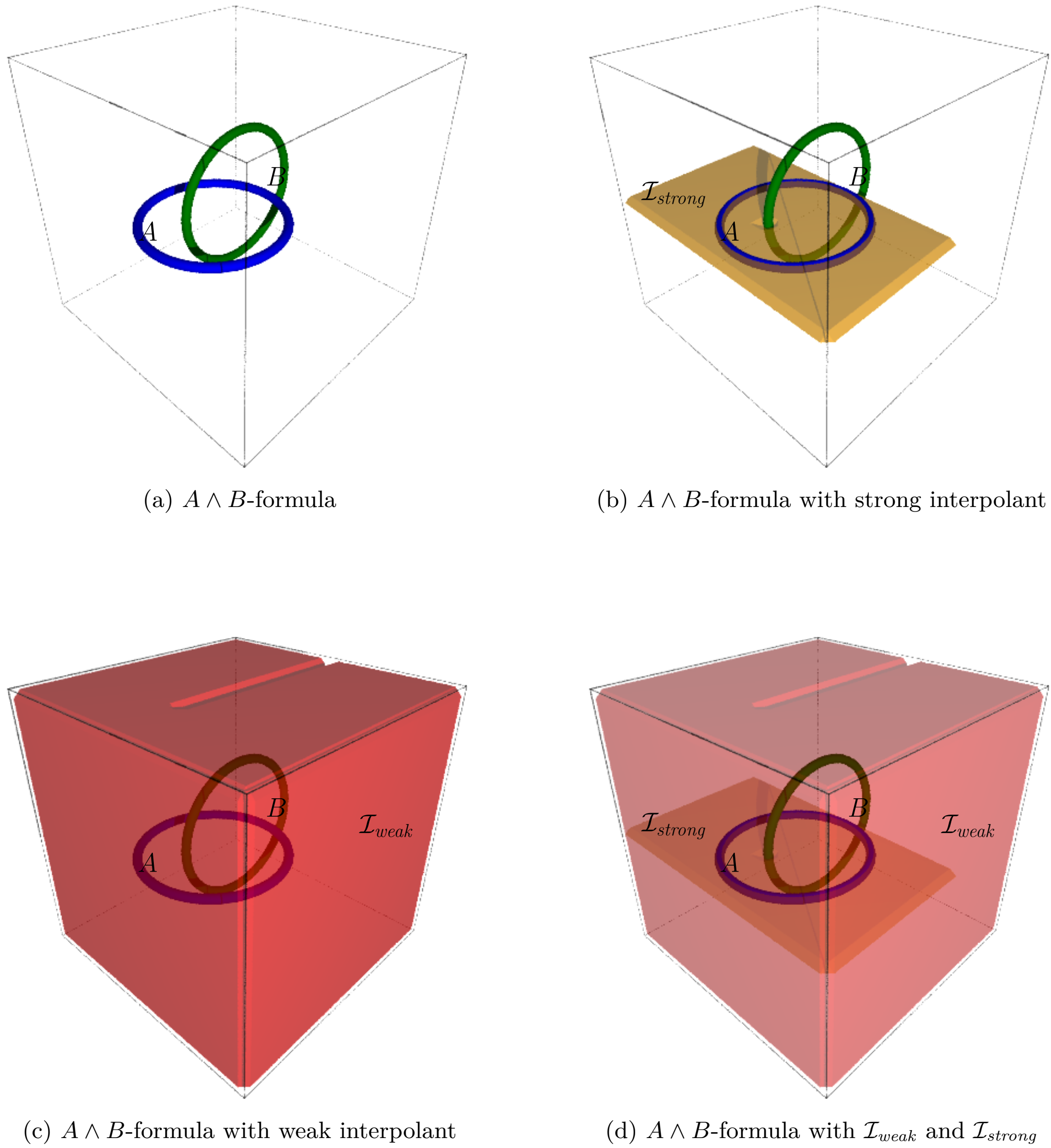

(c) $A \wedge B$-formula with weak interpolant




(d) $A \wedge B$-formula with $\mathcal{I}_{weak}$ and $\mathcal{I}_{strong}$

Figure 4.10: Two disjoint tori and two different interpolants with sufficient slackness.

On the other hand, it can be seen as a limitation of our approach, we cannot always get an interpolant with simple bounds.




(a) $A \wedge B$-formula

(b) $A \wedge B$-formula with strong interpolant

(c) $A \wedge B$-formula with weak interpolant

(d) $A \wedge B$-formula with $\mathcal{I}_{weak}$ and $\mathcal{I}_{strong}$

Figure 4.11: Two disjoint tori and two different interpolants with *semi* slackness.

### Limitations

**Example 4.9: Two tori with semi-slackness**

Consider the following two disjoint tori depicted in Figure 4.11:

$$A : (\sqrt{(x-0.5)^2 + (y+2)^2} - 3)^2 + (z+0.1)^2 = 0.04$$

$$B : (\sqrt{y^2 + z^2} - 3)^2 + x^2 = 0.04$$

Also, in Figure 4.11 we get two different interpolants i.e. $\mathcal{I}_{strong} : (y \leq 1.2 \wedge z \leq 0.1 \wedge (x < -0.2 \vee y < -3.2 \vee y > -2.784 \vee x > 0.2) \wedge z > -0.3)$ and $\mathcal{I}_{weak} : (x < -0.2 \vee y < -3.2 \vee (y \leq 1.186 \wedge z \leq 0.1 \wedge y \geq 0.711 \wedge z \geq -0.3) \vee x > 0.2))$ where they are computed according to McMillan's rules with $a$-biasing and duality of McMillan's rules with $b$-biasing respectively. However, according to Definition 4.10, we did not get slackness, since $\mathcal{I}_{strong}$ is not stronger than $\mathcal{I}_{weak}$. This result does not conflict with Lemma 4.1, since the limitation comes from iSAT3 implementation (technical issue) as the latter cannot always guarantee $a$-biasing or $b$-biasing due to **early deduction steps** performed by the solver while encoding the problem. Adjusting this soft spot of the solver requires us to change the core of the solver, thus we report this as a limitation of our implementation rather a tedious change of the solver core-engine.

#### Integrating slackness with downsizing interpolants [PS13a]

The slackness of interpolants aims at finding simple inteprolants as follows. First, by using biasing options to guide heuristics together with (duality of) McMillan's rules, we can obtain two different, linear and partially-ordered interpolants for the pair $(A, B)$, namely $\mathcal{I}_{strong}$ by using $a$-biasing with McMillan's rules and $\mathcal{I}_{weak}$ by using $b$-biasing with duality of McMillan's rules. Second, we integrate our approach with SMT downsizing interpolants [PS13a, SPDA14] in order to find a simple interpolant for the pair $(\mathcal{I}_{strong}, \neg\mathcal{I}_{weak})$. By simple interpolants, we mean interpolants with simple structure and less constraints with comparison to $\mathcal{I}_{strong}$ and $\mathcal{I}_{weak}$.

**Lemma 4.2: Simple interpolant**

An interpolant obtained from the pair $(\mathcal{I}_{strong}, \neg\mathcal{I}_{weak})$ is a valid interpolant for the pair $(A, B)$.

**Proof of Strong and weak interpolants**

In order to prove that the simple interplant $\mathcal{I}_{simple}$ is a valid interpolant for $(A, B)$, we have to prove the following:

- $A \rightarrow \mathcal{I}_{simple}$,
  Since $\mathcal{I}_{strong}$ is a valid interpolant for $(A, B)$, we know that: $A \rightarrow \mathcal{I}_{strong}$.
  Also, as $\mathcal{I}_{simple}$ is a valid interpolant for $(\mathcal{I}_{strong}, \neg\mathcal{I}_{weak})$, we know that: $\mathcal{I}_{strong} \rightarrow \mathcal{I}_{simple}$.
  Thus, from previous results, the first condition of valid interpolant holds.
- $\mathcal{I}_{simple} \rightarrow \neg B$,
  Since $\mathcal{I}_{weak}$ is a valid interpolant for $(A, B)$, we know that: $\mathcal{I}_{weak} \rightarrow \neg B$.
  Also, as $\mathcal{I}_{simple}$ is a valid interpolant for $(\mathcal{I}_{strong}, \neg\mathcal{I}_{weak})$, we know that: $\mathcal{I}_{simple} \rightarrow \mathcal{I}_{weak}$.
  Thus, from previous results, the second condition of valid inteprolant

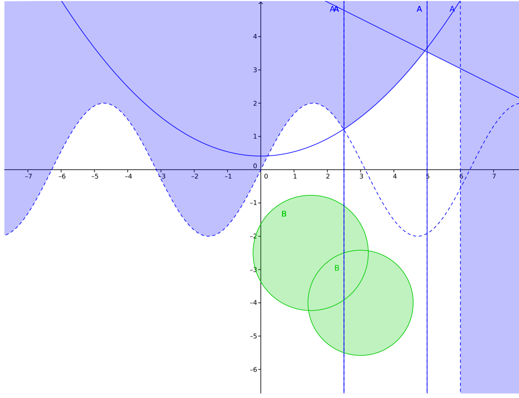
holds.

- $Var(\mathcal{I}_{simple}) \subseteq Var(A) \cap Var(B)$.

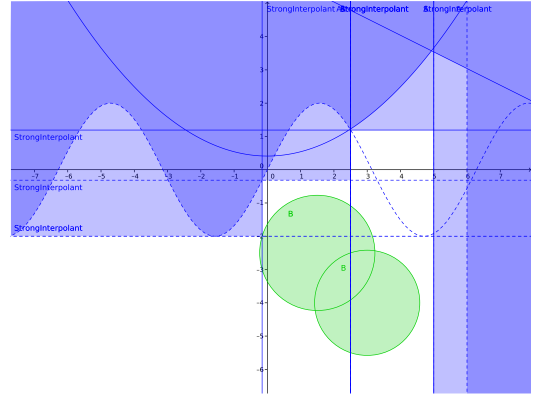
  Since $\mathcal{I}_{strong}$ and $\mathcal{I}_{weak}$ are valid interpolants for $(A, B)$, we know that: $Var(\mathcal{I}_{strong}) \subseteq Var(A) \cap Var(B)$ and $Var(\mathcal{I}_{weak}) \subseteq Var(A) \cap Var(B)$ and Also, as $\mathcal{I}_{simple}$ is a valid interpolant for $(\mathcal{I}_{strong}, \neg\mathcal{I}_{weak})$, we know that: $Var(\mathcal{I}_{simple}) \subseteq Var(\mathcal{I}_{weak}) \cap Var(\mathcal{I}_{strong})$.

  Thus, from previous results, the third condition of valid inteprolant holds.

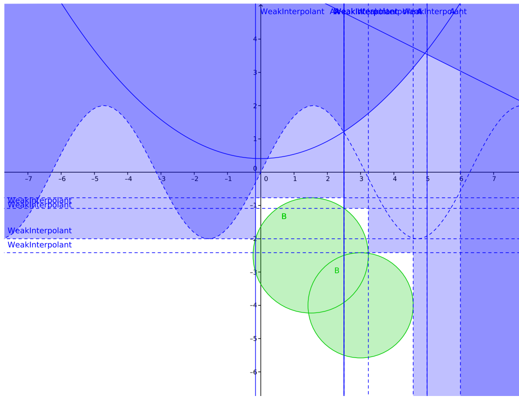
At the end, the three conditions of valid interpolants apply to $\mathcal{I}_{simple}$.



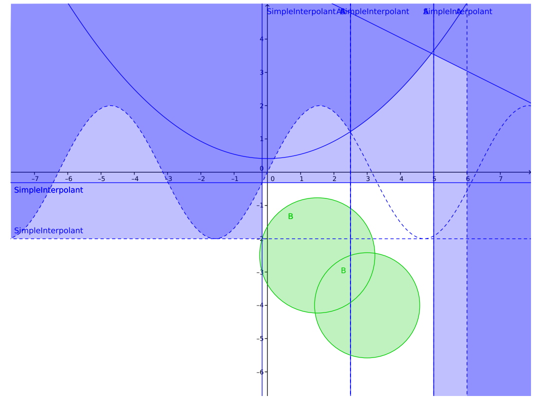

(a) $A \wedge B$-formula




(b) $A \wedge B$-formula with strong interpolant




(c) $A \wedge B$-formula with weak interpolant




(d) $A \wedge B$-formula with simple interpolant

Figure 4.12: Example of integrating iSAT3 with downsizing interpolants method where blue area represents $A$ formula and green area represents $B$ formula.

**Example 4.10: Simple interpolant**

Let us consider the following formula: $\varphi = A \wedge B$, where $A = ((x < 2.5) \rightarrow (y > 2 * sin(x))) \wedge ((x \geq 2.5 \wedge x < 5) \rightarrow (y \geq 0.125 * x^2 + 0.41)) \wedge ((x \geq 5 \wedge x \leq 6) \rightarrow (y \geq -0.5 * x + 6.04))$, and $B = ((x - 1.5)^2 + (y + 2.5)^2 \leq 3) \vee ((x - 3)^2 + (y + 4)^2 \leq 2.5)$

as in Figure 4.12a. By using McMillan's rules together with $a$-biasing technique, we get the following interpolant as in Figure 4.12b:

$$\begin{aligned}\mathcal{I}_{strong} : ((x \geq 2.5) \vee (x \leq -0.16) \vee (x \geq 2.5) \vee (y > -0.32))\wedge \\ (y > -2)\wedge \\ ((x < 2.5) \vee (x \geq 5) \vee (y \geq 1.19))\wedge \\ ((x < 2.5) \vee (x \geq 5) \vee (y > -2))\end{aligned}$$

where it has *four* clauses with clause average size *three*. By using duality of McMillan's rules together with $b$-biasing technique, we get the following interpolant as in Figure 4.12c:

$$\begin{aligned}\mathcal{I}_{weak} : (x < 2.5)\wedge \\ ((x \leq -0.16) \vee (y > -0.77))\wedge \\ ((y > -2) \vee (x \geq 2.5))\wedge \\ ((y > -1.09) \vee (x > 3.23))\wedge \\ ((y > -2.42) \vee (x > 4.58))\end{aligned}$$

where it has *five* clauses with clause average size (w.r.t. number of literals in each clause) *two*. Now, we pass the pair $(\mathcal{I}_{strong}, \neg\mathcal{I}_{weak})$ to SMT downsizing interpolants technique, which gives us the following simple interpolant as in Figure 4.12d

$$\mathcal{I}_{simple} : ((y \geq 0.32) \vee (x \geq 5) \vee (x < -2.5)) \wedge (y > -2) \wedge (x \leq -0.16)$$

where it has *three* clauses with average size *two*. Thus, baring slackness technique with downsizing interpolants technique en masse allows us to get simpler interpolants for non-linear problems, despite its incompleteness.

One observes the benefits of integrating these approaches together in this small non-linear example.

### 4.3.4 BMC problems in iSAT3

iSAT3 can solve a single quantifier-free formula as a constraint expression. Also, it can verify bounded and unbounded model checking reachability problems in symbolic transition-system based models. These transition systems may contain complex non-linear constraints which cannot be handled directly by other solvers without aggressive approximation or normalization due to non-linearity. In 2003, McMillan succeeded to use Craig interpolation [Cra57] to (dis)prove reachability of a specification in transition systems and programs over propositional and linear theories [McM03]. In 2011, Kupferschmid et al. [KB11] succeeded to extend the latter approach and solve unbounded reachability problems in the presence of non-linear constraints in the transition-system based models. The current status of iSAT3 can solve bounded and unbounded model checking problems by using Craig interpolation [Cra57] or k-induction [SSS00].

The general iSAT3 BMC-format [SKB13] is shown in Figure 4.13. It consists of *declaration*, *initialization*, *transition*, and *target* sections, which are all started by the respective keywords. The declaration part (DECL) defines all used variables and constants in all other parts. Non-Boolean variables must have an assigned initial range over which a solution is sought. Each defined constant has an assigned real-value. It is possible to define static variables that are assigned at most once in all transition system unrolling steps.

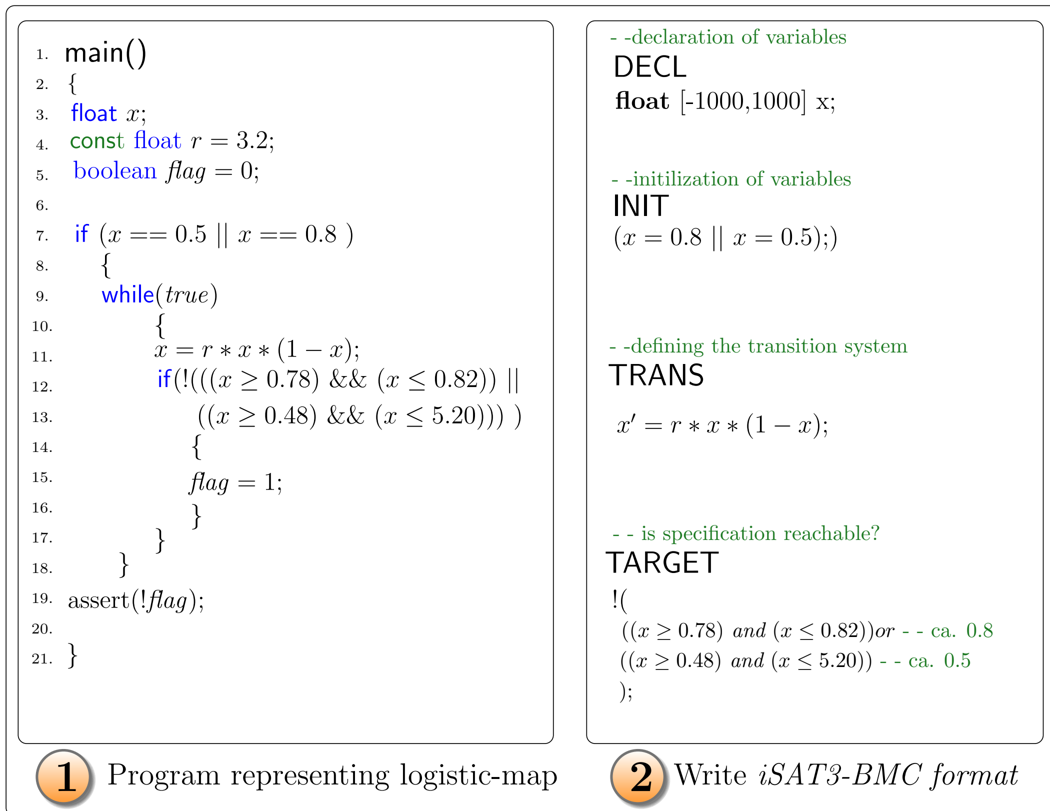



Figure 4.13: iSAT3 bounded model checking problem format. Left: a transition system representing logistic map problem [KB11], right: the corresponding encoding in iSAT3 format.

The initial part (INIT) defines the initial configuration of the transition system. Identifiers have not to be primed, because they represent the current status of the model (no next states).

The transitions part (TRANS) represents transition steps from current states to next states that are denoted by prime i.e. "′". One has to take in the consideration that iSAT3 treats all unspecified assignments and behaviours during unrolling as *open* or *free* assignments. That is, *no implicit assignments* or *values propagation* will be performed during unrolling. Thus, modeller must encode all necessary information; otherwise incorrect answers may appear at the end. The intuition behind this semantics is to consider that transitions part restricts all possible behaviours of the environment and unspecified constraints or assignments are left to hold any value from the environment. This semantics is a general one that can handle also programs where expected propagated assignments must be explicitly stated.

The final part is a target part (TARGET) which represents the set of targets or specifications to be reached.

iSAT3 considers a BMC problem to be mathematically encoded as follows:

$$INIT \wedge \bigwedge_{i=0}^{n} TRANS(x_i, x_{i+1}) \wedge \bigvee_{i=0}^{n} TARGET(x_i)$$

### 4.3.5 CFA problems in iSAT3

In order to encode control flow graphs in the iSAT3 input language, we extend the previous syntax of iSAT3 as shown in Figure 4.1 in order to encode control flow automaton problems directly in iSAT3. A control flow graph file in iSAT3 (iSAT3-CFG) contains five parts, namely the *declaration*, *nodes*, *initialization*, *edges*, and *specification* sections, which are all started by the respective keywords.

As in iSAT3, the declaration part (DECL) has the same description as before. The second part is the newly introduced nodes part (NODES), which defines the set of control flow graph nodes to be used as *source* or *destination* locations of transitions. The initialization part (INIT) then defines both the initial edge of the CFA and the permissible initial values of all program variables. The latter is achieved by stating a predicate confining the possible values. Its counterpart is the reachability specification, which may name the destination node or define a set of variable valuations to be reached.

The edges part, introduced by the keyword EDGES, represents the control flows in the graph. This part contains a list of edges as defined in Definition 4.3, each defined by a source node, a list of guards, a list of assigned variables that are changed, a list of assignments where the assigned variable has to be primed, and a destination node which has to be primed as well. In case that the list of assigned variables is empty, it means all previous values of variables are propagated. In contrast to the iSAT3 tradition (as in BMC format), a framing rule is applied such that all unspecified assignments and behaviours during unrolling are not considered to be non-deterministic choices, but values are maintained by implicit equations $x' = x$ for all unassigned variables.

The final part is a specification part; i.e., SPECIFICATION, represents a single node. The solver will investigate whether this node is reachable or not. The specification part here admits the syntax and semantics of the reachability property as in Definition 4.6 and Definition 4.7 respectively. iSAT3 considers a CFG as BMC problem to be mathematically encoded as follows:

$$INIT \wedge \bigwedge_{i=0}^{n} EDGES(x_i, x_{i+1}) \wedge \bigvee_{i=0}^{n} SPECIFICATION(x_i)$$

where *INIT* and *SPECIFICATION* are encoded as boolean predicates. *EDGES* part is encoded as a disjunction of edges formulas, i.e., $EDGES := \bigvee_{i=0}^{m} n_i \wedge \phi_i \wedge \vec{\psi}_i \wedge n_{i+1}$ where $m$ represents the number of edges in CFG.

## 4.4 Interpolation-based CEGAR technique

The basic steps in counterexample guided abstraction refinement (CEGAR) are to, first, compute an initial abstraction, then model-check it, thereafter terminate if no counterexample is found or trying to concretize the counterexample otherwise. If concretization succeeds then the counterexample is real, else spurious. In the latter case, a reason for the occurrence of the spurious counterexample is extracted and subsequently used for refining the abstraction, after which model checking is repeated.

As concretization of the abstract counterexample involves solving its concrete path condition, which is a conjunctive constraint system in the logical theory corresponding to the data domain of the program analyzed, SAT-modulo-theory solving often is the method of choice for concretization and Craig interpolation consequently a natural candidate for the extraction of reasons. It has been suggested by Henzinger et al. [HJMM04]. Of these classical approaches, we do in particular adopt lazy abstraction [HJMS02, McM06] and inductive interpolants in [BL12], yet lift them to the analysis of programs featuring arithmetic beyond decidable fragments. While CEGAR on such rich fragments of arithmetic has been pursued within the field of hybrid-system verification, in particular by Ratschan et al. [RS07], refinement there has not been directed by Craig interpolation and, using explicit-state techniques, the targets where relatively small control skeletons rather possibly unwieldy CFGs. By a tight integration of checking the abstraction and SMT including CI, we are trying to overcome such limitations.

### 4.4.1 Interpolation-based refinement procedure in iSAT3: algorithm

This subsection presents the four main steps of CEGAR in iSAT3; namely abstraction, abstract model verification, predicate extraction during counterexample validation, and the refinement step.

**Initial abstraction.** The first step of applying CEGAR is to extract an initial abstraction from the concrete model by a well-defined abstraction function. The first abstraction represents just the graph structure of the CFG without considering edge interpretations by assignments and guards. It is formally introduced as follows:

**Definition 4.11: Initial abstraction function**

Given a control flow automaton $\gamma = (N, E, i) \in \Gamma$, its initial abstraction mediated by the abstraction function $\alpha : \Gamma \to \Gamma$ is the CFA $\alpha(\gamma) = (N, E', i)$, where $E' = \{(n, \mathit{true}, \langle\rangle, n') \mid (n, \phi, \vec{\psi}, n') \in E\}$.

The abstraction function $\alpha$ allows any evolution w.r.t. reachability properties of the system that respects the flows between the nodes. For technical reasons, we assume that each edge in $\gamma$ is labelled by a *unique variable*. Thereby, for each edge in the concrete model, there is a corresponding edge in the abstract one, even if two edges have the same source and destination nodes, *they will not be merged as one edge* in the abstract model.

**Verifying the abstraction.** In the model checking community it is habitual to verify reachability problems in the abstract model by using finite-state model checkers, like BDD-based approaches [BR00]. In this work, we verify reachability properties in the abstract models by SMT solving together with interpolation [McM03] in order to verify reachability for unbounded depths. The individual runs of thus unbounded SMT-based model-checking are bound to terminate, as the initial abstraction is equivalent to a finite-state problem and as the predicates that are added to enrich the abstraction during refinement are just logical formula over simple bounds $x \sim c$ which are bounds on Boolean propositions; i.e., *literals*, thus keeping the model finite-state. By this idea, we can pursue model checking of the abstraction and the concretizability test of abstract counterexamples within the same tool, thus avoiding back-and-forth translation between different tools and checking technologies.

**Path-condition generation and extraction of reasons.** Given that the abstract model $\alpha(\gamma)$ is a CFA, it induces a set of paths. We call any path $\sigma_{sp} \in \Sigma(\alpha(\gamma))$ an *abstract path.* As the abstraction function just relaxes edge conditions, we can build a corresponding concrete path – if existent – by just reintroducing the missing constraints and assignments such that the variable instances are adequately renamed respecting the step-dependent depth as in BMC. It is formally defined as follows.

**Definition 4.12: Path-conditions generation function**

Given a control flow graph $\gamma = (N, E, i)$ and its abstraction $\alpha(\gamma) = (n, E', i) \in \Gamma$ and a finite abstract path $\sigma_{sp} : \langle i, \nu'_{init} \rangle \xrightarrow{e'_1} \langle n_1, \nu'_1 \rangle \xrightarrow{e'_2} ... \xrightarrow{e'_m} \langle n_m, \nu'_m \rangle \in \Sigma(\alpha(\gamma))$, the path-conditions generation function $\kappa : \Gamma \times \Sigma \to \Sigma$ that builds a concrete path semantically by collecting its conditions, is defined as follows:

$$\kappa(\gamma, \sigma_{sp}) = \sigma \text{ where, } \sigma : \langle i, \nu_{init} \rangle \xrightarrow{e_1} \langle n_1, \nu_1 \rangle \xrightarrow{e_2} ... \xrightarrow{e_m} \langle n_m, \nu_m \rangle,$$

$$\{e_1, ..., e_m\} \subseteq E \text{ and } \{n_1, ..., n_m\} \subseteq N$$

We say that $\sigma$ is a real path if and only if its generated path condition, i.e., $\nu_{ini} \wedge \bigwedge_{i=1}^{m} \phi_i \wedge \vec{\psi_i}$ is satisfiable, else it is spurious. In $\nu_{ini} \wedge \bigwedge_{i=1}^{m} \phi_i \wedge \vec{\psi_i}$ formula, each variable has to renamed in order to respect its depth akin to BMC renaming mechanism.

The crucial step in the CEGAR loop is to extract a reason for counterexamples being spurious such that case splitting on that reason would exclude the particular (and similar) counterexamples. Several previous works used different approaches and schemes to capture such reasons, like state splitting [RS07], word matching by using $\omega$-automata [Seg07], or interpolants [McM06, HJMS02, HJMM04, EKS08, BL12]. In our work, we exploit stepwise interpolants as in [EKS08, BL12] in order to obtain predicates capturing the reasons, where the first and last interpolants during refining any spurious counterexample are always *true* and *false* respectively [EKS08]. This can be carried out as follows: When encountering a spurious counterexample $\sigma_{sp} = \langle i, \nu'_{init} \rangle \xrightarrow{e'_1} ... \xrightarrow{e'_m} \langle n_m, \nu'_m \rangle \in \Sigma(\gamma')$, where $\gamma'$ is an abstraction, $\{e'_1, .., e'_m\} \subseteq E'$ – primed edges denote abstract ones –, $m > 0$ and $\theta = n_m$ is the goal to be reached,

- we complete the abstract path $\sigma_{sp}$ in the original model $\gamma$ semantically by using the path-conditions generation function $\kappa$ as in Definition 4.12.
- as $\sigma_{sp}$ is spurious, we obtain an unsatisfiable path formula $\kappa(\gamma, \sigma_{sp}) \notin \Sigma(\gamma)$, i.e., $\nu_{init} \wedge \bigwedge_{i=1}^{m} \phi_i \wedge \vec{\psi}_i \models \texttt{false}$.
- by using CI in order to extract reasonable predicates as in lazy abstraction [HJMM04], one computes a reason of unsatisfiability at each control point (node) of $\gamma$. For example, consider that $\kappa(\gamma, \sigma_{sp})$ equals $A \wedge B$, where $A = \nu_{init} \wedge \bigwedge_{j=1}^{k} \phi_j \wedge \vec{\psi}_j$, $B = \bigwedge_{j=k+1}^{m} \phi_j \wedge \vec{\psi}_j$ and $0 \leq k \leq m$. If we run the iSAT3 solver iteratively for all possible values of $k$, we obtain $m+1$ interpolants, where interpolant $I_k$ is an adequate reason at edge $e_k$ justifying the spuriousness of $\sigma_{sp}$.
- in case of using *inductive interpolants*, one uses the interpolant of iteration $k$, i.e., $\mathcal{I}_k$ as $A$-formula while interpolating against the above formula $B$ in order to obtain interpolant $\mathcal{I}_{k+1}$. As $\mathcal{I}_k$ overapproximates the prefix path formula till $k$, we compute the next interpolant $\mathcal{I}_{k+1}$ that overapproximates $\mathcal{I}_k \wedge \phi_{k+1} \wedge \vec{\psi}_{k+1}$. This step assures that the interpolant at step $k$ implies the interpolant at step $k+1$.

This guarantees that the interpolants at the different locations achieve the goal of providing a reason eliminating the infeasible error path from further exploration.

**Abstraction refinement.** After finding a spurious counterexample and extracting adequate predicates from the path, we need to refine the abstract model in a way such that at least this counterexample is excluded from the abstract model behaviour. This refinement step can be performed in different ways. The first way is a *global refinement* a.k.a. *entire refinement* procedure which is the earliest traditional approach, where the whole abstract model is refined after adding a new predicate [Cla03]. It is considered to be quite expensive and unwieldy approach unless the discovered predicates are useful for other unexplored paths. The second way is a *lazy abstraction* [HJMS02, McM06, Jha06] where instead of iteratively refining an abstraction, it refines the abstract model on demand, as it is constructed. This refinement has been based on predicate abstraction [HJMS02] or on interpolants derived from refuting program paths [McM06]. The common theme, however, has been to refine and thus generally enlarge the discrete state-space of the abstraction on demand such that the abstract transition relation could locally disambiguate post-states (or pre-states) in a way eliminating the spurious counterexample. The third way is an *eager abstraction* [SS14], where in each call two traces are explored; i.e., an error path if it exists and a safe path. Then a maximum common prefix is determined, and state sets which separate between safe and unsafe states are constructed without requiring any refinement.

Our approach of checking the abstraction within an SMT solver (by using interpolation-based model checking) rather than a separate finite-state model checker facilitates a subtly different solution.

At this point we would like to draw a special attention to our contribution in CEGAR refinement. **Instead of explicitly splitting states in the abstraction**, i.e., refining the nodes of the initial abstraction, we stay with the initial abstraction **and just add adequate pre-post-relations to its edges** that imitate the same role of splitting cases,

without, however increasing the number of neither states nor the transitions. These pre-post-relations are akin to the ones analysed when locally determining the transitions in a classical abstraction refinement, yet play a different role here in that they are not mapped to transition arcs in a state-enriched finite-state model, but rather added merely syntactically to the existing edges, whereby they only refine the transition effect on an unaltered state space. It is only during path search on the (refined) abstraction that the SMT solver may actually pursue an implicit state refinement by means of case splitting; being a tool for proof search, it would, however, only do so on demand, i.e., only when the particular case distinction happens to be instrumental to reasoning. We support this implicit refinement technique for both lazy abstraction (with inductive interpolants as optional configuration) and global refinement.

In the following we concisely state how the implicit refinement is performed by attaching pre-post-conditions to edges. Given a spurious counterexample $\sigma_{sp} = \langle i, \nu_{init} \rangle \xrightarrow{e'_1} ... \xrightarrow{e'_m} \langle n_m, \nu_m \rangle \in \Sigma(\gamma')$ with $\theta = n_m$ as shown in the previous subsection, we obtain $m+1$ (optionally inductive) interpolants, where $\mathcal{I}_k$ and $\mathcal{I}_{k+1}$ are consecutive interpolants at edges $e_k$ and $e_{k+1}$, respectively, and $0 < k < m$. We continue as follows:

1. if $\mathcal{I}_k \wedge \phi_{k+1} \wedge \vec{\psi}_{k+1} \rightarrow \mathcal{I}_{k+1}$ holds, then we add $\mathcal{I} \rightarrow \mathcal{I}'$ to $e'_{k+1}$,
2. if $\mathcal{I}_k \wedge \phi_{k+1} \wedge \vec{\psi}_{k+1} \rightarrow \neg\mathcal{I}_{k+1}$ holds, then we add $\mathcal{I} \rightarrow \neg\mathcal{I}'$ to $e'_{k+1}$,
3. if $\neg\mathcal{I}_k \wedge \phi_{k+1} \wedge \vec{\psi}_{k+1} \rightarrow \neg\mathcal{I}_{k+1}$ holds, then we add $\neg\mathcal{I} \rightarrow \neg\mathcal{I}'$ to $e'_{k+1}$,
4. if $\neg\mathcal{I}_k \wedge \phi_{k+1} \wedge \vec{\psi}_{k+1} \rightarrow \mathcal{I}_{k+1}$ holds, then we add $\neg\mathcal{I} \rightarrow \mathcal{I}'$ to $e'_{k+1}$,

where $\mathcal{I}$ is $\mathcal{I}_k$ with all its indexed variable instances $x_k$ replaced by undecorated base names $x$ and $\mathcal{I}'$ is $\mathcal{I}_{k+1}$ with all its indexed variable instances $x_{k+1}$ replaced by primed base names $x'$. These **four checks** capture all possible sound relations between the predecessor and successor interpolants.

For example, consider for the moment the abstract model in the first iteration as in Figure 4.14b, Image 2 and Image 3, where counterexample is found at depth 4. Interpolation on the path condition of the spurious counterexample yields $\mathcal{I}_1 := true$ and $\mathcal{I}_2 := x \geq 0.0002$ at nodes $n_3$ and $n_4$ respectively. By performing the previous four checks, we obtain three valid checks, namely

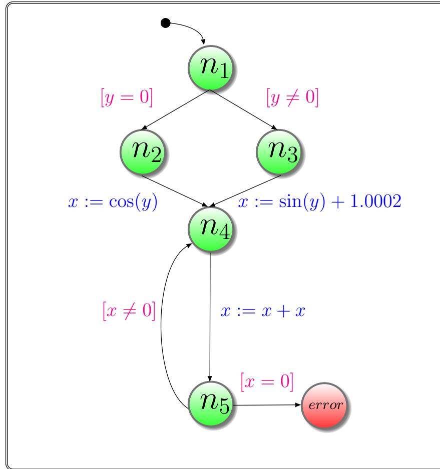
$$\underbrace{true}_{\mathcal{I}_1} \wedge \underbrace{true}_{\phi_1} \wedge \underbrace{x_2 = sin(y_1) + 1.0002 \wedge y_2 = y_1}_{\vec{\psi}_1} \rightarrow \underbrace{x_2 \geq 0.0002}_{\mathcal{I}_2}$$

$$\underbrace{false}_{\neg\mathcal{I}_1} \wedge \underbrace{true}_{\phi_1} \wedge \underbrace{x_2 = sin(y_1) + 1.0002 \wedge y_2 = y_1}_{\vec{\psi}_1} \rightarrow \underbrace{x_2 \geq 0.0002}_{\mathcal{I}_2}$$

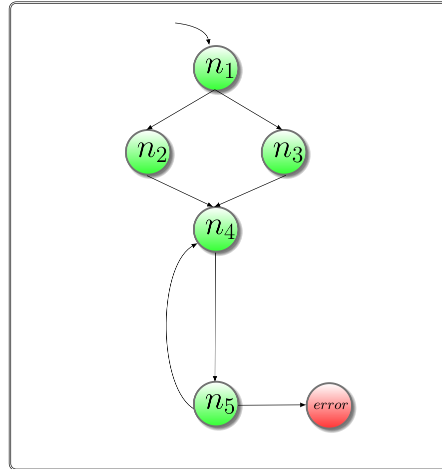
$$\underbrace{false}_{\neg\mathcal{I}_1} \wedge \underbrace{true}_{\phi_1} \wedge \underbrace{x_2 = sin(y_1) + 1.0002 \wedge y_2 = y_1}_{\vec{\psi}_1} \rightarrow \underbrace{x_2 < 0.0002}_{\neg\mathcal{I}_2}$$

As a result of these valid checks, we get the following conjunction:

$$(true \rightarrow x' \geq 0.0002) \wedge (false \rightarrow x' \geq 0.0002) \wedge (false \rightarrow x' < 0.0002)$$

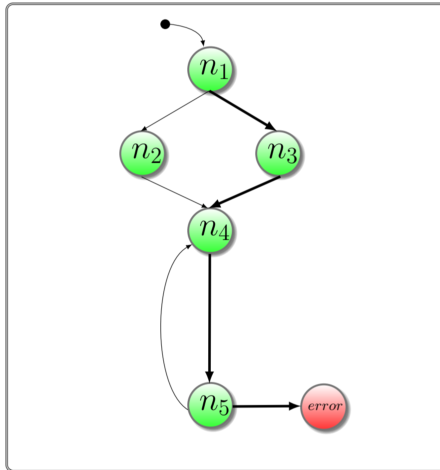



0 Original model

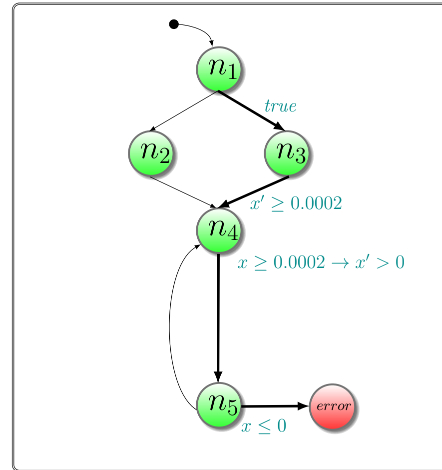



1 Initial abstract model after applying $\alpha$

(a) First step in CEGAR procedure.

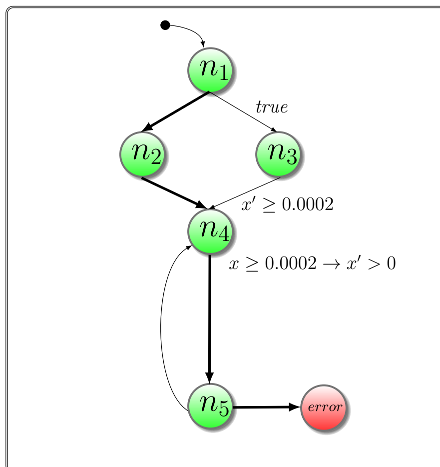



2 Counterexample in $1^{st}$ iteration

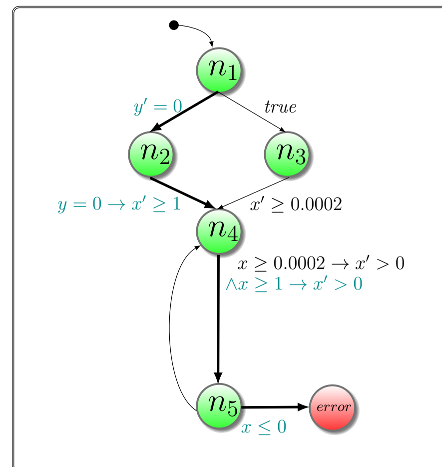



3 Abstract model after $1^{st}$ iteration

(b) Second step in CEGAR procedure.

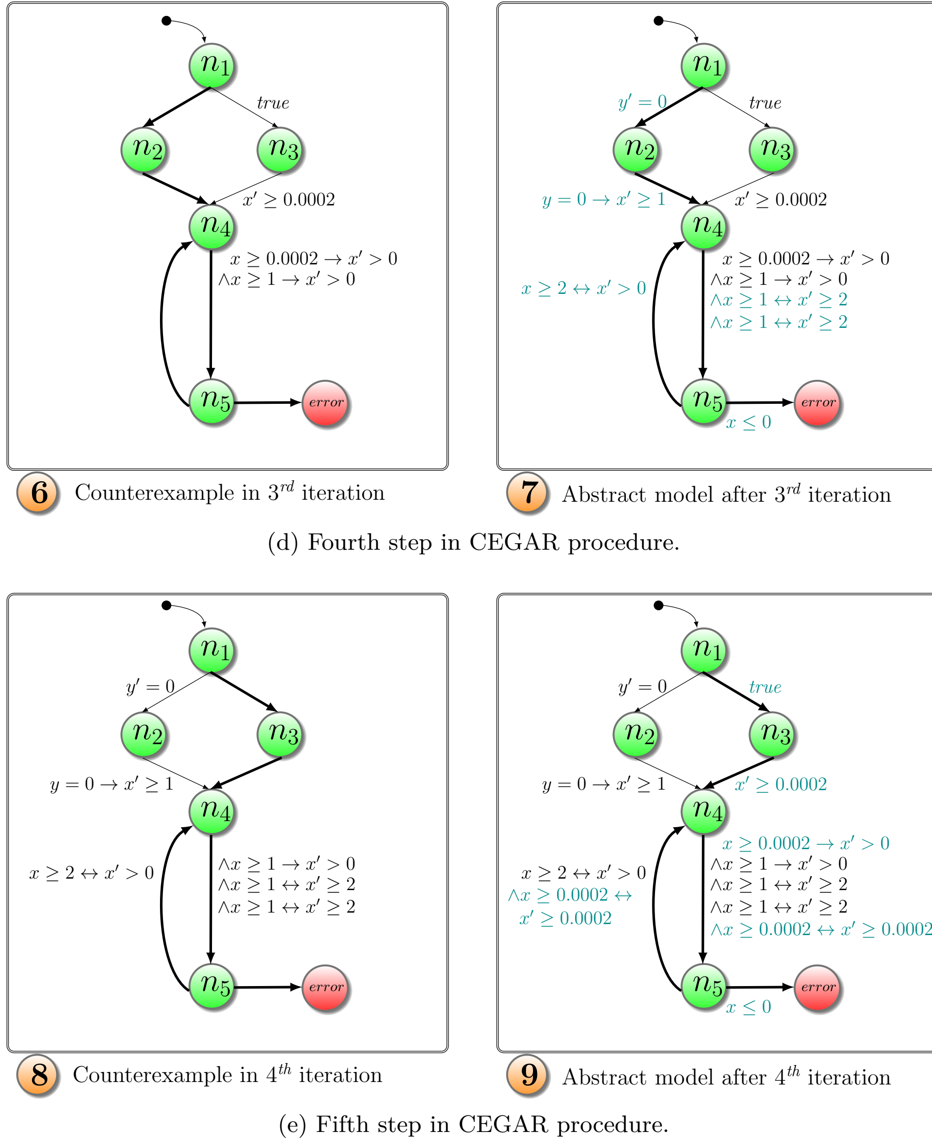



4 Counterexample in $2^{nd}$ iteration




5 Abstract model after $2^{nd}$ iteration

(c) Third step in CEGAR procedure.




(d) Fourth step in CEGAR procedure.

(e) Fifth step in CEGAR procedure.

Figure 4.13: CEGAR procedure to solve Example 4.1, where bold **paths** and cyan predicates represent the current counterexample and added constraints in each iteration after refinement respectively.

Now, from previous conjunction and after simplification, we construct the pre-post-predicate $true \wedge x' \geq 0.0002$ as shown on the arc from $n_3$ to $n_4$ of Figure 4.14a. We can derive that the pre-post-predicate thus obtained is a sufficient predicate to refine not only the abstract model at edge $e'_{k+1}$ for eliminating the current spurious counterexample, but also for any other spurious counterexample that (1) has a stronger or the same precondition before traversing edge $e'_{k+1}$ and (2) has a stronger or the same postcondition after traversing edge $e'_{k+1}$. This result is formally introduced as follows:

**Lemma 4.3: Eliminating several CEXs in one refinement**

Given an unsatisfiable path formula $\nu_{init} \wedge \bigwedge_{i=1}^{m} \phi_i \wedge \vec{\psi_i}$, where $\mathcal{I}_k$ and $\mathcal{I}_{k+1}$ are the interpolants extracted at $e_k$ and $e_{k+1}$ respectively, then $\mathcal{I}_k$ and $\mathcal{I}_{k+1}$ are sufficient reasons to eliminate any counterexample that

- has a stronger or the same precondition before traversing edge $e'_{k+1}$ and
- has a stronger or the same postcondition after traversing edge $e'_{k+1}$

**Proof of Eliminating several CEXs in one refinement**

The proof of this lemma is straightforward. Firstly, for the current counterexample assume that the prefix path formula $\nu_{init} \wedge \bigwedge_{i=1}^{k} \phi_i \wedge \vec{\psi_i}$ is *pre*, the conditions $\phi_k \wedge \vec{\psi}_{k+1}$ are *cond*, and the suffix of path formula $\bigwedge_{i=1}^{k+2} \phi_i \wedge \vec{\psi_i}$ is *suff*.
Consider that we have another spurious counterexample say $\sigma_{sp}$ which only shares with the current one the edge $e_{k+1}$ and fulfils the condition of this lemma; namely its prefix formula $pre_s$ is stronger than $pre$ i.e. $pre_s \rightarrow pre$ and its suffix formula $suff_s$ is stronger than $suff$ i.e. $suff_s \rightarrow suff$.
We know that $pre_s \rightarrow pre$ and $pre \rightarrow \mathcal{I}_k$. Thus $pre_s \rightarrow \mathcal{I}_k$ (by transitivity). Also, $suff_s \rightarrow suff$ and $suff \rightarrow \neg\mathcal{I}_k$. Thus $suff_s \rightarrow \neg\mathcal{I}_k$ (by transitivity). So $\mathcal{I}_k$ is a valid interpolant for the pair $(pre_s, cond \wedge suff_s)(*)$.
By the same way, we can prove that $\mathcal{I}_{k+1}$ is a valid interpolant for the pair $(pre_s \wedge cond, suff_s)(**)$.
From (*) and (**), the lemma holds.

One has to observe that although the previous four checks seem to be checked individually, there are certain dependencies between them. In other words, under certain conditions; namely if the prefix formula that is interpolated by $\mathcal{I}_k$, is valid (a tautology), then many relations can be deduced during solving the problem. These relation are stated formally in the following corollary:

**Corollary 4.1: Dependencies between the four interpolants checks**

Given an unsatisfiable path formula $\nu_{init} \wedge \bigwedge_{i=1}^{m} \phi_i \wedge \vec{\psi_i}$, where $\mathcal{I}_k$ and $\mathcal{I}_{k+1}$ are the interpolants extracted at the edges $e_k$ and $e_{k+1}$ respectively. If we assume that $\nu_{init} \wedge \bigwedge_{i=1}^{k} \phi_i \wedge \vec{\psi_i}$ is valid where $k < m$ and the following relation checks hold:

1. $\mathcal{I}_k \wedge \phi_{k+1} \wedge \vec{\psi}_{k+1} \rightarrow \mathcal{I}_{k+1}$ $(check_1)$
2. $\mathcal{I}_k \wedge \phi_{k+1} \wedge \vec{\psi}_{k+1} \rightarrow \neg\mathcal{I}_{k+1}$ $(check_2)$
3. $\neg\mathcal{I}_k \wedge \phi_{k+1} \wedge \vec{\psi}_{k+1} \rightarrow \neg\mathcal{I}_{k+1}$ $(check_3)$
4. $\neg\mathcal{I}_k \wedge \phi_{k+1} \wedge \vec{\psi}_{k+1} \rightarrow \mathcal{I}_{k+1}$ $(check_4)$

Then the following dependencies/consequences hold:

1. if $check_1$ holds, then $check_3$ and $check_4$ hold.
2. if $check_2$ holds, then $check_1$, $check_3$ and $check_4$ hold.
3. if $check_3$ holds, then $check_1$ and $check_4$ hold.
4. if $check_4$ holds, then $check_1$ and $check_3$ hold.

**Proof of Dependencies between the four interpolants checks**

We will prove the first two dependencies and the others hold analogously. Assume that $\nu_{init} \wedge \bigwedge_{i=1}^{k} \phi_i \wedge \vec{\psi}_i$ is $p$, $\nu_{init} \wedge \bigwedge_{i=1}^{k+2} \phi_i \wedge \vec{\psi}_i$ is $s$, and $\phi_{k+1} \wedge \vec{\psi}_{k+1}$ is $c$. According to computed interpolants that follows lazy abstraction technique, we have the following valid relations

- $p \rightarrow \mathcal{I}_k$ and $\mathcal{I}_k \rightarrow \neg(c \wedge s)$.
- $p \wedge c \rightarrow \mathcal{I}_{k+1}$ and $\mathcal{I}_{k+1} \rightarrow \neg s$.
- $(\neg p \vee \neg c \vee \neg s)$ and $p$.

**if $check_1$ holds, then $check_3$ and $check_4$ hold.** if $check_1$ holds, then we get $\mathcal{I}_k \wedge c \rightarrow \mathcal{I}_{k+1}$. We know that as $p$ is a tautology and $p \rightarrow \mathcal{I}_k$, then $I_k$ is valid. Thus

- $(\mathcal{I}_k \vee \neg c \vee \neg \mathcal{I}_{k+1})$ is a tautology. Thus, $check_3$ holds.
- $(\mathcal{I}_k \vee \neg c \vee \mathcal{I}_{k+1})$ is a tautology. Thus, $check_4$ holds.

**if $check_2$ holds, then $check_1$, $check_3$ and $check_4$ hold.** if $check_2$ holds, then we get $\mathcal{I}_k \wedge c \rightarrow \neg \mathcal{I}_{k+1}$. We know that as $p$ is a tautology and $p \rightarrow \mathcal{I}_k$, then $I_k$ is valid. Thus

- $(\mathcal{I}_k \vee \neg c \vee \neg \mathcal{I}_{k+1})$ is a tautology. Thus, $check_3$ holds.
- $(\mathcal{I}_k \vee \neg c \vee \mathcal{I}_{k+1})$ is a tautology. Thus, $check_4$ holds.

Additionally, as $\mathcal{I}_k$ is valid and $\mathcal{I}_k \rightarrow \neg \mathcal{I}_{k+1} \vee \neg c$ hold, then we get that $\neg \mathcal{I}_{k+1} \vee \neg c$ is a tautology. Thus $(\neg \mathcal{I}_k \vee \neg \mathcal{I}_{k+1} \vee \neg c)$ is valid as well. It follows that $check_1$ holds. The other proofs hold analogously.

As aforementioned, $\mathcal{I}_k$ and $\mathcal{I}_{k+1}$ are interpolants generated from the whole unsatisfiable path formula, before and after traversing $e_k$. In order to refine the abstraction, we introduced four checks to find out which relation holds so that we can eliminate the spuriousness of current erroneous counterexample. But it is necessary at this point to validate the claim whether these checks are sufficient to eliminate the spurious counterexample. Should this not be the case, we explain when these checks fail and succeed.

Strictly speaking, in general, there are uncommon situations, where none of these checks hold, which means that no side-conditions will be attached to some edges. Consequently, no progress is guaranteed and our approach will fail. The following example elaborates this case.[5]

**Example 4.11: Weakness of four checks**

Consider that we have a control flow automaton as in Figure 4.14, where a first spurious counterexample is marked in Image 2. The following formula:

$$\underbrace{x_0 = 0 \wedge y_0 = 1}_{init} \wedge \underbrace{x_1 = x_0 + 2 \wedge y_1 = y_0 + x_0 - 1}_{tr} \wedge \underbrace{y_2 = y_1 \wedge x_2 = x_1 \wedge y_2 = 4 \wedge x_2 = 3}_{result}$$

represents the unsatisfiable path formula of the marked spurious counterexample. At the first step, we compute $\mathcal{I}_0$ for the pair $(init, tr \wedge result)$ which is nothing but

[5] This artifical example is created for a purpose of addressing the problem of independent interpolants, although it did not appear in any benchmark seen by the author of this thesis.

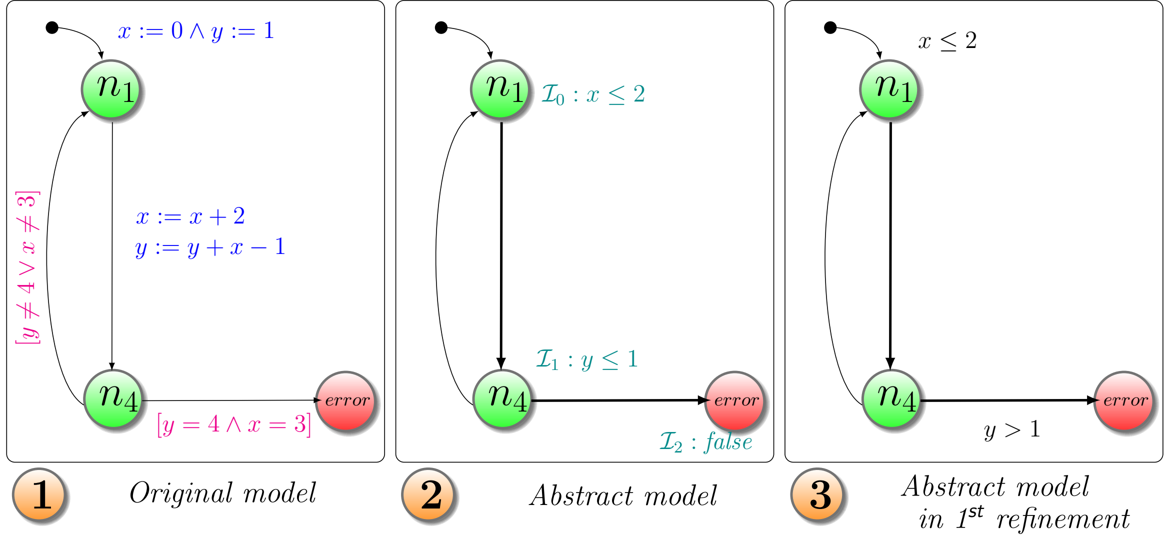



Figure 4.14: An example of useless refinement since none of four checks holds. Image 2 represents the abstract model with the marked spurious counterexample and computed interpolants. Image 3 represents the abstraction after first refinement, where none of checks holds between $\mathcal{I}_0$ and $\mathcal{I}_1$.

$x \leq 2$. Then, when we compute $\mathcal{I}_1$ for the pair ($init \wedge tr, result$), we get $y \leq 1$. Now if we try to find which one(s) of the four checks do hold, unfortunately none of them would hold, since we obtained two valid independent interpolants. At this point, no refinement is applied for this edge as in Figure 4.14 – Image 3 – and the current counterexample is still present forever. This may happen due to several reasons:

- different reasons of unsatisfiability could be extracted from a spurious counterexample which may lead to compute uncorrelated consecutive interpolants.
- using different (deduce and decide) heuristics at each step of computing interpolants which may affect the shape of interpolants (slackness).

To address a solution (theoretically and practically) for the previous problem, we propose the following steps. *First*, using inductive interpolants will avert this problem completely, since using inductive interpolants guarantees the correlation between consecutive interpolants such that the predecessor interpolant of an edge implies the successor interpolant (first check always holds). The following lemma proves that.

**Lemma 4.4: Inductive interpolants eliminate spurious CEXS**

Given a control flow graph $\gamma \in \Gamma$, its abstraction $\alpha(\gamma)$ and a spurious counterexample $\sigma_{sp} \in \Sigma(\alpha(\gamma))$ over the sequence of edges $e_1, ...e_m$, adding side-conditions is sufficient to eliminate the spurious counterexample.

**PROOF OF INDUCTIVE INTERPOLANTS ELIMINATE SPURIOUS CEXS**

By using stepwise inductive interpolants, we get a sequence of interpolants $\mathcal{I}_0, ..., \mathcal{I}_m$ attributing the previous (spurious) abstract counterexample with the path condition $\bigwedge_{i=0}^{m-1}(\mathcal{I}_i \rightarrow \mathcal{I}_{i+1})$, where "$\mathcal{I}_i \rightarrow \mathcal{I}_{i+1}$" is obtained from Check 1, since $\mathcal{I}_i \wedge \phi_{i+1} \wedge \vec{\psi}_{i+1} \rightarrow \mathcal{I}_{i+1}$ is a tautology. As the first and – at least – the last interpolants are true and false respectively, the path formula $(\bigwedge_{i=0}^{m-1} \mathcal{I}_i \rightarrow \mathcal{I}_{i+1})$ becomes contradictory. Thus the added side-conditions eliminate the current spurious counterexample.

*Second*, we can mostly guarantee that one of four checks hold by forcing the solver to deduce and decide on literals from the $A$ part (see Subsection 4.3.3). This often enables us to generate dependent consecutive interpolants. *Third*, one can also generate consecutive interpolants by solving the unsatisfiable path formula incrementally such that if the unsatisfiability of a formula would be justified by several reasons, the latter incremental approach assures that we justify the unsatisfiability of the formula by only one reason, namely the first discovered one. Thereby, the interpolants are not independent.

Due to the predicates' implicational pre-post-style, we can simply conjoin all discovered predicates at an edge, regardless on which path and after how many refinement steps they are discovered. Such incremental refinement of the symbolically represented pre-post-relation attached to edges by means of successively conjoining new cases proceeds until finally we can prove the safety of the model by proving that the bad state is disconnected from all reachable states of the abstract model, or until an eventual counterexample gets real in the sense of its concretization succeeding. To prove unreachability of a node in the new abstraction, we use Craig interpolation for computing a safe overapproximation of the reachable state space as proposed by McMillan [McM03]. The computation of the overapproximating CI exploits the pre-post conditions added.

### 4.4.2 Example

**CONTINUE WITH EXAMPLE 4.1**

In the following, we illustrate how the program in Figure 4.1 is proven to be safe; i.e., that location error is unreachable. The arithmetic program, the corresponding control flow graph, and the encoding of the control flow graph in iSAT3 are aforestated in the Figure 4.1.

**First iteration.** we get the initial coarse abstraction according to Definition 4.11. The first spurious counterexample as in Figure 4.14b has the following syntax:

$$\sigma_{sp} : \langle n_1, \nu_1 \rangle \rightarrow \langle n_3, \nu_3 \rangle \rightarrow \langle n_4, \nu_4 \rangle \rightarrow \langle n_5, \nu_5 \rangle \rightarrow \langle error, \nu_e \rangle$$

While concretizing the first counterexample, we found it spurious, thus we need to refine the abstraction model as follows:

- the path formula is: $y_0 = 0 \;\wedge\; x_1 = sin(y_0) + 1.0002 \;\wedge\; y_1 = y_0 \;\wedge\; x_2 = x_1 + x_1 \;\wedge\; y_2 = y_1 \wedge x_3 = 0 \wedge x_3 = x_2 \wedge y_3 = y_2 \models \perp$.
- for the first edge from $n_1$ to $n_3$, we consider that $y = 0$ is the $A$-formula and the rest is the $B$-formula. In this case, we get *true* as valid interpolant added

to this edge (first, third and fourth check hold, cf. Corollary 4.1).

- for the second edge from $n_3$ to $n_4$, we consider that $true \wedge x_1 = sin(y_0) + 1.0002 \wedge y_1 = y_0$ is the $A$-formula and the rest is the $B$-formula. In this case, we get $x' \geq 0.0002$ as valid interpolant added to this edge (first, third and fourth check hold, cf. Corollary 4.1).
- for the third edge from $n_4$ to $n_5$, we consider that $x_2 \geq 0.0002 \wedge x_2 = x_1 + x_1 \wedge y_2 = y_1$ is the $A$-formula and the rest is the $B$-formula. In this case, we get $x' \geq 0.0002 \rightarrow x' > 0$ as valid interpolant added to this edge (first check holds).
- for the last edge from $n_5$ to $error$, we consider that $x_3 \geq 0.0002 \rightarrow x_3 > 0$ is the $A$-formula and $x_3 = 0 \wedge x_3 = x_2 \wedge y_3 = y_2$ is the $B$-formula. In this case, we get $x < 0$ as a valid interpolant added to this edge (first and second checks hold).

**From second till fourth iterations.** we continue as before, where we get another three spurious counterexamples depicted in Figures 4.14c, 4.13d and 4.13e respectively. After that, the solver proves that the error is not reachable in the abstract model.

**Necessity of third and fourth counterexamples.** additionally, the third and fourth counterexamples have a common suffix, but differ in the prefix formula, therefore both are needed for refining the abstraction in the third and fourth iterations. However, as all following paths from loop unwindings share the prefix formula with the latter two counterexamples, yet have stronger suffix formulas, the already added pre-post predicates are sufficient to eliminate them. Consequently, there is no need for more refinements.

### 4.4.3 Case studies

We have implemented our approach, in particular the control flow graph encoding and the interpolation-based CEGAR verification, within the iSAT3 solver. We verified reachability in several linear and non-linear arithmetic programs and CFG encodings of hybrid systems. In all of these benchmarks, the encoding in iSAT3 format as shown in Section 4.3.5 is done manually. The following tests are mostly C-programs modified from [Din13] or hybrid models discussed in [KB11] and [ACH+95]. As automatic translation into CFG format is not yet implemented, the C benchmarks are currently mostly of moderate size (as encoding of problems is done manually, but later large test cases will be under investigation), but challenging; e.g., Hénon map and logistic map [KB11].

We compared our approach with interpolation-based model checking implemented in both CPAchecker [BHT07] (IMPACT configuration [McM06]), version 1.6.1, and iSAT3,[6] where the interpolants are used as overapproximations of reachable state sets [KB11]. Also, we

[6] Although we contacted the authors of *dReal* [GKC13] which supports unbounded model checking for non-linear constraints [GZ16], they referred us to the latest version which does not support unbounded model checking, thus it is excluded from this comparison.

compared with CBMC [CKL04] as it can verify linear and polynomial arithmetic floating-point dominated C-programs. Comparison on programs involving transcendental functions could, however, only be performed with interpolant-based model checking in iSAT3 as CBMC does not support these functions and CPAchecker treats them as uninterpreted functions.

| Program features / Approach | | | | | | iSAT3 CEGAR, lazy abstraction | | | | iSAT3 Interpolation-based MC | | | | CBMC maximum depth 250 | | | | CPAchecker ITP + lazy abstraction | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| No | Name | Non-linear | Loops | #Nodes | #Edges | Time(s) | Memory(KB) | Iteration | Result | Time(s) | Memory(KB) | Depth | Result | Time(s) | Memory(KB) | unwinding loop | Result | Time(s) | Memory(KB) | #Refinements | Result |
| 1 | cfa_test0001 [Din13] | ✗ | ✓ | 11 | 13 | **1.962** | **17256** | **14** | **SAFE** | TO | 6038428 | 20 | UNKNOWN | 98.396 | 151028 | 56 | SAFE | 2.782 | 150984 | 2 | SAFE |
| 2 | cfa_test0002 [Din13] | ✗ | ✓ | 11 | 13 | **0.173** | **6352** | **5** | **SAFE** | TO | 168240 | 801 | UNKNOWN | 9.406 | 141152 | 56 | SAFE | 2.242 | 143948 | 1 | SAFE |
| 3 | cfa_test0003 [Din13] | ✗ | ✓ | 11 | 13 | **0.127** | **5716** | **5** | **SAFE** | TO | 169072 | 800 | UNKNOWN | 8.160 | 140996 | 56 | SAFE | 2.202 | 128216 | 1 | SAFE |
| 4 | cfa_test0004 [Din13] | ✗ | ✓ | 11 | 13 | **0.174** | **6568** | **8** | **UNSAFE** | 55.653 | 5883156 | 15 | UNSAFE | 3.801 | 140936 | 56 | UNSAFE | 2.818 | 149652 | 2 | UNSAFE |
| 5 | cfa_test0005 [Din13] | ✓ | ✓ | 15 | 18 | 0.455 | 8812 | 9 | CAND. | 1.657 | 30840 | 16 | CAND. | **0.150** | **22972** | **6** | **UNSAFE** | 3.690 | 158588 | 3 | UNSAFE |
| 6 | cfa_test0006 [Din13] | ✓ | ✗ | 13 | 17 | **0.043** | **5196** | **2** | **SAFE** | 0.070 | 6500 | 7 | SAFE | 0.137 | 22320 | 0 | SAFE | 2.561 | 141876 | 2 | SAFE |
| 7 | cfa_test0007 [Din13] | ✓ | ✗ | 7 | 8 | **0.047** | **4856** | **2** | **SAFE** | TO | 4541444 | 3 | UNKNOWN | unsupported functions | | | | 2.424 | 127384 | 1 | UNSAFE |
| 8 | cfa_test0008 [Din13] | ✗ | ✗ | 3 | 3 | **0.017** | **4180** | **1** | **UNSAFE** | 0.023 | 4112 | 1 | UNSAFE | 0.140 | 22600 | 0 | UNSAFE | 2.456 | 145936 | 2 | UNSAFE |
| 9 | cfa_test0009 [DHKT12] | ✗ | ✓ | 6 | 8 | **0.054** | **5048** | **3** | **SAFE** | TO | 157748 | 864 | UNKNOWN | 7.702 | 50248 | 56 | SAFE | 2.510 | 145684 | 1 | SAFE |
| 10 | cfa_test0010 | ✓ | ✓ | 6 | 8 | **0.075** | **5268** | **3** | **SAFE** | TO | 775032 | 2 | UNKNOWN | unsupported functions | | | | 2.229 | 128328 | 1 | UNSAFE |
| 11 | control flow [KB11] | ✗ | ✗ | 7 | 8 | **0.035** | **4904** | **1** | **SAFE** | 0.039 | 4716 | 5 | SAFE | 0.303 | 26820 | 116 | SAFE | 2.330 | 127968 | 1 | SAFE |
| 12 | cruise control [KB11] | ✓ | ✓ | 8 | 15 | **0.103** | **5724** | **8** | **SAFE** | TO | 3196492 | 130 | UNKNOWN | 0.147 | 22528 | 18 | SAFE | 2.819 | 146284 | 3 | UNSAFE |
| 13 | frontier_01 [Kup13] | ✓ | ✓ | 3 | 4 | **0.050** | **4744** | **1** | **CAND.** | 0.056 | 4824 | 2 | CAND. | 3.367 | 101844 | 32 | UNSAFE | 2.650 | 145524 | 2 | UNSAFE |
| 14 | frontier_02 [Kup13] | ✓ | ✓ | 3 | 4 | **0.173** | **6148** | **3** | **SAFE** | TO | 94200 | 721 | UNKNOWN | 1.276 | 102124 | 32 | SAFE | 3.046 | 149128 | 2 | UNSAFE |
| 15 | frontier_03 [Kup13] | ✓ | ✓ | 3 | 4 | **0.141** | **5332** | **3** | **SAFE** | TO | 97580 | 796 | UNKNOWN | 1.284 | 102056 | 32 | SAFE | 2.868 | 148964 | 2 | UNSAFE |
| 16 | hénon map [KB11] | ✓ | ✓ | 3 | 4 | **0.033** | **4628** | **3** | **CAND.** | 0.041 | 4628 | 3 | CAND. | 0.694 | 24496 | 25 | SAFE | 2.216 | 129908 | 1 | UNSAFE |
| 17 | logistic map [KB11] | ✓ | ✓ | 3 | 4 | **0.149** | **7380** | **3** | **SAFE** | 0.205 | 5932 | 12 | CAND | 4.759 | 188928 | 38 | SAFE | 2.142 | 124620 | 1 | UNSAFE |
| 18 | two circles_01 | ✓ | ✓ | 6 | 7 | **0.067** | **4608** | **1** | **SAFE** | 48.938 | 13848 | 8 | SAFE | 0.163 | 22452 | 54 | SAFE | 2.359 | 145832 | 1 | UNSAFE |
| 19 | two circles_02 | ✓ | ✓ | 6 | 7 | **0.033** | **4584** | **1** | SAFE | 0.144 | 5260 | 8 | SAFE | 0.155 | 22204 | 55 | SAFE | 2.383 | 145204 | 1 | UNSAFE |
| 20 | tank_controller [ACH+95] | ✗ | ✓ | 5 | 13 | 7.822 | 159708 | 24 | SAFE | **0.107** | **6784** | **20** | **SAFE** | 0.149 | 22344 | 69 | SAFE | 2.446 | 143688 | 1 | UNSAFE |
| 21 | gas_burner [ACH+95] | ✗ | ✓ | 4 | 8 | 3.776 | 12720 | 42 | SAFE | **0.361** | **7260** | **33** | **SAFE** | 27.843 | 43580 | 1282 | SAFE | 4.511 | 151460 | 3 | UNSAFE |
| 22 | cfa_test0022 [Seg10] | ✗ | ✓ | 8 | 19 | 2.264 | 18468 | 23 | SAFE | 0.845 | 126680 | 6 | SAFE | **0.151** | **22312** | **25** | **SAFE** | 5.358 | 202840 | 2 | SAFE |
| 23 | cfa_test0023 [Seg10] | ✗ | ✓ | 3 | 4 | 8.189 | 42620 | 21 | SAFE | 0.143 | 6716 | 22 | SAFE | **0.264** | **26880** | **56** | **SAFE** | 3.025 | 145360 | 2 | UNSAFE |

Table 4.1: Verification results of linear/non-linear hybrid models. Bold lines refer to best results w.r.t. best verification time. Red lines refer to false alarms reported by the solver and blue lines refer to inability to solve the problem due to unsupported functions.

CBMC, version 4.9, was used in its native bounded model-checking mode with an adequate unwinding depth, which represents a logically simpler problem, as the $k$-inductor [DHKR11] built on top of CBMC requires different parameters to be given in advance for each benchmark, in particular for loops, such that it offers a different level of automation. We limited solving time for each problem to five minutes and memory to 4 GB. The benchmarks were run on an Intel(R) Core(TM) i7 M 620@2.67GHz with 8 GB RAM.

**Comparison between different tools.** Table 4.1 summaries the results of our experimental evaluation. It comprises five groups of columns. The first includes the name of the benchmark, type of the problem (whether it includes non-linear constraints or loops), number of control points, and number of edges. The second group shows the result of verifying the benchmarks when using iSAT3 CEGAR (lazy abstraction), thereby stating the verification time in seconds, memory usage in kilobytes, number of abstraction refinements, and the final verdict. The third group has the same structure, yet reports results for using iSAT3 with interpolation-based reach-set overapproximation used for model checking. The fourth part provides figures for CBMC with a maximum unwinding depth of 250. CBMC could not address the benchmarks 7 and 10 as they contain unsupported transcendental functions. The fifth part provides the figures for CPAchecker while using the default IMPACT configuration where the red lines refer to false alarms (*for comparison, CPAchecker was run with different configurations, yet this didn't affect the presence of false alarms.*) reported by IMPACT due to non-linearity or non-deterministic behaviour of the program.

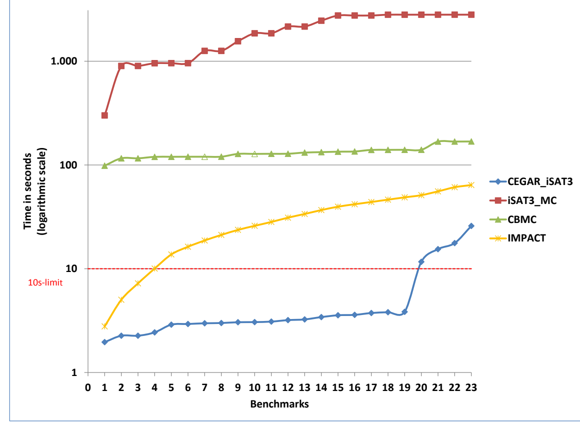



Figure 4.15: Accumulated verification times for the first $n$ benchmarks.

For each benchmark, we mark in **boldface** the best results in terms of time. iSAT3-based CEGAR outperforms the others in 18 cases, interpolation-based MC in iSAT3 outperforms the others in 2 cases, and CBMC outperforms the others in 3 cases. Figures 4.15 and 4.16 summarize the main findings. The tests demonstrate the efficacy of the new CEGAR approach in comparison to other competitor tools. Concerning verification time, we observe that iSAT3 with CEGAR scores the best results. Namely, iSAT3-based CEGAR needs about 27 s for processing the full set of benchmarks, equivalent to an average verification time of 1.2 s, iSAT3 with the interpolation-based approach needs 2809 s total and 122 s on average, CBMC needs 168 s total and 8 s on average, and IMPACT needs 64 s total and 2.7 s on average.

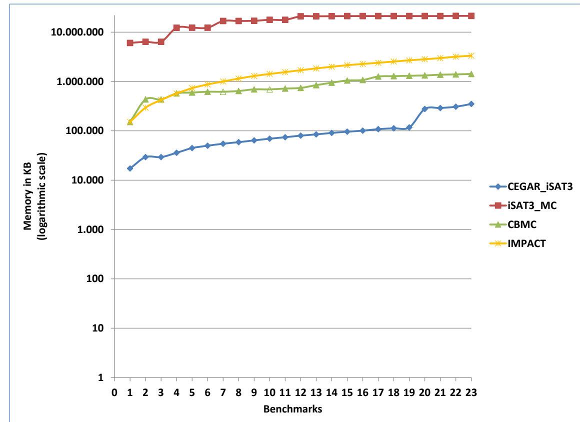



Figure 4.16: Memory usage (#benchmarks processed within given memory limit).

Figure 4.15 shows the results in logarithmic scale where the plot refers to the accumulated verification times for 23 benchmarks. Concerning memory, we observe that iSAT3 with CEGAR needs about 15 MB on average, iSAT3 with interpolation 906 MB on average, CBMC needs 66 MB on average, and IMPACT needs 141 MB on average. The findings confirm that at least on the current set of benchmarks, the CEGAR approach is by a fair margin the most efficient one. Also, Figure 4.16 shows the results in logarithmic scale where the plot refers to the accumulated memory usages for 23 benchmarks.

**Comparison between different abstraction technique within iSAT3.** In addition to previous interesting results, we would like to shed some lights on the effect of different abstraction techniques while using iSAT3 together with CEGAR technique. Namely, we have lazy abstraction while using inductive interpolants which is used in the latter table. Additionally, we have the pure lazy abstraction that performs the four relations checks individually[7]. The last refinement technique is the entire or complete one, which refines the whole model after extracting each predicate. For each benchmark, we mark in boldface the best results in terms of time.

[7] In case of using inductive interpolants only three checks are needed; namely the second, third and fourth ones. Since the inductive relation guarantees the validity of the first check.

| Program features \ Approach | | | | | | iSAT3 CEGAR, inductive interpolants | | | | iSAT3 CEGAR, pure lazy abstraction | | | | iSAT3 CEGAR, entire refinement | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| No | Name | Non-linear | Loops | #Nodes | #Edges | Time(s) | Memory(KB) | Iteration | Result | Time(s) | Memory(KB) | Iteration | Result | Time(s) | Memory(KB) | Iteration | Result |
| 1 | cfa_test0001 [Din13] | ✗ | ✓ | 11 | 13 | **1.962** | **17256** | **14** | **SAFE** | 2.356 | 17764 | 14 | SAFE | 2.917 | 30768 | 8 | SAFE |
| 2 | cfa_test0002 [Din13] | ✗ | ✓ | 11 | 13 | 0.173 | 6352 | 5 | SAFE | 0.200 | 6500 | 5 | SAFE | **0.074** | **5520** | **1** | **SAFE** |
| 3 | cfa_test0003 [Din13] | ✗ | ✓ | 11 | 13 | 0.127 | 5716 | 5 | SAFE | 0.162 | 5588 | 5 | SAFE | **0.075** | **5292** | **1** | **SAFE** |
| 4 | cfa_test0004 [Din13] | ✗ | ✓ | 11 | 13 | **0.174** | **6568** | **8** | **UNSAFE** | 0.243 | 6620 | 8 | UNSAFE | 0.383 | 6868 | 6 | UNSAFE |
| 5 | cfa_test0005 [Din13] | ✓ | ✓ | 15 | 18 | **0.455** | **8812** | **9** | **CAND.** | 0.643 | 8408 | 9 | CAND. | 1.787 | 9144 | 8 | CAND. |
| 6 | cfa_test0006 [Din13] | ✓ | ✗ | 13 | 17 | **0.043** | **5196** | **2** | **SAFE** | 0.051 | 5212 | 2 | SAFE | 0.118 | 5068 | 1 | SAFE |
| 7 | cfa_test0007 [Din13] | ✓ | ✗ | 7 | 8 | **0.047** | **4856** | **2** | **SAFE** | 0.048 | 4744 | 2 | SAFE | 0.096 | 4816 | 2 | SAFE |
| 8 | cfa_test0008 [Din13] | ✗ | ✗ | 3 | 3 | 0.017 | 4180 | 1 | UNSAFE | **0.015** | **4172** | **1** | **UNSAFE** | 0.017 | 4252 | 1 | UNSAFE |
| 9 | cfa_test0009 [DHKT12] | ✗ | ✓ | 6 | 8 | **0.054** | **5048** | **3** | **SAFE** | 0.063 | 5224 | 3 | SAFE | 0.068 | 4932 | 2 | SAFE |
| 10 | cfa_test0010 | ✓ | ✓ | 6 | 8 | **0.075** | **5268** | **3** | **SAFE** | 0.152 | 5500 | 5 | SAFE | 0.296 | 5300 | 3 | SAFE |
| 11 | control flow [KB11] | ✗ | ✗ | 7 | 8 | **0.035** | **4904** | **1** | **SAFE** | 0.038 | 4756 | 1 | SAFE | 0.039 | 4480 | 1 | SAFE |
| 12 | cruise control [KB11] | ✓ | ✓ | 8 | 15 | **0.103** | **5724** | **8** | **SAFE** | 0.115 | 5584 | 8 | SAFE | 0.126 | 5072 | 2 | SAFE |
| 13 | frontier_01 [Kup13] | ✓ | ✓ | 3 | 4 | **0.050** | **4744** | **1** | **CAND.** | 0.051 | 4468 | 1 | CAND. | 0.054 | 4376 | 1 | CAND. |
| 14 | frontier_02 [Kup13] | ✓ | ✓ | 3 | 4 | **0.173** | **6148** | **3** | **SAFE** | 0.262 | 7116 | 3 | SAFE | 0.301 | 7064 | 3 | SAFE |
| 15 | frontier_03 [Kup13] | ✓ | ✓ | 3 | 4 | **0.141** | **5332** | **3** | **SAFE** | 0.168 | 5352 | 3 | SAFE | 0.185 | 5288 | 3 | SAFE |
| 16 | hénon map [KB11] | ✓ | ✓ | 3 | 4 | **0.033** | **4628** | **3** | **CAND.** | 0.036 | 4372 | 3 | CAND. | 0.048 | 4652 | 3 | CAND. |
| 17 | logistic map [KB11] | ✓ | ✓ | 3 | 4 | 0.149 | 7380 | 3 | SAFE | **0.055** | **4360** | **2** | **SAFE** | 0.063 | 4704 | 2 | SAFE |
| 18 | two circles_01 | ✓ | ✓ | 6 | 7 | **0.067** | **4608** | **1** | **SAFE** | 0.089 | 4640 | 1 | SAFE | 0.150 | 4632 | 1 | SAFE |
| 19 | two circles_02 | ✓ | ✓ | 6 | 7 | **0.033** | **4584** | **1** | SAFE | 0.038 | 4520 | 1 | SAFE | 0.633 | 7768 | 1 | SAFE |
| 20 | tank_controller [ACH+95] | ✗ | ✓ | 5 | 13 | **7.822** | **159708** | **24** | **SAFE** | 8.468 | 160048 | 24 | SAFE | 5.662 | 85592 | 18 | SAFE |
| 21 | gas_burner [ACH+95] | ✗ | ✓ | 4 | 8 | **3.776** | **12720** | **42** | **SAFE** | 5.786 | 12552 | 42 | SAFE | 7.314 | 17728 | 35 | SAFE |
| 22 | cfa_test0022 [Seg10] | ✗ | ✓ | 8 | 19 | 2.264 | 18468 | 23 | SAFE | 2.626 | 17988 | 23 | SAFE | **0.169** | **5384** | **2** | **SAFE** |
| 23 | cfa_test0023 [Seg10] | ✗ | ✓ | 3 | 4 | **8.189** | **42620** | **21** | **SAFE** | 11.813 | 43132 | 21 | SAFE | 17.304 | 75544 | 21 | SAFE |

Table 4.2: Verification results of (non)-linear hybrid models while comparing abstraction techniques. Bold lines refer to best results w.r.t. best verification time.

Table 4.2 summaries the results of our experimental evaluation. It comprises four groups of columns. The first two groups have exactly the same description and information of Table 4.1. The third group has the same structure, yet reports results for using iSAT3 with CEGAR however by using pure lazy abstraction technique [HJMS02]. Also, the fourth group has the same structure, yet reports results for using iSAT3 with CEGAR however by using entire refinement. Figures 4.17 and 4.18 summarize the main findings.

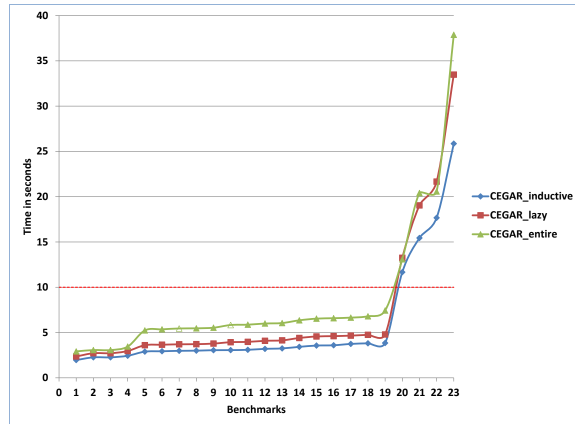



Figure 4.17: Accumulated verification times for the first $n$ benchmarks.

Concerning verification time, we observe that CEGAR with inductive interpolants scores the best results. Namely, it needs about 27 s for processing the full set of benchmarks, equivalent to an average verification time of 1.2 s, pure lazy abstraction needs 34 s total and 1.4 s on average, and entire refinement needs 39 s total and 1.69 s on average.

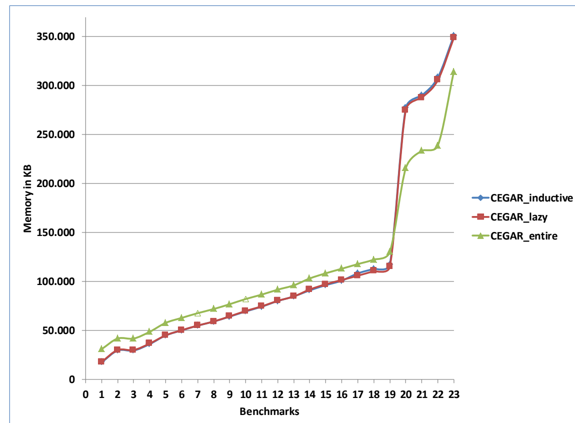



Figure 4.18: Memory usage (#benchmarks processed within given memory limit).

Figure 4.17 shows the results in logarithmic scale where the plot refers to the accumulated verification times for 23

benchmarks. Concerning memory, we observe that CEGAR with inductive interpolants needs about 15 MB on average, pure lazy abstraction needs 14.8 MB on average, and entire refinement needs 13.3 MB on average. The findings confirm that at least on the current set of benchmarks, the CEGAR approach with inductive interpolants is the most efficient one. Also, Figure 4.18 shows the results in logarithmic scale where the plot refers to the accumulated memory usages for 23 benchmarks.

**Discussion.** The only weakness of both iSAT3-based approaches is reporting sometimes a *candidate solution*, i.e., a very narrow interval box that is hull consistent, rather than a firm satisfiability verdict. This effect is due to the incompleteness of interval reasoning, which here is employed in its outward rounding variant providing safe overapproximation of real arithmetic rather than floating-point arithmetic. It is expected that these deficiencies vanish once floating-point support (following IEEE 754) in iSAT3 is fully implemented, which will be discussed in the next section as an alternative theory to real arithmetic. It should, however, be noted that CEGAR with its preoccupation to generating conjunctive constraint systems (the path conditions) already alleviates most of the incompleteness, which arises particularly upon disjunctive reasoning.

Also, we observe that CEGAR with inductive interpolants achieves the best results with comparison to other techniques due to less checks needed by this approach on one hand. On the other hand, no guarantees can be given always about the best abstraction technique, since some times the entire-refinement reports best results as in Benchmarks 2, 3 and 22 as shown in Table 4.2.

## 4.5 Handling floating points dominated C-programs – experiments in industrial-scale

### 4.5.1 Floating point arithmetic due to IEEE 754

In order to represent real-valued quantities in scientific and technical computing, including hardware, a trade-off between range and precision must be accepted; at this moment floating-point numbers enter the scene. They are still – since 1940's – the method of choice for representing real-valued quantities in scientific and technical computing. By floating point numbers; i.e, $\textit{significand} \times \textit{base}^{\textit{exponent}}$ a real number is, in general, represented approximately through a fixed number of *significant digits* and scaled using *an exponent* in some fixed *base*; the base for the scaling is normally *two*, *eight*, *ten*, or *sixteen*. Over the years, a variety of floating-point representations have been used in computers. However, since the 1990's, the most commonly encountered representation is the one defined by the IEEE 754 Standard [IEE85] where the latter is followed by almost all modern machines. First remarkable feature of IEEE 754 is that it provides for many closely related formats, differing in only in precisions details; e.g. single, double, double extended and quadratic precisions. Second feature specifies some special values, and their representations; e.g.,

- signed zeros: a negative zero $-0$ distinct from ordinary (positive) zero $+0$. The two values behave as equal in numerical comparisons, but some operations return different results for $+0$ and $-0$.

- subnormals: it fills the underflow gap around zero in floating-point arithmetic, otherwise one cannot represent zero. Any non-zero number with magnitude smaller than the smallest normal number is considered as "subnormal".
- infinities: positive infinity $+\infty$ and negative infinity $-\infty$, they are often used as replacement values when there is an overflow. One can see them as an exact result upon dividing-by-zero exception.
- not a number i.e. (NaNs): it will be returned as the result of certain invalid operations, such as $0/0$, $\infty \times 0$, or $\sqrt{(-1)}$.

These special representation affects the comparison of floating-point numbers in IEEE 754. For example, negative and positive zero compare equal, and every NaN compares unequal to any value, including itself. All values except NaN are strictly smaller than $+\infty$ and strictly greater than $-\infty$. Finite floating-point numbers are ordered in the same way akin to their values.

### 4.5.2 Floating points in iSAT3

Scheibler et al. [SNM+16b, SNM+16a] succeeds to define an accurate arithmetic reasoning procedure in iSAT3 by extending the ICP arithmetic reasoner with IEEE 754 standard, including signed zeros, normals, subnormals, infinities and NaNs with radix 2 as in [IEE85]. This extension supports all kinds of type casts and bitwise operations usually encountered in imperative programs. In addition to that, it permits to reason about machine data types as well as real numbers, as necessary for the analysis of actual embedded control. Already having intervals whose endpoints are represented with floating point numbers as in iSAT3, makes it look forthright to extend iSAT3 in order to allow accurate reasoning over floating point arithmetic.

However, for NaNs, a separate encoding is used where a special NaN-literal for every floating point variable is introduced. This makes handling the NaN-related propagations completely with BCP outside of the ICP-contractors.

### 4.5.3 Floating point arithmetic in iSAT3 with CEGAR

As aforementioned in Subsection 4.3.5, our CEGAR technique is built on top of iSAT3, however in a separate layer. Thus, integrating iSAT3 with its extended ICP that handles floating points according to IEEE 754, requires us to adjust the CEGAR layer in two directions, namely: one has to adjust the abstract syntax graph of control flow graphs (parsing issue), and one has to adjust the CEGAR core accordingly to support all special operations on floating point numbers.

**Remark 4.2: Floating points in iSAT3 and transcendental functions**

At the moment only the four basic operations, namely addition "+", subtraction "−", multiplication "∗" and division "/" have been supported in the iSAT3 implementation of IEEE 754.

| Program features \ Approach | | | | | | iSAT3 CEGAR, inductive interpolants, FP | | | | iSAT3 CEGAR, pure lazy abstraction, FP | | | | iSAT3 CEGAR, entire refinement, FP | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **No** | **Name** | **Non-linear** | **Loops** | **#Nodes** | **#Edges** | **Time(s)** | **Memory(KB)** | **Iteration** | **Result** | **Time(s)** | **Memory(KB)** | **Iteration** | **Result** | **Time(s)** | **Memory(KB)** | **Iteration** | **Result** |
| 1 | cfa_test0001 [Din13] | ✗ | ✓ | 11 | 13 | 8.570 | 29812 | 13 | SAFE | 8.395 | 35968 | 13 | SAFE | **5.230** | **27788** | **9** | **SAFE** |
| 2 | cfa_test0002 [Din13] | ✗ | ✓ | 11 | 13 | 0.847 | 12500 | 7 | SAFE | 1.327 | 13904 | 8 | SAFE | **0.088** | **5940** | **1** | **SAFE** |
| 3 | cfa_test0003 [Din13] | ✗ | ✓ | 11 | 13 | 0.089 | 6020 | 3 | SAFE | 0.112 | 6464 | 3 | SAFE | **0.083** | **6144** | **1** | **SAFE** |
| 4 | cfa_test0004 [Din13] | ✗ | ✓ | 11 | 13 | **0.324** | **10732** | **8** | **UNSAFE** | 0.589 | 10768 | 10 | UNSAFE | 0.625 | 9492 | 6 | UNSAFE |
| 5 | cfa_test0005 [Din13] | ✓ | ✓ | 15 | 18 | **0.953** | **11992** | **9** | **UNSAFE** | 1.363 | 12660 | 9 | UNSAFE | 4.362 | 17988 | 10 | UNSAFE |
| 6 | cfa_test0006 [Din13] | ✓ | ✗ | 13 | 17 | **0.099** | **7416** | **3** | **SAFE** | 0.123 | 6608 | 4 | SAFE | 0.178 | 7284 | 1 | SAFE |
| 7 | cfa_test0007 [Din13] | ✓ | ✗ | 7 | 8 | unsupported functions in IEEE 754 due to use of sin and cos | | | | | | | | | | | |
| 8 | cfa_test0008 [Din13] | ✗ | ✗ | 3 | 3 | **0.016** | **4556** | **1** | **UNSAFE** | 0.018 | 4664 | 1 | UNSAFE | 0.018 | 4564 | 1 | UNSAFE |
| 9 | cfa_test0009 [DHKT12] | ✗ | ✓ | 6 | 8 | 0.099 | 6280 | 4 | SAFE | 0.184 | 6676 | 4 | SAFE | **0.092** | **6072** | **2** | **SAFE** |
| 10 | cfa_test0010 | ✓ | ✓ | 6 | 8 | unsupported functions in IEEE 754 due to use of sin and cos | | | | | | | | | | | |
| 11 | control flow [KB11] | ✗ | ✗ | 7 | 8 | **0.039** | **5148** | **1** | **SAFE** | 0.046 | 5364 | 1 | SAFE | 0.048 | 5100 | 1 | SAFE |
| 12 | cruise control [KB11] | ✓ | ✓ | 8 | 15 | 0.168 | 7436 | 7 | SAFE | 0.168 | 7740 | 7 | SAFE | **0.162** | **6120** | **2** | **SAFE** |
| 13 | frontier_01 [Kup13] | ✓ | ✓ | 3 | 4 | **0.022** | **5172** | **1** | **UNSAFE** | 0.025 | 5196 | 1 | UNSAFE | 0.029 | 5212 | 1 | UNSAFE |
| 14 | frontier_02 [Kup13] | ✓ | ✓ | 3 | 4 | **0.233** | **9364** | **3** | **SAFE** | 0.267 | 9328 | 3 | SAFE | 80.527 | 30524 | 3 | SAFE |
| 15 | frontier_03 [Kup13] | ✓ | ✓ | 3 | 4 | **0.195** | **8948** | **3** | **SAFE** | 0.233 | 8996 | 3 | SAFE | 9.103 | 27840 | 3 | SAFE |
| 16 | hénon map [KB11] | ✓ | ✓ | 3 | 4 | **0.066** | **6536** | **3** | **UNSAFE** | 0.083 | 6560 | 3 | UNSAFE | 0.090 | 6536 | 3 | UNSAFE |
| 17 | logistic map [KB11] | ✓ | ✓ | 3 | 4 | **0.095** | **6840** | **3** | **SAFE** | 0.144 | 7456 | 3 | SAFE | 0.154 | 8272 | 3 | SAFE |
| 18 | two circles_01 | ✓ | ✓ | 6 | 7 | **0.049** | **6000** | **1** | **SAFE** | 0.064 | 5932 | 1 | SAFE | 0.086 | 6036 | 1 | SAFE |
| 19 | two circles_02 | ✓ | ✓ | 6 | 7 | **0.046** | **5668** | **1** | **SAFE** | 0.051 | 5924 | 1 | SAFE | 0.092 | 5824 | 1 | SAFE |
| 20 | tank_controller [ACH+95] | ✗ | ✓ | 5 | 13 | **3.331** | **22928** | **24** | **SAFE** | 4.845 | 23020 | 24 | SAFE | 7.311 | 29480 | 18 | SAFE |
| 21 | gas_burner [ACH+95] | ✗ | ✓ | 4 | 8 | **24.304** | **25692** | **42** | **SAFE** | 43.022 | 25548 | 42 | SAFE | 41.501 | 30540 | 35 | SAFE |
| 22 | cfa_test0022 [Seg10] | ✗ | ✓ | 8 | 19 | 24.746 | 26784 | 42 | SAFE | 42.324 | 27656 | 42 | SAFE | **0.097** | **5576** | **2** | **SAFE** |
| 23 | cfa_test0023 [Seg10] | ✗ | ✓ | 3 | 4 | **0.618** | **13768** | **11** | **SAFE** | 1.274 | 18964 | 11 | SAFE | 1.641 | 20836 | 11 | SAFE |

Table 4.3: Verification results of (non)-linear hybrid models while supporting IEEE 754 standard. **Bold** lines refer to best results w.r.t. best verification time.

While these basic operations are part of the IEEE 754 standard, there is only a recommendation for *sine* and *cosine*. This means x86 control process units (CPUs) will return results for *sine* and *cosine* which do not comply with our implementation of the IEEE standard, since they implement a specific interpretation of the underspecified *sine* and *cosine*.
Other CPUs will probably have the same problem and return other non-conforming values. Therefore, regarding floating point reasoning, one always has to know in advance the concrete target CPU architecture in order to provide the correct deduction routines for the transcendental functions.
Thus, the aforementioned example in Figure 4.1 in spite of its simplicity, cannot be handled here, since this program contains transcendental functions.

Table 4.3 shows the same list of benchmarks which have been verified in Table 4.1, however this time after using iSAT3 with IEEE 754 together with CEGAR procedure. Table 4.3 has the same description as Table 4.2, however shows results obtained using iSAT3 with CEGAR admitting IEEE 754 standard. Figures 4.19 and 4.20 summarize the main findings. Concerning verification time, we observe that CEGAR with inductive interpolants scores the best results. Namely, it needs about 65 s for process-

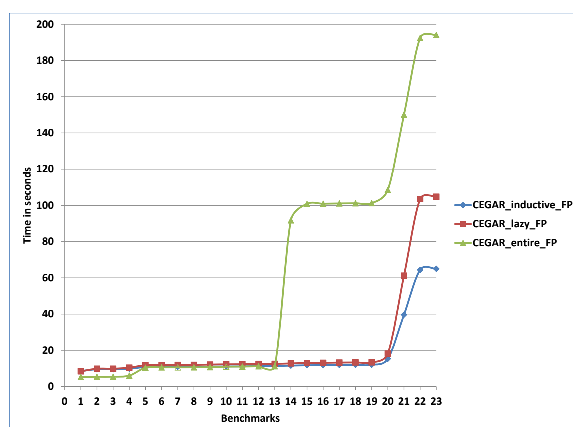



Figure 4.19: Accumulated verification times for the first $n$ benchmarks.

ing the full set of benchmarks, equivalent to an average verification time of 3.1 s, pure lazy abstraction needs 105 s total and 5 s on average, and entire refinement needs 194 s total and 9.2 s on average.

Concerning memory, we observe that CEGAR with inductive interpolants needs about 11.6 MB on average, pure lazy abstraction needs 12.3 MB on average, and entire refinement needs 14.3 MB on average.

Now, if one compares the results obtained in Table 4.2 with the verification results in Table 4.3, one can say:

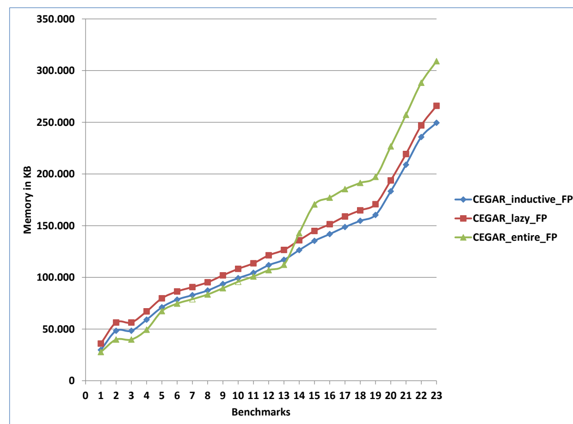



Figure 4.20: Memory usage (#benchmarks processed within given memory limit).

- iSAT3 with CEGAR – in general – needs less verification time with comparison to the same tool when supporting IEEE 754. It might be the case since for each arithmetic variable, iSAT3 introduces five new literals representing the special values; e.g., NaNs.
- iSAT3 with CEGAR – in general – needs more memory with comparison to the same tool when supporting IEEE 754. It might be the case since the former approach may need more iterations to prove safety due to existence of candidate solutions while verifying abstractions which is not the case when the arithmetic reasoner supports IEEE 754.
- although iSAT3 with CEGAR returns weak answers as in Benchmarks 5, 13 and 15, it returns strong answers in Benchmarks 7 and 10 which admit transcendental functions. In contrast to that, iSAT3 with CEGAR which supports IEEE 754, returns always strong answers e.g. SAFE or UNSAFE even in Benchmarks 5, 13 and 15, yet it fails to address a solution for Benchmarks 7 and 10, since implementation of IEEE 754 does not support transcendental functions (cf. Remark 4.2).

### 4.5.4 Industrial case studies

In this section, we will show how iSAT3 with CEGAR technique is used in industrial scope where large models are expected with non-linear behaviour and bitwise operations as well. Figure 4.21 is the flowchart of the transfer project of AVACS project where two industrial partners, namely SICK AG and BTC-ES AG work with academic partners, namely Carl von Ossietzky Universtät Oldenburg and Albert-Ludwigs Universität Freiburg. Figure 4.21 addresses the main steps and challenging problems. We assume that a Simulink model or a C code are given by an industrial partner representing an embedded software system. Moreover, the latter C code will be converted (simplified) to a simpler internal representation language called SMI [WBBL02] which has a well-defined syntax and semantics. The first (practical) task as shown in Figure 4.21 is to convert SMI code which may include loops, switch cases and if-conditions to the iSAT3-CFG language explained

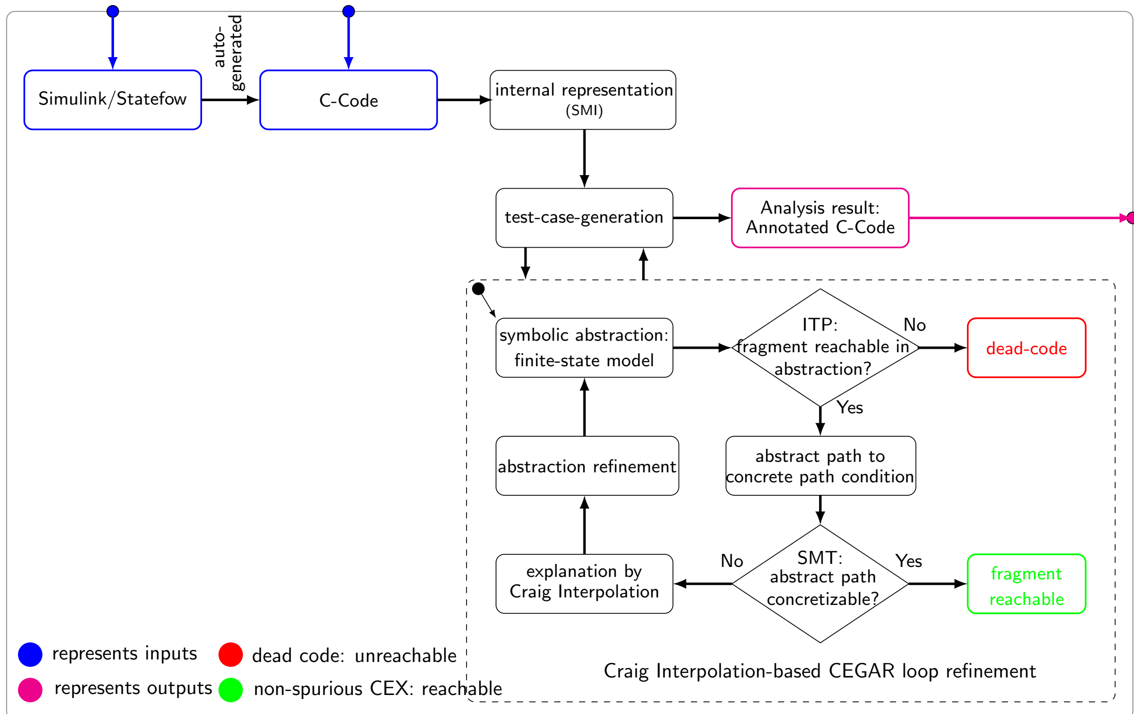



Figure 4.21: State-Chart of resulting analysis by using CI-based CEAGR (adjusted from [FB13]).

in Subsection 4.3.5. In other words, abstract syntax tree (AST) of SMI will be encoded in another scheme, namely control flow automaton, however with the same semantics translation (neither overapproximation nor underapproximation is applied). Thus, CFG will encode all possible paths of the program, even the non-explicit ones. That is, if a switch-case over an arithmetic variable has few conditional states, our CFG has to consider a new program path when none of these variable states are satisfied. The second task is to extend the previous iSAT-CFG approach to handle floating point arithmetic as real numbers where IEEE 754 [IEE85] will be supported as aforenamed in the previous subsection. The third task is to preform CEGAR steps, including initial abstraction as in Definition 4.11, verifying the abstraction, concretizing the discovered counterexamples – if existent – and finally refining the abstraction. CEGAR loop is represented by the large-dashed box in Figure 4.21.

### 4.5.5 Converting SMI code to iSAT3-CFG input language

SMI code consists of two files: the first one is a *symtab*-file which contains all identifiers (variables and constants) declarations with their well-defined domains. These defined domains contain the types, lower and upper bounds, whether the variable is auxiliary one or not, the initial assignments, the assigned domain ... etc. The second one is a *smi*-file which contains the program block.

According to SMI [WBBL02] syntax and semantics which conforms to our work, the SMI program consists of one unconstrained While-block (runs forever). Inside this global While-

block we may have three kinds of expressions:

- normal expressions that assign to the left operand the value of right operand. Right operand can be expression, if-conditional statement, or casting operation, or another variable or a well-defined value.
- case-guard block which has the abbreviation `dcase` that stands for deterministic case distinguishing over a Boolean expression, whereas the cases cannot be intersected. Each case in case-guard block consists of one condition and a sequence of expressions (to be executed) in case the guard block is satisfied.
- while block, where the `while`-loop has a condition that must be fulfilled in order to execute the expressions inside this block.

Informally, the semantics of an SMI program is a transition system in which a transition between two configurations of the program corresponds to a single complete execution of the code part of the SMI program with the values from the source configuration. A set of configurations refers to the Cartesian product between control points of the program and the complete state space of the program comprising all values of all types (cf. [WBBL02]).

Our first task is to convert the following abstract syntax tree to the corresponding control flow graph. Then, we build the corresponding iSAT3-CFG file from the control flow graph.

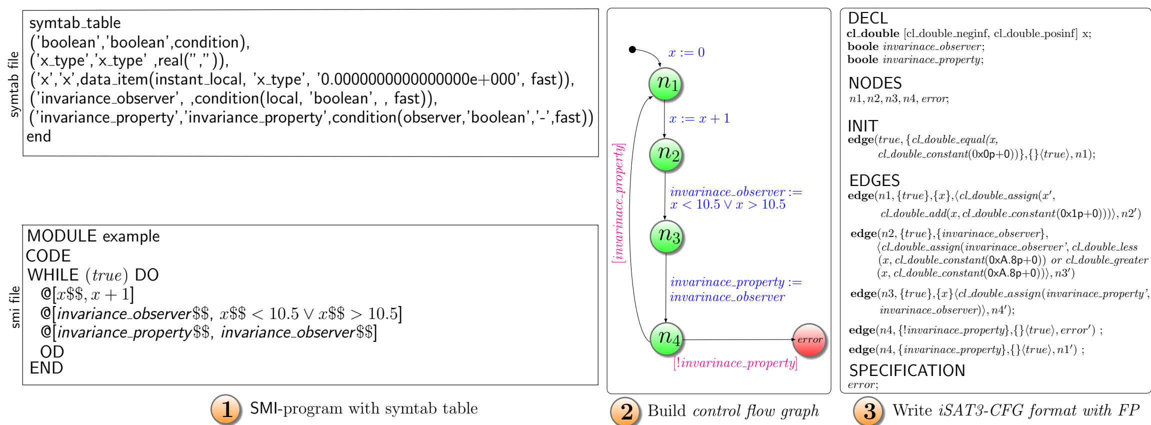



Figure 4.22: Left: An *smi*-program with *symtab*-table, middle: corresponding control flow graph, right: encoding in iSAT3-CFG format with FP new syntax according to [SNM+16a].

### Example 4.12: From SMI to iSAT3-CFG

Figure 4.22 shows a simple SMI-program with its identifiers table, its encoding in control flow graph and the iSAT3-CFG encoding. This program contains one (infinite) while loop where $x$ is incremented by 1 in each iteration. Variables with $$ sign can be considered as primed variables according to iSAT3 semantics, represent the next valuations of the variables after new assigning. Variables without $$ sign refer to the previous valuation of the variable. *invariance_observer* is a Boolean variable that is always `true` as long as $x$ does not equal 10.5. Additionally, the *invariance_property* takes the value of *invariance_observer*. Whenever

*invariance_property* is assigned to *false*, the error location will be reachable. That is, reachability of error location violates the safe invariant. One can encode the same problem with a different way, e.g., all assignments will be encoded in one transition, but all are primed variables. However, it is found that the encoding in Figure 4.22 is more readable and common. iSAT3-CFG file has the same aforementioned description as in Subsection 4.3.5, where all arithmetic operations are conformed with a new syntax of iSAT3 that supports IEEE 754. *cl_double_assign* and *cl_double_equal* represent equal operation, *cl_double_add* represents arithmetic addition, *cl_double_constant* defines constants, and *cl_double_less* represents comparison operators. While verifying this simple SMI-program in iSAT3 with interpolation-based model checking, the solver does not terminate in 300 seconds. However, when we verify the same problem in iSAT3 with CEGAR technique, it takes 4 seconds to prove the safety of this simple example.

### 4.5.6 BTC-ES benchmarks

We interface our tool with BTC-ES AG tools interfaces in order to verify large-scale problems that come from real life applications with CFG-based representation. From BTC-ES AG, the C-code embedded program is auto-generated from Simulink model. After that, the auto-generated code will be simplified/elaborated to SMI code. At this point, our interface comes to the scene by converting the latter SMI code to corresponding iSAT3-CFG syntax such that the whole while-loop in SMI code will be considered as a complete CFG program in iSAT3.[8]

In this subsection, we concisely show a result of verifying 18 benchmarks given by BTC-ES AG. These benchmarks represent several test-cases generated from SMI code. In each benchmark, the model checker is asked whether the negation of invariant is reachable or not. Four test cases are safe, because there exists no counterexample such that it violates the invariant of the model.

The others have been reported with unsafe verdicts, since they contain counterexamples at different depths as shown in Table 4.4. We verify the list of converted BTC benchmarks by using several options:

- CEGAR with ITP, where refinement is performed by using inductive interpolants, however the abstraction is progressively verified by using interpolation-based model checking approach.
- CEGAR with BMC , where refinement is performed by using inductive interpolants too, however the abstraction is verified by using bounded model checking till depth 250. This combination takes the advantage of using CEGAR to avoid the state space explosion, and the advantage of using BMC to dis/prove bounded safety.
- (ITP) interpolation-based model checking approach is used, where McMillan's rules are employed (cf. Subsection 4.2.3).

[8] We refer to the fact that BTC-ES tools treat each execution of the whole while-loop as one step in contrast to our approach. Thus, our step is a micro-step in comparison to BTC-ES terminology.

| Program features \ Approach | | | | | | iSAT3 CEGAR, BMC till 250, FP | | | | iSAT3 CEGAR, ITP, FP | | | | iSAT3 ITP, FP | | | | iSAT3 BMC, max-depth 250, FP | | | | iSAT3 BMC, max-preprocess 250, FP | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **No** | **Name** | **Non-linear** | **Loops** | **#Nodes** | **#Edges** | **Time(s)** | **Memory(KB)** | **Iteration** | **Result** | **Time(s)** | **Memory(KB)** | **Iteration** | **Result** | **Time(s)** | **Memory(KB)** | **Depth** | **Result** | **Time(s)** | **Memory(KB)** | **Depth** | **Result** | **Time(s)** | **Memory(KB)** | **Depth** | **Result** |
| 1 | btc_test0001 | ✓ | ✓ | 306 | 527 | TO | 461400 | 126 | UNKNOWN | TO | 472392 | 114 | UNKNOWN | TO | 1757032 | 46 | UNKNOWN | 3587.543 | 8328048 | 121 | UNSAFE | **2208.344** | **9502356** | **121** | **UNSAFE** |
| 2 | btc_test0002 | ✓ | ✓ | 304 | 522 | TO | 679860 | 108 | UNKNOWN | TO | 681772 | 97 | UNKNOWN | TO | 1929676 | 72 | UNKNOWN | 1101.558 | 8301284 | 125 | UNSAFE | **1006.023** | **9605640** | **125** | **UNSAFE** |
| 3 | btc_test0003 | ✓ | ✓ | 326 | 557 | TO | 660352 | 117 | UNKNOWN | TO | 1891528 | 107 | UNKNOWN | TO | 1879932 | 50 | UNKNOWN | 3723.347 | 11249848 | 143 | UNSAFE | **3098.896** | **11587728** | **143** | **UNSAFE** |
| 4 | btc_test0004 | ✓ | ✓ | 330 | 570 | TO | 467476 | 82 | UNKNOWN | TO | 546140 | 67 | UNKNOWN | TO | 1902692 | 24 | UNKNOWN | TO | 10540136 | 122 | UNKNOWN[a] | TO | 11572152 | 130 | UNKNOWN[b] |
| 5 | btc_test0005 | ✓ | ✓ | 326 | 557 | TO | 664740 | 118 | UNKNOWN | TO | 1513140 | 108 | UNKNOWN | TO | 1843116 | 36 | UNKNOWN | **4285.108** | **11395676** | **143** | **UNSAFE** | 4365.199 | 11868120 | 143 | UNSAFE |
| 6 | btc_test0006 | ✓ | ✓ | 314 | 538 | TO | 475568 | 118 | UNKNOWN | TO | 510964 | 109 | UNKNOWN | TO | 1681276 | 36 | UNKNOWN | 3925.022 | 8769804 | 129 | UNSAFE | **3055.089** | **9699020** | **129** | **UNSAFE** |
| 7 | btc_test0007 | ✓ | ✓ | 300 | 515 | TO | 655108 | 120 | UNKNOWN | TO | 1085276 | 112 | UNKNOWN | TO | 1736668 | 48 | UNKNOWN | 2451.913 | 7206168 | 121 | UNSAFE | **1408.440** | **8525276** | **121** | **UNSAFE** |
| 8 | btc_test0008 | ✓ | ✓ | 306 | 445 | 804.179 | 360312 | 28 | MODEL ERROR | 1173.904 | 410584 | 28 | MODEL ERROR | 739.866 | 1442900 | 20 | SAFE | **21.336** | **1975252** | **36** | **MODEL ERROR** | 41.015 | 3138004 | 36 | MODEL ERROR |
| 9 | btc_test0009 | ✓ | ✓ | 314 | 538 | TO | 481200 | 114 | UNKNOWN | TO | 511900 | 107 | UNKNOWN | TO | 1600756 | 46 | UNKNOWN | 4186.539 | 8644148 | 129 | UNSAFE | **2889.200** | **9605472** | **129** | **UNSAFE** |
| 10 | btc_test0010 | ✓ | ✓ | 300 | 436 | 407.355 | 353056 | 23 | MODEL ERROR | 821.632 | 371036 | 23 | MODEL ERROR | 431.567 | 1703688 | 28 | SAFE | **13.168** | **1546128** | **32** | **MODEL ERROR** | 35.852 | 2826304 | 32 | MODEL ERROR |
| 11 | btc_test0011 | ✓ | ✓ | 312 | 535 | TO | 685556 | 103 | UNKNOWN | TO | 687392 | 88 | UNKNOWN | TO | 2057672 | 62 | UNKNOWN | 1977.382 | 8715148 | 129 | UNSAFE | **1484.234** | **10074184** | **129** | **UNSAFE** |
| 12 | btc_test0012 | ✓ | ✓ | 326 | 477 | 284.879 | 365780 | 14 | MODEL ERROR | 707.925 | 381320 | 14 | MODEL ERROR | 270.010 | 2072268 | 28 | SAFE | **7.891** | **934576** | **16** | **MODEL ERROR** | 40.755 | 3283976 | 16 | MODEL ERROR |
| 13 | btc_test0013 | ✓ | ✓ | 340 | 496 | 1275.801 | 398892 | 26 | MODEL ERROR | 1855.949 | 479656 | 26 | MODEL ERROR | 1600.442 | 1725704 | 20 | SAFE | **19.435** | **2243632** | **32** | **MODEL ERROR** | 46.200 | 3760412 | 32 | MODEL ERROR |
| 14 | btc_test0014 | ✓ | ✓ | 306 | 526 | TO | 687248 | 105 | UNKNOWN | TO | 680448 | 91 | UNKNOWN | TO | 2038640 | 70 | UNKNOWN | 1984.302 | 8603456 | 127 | UNSAFE | **925.990** | **9952384** | **127** | **UNSAFE** |
| 15 | btc_test0015 | ✓ | ✓ | 300 | 515 | TO | 655476 | 124 | UNKNOWN | TO | 1086608 | 112 | UNKNOWN | TO | 1651212 | 52 | UNKNOWN | 2271.046 | 7341844 | 121 | UNSAFE | **1162.619** | **8529512** | **121** | **UNSAFE** |
| 16 | btc_test0016 | ✓ | ✓ | 332 | 568 | TO | 688348 | 102 | UNKNOWN | TO | 636272 | 93 | UNKNOWN | TO | 2251548 | 56 | UNKNOWN | TO | 12477784 | 146 | UNKNOWN | **4488.257** | **13024712** | **147** | **UNSAFE** |
| 17 | btc_test0017 | ✓ | ✓ | 300 | 515 | TO | 665956 | 125 | UNKNOWN | TO | 1019304 | 114 | UNKNOWN | TO | 1672768 | 42 | UNKNOWN | 2061.161 | 7307576 | 121 | UNSAFE | **1326.044** | **8453152** | **121** | **UNSAFE** |
| 18 | btc_test0018 | ✓ | ✓ | 314 | 538 | TO | 481056 | 114 | UNKNOWN | TO | 507320 | 106 | UNKNOWN | TO | 1730644 | 34 | UNKNOWN | 5033.578 | 8648460 | 129 | UNSAFE | **3045.812** | **9517036** | **129** | **UNSAFE** |

Table 4.4: Verification results of linear/non-linear BTC models while supporting IEEE 754 standard for floating points. These models are converted to iSAT-CFG syntax then verified. All benchmarks contain loops and polynomials, but no transcendental functions. In case of bounded model checking techniques as in BMC or preprocessing, if the result is SAFE, it means till depth 250. Generally, if the result is MODEL ERROR, it means the model is SAFE independent of problem-depth. These results were obtained while running tests on AMD Opteron(tm) Processor 6328@2.0 GHZ with 505 GB RAM.

[a] When the verification time is limited to 180 minutes, this case is reported as UNSAFE at depth 131 with 8457 s and 10.5 GB.
[b] When the verification time is limited to 180 minutes, this case is reported as UNSAFE at depth 131 with 5508 s and 12 GB.

- (BMC) bounded model checking with maximum depth 250.
- control flow automaton preprocessing which is nothing but BMC with preprocessing applied in advance in order to minimize the computations of instantiating the whole transitions at each depth. That is, the post image of current reachable set of edges which respects the flow of the graph is provided to the model checker. The maximum number of preprocessing steps is 250.

Table 4.4 summarises the results of our experimental evaluation. It comprises six groups of columns. The first groups has the same description as Table 4.1. The second group has the same structure, yet reports results for using iSAT3 with CEGAR however by using inductive interpolant technique where abstraction is verified by using BMC till depth 250. The third group has the same structure, yet reports results for using iSAT3 with CEGAR however by using inductive interpolants technique where abstraction is verified by using ITP. The fourth group has the same structure, yet reports results for using ITP technique. The fifth group has the same structure, yet reports results for using BMC till depth 250. Finally, the sixth group reports the results for using preprocessing approach, where BMC technique is applied after computing the post-image of current reachable set of transitions. The latter technique optimises the deduce and deicide steps, but requires more preprocessing steps.

Figures 4.23 and 4.24 summarize the main findings. Concerning verification time, we observe that preprocessing with BMC till 250 scores the best results. Namely, it needs about 36035 s for processing the full set of benchmarks, equivalent to an average verification time of 2002 s, BMC till depth 250 needs 47452 s total and 2636 s on average, CEGAR with BMC till depth 250 needs 78374 s total and 4354 s on average, interpolation-based model checking needs 78653 s total and 4370 s on average, and CEGAR with ITP needs 80161 s total and 4453 s on average.

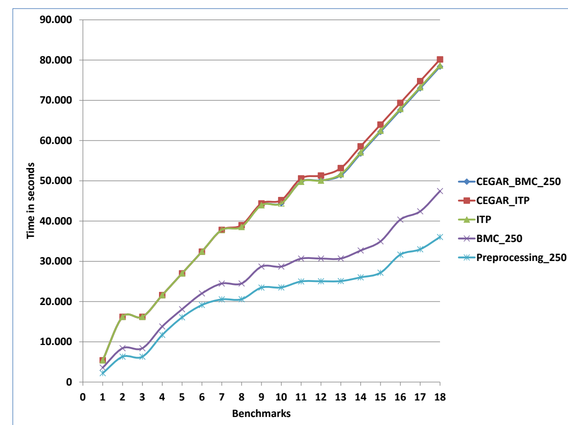



Figure 4.23: Accumulated verification times for the first $n$ benchmarks.

Figure 4.23 shows the verification time results where the plot refers to the accumulated verification times for 18 benchmarks. Concerning memory, we observe that CEGAR with BMC till depth 250 needs about 536 MB on average, CEGAR with ITP needs 731 MB on average, interpolation-based model checking needs 1772 MB on average, BMC till depth 250 needs 7282 MB on average, and preprocessing with BMC till depth 250 needs 8384 MB on average.

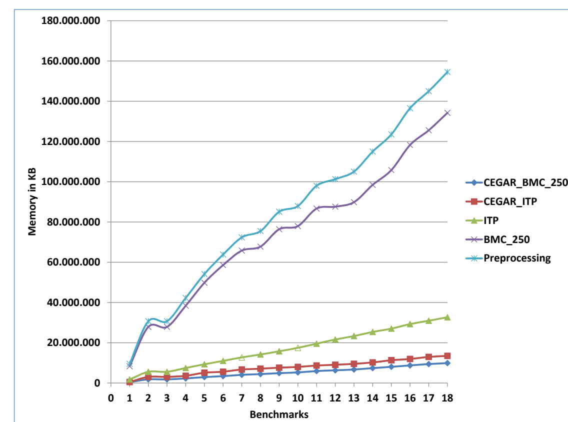
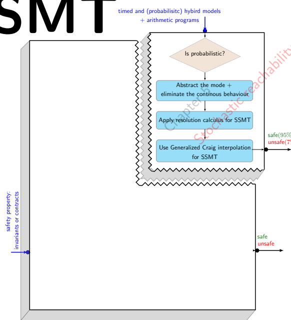



Figure 4.24: Memory usage (#benchmarks processed within given memory limit).

From Table 4.4 and Figures 4.23 and 4.24, we observe:

- proving that such a model is safe can be achieved by any of the aforementioned approaches. Of course, they differ in the verification time and memory, however all can achieve this task partially or totally.
- proving such a model unsafe cannot be achieved by CEGAR or ITP methods within 90 minutes. Both went adrift while computing invariants at each depth, since these test cases are unsafe, thus no safe invariants would be obtained.
- BMC with(out) preprocessing achieves best results with respect to the verification time, however they laboriously overload the memory as shown in Figure 4.24.
- in general, encoding SMI programs in iSAT3-CFG has a good potential to verify the problems with several techniques, in particular preprocessing the problem before applying BMC, where the latter scored the best results.
- the BTC ES AG benchmarks are originally prepared for some in-house tools which can encode the whole while-block as one formula. In contrast to that, we tried to show the feasibility of our iSAT3-CEGAR with its encodings and techniques in solving the same problems without necessarily encoding the whole while-block as one formula. Thus, our tool obviously needs more time and memory due to explicit assignments propagations and due to step-bounded solving of while-blocks.

# 5 Generalized Craig Interpolation for SSMT




An idea is always a generalization, and generalization is a property of thinking. To generalize means to think.

*(Georg Wilhelm Friedrich Hegel)*

## Contents


**5.1 Introduction . . . 123**
- 5.1.1 Motivation . . . 123
- 5.1.2 Related work . . . 124

**5.2 Stochastic Satisfiability Modulo theories (SSMT) . . . 124**
- 5.2.1 SSMT: syntax . . . 125
- 5.2.2 SSMT: semantics . . . 125
- 5.2.3 SSMT: illustrative example . . . 126
- 5.2.4 Complexity of SSMT . . . 127
- 5.2.5 Structure of SSMT formula . . . 127

**5.3 Resolution Calculus for SSMT . . . 129**
- 5.3.1 Resolution rules for SSMT . . . 129
- 5.3.2 Soundness and completeness of SSMT-resolution . . . 131
- 5.3.3 Example of applying SSMT-resolution . . . 133

**5.4 Generalized Craig interpolation for SSMT . . . 134**
- 5.4.1 Generalized Craig Interpolants . . . 135
- 5.4.2 Computation of Generalized Craig Interpolants – Púdlak's rules extension . . . 136

**5.5 Interpolation-based probabilistic bounded model checking . . . 142**
- 5.5.1 Probabilistic bounded reachability – probabilistic safety analysis . . . 143
- 5.5.2 SSMT encoding scheme for PHAs . . . 144
- 5.5.3 PBMC solving by means of generalized Craig interpolation . . . 144
- 5.5.4 Interpolation-based approach for reachability . . . 146
- 5.5.5 Generalized Craig interpolation for Stability analysis . . . 151

## 5.1 Introduction

### 5.1.1 Motivation

Papadimitriou [Pap85] proposed the idea of modelling uncertainty within propositional satisfiability (SAT) by adding randomized quantification to the problem description. The resultant stochastic Boolean satisfiability (SSAT) problems consist of a quantifier prefix followed by a propositional formula.

SSAT has many applications such as bounded model checking (BMC) of symbolically represented Markov decision processes. Stochastic satisfiability modulo theories (SSMT) was proposed in 2008 [FHT08] in order to extend SMT-based bounded model-checking to probabilistic hybrid systems. SSMT extends the satisfiability modulo theories (SMT) problem by randomized quantification or, equivalently, generalizes the stochastic Boolean satisfiability problem (SSAT) [Pap85] to background theories. An SSMT formula consists of a quantifier prefix and an SMT formula. The quantifier prefix is an alternating sequence of existentially quantified variables and variables bound by randomized quantifiers. All the quantified variables have discrete (finite) domains. The meaning of a randomized variable $x \in \mathcal{D}_x = \{val_1 \mapsto p_1, ..., val_n \mapsto p_n\}$ is that $x$ takes value $val_1$ with probability $p_1$, value $val_2$ with probability $p_2$, and so on. The summation of all probabilities for the same variable has to be one.

Due to the presence of probabilistic assignments and randomized quantification, the semantics of an SSMT formula $\delta$ is no longer qualitative in the sense that $\delta$ is satisfiable or unsatisfiable, as for propositional or predicate logic, but rather *quantitative* [FHT08, Tei12]. For an SSMT formula $\delta$, we ask for the maximum probability of satisfaction or, if formulated as a decision problem, whether this probability of satisfaction exceeds a threshold. Intuitively, a solution of $\delta$ is a strategy in form of a tree suggesting optimal assignments to the existential variables depending on the probabilistically determined values of preceding randomized variables, in order to maximize the probability of satisfying the SMT formula. SSMT as proposed by Fränzle et al. [FHT08] can encode bounded probabilistic reachability problems of probabilistic hybrid automaton (PHA) over discrete time. That means many practical problems exhibiting uncertainty can be described as SSMT problems or sometimes even its propositional subset SSAT, in particular probabilistic planning problems [ML98, ML03], belief networks [BDP03], trust management [FK03], or depth-bounded PHA reachability [FHT08, TEF11] and stability problems [Tei12]. Probabilistic bounded model-checking (PBMC) problems, for example, ask whether the probability of *reaching bad states* from the PHA's initial states stays below a given threshold, irrespective of how non-determinism in the PHA is resolved.

Solving a PBMC problem can be achieved by taking its equivalent SSMT encoding and solving it with an SSMT solver, such as Teige's SiSAT tool [Tei12].

Non-polynomial SSMT problems, i.e., SSMT formulae involving transcendental arithmetic, are generally undecidable due to the undecidable underlying arithmetic theory as introduced in Chapter 4. There are some decidable classes of SSMT however; e.g., SSMT formulae without free variables due to the finite domains of bound variables, or SSMT formulae over decidable background theories, like linear order. Undecidability implies that the Craig interpolation problem also cannot be solved exactly in general. In this chapter,

we propose a Craig interpolation procedure for SSMT that is sound and complete when the theory is order theory of the reals, and we extend it to non-polynomial SSMT by using interval constraint propagation (ICP) [Ben96], then obviously sacrificing completeness, yet maintaining soundness.

Essentially, we first use ICP for reducing the general, non-polynomial SSMT problem to an SSMT problem over the linear order over the reals. As an unsatisfied SSMT problem may have satisfying assignments —just not sufficiently many to exceed the target probability threshold—, we then have to compute a *generalized interpolant*, which is a Craig interpolant for $(A, B \wedge \neg S_{A,B})$, where $S_{A,B}$ represents an overapproximation of the satisfying assignments of the formula $A \wedge B$. We do so by extending Púdlak's rules [Pud97] to compute that generalized Craig interpolant. Instrumental to that adaptation of Púdlak's rules is the observation that the theory of linear order, with simple bounds as its atoms, admits a resolution rule akin to the propositional counterpart.

### 5.1.2 Related work

As in Chapter 4, we referred to previous works that use Craig interpolation with(out) other techniques in verifying safety. However, what sets this chapter aside, is that it deals with the same bounded model checking problems admitting stochastic behaviour.

**Generalized interpolation for SSAT.** Teige in [TF12b] proposed generalized Craig interpolation for stochastic Boolean satisfiability (SSAT) problems. Our work extends this to SSMT involving non-polynomial arithmetic constraints. Furthermore, Teige's approach did not address a solution of stochastic models having continuous dynamics, which will be shown in our work.

**Interpolants in presence of non-linear constraints.** Kupferschmid et al. [KB11] was the first to suggest Craig interpolation for non-polynomial and thus undecidable SMT problems by means of ICP and resolution in SMT of linear order. Our approach employs the same mechanism for dealing with arithmetic constraints, but extends the approach to SSMT problems, thus necessitating computation of generalized rather than traditional interpolant.

## 5.2 Stochastic Satisfiability Modulo theories (SSMT)

In this section, we introduce the syntax and semantics of stochastic satisfiability modulo theories (SSMT) formulae, as originally proposed in [FHT08].

### 5.2.1 SSMT: syntax

**Definition 5.1: Syntax of SSMT**

A stochastic satisfiability modulo theories (SSMT) formula $\delta$ is of the form $\mathcal{Q} : \varphi$ where

1. $\varphi$ is an arbitrary SMT formula with respect to the theory of non-polynomial arithmetic over the reals and integers, called the *matrix* of the formula, and
2. $\mathcal{Q} = Q_1 x_1 \in \mathcal{D}_{x_1} \odot .... \odot Q_n x_n \in \mathcal{D}_{x_n}$ is a quantifier prefix binding some variables $x_i \in V(\varphi)$ over finite domains $\mathcal{D}_{x_i} := \{val_1, ..., val_m\}$ by a sequence of existential and randomized quantifiers $Q_i$; i.e., $\exists$ and $\text{Ԋ}_{[val_1 \mapsto p_1, ..., val_m \mapsto p_m]}$ respectively, where $\sum_{i=1}^{m} p_i := 1$.

Free, i.e., unbound by quantifiers, variables are permitted in SSMT formulae. For simplicity, we assume that the matrix $\varphi$ of an SSMT formula $\mathcal{Q} : \varphi$ is in CNF form, as one can convert any formula to a CNF of linear size by introducing auxiliary variables [Tse83] as aforementioned in Section 4.3.

### 5.2.2 SSMT: semantics

**Definition 5.2: Semantics of SSMT**

The semantics of an SSMT formula $\delta$ is given by its maximum probability of satisfaction $Pr(\delta)$ defined as follows:

$$
\begin{aligned}
Pr(\varepsilon : \varphi) &= \begin{cases} 0 \text{ if } \varphi \text{ is unsatisfiable,} \\ 1 \text{ if } \varphi \text{ is satisfiable,} \end{cases} \\
Pr(\exists x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi) &= max_{val \in \mathcal{D}_x} Pr(\mathcal{Q} : \varphi[val/x]), \\
Pr(\text{Ԋ} x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi) &= \sum_{val \in \mathcal{D}_x} d_x(val) \cdot Pr(\mathcal{Q} : \varphi[val/x]).
\end{aligned}
$$

where $d_x$ is a discrete probability distribution over $\mathcal{D}_x$, $\varepsilon$ is an empty prefix quantifier, $\varphi$ is the matrix of $\delta$ and $\mathcal{Q}$ is an arbitrary quantifier prefix.

Definition 5.2 is an extension of the semantics of SSAT, cf. [Pap85, TF12b]. While the interpretation of quantifiers remains the same as for SSAT, their treatment is adapted to handle *discrete domains* with more than two values.

That is, the maximum probability of satisfaction $Pr(\delta)$ of an SSMT formula $\delta$ with a leftmost existential quantifier in the prefix, i.e., $\delta = \exists x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi$, is defined as the maximum of the satisfaction probabilities of all subformulae $\mathcal{Q} : \varphi[val/x]$ that is obtained after removing the leftmost quantified variable from the prefix and substituting values $val \in \mathcal{D}_x$ for variable $x$ in the matrix $\varphi$. If the leftmost variable is randomized, i.e. $\delta = \text{Ԋ}_{d_x} x \in Q_x \odot \mathcal{Q} : \varphi$, then $Pr(\delta)$ demands to compute the weighted sum of the satisfaction probabilities of all subformulae $\mathcal{Q} : \varphi[val/x]$.

The base cases of this definition, that are reached whenever the quantifier prefix becomes empty i.e. $\varepsilon$, yield SMT formulae over the non-quantified (free) variables.

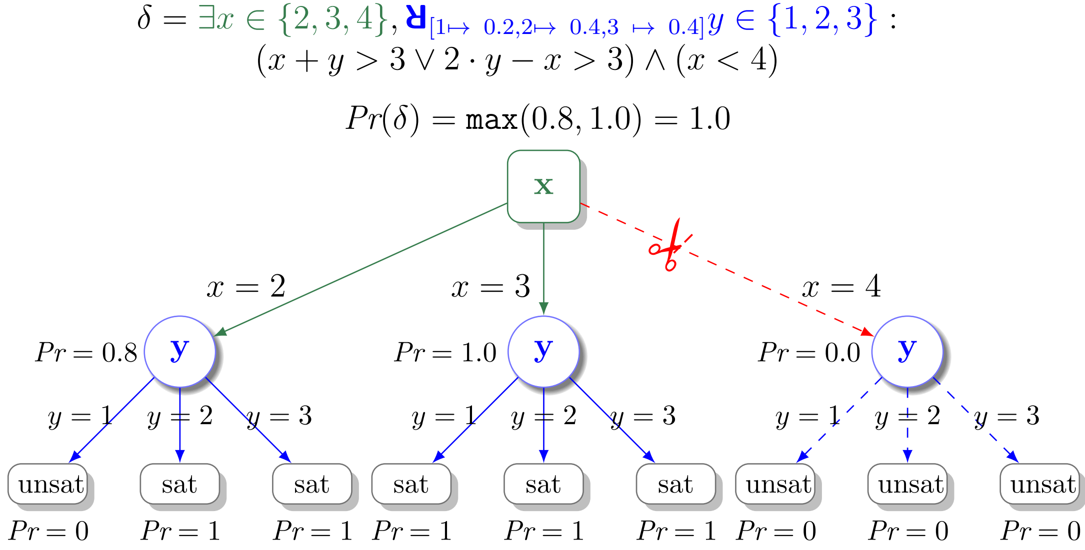



Figure 5.1: $1\frac{1}{2}$ player game semantics of an SSMT formula. In recursive solvers, traversal of the dashed part of the quantifier tree will be skipped due to pruning [Tei12].

This is one of the main differences between SSMT and SSAT, where all variables of the later formula are quantified and each base case thus gives a formula equivalent to either `true` or `false`. Being conformed with the intuition of the maximum probability of satisfaction, we assign satisfaction probability 1 to the remaining quantifier-free SMT formula $Q$ in case $\varphi$ is satisfiable, and probability 0 otherwise, i.e. if $\varphi$ is unsatisfiable. Thereby, the *non-quantified, free, variables* of an SSMT formula can be seen as innermost existentially quantified over possibly (finite) continuous domains.

### 5.2.3 SSMT: illustrative example

**Example 5.1: SSMT semantics: $1\frac{1}{2}$ player game**

Let us consider the following formula which is depicted in Figure 5.1:

$$\exists x \in \{1,2,3\}, ᴚ_{[1\mapsto 0.2, 2\mapsto 0.4, 3\mapsto 0.4]} y \in \{1,2,3\} : (x + y > 3 \lor 2 \cdot y - x > 3) \land (x < 4)$$

The semantics of the previous SSMT formula is a $1\frac{1}{2}$ player game. In naïve SSMT solving, the quantifier tree would be fully unravelled and all resulting instances of the matrix (leaves of the tree) passed to an SMT solver which returns in the most times satisfiable a.k.a. *sat* or unsatisfible a.k.a. *unsat* answers. After that, we compute back the satisfiability probability of parent nodes. For example for the node where the evalation of $x$ is 2, the probability equals the weight of all branches, namely $(0 \cdot 0.02) + (1 \cdot 0.4) + (1 \cdot 0.4) = 0.8$. By the same way we compute the probability of all nodes at the same level (depth). Now, since $x$ is existentially quantified, we need to compute the maximum probability of all branches, i.e., find

the maximum of 0.8, 1.0 and 0.0, which is nothing but 1.0. At this point, we reach the root of the tree and compute the maximum probability of satisfying the given formula.
Pruning rules also shown in Figure 5.1, yet permit to skip investigating a major portion of the instances in general. For more information about these pruning techniques, one may consider the SiSAT model checker and Teige's thesis [Tei12].

### 5.2.4 Complexity of SSMT

Quantified Boolean formulae (QBF) or QSAT are decidable problems; namely PSPACE-complete [Pap94]. QBF is a special case of SSAT [TF10] and the latter problem is also PSPACE-complete [TF10, Pap94] even for S2SAT problems [TF10].

Furthermore, SSAT problems are special cases of SSMT [FHT08], where the latter problems with contrast to SSAT [Pap85], are either fully quantified or containing free variables, i.e. general SSMT formulae. On one hand SSMT and S2SMT problems with linear order (total order) are decidable and PSPACE-complete as one can polynomially reduce [Pap94] SSAT problems to SSMT problems. On the other hand SSMT problems with non-linear constraints, e.g., *exponential*, *sin* functions are undecidable.

### 5.2.5 Structure of SSMT formula

An SSMT formula is consisting of two layers; namely an SSMT layer and an SMT layer. However in this section we show explicitly how these layers are built and communicated. Our proposed structure in Figure 5.2 follows yet adjusts the structure of SiSAT tool [Tei12].

**Topmost layer: SSMT layer.** In this layer we have the SMT formula $\varphi$ with the quantifier prefix $\mathcal{Q}$. The quantifier prefix of this formula is built as shown before in Section 5.2. and the non-quantified SMT (matrix) will be passed to the middle layer (SMT layer) as shown in Figure 5.2.

**Middle layer: SMT layer.** In this layer, one can have conjunctive of linear constraints or non-linear constraints or both of them. This layer employs the lowermost layer by passing the conjunctive model of the system, *where each variable in this layer is assigned to an interval instead of a single assignment.* Consequently all DPLL techniques, such as *deduction*, *unit propagation* are performed in the term of intervals; namely decide means *case split* of variable interval as shown in the example in Figure 5.2 for the variables $n$ and $m$.

Normally if the formula contains only linear constraints, then it can be solved by using Fourier-Motzkin elimination [DE73] or simplex algorithm [Dan63] in this layer. However, we introduced a general architecture to deal with general cases, e.g. linear and non-linear constraints where the latter case needs a special treatment as will be illustrated in the next layer.

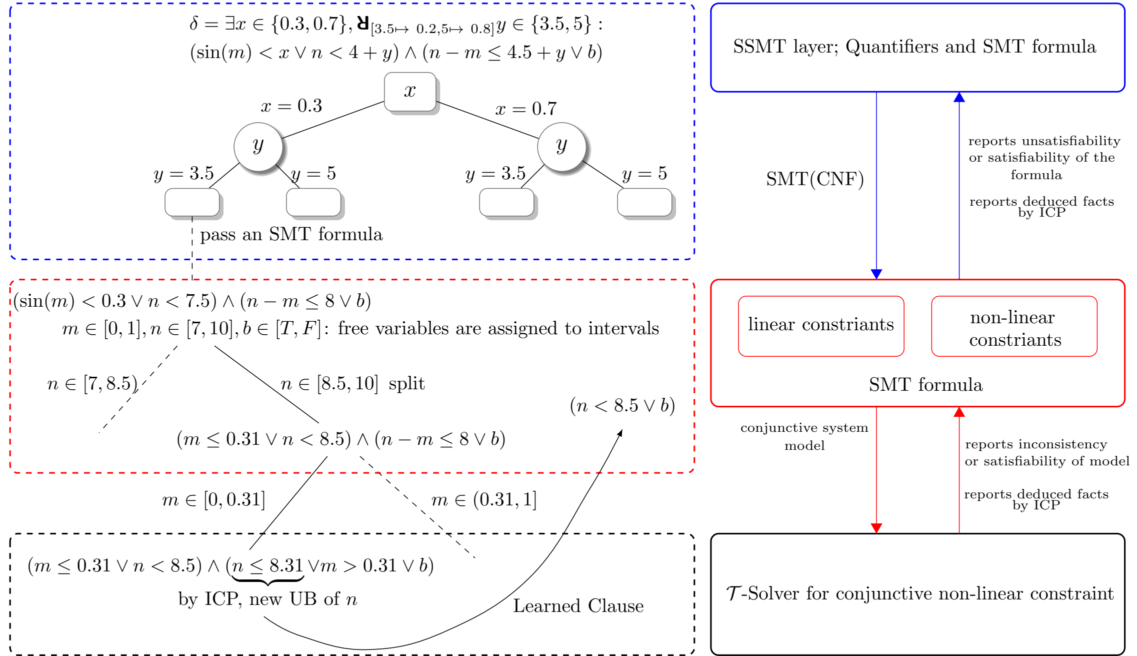



Figure 5.2: On the right side, an architecture of SSMT solver, e.g. SiSAT. On the left side, an example of solving SSMT formula and how this will be mapped to the architecture of an SSMT solver.

**Lowermost layer: $\mathcal{T}$-layer.** This layer is responsible for reasoning about conjunctive systems of non-linear arithmetic constraints over bounded reals and integers [Tei12]. One of efficient mechanisms to handle the latter problems is to use a *safe interval analysis* [Moo95] and an *interval constraint propagation* (ICP) as proposed in SiSAT.

- **Interval analysis:** It is used to evaluate the interval consistency of a conjunctive of non-linear arithmetic constraints involving functions like `sin` and `exp`. Interval consistency is a necessary but not sufficient condition for real-valued satisfiability of the model of constraints. Thus, sometimes iSAT can return weak answers as "*candidate solution*" (cf. Section 4.3).

  There are several definitions of interval consistency [BG06]. They differ only in the strength of their consistency notions and in the computational effort to decide consistency. Our consistency concept (as aforementioned in Section 4.3) is *hull*-consistency. Hull-consistency concept is applied to unary, binary arithmetic operations and simple bounds. When we say $hull(A)$ for some set $A \subseteq \mathbb{R}$ or $(A \subseteq \mathbb{Z})$, called the interval hull of $A$, is the smallest interval containing the set $A$.

- **Interval constraint propagation (ICP) [BMH94]:** It is integrated with interval consistency as a deduction mechanism to cut-off irrelevant parts from the variable assignments by narrowing the intervals (contractors) [Ben96, BMH94] while trying to achieve hull consistency. Intuitively if we are given a constraint[1]and a certain area $B$ where the solution is expected, then ICP technique finds another area $B'$ such

[1]In SiSAT, only primitive constraints are considered, i.e. constraints containing one relation and at most one arithmetic operator, and at most three variables [KBTF11]

that the new area $B'$ is a subset of the original area $B$ and it contains all solutions of the constraint in $B$. For example in Figure 5.2 variable $n$ was assigned to the interval $[8.5, 10]$. After that by assigning $m$ to $[0, 0.31]$, $b$ to *false* and by using ICP, $n$ was assigned to $(-\infty, 8.31]$ (new upper bound of $n$ was deduced). The new interval of $n$ conflicts (is inconsistent) with the first interval assigned to $n$ i.e., $[8.5, 10]$.

## 5.3 Resolution Calculus for SSMT

The existing SSMT solving algorithms of Teige [Tei12] are tightly integrated with the CDCL(ICP) proof search of the iSAT tool [FHR+07] and do, in principle, traverse the quantifier tree of the formula as in Figure 5.1 to recursively compute the maximum satisfaction probability bottom-up. Note that this does by no means imply that they are bound to traverse the whole, exponentially sized quantifier tree, as Teige proposed various mechanisms to drastically prune that tree and thus accelerate the actual computation.

### 5.3.1 Resolution rules for SSMT

In contrast to the CDCL(ICP) approach, the SSMT resolution calculus, as proposed by the author based on Teige's SSAT resolution [TF10], solves SSMT problems by a resolution mechanism. SSMT-resolution works by deriving attributed clauses $cl^p$, where $cl$ is a clause and $p$ is a probability. When such a clause $cl^p$ is derived during resolution, it expresses that the maximum probability of violation of $cl$ is $p$. If the probabilistic variant $\emptyset^p$ of a conflict clause happens to be derived at the end of resolution, then the maximum probability that the formula holds is $p$. The related SSAT-resolution calculus proposed by Teige [TF10, TF12b] is *sound* and *complete*. The same applies for SSMT resolution if the theory is confined to linear order over the reals, yet if (e.g., non-polynomial) arithmetic is involved, the resolution calculus of SSMT is *sound* but only *relatively complete* with interval constraint propagation (ICP) [BMH94] being its "oracle" for resolving arithmetic [AM13a].

All derived clauses $cl^p$ are forced to have a tight bound $p$ in the sense that under each assignment which falsifies $cl$, the satisfaction probability of the remaining subproblem is exactly $p$.[2] Before illustrating the resolution rules, we define the symbolic falsifying assignment $\mathit{ff}_{sp}$ that captures variable assignments falsifying a clause $cl$. A simple bound $x \sim c \in \mathbb{SB}$ is introduced in Definition 4.1, means that a variable $x$ is restricted by comparison operator, i.e., $\sim \in \{>, \geq, <, \leq\}$, relative to value $c$, where the latter value is a real number. Also, we assign to each variable a domain which is a bounded interval. Let $cl$ be a non-tautological disjunction of simple bounds. We define the falsification function $\mathit{falsify}_c$ that falsifies $cl$ as follows:

[2] If we relax the condition to a probability of less than or equal to $p$, it works also. The stronger form used here makes interpolation simpler, however with both forms the resolution and interpolation can be applied forthrightly.

**Definition 5.3: Falsification function**

Let $\mathbb{CL}$ be a set of all non-tautological clauses with disjunction of simple bounds. If $cl \in \mathbb{CL}$ is a typical element; i.e., $cl : sb_1 \vee ... \vee sb_n$. The falsification function $falsify_{cl} : \mathbb{CL} \to \mathbb{CL}$ is defined as follows:

- $falsify_{cl}(cl) := \bigvee_{i=1}^{n} ff_{sp}(sb_i)$,
- $ff_{sp} : \mathbb{SB} \to \mathbb{SB}$ s.t. $ff_{sp}(x \sim c) := x \sim' c$ where $\sim'$ is the converse relation to $\sim$, e.g., $\leq'$ is $>$.

where $x \in X$, $c \in \mathbb{R}$, $\sim, \sim' \in \{\leq, <, \geq, >\}$ and $x$ has a well-defined domain.

In the following proposition which will be used latter, we show an important property of SSMT formulae; namely under an assignment $\tau$ that *falsifies* a clause $cl$ in a SSMT formula $\varphi$ in CNF, the satisfaction probability of the SSMT formula $\mathcal{Q} : \varphi$ under $\tau$ is 0.

**Proposition 5.1: Falsification property**

Let $\varphi$ be some SMT formula with $V(\varphi) = \{x_1, ..., x_n\}$, $\mathcal{Q} = Q_{i+1}x_{i+1}...Q_n x_n$ be a quantifier prefix, and $V(\varphi) \downarrow_j := \{x_1, ..., x_j\}$ for $j \leq n$. Then if $\varphi$ is in CNF and there is a non-tautological clause $cl \in \varphi$ s.t. $V(cl) \subseteq V(\varphi) \downarrow_i$ then for each $\tau : V(\varphi) \downarrow_i \to \mathbb{SB}$ with $\forall x \in V(cl) : \tau(x) = ff_{sp}(x \sim c)$, where $x \sim c \in cl$ then:

$$Pr(\mathcal{Q} : \varphi[\tau(x_1)/x_1]...[\tau(x_i)/x_i]) = 0.$$

**Proof of falsification property**

The idea of the proof: we can construct $\tau$ and since clause $cl$ is non-tautological, it holds that $cl[\tau(x_1)/x_1]...[\tau(x_i)/x_i] \equiv$ `false`. Since $\varphi$ is in CNF and $cl \in \varphi$, it follows that formula $\varphi[\tau(x_1)/x_1]...[\tau(x_i)/x_i]$ with variables $x_{i+1}, ..., x_n$ is unsatisfiable. Immediately, $Pr(\mathcal{Q} : \varphi[\tau(x_1)/x_1]...[\tau(x_i)/x_i]) = 0$

In order to extend the SSAT resolution rules to SSMT formulae, we assume w.l.o.g. that any clause $cl$ where resolution is applied consists of disjunctions of simple bounds only, as ICP yields a reduction to simple bounds by propagating arithmetic constraints into simple bounds [Tei12, AM13a]. We will introduce four resolution rules that define the resolution calculus for SSMT problems. Rule RR.1 derives a clause $cl^{0.0}$ from an original clause $cl \in \varphi$ such that $cl$ is not a tautological clause. One can consider RR.1 corresponds to the quantifier-free base case where $\varphi$ is `false` under any assignment that falsifies $cl$.

$$\frac{\left(cl \in \varphi\right)}{cl^{0.0}} \tag{RR.1}$$

Rule RR.2 reflects the quantifier-free base case in which $\varphi$ is `true` under any assignment that conforms to the partial assignment $\tau$, since $\models \varphi[\tau(x_1)/x_1]...[\tau\ (x_i)\ /x_i]$. The constructed $cl^{1.0}$ represents the negation of the satisfiable partial assignment $\tau$ of $\varphi$.

$$\frac{\left(\begin{array}{c} cl \subseteq \{x \sim c \mid x \in V(cl)\}, \not\models cl, \mathcal{Q}(cl) = Q_1 x_1 ... Q_i x_i, \\ \text{for each } \tau : V(\varphi) \downarrow_i \rightarrow \mathbb{SB} \text{ with } \forall x \in V(\varphi) : \tau(x) \text{ in } \mathit{ff}_{sp}(x \sim a) : \\ \models \varphi[\tau(x_1)/x_1]...[\tau(x_i)/x_i] \end{array}\right)}{cl^{1.0}} \tag{RR.2}$$

Rule RR.3 computes the actual probability of a resolvent depending on the type of the quantifier governing the pivot variable, where a bound on the pivot variable is used as the resolution literal. Definition 5.2 enforces that the domain of any quantified variable is discrete, which implies that we can evaluate the probability by simply summing up or selecting the maximum of the probabilities of satisfying assignments for Ɐ- or ∃-quantified variable $x$, respectively

$$\frac{\left(\begin{array}{c} (x \sim c_1 \vee cl_1)^{p_1}, (x \sim' c_2 \vee cl_2)^{p_2}, (x \in \mathcal{D}_(x) \wedge x \sim c_1 \wedge x \sim' c_2 \vdash \texttt{false}) \\ Q_x \in \mathcal{Q}, \not\models (cl_1 \vee cl_2) \\ p = \begin{cases} max(p_1, p_2) & \text{if } Q_x = \exists x \in \mathcal{D}_x \\ p_1 \cdot Pr(x \sim' c_1) + p_2 \cdot Pr(x \sim c_2) & \text{if } Q_x = Ɐ^{Pr} x \in \mathcal{D}_x \end{cases} \end{array}\right)}{(cl_1 \vee cl_2)^p} \tag{RR.3}$$

Rule RR.3e is a counterpart of RR.3 for free variables in SSMT formulae. All free variables are implicitly existentially quantified at innermost level, yet —in contrast to explicit quantification— to continuous domains in general.

$$\frac{\left(\begin{array}{c} (x \sim c_1 \vee cl_1)^{p_1}, (x \sim' c_2 \vee cl_2)^{p_2}, Q_x \notin \mathcal{Q}, x \text{ has domain } \mathcal{D}_x \\ (x \in \mathcal{D}_x \wedge x \sim c_1 \wedge x \sim' c_2) \vdash \texttt{false}, \not\models (cl_1 \vee cl_2) \\ p = max(p_1, p_2) \end{array}\right)}{(cl_1 \vee cl_2)^p} \tag{RR.3e}$$

Note that the SSMT-resolution calculus is *sound* and *relatively complete* w.r.t. to its underlying arithmetic reasoner ICP. On SSMT problems over the theory of linear order, SSMT resolution is *complete.*

### 5.3.2 Soundness and completeness of SSMT-resolution

In the following, we prove the *soundness and relatively completeness* of the SSMT-resolution calculus.

It is important to notice that applying SSMT resolution rules has to respect the SSMT quantifier prefix orders (from innermost to outermost). If the latter condition is not taken into our consideration, then the completeness of this calculus may not hold.

**LEMMA 5.1: $cl^p$-RESOLUTION**

Let clause $cl^p$ be derivable by SSMT-resolution. Further, let be $\mathcal{Q}(cl) = Q_1 x_1 ... Q_i x_i$. Then for each $\tau : V(\delta) \downarrow_i \rightarrow \mathbb{SB}$ *with* $\forall x \in V(cl) : \tau(x) = \mathit{ff}_{sp}(x \sim a)$, where $x \sim c \in cl$ it holds that

$$Pr(Q_{i+1} x_{i+1} ... Q_n x_n : \varphi[\tau(x_1)/x_1]...[\tau(x_i)/x_i]) \leq p$$

### Proof of $cl^p$-resolution

We will prove the lemma by induction over application of rules as follows:

**Base case:** In the base case, we can only use RR.1 or RR.2. If $\tau$ is constructed correctly, we get that $\varphi[\tau(x_1)/x_1]...[\tau(x_i)/x_i]$ is unsatisfiable in case of RR.1 (see Proposition 5.1) and tautological in case of using RR.2 which immediately establishes the result for the base case.

**Hypothesis:** Assume that the premises of rule RR.3 and RR.3e hold; namely

$$Pr(Q_{j+1}x_{j+1}...q_n x_n : \varphi[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][\overline{x \sim c_1}/x_j]) \leq p_1$$

$$Pr(Q_{j+1}x_{j+1}...Q_n x_n : \varphi[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][\overline{x \sim' c_2}/x_j]) \leq p_2$$

where $x_j = x$ with $j \geq i+1$. By definition of SSMT semantics, for each $\tau$ with $\tau(x) = \tau_1(x)$ if $x \in V(cl_1)$ and $\tau(x) = \tau_2(x)$ if $x \in V(cl_2)$, then we have

$$Pr(Q_j x_j...Q_n x_n : \varphi[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}]) \leq p \quad (*)$$

The result for $j = i+1$ holds as all variables from $x_1, ..., x_{j-1}$ are not quantified in $(*)$. For case that $j > i+1$, all variables $x_{i+1}, ..., x_{j-1}$ do not occur in the derived clause $(cl_1 \vee cl_2)$. Thus, for $k = j-1$ to $i+1$ (we have only two elements), thereby we successively conclude that

$$Pr(Q_{k+1}x_{k+1}...Q_n x_n : \varphi[\tau(x_1)/x_1]...[\tau(x_{k-1})/x_{k-1}][\overline{x \sim c_1}/x_k]) \leq p$$

$$Pr(Q_{k+1}x_{k+1}...Q_n x_n : \varphi[\tau(x_1)/x_1]...[\tau(x_{k-1})/x_{k-1}][\overline{x \sim' c_2}/x_k]) \leq p$$

**Induction step:** For case $k = i+1$, the lemma follows.

### Corollary 5.1: Soundness of SSMT-resolution

If the empty clause, i.e., $\emptyset^p$ is derivable by SSMT-resolution from a given SSMT formula $\mathcal{Q} : \varphi$, then $Pr(\mathcal{Q} : \varphi) \leq p$.

### Proof of Soundness of SSMT-resolution

This proof follows Lemma 5.1 and its proof, since the conflict clause $\emptyset^p$ is a special case of $cl^p$.

### Theorem 5.1: Relatively Completeness of SSMT-resolution

If $Pr(\mathcal{Q} : \varphi) \leq p < 1$ for some SSMT formula $\delta := \mathcal{Q} : \varphi$, then the empty clause, i.e., $\emptyset^p$ is derivable by SSMT-resolution; i.e., SSMT-resolution is relatively complete.

**Proof of Relatively Completeness of SSMT-resolution**

In order to prove this theorem, we split the proof into two parts:

- if $\emptyset \in \varphi$, then the formula is unsatisfiable. Consequently $\emptyset^0$ is derived by Rule RR.1.
- if $\emptyset \notin \varphi$, then we prove this case by induction over the number of quantifiers as follows:

  **Base Case:** $Q = Q_x$:
  - $\varphi = (x \sim c_1) \wedge (x \sim' c_2)$, where the latter clauses are disjoint. So by RR.1 we derive $(x \sim c_1)^{0.0}$ and $(x \sim' c_2)^{0.0}$. By RR.3 or RR.3e, we derive $\emptyset^{0.0}$.
  - $\varphi = (x \sim c_1)$. By RR.1 we derive $(x \sim c_1)^{0.0}$. By RR.2 we derive $(\overline{x \sim c_1})^{1.0}$. Then if $Q = \exists$, by RR.3 or RR.3e we get the empty set with the maximum probability, i.e. 1.0. In case that $Q = \text{Ⅎ}$, then by RR.3 we get the empty set with probability $(Pr(x \sim c_1) \cdot 1.0 + Pr(\overline{x \sim c_1}) \cdot 0.0)$, which is nothing but $Pr(x \sim c_1)$ i.e. $\sum_{val \in \mathcal{D}_x} Pr(val \sim c_1)$.

  **Hypothesis:** We assume that $p_1 \geq Pr(\mathcal{Q} : \varphi[val_1/x])$ , ..., $p_n \geq Pr(\mathcal{Q} : \varphi[val_n/x])$ where $p_1, ..., p_n < 1$. Then $\emptyset^{p_1}$,...,$\emptyset^{p_n}$ are derived by $\mathcal{Q} : \varphi[val_1/x]$,...,$Q : \varphi[val_n/x]$ respectively.

  **Induction step:**
  - consider that the domain of $x$, i.e., $\mathcal{D}_x = \{val_1, val_2\}$. If we apply the resolution sequence to derive $\emptyset^{p_1}$ from $\mathcal{Q} : \varphi[val_1/x]$ on $Q_x\mathcal{Q} : \varphi$, then we get either $\emptyset^{p_1}$ or $(\overline{x = val_1})^{p_1}$. With the same procedure, we get $\emptyset^{p_2}$ or $(\overline{x = val_2})^{p_2}$. If $\emptyset^{p_1}$ or $\emptyset^{p_2}$ is derived, then it means that $p = p_1$ or $p = p_2$ respectively. Otherwise, we apply the resolution rule RR.3 between $(\overline{x = val_1})$ and $(\overline{x = val_2})$ to derive the empty clause; i.e., $\emptyset^p$.
  - now if $|\mathcal{D}_x| = n$ and $n > 2$. Then applying resolution sequence yields $\emptyset^{p_1}$ or $(\overline{x = val_1})^{p_1}$,..., $\emptyset^{p_n}$ or $(\overline{x = val_n})^{p_n}$. If $\emptyset^{p_1}$... $\emptyset^{p_n}$ are derived, then it means that $p = p_1$ or...or $p = p_n$. Otherwise, we apply RR.3 between $(\overline{x = val_1})^{p_1}$,..., $(\overline{x = val_n})^{p_n}$ to get the conflict clause with probability $p$ where $p$ is computed according to RR.3 or RR.3e; namely if $x$ is existentially quantified, then $p = \mathtt{max}(p_1, \mathtt{max}(p_2, \mathtt{max}(...\mathtt{max}(p_{n-1}, p_n)))) = \mathtt{max}(p_1, ..., p_n)$. If $x$ is quantified by Ⅎ, then $p$ will be computed according to the weight function in RR.3.

### 5.3.3 Example of applying SSMT-resolution

**Example 5.2: Example of applying SSMT-resolution**

Consider the following SSMT formula:

$$\text{Ⅎ}_{[1 \mapsto 0.2, 3 \mapsto 0.35, 5 \mapsto 0.45]} x \ \exists y \in \{2, 4\} \ \text{Ⅎ}_{[-1 \mapsto 0.5, 0 \mapsto 0.5]} z \ \text{Ⅎ}_{[0 \mapsto 0.15, 1 \mapsto 0.15, 2 \mapsto 0.7]} w : (z < -0.5) \wedge (x > 2.5 \vee y > 2.8) \wedge (y < 3) \wedge (z \geq 0 \vee w \leq 1.7).$$

This formula is satisfiable with the probability 0.12 by solving it with SiSAT. However, we will show how to get the same result while solving it with SSMT-resolution. Figure 5.3 represents SSMT-resolution of the formula. Here at the end of the SSMT-resolution tree, the conflict clause with the least upper bound probability, i.e., the

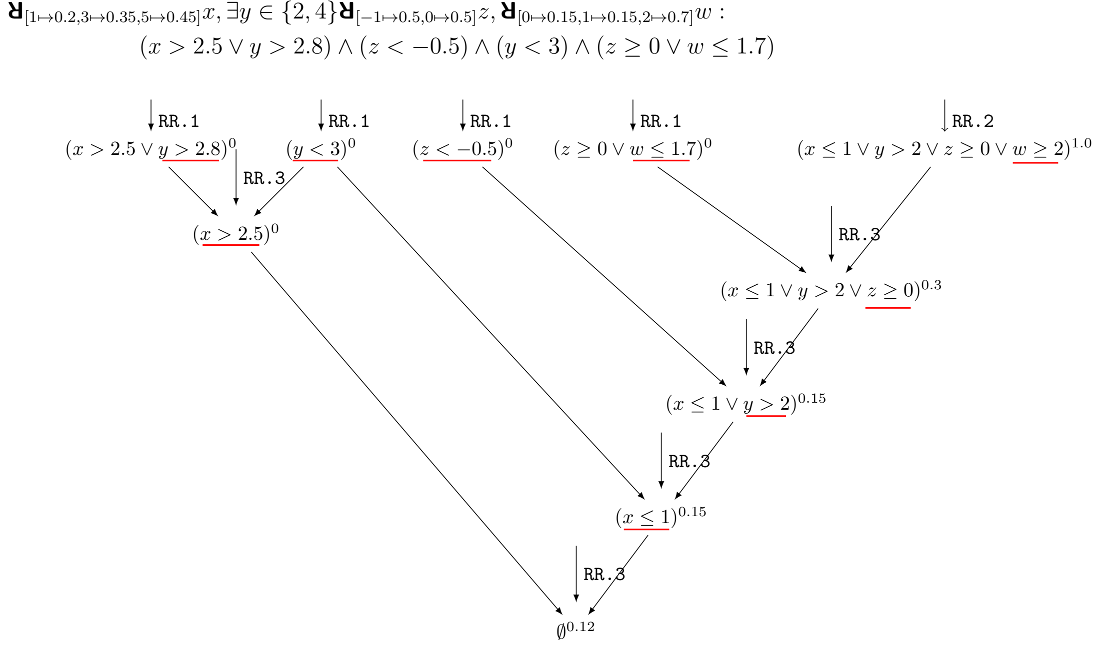



Figure 5.3: Example of SSMT-resolution and computing the satisfaction probability 0.12. Red lines identify the pivots.

exact satisfiable probability is obtained. One can derive the conflict clause with different upper bounds probabilities in case that Rule RR.2 condition is relaxed. In other words, if we compute the negation of overapproximation of satisfying assignments rather the the negation of the satisfying assignments.

## 5.4 Generalized Craig interpolation for SSMT

Craig interpolation is a logical concept suggested by Craig in 1957 [Cra57] that has been widely used in model theory and automatic verification (cf. Section 4.3), since its classical, non-probabilistic form, provides a reason for mutual inconsistency between two formulae as introduced in Definition 4.8 and discussed in details in Chapter 4.

Following efficient schemes that have been devised for propositional logic and for SAT-modulo-theory by exploiting the connection between resolution and variable elimination [Pud97, EKS06], Teige et al. [TF12b] succeeded to generalize the Púdlak rules for interpolant synthesis [Pud97] from the propositional SAT case to stochastic SAT, where a more general definition of interpolant is needed, based on S-resolution [TF10] for SSAT. In the sequel of this chapter, we will do the same, yet for SSMT.

### 5.4.1 Generalized Craig Interpolants

Traditional interpolation requires that $A \wedge B$ is unsatisfiable for the formulae $A$ and $B$ to interpolate. The precondition $A \wedge B \models \texttt{false}$, which would be translated to $Pr(A \wedge B) = 0$ in a stochastic setting, however is too restrictive for use in probabilistic model-checking, as a residual chance of failure — which amounts to satisfying a path condition $A \wedge B$ in that context — is well acceptable in many engineering problems [Tei12, TF12b]. As an example consider the quantitative safety target "*The probability that a plane will crash is at most* $10^{-9}$ *per year*".

For a violation of this quantitative safety goal, we cannot find a classical interpolant in general. Teige et al. proposed a general concept which can be used to form an adequate lattice of interpolants for stochastic problems.

In order to build interpolant lattice for SMT formulae $(A, B)$ which may collapse to the empty one, we need to redefine the bottom and top elements of the interpolant lattice. Namely, instead of using $A^{\exists}$ as a top element of the lattice and using $\overline{B}^{\forall}$ as bottom element of the lattice, we use $A^{\exists} \vee \overline{B}^{\forall}$ and $A^{\exists} \wedge \overline{B}^{\forall}$ respectively.

**Definition 5.4: Generalized Craig Interpolant [TF12b]**

Let $A$ and $B$ be some SMT formulae where $V_A := V(A) \setminus V(B) = \{a_1, ..., a_\alpha\}$, $V_B := V(B) \setminus V(A) = \{b_1, ..., b_\beta\}$, $V_{A,B} := V(A) \cap V(B)$, $A^{\exists} = \exists a_1, ..., a_\alpha : A$, and $\overline{B}^{\forall} = \neg\exists b_1, ..., b_\beta : B$. An SMT formula $\mathcal{I}$ is called a generalized Craig interpolant for $(A, B)$ if and only if the following properties are satisfied:

1. $\models_{\mathcal{L}} (A^{\exists} \wedge \overline{B}^{\forall}) \rightarrow \mathcal{I}$,
2. $\models_{\mathcal{L}} \mathcal{I} \rightarrow (A^{\exists} \vee \overline{B}^{\forall})$, and
3. $V(\mathcal{I}) \subseteq V_{A,B}$.

For SMT calculi admitting quantifier elimination, like the linear fragments of integer [Coo72] and rational [FR75] as well as the polynomial fragment of real arithmetic [Tar48, DH88], the four quantifier-free SMT formulae equivalent to $A^{\exists} \wedge \overline{B}^{\forall}$, to $A^{\exists}$, to $\overline{B}^{\forall}$, and to $A^{\exists} \vee \overline{B}^{\forall}$ can serve as generalized Craig interpolants for $(A, B)$. These fragments of arithmetic are, however, very confined. A – necessarily incomplete – interpolation procedure can, however, be obtained for the non-polynomial case based on ICP, which reduces arithmetic reasoning to bound reasoning, i.e., to the decidable case of the theory of linear order over the reals and integers.

An interpolation procedure for SMT involving transcendental functions based on the latter principle has been pioneered by Kupferschmid et al. [KB11] without, however, addressing the stochastic case of generalized Craig interpolants (GCI). GCI for the propositional case of SSAT, on the other hand, have been explored by Teige et al. [TF12b]. We will here reconcile these lines in order to compute GCI for SSMT.

### 5.4.2 Computation of Generalized Craig Interpolants – Púdlak's rules extension

In this subsection, we present a formal way of computing the Craig interpolants for SSMT formulae by defining certain rules based on the SSMT resolution calculus.

In order to compute systemically the Craig interpolants, one can use Púdlak's technique [Pud97] (symmetric) or McMillan's technique [McM03] (asymmetric) or the duality of McMillan's technique, which are built on top of the resolution calculus for propositional logic. For SSAT problems, Teige [TF12b] extended the SAT resolution [TF10] and Púdlak rules and succeeded to compute the interpolants for SSAT.

We use SSMT resolution for computing generalized Craig interpolants. For this purpose, the rules of SSMT resolution are extended to deal with pairs $(cl^p, \mathcal{I})$ of annotated clauses $cl^p$ and an SMT formulae $\mathcal{I}$, where $\mathcal{I}$ represents a partial generalized interpolant [TF12b, KB11].Whenever a pair $(\emptyset^p, \mathcal{I})$ denoting the empty clause is derived, a generalized Craig interpolant for the given SSMT formula has been computed. We compute the interpolant according to the following three rules GR.1, GR.2 and GR.3 given below. The first Rule GR.1 represents a base case assigning initial interpolants to each clause of $A$ and $B$.

$$\frac{cl \vdash_{\mathsf{RR.1}} cl^{0.0}, \quad \mathcal{I} = \begin{cases} \texttt{false}, & cl \in A \\ \texttt{true}, & cl \in B \end{cases}.}{(cl^{0.0}, \mathcal{I})} \tag{GR.1}$$

Rule GR.2 does not exist in non-stochastic interpolation, as it refers to rule RR.2 of SSMT resolution, where the partial assignment satisfies $A \wedge B$, which is impossible in the traditional setting. If we take the negation of the satisfying assignments of $A \wedge B$; i.e., $\neg S_{A,B}$, then $A \wedge \neg S_{A,B}$, and $\neg S_{A,B} \wedge B$ are unsatisfiable. Therefore, we can choose the interpolant freely over the shared variable between $A$ and $B$, i.e., $V_{A,B}$. This freedom enables us to control the *geometric extent* of generalized Craig interpolants within "don't care"-region provided by the models of $S_{A,B}$ [TF12b].

$$\frac{\vdash_{\mathsf{RR.2}} cl^{1.0} \quad \mathcal{I} \text{ is any formula over } V_{A,B}}{(cl^{1.0}, \mathcal{I})} \tag{GR.2}$$

The third rule extends Púdlak's rule for resolution in the direction of SMT simple bounds. Whenever we have two conflicting simple bounds in different clauses, we can apply SSMT resolution provided that the resolvent is not a tautology. If $x$ is a quantified variable, we apply RR.3, otherwise we use RR.3e in case that $x$ is a free variable.

$$\frac{\begin{array}{c}((x \sim c_1 \vee cl_1)^{p_1}, \mathcal{I}_1), ((x \sim' c_2 \vee cl_2)^{p_2}, \mathcal{I}_2),\\ (x \sim c_1 \vee cl_1)^{p_1}, (x \sim' c_2 \vee cl_2)^{p_2} \vdash_{\text{RR.3,RR.3e}} (cl_1 \vee cl_2)^p,\\ \mathcal{I} = \begin{cases} \mathcal{I}_1 \vee \mathcal{I}_2 & \text{if } x \in V_A \\ \mathcal{I}_1 \wedge \mathcal{I}_2 & \text{if } x \in V_B \\ (x \sim c_1 \vee \mathcal{I}_1) \wedge (x \sim' c_2 \vee \mathcal{I}_2) & \text{if } x \in V_{A,B} \end{cases}\end{array}}{((cl_1 \vee cl_2)^p, \mathcal{I})} \quad \text{(GR.3)}$$

**Lemma 5.2: Generating generalized SSMT interpolants**

Let $\delta = \mathcal{Q} : (A \wedge B)$ with $\mathcal{Q} = Q_1x_1...Q_nx_n$ be some SSMT formula, and the pair $(cl^p, \mathcal{I})$ be derivable from $\delta$ by interpolating SSMT-resolution, where $Q(cl) = Q_1x_1...Q_ix_i$. Then, for each $\tau : V(\varphi) \downarrow_i := \{x_1, ..., x_i\}$ for $i \leq n$ with $\forall x \in V(cl) : \tau(x) = \mathit{ff}_{sp}(x \sim c)$, where $x \sim c \in cl$, it holds that:

1. $V(\mathcal{I}) \subseteq V_{A,B}$,
2. $Pr(Q_{i+1}x_{i+1}...Q_nx_n : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I})[\tau(x_1)/x_1]...[\tau(x_i)/x_i]) = 0$, and
3. $Pr(Q_{i+1}x_{i+1}...Q_nx_n : (\mathcal{I} \wedge B \wedge \neg S_{A,B})[\tau(x_1)/x_1]...[\tau(x_i)/x_i]) = 0$.

**Proof of Generating generalized SSMT interpolants**

We will prove this lemma by induction over the application of SSMT-resolution rules.

**Base case:** We know that in the base case, either GR.1 or GR.2 will be applied.

- For Rule GR.1 and $cl \in A$:
  - The first item holds, as $V(\mathcal{I})$ is empty which is subset of any set.
  - By construction $\tau$ such that $cl$ evaluates to false, then $A[\tau(x_1)/x_1]...[\tau(x_i)/x_i]$ is unsatisfiable. Thus the second item holds.
  - As the clause $cl \in A$, then $\mathcal{I} = \texttt{false}$. Consequently the third item holds.

- For Rule GR.1 and $cl \in B$:
  - The first item holds, as $V(\mathcal{I})$ is empty which is subset of any set.
  - As the clause $cl \in B$, then $\mathcal{I} = \texttt{true}$. Consequently the second item holds.
  - By construction $\tau$ such that $cl$ evaluates to false, then $B[\tau(x_1)/x_1]...[\tau(x_i)/x_i]$ is unsatisfiable. Thus the third item holds.

- For Rule GR.2:
  - The first item holds, as the condition of GR.2 is to build the interpolants over the shard variable; i.e., $V_{A,B}$
  - The second item holds directly as $\neg S_{A,B} \models \texttt{false}$, so $Pr(A \wedge \neg S_{A,B} \wedge \neg \mathcal{I}) = 0$.
  - The third item holds directly as $\neg S_{A,B} \models \texttt{false}$, so $Pr(B \wedge \neg S_{A,B} \wedge \mathcal{I}) = 0$.

**Induction hypothesis:** We assume that the lemma holds for all clauses in the

premises of Rule GR.3. Then by construction of $\mathcal{I}$, the first item of the lemma holds, i.e., $V(\mathcal{I}) \subseteq V_{A,B}$. We assume that

$$Pr(\mathcal{Q}' : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I}_1)[\tau_1(x_1)/x_1]...[\tau_1(x_{j-1})/x_{j-1}][val_a/x_j]) = 0$$

$$Pr(\mathcal{Q}' : (\mathcal{I}_1 \wedge \neg S_{A,B} \wedge B)[\tau_1(x_1)/x_1]...[\tau_1(x_{j-1})/x_{j-1}][val_a/x_j]) = 0$$

holds for $((cl_1 \vee x \neq val_a)^{p_1}, \mathcal{I}_1)$, and

$$Pr(\mathcal{Q}' : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I}_2)[\tau_2(x_1)/x_1]...[\tau_2(x_{j-1})/x_{j-1}][(\mathcal{D}_x \setminus \{val_a\})/x_j]) = 0$$

$$Pr(\mathcal{Q}' : (\mathcal{I}_2 \wedge \neg S_{A,B} \wedge B)[\tau_2(x_1)/x_1]...[\tau_2(x_{j-1})/x_{j-1}][(\mathcal{D}_x \setminus \{val_a\})/x_j]) = 0$$

holds for $((cl_2 \vee x = val_a)^{p_2}, \mathcal{I}_2)$, where $x_j = x$, $j \geq i+1$, $\mathcal{Q}' = \mathcal{Q}_{j+1}x_{j+1}...\mathcal{Q}_n x_n$, and $val_a \in \mathcal{D}_x$.
Let $\tau$ be any assignment that maps the shared variable to intervals (or simple bounds) and $\tau(x) = \tau_1(x)$ if $x \in V(cl_1)$. Likewise, $\tau = \tau_2(x)$ if $x \in V(cl_2)$. Additionally, if $x$ is a shared variable, then $\tau_1 = \tau_2 = \tau$, because $(cl_1 \vee cl_2)$ will appear in the resolution tree that leads to $\emptyset$ if and only if $\not\models (cl_1 \vee cl_2)$.

**Induction step:** We want to prove that

$$Pr_A = Pr(\mathcal{Q}' : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I})[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}]) = 0 \quad \text{(A)}$$

$$Pr_B = Pr(\mathcal{Q}' : (\mathcal{I} \wedge \neg S_{A,B} \wedge B)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}]) = 0 \quad \text{(B)}$$

by showing four cases:

$$\begin{aligned}
Pr_{A,val_a} &= Pr(\mathcal{Q}' : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I})[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][val_a/x_j]) \\
&= 0, \\
Pr_{A,\overline{val_a}} &= Pr(\mathcal{Q}' : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I})[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][(\mathcal{D}_x \setminus \{val_a\})/x_j]) \\
&= 0, \\
Pr_{B,val_a} &= Pr(\mathcal{Q}' : (\mathcal{I} \neg S_{A,B} \wedge B)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][val_a/x_j]) \\
&= 0, \\
Pr_{B,\overline{val_a}} &= Pr(\mathcal{Q}' : (\mathcal{I} \wedge \neg S_{A,B} \wedge B)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][(\mathcal{D}_x \setminus \{val_a\})/x_j]) \\
&= 0.
\end{aligned}$$

In Rule GR.3, we have three different cases:

- Case 1: If $x \in V_A$, then $\mathcal{I} = \mathcal{I}_1 \vee \mathcal{I}_2$. By construction $\tau$, $\mathcal{I}$, and the induction hypothesis,

  1. $Pr_{A,val_a} \leq Pr(\mathcal{Q}' : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I})[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][val_a/x_j])$.
     $Pr_{A,val_a} \leq Pr(\mathcal{Q}' : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I}_1 \wedge \neg \mathcal{I}_2)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}]$ $[val_a/x_j])$.
     $Pr_{A,val_a} \leq Pr(\mathcal{Q}' : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I}_1)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][val_a/x_j])$ $= 0$.

2. $Pr_{A,\overline{val_a}} \leq Pr(\mathcal{Q}^{'} : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I})[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][\overline{val_a}/x_j])$.
   $Pr_{A,\overline{val_a}} \leq Pr(\mathcal{Q}^{'} : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I}_1 \wedge \mathcal{I}_2)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][\overline{val_a}/x_j])$.
   $Pr_{A,\overline{val_a}} \leq Pr(\mathcal{Q}^{'} : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I}_2)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][\overline{val_a}/x_j]) = 0$.

3. Since $x_j \in A$, then $Pr_{B,\overline{val_a}} = Pr_{B,val_a}$ and
   $Pr(\mathcal{Q}^{'} : (\mathcal{I} \wedge B \wedge \neg S_{A,B})[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][val_a/x_j]) = Pr(\mathcal{Q}^{'} : (\mathcal{I} \wedge B \wedge \neg S_{A,B})[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}])$ $Pr_{B,val_a} = Pr(\mathcal{Q}^{'} : (\mathcal{I} \neg S_{A,B} \wedge B)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}])$
   $Pr_{B,val_a} = Pr(\mathcal{Q}^{'} : (\mathcal{I}_1 \vee \mathcal{I}_2 \neg S_{A,B} \wedge B)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}])$
   $Pr_{B,val_a} = Pr(\mathcal{Q}^{'} : (\mathcal{I}_1 \neg S_{A,B} \wedge B) \vee (\mathcal{I}_2 \vee \neg S_{A,B} \wedge B)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}]) = 0$, since $\mathcal{Q}$ is either $\exists$ or $\text{Я}$.

- Case 2: If $x \in V_B$, it follows the same reasoning as the previous case except that this case is for $B$.

- Case 3: If $x \in V_{A,B}$, then the Interpolant $\mathcal{I} = (\mathcal{I}_1 \vee x \neq val_a) \wedge (\mathcal{I}_2 \vee x = val_a)$.

1. $Pr_{A,val_a} \leq Pr(\mathcal{Q}^{'} : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I})[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][val_a/x_j])$
   $Pr_{A,val_a} \leq Pr(\mathcal{Q}^{'} : (A \wedge \neg S_{A,B} \wedge ((\neg \mathcal{I}_1 \wedge x = val_a) \vee (\mathcal{I}_2 \wedge x \neq val_a)))[\tau(x_1)/x_1]\ [\tau(x_{j-1})/x_{j-1}][val_a/x_j])$
   $Pr_{A,val_a} \leq Pr(\mathcal{Q}^{'} : (A \wedge \neg S_{A,B} \wedge ((\neg \mathcal{I}_1 \wedge x = val_a) \vee (A \wedge \neg S_{A,B} \wedge \mathcal{I}_2 \wedge x \neq val_a)))[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][val_a/x_j]) = 0$

2. $Pr_{A,\overline{val_a}} \leq Pr(\mathcal{Q}^{'} : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I})[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][(\mathcal{D}_x \setminus \{val_a\})/x_j])$
   $Pr_{A,val_a} \leq Pr(\mathcal{Q}^{'} : (A \wedge \neg S_{A,B} \wedge ((\neg \mathcal{I}_1 \wedge x = val_a) \vee (\mathcal{I}_2 \wedge x \neq val_a)))[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][(\mathcal{D}_x \setminus \{val_a\})/x_j])$
   $Pr_{A,\overline{val_a}} \leq Pr(\mathcal{Q}^{'} : (A \wedge \neg S_{A,B} \wedge ((\neg \mathcal{I}_1 \wedge x = val_a) \vee (A \wedge \neg S_{A,B} \wedge \mathcal{I}_2 \wedge x \neq val_a)))[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][(\mathcal{D}_x \setminus \{val_a\})/x_j]) = 0$

3. $Pr_{B,val_a} \leq Pr(\mathcal{Q}^{'} : (\mathcal{I} \wedge \neg S_{A,B} \wedge B)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][val_a/x_j])$
   $Pr_{B,\overline{val_a}} \leq Pr(\mathcal{Q}^{'} : (B \wedge \neg S_{A,B} \wedge ((\mathcal{I}_1 \vee x \neq val_a) \wedge (\mathcal{I}_2 \vee x = val_a)))[\tau(x_1)/x_1]...\ [\tau(x_{j-1})/x_{j-1}][val_a/x_j])$
   $Pr_{B,\overline{val_a}} \leq Pr(\mathcal{Q}^{'} : (B \wedge \neg S_{A,B} \wedge ((\mathcal{I}_1 \vee x \neq val_a)) \wedge (B \wedge \neg S_{A,B} \wedge (\mathcal{I}_2 \vee x = val_a)))[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][val_a/x_j]) = 0$

4. $Pr_{B,\overline{val_a}} \leq Pr(\mathcal{Q}^{'} : (\mathcal{I} \wedge \neg S_{A,B} \wedge B)[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][(\mathcal{D}_x \setminus \{val_a\})/x_j])$
   $Pr_{B,\overline{val_a}} \leq Pr(\mathcal{Q}^{'} : (B \wedge \neg S_{A,B} \wedge ((\mathcal{I}_1 \vee x \neq val_a) \wedge (\mathcal{I}_2 \vee x = val_a)))[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][(\mathcal{D}_x \setminus \{val_a\})/x_j])$
   $Pr_{B,\overline{val_a}} \leq Pr(\mathcal{Q}^{'} : (B \wedge \neg S_{A,B} \wedge ((\mathcal{I}_1 \vee x \neq val_a)) \wedge (B \wedge \neg S_{A,B} \wedge (\mathcal{I}_2 \vee x = val_a)))[\tau(x_1)/x_1]...[\tau(x_{j-1})/x_{j-1}][(\mathcal{D}_x \setminus \{val_a\})/x_j]) = 0$

- $j-$th step: From all previous cases, $Pr_{A,val_a}, Pr_{A,\overline{val_a}}, Pr_{B,val_a}$ and $Pr_{B,\overline{val_a}} = 0$ are proven. Now we want to prove that for $j$th quantifier as follows:

  - $\mathcal{Q} = \exists$, then $Pr_A = max(Pr_{A,val_a}, Pr_{A,\overline{val_a}}) = 0$. Likewise $Pr_B = 0$.
  - $\mathcal{Q} = \text{Я}$, then $Pr_A = Pr_{A,val_a} \cdot p_{val_a} + Pr_{A,\overline{val_a}} \cdot p_{\overline{val_a}} = 0$. Likewise $Pr_B = 0$.

- $j+1$-th-step: We need to prove that

$$Pr_A = Pr(Q_{i+1}x_{i+1}...Q_nx_n : (A \wedge \neg S_{A,B} \wedge \neg \mathcal{I})[\tau(x_1)/x_1]...[\tau(x_i)/x_i]) = 0,$$

$$Pr_B = Pr(Q_{i+1}x_{i+1}...Q_nx_n : (\mathcal{I} \wedge \neg S_{A,B} \wedge B)[\tau(x_1)/x_1]...[\tau(x_i)/x_i]) = 0$$

  From the previous proof, if $j = i+1$, then the previous equations directly hold. But if $j > i+1$, then the variables $x_{i+1}...x_{j-1}$ do not occur in the resulting clause $(cl_1 \vee cl_2)$, since the latter clause is quantified till $i$. By the definition of $\tau$ which is restricted to $j-1$, the probabilities of Equation A, and Equation B are zero.
- for case $i+1$, the lemma directly follows.

By using the previous lemma with the relatively complete SSMT resolution calculus, we get the following corollary:

**Corollary 5.2: Generating generalized SSMT interpolants**

If interpolating SSMT resolution derives $(\emptyset^p, \mathcal{I})$ from an SSMT formula $\delta = \mathcal{Q} : (A \wedge B)$, then $\mathcal{I}$ is a generalized Craig interpolant for $(A, B)$ witnessing $Pr(\delta) = p$.

**Proof of Generating generalized SSMT interpolants**

(sketch): The proof of that corollary follows these facts:

- If $Pr(\mathcal{Q} : \varphi) \leq p < 1$ for some SSMT formula $\delta := \mathcal{Q} : \varphi$, then the empty clause, i.e., $\emptyset^p$ is derivable by SSMT-resolution (Theorem 5.1).
- If $(\emptyset^p, \mathcal{I})$ is derived by SSMT interpolating, then:
  - $V(\mathcal{I}) \subseteq V_{A,B}$,
  - $Pr(A \wedge \neg S_{A,B} \wedge \neg \mathcal{I}) = 0$, and
  - $Pr(\mathcal{I} \wedge B \wedge \neg S_{A,B}) = 0$.

  by using Lemma 5.2.

Thus, $\mathcal{I}$ is a generalized Craig interpolant for $(A, B)$ witnessing $Pr(\delta) = p$.

**Corollary 5.3: Controlling strength of SSMT interplants**

If $\mathcal{I} = \texttt{true}$ is used within each application of Rule GR.2, then $Pr(\mathcal{Q} : (A \wedge \neg \mathcal{I})) = 0$. Likewise, if $\mathcal{I} = \texttt{false}$ is used within each application of Rule GR.2, then $Pr(\mathcal{Q} : (\mathcal{I} \wedge B)) = 0$.

**Proof of Controlling strength of SSMT interpolants**

In order to prove this corollary, let us consider the induction proof.
**Base case:** For RR.1, it is totally independent from affecting $S_{A,B}$. For RR.2, if $\mathcal{I}$

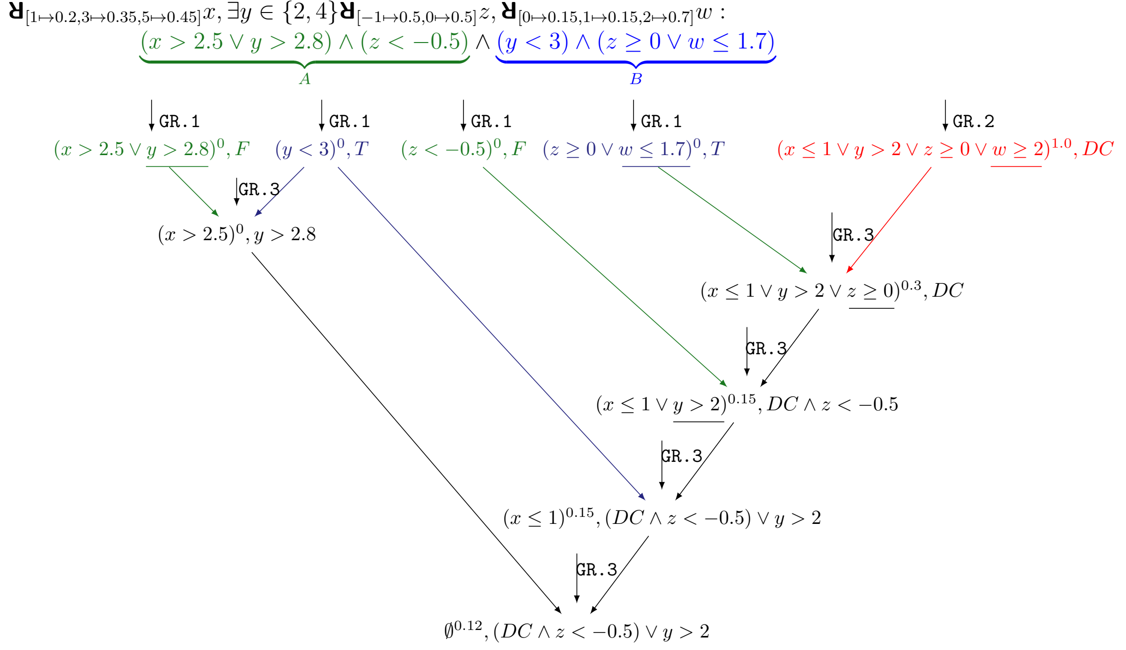



Figure 5.4: Generalized Craig interpolant for Example 5.1. The green part is $A$ and the blue one is $B$. The red part represents $\neg S_{A,B}$ with a don't-care interpolant.

is `true`, then $Pr(A \wedge \neg \mathcal{I}) = 0$. Likewise, if $\mathcal{I}$ is `false`, then $Pr(\mathcal{I} \wedge B) = 0$.
**Induction hypothesis:** We assume for $\mathcal{I} = \texttt{true}$ that

$$Pr(\mathcal{Q}' : (A \wedge \neg \mathcal{I}_1)[\tau(x_1)/x_1]...[val_a/x_j]) = 0 \text{ and}$$

$$Pr(\mathcal{Q}' : (A \wedge \neg \mathcal{I}_2)[\tau(x_1)/x_1]...[(\mathcal{D}_x \setminus \{val_a\}/x_j]) = 0.$$

Additionally, we assume for $\mathcal{I} = \texttt{false}$ that

$$Pr(\mathcal{Q}' : (\mathcal{I}_1 \wedge B)[\tau(x_1)/x_1]...[val_a/x_j]) = 0 \text{ and}$$

$$Pr(\mathcal{Q}' : (\mathcal{I}_2 \wedge B)[\tau(x_1)/x_1]...[(\mathcal{D}_x \setminus \{val_a\}/x_j]) = 0.$$

**Induction step:** It follows the previous proof of Lemma 5.2.

### Continue with Example 5.2 to compute Generalized Craig Interpolant

In order to get the idea of computing the Craig interpolants for SSMT problems, let us consider the following formula in Example 5.2:
$Я_{[1\mapsto 0.2, 3\mapsto 0.35, 5\mapsto 0.45]} x,\ \exists y \in \{2, 4\}\ Я_{[-1\mapsto 0.5, 0\mapsto 0.5]}, z\ Я_{[0\mapsto 0.15, 1\mapsto 0.15, 2\mapsto 0.7]} w : A \wedge B$
where $A = (z < -0.5) \wedge (x > 2.5 \vee y > 2.8)$ and $B = (y < 3) \wedge (z \geq 0 \vee w \leq 1.7)$
Figure 5.4 shows formally how the generalized Craig interpolant is computed. `DC` stands for a *don't care*-formula which can replaced by `true` or `false`, a.o. If we replace `DC` with `true`, then the interpolant becomes $z < -0.5 \vee y > 2$ which is implied by $A$. Likewise, if it is replaced by `false`, then the resulting interpolant

$y > 2$ implies the negation of $B$ as in Corollary 5.3.

In the previous example in Figure 5.4, one may compute more interpolants with different strength in case that one relaxes the simple bounds in any direction, but without changing their interpretations in the original formula. For example a weaker interpolant that can generated by the resolution tree in Figure 5.4 is $\mathcal{I}_1 = (\mathtt{DC} \wedge z < 0) \vee y > 2$. On the other hand, a stronger interpolant can be computed by the same resolution tree, is $\mathcal{I}_2 = (\mathtt{DC} \wedge z \leq -0.5) \vee y \geq 4$, where $\mathcal{I}_2 \rightarrow \mathcal{I}_1$ and both $\mathcal{I}_1$ and $\mathcal{I}_2$ are generalized Craig interpolants.

## 5.5 Interpolation-based probabilistic bounded model checking

In this section, we demonstrate an application of generalized Craig interpolation to quantitative model-checking of probabilistic hybrid automata, namely probabilistic bounded model checking (PBMC). Probabilistic hybrid automata (PHA) are Markov decision processes (MDPs) over infinite state space, with arithmetic-logical transition guards and actions. It is defined formally as follows:

**Definition 5.5: Probabilistic hybrid automaton [FHT08]**

A discrete-time probabilistic hybrid automaton

$$PHA = (L^{PHA}, X^{PHA}, E^{PHA}, \ell^{PHA}_{ini})$$

where
- $L^{PHA}$ is a finite set of discrete locations,
- $X^{PHA}$ is a space of continuous states components,
- $E^{PHA} \subseteq L^{PHA} \times \Phi(V^{PHA}) \times P^{PHA} \times R(X^{PHA})^* \times L^{PHA}$ is a finite set of directed transitions, where
  - $\Phi(X^{PHA})$ is a set of arithmetic constraints in our arithmetic theory $\mathcal{T}$ with free variables in $X^{PHA}$,
  - $P^{PHA}$ assigning to each transition a *positive* probability distribution over the target locations, and
  - $R(X^{PHA})^*$ is a set of assignments defined by means of a $\mathcal{T}$-predicate over variables in $X^{PHA}$ and $X^{PHA'}$ denotes primed variants of the state components in $X^{PHA}$ (a.k.a. *set of reset operations*).

  An element $(\ell^{PHA}, \varphi^{PHA}, p, \vec{r}, \ell^{PHA'}) \in E^{PHA}$ describes a transition with probability $p$ from location $\ell^{PHA}$ to $\ell^{PHA'}$ with guard $\varphi^{PHA}$ and assignments $\vec{r}$, and
- $\ell^{PHA}_{ini} \subseteq L^{PHA} \times X^{PHA}$ is a set initial state predicates, For technical reasons, we demand that for each $\ell \in L^{PHA}$, there is at most one $x \in X^{PHA} \rightarrow \mathbb{R}$ which satisfies the predicate $\ell^{PHA}_{Init}$.

is an automaton $\mathcal{A}_{PHA}$ as in Definition 3.1 where
- $Loc := L^{PHA}$,
- $Act := \Phi(X^{PHA}) \times P^{PHA} \times R(X^{PHA})^*$,
- $E := \{(\ell, (\varphi^{PHA}, p, \vec{r}), \ell') \mid (\ell^{PHA}, \varphi^{PHA}, p, \vec{r}, \ell^{PHA'}) \in E^{PHA}\}$, and

- $L_{ini} := \ell_{ini}^{PHA}$.

Since we are interested in the probability of reaching a given set of undesirable locations within a given number of steps (say $k$). In the first step, we need to consider a particular policy (scheduler, adversary) that resolves the non-determinism introduced by $PHA$. As this problem is undecidable due to employed $\mathcal{T}$-arithmetic theory (cf. Section 4.3), the worst-case has to be considered, i.e., maximum probability of reaching the unsafe states achieved under any arbitrary policy that may resolve non-determinism using randomization, the history, etc. As introduced in [FHT08], we define the probability of reaching some target states within $k$ steps directly.

### 5.5.1 Probabilistic bounded reachability – probabilistic safety analysis

In order to solve the decision problem, the probability that a probabilistic hybrid system model $\mathcal{M}$ reaches bad states *target* is less than or equal a certain threshold $\vartheta$, i.e.

$$MaxReach(\mathcal{M}, target) \leq \vartheta \tag{5.1}$$

one has to consider one of the following two approaches:

- The first approach is to maximize (overapproximate) the probability of reaching bad states (which often is defined as *target*) under each possible scheduler (beginning from the initial states *init*) and assuring that the latter probability is under the threshold point (cf. [FHT08]):

$$MaxReach(\mathcal{M}, target) := \lim_{k\to\infty} MaxReach^{k}_{\mathcal{M},target}(init) \tag{5.2}$$

  and

$$MaxReach^{k}_{\mathcal{M},target}(\ell) = \begin{cases} 1 & \text{if } \ell \in target \\ 0 & \text{if } \ell \notin target \text{ and } k = 0 \\ \max_{t\in Enabled} \sum_{\ell'\in Loc} p(t) \cdot MaxReach^{k-1}_{\mathcal{M},target}(\ell') & \\ & \text{if } \ell \notin target \text{ and } k > 0 \end{cases} \tag{5.3}$$

  where *Enabled* refers to transitions that have a source $\ell$, a destination $\ell'$, and their guards are satisfied.

- The second approach is to minimize the probability $p'$ of staying in safe states (all system states without target) and assure that the complement of the latter probability $1 - p'$ will not exceed $\vartheta$; i.e., $1 - p' \leq \vartheta$.

In the sequel of this chapter, the first approach, namely the maximum probability of reaching bad states as introduced in Scheme 5.1 will be taken in our consideration.

### 5.5.2 SSMT encoding scheme for PHAs

Consider that there are some given set of target states in the PHA model, and we try to maximize the probability of reaching these states over all policies resolving the non-determinism in the PHA model. The first step is to encode the PHA into SSMT formulae. The encoding pioneered in [FHT08] directly applies to PHA as in Definition 5.5 capturing continuous dynamics by pre-post relations. Fränzle et al. proposed a straightforward reducing of PBMC to SSMT formulae, where this reduction is proven to be correct (for more details, see [FHT08], Proposition 1). Concisely, it works as follows.

- we generate the matrix of the SSMT formula $\delta$. This matrix is an SMT formula $\varphi$ encoding all runs of *PHA* of the given length $k \in \mathbb{N}_0$ (cf. Section 4.3 where iSAT3 encodes hybrid systems problems).
- we add the quantifier prefix encoding the probabilistic and the non-deterministic choices, whereby a probabilistic choice reduces to a randomized quantifier (ꓤ) while a non-deterministic choice yields an existential quantifier ($\exists$).
- the initial states and target states are encoded by predicates.

### 5.5.3 PBMC solving by means of generalized Craig interpolation

In this subsection, we propose a symbolic verification procedure for above Scheme 5.1 by means of generalized Craig interpolation for SSMT.

Our proposed technique integrates SSMT encoding of PHA as proposed in [FHT08] with interpolation-based probabilistic bounded model checking problem for SSAT problem, introduced by Teige et al. [TF12b, Tei12].

**PHA normalizing or abstracting.** According to the definition of PHA, it may contain ordinary differential equations (ODEs) attributing the dynamics of the model, thus one has to deal with this problem before computing PBMC by GCI.

One feasible solution is to add ICP for ODE, as suggested in [EFH08], where interval-based safe numeric approximation of ODE is used as an interval contractor being able to narrow candidate sets in phase space of reachable sets. From a technical viewpoint, it permits the embedding of interval-based safe numeric approximation of ODE images and ODE pre-images as a further rule for theory propagation during SMT solving. This approach was integrated into SSMT solving in [TEF11] while solving a network of automation systems.

Another idea is to resort to abstraction of ODE into pre-post relations by tools like PHAVer [Fre08], as pursued in ProHVer [ZSR+10, FHH+11]. However, this approach relaxes the problem and obtains a coarse finite-state abstraction which may not fulfil the stochastic requirements without several refinements.

In both cases, as we deal with safe overapproximation either in ICP with ODE or after finite-state abstraction, one has to assure that we can use both preprocessing approaches to prove safety only. That is, if we get a counterexample or find that the maximum

probability lies above the threshold, it is essential to refine the abstract model, in order to imitate the CEGAR procedure in excluding bogus counterexamples.

**Problem encoding.** We compute a symbolic representation of an overapproximation of the backward reachable state set, where a state is *backward reachable* if it is the origin of a transition sequence leading into target. This can be integrated into PBMC, as used to *falsify* the probabilistic safety property. Whenever such falsification fails for a given step depth $k$, we apply generalized Craig interpolation to the PBMC proof to compute a symbolic overapproximation of the backward reachable state set at depth $k$ and then proceed to PBMC at some higher depth $k' > k$.

As an alternative to the integration into PBMC, interpolants describing the backward reachable state sets can be extended by "stepping" them by concatenating another transition, as explained below. In either case, we continue until the backward reachable state set becomes stable, in which case we have computed a symbolic overapproximation of the whole backward reachable state set[3].

Assume that we are given PHA model $\mathcal{M}$, a predicate *target* in CNF that encodes the target states. Then, the state-set predicate $\mathcal{B}^k(x)$ for $k \in \mathbb{N}_0$ over state variables $x$ is inductively defined as:

- $\mathcal{B}^0(x) = target(x)$,
- $\mathcal{B}^{k+1}(x) = \mathcal{B}^k(x) \vee \mathcal{I}^{k+1}(x)$, and
- $$\mathcal{I}^{k+1}(x_{j-1}) = \left( \underbrace{TRANS(x_{j-1}, x_j) \wedge \mathcal{B}^k(x_j)}_{=A}, \underbrace{INIT(x_0) \wedge \bigwedge_{i=1}^{j-1} TRANS(x_i, x_{i+1})}_{=B} \right) \tag{5.4}$$

  where $\mathcal{Q} : A \wedge B$ is an SSMT formula encodes the problem till depth $k$. The quantifier prefix of SSMT formula has the following form:

  $$\mathcal{Q} : Q_0 ... Q_k \exists x_0 ... \exists x_k$$

  where $Q_0 ... Q_k$ are either existential or randomized quantifiers encoding the branching choices in PHA. However $\exists x_0$ to $\exists x_k$ are innermost existential quantifiers encoding

[3] As introduced in Chapter 2, safety analysis of a given system seeks to discover whether the mathematical model representing the system can enter a specified set of unsafe states [Mit07]. For solving such probabilistic safety problems, one can apply either backward or forward analysis approaches. *In backward reachability analysis*, we begin from the target state in order to overapproximate backward reachable state sets. We continue until the set of reachable states stabilizes (no more states added). Then we verify, if the probability of reaching bad states is less than or equal a certain probability. On the other hand, in *forward reachability analysis*, we begin from the initial set of states and discover one step further in order to make overapproximation of reachable states from the initial states. We continue with this overapproximation, until we reach the stabilization situation, then we verify if the bad states are reached with less probability than the certain given probability. The choice to use backward or forward analysis depends on the problem itself; in some problems using backward analysis is more efficient than forward analysis and vice verse in other situations.

the continuous-domain state variables in SSMT.

$\mathcal{I}^{k+1}(x_{j-1})$ can be computed by interpolating SSMT-resolution rules in case that the backward reachable-set $\mathcal{B}^k(x_j)$ is rewritten into CNF as introduced in Section 4.3 by using the Tseitin transformation. Since GR.2 depends mainly on a *don't care*-interpolant, we will substitute $\mathcal{I}$ with `true` in in every application of rule RR.2 such that $\mathcal{B}^k(x)$ is guaranteed to overapproximate all system states which are backward reachable from the target states within $k$ steps (cf. Corollary 5.3). Whenever $\mathcal{B}^k(x)$ has reached a fixed point, i.e. if

$$\mathcal{B}^{k+1}(x) \rightarrow \mathcal{B}^k(x)$$

holds for some $k$, then $\mathcal{B}(x) := \mathcal{B}^k(x)$ and it overapproximates all backward reachable states. The parameter $j \geq 1$ in Scheme 5.4 can be chosen freely, i.e., the system may execute any number of transitions until state $x_{j-1}$ is reached, since this does not destroy the "backward-overapproximating" property of $\mathcal{B}^{k+1}(x)$. Hence, $j$ influences the shape of $\mathcal{I}^{k+1}(x)$ during computation. As long as we increase $j$, we obtain more precise results but more complex models as well (cf. Appendix A).

**SSMT formula building.** We construct an SSMT formula with parameter $k$ that represents the behaviour of the system such that the latter stays within the backward reachable state set for $k$ steps. The maximum satisfaction probability of that SSMT formula gives an upper bound on the maximum probability of reaching the target states (representing bad states). This can be computed by the following equation:

$$ub_k = Pr\left(\mathcal{Q}_k : \left(\underbrace{INIT(x_0) \wedge \bigwedge_{i=1}^{k} TRANS(x_{i-1}, x_i)}_{reachble\ states\ within\ k} \wedge \overbrace{\bigwedge_{i=0}^{k} \mathcal{B}^k(x_i)}^{backwards\ reachble\text{-}set\ of\ states}\right)\right) \tag{5.5}$$

In Scheme 5.5, the upper bound probability is computed for $k$ steps, without however considering all system runs that leave the set of backward reachable states, since the latter runs do not reach the target states.

### 5.5.4 Interpolation-based approach for reachability

In order to use generalized interpolation in unbounded probabilistic model checking, one needs to *encode* the model's transition relations by an SMT representation. Then one has to generate a probabilistic bounded model checking problem (PBMC) in SSMT [FHT08] and determine whether the targets are reachable with probability exceeding the safety target within some step bound $k$. Should this not be the case, one can use generalized Craig interpolation to compute an *overapproximation of the states backward reachable*[4] from the targets within that step bound. Technically, we interpolate between the initial state predicate and the $k$-fold iteration of the transition relation plus the target predicate, albeit under quantification as explained in the previous subsection. PBMC is iterated for

[4]One can use overapproximation of the states forward reachable as well, however by exchanging the formulae $A$ and $B$ accordingly.

increasingly larger $k$ until either the safety property is falsified or the generalized Craig interpolant stabilizes, i.e., a superset of all states backward reachable from the target has been computed.

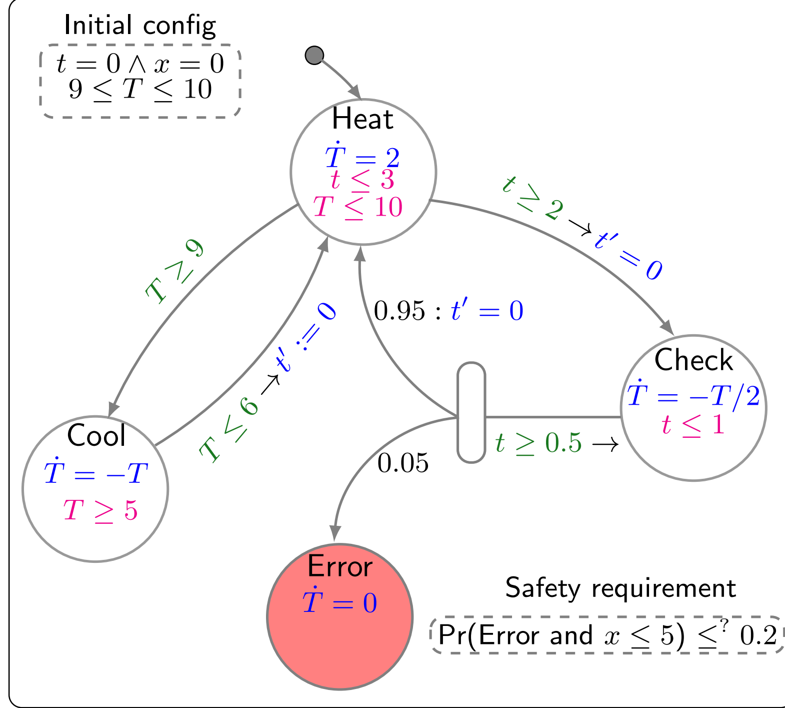


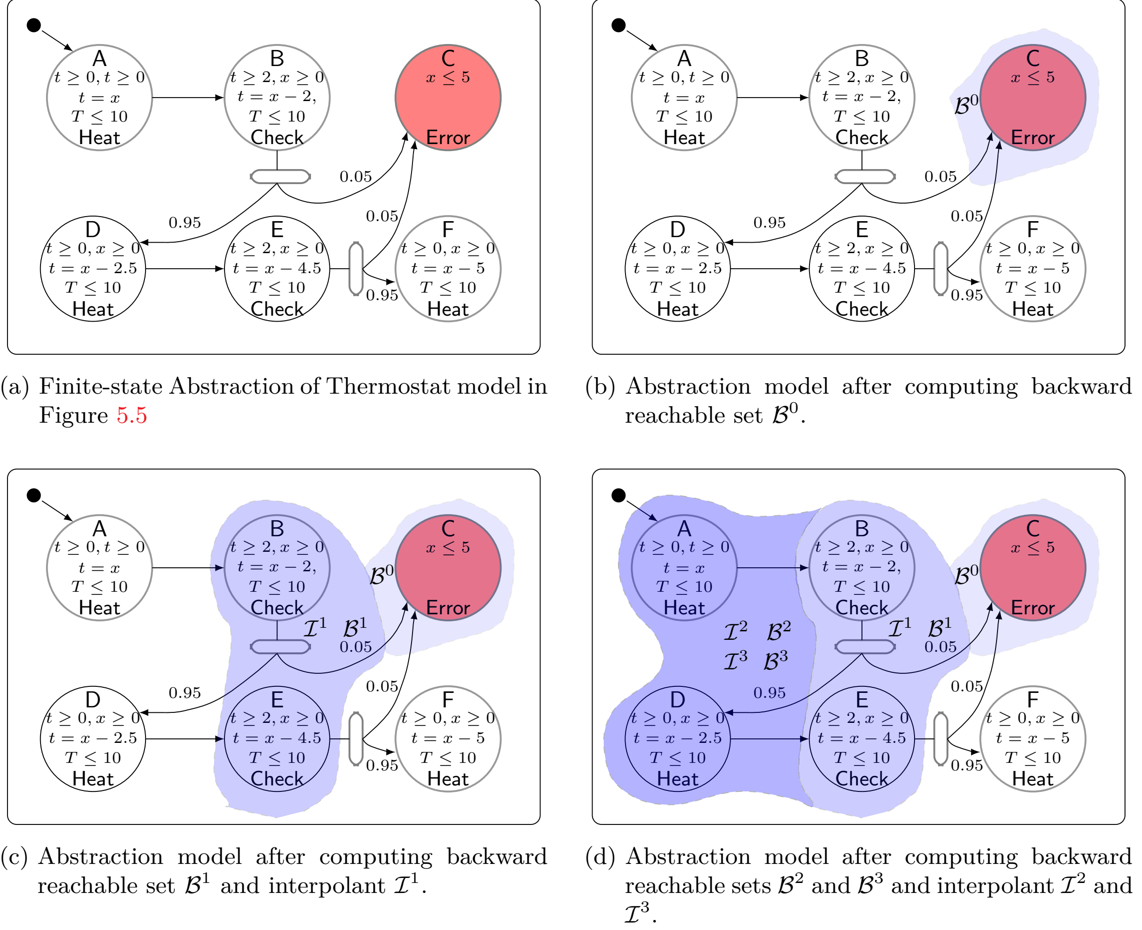

Figure 5.5: Thermostat case-study discussed in [ZSR+10, FHH+11]. Blue expressions represent the assignments, green ones represent the guards and the magenta ones represent the invariants at each location.

| $j$ | $\mathcal{I}^1$ | $\mathcal{B}^1$ | $\mathcal{I}^2$ | $\mathcal{B}^2$ | $\mathcal{I}^3$ | $\mathcal{B}^3$ | $\mathcal{B}$ |
|---|---|---|---|---|---|---|---|
| 1 | $\neg A$ | $\neg A \vee (C \wedge x \leq 5)$ | `true` | `true` | `true` | `true` | `true` |
| 2 | $\neg F$ | $\neg F \vee (C \wedge x \leq 5)$ | $\neg F \vee (C \wedge x \leq 5)$ | $\neg F \vee (C \wedge x \leq 5)$ | – | – | $\neg F \vee (C \wedge x \leq 5)$ |
| 3 | $\neg A \wedge \neg D \wedge \neg F$ | $(\neg A \wedge \neg D \wedge \neg F) \vee (C \wedge x \leq 5)$ | $\neg F$ | $\neg F$ | $\neg F$ | $\neg F$ | $\neg F$ |

Table 5.1: Results of interpolation-based approach of Example 5.3, where $j$ represents the number of the transitions considered by the interpolation, $\mathcal{I}$ represents the interpolant computed at $j$-th step, and $\mathcal{B}$ represents the backward reachable states.

**Example 5.3: Thermostat case study [ZSR+10]**

Let us consider the PHA of Figure 5.5 modelling a thermostat system. Having continuous-dynamics in this model drives us to use ProHVer to obtain a safe abstraction which is depicted in Figure 5.6a (cf. Subsection 6.2.4).
Now, we would like to verify whether *the maximum probability to reach the location* `Error` *within 5 time units is at most* $\frac{1}{5}$ or not.
Note that the property is expressed in terms of *time units* rather than computation steps. As there is *no immediate correspondence between time units and computation*




(a) Finite-state Abstraction of Thermostat model in Figure 5.5

(b) Abstraction model after computing backward reachable set $\mathcal{B}^0$.

(c) Abstraction model after computing backward reachable set $\mathcal{B}^1$ and interpolant $\mathcal{I}^1$.

(d) Abstraction model after computing backward reachable sets $\mathcal{B}^2$ and $\mathcal{B}^3$ and interpolant $\mathcal{I}^2$ and $\mathcal{I}^3$.

Figure 5.6: Illustration of computed backward reachable sets together with generalized Craig interpolants to compute the maximum probability of reaching Error state over number $k$ of transition steps.

*steps*, this verification problem cannot be solved by PBMC, since PBMC computes the lower bound of reaching Error state. Thus, it requires unbounded reachability computation by GCI.

In the abstract model, the probability to reach the error states within 5 time units is 0.0975, which is less than $\frac{1}{5}$ and thus acceptable. To determine this probability, we encode the abstraction of the thermostat as an SSMT formula and then compute overapproximations of the backward reachable states incrementally by GCI until it stabilizes. The target is `C-Error` which cannot be reached from the initial `A-Heat` via a single transition. In the first interpolation, the target `C-Error` together with a single transition relation represents the $A$ part, while the initial state predicate `A-Heat` constitutes $B$. The first computed interpolant (while $j = 1$ in Scheme 5.4) will thus equal all states except the initial one, providing a useless upper bound of 1 on the probability of eventually hitting the target. Successive interpolations (with $j$ larger than 1 as in Table 5.1) for larger step numbers yield tighter approximations.

For example, when $j = 2$, we get the first overapproximated backward reachable-set equals to $\neg F$ (means $A \vee B \vee C \vee D \vee E$), which is not so precise since $A$ and $D$ are reachable after two steps. However, if we increase $j$ to be 3, then we get in the first overapproximation a more precise backward reachable-set, namely $\neg A \wedge \neg D \wedge \neg F$ (means $B \vee C \vee E$) which emphasises our aforementioned observation that whenever $j$ increases, one get more realistic overapproximation. In the latter case, it is noticed that the interpolant stabilizes after 2 steps. This result was used while computing the upper bound probability of reaching Error state in SiSAT.
In this model, the interpolant stabilizes after three iterations and yields a tight enough overapproximation of the backward reachable state set (cf. Appendix A for more details). As aforementioned, one can in each step use interpolants for computing an upper approximation of the (unbounded) reachability probability, while PBMC yields a valid lower approximation. Figure 5.9 represents three results: the upper (red) curve represents the upper bound on the step-unbounded probability to reach location `Error` within 5 time units, as computed by GCI.
The numbers on the horizontal axis here refer to the iteration (the number of steps), while the vertical axis refers to the computed probabilities. The middle (green) line represents the exact probability to reach location `Error` within 5 time units. The lower (blue) curve represents the lower bound on the probability to reach an `Error` state within 5 time units, as computed by PBMC. One may observe that upper and lower bounds almost coincide after step $k = 4$. In fact, interpolation then tells us that the reachability probability is below 0.1, i.e., well below the safety target. All details of computing interpolants for $j = 1$ and $j = 2$ are depicted in Appendix A.

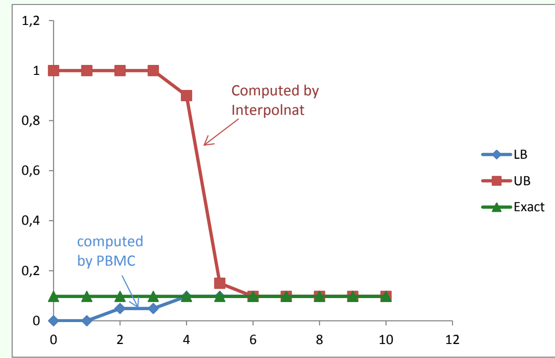


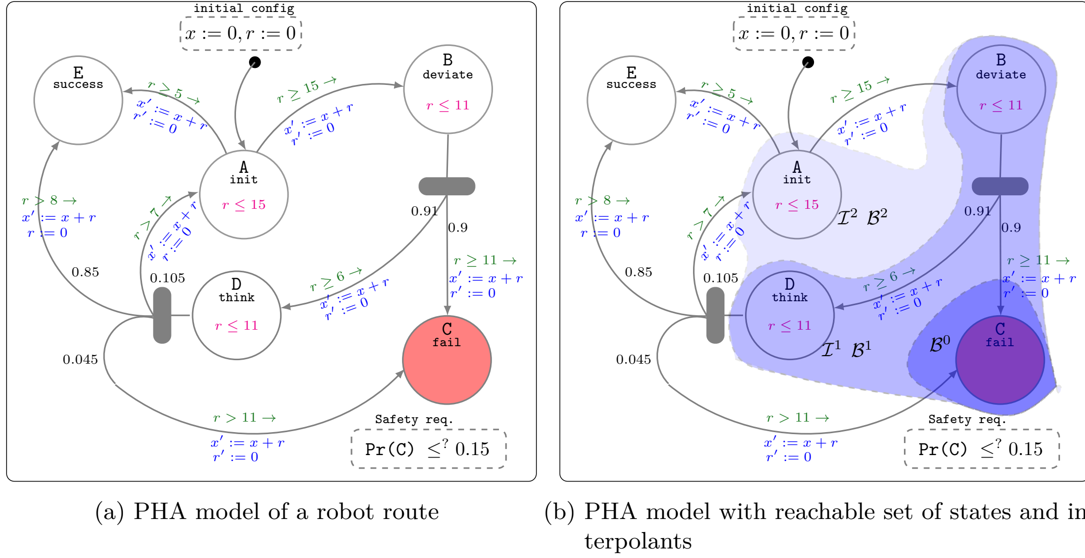

Figure 5.7: Probability of reaching Error within 5 time units once by using PBMC and once by using GCI.

## Remark 5.1: Using ProHVer

The main reason to use ProHVer comes from the limitation to integrate our stochastic resolution tool with SiSAT tool. Thus, we encode the problem back and forth between SiSAT and the resolver manually. Finally, if one can integrate the resolver in SiSAT, then it is much better not to use any abstraction, but direct encoding in SiSAT will solve these problems in non-linear PHA efficiently (cf. Subsection 6.2.4).

## Example 5.4: Action planning

In Figure 5.8a, we have another example of probabilistic hybrid automaton where continuous dynamics at each state is represented by linear ordinary differential equation i.e. $\dot{r} := 1$. It represents a behaviour of a robot (e.g., rescue robot) such




(a) PHA model of a robot route

(b) PHA model with reachable set of states and interpolants

Figure 5.8: PHA model represents action planning of a robot, where fail state represents unwanted behaviour.

| $j$ | $\mathcal{I}^1$ | $\mathcal{B}^1$ | $\mathcal{I}^2$ | $\mathcal{B}^2$ | $\mathcal{I}^3$ | $\mathcal{B}^3$ | $\mathcal{B}$ |
|---|---|---|---|---|---|---|---|
| 1 | $\neg A$ | $\neg A \vee C$ | true | true | true | true | true |
| 2 | $\neg E$ | $\neg E \vee C$ | $\neg E \vee C$ | $\neg E \vee C$ | – | – | $\neg E \vee C$ |
| 3 | $\neg E \wedge \neg A$ | $(\neg E \wedge \neg A) \vee C$ | $\neg E$ | $\neg E \vee C$ | $\neg E$ | $\neg E \vee C$ | $\neg E \vee C$ |

Table 5.2: Results of interpolation-based approach of Example 5.4, where $j$ represents the number of the transitions considered by the interpolation to increase the preciseness, $\mathcal{I}$ represents the interpolant computed at $j$-th step, and $\mathcal{B}$ represents the backward reachable states.

that it begins from `init` location. After certain steps (transitions) it can either eventually end with `success` state (right route) or with `fail` state (bad route). From initial state it can non-deterministically either directly go to `success` (the right direction) or go to `deviate` state. If the latter choice was the case, then the robot can either with a probability 0.09 go to `fail` state or with a probability 0.91 go to a situation to decide (`think` state). After that, from `think` state it can go probabilistically either to `success` or to the initial situation or to `fail` state. Now, we want to *verify that over all policies the property that the robot will reach* fail *is less than or equal 0.15*. This property is unbounded property, where GCI can compute it efficiently.

We will apply the same procedure as done before. Namely, we encode the model as an SSMT formula. We compute the interpolant for the transition system while $j = 1, 2, \ldots$ as performed in thermostat case study, until either the interpolant stabilizes or the safety property is violated.

We summarize the results obtained by our prototypical tool in Table 5.2, where different sizes of transition system were used during interpolating, namely $j = 1, 2$ and 3. We observe that the interpolant stabilizes in case of $j \geq 2$, where it overapproximates the reachable states; i.e, $C \vee \neg E$. Figure 5.9 represents two results: the upper (red) curve represents the upper bound on the step-unbounded probability to reach location `fail`, as computed by GCI. The numbers on the horizontal axis here refer to the iteration (the number of steps), while the vertical axis refers to the computed probabilities. The lower (blue) curve represents the lower bound on the probability to reach an `fail` state, as computed by PBMC. One may observe that upper and lower bounds almost coincide after step $k = 11$.

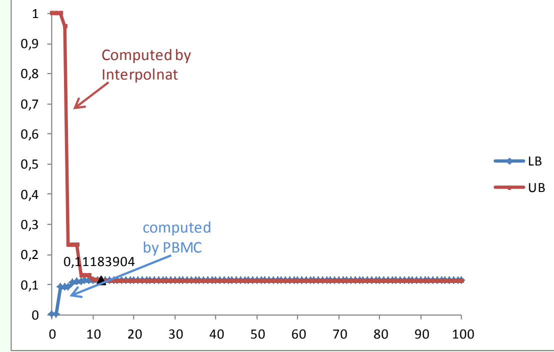



Figure 5.9: Probability of reaching fail once by using PBMC and once by using GCI.

## 5.5.5 Generalized Craig interpolation for Stability analysis

With the same idea of what Teige et al. [Tei12] presented of using interpolation in probabilistic finite-state models and akin to what is stated in probabilistic reachability analysis, one can apply generalized Craig interpolation for *interpolation-based probabilistic region stability* in probabilistic hybrid automata models. *Region stability* in deterministic settings for non-probabilistic models refers to the possibility of proving that the system is stable with respect to *a set of states* a.k.a. region $\mathcal{R}$, if and only if for every valid infinite run of the system, the system eventually globally stays in $\mathcal{R}$.

In order to verify region stability in stochastic settings, it is necessary to adjust the previous concept to our context. Let us consider that we are given a probabilistic hybrid automaton model $\mathcal{M}$ with discrete time steps and a region $\mathcal{R}$ defined over the set of states of $\mathcal{M}$. Probabilistic region stability means that under any possible behaviour, i.e. independent of the non-deterministic and probabilistic choices the system will execute, the probabilistic hybrid automaton model $\mathcal{M}$ eventually globally remains in *region*. That is, $\mathcal{R}$ is a proper overapproximation of possible reachable set of states in $\mathcal{M}$. Thus, we can not speak now about the maximum probability of reaching this region – since it does not represent unwanted behaviour –, but the minimum probability under any scheduler such that the model reaches $\mathcal{R}$. The probability measure is then defined by the minimum probability of reaching the maximal invariance kernel which is a proper subset of $\mathcal{R}$.

$$MinStable(\mathcal{M}, \mathcal{R}) \geq \vartheta \tag{5.6}$$

In order to compute Scheme 5.6 properly:

- we need to minimize the probability of reaching $\mathcal{R}$ under each possible scheduler (beginning from the initial states *init*) and assuring that the latter probability is at

least as the threshold point:

$$MinStable(\mathcal{M}, \mathcal{R}) := \lim_{k\to\infty} MinReach^k_{\mathcal{M},\mathcal{R}}(init) \tag{5.7}$$

and

$$MinReach^k_{\mathcal{M},\mathcal{R}}(\ell) = \begin{cases} 1 & \text{if } \ell \in \mathcal{R} \\ 0 & \text{if } \ell \notin \mathcal{R} \text{ and } k = 0 \\ \min_{t\in Enabled} \sum_{\ell'\in Loc} p(t) \cdot MinReach^{k-1}_{\mathcal{M},\mathcal{R}}(\ell') & \\ & \text{if } \ell \notin \mathcal{R} \text{ and } k > 0 \end{cases} \tag{5.8}$$

where *Enabled* refers to transitions that have a source $\ell$, a destination $\ell'$, and their guards are satisfied.

- considering stabilization within $\mathcal{R}$ as a desired-property allows us to establish the probability of stabilizing in the worst scenario, i.e. under an optimal opposing scheduler, namely whether $MinReach(\mathcal{M}, \mathcal{R}) \geq \vartheta$ holds or not.

$MinReach(\mathcal{M}, \mathcal{R}) \geq \vartheta$ can be addressed in terms of the complement of the maximum probability of avoiding the *region* too, i.e.

$$MinStable(\mathcal{M}, \mathcal{R}) = 1 - MaxAvoid(\mathcal{M}, \mathcal{R}) = 1 - \lim_{k\to\infty} MaxAvoid^k_{\mathcal{M},\mathcal{R}}(init) \tag{5.9}$$

where

$$MaxAvoid^k_{\mathcal{M},\mathcal{R}}(\ell) = \begin{cases} 1 & \text{if } \ell \in \mathcal{R} \\ 0 & \text{if } \ell \notin \mathcal{R} \text{ and } k = 0 \\ \max_{t\in Enabled} \sum_{\ell'\in Loc} p(t) \cdot MaxAvoid^{k-1}_{\mathcal{M},\mathcal{R}}(\ell') & \\ & \text{if } \ell \notin \mathcal{R} \text{ and } k > 0 \end{cases} \tag{5.10}$$

Teige et al. [Tei12] used the last scheme, Scheme 5.9, to compute the minimum probability of staying in $\mathcal{R}$ in MDPs, where the encoding is done in terms of SSAT formulae.

Now, using generalized Craig interpolation together with Scheme 5.9 or Scheme 5.6 is straightforward, where the same procedure is applied, with typical steps used in reachability in Subsection 5.6.

| $j$ | $\mathcal{I}^1$ | $\mathcal{R}^1$ | $\mathcal{I}^2$ | $\mathcal{R}^2$ | $\mathcal{R}$ |
|---|---|---|---|---|---|
| 1 | `true` | `false` | `true` | `false` | `false` |
| 2 | $B \vee D$ | $\neg C \wedge x \leq 7 \wedge \neg B \wedge \neg D$ | $B \vee D$ | $\neg C \wedge x \leq 7 \wedge \neg B \wedge \neg D$ | $\neg C \wedge x \leq 7 \wedge \neg B \wedge \neg D$ |
| 3 | $B \vee D$ | $\neg C \wedge x \leq 7 \wedge \neg B \wedge \neg D$ | $B \vee D$ | $\neg C \wedge x \leq 7 \wedge \neg B \wedge \neg D$ | $\neg C \wedge x \leq 7 \wedge \neg B \wedge \neg D$ |

Table 5.3: Results of interpolation-based approach of Example 5.5, where $j$ represents the number of the transitions considered by the interpolation to increase the preciseness, $\mathcal{I}$ represents the interpolant computed at $j$-th step, and $\mathcal{R}$ represents an overapproximation of possible reachable set of states in $\mathcal{M}$.

### Example 5.5: Action planning: stability problem

In this example, we consider the stability problem for the action planning example depicted in Figure 5.8. We want to verify by using GCI the maximum probability of avoiding the region representing "fail state and $x$ is less than or equal 7" is 0.8. However, we use the Scheme 5.6 to prove that the minimum probability of staying outside the region representing "fail state and $x$ is less than or equal 7" is 0.8. This can be achieved by using GCI as follows:

- We have $\mathcal{R}^0 := \neg C \wedge x \leq 7$. After that we compute $\mathcal{I}_1 := B \vee D$ as shown in interpolation based reachability procedure.
- Now, we compute $\mathcal{R}^1$ by considering $\mathcal{R}^0 \wedge \neg \mathcal{I}_1$.
- We continue as before until $\mathcal{R}$ reaches a fixed point i.e. once it is entered, the system cannot leave it. Formally, it means that $\mathcal{R}^{k+1} \rightarrow \mathcal{R}^k$.

We summarize the results obtained by our prototypical tool in Table 5.3, where different sizes of transition system were used during interpolating, namely $j = 1, 2$ and 3. We observe that the interpolant stabilizes in case of $j \geq 2$, where it overapproximates the reachable states. Figure 5.10 represents two results: the upper (green) line represents the upper bound on the step-unbounded probability to avoid unwanted region. The numbers on the horizontal axis here refer to the iteration (the number of steps), while the vertical axis refers to the computed probabilities. The lower (blue) curve represents the lower bound on the probability to stay outside "fail state and $x$ is less than or equal 7", as computed by Scheme 5.6. One may observe that the lower bound and the exact value almost coincide after step $k = 3$.

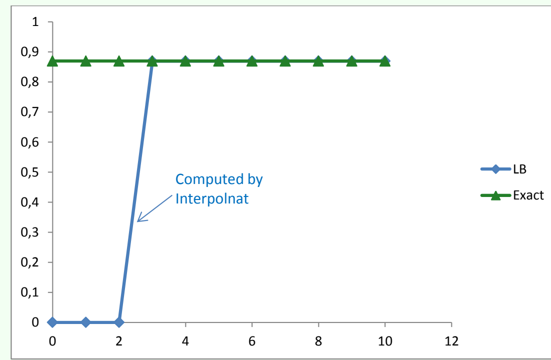
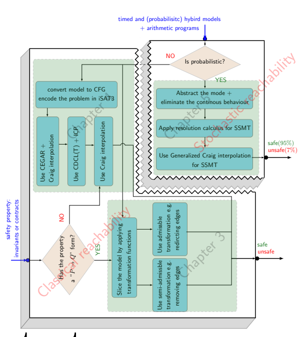



Figure 5.10: Probability of avoiding fail $\wedge$ $x \leq 7$ by using GCI.

To this end, one can encode region stability problems in SSMT formulae and probabilistic reachability problems as well, which reflects the main contributions of our approach to probabilistic unbounded model checking problems.

# Conclusion

# 6



But then of course you reach a point where you have to say, I've got to figure out how this book's going to end. Otherwise, you're going to write yourself into so many dead-ends.

*(Anthony Doerr)*

## Contents


**6.1 Achievements of this dissertation . . . . . . . . . . . . . . . . . . 155**

**6.2 Outlook . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 158**

6.2.1 Applying transformation for models admitting system modes . . 158

6.2.2 Extending iSAT3-CFG with interprocedural calls . . . . . . . . . 158

6.2.3 Computing loop summaries – maximum number of while-loop unwindings . . . . . . . . . . . . . . . . . . . . . . . . . . . . 159

6.2.4 Integrating generalized Craig interpolation with DPLL-based SSMT solving . . . . . . . . . . . . . . . . . . . . . . . . . . . . 159


In this chapter, we recap the main achievements and findings of this dissertation and sketch possible future tracks to take from here.

## 6.1 Achievements of this dissertation

In this thesis, we made three contributions to software model checking. Aside from presenting a consistent view of various related formal models that cover real, embedded and hybrid systems, the core achievements presented in this thesis lie in advancing model checking by using interpolation beyond decidable theories; covering stochastic and deterministic reachability analyses.

At first, we introduced a novel preprocessing and verification approach that deals with a wide scope of models ranging from programs, finite, timed and hybrid automata and even more system models as long as they induce computational transition systems. Given consistent transition systems and specifications with assumption-commitment form, one can apply the suggested transformations to eliminate some computational paths of these models – as required for reducing verification complexity – without changing the verification verdict. The idea is that all non-persistent traces trivially satisfy the specification, since the assumption is broken in the latter traces. Although its principle seems to be

simple, it significantly optimises the verification time by up to a factor of ten as shown on Fischer's protocol and WFAS's models in Chapter 3. The first main contribution in that direction was introducing the concept of "an edge supporting a specification", which generalizes the linear-time, trace-based satisfaction relation with respect to a single edge as a model element. Informally, an edge supports a specification, if there exists a valid computation path of the model such that the edge is used and the specification is satisfied. Based on this, two transformation functions are proposed as valid instances and exploited within source-to-source transformations, which will mark edges that do not support a specification as to be removed or redirected. Both transformation functions lead to simpler and often considerably smaller models in comparison to the original one. It is found that proving the original model satisfies the assumption-commitment property can be assured by proving that the commitment only is satisfied in the resultant model in case of removing non-support edges. Likewise, verifying the assumption-commitment property in the original model can be performed by verifying the whole property in the resultant model after redirecting non-support edges.

Second, we built an unprecedented framework to handle subtle reachability problems in non-trivial embedded software, namely the rigorous detection of dead code. It is found that dead code has a bad impact in automotive and avionics domains since it affects the testability of embedded programs. Therefore several pertinent standards for embedded system development demand adequate handling of dead code during testing or even bar it altogether, like DO-178C [EH10], DO-278A [Che09], or ISO/IEC PDTR 24772 [TRn09]. In non-trivial embedded software like Simulink-Stateflow auto-generated programs, we expect industrial-scale programs with richer arithmetic operations including polynomials and transcendental functions combined with long chains of conditional and loop statements that affect the control flow of these programs. In such a situation, all verification approaches; e.g., SMT model checkers, abstract interpretation, static analysers and CEGAR are inapt to address a solution for this problem individually since they lack the exactness or they are currently confined to linear and polynomial arithmetics only. However, the combination is beneficial if all are tightly integrated in a way such that each approach is used in its proper field.

For that purpose, CEGAR is employed in order to handle large arithmetic programs and avert the state space explosion problem due its well defined abstraction. In each iteration, either a refinement step is performed by adding necessary side-conditions to the desired model edges in case of a spurious counterexample, or a real counterexample violating the safety property is obtained at the level of program code. In order to economize the time consumption needed for back-and-forth translation between different tools, all steps are done within iSAT3. So the iSAT3 input language is extended to read control-flow graphs based on programs in order to use CEGAR as a frontend of our toolchain. Moreover, verifying the abstraction is done by using interpolation-based model checking techniques. Furthermore, in this approach conflict-driven clause learning and interval constraint propagation are used to solve very large complex Boolean formulae, and capture the arithmetic reasoning over non-polynomial constraints respectively.

Craig interpolation with SAT-based as well as SMT-based bounded model checking is able to verify non-probabilistic safety properties by proving that certain target states or rather code fragments are unreachable, namely if the overapproximated set of all reachable states has an empty intersection with the set of unsafe states. Refinement in CEGAR is done

by using (inductive) interpolants as in lazy abstraction where refinement is accomplished by adding necessary predicates to edges as side-conditions with assumption-commitment form. The latter form can restrict the current and the next valuations of variables as iSAT3 input language supports that option. That is, we conjunct these side conditions and eke out the size of the abstraction model during verification.

In order to use our approach on real industrial problems, we built a special parser that converts SMI code provided by BTC-ES AG to iSAT3-CFG input language. SMI code is an intermediate language representation of the C language that consists of one unconstrained while-loop block with a list of assignments.

These SMI programs admit linear and non-linear assignments and conditions besides bit-wise operations, loops, and distinguishing cases as well. Also, for the purpose of real certification, IEEE 754 for floating point arithmetic is extended from iSAT3 to iSAT3-CFG where special values such as *NaNs*, $+\infty$ $-\infty$, $-0$, $+0$ and subnormal numbers are handled. This support enables us to precisely solve the cases where a weak satisfiability (*candidate solution*) often appears. After that, these programs are verified by using our approach with several options where the verification results shows the effectiveness of our approach.

The last but the not least contribution is a generalization of Craig interpolation such that it deals with all SAT, SMT, SSAT and SSMT problems. It does not only go beyond probabilistic bounded state reachability problems, but also covers richer fragment of arithmetic theories beyond Teige's approach for probabilistic finite-state models like Markov decision processes. Namely, this approach addresses a solution for both reachability unbounded model checking and stability problems in probabilistic hybrid system models with discrete time steps. For this purpose, the generalized Craig interpolation for SSMT formulae was introduced. At the first point, a sound and relatively-complete resolution calculus for SSMT formulae called SSMT resolution was introduced. We augmented it – non-exclusively – with an extension of Pudlák-style symmetric rules for interpolant generation. This resolution misses the completeness due to interval constraint propagation used as arithmetic reasoner for non-linear constraints, where the latter are i.g. undecidable problems when non-linear constraints contain transcendental functions.

In order to utilize the generalized Craig interpolation in model checking, a probabilistic state reachability is introduced for probabilistic hybrid automata such that we get a probabilistic (in)finite-state systems at the end, akin to SSAT's approach, however in our case, either finite-state abstraction or safe approximation must be used. We developed a symbolic verification procedure for probabilistic safety properties of probabilistic (in)finite-state systems obtained after abstraction or approximation.

Akin to symbolic methods for non-probabilistic systems, generalized Craig interpolation provides a technique for computing a symbolic overapproximation of the (backward) reachable state set of probabilistic systems. While Craig interpolation-based model checking for stochastic propositional satisfiability problems was able to verify safety properties of the shape "*the probability of reaching the unsafe states is at most 1% in worst case*", many safety properties representing richer fragment of arithmetic constraints are frequently unavoidable in probabilistic scenarios. Thereby, in this thesis, verifying safety properties of the shape "*the probability that $x$ is larger than or equal 3 is at most 1% in worst case*", where $x$ is a real number representing continuous behaviour in hybrid models and appears

in polynomials and transcendental functions, is solved by using a Craig interpolation for SSMT. The verification procedure devised in this thesis abstracts the probabilistic hybrid automata with discrete time step in order to get a finite-state probabilistic model due to the use of ProHVer and the limitation of integrating the SSMT-resolution with SiSAT (cf. Remark 5.1). Then our procedure exploits the symbolic overapproximation of the backward reachable state set, being the fixed point of an iterative computation of generalized Craig interpolants, as well as a predicative description of the system in order to construct SSMT formulae whose quantitative interpretations yield upper bounds on the worst-case probability of reaching the unsafe states. Whenever an upper bound of at most 1% is computed using an SSMT solver; e.g., SiSAT, then above probabilistic safety property is verified and satisfied in the probabilistic hybrid automaton. On the other hand, if the upper bound exceeds 1%, then we need to find a finer finite-state abstraction or/and increase the size of transition systems in order to get more precise symbolic overapproximation of the backward reachable state set.

## 6.2 Outlook

### 6.2.1 Applying transformation for models admitting system modes

Some real time systems admit several *mode-switching*; e.g., *initial*, *run*, *success* and *fail*, where there is a possibility to leave any of the defined modes. For example, *fail* state is not a deadend but it can be left to *init* state. Such a real-time system model cannot be handled by our transformational approach that detects non-supporting edges of assumption and either removes or redirects them, since several edges which lead to a violation of the assumption of contract (reach *fail* state), may not be targeted by redirecting or removing operations due to mode-switching mechanism. One solution is to define Boolean observers to capture only the violation of the contract assumption independent from mode-switching mechanism. One may consequently think about defining special model patterns that capture the necessary features/conditions of applying transformation functions on automata models which admit mode-switching. Whenever a model meets these preconditions, one can apply our proposed compositional verification as introduced in Chapter 3, even though these models have *mode-switching* feature.

### 6.2.2 Extending iSAT3-CFG with interprocedural calls

The current procedure of iSAT3-CFG enables us to encode imperative arithmetic while-programs as control flow automata, where each segment of program code can be presented by either a guard or an assignment attached to the edges of a control flow automaton. However, dealing with programs that have procedures or pointers was not discussed in this thesis. Henzinger et al. [HJMM04] proposed a sound procedure to handle programs with function calls and pointers by summarizing all effects that the callee may have had on the caller's store at the point of return. One possibility to achieve this is to assume the callee starting with a copy of the caller's store and, upon return, the caller refreshing his store appropriately using the callee's store.

Another idea is to use abstract interpretation AI [CC77] as a necessary, yet not a sufficient scheme to annotate the source code of C programs at each entry and end points of functions by a safe overapproximation of reachable set of variables valuations before and after the function invokes. Using AI as demonstrated in the Astreé [SD07] static analyser would permit a computing of interprocedural calls summary in embedded system programs. Thereby, using AI would be a good preprocessing step before using iSAT3-CFG to solve programs containing non-linear constraints and interprocedural calls. This enables us to handle programs containing interrupts routines as well, which are widely used in industrial community.

### 6.2.3 Computing loop summaries – maximum number of while-loop unwindings

Proving a termination of a loop – in general[1] – is an undecidable problem even in case of loops having only linear constraints. However, this problem throws a shadow on non-termination problem of unbounded model checking technique while solving hybrid models admitting non-linear constraints. For programs that contain internal/nested loops such as for-loops or while-loops, one can syntactically tackle them by considering all iterations of these loops provided that the loop bounds are given by constants. However, for conditional for-loops or while-loops which their bounds not being constants, this is not the case. The most appropriate approach for these corner cases is to compute a proper summary for each loop in a program. In the following, I consider while-loop since BTC-ES AG benchmarks that are discussed in Subsection 4.5.4 contain this kind of loops.

This summary is not required to compute the exact behaviour of the loop, however a safe but precise overapproximation will often suffice for the model checker's needs. To do so, one can use abstract interpretation to find summaries of loops as aforestated. An alternative solution would consider each loop as a standalone problem. That is, one can use the solver to explore/unwind the while-loops till certain depth to build a tight but precise approximation of that problem or to annotate the loops with proper pre-and post-conditions that guarantee a proof of safety of the higher model, then pass the subproblem as a complete task to the solver. If a maximum number of a while-loop iterations is computed, one would elaborate the while-loops and flatten the C-programs. Flattening a program leads to convert a complete program to one large formula in a non-conditional while-loop. Thus, instead of solving a program iteratively by using BMC and considering each conditional branching in a nested while-block as a step, one can flatten the while-block and encode it as one SMT formula which can be solved in one step. This will optimise the verification process by a fair margin and fulfil the industrial needs.

### 6.2.4 Integrating generalized Craig interpolation with DPLL-based SSMT solving

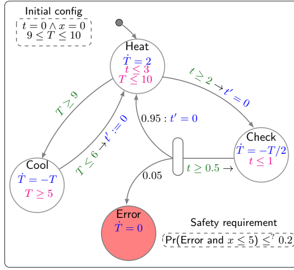
SiSAT [Tei12] is a DPLL-based SSMT model checker that can solve probabilistic bounded reachability problems besides solving SSAT and SSMT problems as well. Generalized

[1]There are some classes proven to be decidable. e.g. [Tiw04, Bra06], where other classes are undecidable [XZ10]

Craig inteprolation has three rules, namely GR.1, GR.2 and GR.3 (cf. Section 5.4), which are built on top of the SSMT resolution rules, namely RR.1, RR.2, RR.3 and RR.3e, where all rules can be forthrightly integrated with DPLL-based SSMT solvers except for GR.2 and RR.2, which need special handling. In RR.2 and GR.2, we aim at finding a clause $cl$ – which in general is undecidable, since it is equivalent to finding a satisfying (partial) assignment of an SMT formula (with possible non-linear constraints) in CNF form – that falsifies the formula. This strong application condition of RR.2 can however be justified with regard to a potential integration of SSMT resolution into DPLL-based SSMT solvers, since DPLL-SSMT strongly relies on finding satisfying assignments (or an overapproximation thereof), confer the base case of DPLL-based SSMT where all clauses in $\varphi$ are equivalent to true. Observe that whenever a satisfying (partial) assignment $\tau'$ of $\varphi$ is found by a DPLL-based SSMT solver then $\models \varphi[\tau'(y_1)/y_1]...[\tau'(y_k)/y_k]$ with $y_1, ..., y_k \in Var(\varphi)$ being all variables for which $\tau'(y_1), ..., \tau'(y_k)$ are defined.

It is then straightforward to construct from $\tau'$ a clause $cl$ which meets the requirements of RR.2, namely for each $x \in V(cl)$: $(x \sim a) \in cl$ if and only if $\tau'(x) = b$ where $(x = b) \wedge (x \sim a) \models \texttt{false}$. This allows us to compute the GCI in DPLL-based SSMT solvers, and consequently handle probabilistic unbounded reachability problems within SiSAT instead of using ProHVer for abstracting probabilistic hybrid automata.

# Appendix A




Appendix usually means "small outgrowth from large intestine," but in this case it means "additional information accompanying main text." Or are those really the same things? Think carefully before you insult this book.

*(Pseudonymous Bosch)*

## Computation of interpolants in Thermostat example

In this subsection, we will show how the results of Table 5.1 are obtained[2]

1. we abstract the PHA model Figure 5.5 by using ProHVer tool [ZSR+10, Fre08] and the result is shown in Figure 5.8a.
2. we encode the abstract finite-state model into an SSMT formula as proposed in [Tei12, FHT08]:
   - for the first call, we encode the target states in $TARGET(x)$. So the `state-set predicate` $\mathcal{B}^0(x)$ equals $TARGET(x)$.
   - for the second call or step, $\mathcal{B}^1(x)$ equals $\mathcal{B}^1(x) \vee \mathcal{I}^1(x)$.
   - for $k$-th step, $\mathcal{B}^{k+1}(x)$ equals $\mathcal{B}^k(x) \vee \mathcal{I}^{k+1}(x)$, where

$$\mathcal{I}^{k+1} = (\underbrace{TRANS_{\mathcal{M}}(x_{j-1}, x_j) \wedge \mathcal{B}^k(x_j)}_{A}\ ,\underbrace{INIT_{\mathcal{M}}(x_0) \wedge \bigwedge_{i=1}^{j-1} TRANS_{\mathcal{M}}(x_{i-1}, x_i))}_{B}$$

   - we continue with this step until $\mathcal{B}^k$ reaches the least fixed point; i.e., $\mathcal{B}^{k+1}(x) \to \mathcal{B}^k(x)$.
   - using $j$ does not destroy the “backward-overapproximating” property of $\mathcal{B}^{k+1}(x)$. Variable $j$ gives us an additional freedom in constructing generalized interpolants since $j$ may influence the shape of $\mathcal{I}_{k+1}(x)$ [Tei12].

[2] By using a prototypical tool (developed under Java 1.7) that performs direct SSMT resolution, respecting the quantifiers order as in OBF problems, we compute a general conflict clause and consequently the generalized Craig interpolation.

- the probability (lowest upper bound) of reaching the target states from the initial states is computed according to the following scheme:

$$ub_k = Pr\left(\mathcal{Q}(k) : (\overbrace{INIT(x_0) \wedge \bigwedge_{i=1}^{k} TRANS(x_{i-1}, x_i)}^{\texttt{reachable with k steps}} \wedge \overbrace{\bigwedge_{i=0}^{k} B^k(x_i))}^{\texttt{stay in B}}\right) \quad (0.1)$$

- the probability of reaching the target states from the initial states is computed according to the following scheme:

$$lb_k = Pr\left(\mathcal{Q}(k) : (\overbrace{INIT(x_0) \wedge \bigwedge_{i=1}^{k} TRANS(x_{i-1}, x_i)}^{\texttt{reachable with k steps}} \wedge \overbrace{\bigvee_{i=0}^{k} TARGET^k(x_i)}^{\texttt{hit one target}}\right) \quad (0.2)$$

Now the initial encoding variables are:

- $INIT = A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg F \wedge t = 0 \wedge x = 0 \wedge 9 \leq T \wedge T \leq 10$.
- `Trans-set` $=\{(A, \{t \geq 0, x \geq 0, T \leq 10, t = x\}, A), (A, true, B), (B, \{t \geq 2, x \geq 0, t = x - 2, T \leq 10\}, B), (B, true, C), (B, true, D), (C, x \leq 5, C), (D, \{t \geq 0, x \geq 0, t = x - 2.5, T \leq 10\}, D), (D, true, E), (E, \{t \geq 2, x \geq 0, t = x - 4.5, T \leq 10\}, E), (E, true, C), (E, true, F), (F, \{t \geq 0, x \geq 0, t = x - 5, T \leq 10\})\}$.
- $BDC = (B, D), \neg BDC = (B, C),\ EFC = (E, F), \neg EFC = (E, C)$
- $TRANS = (A \rightarrow t \geq 0 \wedge x \geq 0 \wedge t = x \wedge T \leq 10 \wedge A), (A \rightarrow t' \geq 2 \wedge x' \geq 0 \wedge t' = x' - 2 \wedge T' \leq 10 \wedge B'), (B \wedge BDC \rightarrow D' \wedge t' \geq 0 \wedge x' \geq 0 \wedge t' = x' - 2.5 \wedge T' \leq 10), (B \wedge \neg BDC \rightarrow C' \wedge x' \leq 5), (D \rightarrow t' \geq 2 \wedge x' \geq 0 \wedge t' = x' - 4.5 \wedge T' \leq 10 \wedge E'), (E \wedge EFC \rightarrow F \wedge x' \geq 0 \wedge t' \geq 0 \wedge t' = x' - 5), (E \wedge \neg EFC \rightarrow C' \wedge x' \leq 5), (F \rightarrow t' \geq 0 \wedge x' \geq 0 \wedge t' = x' - 5 \wedge T' \leq 10 \wedge F'), (C \rightarrow x' \leq 5 \wedge C')$.
- $TARGET = C \wedge x \leq 5$
- Quantifiers are $\mathcal{Q}_s = \exists A, B, C, D, E, F$ representing the current state, $\mathcal{Q}_{s'} = \exists A', B', C', D', E', F'$ representing the next state, $\mathcal{Q}_{pc} = Я^{0.95} BDC\ Я^{0.95} EFC$ representing the probabilistic choices in the finite-state abstraction, $\mathcal{Q}_v = \exists T, x, t$ and $\mathcal{Q}_{v'} = \exists T', x', t'$ representing the innermost implicit existential quantifiers for continuous state variables in the current and next transitions respectively.

**First Step:** Choosing $j = 1$,

$$\mathcal{Q}_{x_0} \mathcal{Q}_{v_0} \mathcal{Q}_{pc_1} \mathcal{Q}_{x_1} \mathcal{Q}_{v_1} : (\underbrace{INIT(x_0)}_{B} \wedge \underbrace{TRANS(x_0, x_1) \wedge \mathcal{B}^0(x_1)}_{A})$$

where $\mathcal{B}^0(x_1) = TARGET(s_1) = C_1 \wedge x_1 \leq 5$. $V_A = \{A_1, B_1, C_1, D_1, E_1, F_1, BDC_1, EFC_1, t_1, x_1, T_1\}$, $V_B = \{\}$, and $V_{A,B} = \{A_0, B_0, C_0, D_0, E_0, F_0, T_0, t_0, x_0\}$. Resolution with Interpolation is preformed as follows:

| | | |
|---|---|---|
| $c_1 = (A_0)^{0.0}$ | $\mathcal{I}_1 = \texttt{true}$ | (GR.1) |
| $c_2 = (\neg B_0)^{0.0}$ | $\mathcal{I}_2 = \texttt{true}$ | (GR.1) |
| $c_3 = (\neg C_0)^{0.0}$ | $\mathcal{I}_3 = \texttt{true}$ | (GR.1) |
| $c_4 = (\neg D_0)^{0.0}$ | $\mathcal{I}_4 = \texttt{true}$ | (GR.1) |
| $c_5 = (\neg E_0)^{0.0}$ | $\mathcal{I}_5 = \texttt{true}$ | (GR.1) |
| $c_6 = (\neg F_0)^{0.0}$ | $\mathcal{I}_6 = \texttt{true}$ | (GR.1) |
| $c_7 = (t_0 = 0)^{0.0}$ | $\mathcal{I}_7 = \texttt{true}$ | (GR.1) |
| $c_8 = (x_0 = 0)^{0.0}$ | $\mathcal{I}_8 = \texttt{true}$ | (GR.1) |
| $c_9 = (T_0 \geq 9)^{0.0}$ | $\mathcal{I}_9 = \texttt{true}$ | (GR.1) |
| $c_{10} = (T_0 \leq 10)^{0.0}$ | $\mathcal{I}_{10} = \texttt{true}$ | (GR.1) |
| $c_{11} = (A_1 \vee B_1 \vee C_1 \vee D_1 \vee E_1 \vee F_1)^{0.0}$ | $\mathcal{I}_{11} = \texttt{false}$ | (GR.1) |
| $c_{12} = (\neg A_1 \vee \neg B_1)^{0.0}$ | $\mathcal{I}_{12} = \texttt{false}$ | (GR.1) |
| $c_{13} = (\neg A_1 \vee \neg C_1)^{0.0}$ | $\mathcal{I}_{13} = \texttt{false}$ | (GR.1) |
| $c_{14} = (\neg A_1 \vee \neg D_1)^{0.0}$ | $\mathcal{I}_{14} = \texttt{false}$ | (GR.1) |
| $c_{15} = (\neg A_1 \vee \neg E_1)^{0.0}$ | $\mathcal{I}_{15} = \texttt{false}$ | (GR.1) |
| $c_{16} = (\neg A_1 \vee \neg F_1)^{0.0}$ | $\mathcal{I}_{16} = \texttt{false}$ | (GR.1) |
| $c_{17} = (\neg B_1 \vee \neg C_1)^{0.0}$ | $\mathcal{I}_{17} = \texttt{false}$ | (GR.1) |
| $c_{18} = (\neg B_1 \vee \neg D_1)^{0.0}$ | $\mathcal{I}_{18} = \texttt{false}$ | (GR.1) |
| $c_{19} = (\neg B_1 \vee \neg E_1)^{0.0}$ | $\mathcal{I}_{19} = \texttt{false}$ | (GR.1) |
| $c_{20} = (\neg B_1 \vee \neg F_1)^{0.0}$ | $\mathcal{I}_{20} = \texttt{false}$ | (GR.1) |
| $c_{21} = (\neg C_1 \vee \neg D_1)^{0.0}$ | $\mathcal{I}_{21} = \texttt{false}$ | (GR.1) |
| $c_{22} = (\neg C_1 \vee \neg E_1)^{0.0}$ | $\mathcal{I}_{22} = \texttt{false}$ | (GR.1) |
| $c_{23} = (\neg C_1 \vee \neg F_1)^{0.0}$ | $\mathcal{I}_{23} = \texttt{false}$ | (GR.1) |
| $c_{24} = (\neg D_1 \vee \neg E_1)^{0.0}$ | $\mathcal{I}_{24} = \texttt{false}$ | (GR.1) |
| $c_{25} = (\neg D_1 \vee \neg F_1)^{0.0}$ | $\mathcal{I}_{25} = \texttt{false}$ | (GR.1) |
| $c_{26} = (\neg E_1 \vee \neg F_1)^{0.0}$ | $\mathcal{I}_{26} = \texttt{false}$ | (GR.1) |
| $c_{27} = (\neg A_0 \vee t_0 \geq 0)^{0.0}$ | $\mathcal{I}_{27} = \texttt{false}$ | (GR.1) |
| $c_{28} = (\neg A_0 \vee \vee x_0 \geq 0)^{0.0}$ | $\mathcal{I}_{28} = \texttt{false}$ | (GR.1) |
| $c_{29} = (\neg A_0 \vee t_0 = x_0)^{0.0}$ | $\mathcal{I}_{29} = \texttt{false}$ | (GR.1) |
| $c_{30} = (\neg A_0 \vee T_0 \leq 10)^{0.0}$ | $\mathcal{I}_{30} = \texttt{false}$ | (GR.1) |
| $c_{31} = (\neg A_0 \vee B_1)^{0.0}$ | $\mathcal{I}_{31} = \texttt{false}$ | (GR.1) |
| $c_{32} = (\neg A_0 \vee t_1 \geq 2)^{0.0}$ | $\mathcal{I}_{32} = \texttt{false}$ | (GR.1) |
| $c_{33} = (\neg A_0 \vee x_1 \geq 0)^{0.0}$ | $\mathcal{I}_{33} = \texttt{false}$ | (GR.1) |
| $c_{34} = (\neg A_0 \vee t_1 = x_1 - 2)^{0.0}$ | $\mathcal{I}_{34} = \texttt{false}$ | (GR.1) |
| $c_{35} = (\neg A_0 \vee T_1 \leq 10)^{0.0}$ | $\mathcal{I}_{35} = \texttt{false}$ | (GR.1) |
| $c_{36} = (\neg B_0 \vee \neg BDC_1 \vee D_1)^{0.0}$ | $\mathcal{I}_{36} = \texttt{false}$ | (GR.1) |
| $c_{37} = (\neg B_0 \vee \neg BDC_1 \vee T_1 \leq 10)^{0.0}$ | $\mathcal{I}_{37} = \texttt{false}$ | (GR.1) |
| $c_{38} = (\neg B_0 \vee \neg BDC_1 \vee t_1 \geq 0)^{0.0}$ | $\mathcal{I}_{38} = \texttt{false}$ | (GR.1) |
| $c_{39} = (\neg B_0 \vee \neg BDC_1 \vee x_1 \geq 0)^{0.0}$ | $\mathcal{I}_{39} = \texttt{false}$ | (GR.1) |
| $c_{40} = (\neg B_0 \vee \neg BDC_1 \vee t_1 = x_1 - 2.5)^{0.0}$ | $\mathcal{I}_{40} = \texttt{false}$ | (GR.1) |
| $c_{41} = (\neg B_0 \vee BDC_1 \vee C_1)^{0.0}$ | $\mathcal{I}_{41} = \texttt{false}$ | (GR.1) |
| $c_{42} = (\neg B_0 \vee BDC_1 \vee x_1 \leq 5)^{0.0}$ | $\mathcal{I}_{42} = \texttt{false}$ | (GR.1) |
| $c_{43} = (\neg D_0 \vee E_1)^{0.0}$ | $\mathcal{I}_{43} = \texttt{false}$ | (GR.1) |
| $c_{44} = (\neg D_0 \vee t_1 \geq 2)^{0.0}$ | $\mathcal{I}_{44} = \texttt{false}$ | (GR.1) |
| $c_{45} = (\neg D_0 \vee x_1 \geq 0)^{0.0}$ | $\mathcal{I}_{45} = \texttt{false}$ | (GR.1) |
| $c_{46} = (\neg D_0 \vee t_1 = x_1 - 4.5)^{0.0}$ | $\mathcal{I}_{46} = \texttt{false}$ | (GR.1) |

| | | |
|---|---|---|
| $c_{47} = (\neg D_0 \vee T_1 \leq 10)^{0.0}$ | $\mathcal{I}_{47} = \texttt{false}$ | (GR.1) |
| $c_{48} = (\neg E_0 \vee EFC_1 \vee C_1)^{0.0}$ | $\mathcal{I}_{48} = \texttt{false}$ | (GR.1) |
| $c_{49} = (\neg E_0 \vee EFC_1 \vee x_1 \leq 5)^{0.0}$ | $\mathcal{I}_{49} = \texttt{false}$ | (GR.1) |
| $c_{50} = (\neg E_0 \vee \neg EFC_1 \vee F_1)^{0.0}$ | $\mathcal{I}_{50} = \texttt{false}$ | (GR.1) |
| $c_{51} = (\neg E_0 \vee \neg EFC_1 \vee x_1 \geq 0)^{0.0}$ | $\mathcal{I}_{51} = \texttt{false}$ | (GR.1) |
| $c_{52} = (\neg E_0 \vee \neg EFC_1 \vee t_1 = x_1 - 5)^{0.0}$ | $\mathcal{I}_{52} = \texttt{false}$ | (GR.1) |
| $c_{53} = (\neg E_0 \vee \neg EFC_1 \vee T_1 \leq 10)^{0.0}$ | $\mathcal{I}_{53} = \texttt{false}$ | (GR.1) |
| $c_{54} = (\neg C_0 \vee C_1)^{0.0}$ | $\mathcal{I}_{54} = \texttt{false}$ | (GR.1) |
| $c_{55} = (\neg C_0 \vee x_{\leq}5)^{0.0}$ | $\mathcal{I}_{55} = \texttt{false}$ | (GR.1) |
| $c_{56} = (\neg F_0 \vee F_1)^{0.0}$ | $\mathcal{I}_{56} = \texttt{false}$ | (GR.1) |
| $c_{57} = (\neg F_0 \vee x_1 \geq 0)^{0.0}$ | $\mathcal{I}_{57} = \texttt{false}$ | (GR.1) |
| $c_{58} = (\neg F_0 \vee t_1 \geq 0)^{0.0}$ | $\mathcal{I}_{58} = \texttt{false}$ | (GR.1) |
| $c_{59} = (\neg F_0 \vee t_1 = x_1 - 5)^{0.0}$ | $\mathcal{I}_{59} = \texttt{false}$ | (GR.1) |
| $c_{60} = (C_1)^{0.0}$ | $\mathcal{I}_{60} = \texttt{false}$ | (GR.1) |
| $c_{61} = (x_1 \leq 5)^{0.0}$ | $\mathcal{I}_{61} = \texttt{false}$ | (GR.1) |
| $c_{62} = (\neg B_1)^{0.0}$ | $\mathcal{I}_{62} = \texttt{false}$ | (GR.3), $c_{17}, c_{60}$ |
| $c_{63} = (\neg A_0)^{0.0}$ | $\mathcal{I}_{63} = \texttt{false}$ | (GR.3), $c_{31}, c_{62}$ |
| $c_{64} = \emptyset^{0.0}$ | $\mathcal{I}_{64} = \neg A_0$ | (GR.3), $c_1, c_{63}$ |

The interpolant i.e $\mathcal{I}$ is $\neg A$.[3] Then the maximum upper bound probability is 1 ☹. We can not gain any information. So we need to change the shape of the transition system by increasing $j$.

**Second Step:** Choosing $j = 2$, and we overapproximate the reachable states from the target states; i.e.,

$$\mathcal{Q}_{x_0} \mathcal{Q}_{v_0} \mathcal{Q}_{pc_1} \mathcal{Q}_{x_1} \mathcal{Q}_{v_1} \mathcal{Q}_{pc_2} \mathcal{Q}_{x_2} \mathcal{Q}_{v_2} :$$
$$(\underbrace{INIT(x_0) \wedge TRANS(x_0, x_1)}_{B} \wedge \underbrace{TRANS(x_1, x_2) \wedge \mathcal{B}^0(x_2)}_{A})$$

where $\mathcal{B}^0(x_2) = TARGET(x_2) = C_2 \wedge x_2 \leq 5$, $V_A = \{A_2, B_2, C_2, D_2, E_2, F_2, BDC_2, EFC_2, t_2, x_2, T_2\}$, $V_B = \{A_0, B_0, C_0, D_0, E_0, F_0, T_0, t_0, x_0, BDC_1, EFC_1\}$, $V_{A,B} = \{A_1, B_1, C_1, D_1, E_1, F_1, T_1, t_1, x_1\}$. Resolution with Interpolation is preformed as follows:

| | | |
|---|---|---|
| $c_1 = (A_0)^{0.0}$ | $\mathcal{I}_1 = \texttt{true}$ | (GR.1) |
| $c_2 = (\neg B_0)^{0.0}$ | $\mathcal{I}_2 = \texttt{true}$ | (GR.1) |
| $c_3 = (\neg C_0)^{0.0}$ | $\mathcal{I}_3 = \texttt{true}$ | (GR.1) |
| $c_4 = (\neg D_0)^{0.0}$ | $\mathcal{I}_4 = \texttt{true}$ | (GR.1) |
| $c_5 = (\neg E_0)^{0.0}$ | $\mathcal{I}_5 = \texttt{true}$ | (GR.1) |
| $c_6 = (\neg F_0)^{0.0}$ | $\mathcal{I}_6 = \texttt{true}$ | (GR.1) |
| $c_7 = (t_0 = 0)^{0.0}$ | $\mathcal{I}_7 = \texttt{true}$ | (GR.1) |
| $c_8 = (x_0 = 0)^{0.0}$ | $\mathcal{I}_8 = \texttt{true}$ | (GR.1) |
| $c_9 = (T_0 \geq 9)^{0.0}$ | $\mathcal{I}_9 = \texttt{true}$ | (GR.1) |
| $c_{10} = (T_0 \leq 10)^{0.0}$ | $\mathcal{I}_{10} = \texttt{true}$ | (GR.1) |
| $c_{11} = (A_1 \vee B_1 \vee C_1 \vee D_1 \vee E_1 \vee F_1)^{0.0}$ | $\mathcal{I}_{11} = \texttt{true}$ | (GR.1) |
| $c_{12} = (\neg A_1 \vee \neg B_1)^{0.0}$ | $\mathcal{I}_{12} = \texttt{true}$ | (GR.1) |

[3] The original interpolant is decorated with 0, however, we eliminate this subscript index by considering the original predicate (for more details, see [McM03, McM05]).

| | | |
|---|---|---|
| $c_{13} = (\neg A_1 \vee \neg C_1)^{0.0}$ | $\mathcal{I}_{13} = \texttt{true}$ | (GR.1) |
| $c_{14} = (\neg A_1 \vee \neg D_1)^{0.0}$ | $\mathcal{I}_{14} = \texttt{true}$ | (GR.1) |
| $c_{15} = (\neg A_1 \vee \neg E_1)^{0.0}$ | $\mathcal{I}_{15} = \texttt{true}$ | (GR.1) |
| $c_{16} = (\neg A_1 \vee \neg F_1)^{0.0}$ | $\mathcal{I}_{16} = \texttt{true}$ | (GR.1) |
| $c_{17} = (\neg B_1 \vee \neg C_1)^{0.0}$ | $\mathcal{I}_{17} = \texttt{true}$ | (GR.1) |
| $c_{18} = (\neg B_1 \vee \neg D_1)^{0}$ | $\mathcal{I}_{18} = \texttt{true}$ | (GR.1) |
| $c_{19} = (\neg B_1 \vee \neg E_1)^{0}$ | $\mathcal{I}_{19} = \texttt{true}$ | (GR.1) |
| $c_{20} = (\neg B_1 \vee \neg F_1)^{0.0}$ | $\mathcal{I}_{20} = \texttt{true}$ | (GR.1) |
| $c_{21} = (\neg C_1 \vee \neg D_1)^{0.0}$ | $\mathcal{I}_{21} = \texttt{true}$ | (GR.1) |
| $c_{22} = (\neg C_1 \vee \neg E_1)^{0.0}$ | $\mathcal{I}_{22} = \texttt{true}$ | (GR.1) |
| $c_{23} = (\neg C_1 \vee \neg F_1)^{0.0}$ | $\mathcal{I}_{23} = \texttt{true}$ | (GR.1) |
| $c_{24} = (\neg D_1 \vee \neg E_1)^{0.0}$ | $\mathcal{I}_{24} = \texttt{true}$ | (GR.1) |
| $c_{25} = (\neg D_1 \vee \neg F_1)^{0.0}$ | $\mathcal{I}_{25} = \texttt{true}$ | (GR.1) |
| $c_{26} = (\neg E_1 \vee \neg F_1)^{0.0}$ | $\mathcal{I}_{26} = \texttt{true}$ | (GR.1) |
| $c_{27} = (\neg A_0 \vee t_0 \geq 0)^{0.0}$ | $\mathcal{I}_{27} = \texttt{true}$ | (GR.1) |
| $c_{28} = (\neg A_0 \vee \vee x_0 \geq 0)^{0.0}$ | $\mathcal{I}_{28} = \texttt{true}$ | (GR.1) |
| $c_{29} = (\neg A_0 \vee t_0 = x_0)^{0.0}$ | $\mathcal{I}_{29} = \texttt{true}$ | (GR.1) |
| $c_{30} = (\neg A_0 \vee T_0 \leq 10)^{0.0}$ | $\mathcal{I}_{30} = \texttt{true}$ | (GR.1) |
| $c_{31} = (\neg A_0 \vee B_1)^{0.0}$ | $\mathcal{I}_{31} = \texttt{true}$ | (GR.1) |
| $c_{32} = (\neg A_0 \vee t_1 \geq 2)^{0.0}$ | $\mathcal{I}_{32} = \texttt{true}$ | (GR.1) |
| $c_{33} = (\neg A_0 \vee x_1 \geq 0)^{0.0}$ | $\mathcal{I}_{33} = \texttt{true}$ | (GR.1) |
| $c_{34} = (\neg A_0 \vee t_1 = x_1 - 2)^{0.0}$ | $\mathcal{I}_{34} = \texttt{true}$ | (GR.1) |
| $c_{35} = (\neg A_0 \vee T_1 \leq 10)^{0.0}$ | $\mathcal{I}_{35} = \texttt{true}$ | (GR.1) |
| $c_{36} = (\neg B_0 \vee \neg BDC_1 \vee D_1)^{0.0}$ | $\mathcal{I}_{36} = \texttt{true}$ | (GR.1) |
| $c_{37} = (\neg B_0 \vee \neg BDC_1 \vee T_1 \leq 10)^{0.0}$ | $\mathcal{I}_{37} = \texttt{true}$ | (GR.1) |
| $c_{38} = (\neg B_0 \vee \neg BDC_1 \vee t_1 \geq 0)^{0.0}$ | $\mathcal{I}_{38} = \texttt{true}$ | (GR.1) |
| $c_{39} = (\neg B_0 \vee \neg BDC_1 \vee x_1 \geq 0)^{0.0}$ | $\mathcal{I}_{39} = \texttt{true}$ | (GR.1) |
| $c_{40} = (\neg B_0 \vee \neg BDC_1 \vee t_1 = x_1 - 2.5)^{0.0}$ | $\mathcal{I}_{40} = \texttt{true}$ | (GR.1) |
| $c_{41} = (\neg B_0 \vee BDC_1 \vee C_1)^{0.0}$ | $\mathcal{I}_{41} = \texttt{true}$ | (GR.1) |
| $c_{42} = (\neg B_0 \vee BDC_1 \vee x_1 \leq 5)^{0.0}$ | $\mathcal{I}_{42} = \texttt{true}$ | (GR.1) |
| $c_{43} = (\neg D_0 \vee E_1)^{0.0}$ | $\mathcal{I}_{43} = \texttt{true}$ | (GR.1) |
| $c_{44} = (\neg D_0 \vee t_1 \geq 2)^{0.0}$ | $\mathcal{I}_{44} = \texttt{true}$ | (GR.1) |
| $c_{45} = (\neg D_0 \vee x_1 \geq 0)^{0.0}$ | $\mathcal{I}_{45} = \texttt{true}$ | (GR.1) |
| $c_{46} = (\neg D_0 \vee t_1 = x_1 - 4.5)^{0.0}$ | $\mathcal{I}_{46} = \texttt{true}$ | (GR.1) |
| $c_{47} = (\neg D_0 \vee T_1 \leq 10)^{0.0}$ | $\mathcal{I}_{47} = \texttt{true}$ | (GR.1) |
| $c_{48} = (\neg E_0 \vee EFC_1 \vee C_1)^{0.0}$ | $\mathcal{I}_{48} = \texttt{true}$ | (GR.1) |
| $c_{49} = (\neg E_0 \vee EFC_1 \vee x_1 \leq 5)^{0.0}$ | $\mathcal{I}_{49} = \texttt{true}$ | (GR.1) |
| $c_{50} = (\neg E_0 \vee \neg EFC_1 \vee F_1)^{0.0}$ | $\mathcal{I}_{50} = \texttt{true}$ | (GR.1) |
| $c_{51} = (\neg E_0 \vee \neg EFC_1 \vee x_1 \geq 0)^{0.0}$ | $\mathcal{I}_{51} = \texttt{true}$ | (GR.1) |
| $c_{52} = (\neg E_0 \vee \neg EFC_1 \vee t_1 = x_1 - 5)^{0.0}$ | $\mathcal{I}_{52} = \texttt{true}$ | (GR.1) |
| $c_{53} = (\neg E_0 \vee \neg EFC_1 \vee T_1 \leq 10)^{0.0}$ | $\mathcal{I}_{53} = \texttt{true}$ | (GR.1) |
| $c_{54} = (\neg C_0 \vee C_1)^{0.0}$ | $\mathcal{I}_{54} = \texttt{true}$ | (GR.1) |
| $c_{55} = (\neg C_0 \vee x_{\leq} 5)^{0.0}$ | $\mathcal{I}_{55} = \texttt{true}$ | (GR.1) |
| $c_{56} = (\neg F_0 \vee F_1)^{0.0}$ | $\mathcal{I}_{56} = \texttt{true}$ | (GR.1) |
| $c_{57} = (\neg F_0 \vee x_1 \geq 0)^{0.0}$ | $\mathcal{I}_{57} = \texttt{true}$ | (GR.1) |
| $c_{58} = (\neg F_0 \vee t_1 \geq 0)^{0.0}$ | $\mathcal{I}_{58} = \texttt{true}$ | (GR.1) |

$c_{59} = (\neg F_0 \vee t_1 = x_1 - 5)^{0.0}$ $\mathcal{I}_{59} = \texttt{true}$ (GR.1)
$c_{60} = (A_2 \vee B_2 \vee C_2 \vee D_2 \vee E_2 \vee F_2)^{0.0}$ $\mathcal{I}_{60} = \texttt{false}$ (GR.1)
$c_{61} = (\neg A_2 \vee \neg B_2)^{0.0}$ $\mathcal{I}_{61} = \texttt{false}$ (GR.1)
$c_{62} = (\neg A_2 \vee \neg C_2)^{0.0}$ $\mathcal{I}_{62} = \texttt{false}$ (GR.1)
$c_{63} = (\neg A_2 \vee \neg D_2)^{0.0}$ $\mathcal{I}_{63} = \texttt{false}$ (GR.1)
$c_{64} = (\neg A_2 \vee \neg E_2)^{0.0}$ $\mathcal{I}_{64} = \texttt{false}$ (GR.1)
$c_{65} = (\neg A_2 \vee \neg F_2)^{0.0}$ $\mathcal{I}_{65} = \texttt{false}$ (GR.1)
$c_{66} = (\neg B_2 \vee \neg C_2)^{0.0}$ $\mathcal{I}_{66} = \texttt{false}$ (GR.1)
$c_{67} = (\neg B_2 \vee \neg D_2)^{0.0}$ $\mathcal{I}_{67} = \texttt{false}$ (GR.1)
$c_{68} = (\neg B_2 \vee \neg E_2)^{0.0}$ $\mathcal{I}_{68} = \texttt{false}$ (GR.1)
$c_{69} = (\neg B_2 \vee \neg F_2)^{0.0}$ $\mathcal{I}_{69} = \texttt{false}$ (GR.1)
$c_{70} = (\neg C_2 \vee \neg D_2)^{0.0}$ $\mathcal{I}_{70} = \texttt{false}$ (GR.1)
$c_{71} = (\neg C_2 \vee \neg E_2)^{0.0}$ $\mathcal{I}_{71} = \texttt{false}$ (GR.1)
$c_{72} = (\neg C_2 \vee \neg F_2)^{0.0}$ $\mathcal{I}_{72} = \texttt{false}$ (GR.1)
$c_{73} = (\neg D_2 \vee \neg E_2)^{0.0}$ $\mathcal{I}_{73} = \texttt{false}$ (GR.1)
$c_{74} = (\neg D_2 \vee \neg F_2)^{0.0}$ $\mathcal{I}_{74} = \texttt{false}$ (GR.1)
$c_{75} = (\neg E_2 \vee \neg F_2)^{0.0}$ $\mathcal{I}_{75} = \texttt{false}$ (GR.1)
$c_{76} = (\neg A_1 \vee t_1 \geq 0)^{0.0}$ $\mathcal{I}_{76} = \texttt{false}$ (GR.1)
$c_{77} = (\neg A_1 \vee \vee x_1 \geq 0)^{0.0}$ $\mathcal{I}_{77} = \texttt{false}$ (GR.1)
$c_{78} = (\neg A_1 \vee t_1 = x_1)^{0.0}$ $\mathcal{I}_{78} = \texttt{false}$ (GR.1)
$c_{79} = (\neg A_1 \vee T_1 \leq 10)^{0.0}$ $\mathcal{I}_{79} = \texttt{false}$ (GR.1)
$c_{80} = (\neg A_1 \vee B_2)^{0.0}$ $\mathcal{I}_{80} = \texttt{false}$ (GR.1)
$c_{81} = (\neg A_1 \vee t_2 \geq 2)^{0.0}$ $\mathcal{I}_{81} = \texttt{false}$ (GR.1)
$c_{82} = (\neg A_1 \vee x_2 \geq 0)^{0.0}$ $\mathcal{I}_{82} = \texttt{false}$ (GR.1)
$c_{83} = (\neg A_1 \vee t_2 = x_2 - 2)^{0.0}$ $\mathcal{I}_{83} = \texttt{false}$ (GR.1)
$c_{84} = (\neg A_1 \vee T_2 \leq 10)^{0.0}$ $\mathcal{I}_{84} = \texttt{false}$ (GR.1)
$c_{85} = (\neg B_1 \vee \neg BDC_2 \vee D_2)^{0.0}$ $\mathcal{I}_{85} = \texttt{false}$ (GR.1)
$c_{86} = (\neg B_1 \vee \neg BDC_2 \vee T_2 \leq 10)^{0.0}$ $\mathcal{I}_{86} = \texttt{false}$ (GR.1)
$c_{87} = (\neg B_1 \vee \neg BDC_2 \vee t_2 \geq 0)^{0.0}$ $\mathcal{I}_{87} = \texttt{false}$ (GR.1)
$c_{88} = (\neg B_1 \vee \neg BDC_2 \vee x_2 \geq 0)^{0.0}$ $\mathcal{I}_{88} = \texttt{false}$ (GR.1)
$c_{89} = (\neg B_1 \vee \neg BDC_2 \vee t_2 = x_2 - 2.5)^{0.0}$ $\mathcal{I}_{89} = \texttt{false}$ (GR.1)
$c_{92} = (\neg D_1 \vee E_2)^{0.0}$ $\mathcal{I}_{92} = \texttt{false}$ (GR.1)
$c_{93} = (\neg D_1 \vee t_2 \geq 2)^{0.0}$ $\mathcal{I}_{93} = \texttt{false}$ (GR.1)
$c_{94} = (\neg D_1 \vee x_2 \geq 0)^{0.0}$ $\mathcal{I}_{94} = \texttt{false}$ (GR.1)
$c_{95} = (\neg D_1 \vee t_2 = x_2 - 4.5)^{0.0}$ $\mathcal{I}_{95} = \texttt{false}$ (GR.1)
$c_{96} = (\neg D_1 \vee T_2 \leq 10)^{0.0}$ $\mathcal{I}_{96} = \texttt{false}$ (GR.1)
$c_{97} = (\neg E_1 \vee EFC_2 \vee C_2)^{0.0}$ $\mathcal{I}_{97} = \texttt{false}$ (GR.1)
$c_{98} = (\neg E_1 \vee EFC_2 \vee x_2 \leq 5)^{0.0}$ $\mathcal{I}_{98} = \texttt{false}$ (GR.1)
$c_{99} = (\neg E_1 \vee \neg EFC_2 \vee F_2)^{0.0}$ $\mathcal{I}_{99} = \texttt{false}$ (GR.1)
$c_{100} = (\neg E_1 \vee \neg EFC_2 \vee x_2 \geq 0)^{0.0}$ $\mathcal{I}_{100} = \texttt{false}$ (GR.1)
$c_{101} = (\neg E_1 \vee \neg EFC_2 \vee t_2 = x_2 - 5)^{0.0}$ $\mathcal{I}_{101} = \texttt{false}$ (GR.1)
$c_{102} = (\neg E_1 \vee \neg EFC_2 \vee T_2 \leq 10)^{0.0}$ $\mathcal{I}_{102} = \texttt{false}$ (GR.1)
$c_{103} = (\neg C_1 \vee C_2)^{0.0}$ $\mathcal{I}_{103} = \texttt{false}$ (GR.1)
$c_{104} = (\neg C_1 \vee x_2 \leq 5)^{0.0}$ $\mathcal{I}_{104} = \texttt{false}$ (GR.1)
$c_{105} = (\neg F_1 \vee F_2)^{0.0}$ $\mathcal{I}_{105} = \texttt{false}$ (GR.1)

| | | |
|---|---|---|
| $c_{106} = (\neg F_1 \vee x_2 \geq 0)^{0.0}$ | $\mathcal{I}_{106} = \texttt{false}$ | (GR.1) |
| $c_{107} = (\neg F_1 \vee t_2 \geq 0)^{0.0}$ | $\mathcal{I}_{107} = \texttt{false}$ | (GR.1) |
| $c_{108} = (\neg F_1 \vee t_2 = x_2 - 5)^{0.0}$ | $\mathcal{I}_{108} = \texttt{false}$ | (GR.1) |
| $c_{109} = (C_2)^{0.0}$ | $\mathcal{I}_{109} = \texttt{false}$ | (GR.1) |
| $c_{110} = (x_2 \leq 5)^{0.0}$ | $\mathcal{I}_{110} = \texttt{false}$ | (GR.1) |
| $c_{111} = (\neg A_0 \vee B_0 \vee C_0 \vee D_0 \vee E_0 \vee F_0 \vee x_0 < 0 \vee x_0 > 0 \vee t_0 < 0 \vee t_0 > 0 \vee T_0 > 10 \vee T_0 < 9 \vee A_1 \vee \neg B_1 \vee C_1 \vee D_1 \vee E_1 \vee F_1 \vee t_1 > 2 \vee x_1 > 2 \vee T_1 > 10 \vee A_2 \vee B_2 \vee \neg C_2 \vee D_2 \vee E_2 \vee F_2 \vee BDC_2 \vee x_2 > 5)^{1.0}$ | $\mathcal{I}_{111} = \texttt{DC}$ | (GR.2) |
| . | . | . |
| . | . | . |
| . | . | . |
| $c_{112} = \emptyset^{1.0}$ | $\mathcal{I}_{112} = (((((((\neg D_1 \wedge (((\texttt{DC} \vee t_1 < 2.0) \vee x_1 < 2.0) \vee D_1)) \vee A_1) \vee C_1) \vee E_1) \vee F_1) \vee \neg B_1) \wedge \neg F_1)$ | (GR.3) |

So $\mathcal{I}_{112} = (((((((\neg D_1 \wedge (((\texttt{DC} \vee t_1 < 2.0) \vee x_1 < 2.0) \vee D_1)) \vee A_1) \vee C_1) \vee E_1) \vee F_1) \vee \neg B_1) \wedge \neg F_1)$. In order to maximize the interpolant, we choose the value of DC, to be true. Therefore, $\mathcal{I}_{112} = \neg F_1$. Consequently,

- $\mathcal{I}^1(x) = (\neg F)$.
- $\mathcal{B}^1(x) = \mathcal{B}^0(s) \vee \mathcal{I}^1(x)$.
- $\mathcal{B}^1(x) \not\rightarrow \mathcal{B}^0(x)$ i.e. $(\neg F) \vee (C \wedge x \leq 5) \not\rightarrow (C \wedge x \leq 5)$.

**Third Step**: We over-approximate the reachable states from the target states by replacing $\mathcal{B}^0(x)$ by $\mathcal{B}^1(x)$. Then we apply the exact scheme as before. Also, the clauses from $c_1$ to $c_{108}$ remain the same. We will replace only the target and apply the same procedure as above.

| | | |
|---|---|---|
| $c_{109} = (C_2 \vee \neg F_2)^{0.0}$ | $\mathcal{I}_{109} = \texttt{false}$ | (GR.1) |
| $c_{110} = (x_2 \leq 5 \vee \neg F_2)^{0.0}$ | $\mathcal{I}_{110} = \texttt{false}$ | (GR.1) |
| $c_{111} = (\neg A_0 \vee B_0 \vee C_0 \vee D_0 \vee E_0 \vee F_0 \vee x_0 < 0 \vee x_0 > 0 \vee t_0 < 0 \vee t_0 > 0 \vee T_0 > 10 \vee T_0 < 9 \vee A_1 \vee \neg B_1 \vee C_1 \vee D_1 \vee E_1 \vee F_1 \vee t_1 > 2 \vee x_1 > 2 \vee T_1 > 10 \vee D_2 \vee F_2 \vee BDC_2)^{1.0}$ | $\mathcal{I}_{111} = \texttt{DC}$ | (GR.2) |
| . | . | . |
| . | . | . |
| . | . | . |
| $c_{112} = \emptyset^{0.05}$ | $\mathcal{I}_{112} = ((((((\neg F_1 \wedge (((\texttt{DC} \vee t_1 < 2.0) \vee x_1 < 2.0) \vee F_1)) \vee A_1) \vee C_1) \vee D_1) \vee E_1) \vee \neg B_1)$ | (GR.3) |

$\mathcal{I}_{112} = ((((((\neg F_1 \wedge (((\texttt{DC} \vee t_1 < 2.0) \vee x_1 < 2.0) \vee F_1)) \vee A_1) \vee C_1) \vee D_1) \vee E_1) \vee \neg B_1)$
In order to maximize the interpolant, we choose the value of `DC`, to be `true`. Therefore, $\mathcal{I}_{112} = \neg F_1$. Consequently,

- $\mathcal{I}^2(x) = \neg F$.
- $\mathcal{B}^2(x) = \mathcal{B}^1(x) \vee \mathcal{I}^2(x)$.
- $\mathcal{B}^2(x) \rightarrow \mathcal{B}^1(x)$ i.e. $\neg F \vee (C \wedge x \leq 5) \rightarrow \neg F \vee (C \wedge x \leq 5)$.

So the interpolant stabilizes and that means we have overapproximated the reachable states of the system. Generally, as long as $j$ increases, one gets more precise result in each iteration, however the complexity of the model increases rapidly.

# Bibliography


[ACH$^+$95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.

[ACHH92] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Grossman et al. [GNRR93], pages 209–229.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.

[AGC12] Aws Albarghouthi, Arie Gurfinkel, and Marsha Chechik. Whale: An interpolation-based algorithm for inter-procedural verification. In Kuncak and Rybalchenko [KR12], pages 39–55.

[AHH96] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Software Eng.*, 22(3):181–201, 1996.

[AHS96] Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors. *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, volume 1066 of *Lecture Notes in Computer Science*. Springer, 1996.

[Air16] Aviation Safety Boeing Commercial Airplanes. Statistical summary of commercial jet airplane accidents. Aviation Safety – Boeing Commercial Airplanes, July 2016.

[Ake78] S. B. Akers. Binary decision diagrams. *IEEE Trans. Computers*, 27(6):509–516, 1978.

[Alb15] Aws Albargouthi. *Software Verification with Program-Graph Interpolation and Abstraction.* PhD thesis, University of Toronto, 2015.

[ALGC12] Aws Albarghouthi, Yi Li, Arie Gurfinkel, and Marsha Chechik. Ufo: A framework for abstraction- and interpolation-based software verification. In Madhusudan and Seshia [MS12], pages 672–678.

[Alt95] Russ B. Altman. A probabilistic approach to determining biological structure: integrating uncertain data sources. *Int. J. Hum.-Comput. Stud.*, 42(6):593–616, 1995.

[Alu11] Rajeev Alur. Formal verification of hybrid systems. In Samarjit Chakraborty, Ahmed Jerraya, Sanjoy K. Baruah, and Sebastian Fischmeister, editors, *Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9-14, 2011*, pages 273–278. ACM, 2011.

[AM13a] Martin Fränzle Ahmed Mahdi. Resolution for stochastic modulo theories. Technical report, Carl von Ossietzky University, Escherweg .2 26122, December 2013.

[AM13b] Aws Albarghouthi and Kenneth L. McMillan. Beautiful interpolants. In Sharygina et al. [S+13], pages 313–329.

[And95] Henrik Reif Andersen. Partial model checking (extended abstract). In *LICS*, pages 398–407. IEEE Computer Society, 1995.

[AS12] Stephan Arlt and Martin Schäf. Joogie: Infeasible code detection for java. In Madhusudan and Seshia [MS12], pages 767–773.

[AWD+14] Sergio Feo Arenis, Bernd Westphal, Daniel Dietsch, Marco Muñiz, and Ahmad Siyar Andisha. The wireless fire alarm system: Ensuring conformance to industrial standards through formal verification. In Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun, editors, *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, volume 8442 of *Lecture Notes in Computer Science*, pages 658–672. Springer, 2014.

[BB06] Constantinos Bartzis and Tevfik Bultan. Efficient bdds for bounded arithmetic constraints. *STTT*, 8(1):26–36, 2006.

[BCC+11] Julien Bertrane, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival. Static analysis by abstract interpretation of embedded critical software. *ACM SIGSOFT Software Engineering Notes*, 36(1):1–8, 2011.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In Rance Cleaveland, editor, *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.

[BCD+05] Michael Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs, and K. Rustan M. Leino. Boogie: A modular reusable verifier for object-oriented programs. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures*, volume 4111 of *Lecture Notes in Computer Science*, pages 364–387. Springer, 2005.

[BCKS08] Surendra Bommu, Kameshwar Chandrasekar, Rahul Kundu, and Sanjay

Sengupta. CONCAT: conflict driven learning in ATPG for industrial designs. In Douglas Young and Nur A. Touba, editors, *2008 IEEE International Test Conference, ITC 2008, Santa Clara, California, USA, October 26-31, 2008*, pages 1–10. IEEE, 2008.

[BDG+13] Martin Brain, Vijay D'Silva, Alberto Griggio, Leopold Haller, and Daniel Kroening. Interpolation-based verification of floating-point programs with abstract CDCL. In Francesco Logozzo and Manuel Fähndrich, editors, *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*, volume 7935 of *Lecture Notes in Computer Science*, pages 412–432. Springer, 2013.

[BDG+14] Martin Brain, Vijay D'Silva, Alberto Griggio, Leopold Haller, and Daniel Kroening. Deciding floating-point logic with abstract conflict driven clause learning. *Formal Methods in System Design*, 45(2):213–245, 2014.

[BDL04] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.

[BDP03] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. DPLL with caching: A new algorithm for #sat and bayesian inference. *Electronic Colloquium on Computational Complexity ECCC*, 10(003), 2003.

[Bel57] Richard Bellman. A Markovian Decision Process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[Ben96] Frédéric Benhamou. Heterogeneous constraint solving. In Michael Hanus and Mario Rodríguez-Artalejo, editors, *ALP*, volume 1139 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 1996.

[Ber00] Janick Bergeron. *Writing testbenches : functional verification of HDL models.* Kluwer Academic, Boston, 2000. Index.

[BG96] Frédéric Benhamou and Laurent Granvilliers. Combining local consistency, symbolic rewriting and interval methods. In *Artificial Intelligence and Symbolic Mathematical Computation, International Conference AISMC-3, Steyr, Austria, September 23-25, 1996, Proceedings*, pages 144–159, 1996.

[BG06] Frédéric Benhamou and Laurent Granvilliers. Continuous and interval constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 16. Elsevier, 2006.

[BGD11] Maria Domenica Di Benedetto, Stefano Di Gennaro, and Alessandro D'Innocenzo. Verification of hybrid automata diagnosability by abstraction. *IEEE TAC*, 56(9):2050–2061, 2011.

[BHJM07] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The software model checker blast. *STTT*, 9(5-6):505–525, 2007.

[BHT07] Dirk Beyer, Thomas A. Henzinger, and Grégory Théoduloz. Configurable software verification: Concretizing the convergence of model checking and program analysis. In *CAV*, pages 504–518, 2007.

[BJ06] Thomas Ball and Robert B. Jones, editors. *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*. Springer, 2006.

[BK04] Per Bjesse and James H. Kukula. Using counter example guided abstraction refinement to find complex bugs. In *DATE*, pages 156–161. IEEE Computer Society, 2004.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BK09] Dirk Beyer and M. Erkan Keremoglu. Cpachecker: A tool for configurable software verification. *CoRR*, abs/0902.0019, 2009.

[BK11] Federico Bergero and Ernesto Kofman. Powerdevs: a tool for hybrid system modeling and real-time simulation. *Simulation*, 87(1-2):113–132, 2011.

[BKRW10] Angelo Brillout, Daniel Kroening, Philipp Rümmer, and Thomas Wahl. Beyond quantifier-free interpolation in extensions of presburger arithmetic (extended technical report). *CoRR*, abs/1011.1036, 2010.

[BKW08] Angelo Brillout, Daniel Kroening, and Thomas Wahl. Craig interpolation for quantifier-free presburger arithmetic. *CoRR*, abs/0811.3521, 2008.

[BL12] Dirk Beyer and Stefan Löwe. Explicit-value analysis based on cegar and interpolation. *CoRR*, abs/1212.6542, 2012.

[BLL+95] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In Alur et al. [AHS96], pages 232–243.

[BLW86] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, New York, NY, USA, 1986.

[BMH94] Frédéric Benhamou, David A. McAllester, and Pascal Van Hentenryck. Clp(intervals) revisited. In Maurice Bruynooghe, editor, *ILPS*, pages 124–138. MIT Press, 1994.

[BP01] Dimitris Bertsimas and Ioannis Ch. Paschalidis. Probabilistic service level guarantees in make-to-stock manufacturing systems. *Operations Research*, 49(1):119–133, 2001.

[BR00] Thomas Ball and Sriram K. Rajamani. Bebop: A symbolic model checker for boolean programs. In Klaus Havelund, John Penix, and Willem Visser, editors, *SPIN Model Checking and Software Verification, 7th International SPIN Workshop, Stanford, CA, USA, August 30 - September 1, 2000, Proceedings*, volume 1885 of *Lecture Notes in Computer Science*, pages 113–130. Springer, 2000.

[Bra06] Mark Braverman. Termination of integer linear programs. In Ball and Jones [BJ06], pages 372–385.

[Bro98] Manfred Broy. A functional rephrasing of the assumption/commitment specification style. *Formal Methods in System Design*, 13(1):87–119, 1998.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[BS92] Mark Bickford and Mandayam K. Srivas. Verification of a fault-tolerant property of a multiprocessor system: A case study in theorem prover-based verification. In Victoria Stavridou, Thomas F. Melham, and Raymond T. Boute, editors, *Theorem Provers in Circuit Design, Proceedings of the IFIP TC10/WG 10.2 International Conference on Theorem Provers in Circuit Design: Theory, Practice and Experience, Nijmegen, The Netherlands, 22-24 June 1992, Proceedings*, volume A-10 of *IFIP Transactions*, pages 225–251. North-Holland, 1992.

[BS12] Dirk Beyer and Andreas Stahlbauer. BDD-based software model checking with CPAchecker. In Antonín Kucera, Thomas A. Henzinger, Jaroslav Nesetril, Tomás Vojnar, and David Antos, editors, *MEMICS*, volume 7721 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2012.

[BS14] Dirk Beyer and Andreas Stahlbauer. Bdd-based software verification - applications to event-condition-action systems. *STTT*, 16(5):507–518, 2014.

[Bue62] Julius R. Buechi. On a Decision Method in Restricted Second-Order Arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[BW12] Dirk Beyer and Philipp Wendler. Algorithms for software model checking: Predicate abstraction vs. impact. In Gianpiero Cabodi and Satnam Singh, editors, *FMCAD*, pages 106–113. IEEE, 2012.

[CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. ISOP'76*, pages 106–130. Dunod, Paris, France, 1976.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977.

[CC92] Patrick Cousot and Radhia Cousot. Abstract interpretation and application to logic programs. *J. Log. Program.*, 13(2&3):103–179, 1992.

[CC14] Patrick Cousot and Radhia Cousot. Abstract interpretation: past, present and future. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, page 2. ACM, 2014.

[CCC+93] Chrysler Corporation, Ford Motor Company, General Motors Corporation, American Society for Quality Control, and Automotive Industry Action Group. *Potential Failure Mode and Effects Analysis (FMEA): Reference Manual.* Chrysler Corporation, 1993.

[CCK11] Hong-Zu Chou, Kai-Hui Chang, and Sy-Yen Kuo. Facilitating unreachable code diagnosis and debugging. In *Proceedings of the 16th Asia South Pacific Design Automation Conference, ASP-DAC 2011, Yokohama, Japan, January 25-27, 2011*, pages 485–490. IEEE, 2011.

[CDH+00] James C. Corbett, Matthew B. Dwyer, John Hatcliff, Shawn Laubach, Corina S. Pasareanu, Robby, and Hongjun Zheng. Bandera: extracting finite-state models from java source code. In Carlo Ghezzi, Mehdi Jazayeri, and Alexander L. Wolf, editors, *Proceedings of the 22nd International Conference on on Software Engineering, ICSE 2000, Limerick Ireland, June 4-11, 2000.*, pages 439–448. ACM, 2000.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CF87] William W. Carlson and José A. B. Fortes. On the performance of combined data flow and control flow systems: Experiments using two iterative algorithms. In *International Conference on Parallel Processing, ICPP'87, University Park, PA, USA, August 1987.*, pages 671–679. Pennsylvania State University Press, 1987.

[CFG+10] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Krishnamani Kalyanasundaram, and Marco Roveri. Tighter integration of bdds and smt for predicate abstraction. In *DATE*, pages 1707–1712. IEEE, 2010.

[CFH+03] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci.*, 14(4):583–604, 2003.

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[CGK97] Yih-Farn Chen, Emden R. Gansner, and Eleftherios Koutsofios. A C++ data model supporting reachability analysis and dead code detection. In Mehdi Jazayeri and Helmut Schauer, editors, *Software Engineering - ESEC/FSE '97, 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on Foundations of Software Engi-*

*neering, Zurich, Switzerland, September 22-25, 1997, Proceedings*, volume 1301 of *Lecture Notes in Computer Science*, pages 414–431. Springer, 1997.

[CGK98] Yih-Farn Chen, Emden R. Gansner, and Eleftherios Koutsofios. AC++ data model supporting reachability analysis and dead code detection. *IEEE Trans. Software Eng.*, 24(9):682–694, 1998.

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking.* MIT Press, 2001.

[CGP03] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning assumptions for compositional verification. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, volume 2619 of *Lecture Notes in Computer Science*, pages 331–346. Springer, 2003.

[CGS04] Edmund M. Clarke, Anubhav Gupta, and Ofer Strichman. Sat-based counterexample-guided abstraction refinement. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(7):1113–1123, 2004.

[CGS08] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In Ramakrishnan and Rehof [RR08], pages 397–412.

[CGS10] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient generation of craig interpolants in satisfiability modulo theories. *ACM Trans. Comput. Log.*, 12(1):7, 2010.

[Che09] James Chelini. Working towards do-178c/ed-12c, do-248c/ed-94c, and DO-278A/ED109A. In Greg Gicca and Jeff Boleng, editors, *Proceedings of the 2009 Annual ACM SIGAda International Conference on Ada, Saint Petersburg, Florida, USA, November 1-5, 2009*, pages 103–104. ACM, 2009.

[CHN12] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. SMTInterpol: An interpolating SMT solver. In Alastair F. Donaldson and David Parker, editors, *SPIN*, volume 7385 of *Lecture Notes in Computer Science*, pages 248–254. Springer, 2012.

[CHN13] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. Proof tree preserving interpolation. In Piterman and Smolka [PS13b], pages 124–138.

[CHS12] Jürgen Christ, Jochen Hoenicke, and Martin Schäf. Towards bounded infeasible code detection. *CoRR*, abs/1205.6527, 2012.

[CKL04] Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ANSI-C programs. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, pages 168–176, 2004.

[Cla03] Edmund M. Clarke. Sat-based counterexample guided abstraction refinement in model checking. In Franz Baader, editor, *CADE*, volume 2741 of *Lecture Notes in Computer Science*, page 1. Springer, 2003.

[CNQ03] Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. Improving sat-based bounded model checking by means of BDD-based approximate traversals. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10898–10905. IEEE Computer Society, 2003.

[Coo72] D.C. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7:91–99, 1972.

[Cou12] Patrick Cousot. Formal verification by abstract interpretation. In Alwyn Goodloe and Suzette Person, editors, *NASA Formal Methods - 4th International Symposium, NFM 2012, Norfolk, VA, USA, April 3-5, 2012. Proceedings*, volume 7226 of *Lecture Notes in Computer Science*, pages 3–7. Springer, 2012.

[Cra57] William Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, 22(3):269–285, 1957.

[CW96] Edmund M. Clarke and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643, 1996.

[Dam08] Werner Damm. Contract-based analysis of automotive and avionics applications: The SPEEDS approach. In Darren D. Cofer et al., editors, *FMICS*, volume 5596 of *LNCS*, page 3. Springer, 2008.

[Dan63] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[Dav84] M. H. A. Davis. Piecewise-deterministic Markov processes: a general class of nondiffusion stochastic models. *J. Roy. Statist. Soc. Ser. B*, 46(3):353–388, 1984. With discussion.

[DE73] George B. Dantzig and B. Curtis Eaves. Fourier-motzkin elimination and its dual. *J. Comb. Theory, Ser. A*, 14(3):288–297, 1973.

[DEMS00] Saumya K. Debray, William S. Evans, Robert Muth, and Bjorn De Sutter. Compiler techniques for code compaction. *ACM Trans. Program. Lang. Syst.*, 22(2):378–415, 2000.

[DH88] James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *J. Symb. Comput.*, 5(1/2):29–35, 1988.

[DHH+06] Matthew B. Dwyer, John Hatcliff, Matthew Hoosier, Venkatesh Prasad Ranganath, Robby, and Todd Wallentine. Evaluating the effectiveness of slicing for model reduction of concurrent object-oriented programs. In Hermanns and Palsberg [HP06], pages 73–89.

[DHJ+11] Werner Damm, Hardi Hungar, Bernhard Josko, Thomas Peikenkamp, and Ingo Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *Design, Automation and*

*Test in Europe, DATE 2011, Grenoble, France, March 14-18, 2011*, pages 1023–1028. IEEE, 2011.

[DHK13] Vijay D'Silva, Leopold Haller, and Daniel Kroening. Abstract conflict driven learning. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 143–154. ACM, 2013.

[DHKR11] Alastair F. Donaldson, Leopold Haller, Daniel Kroening, and Philipp Rümmer. Software verification using k-induction. In Eran Yahav, editor, *Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings*, volume 6887 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2011.

[DHKT12] Vijay D'Silva, Leopold Haller, Daniel Kroening, and Michael Tautschnig. Numeric bounds analysis with conflict-driven learning. In Flanagan and König [FK12], pages 48–63.

[Dij72] Edsger W. Dijkstra. Notes on structured programming. In O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors, *Structured Programming*, pages 1–82. Academic Press Ltd., London, UK, 1972.

[DIN97] DIN. Fire detection and fire alarm systems. Technical report, German version En54, 1997.

[Din13] Nam Thang Dinh. Dead code analysis using satisfiability checking. Master's thesis, Carl von Ossietzky Universität Oldenburg, 2013.

[DKK+12] Alastair F. Donaldson, Alexander Kaiser, Daniel Kroening, Michael Tautschnig, and Thomas Wahl. Counterexample-guided abstraction refinement for symmetric concurrent programs. *Formal Methods in System Design*, 41(1):25–44, 2012.

[DKPW10] Vijay D'Silva, Daniel Kroening, Mitra Purandare, and Georg Weissenbacher. Interpolant strength. In Gilles Barthe and Manuel V. Hermenegildo, editors, *Verification, Model Checking, and Abstract Interpretation, 11th International Conference, VMCAI 2010, Madrid, Spain, January 17-19, 2010. Proceedings*, volume 5944 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2010.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[dMB08] Leonardo Mendonca de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In Ramakrishnan and Rehof [RR08], pages 337–340.

[DP96] Ferruccio Damiani and Frédéric Prost. Detecting and removing dead-code using rank 2 intersection. In Eduardo Giménez and Christine Paulin-Mohring, editors, *Types for Proofs and Programs, International Workshop TYPES'96, Aussois, France, December 15-19, 1996, Selected Papers*, volume 1512 of *Lecture Notes in Computer Science*, pages 66–87. Springer, 1996.

[DT13] Parasara Sridhar Duggirala and Ashish Tiwari. Safety verification for linear systems. In *Proceedings of the International Conference on Embedded Software, EMSOFT 2013, Montreal, QC, Canada, September 29 - Oct. 4, 2013*, pages 7:1–7:10. IEEE, 2013.

[DZT14] Sun Ding, Hongyu Zhang, and Hee Beng Kuan Tan. Detecting infeasible branches based on code patterns. In Serge Demeyer, Dave Binkley, and Filippo Ricca, editors, *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*, pages 74–83. IEEE Computer Society, 2014.

[EFH08] Andreas Eggers, Martin Fränzle, and Christian Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In Sungdeok (Steve) Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis (ATVA'08)*, volume 5311 of *Lecture Notes in Computer Science (LNCS)*, pages 171–185. Springer-Verlag, 2008.

[EH10] Michael R. Elliott and Peter Heller. Object-oriented software considerations in airborne systems and equipment certification. In William R. Cook, Siobhán Clarke, and Martin C. Rinard, editors, *Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA, 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA*, pages 85–96. ACM, 2010.

[EHP12] Evren Ermis, Jochen Hoenicke, and Andreas Podelski. Splitting via interpolants. In Kuncak and Rybalchenko [KR12], pages 186–201.

[EKS06] Javier Esparza, Stefan Kiefer, and Stefan Schwoon. Abstraction refinement with Craig interpolation and symbolic pushdown systems. In Hermanns and Palsberg [HP06], pages 489–503.

[EKS08] Javier Esparza, Stefan Kiefer, and Stefan Schwoon. Abstraction refinement with craig interpolation and symbolic pushdown systems. *JSAT*, 5(1-4):27–56, 2008.

[ERNF11] Andreas Eggers, Nacim Ramdani, Nedialko S. Nedialkov, and Martin Fränzle. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *Software Engineering and Formal Methods - 9th International Conference, SEFM 2011, Montevideo, Uruguay, November 14-18, 2011. Proceedings*, volume 7041 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2011.

[FB13] Martin Fränzle and Bernd Becker. Accurate dead code detection in embedded c code by arithmetic constraint solving, 2013. Project proposal.

[FGD+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification -*

*23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 379–395. Springer, 2011.

[FH07] Martin Fränzle and Christian Herde. Hysat: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

[FHH+11] Martin Fränzle, Ernst Moritz Hahn, Holger Hermanns, Nicolás Wolovick, and Lijun Zhang. Measurability and safety verification for stochastic hybrid systems. In Marco Caccamo, Emilio Frazzoli, and Radu Grosu, editors, *HSCC*, pages 43–52. ACM, 2011.

[FHR+07] Martin Fränzle, Christian Herde, Stefan Ratschan, Tobias Schubert, and Tino Teige. Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation – Special Issue on SAT/CP Integration*, 1:209–236, 2007.

[FHT+07] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *JSAT*, 1(3-4):209–236, 2007.

[FHT08] Martin Fränzle, Holger Hermanns, and Tino Teige. Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems. In Magnus Egerstedt and Bud Mishra, editors, *Hybrid Systems: Computation and Control, 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings*, volume 4981 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2008.

[FK03] Eric Freudenthal and Vijay Karamcheti. Qtm: Trust management with quantified stochastic attributes. Technical Report NYU Computer Science Technical Report TR2003-848, Courant Institute of Mathematical Sciences , New York University, 2003.

[FK12] Cormac Flanagan and Barbara König, editors. *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7214 of *Lecture Notes in Computer Science*. Springer, 2012.

[FR75] Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM J. Comput.*, 4(1):69–76, 1975.

[Fre08] Goran Frehse. Phaver: algorithmic verification of hybrid systems past hytech. *STTT*, 10(3):263–279, 2008.

[FTE10] Martin Fränzle, Tino Teige, and Andreas Eggers. Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. *J. Log. Algebr. Program.*, 79(7):436–466, 2010.

[FW16] Moritz Freidank and Bernd Westphal. Detection of non-supporting edges in networks of timed automata. Technical report, Albert Ludwigs-Universität Freiburg, 2016.

[GB03] Jack Ganssle and Michael Barr. *Embedded Systems Dictionary.* CMP Books, 2003.

[GD98] Shankar G. Govindaraju and David L. Dill. Verification by approximate forward and backward reachability. In Hiroto Yasuura, editor, *Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1998, San Jose, CA, USA, November 8-12, 1998*, pages 366–370. ACM / IEEE Computer Society, 1998.

[GF15] Damien Gayle and David Feeney. Spanish air force cargo plane on test flight crashes near seville airport. *The Guardian*, 2015.

[GGW+03] Aarti Gupta, Malay K. Ganai, Chao Wang, Zijiang Yang, and Pranav Ashar. Learning from bdds in sat-based bounded model checking. In *Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003*, pages 824–829. ACM, 2003.

[GKC13] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dreal: An SMT solver for nonlinear theories over the reals. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 208–214. Springer, 2013.

[GKT09] Amit Goel, Sava Krstic, and Cesare Tinelli. Ground interpolation for combined theories. In Schmidt [Sch09], pages 183–198.

[GLS11] Alberto Griggio, Thi Thieu Hoa Le, and Roberto Sebastiani. Efficient interpolant generation in satisfiability modulo linear integer arithmetic. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 2011.

[GNRR93] Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*. Springer, 1993.

[Gom58] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275–278, 1958.

[Gom10] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs *and* an algorithm for the mixed integer problem. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 77–103. Springer, 2010.

[GS97] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In Orna Grumberg, editor, *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*,

volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1997.

[GZ16] Sicun Gao and Damien Zufferey. Interpolants in nonlinear theories over the reals. In Marsha Chechik and Jean-Francois Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 625–641. Springer, 2016.

[H$^+$13] Frédéric Herbreteau et al. Lazy abstractions for timed automata. In Sharygina et al. [S$^+$13], pages 990–1005.

[HA07] Mohamed Hefeeda and Hossein Ahmadi. Network connectivity under probabilistic communication models in wireless sensor networks. In *IEEE 4th International Conference on Mobile Adhoc and Sensor Systems, MASS 2007, 8-11 October 2007, Pisa, Italy*, pages 1–9. IEEE Computer Society, 2007.

[Hac74] Michel Hack. The recursive equivalence of the reachability problem and the liveness problem for petri nets and vector addition systems. In *15th Annual Symposium on Switching and Automata Theory, New Orleans, Louisiana, USA, October 14-16, 1974*, pages 156–164. IEEE Computer Society, 1974.

[HCD$^+$13] Matthias Heizmann, Jürgen Christ, Daniel Dietsch, Evren Ermis, Jochen Hoenicke, Markus Lindenmann, Alexander Nutz, Christian Schilling, and Andreas Podelski. Ultimate automizer with SMTInterpol - (competition contribution). In Piterman and Smolka [PS13b], pages 641–643.

[Hea02] Steve Heath. 1 - what is an embedded system? In Steve Heath, editor, *Embedded Systems Design (Second Edition)*, pages 1 – 14. Newnes, Oxford, second edition edition, 2002.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 278–292. IEEE Computer Society, 1996.

[Her11] Christian Herde. *Efficient solving of large arithmetic constraint systems with complex Boolean structure: proof engines for the analysis of hybrid discrete-continuous systems.* PhD thesis, Carl von Ossietzky University of Oldenburg, 2011.

[HHMW00] Thomas A. Henzinger, Benjamin Horowitz, Rupak Majumdar, and Howard Wong-Toi. Beyond HYTECH: hybrid systems analysis using interval numerical methods. In Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control, Third International Workshop, HSCC 2000, Pittsburgh, PA, USA, March 23-25, 2000, Proceedings*, volume 1790 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2000.

[HHP09] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Refinement of trace abstraction. In Jens Palsberg and Zhendong Su, editors, *Static*

*Analysis, 16th International Symposium, SAS 2009, Los Angeles, CA, USA, August 9-11, 2009. Proceedings*, volume 5673 of *Lecture Notes in Computer Science*, pages 69–85. Springer, 2009.

[HHP10] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Nested interpolants. In Manuel V. Hermenegildo and Jens Palsberg, editors, *POPL*, pages 471–482. ACM, 2010.

[HJMM04] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In Neil D. Jones and Xavier Leroy, editors, *POPL*, pages 232–244. ACM, 2004.

[HJMS02] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Lazy abstraction. In John Launchbury and John C. Mitchell, editors, *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002*, pages 58–70. ACM, 2002.

[HKV12] Krystof Hoder, Laura Kovács, and Andrei Voronkov. Playing in the grey area of proofs. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 259–272. ACM, 2012.

[HP06] Holger Hermanns and Jens Palsberg, editors. *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings*, volume 3920 of *Lecture Notes in Computer Science*. Springer, 2006.

[Hua95] Guoxiang Huang. Constructing craig interpolation formulas. In Ding-Zhu Du and Ming Li, editors, *COCOON*, volume 959 of *Lecture Notes in Computer Science*, pages 181–190. Springer, 1995.

[Hun73] G. Hunter. *Metalogic: An Introduction to the Metatheory of Standard First Order Logic*. Macmillan Student Editions. University of California Press, 1973.

[HWZ08] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic CEGAR. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, volume 5123 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2008.

[IEE85] IEEE. *IEEE standard for binary floating-point arithmetic*. Institute of Electrical and Electronics Engineers, New York, 1985. Note: Standard 754–1985.

[Int96] SAE International. SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, Dec 1996.

[Jha06] Sumit Kumar Jha. Numerical simulation guided lazy abstraction refinement for nonlinear hybrid automata. *CoRR*, abs/cs/0611051, 2006.

[JJ04] Agata Janowska and Pawel Janowski. Slicing timed systems. *FI*, 60(1-4):187–210, 2004.

[JM05] Ranjit Jhala and Rupak Majumdar. Path slicing. In Vivek Sarkar and Mary W. Hall, editors, *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005*, pages 38–47. ACM, 2005.

[JRH05] Ganeshan Jayaraman, Venkatesh Prasad Ranganath, and John Hatcliff. Kaveri: Delivering the indus java program slicer to eclipse. In Maura Cerioli, editor, *Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3442 of *Lecture Notes in Computer Science*, pages 269–272. Springer, 2005.

[JS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In Benjamin Kuipers and Bonnie L. Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island.*, pages 203–208. AAAI Press / The MIT Press, 1997.

[KB11] Stefan Kupferschmid and Bernd Becker. Craig interpolation in the presence of non-linear constraints. In Uli Fahrenberg and Stavros Tripakis, editors, *Formal Modeling and Analysis of Timed Systems - 9th International Conference, FORMATS 2011, Aalborg, Denmark, September 21-23, 2011. Proceedings*, volume 6919 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011.

[KBTF11] Stefan Kupferschmid, Bernd Becker, Tino Teige, and Martin Fränzle. Proof certificates and non-linear arithmetic constraints. In Rolf Kraemer, Adam Pawlak, Andreas Steininger, Mario Schölzel, Jaan Raik, and Heinrich Theodor Vierhaus, editors, *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS 2011, Cottbus, Germany, April 13-15, 2011*, pages 429–434. IEEE, 2011.

[KEB+14] J. Krizan, L. Ertl, M. Bradac, M. Jasansky, and A. Andreev. Automatic code generation from matlab/simulink for critical applications. In *IEEE 27th Canadian Conference on Electrical and Computer Engineering, CCECE 2014, Toronto, ON, Canada, May 4-7, 2014*, pages 1–6. IEEE, 2014.

[Kha09] Leonid Khachiyan. Fourier-motzkin elimination method. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1074–1077. Springer, 2009.

[KKZ05] Joost-Pieter Katoen, Maneesh Khattri, and Ivan S. Zapreev. A markov reward model checker. In *Second International Conference on the Quanti-*

*tative Evaluaiton of Systems (QEST 2005), 19-22 September 2005, Torino, Italy*, pages 243–244. IEEE Computer Society, 2005.

[Kno96] Jens Knoop. Partial dead code elimination for parallel programs. In Luc Bougé, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, *Euro-Par '96 Parallel Processing, Second International Euro-Par Conference, Lyon, France, August 26-29, 1996, Proceedings, Volume I*, volume 1123 of *Lecture Notes in Computer Science*, pages 441–450. Springer, 1996.

[KNP02] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: probabilistic symbolic model checker. In Tony Field, Peter G. Harrison, Jeremy T. Bradley, and Uli Harder, editors, *Computer Performance Evaluation, Modelling Techniques and Tools 12th International Conference, TOOLS 2002, London, UK, April 14-17, 2002, Proceedings*, volume 2324 of *Lecture Notes in Computer Science*, pages 200–204. Springer, 2002.

[KR12] Viktor Kuncak and Andrey Rybalchenko, editors. *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*, volume 7148 of *Lecture Notes in Computer Science*. Springer, 2012.

[Kru07] Rudolf Kruse. Probabilistic graphical models for data mining and planning in automotive industry. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), October 29-31, 2007, Patras, Greece, Volume 1*. IEEE Computer Society, 2007.

[KSU11] Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Predicate abstraction and CEGAR for higher-order model checking. In Mary W. Hall and David A. Padua, editors, *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 222–233. ACM, 2011.

[KT14] Daniel Kroening and Michael Tautschnig. CBMC - C bounded model checker - (competition contribution). In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 389–391. Springer, 2014.

[Kup13] Stefan Kupferschmid. *Über Craigsche Interpolation und deren Anwendung in der formalen Modellprüfung.* PhD thesis, Albert-Ludwigs-Universität Freiburg im Breisgau, 2013.

[Kur94] Robert P. Kurshan. *Computer-aided Verification of Coordinating Processes: The Automata-theoretic Approach.* Princeton University Press, Princeton, NJ, USA, 1994.

[KV09] Laura Kovács and Andrei Voronkov. Interpolation and symbol elimination. In Schmidt [Sch09], pages 199–213.

[Lam87] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Trans. Comput. Syst.*, 5(1):1–11, 1987.

[LH10] Weiyi Liu and Inseok Hwang. Probabilistic 4d trajectory prediction and conflict detection for air traffic control. In *Proceedings of the 49th IEEE Conference on Decision and Control, CDC 2010, December 15-17, 2010, Atlanta, Georgia, USA*, pages 1183–1188. IEEE, 2010.

[LMM06] Alexe E. Leu, Mark McHenry, and Brian L. Mark. Modeling and analysis of interference in listen-before-talk spectrum access schemes. *Int. Journal of Network Management*, 16(2):131–147, 2006.

[LSW95] Kim Guldstrand Larsen, Bernhard Steffen, and Carsten Weise. Fischer's protocol revisited: A simple proof using modal constraints. In Alur et al. [AHS96], pages 604–615.

[LT08] Christopher Lynch and Yuefeng Tang. Interpolants for linear arithmetic in SMT. In Sung Deok Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *Automated Technology for Verification and Analysis, 6th International Symposium, ATVA 2008, Seoul, Korea, October 20-23, 2008. Proceedings*, volume 5311 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 2008.

[Mah12] Ahmed Mahdi. Compositional verification of computation path dependent real-time systems properties. Master's thesis, University of Freiburg, April 2012.

[Mas01] Damien Massé. Combining forward and backward analyses of temporal properties. In Olivier Danvy and Andrzej Filinski, editors, *Programs as Data Objects, Second Symposium, PADO 2001, Aarhus, Denmark, May 21-23, 2001, Proceedings*, volume 2053 of *Lecture Notes in Computer Science*, pages 103–116. Springer, 2001.

[MC81] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Trans. Software Eng.*, 7(4):417–426, 1981.

[McM92] Kenneth Lauchlin McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1992. UMI Order No. GAX92-24209.

[McM03] Kenneth L. McMillan. Interpolation and sat-based model checking. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

[McM05] Kenneth L. McMillan. Applications of craig interpolants in model checking. In Nicolas Halbwachs and Lenore D. Zuck, editors, *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[McM06] Kenneth L. McMillan. Lazy abstraction with interpolants. In Ball and Jones [BJ06], pages 123–136.

[McM10] Kenneth L. McMillan. Lazy annotation for program testing and verification. In Touili et al. [TCJ10], pages 104–118.

[Mey92] Bertrand Meyer. Applying "design by contract". *IEEE Computer*, 25(10):40–51, 1992.

[MF14] Ahmed Mahdi and Martin Fränzle. Generalized craig interpolation for stochastic satisfiability modulo theory problems. In Ouaknine et al. [OPW14], pages 203–215.

[MFH+06] Roman Manevich, John Field, Thomas A. Henzinger, G. Ramalingam, and Mooly Sagiv. Abstract counterexample-based refinement for powerset domains. In Thomas W. Reps, Mooly Sagiv, and Jörg Bauer, editors, *Program Analysis and Compilation*, volume 4444 of *Lecture Notes in Computer Science*, pages 273–292. Springer, 2006.

[MIL] *MIL-STD-1629 - Procedures for performing a failure mode effect and criticality analysis.*

[Mit07] Ian M. Mitchell. Comparing forward and backward reachability as tools for safety analysis. In Alberto Bemporad, Antonio Bicchi, and Giorgio C. Buttazzo, editors, *HSCC*, volume 4416 of *Lecture Notes in Computer Science*, pages 428–443. Springer, 2007.

[MJ11] Ahmed Mahdi and Oday Jubran. Formal analysis of message collision due to clock drift and dynamic message scheduling in a wireless sensor network. Technical report, Albert Ludwigs-Universität Freiburg, 2011.

[ML98] Stephen M. Majercik and Michael L. Littman. Maxplan: A new approach to probabilistic planning. In Reid G. Simmons, Manuela M. Veloso, and Stephen F. Smith, editors, *AIPS*, pages 86–93. AAAI, 1998.

[ML03] Stephen M. Majercik and Michael L. Littman. Contingent planning under uncertainty via stochastic satisfiability. *Artif. Intell.*, 147(1-2):119–162, 2003.

[Moo95] Ramon E. Moore. *Methods and applications of interval analysis*. SIAM studies in applied mathematics. SIAM, 1995.

[MS12] P. Madhusudan and Sanjit A. Seshia, editors. *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*. Springer, 2012.

[MSN+16] Ahmed Mahdi, Karsten Scheibler, Felix Neubauer, Martin Fränzle, and Bernd Becker. Advancing software model checking beyond linear arithmetic theories. In Roderick Bloem and Eli Arbel, editors, *Hardware and Software: Verification and Testing - 12th International Haifa Verification Conference, HVC 2016, Haifa, Israel, November 14-17, 2016, Proceedings*, volume 10028 of *Lecture Notes in Computer Science*, pages 186–201, 2016.

[MW+12] Marco Muñiz, Bernd Westphal, et al.

Timed automata with disjoint activity. In Marcin Jurdzinski et al., editors, *FORMATS*, volume 7595 of *LNCS*, pages 188–203. Springer, 2012.

[MWF14] Ahmed Mahdi, Bernd Westphal, and Martin Fränzle. Transformations for compositional verification of assumption-commitment properties. In Ouaknine et al. [OPW14], pages 216–229.

[NNH05] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of program analysis (2. corr. print)*. Springer, 2005.

[NO79] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.

[NOK10] Takeshi Nagaoka, Kozo Okano, and Shinji Kusumoto. An abstraction refinement technique for timed automata based on counterexample-guided abstraction refinement loop. *IEICE Transactions*, 93-D(5):994–1005, 2010.

[Obe68] Arnold Oberschelp. On the craig-lyndon interpolation theorem. *J. Symb. Log.*, 33(2):271–274, 1968.

[OD08] Ernst-Rüdiger Olderog and Henning Dierks. *Real-time systems*. Cambridge University Press, 2008.

[OPW14] Joël Ouaknine, Igor Potapov, and James Worrell, editors. *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, volume 8762 of *Lecture Notes in Computer Science*. Springer, 2014.

[OS13] Ernst-Rüdiger Olderog and Mani Swaminathan. Structural transformations for data-enriched real-time systems. In Einar Broch Johnsen and Luigia Petre, editors, *Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings*, volume 7940 of *Lecture Notes in Computer Science*, pages 378–393. Springer, 2013.

[OS15] Ernst-Rüdiger Olderog and Mani Swaminathan. Structural transformations for data-enriched real-time systems. *Formal Asp. Comput.*, 27(4):727–750, 2015.

[Pap85] Christos H. Papadimitriou. Games against nature. *J. Comput. Syst. Sci.*, 31(2):288–301, 1985.

[Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[PK02] Lawrence T. Pileggi and Andreas Kuehlmann, editors. *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, 2002, San Jose, California, USA, November 10-14, 2002*. ACM, 2002.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[PQ08] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In Alessandro Armando, Peter

Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 2008.

[PS13a] Florian Pigorsch and Christoph Scholl. Lemma localization: a practical method for downsizing smt-interpolants. In Enrico Macii, editor, *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 1405–1410. EDA Consortium San Jose, CA, USA / ACM DL, 2013.

[PS13b] Nir Piterman and Scott A. Smolka, editors. *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7795 of *Lecture Notes in Computer Science*. Springer, 2013.

[PT15] Luke Pierce and Spyros Tragoudas. Unreachable code identification for improved line coverage. In *Sixteenth International Symposium on Quality Electronic Design, ISQED 2015, Santa Clara, CA, USA, March 2-4, 2015*, pages 345–351. IEEE, 2015.

[Pud97] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.

[PW07] Andreas Podelski and Silke Wagner. Region stability proofs for hybrid systems. In Jean-Francois Raskin and P. S. Thiagarajan, editors, *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings*, volume 4763 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2007.

[Rak11] Astrid Rakow. *Slicing and reduction techniques for model checking Petri nets*. PhD thesis, Carl von Ossietzky University of Oldenburg, 2011.

[Rak12] Astrid Rakow. Safety slicing petri nets. In Serge Haddad and Lucia Pomello, editors, *Application and Theory of Petri Nets - 33rd International Conference, PETRI NETS 2012, Hamburg, Germany, June 25-29, 2012. Proceedings*, volume 7347 of *Lecture Notes in Computer Science*, pages 268–287. Springer, 2012.

[RIS13] Tony Ribeiro, Katsumi Inoue, and Chiaki Sakama. A BDD-based algorithm for learning from interpretation transition. In Gerson Zaverucha, Vítor Santos Costa, and Aline Paes, editors, *Inductive Logic Programming - 23rd International Conference, ILP 2013, Rio de Janeiro, Brazil, August 28-30, 2013, Revised Selected Papers*, volume 8812 of *Lecture Notes in Computer Science*, pages 47–63. Springer, 2013.

[RR08] C. R. Ramakrishnan and Jakob Rehof, editors. *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April*

*6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*. Springer, 2008.

[RS07] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Trans. Embedded Comput. Syst.*, 6(1), 2007.

[S+13] Natasha Sharygina et al., editors. *CAV 2013*, volume 8044 of *LNCS*. Springer, 2013.

[SAE96] SAE Int. ARP-4761. Technical report, Aerospace Recommended Practice, 1996.

[SB06] Carsten Sinz and Armin Biere. Extended resolution proofs for conjoining bdds. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *Computer Science - Theory and Applications, First International Computer Science Symposium in Russia, CSR 2006, St. Petersburg, Russia, June 8-12, 2006, Proceedings*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2006.

[Sch09] Renate A. Schmidt, editor. *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*. Springer, 2009.

[SCN13] Ricardo G. Sanfelice, David A. Copp, and Pablo Nanez. A toolbox for simulation of hybrid systems in matlab/simulink: hybrid equations (hyeq) toolbox. In Calin Belta and Franjo Ivancic, editors, *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*, pages 101–106. ACM, 2013.

[SD07] Jean Souyris and David Delmas. Experimental assessment of astrée on safety-critical avionics software. In Francesca Saglietti and Norbert Oster, editors, *Computer Safety, Reliability, and Security, 26th International Conference, SAFECOMP 2007, Nuremberg, Germany, September 18-21, 2007.*, volume 4680 of *Lecture Notes in Computer Science*, pages 479–490. Springer, 2007.

[Seg07] Marc Segelken. Abstraction and counterexample-guided construction of *omega* -automata for model checking of step-discrete linear hybrid models. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 433–448. Springer, 2007.

[Seg10] Mohamed Nassim Seghir. *Abstraction refinement techniques for software model checking*. PhD thesis, Albert-Ludwigs-Universität Freiburg im Breisgau, 2010.

[SFG14] Neil Sculthorpe, Nicolas Frisby, and Andy Gill. The kansas university

rewrite engine - A haskell-embedded strategic programming language with custom closed universes. *J. Funct. Program.*, 24(4):434–473, 2014.

[SFS12] Ondrej Sery, Grigory Fedyukovich, and Natasha Sharygina. Funfrog: Bounded model checking with interpolation-based function summarization. In Supratik Chakraborty and Madhavan Mukund, editors, *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings*, volume 7561 of *Lecture Notes in Computer Science*, pages 203–207. Springer, 2012.

[SGL+11] Daniel Schleicher, Stefan Grohe, Frank Leymann, Patrick Schneider, David Schumm, and Tamara Wolf. An approach to combine data-related and control-flow-related compliance rules. In Kwei-Jay Lin, Christian Huemer, M. Brian Blake, and Boualem Benatallah, editors, *2011 IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2011, Irvine, CA, USA, December 12-14, 2011*, pages 1–8. IEEE, 2011.

[SH13] Jendrik Seipp and Malte Helmert. Additive counterexample-guided cartesian abstraction refinement. In *AAAI (Late-Breaking Developments)*, volume WS-13-17 of *AAAI Workshops*. AAAI, 2013.

[SHBV03] Rehan Sadiq, Tahir Husain, Neil Bose, and Brian Veitch. Toxaphene distribution in the lake superior trout and associated sublethal ecological risk: a probabilistic approach. *Environmental Modelling and Software*, 18(5):439–449, 2003.

[SKB13] Karsten Scheibler, Stefan Kupferschmid, and Bernd Becker. Recent improvements in the SMT solver isat. In Christian Haubelt and Dirk Timmermann, editors, *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV), Warnemünde, Germany, March 12-14, 2013.*, pages 231–241. Institut für Angewandte Mikroelektronik und Datentechnik, Fakultät für Informatik und Elektrotechnik, Universität Rostock, 2013.

[Sla16] Looking up "slackness" in dictionary. http://http://www.dictionary.com/browse/slack, 2016. Accessed: 2016-09-26.

[SNM+16a] Karsten Scheibler, Felix Neubauer, Ahmed Mahdi, Martin Fränzle, Tino Teige, Tom Bienmüller, Detlef Fehrer, and Bernd Becker. Accurate icp-based floating-point reasoning. In Ruzica Piskac and Muralidhar Talupur, editors, *2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016*, pages 177–184. IEEE, 2016.

[SNM+16b] Karsten Scheibler, Felix Neubauer, Ahmed Mahdi, Martin Fränzle, Tino Teige, Tom Bienmüller, Detlef Fehrer, and Bernd Becker. Extending iSAT3 with ICP-contractors for bitwise integer operations. Reports of SFB/TR 14 AVACS 116, SFB/TR 14 AVACS, 2016. ISSN: 1860-9821, http://www.avacs.org.

[SPDA14] Christoph Scholl, Florian Pigorsch, Stefan Disch, and Ernst Althaus. Simple interpolants for linear arithmetic. In *Design, Automation & Test in Europe*

*Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6. IEEE, 2014.

[Spe10] Christian B. Spear. *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. Springer Publishing Company, Incorporated, 2nd edition, 2010.

[Spr00] Jeremy Sproston. Decidable model checking of probabilistic hybrid automata. In Mathai Joseph, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 6th International Symposium, FTRTFT 2000, Pune, India, September 20-22, 2000, Proceedings*, volume 1926 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2000.

[Spr01] J. Sproston. *Model Checking for Probabilistic Timed and Hybrid Systems*. PhD thesis, School of Computer Science, The University of Birmingham, 2001.

[SS99] João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.

[SS04] Christian Stangier and Thomas Sidle. Invariant checking combining forward and backward traversal. In Alan J. Hu and Andrew K. Martin, editors, *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings*, volume 3312 of *Lecture Notes in Computer Science*, pages 414–429. Springer, 2004.

[SS14] Mohamed Nassim Seghir and Peter Schrammel. Necessary and sufficient preconditions via eager abstraction. In Jacques Garrigue, editor, *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*, volume 8858 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2014.

[SSS00] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a sat-solver. In *Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings*, pages 108–125, 2000.

[Stu96] Gordon L. Stuber. *Principles of Mobile Communication*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1996.

[SVD+12] Alberto L. Sangiovanni-Vincentelli, Werner Damm, et al. Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. *EJC*, 18(3):217–238, 2012.

[SZ14] Fedor Shmarov and Paolo Zuliani. Probreach: Verified probabilistic delta-reachability for stochastic hybrid systems. *CoRR*, abs/1410.8060, 2014.

[Tar48] Alfred Tarski. A decision method for elementary algebra and geometry. RAND Corporation, Santa Monica, Calif., 1948.

[TCJ10] Tayssir Touili, Byron Cook, and Paul Jackson, editors. *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*. Springer, 2010.

[TD13] Cong Tian and Zhenhua Duan. Detecting spurious counterexamples efficiently in abstract model checking. In David Notkin, Betty H. C. Cheng, and Klaus Pohl, editors, *ICSE*, pages 202–211. IEEE / ACM, 2013.

[TEF11] Tino Teige, Andreas Eggers, and Martin Fränzle. Constraint-based analysis of concurrent probabilistic hybrid systems: An application to networked automation systems. *Nonlinear Analysis: Hybrid Systems*, 5(2):343–366, 2011.

[Tei12] Tino Teige. *Stochastic Satisfiability Modulo Theories: A Symbolic Technique for the Analysis of Probabilistic Hybrid Systems*. PhD thesis, FakultÂÍat II âĂŞ Informatik, Wirtschafts- und Rechtswissenschaften , Department fÂÍur Informatik, Germany, Oldenburg, Escherweg.2 26122, Augest 2012.

[TF10] Tino Teige and Martin Fränzle. Resolution for stochastic boolean satisfiability. In Christian G. Fermüller and Andrei Voronkov, editors, *LPAR (Yogyakarta)*, volume 6397 of *Lecture Notes in Computer Science*, pages 625–639. Springer, 2010.

[TF12a] Tino Teige and Martin Fränzle. Generalized craig interpolation. *Logical Methods in Computer Science*, 8(2), 2012.

[TF12b] Tino Teige and Martin Fränzle. Generalized craig interpolation. *Logical Methods in Computer Science*, 8(2), 2012.

[THR82] Philip C. Treleaven, Richard P. Hopkins, and Paul W. Rautenbach. Combining data flow and control flow computing. *Comput. J.*, 25(2):207–217, 1982.

[Tiw04] Ashish Tiwari. Termination of linear programs. In Rajeev Alur and Doron A. Peled, editors, *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, volume 3114 of *Lecture Notes in Computer Science*, pages 70–82. Springer, 2004.

[TRn09] Information Technology âĂŤ Programming Languages âĂŤ Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use. Technical Report ISO/IEC PDTR 24772, May 2009.

[Tse68] Grigori S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic*, Part II:115–125, 1968.

[Tse83] G. S. Tseitin. On the complexity of derivation in propositional calculus. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, Berlin, Heidelberg, 1983.

[UAcLR01] Algirdas Avizienis Ucla, Algirdas Avizienis, Jean claude Laprie, and Brian Randell. Fundamental concepts of dependability, 2001.

[vdBKvdB04] Jan van den Berg, Uzay Kaymak, and Willem-Max van den Bergh. Financial markets analysis by using a probabilistic fuzzy modelling approach. *Int. J. Approx. Reasoning*, 35(3):291–305, 2004.

[VG09] Yakir Vizel and Orna Grumberg. Interpolation-sequence based model checking. In *FMCAD*, pages 1–8. IEEE, 2009.

[VRN13] Yakir Vizel, Vadim Ryvchin, and Alexander Nadel. Efficient generation of small interpolants in CNF. In Sharygina et al. [S+13], pages 330–346.

[WBBL02] Bernd Westphal, Tom Bienmüller, Jürgen Bohn, and Rainer Lochmann. *SMI: syntax and semantics*. OFFIS e. V., Escherweg 2, D-26121, Oldenburg, 1 edition, May 2002.

[WKG07] Chao Wang, Hyondeuk Kim, and Aarti Gupta. Hybrid CEGAR: combining variable hiding and predicate abstraction. In Georges G. E. Gielen, editor, *2007 International Conference on Computer-Aided Design, ICCAD 2007, San Jose, CA, USA, November 5-8, 2007*, pages 310–317. IEEE Computer Society, 2007.

[WKM12] Georg Weissenbacher, Daniel Kroening, and Sharad Malik. Wolverine: Battling bugs with interpolants - (competition contribution). In Flanagan and König [FK12], pages 556–558.

[WRP+02] Chen Wang, Sudhakar M. Reddy, Irith Pomeranz, Xijiang Lin, and Janusz Rajski. Conflict driven techniques for improving deterministic test pattern generation. In Pileggi and Kuehlmann [PK02], pages 87–93.

[XJC09] Dana N. Xu, Simon L. Peyton Jones, and Koen Claessen. Static contract checking for haskell. In Zhong Shao and Benjamin C. Pierce, editors, *POPL*, pages 41–52. ACM, 2009.

[XZ10] Bican Xia and Zhihai Zhang. Termination of linear programs with nonlinear constraints. *J. Symb. Comput.*, 45(11):1234–1249, 2010.

[YM05] Greta Yorsh and Madanlal Musuvathi. A combination method for generating interpolants. In Robert Nieuwenhuis, editor, *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2005.

[ZM02] Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In Pileggi and Kuehlmann [PK02], pages 442–449.

[ZSR+10] Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Moritz Hahn. Safety verification for probabilistic hybrid systems. In Touili et al. [TCJ10], pages 196–211.

[ZTM11] Bin Zeng, Gang Tan, and Greg Morrisett. Combining control-flow integrity and static analysis for efficient and validated data sandboxing. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS*

*2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 29–40. ACM, 2011.

# Index

abstract interpretation, 59, 62
abstraction, 2, 18, 97
ACDCL, 61, 85
ACTL, 76
admissible, 28
assignments, 69
assumption, 17
assumption-commitment, 3, 23
automata, 18, 20
AVACS, 114
available expression analysis, 61
avionics engineering, 4

Büchi, 20
backwards reachability, 12
BDD, 2
Boolean skeleton, 78
branch-and-prune, 79

CBMC, 59
CDCL, 8, 60, 79
CEGAR, 8, 18, 59, 76, 97, 113
CFA, 69
chemical process control, 4
CI, 8, 59
circle, 86, 88
commitment, 17
components, 42
compositional verification, 17, 24, 40
computation path, 18, 22
computation paths, 40
concretizing, 77, 106, 115
configuration, 11
constraints, 69
contract, 3
control flow automaton, 71
control flow graph, 21, 71, 96, 107
counterexample, 77, 144
Craig interpolation, 60

data-flow analysis, 61
dead code, 7, 13, 59, 72
design, 1, 3, 4
design under verification, 4
discrete logic-based circuits, 3

edge supports a specification, 7
electrical engineering, 4
embedded system, 3
equisatisfiable, 79
error, 17
explicit-value analysis, 65

failure, 17
fault, 17, 50
Fischer's protocol, 49
flattening, 19
floating point, 107
floating-point, 60, 62, 111
floating-point arithmetics, 61
FMEA, 17
formal verification, 4
forward reachability, 11
Fourier-Motzkin elimination, 75

Gaussian elimination, 75
generalized Craig interpolation, 150
Gomory cuts, 64

hull-consistency, 78
hybrid automata, 20, 39
hybrid system, 4
hybrid systems, 62

ICP, 8, 60
implementation, 1
implication graph, 80
inductive interpolants, 99
infinities, 111
interpolant, 73
interpolants, 2, 60, 66
interprocedural, 66

invariant, 23

lazy abstraction, 99
live variable analysis:, 61
liveness, 12
local proof, 81
loop invariants, 66
LTL, 23, 56

Markov process, 4
matrix, 126
maximum probability of avoiding region, 150
mechanical engineering, 4
memory, 109, 110, 113, 119
Microcontrollers, 3
microprocessor, 3
model checking, 1, 4, 5, 61, 98, 146

NaN, 112
Nelson-Open, 64
networks of automata, 39
node, 43
non-chronological, 64, 82

observable behaviour, 23
operational semantics, 21, 71

partial model checking, 19
path, 71
PBMC, 123
Petri nets, 12, 19
predicate abstraction, 2
prefix condition, 68
preprocessing, 117, 119, 120
probabilistic hybrid automaton, 123
probabilistic region stability, 149
probabilistic safety property, 3
process, 50
program, 114
programs, 21, 59
PSPACE, 127

quantification, 123, 146

reachability, 5, 11, 59, 72
reachability property, 72
reaching definitions, 61
redirecting edges, 29
region stability, 149
removing edges, 32
repeaters, 43
run, 98
runs, 40

safe, 77
safe inductive invariant, 13, 69
safety, 1–3, 12, 68
safety case, 2, 3
safety property, 76
SAT, 75
semi-admissible, 28
sensor, 43
signed zeros, 111
simulation-based verification, 4
slackness, 81, 91
slackness of interpolants, 7
slicing, 19, 24
slicing models, 3
SMI, 8, 115
SMT, 59, 60, 75, 100
software model checking, 66
soundness, 131
specification, 1, 4, 7, 23
sphere, 86
state explosion problem, 2
state space explosion, 61, 76
subnormals, 111
support, 18, 21
supporting dependent transformation, 37
supporting edges, 34
symbolic model checking, 65, 73
system under design, 3

TCTL, 56
TDMA, 44
testing, 1
theorem proving, 4
timed automata, 19, 20, 39
trace, 71
transcendental functions, 59, 77, 107
transformation, 18
transformation functions, 28
troi, 90, 91

unreachable code, 13
uppaal, 42

valuation, 70
verification time, 109, 110, 113, 119
very busy expressions analysis, 61
virtual integration testing, 3

WFAS, 42
word automaton, 66