Working Formal Methods
2nd International Workshop
FROM 2018

Faculty of Computer Science

Alexandru Ioan Cuza University of Iaşi, Romania

Preliminary Proceedings

Laurenţiu Leuştean, Dorel Lucanu (Eds.)

June 18-20, 2018

Preface

This volume contains the papers presented at FROM 2018: Working Formal Methods Symposium held on June 18-20, 2018 in Iași.

Formal methods emphasize the use of mathematical techniques and rigour for developing software and hardware. They can be used to specify, verify, and analyse systems at any stage in their life cycle: requirements engineering, modeling, design, architecture, implementation, testing, maintenance and evolution. This assumes on one hand the development of adequate mathematical methods and frameworks and on the other hand the development of tools that help the user effectively apply these methods/frameworks.

FROM 2018 is organized by the Faculty of Computer Science at the Alexandru Ioan Cuza University in Iași, The Research Institute of the University of Bucharest (ICUB), and the Faculty of Mathematics and Computer Science at the University of Bucharest. FROM 2018 is the second event in a a yearly workshop series. The first edition was held in 2017 in Bucharest and it included sixteen invited talks, delivered by top researchers in the field, and seven contributed talks (see http://unibuc.ro/∼conference/from2017). Starting with the current edition, the goal is to increase the weight of the contributed talks.

The Working Formal Methods Symposium (FROM) aims to bring together researchers and practitioners who work on formal methods by contributing new theoretical results, methods, techniques, and frameworks, and/or make the formal methods to work by creating or using software tools that apply theoretical contributions.

The program of the symposium includes invited lectures, regular contributions and short contributions. The short contributions may describe work in progress or PhD progress/research reports. Submissions on the general topic of theoretical computer science, formal methods and applications are solicited.

Areas and formalisms of interest include:

- Category theory in computer science
- Distributed systems and concurrency
- Formal languages and automata theory
- Formal modeling, verification and testing
- Logic in computer science
- Logical frameworks
- Mathematical structures in computer science
- Models of computation
- Semantics of programming languages
- Type systems

Methods of interest include:

- Automated reasoning and model generation
- Automated induction
- Certified programs

- Data-flow and control-flow analysis
- Deductive verification
- Mechanized proofs
- Model checking
- Proof mining
- Symbolic computation
- Term rewriting

Applications of interest include:

- Computational logic
- Computer mathematics
- Knowledge representation, ontology reasoning, deductive databases
- Program analysis
- Verification and synthesis of software and hardware
- Uncertainty reasoning and soft computing

At FROM 2018 there were 15 submissions. Each submission was reviewed by at least two program committee members. The committee decided to accept 11 papers as regular submissions and 3 as short contributions. The program also includes 10 invited talks delivered by (in alphabetical order): Călin Belta, Radu Călinescu, Cătălin Dima, Dragoș Gavriluț, Radu Grigore, Cătălin Hrițcu, Mircea Marin, Grigore Roșu, Viorica Sofronie-Stokkermans, and Gheorghe Ștefănescu.

We would like to thank the members of the Program Committee and all the referees for their excellent work in the review and selection process. All of this was possible also thanks to the valuable and detailed reports provided by the sub-reviewers. We benefited from the invaluable assistance of the EasyChair system through all the phases of submission, evaluation, and production of the proceedings.

We gratefully acknowledge financial support from Amazon's Development Center in Iași and Continental Automotive in Iași.

June 11, 2018
Iași

Dorel Lucanu
Laurențiu Leuștean

Table of Contents

Program Committee

Marcello Bonsangue	Leiden University
Ștefan Ciobâcă	Alexandru Ioan Cuza University, Iasi
Gabriel Ciobanu	Romanian Academy, Institute of Computer Science, Iasi
Florin Crăciun	Department of Computer Science, Faculty of Mathematics and Computer Science, Babes-Bolyai University, Cluj, Romania
Florentin Ipate	University of Bucharest
Laurențiu Leuștean	Faculty of Mathematics and Computer Science, University of Bucharest
Dorel Lucanu	Alexandru Ioan Cuza University
Marius Minea	Politehnica University of Timisoara
Victor Mitrana	Universidad Politécnica de Madrid
Paulo Oliva	Queen Mary University of London
Ion Petre	Department of Computer Science, Åbo Akademi University, Finland
Andrei Popescu	Middlesex University London
Vlad Rusu	INRIA

Author Index

Formal Synthesis of Control Strategies for Dynamical Systems

Călin Belta[1,2,3]

[1] Department of Mechanical Engineering, Boston University
[2] Department of Electrical and Computer Engineering, Boston University
[3] Division of Systems Engineering, Boston University
cbelta@bu.edu

Abstract

In control theory, complex models of physical processes, such as systems of differential equations, are analyzed or controlled from simple specifications, such as stability and set invariance. In formal methods, rich specifications, such as formulae of temporal logics, are checked against simple models of software programs and digital circuits, such as finite transition systems. With the development and integration of cyber physical and safety critical systems, there is an increasing need for computational tools for verification and control of complex systems from rich, temporal logic specifications.

In this talk, I will discuss a set of approaches to formal synthesis of control strategies for dynamical systems from temporal logic specifications.

I will first show how automata games for finite systems can be extended to obtain conservative control strategies for low dimensional linear and multilinear dynamical systems. I will then present several methods to reduce conservativeness and improve the scalability of the control synthesis algorithms for more general classes of dynamics.

I will illustrate the usefulness of these approaches with examples from robotics and traffic control.

Observation-enhanced Stochastic Modelling

Radu Călinescu[1]

Department of Computer Science University of York, UK
radu.calinescu@york.ac.uk

Abstract

Stochastic modelling plays a key role in predicting the performance, dependability and other important properties of real-world systems. Supported by today's efficient probabilistic and statistical model checkers, the approach enables the analysis of alternative system designs and configurations, both during development (to select designs that comply with the system requirements) and at runtime (to select new configurations that restore this compliance after adverse events). However, the effectiveness of the approach depends on the quality of the models it operates with, and ensuring that stochastic models are accurate is a significant challenge. The talk will present recent advances in addressing this challenge by exploiting observations of the analysed system or its components, available for instance from logs, unit testing or monitoring.

Bisimulations for Logics for Strategies

Cătălin Dima[1]

Université Paris-Est Créteil, France
dima@univ-paris12.fr

Abstract

The notion of bisimulation is of central importance in the study of process or modal logics, as it frequently allows characterizing the expressive power of a logic within a class of Kripke models. Recently some extensions of modal logics like Alternating-time Temporal Logics or Strategy Logics have been proposed for dealing with strategic aspects in multi-agent models. We present some recent work and work in progress towards the characterization of the expressive power of these logics with the aid of appropriate notions of bisimulations. The talk will give a variety of definitions of bisimulations between multi-agent systems and an even larger variety of counterexamples of bisimilar systems that do not satisfy the same formulas in one of the strategy logics under consideration.

Fileless attacks – PowerShell-based techniques

Dragoș Gavriluț[1,2]

[1] Bitdefender Iași, România
[2] Alexandru Ioan Cuza University, Iași, Româania
gdt@info.uaic.ro

Abstract

Introduction of PowerShell utility in Windows was the beginning of a new series of cyber-attacks that leverage scripting languages and several obfuscation techniques with the fact that security products were at that time designed to skip system based tools from scanning. While fileless techniques were known for some time, they were never prevalent up to the moment PowerShell has become a part of all modern Windows OSs. These attack techniques are based on the usage of command line options available on PowerShell utility that allows for code execution without usage of a file that contain the actual code.

This talk will present how these types of attacks have evolved in terms of both features (from simple downloaders to OS loaders capabilities that can execute an entire binary in memory) and obfuscation techniques. The final part of this talk will present some ideas that can generally be used to identify such cases and are not limited to one scripting language.

Selective Monitoring

Radu Grigore[1]

School of Computing University of Kent, UK
R.Grigore@kent.ac.uk

Abstract

Runtime monitoring achieves a good trade-off in terms of developer effort versus bugs found. Most monitors, however, are used only during development but are turned off in shipped products, because the overhead of monitoring is significant.

We formalize this problem as finding an observation policy for a product between a Markov chain and a finite automaton. Our theoretical results suggest several lessons for a potential implementation of a selective monitor; for example, it is very important to record not only which method was called but also what the call site was. This work has some limitations, which would need to be addressed before we can close the loop back to practice. In particular, the formal model we use has no notion of data.

This is ongoing joint work with Stefan Kiefer.

Formally Secure Compilation

Cătălin Hriţcu[1]

INRIA Paris, France
catalin.hritcu@gmail.com

Abstract

Severe low-level vulnerabilities abound in today's computer systems, allowing cyber-attackers to remotely gain full control. The semantics of mainstream low-level languages like C provide no security against devastating vulnerabilities like buffer overflows and control-flow hijacking. Even for safer languages, establishing security with respect to the language's semantics does not prevent low-level attacks. All the abstraction and security guarantees of the source language are currently lost when interacting with low-level code, e.g., when using libraries.

Secure compilation is an emerging field that puts together advances in security, programming languages, compilers, verification, systems, and hardware architectures in order to devise secure compilation chains that eliminate many of today's low-level vulnerabilities. Secure compilation aims to protect high-level language abstractions in compiled code, even against adversarial low-level contexts, and to allow sound reasoning about security in the source language.

However, what does it mean that a compiler chain is secure? How does one define such secure compilation formally? And to what attacker model does it correspond? In this talk I will argue that a secure compilation chain should preserve some well-specified class of security properties of source programs even against adversarial low-level contexts. Particularly interesting classes include safety properties, hyperproperties (e.g. non-interference), and relational hyperproperties (e.g. observational equivalence).

Such strong secure compilation criteria complement compiler correctness and ensure that no adversarial low-level context can do more harm to a securely compiled program than a source-level context already could with respect to a *safe* source language semantics. We further extend these criteria to also provide end-to-end security guarantees even for software components written in *unsafe*, low-level languages with C-style undefined behavior. These extended criteria are the first to model dynamic compromise in a system of mutually distrustful components running with clearly specified privileges. Each component is protected from all the others—in particular, from components that have encountered undefined behavior and become compromised. Each component receives secure compilation guarantees up to the point when it becomes compromised, after which an attacker can take complete control over the component and use any of its privileges to attack the remaining uncompromised components.

To illustrate this model, we build a secure compilation chain for an unsafe language with buffers, procedures, and components. We compile it to a simple

RISC abstract machine with built-in compartmentalization and provide thorough proofs, many of them machine-checked in Coq, showing that the compiler satisfies our secure compilation criterion. Finally, we show that the protection guarantees offered by the compartmentalized abstract machine can be achieved at the machine-code level using either software fault isolation or a tag-based reference monitor.

Unification and Matching in Unranked Term Algebras With Regular Expression Sorts

Mircea Marin[1]

West University of Timişoara, România
mircea.marin@e-uvt.ro

Abstract

Unranked terms (or trees) are a natural way to model XML documents, multi-threaded recursive program configurations with an unbounded number of parallel processes, information in knowledge-based systems, data types and transformation patterns in programming languages, etc. Their study in sorted algebras is motivated, among others, by the desire to support typed computations, to work with more concise and natural representations of AI problems, and to speed up certain theorem proving methods.

A fundamental problem in applications pertaining to such representations is solving systems of equations. We analyse unification (REOSU) and matching (REOSM) in the general setting of order-sorted algebras where we permit regular expression sorts for variables and the domains of function symbols. We indicate how the problems studied by us generalise some known problems (e.g., order-sorted unification for ranked terms, sequence unification, and word unification with regular constraints), and present our main recent results: REOSU is infinitary and decidable; a procedure to construct a minimal set of REOS unifiers; finitary fragments of REOSU, including the corresponding matching problem (REOSM); NP-completeness of REOSM, and #P-completeness of the corresponding counting problem. Finally, we outline some promising directions of future work.

Formal Design, Implementation and Verification of Blockchain Languages and Virtual Machines

Grigore Roşu[1]

University of Illinois at Urbana-Champaign, US
grosu@illinois.edu

Abstract

Many of the recent cryptocurrency bugs and exploits are due to flaws or weaknesses of the underlying blockchain programming languages or virtual machines. The usual post-mortem approach to formal language semantics and verification, where the language is firstly implemented and used in production for many years before a need for formal semantics and verification tools naturally arises, simply does not work anymore. New blockchain languages or virtual machines are proposed at an alarming rate, followed by new versions of them every few weeks, together with programs (or smart contracts) in these languages that are responsible for financial transactions of potentially significant value. Formal analysis and verification tools are therefore needed immediately for such languages and virtual machines.

We present recent academic and commercial results in developing blockchain languages and virtual machines that come directly equipped with formal analysis and verification tools. The main idea is to generate all these automatically, correct-by-construction from a formal specification. We demonstrate the feasibility of the proposed approach by applying it to two blockchains, Ethereum and Cardano. We also discuss the mathematical foundations underlying our formal analysis and verification tools, specifically matching logic and the K framework.

On Symbol Elimination in Theory Extensions and Applications

Viorica Sofronie-Stokkermans

University Koblenz-Landau, Koblenz, Germany

Abstract. We present a synthesis of our previous work on symbol elimination in extensions of a theory T_0 with additional function symbols whose properties are axiomatised using a set of clauses: We analyze situations in which we can perform such tasks in a hierarchical way, relying on existing mechanisms for symbol elimination in T_0. This is for instance possible if the base theory allows quantifier elimination. We analyze possibilities of extending such methods to situations in which the base theory does not allow quantifier elimination but has a model completion which does. We then briefly discuss the way the method can be used in the analysis of parametric systems (for synthesizing constraints on parameters and also for invariant synthesis).

1 Introduction

Many problems in computer science can be reduced to checking satisfiability of ground formulae w.r.t. a theory. This theory can be a standard theory – for instance linear arithmetic – or an extension or combination of theories. More interesting however is to go beyond yes/no answers, i.e. to consider parametric systems and infer constraints on parameters (which can be values or functions) which guarantee that certain properties are met (e.g. guarantee the unsatisfiability of ground clauses in suitable theory extensions).

In [14, 15] – in a context specially tailored for the parametric verification of safety properties in increasingly more complex systems – we showed that such constraints could be generated in extensions of a theory allowing quantifier elimination. In [16] we proposed a symbol elimination method in theory extensions and analyzed its properties. We also discussed possibilities of applying such methods to extensions of theories which do not allow quantifier elimination provided that they have a model completion which does, and indicated how to use such methods for interpolation. In ongoing work [10] we are using such methods for invariant synthesis. In this paper we give an overview of the results; for details and proofs we refer to [14, 15, 16] and [10].

2 Preliminaries

We assume known standard definitions from first-order logic such as Π-structures, models, homomorphisms, logical entailment, satisfiability, unsatisfiability. We

consider signatures of the form $\Pi = (\Sigma, \mathsf{Pred})$, where Σ is a family of function symbols and Pred a family of predicate symbols. In this paper, (logical) theories are simply sets of sentences. We denote "falsum" with $\bot$. If F and G are formulae we write $F \models G$ (resp. $F \models_{\mathcal{T}} G$ – also written as $\mathcal{T} \cup F \models G$) to express the fact that every model of F (resp. every model of F which is also a model of $\mathcal{T}$) is a model of G. $F \models \bot$ means that F is unsatisfiable; $F \models_{\mathcal{T}} \bot$ means that there is no model of $\mathcal{T}$ in which F is true. If there is a model of $\mathcal{T}$ which is also a model of F we say that F is $\mathcal{T}$-consistent.

If $\mathcal{T}$ is a theory over a signature $\Pi = (\Sigma, \mathsf{Pred})$ we denote by $\mathcal{T}_\forall$ (the universal theory of $\mathcal{T}$) the set of all universal sentences which are logical consequences of $\mathcal{T}$. For Π-structures $\mathcal{A}$ and $\mathcal{B}$, $\varphi : \mathcal{A} \to \mathcal{B}$ is an embedding if and only if it is an injective homomorphism and has the property that for every $P \in \mathsf{Pred}$ with arity n and all $(a_1, \ldots, a_n) \in A^n$, $(a_1, \ldots, a_n) \in P_{\mathcal{A}}$ iff $(\varphi(a_1), \ldots, \varphi(a_n)) \in P_{\mathcal{B}}$. In particular, an embedding preserves the truth of all literals. An elementary embedding between two Π-structures is an embedding that preserves the truth of all first-order formulae over Π. Two Π-structures are elementarily equivalent if they satisfy the same first-order formulae over Π.

Let $\mathcal{A} = (A, \{f_{\mathcal{A}}\}_{f \in \Sigma}, \{P_{\mathcal{A}}\}_{P \in \mathsf{Pred}})$ be a Π-structure. In what follows, we will sometimes denote the universe A of the structure $\mathcal{A}$ by $|\mathcal{A}|$. The *diagram* $\Delta(\mathcal{A})$ of $\mathcal{A}$ is the set of all literals true in the extension $\mathcal{A}^A$ of $\mathcal{A}$ where we have an additional constant for each element of A (which we here denote with the same symbol) with the natural expanded interpretation mapping the constant a to the element a of $|\mathcal{A}|$ (this is a set of sentences over the signature Π^A obtained by expanding Π with a fresh constant a for every element a from $|\mathcal{A}|$). Note that if $\mathcal{A}$ is a Π-structure and $\mathcal{T}$ a theory and $\Delta(\mathcal{A})$ is $\mathcal{T}$-consistent then there exists a Π-structure $\mathcal{B}$ which is a model of $\mathcal{T}$ and in which $\mathcal{A}$ embeds.

A theory $\mathcal{T}$ over signature Π *allows quantifier elimination* if for every formula ϕ over Π there exists a quantifier-free formula ϕ^* over Π which is equivalent to ϕ modulo $\mathcal{T}$.

A *model complete* theory has the property that all embeddings between its models are elementary. Every theory which allows quantifier elimination (QE) is model complete (cf. [6], Theorem 7.3.1).

Example 1. Presburger arithmetic with congruence mod. n, rational linear arithmetic, the theories of real closed fields and of algebraically closed fields, the theory of finite fields and the theory of acyclic lists in the signature $\{\mathrm{car}, \mathrm{cdr}, \mathrm{cons}\}$ ([9, 4]) allow QE, hence are model complete.

A theory $\mathcal{T}^*$ is called a *model companion* of $\mathcal{T}$ if (i) $\mathcal{T}$ and $\mathcal{T}^*$ are co-theories (i.e. every model of $\mathcal{T}$ can be extended to a model of $\mathcal{T}^*$ and vice versa), (ii) $\mathcal{T}^*$ is model complete. $\mathcal{T}^*$ is a *model completion* of $\mathcal{T}$ if it is a model companion of $\mathcal{T}$ with the additional property (iii) for every model $\mathcal{A}$ of $\mathcal{T}$, $\mathcal{T}^* \cup \Delta(\mathcal{A})$ is a complete theory (where $\Delta(\mathcal{A})$ is the diagram of $\mathcal{A}$). Several examples of model completions are mentioned later, in Example 2.

3 Ground Interpolation and Quantifier Elimination

A Π-theory $\mathcal{T}$ has interpolation if, for all Π-formulae ϕ and ψ if $\phi \models_{\mathcal{T}} \psi$ then there exists a formula I containing only symbols common to ϕ and ψ such that $\phi \models_{\mathcal{T}} I$ and $I \models_{\mathcal{T}} \psi$. First order logic has interpolation [3]. It is often important to identify theories for which ground formulae have ground interpolants.

Definition 2. *A theory $\mathcal{T}$ has the* ground interpolation property *(for short: $\mathcal{T}$ has* ground interpolation*) if for every pair of ground formulae $A(\overline{c}, \overline{a})$ (containing constants $\overline{c}, \overline{a}$) and $B(\overline{c}, \overline{b})$ (containing constants $\overline{c}, \overline{b}$), if $A(\overline{c}, \overline{a}) \wedge B(\overline{c}, \overline{b}) \models_{\mathcal{T}} \bot$ then there exists a ground formula $I(\overline{c})$, containing only the constants $\overline{c}$ occurring both in A and B, such that $A(\overline{c}, \overline{a}) \models_{\mathcal{T}} I(\overline{c})$ and $B(\overline{c}, \overline{b}) \wedge I(\overline{c}) \models_{\mathcal{T}} \bot$.*

Let $\mathcal{T}$ be a theory in a signature Σ and Σ' a signature disjoint from Σ. We denote by $\mathcal{T} \cup \mathsf{UIF}_{\Sigma'}$ the extension of $\mathcal{T}$ with uninterpreted symbols in Σ'.

Definition 3 ([5]). *We say that a theory $\mathcal{T}$ in a signature Σ has the* general ground interpolation property *(or, shorter, that $\mathcal{T}$ has* general ground interpolation*) if for every signature Σ' disjoint from Σ and every pair of ground $\Sigma \cup \Sigma'$-formulae A and B, if $A \wedge B \models_{\mathcal{T} \cup \mathsf{UIF}_{\Sigma'}} \bot$ then there exists a ground formula I, such that (i) all predicate, constants and function symbols from Σ' occurring in I also occur in A and B; (ii) $A \models_{\mathcal{T} \cup \mathsf{UIF}_{\Sigma'}} I$ and (iii) $B \wedge I \models_{\mathcal{T} \cup \mathsf{UIF}_{\Sigma'}} \bot$.*

Definition 4 ([5]). *A theory $\mathcal{T}$ has the* sub-amalgamation property *iff whenever we are given models M_1 and M_2 of $\mathcal{T}$ with a common substructure A, there exists a further model M of $\mathcal{T}$ endowed with embeddings $\mu_i : M_i \to M$, $i = 1, 2$ whose restrictions to A coincide. $\mathcal{T}$ has the* strong sub-amalgamation property *if the preceding embeddings μ_1, μ_2 and the preceding model M can be chosen so as to satisfy the following additional condition: if for some m_1, m_2 we have $\mu_1(m_1) = \mu_2(m_2)$, then there exists an element $a \in A$ such that $m_1 = m_2 = a$.*

Definition 5 ([5]). *A theory $\mathcal{T}$ is equality interpolating iff it has the ground interpolation property and has the property that for all tuples $x = x_1, \ldots, x_n$, $y^1 = y^1_1, \ldots, y^1_{n_1}$, $z^1 = z^1_1, \ldots, z^1_{m_1}$, $y^2 = y^2_1, \ldots, y^2_{n_2}$, $z^2 = z^2_1, \ldots, z^2_{m_2}$ of constants, and for every pair of ground formulae $A(x, z^1, y^1)$ and $B(x, z^2, y^2)$ such that $A(x, z^1, y^1) \wedge B(x, z^2, y^2) \models_{\mathcal{T}} \bigvee_{i=1}^{n_1} \bigvee_{j=1}^{n_2} y^1_i = y^2_j$ there exists a tuple of terms containing only the constants in x, $v(x) = v_1, \ldots, v_k$ such that*

$$A(x, z^1, y^1) \wedge B(x, z^2, y^2) \models_{\mathcal{T}} \bigvee_{i=1}^{n_1} \bigvee_{u=1}^{k} y^1_i = v_k \vee \bigvee_{j=1}^{n_2} \bigvee_{u=1}^{k} v_k = y^2_j$$

The following connections between the (sub-)amalgamation property, ground interpolation and equality interpolation are known:

Theorem 6 ([1, 5, 2]). *The following hold for any theory $\mathcal{T}$:*

(1) If $\mathcal{T}$ is universal and has the amalgamation property then $\mathcal{T}$ has ground interpolation [1].

(2) $\mathcal{T}$ has the sub-amalgamation property iff it has ground interpolation [5].

(3) $\mathcal{T}$ is strongly sub-amalgamable iff it has general ground interpolation [5].

(4) If $\mathcal{T}$ has ground interpolation, then $\mathcal{T}$ is strongly sub-amalgamable iff it is equality interpolating [5].

(5) If $\mathcal{T}$ is universal and allows QE, $\mathcal{T}$ is equality interpolating [5].

(6) If $\mathcal{T}^*$ is a model companion of $\mathcal{T}$ then (i) $\mathcal{T}^*$ is a model completion of $\mathcal{T}$ iff (ii) $\mathcal{T}$ has the amalgamation property. If, additionally, $\mathcal{T}$ has universal axiomatization, either of the equivalent conditions (i), (ii) above is equivalent to the fact that $\mathcal{T}^*$ allows quantifier elimination [2].

Corollary 7 ([16]). *If $\mathcal{T}$ is a universal theory which allows quantifier elimination then $\mathcal{T}$ has* general ground interpolation*.*

3.1 Model Companions and Ground Interpolation

In what follows we present links between ground interpolation resp. quantifier elimination in a theory and in its model companions (if they exist) which were established in [16].

Theorem 8 ([16]). *Let $\mathcal{T}$ be a theory with signature Π and $A(c_1, \ldots, c_n, d_1, \ldots, d_m)$ a ground formula over an extension Π^C of Π with additional constants $c_1, \ldots, c_n$, $d_1, \ldots, d_m$. If $\mathcal{T}$ has quantifier elimination then there exists a ground formula $\Gamma(c_1, \ldots, c_n)$ containing only constants $c_1, \ldots, c_n$, which is satisfiable w.r.t. $\mathcal{T}$ iff $A(c_1, \ldots, c_n, d_1, \ldots, d_m)$ is satisfiable w.r.t. $\mathcal{T}$.*

Lemma 9 ([16]). *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two co-theories with signature Π, and $A(c_1, \ldots, c_n)$ be a ground formula over an extension Π^C of Π with new constants $c_1, \ldots, c_n$. Then A is satisfiable w.r.t. $\mathcal{T}_1$ iff it is satisfiable w.r.t. $\mathcal{T}_2$.*

Theorem 10 ([16]). *Let $\mathcal{T}$ be a theory and $\mathcal{T}^*$ a model companion of $\mathcal{T}$.*

(1) If $\mathcal{T}$ is universal and has ground interpolation, then $\mathcal{T}^*$ allows QE.

(2) If $\mathcal{T}^*$ has ground interpolation then so does $\mathcal{T}$; the ground interpolants computed w.r.t. $\mathcal{T}^*$ are also interpolants w.r.t. $\mathcal{T}$.

(3) A universal theory $\mathcal{T}$ has ground interpolation iff $\mathcal{T}^*$ has ground interpolation.

(4) If $\mathcal{T}^*$ allows quantifier elimination then $\mathcal{T}$ has ground interpolation.

Example 11. The following theories have ground interpolation:

(1) The pure theory of equality (its model completion is the theory of an infinite set, which allows QE).

(2) The theory of total orderings (its model completion is the theory of dense total orders without endpoints, which allows QE [4]).

(3) The theory of Boolean algebras (its model completion is the theory of atomless Boolean algebras, which allows QE [2]).

(4) The theory of fields (its model completion is the theory of algebraically closed fields, which allows QE [2, 6]).

4 Symbol Elimination in Theory Extensions

In [16] we showed that in theory extensions $\mathcal{T}_0 \subseteq \mathcal{T} = \mathcal{T}_0 \cup \mathcal{K}$ for which $\mathcal{T}_0$ (resp. its model completion $\mathcal{T}_0^*$) allows quantifier elimination, for every ground formula G containing function symbols considered to be "parameters" we can generate a (universal) constraint Γ on the parameters of G such that $\mathcal{T} \cup \Gamma \cup G \models \bot$.

Let $\Pi_0 = (\Sigma_0, \mathsf{Pred})$. Let $\mathcal{T}_0$ be a Π_0-theory and Σ_P be a set of parameters (function and constant symbols). Let Σ be a signature such that $\Sigma \cap (\Sigma_0 \cup \Sigma_P) = \emptyset$, containing functions not in $(\Sigma_0 \cup \Sigma_P)$. Let $\mathcal{K}$ be a set of clauses in the signature $\Pi_0 \cup \Sigma_P \cup \Sigma$ in which all variables occur also below functions in $\Sigma_1 = \Sigma_P \cup \Sigma$. Let G be a finite set of ground clauses, and T a finite set of ground terms over the signature $\Pi_0 \cup \Sigma_P \cup \Sigma \cup C$, where C is a set of additional constants.
We construct a universal formula $\forall y_1 \dots y_n \Gamma_T(y_1, \dots, y_n)$ over the signature $\Pi_0 \cup \Sigma_P$ by following Steps 1–5 below (cf. also [16]):

Step 1: Let $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ be the set of Π_0^C clauses obtained from $\mathcal{K}[T] \cup G$ by introducing, in a bottom-up manner, new constants $c_t \in C$ for subterms $t{=}f(c_1, \dots, c_n)$ where $f{\in}\Sigma_1$ and c_i are constants, together with definitions $c_t{=}f(c_1, \dots, c_n)$ included in a set Def, and then replacing Def with the following set, corresponding to the instances of the congruence axioms:
$\mathsf{Con}_0{=}\{\bigwedge_{i=1}^{n} c_i{\approx}d_i \rightarrow c{\approx}d \,|\, \begin{smallmatrix} f(c_1, \dots, c_n){\approx}c{\in}\mathsf{Def} \\ f(d_1, \dots, d_n){\approx}d{\in}\mathsf{Def} \end{smallmatrix}\}$.

Step 2: Let $G_1 = \mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$. Among the constants in G_1, we identify (i) the constants $\overline{c}_f$, $f \in \Sigma_P$, where either $c_f = f \in \Sigma_P$ is a constant parameter, or is introduced by a definition $c_f := f(c_1, \dots, c_k)$ in the hierarchical reasoning method, and (ii) all constants $\overline{c}_p$ occurring as arguments of functions in Σ_P in such definitions. Let $\overline{c}$ be the remaining variables. We replace the constants in $\overline{c}$ with existentially quantified variables $\overline{x}$ in G_1, i.e. replace $G_1(\overline{c}_p, \overline{c}_f, \overline{c})$ with $G_1(\overline{c}_p, \overline{c}_f, \overline{x})$, and consider the formula $\exists \overline{x} G_1(\overline{c}_p, \overline{c}_f, \overline{x})$.

Step 3: Using the method for quantifier elimination in $\mathcal{T}_0$ we can construct a formula $\Gamma_1(\overline{c}_p, \overline{c}_f)$ equivalent to $\exists \overline{x} G_1(\overline{c}_p, \overline{c}_f, \overline{x})$ w.r.t. $\mathcal{T}_0$.

Step 4: Let $\Gamma_2(\overline{c}_p)$ be the formula obtained by replacing back in $\Gamma_1(\overline{c}_p, \overline{c}_f)$ the constants c_f introduced by definitions $c_f := f(c_1, \dots, c_k)$ with the terms $f(c_1, \dots, c_k)$. We replace $\overline{c}_p$ with existentially quantified variables $\overline{y}$ and obtain the formula $\exists \overline{y} \Gamma_2(\overline{y})$.

Step 5: Let $\forall \overline{y} \Gamma_T(\overline{y})$ be $\neg(\exists \overline{y} \Gamma_2(\overline{y}))$, i.e. $\forall \overline{y} \neg \Gamma_2(\overline{y})$.

A similar approach is used in [14] for generating constraints on parameters which guarantee safety of parametric systems. We show that $\forall \overline{y} \Gamma_T(\overline{y})$ guarantees unsatisfiability of G and further study the properties of these formulae.

We first analyze the case in which $\mathcal{T}_0$ allows quantifier elimination.

Theorem 12 ([16]). *Assume that $\mathcal{T}_0$ allows quantifier elimination. For every finite set of ground clauses G, and every finite set T of terms over the signature $\Pi_0 \cup \Sigma \cup \Sigma_P \cup C$ with $\mathsf{est}(G) \subseteq T$, Steps 1–5 yield a universally quantified $\Pi_0 \cup \Sigma_P$-formula $\forall \overline{x} \Gamma_T(\overline{x})$ with the following properties:*

(1) For every structure $\mathcal{A}$ with signature $\Pi_0 \cup \Sigma \cup \Sigma_P \cup C$ which is a model of $\mathcal{T}_0 \cup \mathcal{K}$, if $\mathcal{A} \models \forall \overline{y} \Gamma_T(\overline{y})$ then $\mathcal{A} \models \neg G$.

(2) $\mathcal{T}_0 \cup \forall \overline{y} \Gamma_T(\overline{y}) \cup \mathcal{K} \cup G$ is unsatisfiable.

We now analyze the case in which $\mathcal{T}_0$ does not necessarily allow quantifier elimination, but has a model completion which allows quantifier elimination.

Theorem 13 ([16]). *Let $\mathcal{T}_0$ be a theory having a model completion $\mathcal{T}_0^*$ with $\mathcal{T}_0 \subseteq \mathcal{T}_0^*$. Let $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{K}$ be an extension of $\mathcal{T}_0$ with new function symbols $\Sigma_1 = \Sigma_P \cup \Sigma$ whose properties are axiomatized by a set of clauses $\mathcal{K}$ in which all variables occur also below extension functions in Σ_1. Assume that (i) every model of $\mathcal{T}_0 \cup \mathcal{K}$ embeds into a model of $\mathcal{T}_0^* \cup \mathcal{K}$, and (ii) $\mathcal{T}_0^*$ allows quantifier elimination.*

Then, for every finite set of ground clauses G, and every finite set T of terms over the signature $\Pi^C = \Pi_0 \cup \Sigma \cup \Sigma_P \cup C$ with $\mathsf{est}(G) \subseteq T$ we can construct a universally quantified $\Pi_0 \cup \Sigma_P$-formula $\forall \overline{x} \Gamma_T(\overline{x})$ such that:

(1) For every structure $\mathcal{A}$ with signature $\Pi_0 \cup \Sigma \cup \Sigma_P \cup C$ which is a model of $\mathcal{T}_0 \cup \mathcal{K}$, if $\mathcal{A} \models \forall \overline{x} \Gamma_T(\overline{x})$ then $\mathcal{A} \models \neg G$.

(2) $\mathcal{T}_0 \cup \forall y \Gamma_T(y) \cup \mathcal{K} \cup G$ is unsatisfiable.

4.1 Symbol Elimination in Local Theory Extensions

We identify a class of theory extensions for which we can find a finite set T of ground terms for which the algorithm presented in the previous section provides the *weakest* constraint Γ on parameters such that $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K} \cup G$ is unsatisfiable.

As before, we assume that $\mathcal{T}_0$ is a theory with signature $\Pi_0 = (\Sigma_0, \mathsf{Pred})$, and that we consider extensions $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$, where $\mathcal{K}$ is a set of clauses over signature $\Pi = (\Sigma_0 \cup \Sigma_1, \mathsf{Pred})$, where $\Sigma_1 = \Sigma_P \cup \Sigma$ is a set of extension functions (disjoint from Σ_0) among which we distinguish a set of *parametric* function symbols Σ_P.

Let G be a ground clause over $\Pi_0 \cup \Sigma_P \cup \Sigma \cup C$. Let $\mathsf{est}(\mathcal{K}, G)$ be the set of all ground *extension terms* (i.e. terms starting with an extension function in $\Sigma_P \cup \Sigma$) occurring in $\mathcal{K}$ or in G.

Definition 14 ([12]). *Let $\mathcal{T}_0 \cup \mathcal{K}$ be an extension of $\mathcal{T}_0$ with clauses in $\mathcal{K}$. We say that the extension is local if it satisfies the following condition:*

(Loc_f) *For every finite set G of ground clauses in Π^C it holds that $\mathcal{T}_0 \cup \mathcal{K} \cup G \models \bot$ if and only if $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ is unsatisfiable.*

where $\mathcal{K}[G] = \mathcal{K}[\mathsf{est}(\mathcal{K}, G)]$.

In [12, 7] we defined the following completability condition for functions with a finite domain of definition for a theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$:

(Comp_f) Every partial model $\mathcal{A}$ of $\mathcal{T}_0 \cup \mathcal{K}$ in which the Σ_0-functions are total and the (possibly partial) extension functions have a finite domain of definition embeds into a total model of $\mathcal{T}_0 \cup \mathcal{K}$ with *the same support.*

We say that a Π-clause D *is flat* when all symbols below a Σ_1-function symbol in D are variables. D is *linear* if whenever a variable occurs in two terms of D starting with Σ_1-functions, the terms are equal, and no term contains two occurrences of a variable. A *ground clause D is flat* if all symbols below a Σ_1-function in D are constants. A *ground clause D is linear* if whenever a constant occurs in two terms in D whose root symbol is in Σ_1, the two terms are identical, and if no term which starts with a Σ_1-function contains two occurrences of the same constant.

Theorem 15 ([12, 7]). *Let $\mathcal{T}_0$ be a first-order theory and $\mathcal{K}$ a set of universally closed flat clauses in the signature Π. If all clauses in $\mathcal{K}$ are linear and condition (Comp_f) holds then the extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is local.*

In what follows, $\forall \overline{y} \Gamma_G(\overline{y})$ denotes the formula obtained when $T = \mathsf{est}(\mathcal{K}, G)$.

Theorem 16 ([16]). *If the extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ satisfies condition (Comp_f) and $\mathcal{K}$ is flat and linear then $\forall y \Gamma_G(y)$ is entailed by every conjunction Γ of clauses such that $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K} \cup G$ is unsatisfiable (i.e. it is the weakest such constraint).*

In [16] we showed that this result does not hold if $\mathcal{T}_0$ does not allow quantifier elimination but its model completion does: Under the assumptions of Theorem 5 we cannot guarantee that $\forall y \Gamma_G(y)$ is the weakest set of constraints in the set of all Γ with $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K} \cup G \models \bot$.

5 Applications

We briefly present two areas in which these results can be used: interpolation and verification/synthesis.

5.1 Interpolation

Criteria linking hierarchical ground interpolation to "separability" and to an amalgamability property for partial algebras were given in [11].

Definition 17 ([11]). *An amalgamation closure for a theory extension $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{K}$ is a function W associating with finite sets of ground terms T_A and T_B, a finite set $W(T_A, T_B)$ of ground terms such that*

(1) all ground subterms in $\mathcal{K}$ and T_A are in $W(T_A, T_B)$;

(2) W is monotone, i.e., for all $T_A \subseteq T'_A$, $T_B \subseteq T'_B$, $W(T_A, T_B) \subseteq W(T'_A, T'_B)$;

(3) W is a closure, i.e., $W(W(T_A, T_B), W(T_B, T_A)) \subseteq W(T_A, T_B)$;

(4) W is compatible with any map h between constants satisfying $h(c_1) \neq h(c_2)$, for all constants $c_1 \in \mathsf{st}(T_A), c_2 \in \mathsf{st}(T_B)$ that are not shared between T_A and T_B, i.e., for any such h we require $W(h(T_A), h(T_B)) = h(W(T_A, T_B))$; and

(5) $W(T_A, T_B)$ only contains T_A-pure terms (i.e. terms containing only constants in C which occur in T_A).

For sets of ground clauses A, B we write $W(A,B)$ for $W(\mathsf{st}(A),\mathsf{st}(B))$. In what follows, when using a function W we always refer to an amalgamation closure.

Definition 18 ([11]). *A theory extension $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{K}$ is W-separable if for all sets of ground clauses A and B,*

$$T_0 \cup \mathcal{K} \cup A \cup B \models \bot \text{ iff } T_0 \cup \mathcal{K}[W(A,B)] \cup A \cup \mathcal{K}[W(B,A)] \cup B \models \bot .$$

Theorem 19 ([16], [11]). *Let W be an amalgamation closure operator. Assume that the theory $\mathcal{T}_0$ has general ground interpolation, and there is a method for effectively computing general ground interpolants w.r.t. $\mathcal{T}_0$. Let $\mathcal{T}_0 \cup \mathcal{K}$ be a W-separable extension of $\mathcal{T}_0$ with a set of clauses $\mathcal{K}$ in which every variable occurs below an extension function. Let A and B be two ground $\Sigma_0 \cup \Sigma$-formulae. If $A \wedge B \models_{\mathcal{T}_0 \cup \mathcal{K}} \bot$ then we can effectively compute a ground interpolant for A and B, by computing an interpolant of $\mathcal{K}[W(A,B)] \cup A$ and $\mathcal{K}[W(B,A)] \cup B$.*

It is sometimes difficult to check directly whether the theory $\mathcal{T}_0$ has ground interpolation. If $\mathcal{T}_0$ has a model completion with good properties, this becomes easier to check. In this case, we can use quantifier elimination in the model completion to compute the interpolant.

Theorem 20 ([16]). *Let W be an amalgamation closure operator, and let $\mathcal{T}_0 \cup \mathcal{K}$ be a W-separable extension of $\mathcal{T}_0$ with a set of clauses $\mathcal{K}$ in which every variable occurs below an extension function. Assume that $\mathcal{T}_0$ has a model companion $\mathcal{T}_0^*$ such that $\mathcal{T}_0 \subseteq \mathcal{T}_0^*$ and $\mathcal{T}_0^*$ has general ground interpolation. (This can happen for instance when $\mathcal{T}_0^*$ allows quantifier elimination and is equality interpolating.) Then $\mathcal{T}_0 \cup \mathcal{K}$ has ground interpolation.*

5.2 Verification/Synthesis

For modeling (parametric) systems we use transition constraint systems $T = (V, \Sigma, \mathsf{Init}, \mathsf{Update})$ which specify: the variables (V) and function symbols (Σ) whose values change over time; a formula Init specifying the properties of initial states; a formula Update with variables in $V \cup V'$ and function symbols in $\Sigma \cup \Sigma'$ (where V' and Σ' are copies of V resp. Σ, denoting the variables resp. functions after the transition) which specifies the relationship between the values of variables v and function symbols f before a transition and their values (v', f') after the transition. Such descriptions can be obtained from system specifications. With every specification of a system S, a *background theory* $\mathcal{T}_S$ – describing the data types used in the specification and their properties – is associated.

We can check in two steps whether a formula Ψ is an inductive invariant of a transition constraint system $T{=}(V, \Sigma, \mathsf{Init}, \mathsf{Update})$:

(1) prove that $\mathcal{T}_S, \mathsf{Init} \models \Psi$;
(2) prove that $\mathcal{T}_S, \Psi, \mathsf{Update} \models \Psi'$, where Ψ' results from Ψ by replacing each $v \in V$ by v' and each $f \in \Sigma$ by f'.

If Ψ is a universal formula then these verification tasks can be reduced to the problem of checking satisfiability of a ground formula (corresponding to $\neg\Psi$) w.r.t. a theory $\mathcal{T}$. In [8, 14, 15] we identify situations when the theory $\mathcal{T}$ can be expressed using a chain of extensions, typically including:

$$\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \Psi \subseteq \mathcal{T} = \mathcal{T}_0 \cup \Psi \cup \mathsf{Update}$$

with the property that checking satisfiability of ground formulae w.r.t. $\mathcal{T}$ can be reduced to checking satisfiability w.r.t. $\mathcal{T}_1$ and ultimately to checking satisfiability w.r.t. $\mathcal{T}_0$. This is the case for instance when the theory extensions in the chain above are *local*.

If Ψ is not an inductive invariant, we consider two orthogonal problems:

(1) Determine constraints on parameters which guarantee that Ψ is an invariant.
(2) Determine a formula I such that $\mathcal{T}_S \models I \rightarrow \Psi$ and I is an inductive invariant.

In [14, 15] we showed that if the theory $\mathcal{T}$ can be expressed using a chain of local extensions as explained above, then locality and the reduction to a satisfiability problem w.r.t. the base theory can also be used for obtaining constraints on parameters which guarantee that Ψ is an invariant.

In [10] we propose a method for property-directed invariant generation which uses the symbol elimination described in Section 4, and analyze its properties.

References

[1] Bacsich, P.D. Amalgamation properties and interpolation theorem for equational theories. *Algebra Universalis*, 5:45–55, 1975.
[2] Chang, C.C., Keisler, J.J. Model Theory. North-Holland, Amsterdam (1990)
[3] Craig, W. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *J. Symb. Log.*, 22(3):250–268, 1957.
[4] Ghilardi, S. Model-theoretic methods in combined constraint satisfiability. Journal of Automated Reasoning 33(3-4), 221–249 (2004)
[5] Bruttomesso, R., Ghilardi, S. and Ranise, S. Quantifier-free interpolation in combinations of equality interpolating theories. ACM Trans. Comput. Log. 15(1):5 (2014)
[6] Hodges, W. A Shorter Model Theory. Cambridge University Press (1997)
[7] Ihlemann, C., Sofronie-Stokkermans, V. On Hierarchical Reasoning in Combinations of Theories. In: Giesl, J. and Hähnle, R.(eds) IJCAR 2010, LNAI, vol. 6173, 30-45. Springer (2010)
[8] Ihlemann, C., Jacobs, S., Sofronie-Stokkermans, V. On local reasoning in verification. In: Ramakrishnan, C.R., Rehof, J.(eds.), TACAS'08. LNCS, vol. 4963, 265-281. Springer (2008)
[9] Mal'cev, A.I. Axiomatizable classes of locally free algebras of various types. The Metamathematics of Algebraic Systems. Collected Papers: 1936-1967, Studies in Logic and the Foundation of Mathematics, vol. 66, chap. 23. North-Holland, Amsterdam (1971)
[10] Peuter, D. and Sofronie-Stokkermans, V. On Inductive Verification and Synthesis. SYNT 2018.

[11] Totla, N. and Wies, T. Complete instantiation-based interpolation. In: Giacobazzi, R. and Cousot, R.(eds), POPL 2013, ACM (2013)
[12] Sofronie-Stokkermans, V. Hierarchic reasoning in local theory extensions. In: Nieuwenhuis, R.(ed.), CADE-20. LNAI, vol. 3632, 219-234. Springer (2005)
[13] Sofronie-Stokkermans, V. Hierarchical and modular reasoning in complex theories: The case of local theory extensions. In: Konev, B., Wolter, F.(eds.), FroCos'07. LNCS, vol. 4720, 47-71. Springer (2007)
[14] Sofronie-Stokkermans, V. Hierarchical reasoning for the verification of parametric systems. In: Giesl, J. and Hähnle, R.(eds), IJCAR 2010, LNCS, vol. 6173, 171-187. Springer (2010)
[15] Sofronie-Stokkermans, V.: Hierarchical reasoning and model generation for the verification of parametric hybrid systems. In: Bonacina, M.P.(ed), CADE-24, LNCS vol. 7898, 360-376. Springer (2013)
[16] Sofronie-Stokkermans, V.: On Interpolation and Symbol Elimination in Theory Extensions. In Olivetti, N. and Tiwari, A. (eds), IJCAR 2016, LNCS vol. 9706, 273–289. Springer (2016)

Adaptive Virtual Organisms: A Compositional Model for Hardware-Software Binding in the IoT Era

Gheorghe Ştefănescu[1]

University of Bucharest, România
gheorghe.stefanescu@fmi.unibuc.ro

Abstract

This talk focuses on extensions of regular languages / expressions in 2 and 3 dimensions, with applications to interactive, distributed, adaptive systems. We start with a new model of adaptive systems, called "virtual organisms"; then we briefly present Agapia, a structured HPC programming environment, here used for getting quick implementations for virtual organisms simulations.

The relation between a structure and the function(s) running on that structure is of central interest in many fields. Our presentation addresses this question with reference to computer science recent hardware/software advances, particularly in areas as IoT, CPS, robotics, self-systems, etc.

At the modeling, conceptual level, the key ingredient is the introduction of the concept of "virtual organism", to populate the intermediary level between rigid, slightly reconfigurable, hardware agents and abstract, intelligent, adaptive software agents. A virtual organism has a structure, resembling the hardware capabilities, and it runs low-level functions, implementing the software requirements. Roughly speaking, it is an adaptive, reconfigurable, distributed, interactive, open system, consisting of a network of heterogeneous computing nodes, with a constrained structural shape, and running a bunch of overlapping functions. The model is compositional, both in space (allowing the virtual organisms to aggregate into larger organisms) and in time (allowing the virtual organisms to get composed functionalities).

Technically, the virtual organisms presented here are in two dimensions (2D) and their structures are described by regular 2D pattens (including also the time dimension, we get a model using 3D regular patterns). By reconfiguration, an organism may change its structure to another structure belonging to the same 2D pattern. We illustrate the approach briefly describing three simple organisms: (1) a tree collector organism; (2) a feeding cell organism, consisting of a membrane, with collecting/releasing trees attached on its external/internal side; and (3) an organism consisting of a collection of connected feeding cell organisms.

The second part is a brief survey of work on register-voice structured interactive systems (rv-IS model) and on Agapia programming. The rv-IS model is based on space-time duality and is used for modeling, programming and reasoning about structured, open, interactive systems. Agapia, introduced ten years

ago, is a structured programming language where dataflow and control flow structures can be freely mixed. Currently, its compiler produces HPC executions, within either MPI or OpenMP environments

Infinite Sets in Fraenkel-Mostowski Theory

Andrei Alexandru and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science, Iasi, Romania

Abstract. We present some definitions for infinite sets with finite support, and prove that they are not equivalent in axiomatic Fraenkel Mostowski set theory although they are equivalent in several models of Zermelo-Fraenkel set theory with atoms.

The Fraenkel-Mostowski (FM) permutation models of Zermelo Fraenkel set theory with atoms (ZFA) were developed in 1930s by Fraenkel, Lindenbaum and Mostowski in order to prove the independence of the axiom of choice from the other axioms of ZFA, where ZFA is Zermelo-Fraenkel (ZF) set theory with the Axiom of Extensionality weakened to allow the existence of atoms. However, these models have been recently rediscovered by Gabbay and Pitts, and presented as a new axiomatic set theory called FM set theory, in order to formally explain concepts such as renaming, binding or fresh name in syntax. The axioms of FM set theory are precisely the ZFA axioms over an infinite set of atoms, together with the special axiom of finite support which claims that for each element x in an arbitrary set we can find a finite set supporting x according to an hierarchically constructed group action of the group of all permutation of atoms. Inductively defined finitely supported sets involving the name-abstraction together with Cartesian product and disjoint union can encode syntax modulo renaming of bound variables. More exactly, because they have no internal structure, atoms can be used to represent names. The finite support axiom is motivated by the fact that syntax can only involve finitely many names. Fresh names for the bound variables of a term can always be chosen from the atoms that are outside of the support (outside the set of the free names) of the related term. Binding is modelled by a certain concept of FM abstraction generalizing the notion of abstraction in the λ-calculus; actually FM set theory provides a formal framework for dealing with λ-terms modulo α-conversion. The construction of the universe of all FM-sets [3] is inspired by the construction of the universe of all admissible sets over an arbitrary collection of atoms [2]. The FM-sets represent a generalization of hereditary finite sets (which are particular admissible sets used to describe 'Gandy machines' [4]); actually, any FM-set is an hereditary finitely supported set.

Formally, let us denote the set of atoms in ZFA by A. We describe a model $\nu(A)$ of ZFA by generalizing the classical von-Neumann hierarchy on ordinals:

- $\nu_0(A) = \emptyset$;
- $\nu_{\alpha+1}(A) = A + \wp(\nu_\alpha(A))$ for every non-limit ordinal α;
- $\nu_\lambda(A) = \bigcup_{\alpha<\lambda} \nu_\alpha(A)$ (λ a limit ordinal);

– $\nu(A) = \underset{\alpha}{\cup}\nu_\alpha(A)$,

where $+$ is the disjoint union of sets, and $\wp(X)$ represents the powerset of X.

An invariant set $(X, \cdot)$ is actually a classical ZFA set X equipped with an action $\cdot$ on X of the group of all permutations of atoms, having the additional property that any element $x \in X$ is finitely supported. In a pair $(X, \cdot)$ formed by a ZFA set X and a group action $\cdot$ on X of the group of all permutations of atoms, an arbitrary element $x \in X$ is finitely supported if there exists a finite family $S \subseteq A$ such that any permutation of atoms that fixes S pointwise also leaves x invariant under the group action $\cdot$. An empty supported element $x \in X$ is called equivariant. If there exists an action $\cdot$ of the group of permutations of atoms on a set X, then there is an action $\star$ of the group of permutations of atoms on $\wp(X) = \{Y \,|\, Y \subseteq X\}$, defined by $(\pi, Y) \mapsto \pi \star Y := \{\pi \cdot y \,|\, y \in Y\}$ for all permutations of atoms π and all $Y \subseteq X$. The set $\wp_{fs}(X)$ represents the family of those finitely supported subsets of X as elements in $\wp(X)$ with respect to the action $\star$. An equivariant subset of an invariant set is itself an invariant set. For a set X we consider $\wp_{fin}(X) = \{Y \subseteq X \,|\, Y \text{finite}\}$ and $\wp_{cofin}(X) = \{Y \subseteq X \,|\, X \setminus Y \text{finite}\}$. If X is an invariant set, then $\wp_{fin}(X)$, $\wp_{cofin}(X)$ and $\wp_{fs}(X)$ are all are invariant sets.

On $\nu(A)$ we define the action $\cdot$ of the group of all permutation of atoms recursively by $\pi \cdot a = \pi(a)$ for all $a \in A$, $\pi \cdot X = \{\pi \cdot x \,|\, x \in X\}$ for all elements $X \in \nu(A)$ that are not atoms. A model of axiomatic FM set theory is represented by the von-Neumann cumulative hierarchy $FM(A)$ which is a subset of $\nu(A)$ defined as follows:

– $FM_0(A) = \emptyset$;
– $FM_{\alpha+1}(A) = A + \wp_{fs}(FM_\alpha(A))$ for every non-limit ordinal α;
– $FM_\lambda(A) = \underset{\alpha<\lambda}{\cup} FM_\alpha(A)$ (λ a limit ordinal);
– $FM(A) = \underset{\alpha}{\cup} FM_\alpha(A)$.

The disjoint union is used to emphasize the difference between 'sets' and 'atoms', i.e. atoms are not sets. A ZFA set X is an FM set (i.e. an element in $FM(A) \setminus A$) if and only if it is finitely supported as an element of $\nu(A)$ under the action $\cdot$ and Y is a FM-set or an atom for all $Y \in X$. $FM(A)$ is a subset of $\nu(A)$ which is itself an invariant set with the action $\cdot$ defined as above. Any FM set (i.e. any set belonging to the model $FM(A)$) is an hereditary finitely supported subset of the invariant set $FM(A)$. Thus, any FM set is a finitely supported subset of an invariant set. Furthermore, 'being an invariant set' means 'being an equivariant element at the following order stage in the hierarchical construction of $FM(A)$'. Thus, the invariant sets in the FM cumulative hierarchy are defined as those equivariant elements of $FM(A)$. Whenever an atom a appears in (the construction of) an FM set X, the effect of a permutation of atoms π on a under $\cdot$ will be $\pi(a)$.

The main idea of reformulating a classical result into the FM framework is to analyze if there exists a valid result obtained by replacing "object" with "atomic object with finite support (under canonical permutation action)" in

the classical result. The invariant set $FM(A)$ contains both the family of 'non-atomic' (ordinary) ZF sets, defined by employing an hierarchical construction over $\emptyset$ (these are proved to be trivial FM sets) and the family of 'atomic' sets with finite support (hierarchically constructed from the set A of atoms); the question is if a classical ZF result (obtained in ZF for non-atomic sets) can be adequately reformulated by replacing 'set' with 'finitely supported set' in order to remain valid also for atomic sets with finite support. In order to reformulate a general ZF result into the FM framework, the proof of the related FM result should not brake the principle that any construction has to be finitely supported, which means that the related proof should be internally consistent in FM and not retrieved from ZF. There exist ZF results (and also ZFA results) that does not remain valid when translated into FM (see [1]). In the following we describe four classical definitions on 'infinite' that are not equivalent in FM axiomatic set theory, although they are equivalent in several models of ZFA set theory and in several models of ZF set theory.

Let X be a finitely supported subset of an invariant set (particularly, X is an FM set). We consider the following definitions of infinite sets:

- X is called *FM usual infinite* if X does not correspond one-to-one and onto to a finite ordinal.
- X is called *FM Tarski infinite* if there exists a finitely supported one-to-one mapping of X onto $X \times X$.
- X is called *FM Dedekind infinite* if there exist a finitely supported one-to-one mapping of X onto a finitely supported proper subset of X.
- X is called *FM ascending infinite* if there is a finitely supported increasing countable chain $X_0 \subseteq X_1 \subseteq \ldots \subseteq X_n \subseteq \ldots$ (i.e. the map $n \mapsto X_n$ is finitely supported) with $X \subseteq \cup X_n$, but there does not exist $n \in \mathbb{N}$ such that $X \subseteq X_n$;

The properties of FM Dedekind infinite and FM ascending infinite sets are:

1. Let X be a finitely supported subset of an invariant set Y (particularly, X is an FM set). Then X is FM Dedekind infinite if and only if there exists a finitely supported one-to-one mapping $f : \mathbb{N} \to X$.
2. Let X be an infinite finitely supported subset of an invariant set Y (particularly, X is an FM set). Then the sets $\wp_{fs}(\wp_{fin}(X))$ and $\wp_{fs}(\wp_{fs}(X))$ are FM Dedekind infinite.
3. Let X be a finitely supported subset of an invariant set Y (particularly, X is an FM set) such that X does not contain an infinite subset Z with the property that all the elements of Z are supported by the same set of atoms. Then neither X nor $\wp_{fin}(X)$ is FM Dedekind infinite.
4. Let X and Y be two finitely supported subsets of an invariant set Z (particularly, X and Y are FM sets). If neither X nor Y is FM Dedekind infinite, then neither $X \times Y$ nor $X + Y$ is FM Dedekind infinite.
5. Let X be a finitely supported subset of an invariant set Y (particularly, X is an FM set). If X is FM Dedekind infinite, then X is FM ascending infinite.

The reverse implication is not valid because the set $\wp_{fin}(A)$ is FM ascending infinite while it is not FM Dedekind infinite.

6. Let X be a finitely supported subset of an invariant set Y. If X is FM usual infinite, then the set $\wp_{fin}(X)$ is FM ascending infinite.

The following sets and all of their FM usual infinite subsets are FM usual infinite, but they are not FM Dedekind infinite.

- The invariant set A of atoms and its FM power set $\wp_{fs}(A)$.
- The invariant set P_A of all finitely supported bijections of A onto A.
- The invariant sets $FM_0(A)$, $FM_1(A)$ and $FM_2(A)$.
- The FM power sets: $\wp_{fin}(A)$, $\wp_{cofin}(A)$, $\wp_{fin}(\wp_{fs}(A))$, $\wp_{fin}(\wp_{cofin}(A))$, $\wp_{fin}(\wp_{fin}(A))$, $\wp_{fin}(P_A)$.
- Any inductive construction of finite power sets of form: $\wp_{fin}(\dots\wp_{fin}(A))$, $\wp_{fin}(\dots\wp_{fin}(\wp_{cofin}(A)))$, $\wp_{fin}(\dots\wp_{fin}(\wp_{fs}(A)))$ and $\wp_{fin}(\dots\wp_{fin}(P_A))$.
- Every finite Cartesian combination between A, $\wp_{fin}(A)$, $\wp_{cofin}(A)$, $\wp_{fs}(A)$ and P_A.
- The disjoint unions $A+P_A$, $A+\wp_{fs}(A)$, $\wp_{fs}(A)+P_A$ and $A+\wp_{fs}(A)+P_A$.

The following sets and all of their supersets, their power sets and the families of their finite subsets, are both FM usual infinite and FM Dedekind infinite.

- The invariant sets $\wp_{fs}(\wp_{fs}(A))$ and $\wp_{fs}(\wp_{fin}(A))$.
- The invariant sets $FM_\alpha(A)$ whenever $\alpha > 2$ and α is not as limit ordinal.
- The invariant set $FM_\lambda(A)$ whenever λ is a limit ordinal.
- The invariant FM universe $FM(A)$.

The following sets are FM usual infinite, but they are not FM Tarski infinite.

- The invariant sets A.
- The invariant sets $\wp_{fin}(A)$, $\wp_{cofin}(A)$ and $\wp_{fs}(A)$.

The results presented in this paper are also valid in the framework of nominal sets presented by A Pitts as a ZF categorical alternative approach to FM set theory. This is because if we consider A as a fixed ZF set, invariant sets defined as in this paper (as sets equipped with canonical group actions of the group of all permutation of A satisfying a finite support requirement) are actually nominal sets (developed for possibly non-countable sets of atoms); we used the term 'invariant' motivated by Tarski's approach regarding logicality.

References

1. Alexandru, A., Ciobanu, G.: Finitely Supported Mathematics: An Introduction, Springer, 2016.
2. Barwise, J., Admissible Sets and Structures: An Approach to Definability Theory, Perspectives in Mathematical Logic, vol.7, Springer, 1975.
3. Gabbay, M.J., Pitts, A.M., A new approach to abstract syntax with variable binding, Formal Aspects of Computing. 13(3-5) (2001), 341–363.
4. Gandy, R., Church's thesis and principles for mechanisms, In: Barwise, J., Keisler, H.J., Kunen, K. (eds.) The Kleene Symposium, North-Holland, 1980, pp. 123–148.
5. Howard, P., Rubin, J.E.: Consequences of the Axiom of Choice. Mathematical Surveys and Monographs, vol.59, American Mathematical Society, 1998.

Timed Migration with Costs in Distributed Systems

Bogdan Aman and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science, Iaşi, Romania
and "A.I.Cuza" University, Iaşi, Romania
bogdan.aman@iit.academiaromana-is.ro, gabriel@info.uaic.ro

Abstract. This paper presents an extension of TiMo , a process algebra for mobile systems equipped with timers that serve as deadlines for process migration and communication. TiMo is extended with a cost function that characterizes the price of remaining at a location or performing a communication action. Beyond providing an operational semantics for costs, a translation of this extension to weighted timed automata is also presented. In this way, verification is possible by using a CTL-based verifier capable of reasoning about timed automata.

In distributed systems, mobile agents and the interaction between these agents may introduce new and unexpected behaviours. We employ a timed language named TiMo [3] able to describe complex distributed systems in which processes can migrate within a network defined by a number of explicit locations. Processes may communicate when they are present at the same location. Timeouts are used to coordinate interactions in time and space by using migration and communication; a timeout for migration specifies a temporal interval after which the process must move to another location. An extension called cTiMo uses costs on locations and actions: the price of a location gives the cost of staying in that location (per time unit), and the price of a action gives the cost of performing that action. For example, our formalism can be used to model taxi trips as the fares are commonly priced based on the distance and time taken for a trip.

1 Syntax and Semantics of cTiMo

The syntax of cTiMo (cost TiMo) is given in Table 1. The timeout applied to a migrating process allow to specify how many time units are required by a process to move from one location to another, while cost constraints applied to a migrating process specify the costs needed to perform the movements between locations. As location l can be a variable, the migration supports a flexible scheme for the movement of processes from one location to another. A timer in cTiMo is denoted either by t (for migration actions) or Δt (for output and input actions). When it is associated with a migration process $\mathsf{go}_c^t\, l\ \mathsf{then}\ P$ it indicates that process P moves from the current location to location l after t time units, while the cost of movement is c.

Processes are further constructed from the terminated process 0. A located process $l[[_c\, P]]$ specifies a process P running at location l where the cost of

Processes	P, Q	$::= \mathsf{go}_c^t\ l\ \mathsf{then}\ P \mid$	(move)
		$a_c^{\Delta t}!\langle v\rangle\ \mathsf{then}\ P\ \mathsf{else}\ Q \mid$	(output)
		$a_c^{\Delta t}?(u)\ \mathsf{then}\ P\ \mathsf{else}\ Q \mid$	(input)
		$0 \mid$	(termination)
		$id(v)$	(recursion)
Located processes	L	$::= l[[_c\ P]]$	
Networks	N	$::= L \mid L \mid N$	

Table 1. cTiMo Syntax

computing is c, and a network is built from its components $N \mid N'$. The structural equivalence $\equiv$ over networks is defined as the smallest congruence given by the equalities: $N \mid 0 \equiv N$, $N \mid N' \equiv N' \mid N$ and $(N \mid N') \mid N'' \equiv N \mid (N' \mid N'')$. Its role is to rearrange a network in order to apply the rules of the *operational semantics* given in Table 2.

$$\text{(DMove)}\quad \frac{t \geq t' \geq 0}{l[[_c\ \mathsf{go}_{c'}^{t}\ l'\ \mathsf{then}\ P]] \overset{t',c*t'}{\rightsquigarrow} l[[_c\ \mathsf{go}_{c'}^{t-t'}\ l'\ \mathsf{then}\ P]]}$$

$$\text{(Move0)}\quad l[[_c\ \mathsf{go}_{c''}^{0}\ l'\ \mathsf{then}\ P]] \xrightarrow{l \triangleright l', c''} l'[[_{c'}\ P]]$$

$$\text{(Com)}\quad \begin{array}{c} l[[_c\ a_{c_1}^{\Delta t}!\langle v\rangle\ \mathsf{then}\ P\ \mathsf{else}\ Q]] \mid l[[_c\ a_{c_2}^{\Delta t'}?(u)\ \mathsf{then}\ P'\ \mathsf{else}\ Q']] \\ \xrightarrow{!v@l \mid ?u@l, c_1 \mid c_2} l[[_c\ P]] \mid l[[_c\ \{v/u\}P']] \end{array}$$

Table 2. Operational Semantics of cTiMo (selection)

The operational semantics of cTiMo is presented in Table 2. The multiset labelled transitions of form $N \xrightarrow{\Lambda, C} N'$ use a multiset Λ to indicate the actions executed in parallel in one step, and a multiset C of costs to indicate the costs of each of the actions from Λ. The order of the costs in C depends on the order of actions in Λ, as each action has an unique execution cost. The transitions of form $N \overset{t,C}{\rightsquigarrow} N'$ represent a time step of length t and costs C for a network N.

In rule (Move0), the process $\mathsf{go}_{c''}^{t}\ l'\ \mathsf{then}\ P$ migrates from location l to l' and then evolves as process P. The migration cost is c'', and the cost of its computation at the new location l' is computed by using the cost c'.

In rule (Com), a process $a_{c_1}^{\Delta t}!\langle v\rangle\ \mathsf{then}\ P\ \mathsf{else}\ Q$ at location l succeeds in sending a tuple of values v over channel a to process $a_{c_2}^{\Delta t}?(u)\ \mathsf{then}\ P'\ \mathsf{else}\ Q'$ at the same location l. Both processes continue to execute at location l, the first one as P and the second one as $\{v/u\}P'$. The label $!v@l|?u@l, c_1|c_2$ indicates the fact that the first process performs the output action $!v@l$ with cost c_1, and the second process performs the complementary input action $?u@l$ with cost c_2. The rules devoted to the passing of time are starting with 'D'. A complete computational step is captured by a derivation $N \xrightarrow{\Lambda, C} N_1 \overset{t,C'}{\rightsquigarrow} N'$. This means that a derivation is a sequence of individual actions with costs, followed by a time step with costs. We say that N' is directly reachable from N.

As the cost cannot be tested upon in any guard or invariants, it does not influence the behaviour of the system. This means that a network N is guaranteed to proceed, assuming progress of its counterpart without costs (i.e., $erase(N)$). The erasure $erase(N)$ of a network N is defined inductively by:

$$erase(N) = \begin{cases} erase(L) & \text{if } N = L \\ erase(L) \mid erase(N') & \text{if } N = L \mid N' \end{cases}$$

$$erase(L) = l[[erase(P)]] \text{ if } L = l[[_c\, P]]$$

$$erase(P) = \begin{cases} \mathsf{go}^t\ l \text{ then } erase(P') & \text{if } P = \mathsf{go}_c^t\ l \text{ then } P' \\ a^{\Delta t}!\langle v\rangle \text{ then } erase(R) \text{ else } erase(Q) & \text{if } P = a_c^{\Delta t}!\langle v\rangle \text{ then } R \text{ else } Q \\ a^{\Delta t}?(u) \text{ then } erase(R) \text{ else } erase(Q) & \text{if } P = a_c^{\Delta t}?(u) \text{ then } R \text{ else } Q \\ 0 & \text{if } P = 0 \\ erase(id(v)) & \text{if } P = id(v) \end{cases}$$

We get the following relevant results.

Theorem 1. *1. If $N \xrightarrow{\Lambda,C} N'$, then $erase(N) \xrightarrow{\Lambda} erase(N')$.*
2. If $N \overset{t,C'}{\rightsquigarrow} N'$, then $erase(N) \overset{t}{\rightsquigarrow} erase(N')$.

Corollary 1. *1. If $erase(N) \xrightarrow{\Lambda}$, then there exist N', N'' and C such that $N \xrightarrow{\Lambda,C} N''$, $erase(N) \xrightarrow{\Lambda} N'$ and $erase(N'') = N'$.*
2. If $erase(N) \overset{t}{\rightsquigarrow}$, then there exist N', N'' and C such that $N \overset{t,C}{\rightsquigarrow} N''$, $erase(N) \overset{t}{\rightsquigarrow} N'$ and $erase(N'') = N'$.

Theorem 2. *For any networks N, N' and N'', the following sentences hold:*

1. $N \overset{0,0}{\rightsquigarrow} N$;
2. *If* $N \overset{t,C}{\rightsquigarrow} N'$ *and* $N \overset{t,C}{\rightsquigarrow} N''$, *then* $N' \equiv N''$;
3. $N \overset{(t+t'),C''}{\rightsquigarrow} N'$ *if and only if there is a* N'' *such that* $N \overset{t,C}{\rightsquigarrow} N''$ *and* $N'' \overset{t',C'}{\rightsquigarrow} N'$ *where* $C'' = C' + C''$ *(the sum is component-wise).*

In what follows, if $L_1|L_2 \xrightarrow{!v@l|?u@l,c_1|c_2} L_1'|L_2'$, then we have that L_1 and L_2 evolve in parallel such that $L_1 \xrightarrow{!v@l,c_1} L_1'$ and $L_2 \xrightarrow{?u@l,c_2} L_2'$.

Definition 1 (Execution Cost). *Let $e_L = L_0 \xrightarrow{\Lambda_0,c_0} L_1 \overset{d_1,c_1}{\rightsquigarrow} L_2 \ldots \overset{d_n,c_n}{\rightsquigarrow} L_{n+1}$ be a finite execution of a located process $L = l[[_c\, P]]$. Then $cost(e_L)$ representing the cost of the execution e_L is the sum $\sum_{0\le i\le n} c_i$, while $time(e_L)$ representing the execution time of e_L is the sum $\sum_{0\le i\le n} d_i$.*

For a given process $L' = l'[[_{c'}\, P']]$, the minimal cost $mincost_L(L')$ of reaching L' starting from $L = l[[_c\, P]]$ is the infimum of the costs of executions starting in L and ending in L'. Similarly, the minimal cost $mincost_L(l')$ of reaching a location l' is the infimum of the costs of finite executions ending in located processes of the form $l'[[_{c'}\, P']]$.

Proposition 1. *Let e_L be a finite execution of a located process L. If the location costs are set to 1 and the migration and communication costs are set to 0, then $cost(e_L) = time(e_L)$.*

Proposition 2. *If a located process $L = l[[_c\, P]]$ has its process P without any communication action, and any finite execution e_L of L leads to a located process $L' = l'[[_{c'}\, 0]]$, then $cost(e_L) = mincost_L(L')$.*

Additionally, if P contains only a migration towards location l' in its last step of the execution, then $cost(e_L) = mincost_L(L') = mincost_L(l')$.

2 Translating cTiMo into Weighted Timed Automaton

Weighted timed automata [2] extend classical timed automata [1] with cost information added to both locations and edges. The cost added to a location represents the price per time unit for staying in that location, while the cost of an edge represents the price for taking that transition. That way, every run in the automaton has a global cost which is the sum of the costs along the run of every delay and discrete transition.

Given a component $l[[_{c_0}\, P]]$ of an cTiMo network, we translate it into a weighted timed automaton $\mathcal{A} = (L, l_0, T, \lambda, \mathit{cost})$ with a local clock x, where $L = \{l_0\}$, $T = \emptyset$, $\lambda(l_0) = \emptyset$ and $cost(l_0) = c_0$. The components L, T, λ and *cost* are updated depending on the structure of process P. In what follows we show how to update these components for the output $P = a_c^{\Delta t}!\langle v\rangle$ then P_1 else P_1':

- $L = L \cup \{l_{i+1}, l_{i+2}\}$;
- $T = T \cup \{n, x <= t, a!, \{x = 0, at = v\}, l_{i+1}\} \cup \{n, x == t, \tau, x = 0, l_{i+2}\}$;
- $\lambda(n) = \{x <= t\}$.
- $cost(n) = \begin{cases} c_n & \text{if } n = l \\ c_0 & \text{if } n = l_0 \end{cases}$.
- $cost(\{n, x <= t, a!, \{x = 0, at = v\}, l_{i+1}\}) = c$;
- $cost(\{n, x == t, \tau, x = 0, l_{i+2}\}) = c$.

Building a weighted timed automaton for each component of a network N leads to the equivalence between the cTiMo network N and its corresponding weighted timed automaton $\mathcal{A}$ in the initial state $\langle l_N, v_N\rangle$ (i.e., $(\mathcal{A}, \langle l_N, v_N\rangle)$).

Theorem 3. *Given a cTiMo network N, there exists a network $\mathcal{A}_N$ of parallel timed automata having a bisimilar behaviour. Formally, $N \sim (\mathcal{A}, \langle l_N, v_N\rangle)$.*

The size of a weighted timed automata $\mathcal{A}_N$ is polynomial with respect to the size of a network N, and the state spaces have the same number of states.

This translation to weighted timed automata allows to verify various properties of cTiMo networks by using a CTL-based verifier capable of reasoning about timed automata. It exists a version of Uppaal called Uppaal CORA [4] working on priced timed automata; however, Uppaal CORA is used only to find optimal paths matching goal conditions, and cannot refer to the cost variable in expressions or verifications.

References

1. R. Alur, D.L. Dill. A theory of timed automata. *Theoretical Computer Science* **126**, 183–235 (1994).
2. R. Alur, S. La Torre, G.J. Pappas, Optimal paths in weighted timed automata. *Lecture Notes in Computer Science* **2034**, 49–62 (2001).
3. G. Ciobanu, M. Koutny. Modelling and verification of timed interaction and migration. *Lecture Notes in Computer Science* **4961**, 215–229 (2008).
4. Uppaal CORA, http://people.cs.aau.dk/∼adavid/cora, Oct 2017.

Unification in Matching Logic

Andrei Arusoaie

Alexandru Ioan Cuza, University of Iaşi

1 Introduction

Unification of terms is widely used as a basic operation in automated reasoning, proofs assistants, rewriting systems, programming language-type systems implementations, and other tools. In this extended abstract we present an preliminary investigation of the interaction between unification and unification algorithms on the one side, and Matching Logic [5] on the other side. Matching Logic (ML) is a novel framework for specifying programming languages semantics and for reasoning about programs. Its formulas, also called *patterns*, resemble term templates (with variables) which are eventually constrained by some logical conditions. In the latest version of ML, simple unification of terms is advertised as being a simple conjunction of ML formulas. In this work we are interested in expressing the basic notions from the unification theory into ML and then investigate to what extent existing algorithms for unification are useful in this new setting.

1.1 Unification

Unification comes in many flavours. The simplest form of unification is the *(first-order) syntactic unification*. For example, if terms are defined as usual (i.e., using *variables*, *constants*, and *functions symbols*) then the terms $f(x, 3)$ and $f(y, z)$ are syntactically unifiable by replacing variables x, y, and z by constant 3. This replacement is modelled using *substitutions*, which are mappings from variables to terms: $\sigma = \{x \mapsto 3, y \mapsto 3, z \mapsto 3\}$. Formally, a substitution σ is a unifier of two terms t and t' if $t\sigma = t'\sigma$, that is, applied to t and t', the substitution σ produces the same term: $f(x, 3)\sigma = f(3, 3) = f(y, z)\sigma$. Some unifiers can be *more general* than others. A unifier σ' is more general than σ, written $\sigma' \leq \sigma$, if there exists a substitution η such that $\sigma'\eta = \sigma$. For instance, if $\sigma' = \{x \mapsto y, y \mapsto y, z \mapsto 3\}$ then $\sigma' \leq \sigma$ because $\sigma'\eta = \sigma$ with $\eta = \{y \mapsto 3\}$. [1] In general, we are interested in finding the *most general* unifier w.r.t. $\leq$ (a.k.a. *mgu*). An efficient algorithm for finding the *mgu* was proposed in [3].

Unification algorithms are also important components for deductive systems. In this context, it is quite common to combine unification with equational theories (the so-called *(first-order) unfication modulo equations*) because it has some advantages. Terms that are not syntactically unifiable can be *unifiable modulo (some) theories*. For example, terms $f(2, x)$ and $f(3, y)$ are unifiable modulo

[1] Composition of substitutions: $\sigma'\eta = \{x \mapsto y, y \mapsto y, z \mapsto 3\}\eta = \{x \mapsto (y\eta), y \mapsto (y\eta), z \mapsto (3\eta)\} = \{x \mapsto 3, y \mapsto 3, z \mapsto 3\} = \sigma$.

commutativity of f with the unifier $\sigma = \{x \mapsto 3, y \mapsto 3\}$: $f(2,x)\sigma = f(2,3) =_C f(3,2) = f(3,y)\sigma$. An interesting fact is that terms $f(x,y)$ and $f(3,2)$ have two unifiers: $\sigma_1 = \{x \mapsto 2, y \mapsto 3\}$ and $\sigma_2 = \{x \mapsto 3, y \mapsto 2\}$. Note that these unifiers are not comparable w.r.t. $\leq$. In unfication modulo equations the role of the *mgu* is taken by a *complete set of unifiers*. In general we are interested to find the minimal complete set of unifiers but these do not always exist. There are plenty of results in the literature related to unification modulo equations; here we recall only a few: associative [4], associative-idempotent [1], associative-commutative [2, 6].

2 Matching Logic

Matching logic (ML) [5] started as a logic over a particular case of constrained terms, but now it is developed as a solid program logic framework. The formulae in ML are *patterns*, where a pattern φ_s of sort s is defined by:

$$\varphi_s ::= x_s \mid f(\varphi_{s_1}, \ldots, \varphi_{s_n}) \mid \neg\varphi_s \mid \varphi_s \wedge \varphi_s \mid \exists x.\varphi_s$$

where x_s ranges over the variables of sort s ($x_s \in \mathcal{X}_s$), f ranges over $F_{s_1 \ldots s_n, s}$ and x ranges over the set of variables (of any sort). The models in ML are similar to Σ-models, but here the symbols f are interpreted as relations instead of functions: $M_f : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$ (note the use of $\mathcal{P}(M_s)$ as the co-domain). The meaning of patterns is given by the means of valuations ρ as well, but the result of the interpretation is a set of elements that the pattern "matches", similar to the worlds in modal logic. If $\rho : \mathcal{X} \to M$ is a variable valuation and φ a pattern, then $\overline{\rho}(\varphi)$ is inductively defined as follows:

1. $\overline{\rho}(x) = \{\rho(x)\}$;
2. $\overline{\rho}(f(\varphi_1, \ldots, \varphi_n)) = \bigcup\{M_f(v_1, \ldots, v_n) \mid v_i \in \overline{\rho}(\varphi_i), i = 1, \ldots, n\}$[2];
3. $\overline{\rho}(\neg\varphi) = M_s \setminus \overline{\rho}(\varphi)$, where the sort of φ is s;
4. $\overline{\rho}(\varphi_1 \wedge \varphi_2) = \overline{\rho}(\varphi_1) \cap \overline{\rho}(\varphi_2)$, where φ_1 and φ_2 have the same sort;
5. $\overline{\rho}(\exists x.\varphi) = \bigcup_{v \in M_s} \overline{\rho}[v/x](\varphi)$, where $x \in \mathcal{X}_s$ and $\rho[v/x]$ is the valuation ρ' s.t. $\rho'(y) = \rho(y)$ for all $y \neq x$, and $\rho'(x) = v$.

The derived patterns are defined as expected: $\top_s \triangleq \exists x.x$ (x of sort s), $\bot_s = \neg\top_s$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \varphi_2)$, $\varphi_1 \implies \varphi_2 = \neg\varphi_1 \vee \varphi_2$, and so on. A model M_s *satisfies* φ_s, written $M_s \models \varphi_s$, if $M_s = \overline{\rho}(\varphi_s)$ for each variable valuation ρ. A pattern φ is *valid* (written $\models \varphi$) iff $M \models \varphi$ for all models M. A *matching logic specification* is a triple (S, Σ, F) where F contains ML patterns which serve as axioms.

Matching logic uses a special *definedness symbol* $\lceil\_\rceil^{s_2}_{s_1} \in F_{s_1,s_2}$. The semantics of $\lceil\varphi\rceil^{s_2}_{s_1}$ is as follows: if $\rho : \mathcal{X} \to M$ then $\overline{\rho}(\lceil\varphi\rceil^{s_2}_{s_1})$ is either $\emptyset$ (i.e., $\overline{\rho}(\bot_{s_2})$) when $\overline{\rho}(\varphi) = \emptyset$ ($= \overline{\rho}(\bot_{s_1})$, i.e., φ undefined in ρ), or is M_{s_2} (i.e., $\overline{\rho}(\top_{s_2})$) when $\overline{\rho}(\varphi) \neq \emptyset$ (i.e., φ defined). Another construct introduced in ML is *totality*, which is dual to definedness and is defined as: $\lfloor\varphi\rfloor^{s_2}_{s_1} \equiv \neg\lceil\neg\varphi\rceil^{s_2}_{s_1}$. Finally, the *equality* $=^{s_2}_{s_1}$ in ML is defined as: $\varphi =^{s_2}_{s_1} \varphi' \equiv \lfloor\varphi \leftrightarrow \varphi'\rfloor^{s_2}_{s_1}$. When the sorts s_1 and s_2 are understood from the context we use simplified notation $=$ instead of $=^{s_2}_{s_1}$.

[2] For constants c (case $n = 0$) we have $\overline{\rho}(c) = M_c$.

3 From Unification Theory to Matching Logic

Terms can be naturally expressed using the ML syntax. They are in fact a particular type of formulas called *term patterns*, which are formulas containing only *functional symbols*. A symbol $f \in \Sigma_{s_1 \dots s_n, s}$ is a *functional symbol* if the pattern $\exists y.f(x_1, \dots, x_n) = y$ is an axiom in F.

Another useful type of ML formulas are M-*predicates*. Pattern φ_s is an M-*predicate* in (S, Σ, F) iff for any valuation $\rho : \mathcal{X} \to M$, $\overline{\rho}(\varphi_s)$ is either M_s or $\emptyset$. Also, φ_s is called a *predicate* iff it is a predicate in all models M.

Since term patterns are in fact terms with variables, the existing unification algorithms can be used to get either the most general unifier or a complete set of unifiers for two given terms patterns, say t and t'. Unifiers are substitutions which applied to t and t' produce the same term. These substitutions can be encoded as predicates which we call *substitution patterns*. A *substitution pattern* that corresponds to a unifier $\sigma = \{x_i \mapsto u_i \mid i = 1, \dots, n\}$ is a predicate of the form $\phi^\sigma \triangleq \bigwedge_{i=1}^m x_i = u_i$. We use the same notation $t\sigma$ to denote the corresponding ML formula obtained after applying σ to term pattern t as follows: $x_i\sigma = u_i$ if $(x_i \mapsto u_i) \in \sigma$; $x\sigma = x$ if $(x \mapsto \_) \notin \sigma$; and finally, $f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$.

We explain the above notions by simple examples. Terms $t \triangleq f(x, g(1), g(z))$ and $t' \triangleq f(g(y), g(y), g(g(x))$ are terms patterns in ML. For a unifier $\sigma = \{x \mapsto g(1), y \mapsto 1, z \mapsto g(g(1))\}$ of t and t' there is a corresponding predicate $\phi^\sigma \triangleq x = g(1) \wedge y = 1 \wedge z = g(g(1))$. Now, both $t\sigma$ and $t'\sigma$ are the ML formula $f(g(1), g(1), g(g(1)))$. One may be tempted to say that $t\sigma$ is equivalent to $t \wedge \phi^\sigma$. However, this is not true if ϕ^σ is not valid, i.e., there is a model M and a variable valuation $\rho : \mathcal{X} \to M$ such that $\overline{\rho}(\phi^\sigma) = \emptyset$; in this case, if $t = x$ such that x does not occur in σ (i.e., $x\sigma = x$) then $\overline{\rho}(x\sigma) \neq \emptyset = \rho(x) \cap \overline{\rho}(\phi^\sigma) = \overline{\rho}(x \wedge \phi^\sigma)$; that is, $x\sigma \neq x \wedge \phi^\sigma$. The following lemma captures the link between $t\sigma$ and $t \wedge \phi^\sigma$[3]:

Lemma 1. *If t is a term pattern and σ is a substitution then $t\sigma \wedge \phi^\sigma \leftrightarrow t \wedge \phi^\sigma$.*

Proof. Let us choose an arbitrary model M. We prove that $M \models t\sigma \wedge \phi^\sigma \leftrightarrow t \wedge \phi^\sigma$. By Proposition 5.9 in [5] $M \models t\sigma \wedge \phi^\sigma \leftrightarrow t \wedge \phi^\sigma$ iff $\overline{\rho}(t\sigma \wedge \phi^\sigma) = \overline{\rho}(t \wedge \phi^\sigma)$ for any $\rho : Var \to M$. By Definition 2.3 in [5] this holds iff $\overline{\rho}(t) \cap \overline{\rho}(\phi^\sigma) = \overline{\rho}(t\sigma) \cap \overline{\rho}(\phi^\sigma)$. When $\overline{\rho}(\phi^\sigma) = \emptyset$ this equality holds trivially. When $\overline{\rho}(\phi^\sigma) = M$ then $\overline{\rho}(t\sigma) \cap \overline{\rho}(\phi^\sigma) = \overline{\rho}(t) \cap \overline{\rho}(\phi^\sigma)$ iff $\overline{\rho}(t\sigma) \cap M = \overline{\rho}(t) \cap M$ iff $\overline{\rho}(t\sigma) = \overline{\rho}(t)$. We proceed by induction on t:

- *Base case.* $t = x$. Recall that $\phi \triangleq \bigwedge_{i=1}^m x_i = u_i$. We have two sub-cases here:
  1. $x \in \{x_1, \dots, x_m\}$: since $\overline{\rho}(\phi) = M$ then $\overline{\rho}(\bigwedge_{i=1}^m x_i = u_i) = M$ which implies $\bigcap_i^m \overline{\rho}(x_i = u_i) = M$. Thus, $\overline{\rho}(x_i = u_i) = M$ and particularly $\rho(x) = \overline{\rho}(u) = \overline{\rho}(x\sigma)$.
  2. $x \notin \{x_1, \dots, x_m\}$: in this case $\overline{\rho}(x\sigma) = \rho(x)$.
- *Inductive step.* $t = f(t_1, \dots, t_n)$ and the inductive hypothesis holds for all subterm patterns $t_1, \dots, t_n$. Then: $\overline{\rho}(f(t_1, \dots, t_n)[\phi]) = \overline{\rho}(f(t_1\sigma, \dots, t_n\sigma)) = M_f(\overline{\rho}(t_1\sigma), \dots, \overline{\rho}(t_n\sigma))) = M_f(\overline{\rho}(t_1), \dots, \overline{\rho}(t_n))) = \overline{\rho}(f(t_1, \dots, t_n))$ using the inductive hypothesis and the definition of $\overline{\rho}$. □

[3] Proof can be found in Appendix.

In the unification theory the unifiers of two terms t and t' are substitutions which map t and t' to the same term. One question is how to express in ML the fact that σ is a unifier of t and t'? Another more interesting question is how to express in ML the fact that σ is the most general unifier of t and t'? In [5] the author claims that the most general unifier of t and t' is the conjunction $t \wedge t'$. He also suggests that $t \wedge t'$ can be transformed into an equivalent formula of a convenient form, that is, a term pattern constrained by some predicate containing equalities (as suggested by the definition of ϕ^σ). By applying Propositions 5.26, 5.24, 5.12, and 5.10 to the conjunction of terms t and t', we obtain:

$$\begin{aligned}
t \wedge t' &= f(x, g(1), g(z)) \wedge f(g(y), g(y), g(g(x)) \\
&= f(x \wedge g(y), g(1) \wedge g(y), g(z) \wedge g(g(x))) \\
&= f(x \wedge g(y), g(1 \wedge y), g(z \wedge g(x))) \\
&= f(x \wedge (x = g(y)), g(1 \wedge (1 = y)), g(z \wedge (z = g(x)))) \\
&= f(x \wedge (x = g(y)), g(1) \wedge (1 = y), g(z) \wedge (z = g(x))) \\
&= f(x, g(1), g(z)) \wedge (x = g(y)) \wedge (1 = y) \wedge (z = g(x)) \\
&= f(x, g(1), g(z)) \wedge (x = g(y)) \wedge (y = 1) \wedge (z = g(x)) \\
&= t \wedge (x = g(y)) \wedge (y = 1) \wedge (z = g(x))
\end{aligned}$$

For the example above the constraint $(x = g(y)) \wedge (y = 1) \wedge (z = g(x))$ corresponds to the mgu $\sigma = \{x \mapsto g(y), y \mapsto 1, z \mapsto g(x)\}$ of t and t'. Moreover, the (ML) mgu $t \wedge t'$ can be written as a constrained term pattern: $t \wedge \phi^\sigma$.

Lemma 2. *If σ is a unifier of t and t' then $\phi^\sigma \rightarrow (t = t')$ is valid in ML.*

Proof. We have to prove that for all models M and for all valuations $\rho : \mathcal{X} \rightarrow M$, $\overline{\rho}(\phi^\sigma \rightarrow (t = t')) = M$. This is equivalent to showing that $\overline{\rho}(\phi^\sigma) \subseteq \overline{\rho}(t = t'))$ (by the Proposition 2.6 in [5]). The case $\overline{\rho}(\phi^\sigma) = \emptyset$ is trivial. When $\overline{\rho}(\phi^\sigma) = M$ it is sufficient to show that $\overline{\rho}(t = t') = M$, namely, $\overline{\rho}(t) = \overline{\rho}(t')$.

From Lemma 1 we have $t\sigma \wedge \phi^\sigma \leftrightarrow t \wedge \phi^\sigma$ and $t'\sigma \wedge \phi^\sigma \leftrightarrow t' \wedge \phi^\sigma$. This implies that $\overline{\rho}(t\sigma) \cap \overline{\rho}(\phi^\sigma) = \overline{\rho}(t) \cap \overline{\rho}(\phi^\sigma)$ and $\overline{\rho}(t'\sigma) \cap \overline{\rho}(\phi^\sigma) = \overline{\rho}(t') \cap \overline{\rho}(\phi^\sigma)$. Because $\overline{\rho}(\phi^\sigma) = M$, we have $\overline{\rho}(t) = \overline{\rho}(t\sigma)$ and $\overline{\rho}(t'\sigma) = \overline{\rho}(t')$. Next, using the fact that σ is a unifier (i.e., $t\sigma = t'\sigma$) we obtain $\overline{\rho}(t) = \overline{\rho}(t\sigma) = \overline{\rho}(t'\sigma) = \overline{\rho}(t')$. □

Lemma 2 captures nicely the intuition of unification: the ML formula ϕ^σ which corresponds to a unifier σ of t and t' implies the equality of t and t'.

Conclusions and future work. We presented a preliminary investigation of unification in the context of Matching Logic. Unifiers can be encoded as a special type of predicates in Matching Logic. Also, the most general unifier of two terms can be expressed as their conjunction. However, it is often convenient to express them as a term pattern constrained by a predicate. One way to achieve this is to use existing unification algorithms to obtain the most general unifier and then encode it as a ML formula.

An interesting path to investigate is unification modulo equations in the context of ML. Also, unification of ML formulas that are not necessarily term patterns is a challenging task.

References

1. Franz Baader. The theory of idempotent semigroups is of unification type zero. *J. Autom. Reason.*, 2(3):283–286, August 1986.
2. Franois Fages. Associative-commutative unification. *Journal of Symbolic Computation*, 3(3):257 – 275, 1987.
3. Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, April 1982.
4. G. Plotkin. Building in equational theories. *Machine Intelligence*, 7:73–90, 1972. http://www.cs.york.ac.uk/mlg/MI/mi.htmlJournal Webpage.
5. Grigore Roşu. Matching logic. *Logical Methods in Computer Science*, 13(4):1–61, December 2017.
6. Mark E. Stickel. A unification algorithm for associative-commutative functions. *J. ACM*, 28:423–434, 1981.

Encoding Causality via Modal Formulae

Georgiana Caltais[1] and Mohammad Reza Mousavi[2]

[1] Department for Computer and Information Science, University of Konstanz
[2] Department of Informatics, University of Leicester

Abstract. This work introduces an encoding of causality for labelled transition systems and Hennessy Milner logic, in terms of modal formulae with data. The approach paves the way to the automatic identification of causalities using the mCRL2 model checker.

Introduction. Determining and computing causalities is a frequently addressed issue in the philosophy of science and engineering. A notion of causality that is frequently used in relation to technical systems relies on counterfactual reasoning [8]. In short, the counterfactual argument defines when an event is considered a cause for some effect, in the following way: a) whenever the event presumed to be a cause occurs, the effect occurs as well, and b) when the presumed cause does not occur, the effect will not occur either. The seminal papers [4, 5] describes an event model and a notion of actual causation encompassing the counterfactual argument. Most relevant for our work are the contributions in [7, 1]. The results in [7] provide an interpretation of the results in [4] in the context of transition systems and trace models for concurrent system computations. In [1] we adopt the aforementioned trace-based interpretation to the context of labelled transitions systems (LTS's) and Hennesy Milner logic (HML) [6] and devise a series of preliminary results on compositionality of causality.

The objective of this paper is to provide an encoding of causality as in [1] in terms of modal formulae with data [3], thus paving the way to the identification of causalities in an algorithmic fashion using the mCRL2 model-checker [2].

Preliminaries. Next, we provide a brief overview of LTS's and their computations, and HML. A *labeled transition system* (LTS) is a triple $T = (\mathbb{S}, s_0, A, \rightarrow)$, where $\mathbb{S}$ is the set of states, $s_0 \in \mathbb{S}$ is the initial state, A is the action alphabet and $\rightarrow \subseteq \mathbb{S} \times A \times \mathbb{S}$ is the transition relation. Let A^* be the set of words over A, and let ε be the empty word. We write $\twoheadrightarrow \subseteq \mathbb{S} \times A^* \times \mathbb{S}$, to denote the extension of $\rightarrow$ to words, defined recursively in the expected way: $s \xrightarrow{a}\!\!\!\!\!\rightarrow s'$ iff $s \xrightarrow{a} s'$, $s \xrightarrow{\varepsilon}\!\!\!\!\!\rightarrow s$, $s \xrightarrow{aw}\!\!\!\!\!\rightarrow s'$ iff $s \xrightarrow{a}\!\!\!\!\!\rightarrow s'$ and $s \xrightarrow{w}\!\!\!\!\!\rightarrow s'$, for $a \in A$ and $w \in A^*$.

Let $\mathcal{D}$, $\mathcal{D}_i$ range over possibly infinite lists of words in A^*. We say that two such lists are *size-compatible* if they are finite lists of the same length, or if they are all infinite lists.

Let $\pi = (s_0, l_0, \mathcal{D}_0), \ldots (s_n, l_n, \mathcal{D}_n), s_{n+1} \in (\mathbb{S} \times A \times [A^*])^* \times \mathbb{S}$. Assume that $\mathcal{D}_0, \ldots, \mathcal{D}_n$ are size-compatible. We write $traces(\pi)$ to denote the pairwise extensions of $l_0 \ldots l_n$ with words "at the same level" in $\mathcal{D}_0, \ldots, \mathcal{D}_n$. For instance, if $\pi = (s_0, l_0, [w_1^0, w_2^0]), (s_1, l_1, [w_1^1, w_2^1]), s_2$, then $traces(\pi) = \{l_0 w_1^0 l_1 w_1^1,\ l_0 w_2^0 l_1 w_2^1\}$. π is a *computation* of T whenever the following hold: (i) $s_0 \xrightarrow{l_0}\!\!\!\!\!\rightarrow s_1 \ldots \xrightarrow{l_n}\!\!\!\!\!\rightarrow s_{n+1}$,

(ii) $\mathcal{D}_0, \ldots, \mathcal{D}_n$ are size-compatible, and (iii) for all $w \in traces(\pi)$ there exists $s \in \mathbb{S}$ s.t. $s_0 \stackrel{w}{\twoheadrightarrow} s$. $sub(\pi)$ stands for the set of all computations $\pi' = (s_0, l'_0, \mathcal{D}'_0), \ldots, (s_m, l'_m, \mathcal{D}'_m), s'_{m+1}$ s.t. $l'_0 \ldots l'_m$ is a sub-word of $l_0 \ldots l_n$.

We consider formulae in *Hennessy-Milner logic* (HML) [6] given by the following grammar:

$$\phi, \psi ::= \top \mid \langle a \rangle \phi \mid [a]\phi \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \qquad (a \in A). \tag{1}$$

We say that an HML formula ϕ as above is *built over* A. The associated satisfaction relation $\models$ is defined in the standard way, over states $s \in \mathbb{S}$ and HML formulae. Intuitively, $s \models \langle a \rangle \phi$ states that s can execute a and reach a state that satisfies ϕ afterwards. Orthogonally, $s \models [a]\phi$ refers to the fact that no matter what state is reached from s by executing a, the reached state satisfies ϕ. $\top$ is the formula that holds in any state, whereas $\wedge, \vee$ and $\neg$ stand for conjunction, disjunction and negation.

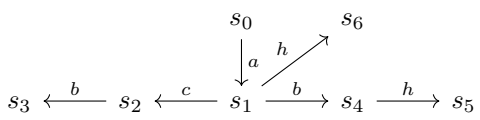
Defining causality. Our notion of causality is an adoption of the "actual causation" proposed in [4], previously adapted to the setting of concurrent systems in [7]. Consider an LTS $T = (\mathbb{S}, s_0, A, \rightarrow)$ and a "hazard" HML formula ϕ. A causal analysis of T w.r.t. ϕ is justified under the assumption that ϕ does not hold in all states of T, *i.e.*, T can display correct behaviour as well (item 2 below). A computation $\pi = (s_0, l_0, \mathcal{D}_0), \ldots, (s_n, l_n, \mathcal{D}_n), s_{n+1}$ is a *causal trace* if, intuitively:

- the execution of $l_0 \ldots l_n$ leads to a state satisfying the hazard (item 1 below)
- the occurrence of the actions $l_0, \ldots, l_n$ along a trace χ' in T guarantees that executing χ' leads to the hazard (item 3), as long as χ' does not encode elements in $\mathcal{D}_0, \ldots, \mathcal{D}_n$ which are causal by their non-occurrence (item 4). Non-occurrence is useful to explain situations in which the hazard holds if certain words in $\mathcal{D}_0, \ldots, \mathcal{D}_n$ are not executed, whereas executing these words removes the hazard
- π is the "shortest" computation satisfying the above properties (item 5)

Formally, *causal traces* in T w.r.t. ϕ, denoted by $Causes(\phi, T)$, is the set of all computations $\pi = (s_0, l_0, \mathcal{D}_0), \ldots, (s_n, l_n, \mathcal{D}_n), s_{n+1}$ s.t. :

1. $s_0 \stackrel{l_0}{\longrightarrow} \ldots s_n \stackrel{l_n}{\longrightarrow} s_{n+1} \wedge s_{n+1} \models \phi$ *(Positive causality)*
2. $\exists \chi \in A^*, s' \in \mathbb{S} : s_0 \stackrel{\chi}{\twoheadrightarrow} s' \wedge s' \models \neg\phi$ *(Counter-factual)*
3. $\forall \chi' = l_0\chi_0 \ldots l_n\chi_n \in \{l_0 \ldots l_n\} \cup (A^* \setminus traces(\pi))$, $s' \in \mathbb{S}$: $s_0 \stackrel{\chi'}{\twoheadrightarrow} s' \Rightarrow s' \models \phi$ *(Occurrence)*
4. $\forall \chi' \in traces(\pi) \setminus \{l_0 \ldots l_n\}$, $s' \in \mathbb{S} : s_0 \stackrel{\chi'}{\twoheadrightarrow} s' \Rightarrow s' \models \neg\phi$ *(Non-occurrence)*
5. $\forall \pi' \in sub(\pi) : \pi'$ *does not satisfy items 1. – 4. above (Minimality)*

Consider, for an example, the following LTS and the HML formula $\phi = \langle h \rangle \top$:



(2)

Item 1 suggests that action a should be a cause for the hazard ϕ. Item 2 indicates that ϕ does not hold trivially everywhere as, for instance, $s_0 \xrightarrow{acb} s_3$ and $s_3 \not\models \phi$. Item 4 states that $(s_0, a, [\varepsilon]), s_1$ is not a cause for ϕ because extending a with cb, for instance, violates ϕ and thereby violates item 3. However, $(s_0, a, [h, c, cb, bh]), s_1$ is a good candidate as all possible extensions of a with anything but h, c, cb or bh also keep the hazard, and thus satisfies items 3 and 4. Item 5 states that $(s_0, a, [\varepsilon, c]), (s_1, b, [h, \varepsilon]), s_4$ is not a cause as it is not minimal. This is because its sub-computation $(s_0, a, [h, c, cb, bh]), s_1$ satisfies items 1–4 as previously discussed.

Causality as modal formulae with data. In this section we introduce an attempt of encoding causality via modal formulae with data. This paves the way to the automatic identification of causes in mCRL2.

We proceed by first introducing modal formulae with data as in [3], used in order to model various real world phenomena. For space considerations, we only provide the fragment relevant for our work:

$$\begin{array}{l} R ::= a \mid \varepsilon \mid R \cdot R \mid R + R \mid R^* \mid R^+ \qquad (a \in A) \\ \phi ::= \mathit{true} \mid \mathit{false} \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \forall d : D.\phi \mid \exists d : D.\phi \mid \langle R \rangle \phi \mid [R] \\ \qquad \mu X(d_1 : D_1 := t_1, \ldots, d_n : D_n := t_n).\phi \mid \\ \qquad \mathcal{V} X(d_1 : D_1 := t_1, \ldots, d_n : D_n := t_n).\phi \mid X(t_1, \ldots, t_n) \end{array} \tag{3}$$

Formulae R are defined as regular expressions in the standard way. Formulae $\langle R \rangle$ and, respectively, $[R]$ extend the diamond $\langle - \rangle$ and, respectively, box $[-]$ modalities in (1) to words. Existential and universal quantifiers for ranging over data domains are also introduced. μ and, respectively, $\mathcal{V}$ stand for the minimal and, respectively, the maximal fixed point equations.

Assume a computation $\pi = (s_0, a_0, \mathcal{D}_0), \ldots, (s_n, a_n, \mathcal{D}_n), s_{n+1}$. We write l for the list $a_0 : \ldots : a_n : []$, and $\mathcal{LD}$ for the list of lists $\mathcal{D}_0 : \ldots : \mathcal{D}_n$. In order to check whether π is a cause w.r.t. a HML formula ϕ, we propose a straightforward encoding the corresponding items $1 - 5$ in terms of modal formulae with data as below.

We write A^* for the "type" of words of actions in A (*e.g.*, $l : A^*$), $[A^*]$ for the "type" of lists of words of actions in A (*e.g.*, $\mathcal{D}_0 : [A^*]$), $[[A^*]]$ for the "type" of lists of lists of words of actions in A (*e.g.*, $\mathcal{LD} : [[A^*]]$), $[l]\phi$ to denote the formula $[a_0 \cdot \ldots \cdot a_n]\phi$ (symmetrically for the diamond modality $\langle - \rangle$). We use the notations $\stackrel{\mu}{=}$ and, respectively, $\stackrel{\nu}{=}$ in order to represent minimal and, respectively, maximal fixed point equations. The encodings are:

$$PC(l : A^*, \mathcal{LD} : [[A^*]]) \stackrel{\mu}{=} \langle l \rangle \phi \qquad \text{(encoding Positive Causality)}$$

$$C(l : A^*, \mathcal{LD} : [[A^*]]) \stackrel{\mu}{=} \exists l' : A^*.\langle l' \rangle \neg\phi \qquad \text{(encoding Counter-factual)}$$

$$CON(l : A^*, \mathcal{LD} : [[A^*]], n : \mathbb{N}, k : \mathbb{N}, j : \mathbb{N}) \stackrel{\nu}{=}$$
$$\forall l_0 : A^*.\ \ldots\ .\forall l_n : A^*.\exists l' : A^*.\ (j == k) \vee$$
$$((j \neq k) \wedge (l' == zip(l, l_0 : \ldots : l_n)) \wedge (l' \neq zip(l, row(\mathcal{LD}, j)) \wedge [l']\phi \wedge$$
$$CON(l, \mathcal{LD}, n, k, j+1))) \vee \qquad (\clubsuit)$$
$$((j \neq k) \wedge (l' == zip(l, l_0 : \ldots : l_n)) \wedge (l' == zip(l, row(\mathcal{LD}, j)) \wedge [l']\neg\phi \wedge$$
$$CON(l, \mathcal{LD}, n, k, j+1))) \qquad (\spadesuit)$$

(encoding Causality of (non-)occurrence)

where $n+1$ is the size of l, and $k+1$ is the length of the size-compatible lists $\mathcal{D}_i$ in $\mathcal{LD}$. Additionally, j is an index used for iterating through the words at location j in each of the lists $\mathcal{D}_i$. The words at location j in all $\mathcal{D}_i$'s are given by the "row" j in $\mathcal{LD}$: $row(\mathcal{LD}, j)$. Variables l, n, k and j are examples of data tokens for formulae as in (3). Furthermore, $zip(l, l_0 : \ldots : l_n)$ denotes the pairwise extension of l with $l_0 : \ldots : l_n$ as expected: $a_0 : l_0 : \ldots : a_n : l_n$. Intuitively, $zip(l, l_0 : \ldots : l_n)$ corresponds to an element in $traces(\pi)$. Hence, the disjunct ($\clubsuit$) encodes *causality of occurrence*, whereas the disjunct ($\spadesuit$) encodes *causality of non-occurrence*.

$$M(l : A^*, \mathcal{LD} : [[A^*]]) \stackrel{\mu}{=}$$
$$\exists l' : A^*.\exists \mathcal{LD}' : [[A^*]].\ (subtrace(l', l) == true) \wedge$$
$$(\mid l' \mid +1 == \mid \mathcal{LD}' \mid) \wedge szCompatible(\mathcal{LD}') \wedge$$
$$PC(l', \mathcal{LD}') \wedge C(l', \mathcal{LD}') \wedge CON(l', \mathcal{LD}', \mid l' \mid, \mid \mathcal{LD}'[0] \mid, 0)$$

(encoding Minimality)

In the formula above, we write $subtrace(l', l) == true$ whenever $l' \in sub(l)$. Moreover, $szCompatible(\mathcal{LD}') == true$ whenever the elements of $\mathcal{LD}'$ are size-compatible lists. A non-empty solution w.r.t. $M(l, \mathcal{LD})$ denotes that π violates the minimality condition.

Let $\pi = (s_0, a, [h, c, cb, bh]), s_1$ be the causal computation of the LTS in (2), w.r.t. the HML formula $\langle h \rangle \top$. We fix $l = a$ and $\mathcal{LD} = [[h], [c], [cb], [bh]]$. It is straightforward to see that s_0 satisfies the formula in (encoding Positive Causality). It follows immediately that (encoding Counter-factual) holds in s_0 when $l' = acb$, for instance. Moreover, s_0 satisfies ($\spadesuit$) for all l' ranging over $\{ah, ac, acb, abh\}$, as $s_0 \stackrel{l'}{\twoheadrightarrow} s_i \Rightarrow s_i \not\models \langle h \rangle \top$. Symmetrically, s_0 satisfies ($\clubsuit$) for all remaining transitions l' ranging over $\{a, ab\}$ as $s_0 \stackrel{l'}{\twoheadrightarrow} s_i \Rightarrow s_i \models \langle h \rangle \top$. Similarly, it can be shown that (encoding Minimality) is not satisfied by any state in (2). Hence, the proposed modal formulae confirm π as being causal.

Conclusions. We provide an encoding of the causality for LTS's and HML in [1] in terms of modal formulae with data. This is the first step towards the implementation of an algorithm for computing such causalities. As future work we consider implementing the corresponding encodings in mCRL2 [2]. While the trace component $l = l_0, \ldots, l_n$ in the proposed definition of causality can be easily identified as a counterexamples violating the specification, one of the biggest challenges remains the automatic identification of the words in $\mathcal{D}_i$ causal by their non-occurrence as in item 4. Corresponding case studies and comparison with other approaches (*e.g.*, [7]) will be considered as well.

References

1. G. Caltais, S. Leue, and M. R. Mousavi. (De-)composing causality in labeled transition systems. In G. Gößler and O. Sokolsky, editors, *Proceedings First Workshop on Causal Reasoning for Embedded and safety-critical Systems Technologies, CREST@ETAPS 2016, Eindhoven, The Netherlands, 8th April 2016.*, volume 224 of *EPTCS*, pages 10–24, 2016.
2. S. Cranen, J. F. Groote, J. J. A. Keiren, F. P. M. Stappers, E. P. de Vink, W. Wesselink, and T. A. C. Willemse. An overview of the mCRL2 toolset and its recent advances. In N. Piterman and S. 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*, pages 199–213. Springer, 2013.
3. J. F. Groote and M. R. Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.
4. J. Halpern and J. Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *The British Journal for the Philosophy of Science*, 2005.
5. J. Y. Halpern. A modification of the Halpern-Pearl definition of causality. In Q. Yang and M. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3022–3033. AAAI Press, 2015.
6. M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherland, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer, 1980.
7. F. Leitner-Fischer and S. Leue. Causality checking for complex system models. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, volume 7737 of *Lecture Notes in Computer Science*, pages 248–267. Springer, 2013.
8. D. Lewis. *Counterfactuals*. Blackwell Publishers, 1973.

Continuation Semantics for Concurrent Languages

Gabriel Ciobanu[1] and Eneia Nicolae Todoran[2]

[1] Romanian Academy, Institute of Computer Science
and A.I.Cuza University of Iaşi, Romania
[2] Technical University of Cluj-Napoca
Department of Computer Science
gabriel@info.uaic.ro, Eneia.Todoran@cs.utcluj.ro

Abstract. We have introduced and studied the *continuation semantics for concurrency* (CSC), a technique which can be used as a general tool for designing denotational and operational semantics for concurrent programming languages. Most results related to CSC were obtained by using the mathematical methodology of metric semantics. The distinctive characteristic of the CSC technique is the modelling of continuations as application-specific structures rather than the functions to some answer type that are used in the classic technique of continuations. In this work we survey some applications of CSC, and present an optimality criterion that we name *weak abstractness*. The weak abstractness criterion preserves the correctness condition, but relaxes the completeness condition of the classic full abstractness criterion introduced by Robin Milner.

1 Introduction

Continuations constitute a classical tool in functional programming and denotational semantics. A continuation is a semantic representation of the remaining part of a computation [26]. This apparently simple concept has proved to be very useful in computer science. As John Reynolds pointed, "in the early history of continuations, basic concepts were independently discovered an extraordinary number of times ... due less to poor communication among computer scientists than to the rich variety of settings in which continuations were found useful" [23]. In this work we focus on the utility of continuations as a general tool for designing denotational and operational semantics for concurrent programming languages.

The usefulness of continuations as a tool for designing semantic models of programming languages is well-known. Traditional continuations [26] can express the semantics of sequential composition, and can be used to model a variety of advanced control concepts including non-local exits [26], coroutines [16], multitasking [29, 15] and notions encountered in parallel OO programming like process creation and rendez-vous between processes[24, 2]. However, the traditional continuations do not work well enough in the presence of concurrency [17].

In [27] and [11] we introduced a technique for denotational and operational semantic design that we named *continuation semantics for concurrency* (CSC);

it can be used to model both sequential and parallel composition in interleaving semantics while providing the general advantages of the technique of continuations [7]. The central characteristic of the CSC technique is the modelling of continuations as structured configurations of computations, where by computation we understand a partially evaluated denotation (meaning function). Intuitively, the CSC technique is a semantic formalization of a process scheduler simulated on a sequential machine. The space of computations is divided into one *active* computation and the rest of the computations which are encapsulated in a continuation. Each computation (or process) remains active only until it performs an elementary action; subsequently, another computation taken from the continuation is planned for execution. In this way it can be obtained the desired interleaving behaviour for parallel composition.

In the CSC approach a continuation is an application-specific structure of computations. The language designer can establish a simple relation between the structure of continuations and the control flow concepts of the language under investigation. Continuation structures can be designed by using two abstract concepts: the *stack* to model sequential composition, and the *multiset* to model parallel composition. In order to model a general combination of sequential and parallel composition, a continuation can be structured as a tree with active computations at its leaves. This continuation structure is inspired by the concept of a *cactus stack*, namely a stack with multiple tops that are active concurrently [4].

In this paper we survey some applications of CSC based on our previous work [27, 10–12, 14, 28], and present an optimality criterion specific to CSC that we call *weak abstractness* [13]. The semantic models presented in this paper (and in most of our previous works related to CSC) are designed by using the mathematical methodology of metric semantics [3].

2 Denotational Semantics designed with CSC

The continuation semantics for concurrency (CSC) approach was introduced in [27, 11] by using techniques from metric semantics [3]. It relies on the general method for solving reflexive domain equations presented in [1], as continuations in the CSC approach are elements of a complete metric space which is the solution of a domain equation where the domain variable occurs in the left-hand side of a function space construction. In [27], the CSC technique was applied in designing denotational and operational models for two concurrent languages: a CSP-like language [18] and the asynchronous formalism introduced in [5]. In [11] we presented a method of reasoning about the behaviour of concurrent programs in denotational models designed with CSC and metric spaces.

By using the CSC technique, we designed (continuation-based) denotational models for various advanced concurrent control concepts [28, 12, 14, 10]. In [28] we presented a denotational semantics for Warren's Basic Andorra Model [30]. Basic Andorra Model is a general model of AND/OR parallel logic programming, which combines AND parallelism, with 'don't care' nondeterminism and Prolog-like 'don't know' nondeterminism. In [12] we designed a continuation-based de-

notational semantics for a CSP-like language extended with communication and synchronization on multiple channels and we established the correctness of the denotational semantics with respect to a corresponding operational semantics. In [14, 10] we presented continuation-based denotational models for some core concepts of membrane computing [21] including maximal parallelism, membrane creation and dissolution, nondeterministic behaviour and trans-membrane communication. To the best of our knowledge, the advanced concurrent control concepts studied in [28, 12, 14, 10] have not been investigated denotationally without CSC until now.

3 Weak Abstractness

The full abstractness problem was raised by Milner in [19], and the difficulty of designing fully abstract semantics is well-known [19, 6]. The full abstractness problem seems to be even more difficult in continuation semantics. In [8, 25] it is noted that continuation-passing semantics for sequential languages are not fully abstract, essentially because they rely on continuous function spaces containing more functions than can be denoted by the program constructs. Moreover, several papers employ continuations in the denotational design of concurrent languages (for instance, [3, 24, 27, 12]). However, we are not aware of any full abstractness result for a concurrent language designed with continuations. Therefore, in [13] we introduced a new criterion that we named *weak abstractness*; it preserves the correctness condition, but relaxes the completeness condition of the classic full abstractness criterion. The weak abstractness condition is similar to the classic full abstractness condition; it is weaker only in the sense that it has to be verified merely for a class of *denotable continuations* [13]. Weak abstractness may be useful for a wide class of denotational models designed with continuations in which full abstractness is difficult (or impossible) to achieve. To illustrate the approach, in [13, 9] we presented a denotational semantics designed with CSC and metric spaces for the asynchronous formalism introduced in [5], and proved that the denotational semantics is weakly abstract with respect to a corresponding operational semantics. We believe that similar weak abstractness results can be obtained for other languages designed with CSC, including those surveyed in Section 2.

Full abstractness Let $(x \in)L$ be a language. A denotational semantics $\mathcal{D} : L \to \mathbf{D}$ is said to be *fully abstract* with respect to an operational semantics $\mathcal{O} : L \to \mathbf{O}$ if it is *correct* and *complete*. We only recall the completeness condition, where S is a typical element of a class of *syntactic contexts* for L. $\mathcal{D}$ is said to be *complete* with respect to $\mathcal{O}$ if the following condition holds:

$$\forall x_1, x_2 \in L[\mathcal{D}(x_1) \neq \mathcal{D}(x_2) \;\Rightarrow\; \exists S[\mathcal{O}(S(x_1)) \neq \mathcal{O}(S(x_2))]].$$

The weak abstractness criterion A denotational semantics $\mathcal{D} : L \to \mathbf{D}$ designed with continuations maps language constructs ($\in L$) to values in a mathematical domain $\mathbf{D} = \mathbf{Cont} \to \mathbf{F}$, where $(\gamma \in)\mathbf{Cont}$ is a domain of continuations

and **F** is some domain of final program answers. In continuation semantics, the *completeness condition* of full abstractness can be expressed as follows:
$\forall x_1, x_2 \in L[\,(\exists \gamma \in \mathbf{Cont}[\mathcal{D}(x_1)\gamma \neq \mathcal{D}(x_2)\gamma]) \Rightarrow (\exists S[\mathcal{O}(S(x_1)) \neq \mathcal{O}(S(x_2))])\,]$.
In the CSC approach, we cannot prove this condition for the whole domain of continuations. However the condition can be established if we restrict the investigation to a class of *denotable continuations*. The resulting condition is called *weak completeness*. We define this condition within the mathematical framework of complete metric spaces. Thus, the weak abstractness criterion is formulated by using a *class of denotable continuation*. Following [3], we use the term *resumption* as an operational counterpart of the term *continuation*. The formal definitions of these notions are provided in [13]. In the CSC approach, a *denotable continuation* is a semantic version of a *resumption*. A resumption is a configuration of program statements; a denotable continuation is a corresponding configuration of *denotations* of program statements. We also define a *domain of denotable continuations* as the *metric completion* of the *class of denotable continuations*.

Definition 31 *(Weak abstractness for CSC) Let* $(x \in)L$ *be a language and let* $\mathcal{D} : L \to \mathbf{D}$ *be a denotational semantics of* L *designed with CSC, where* $\mathbf{D} \cong \mathbf{\Gamma} \stackrel{1}{\to} \mathbf{F}$, $\mathbf{\Gamma}$ *is the* domain of continuations. *Let* $\mathcal{O} : L \to \mathbf{O}$ *be an operational semantics of* L *and* S *a typical element of the class of syntactic contexts for* L.

(a) $\mathcal{D}$ *is* correct *with respect to* $\mathcal{O}$ *iff*
$\forall x_1, x_2 \in L[\mathcal{D}(x_1) = \mathcal{D}(x_2) \Rightarrow \forall S[\mathcal{O}(S(x_1)) = \mathcal{O}(S(x_2))]]$.
(b) Let $\mathbf{\Gamma}^{\mathcal{D}}$ *be the* domain of denotable continuations *for* $\mathcal{D}$, *as defined in [13]. We say that* $\mathcal{D}$ *is* weakly complete *with respect to* $\mathcal{O}$ *iff*
$\forall x_1, x_2 \in L[(\exists \gamma \in \mathbf{\Gamma}^{\mathcal{D}}[\mathcal{D}(x_1)\gamma \neq \mathcal{D}(x_2)\gamma]) \Rightarrow (\exists S[\mathcal{O}(S(x_1)) \neq \mathcal{O}(S(x_2))])]$.
(c) We say that $\mathcal{D}$ *is* weakly abstract *with respect to* $\mathcal{O}$ *iff* $\mathcal{D}$ *is correct and weakly complete with respect to* $\mathcal{O}$.

Only the completeness condition makes the difference between full abstractness and weak abstractness. The condition is similar to the classic full abstractness condition; it is weaker only in the sense that it has to be verified merely for the domain of denotable continuations. According to [13, 9], it is enough to verify the (weak) completeness condition for the class of denotable continuations $\Gamma^{\mathcal{D}}$ which is a subspace of the *domain* of denotable continuations.

Formally, $\Gamma^{\mathcal{D}} \lhd \mathbf{\Gamma}^{\mathcal{D}}$ and $\mathbf{\Gamma}^{\mathcal{D}} \lhd \mathbf{\Gamma}$; however, in general, $\mathbf{\Gamma}^{\mathcal{D}} \neq \mathbf{\Gamma}$ (see [13, 9]).

4 Conclusion

Compared with the traditional approach to concurrency semantics [22, 3], the continuation semantics for concurrency discussed in this paper may provide a finer control. We provide some denotational models designed with continuations for various advanced concurrent control concepts that have not been investigated without continuations before. We present an abstractness criterion specific to CSC that preserves the correctness condition and relaxes the completeness condition of the full abstractness criterion introduced by Milner.

References

1. P. America and J.J.M.M. Rutten. Solving Reflexive Domain Equations in a Category of Complete Metric Spaces, *Journal of Computer and System Sciences*, vol.39(3), 343–375, 1989.
2. J.W. de Bakker, E.P. de Vink. CCS for OO and LP, *Lecture Notes in Computer Science*, vol.494, 1–28, 1991.
3. J.W. de Bakker, E.P. de Vink. *Control Flow Semantics,* MIT Press, 1996.
4. D.G. Bobrow, B. Wegbreit. A Model and Stack Implementation of Multiple Environments, *Communications of the ACM*, vol.16(10), 591–603, 1973.
5. F.S. De Boer, J.N. Kok, C. Palamidessi, J.J.M.M. Rutten. "A paradigm for asynchronous communication and its application to concurrent constraint programming," In *Logic Programming Languages: Constraints, Functions and Objects*, 82–114, MIT Press, 1993.
6. S.D. Brookes. Full Abstraction for a Shared-Variable Parallel Language, *Information and Computation*, vol.127(2), 145–163, 1996.
7. A. de Bruijn. Experiments with Continuation Semantics: Jumps, Backtracking, Dynamic Networks, Ph.D. Thesis, Free University, Amsterdam, 1986.
8. R. Cartwright, P.-L. Curien, M. Felleisen. Fully Abstract Semantics for Observably Sequential Languages, *Information and Computation*, vol.111(2), 297–401, 1994.
9. G. Ciobanu, E.N. Todoran. Abstract Continuation Semantics for Asynchronous Concurrency. Technical Report FML-12-02, 2012. Available at `http://iit.iit.tuiasi.ro/TR/reports/fml1202.pdf`
10. G. Ciobanu, E.N. Todoran. Relating two metric semantics for parallel rewriting of multisets, *Proceedings SYNASC*, 273–280, IEEE Computer Press, 2012.
11. G. Ciobanu, E.N. Todoran. Continuation Semantics for Asynchronous Concurrency, *Fundamenta Informaticae*, vol.131, 373–388, 2014.
12. G. Ciobanu, E.N. Todoran. Continuation Semantics for Concurrency with Multiple Channels Communication, *Lecture Notes in Computer Science*, vol.9407, 400–416, 2015.
13. G. Ciobanu, E.N. Todoran. Abstract Continuation Semantics for Asynchronous Concurrency, *Proceedings SYNASC*, IEEE Computer Press, 2018 (to appear).
14. G. Ciobanu, E.N. Todoran. Denotational Semantics of Membrane Systems by using Complete Metric Spaces, *Theoretical Computer Science*, vol.701, 85–108, 2017.
15. R.K. Dybvig, R. Hieb. Engines from Continuations, *Computer Languages*, vol.14(2), 109–123, 1989.
16. D.P. Friedman, C. T. Haynes, M. Wand. Obtaining Coroutines with Continuations, *Computer Languages*, vol.11(3/4), 143–153, 1986.
17. R. Hieb, R. K. Dybvig, C.W. Anderson. Subcontinuations, *Lisp and Symbolic Computation*, vol.7(1), 83–110, 1994.
18. C.A.R. Hoare. *Communicating Sequential Processes*, Prentice Hall, 1985.
19. R. Milner. Fully Abstract Models of Typed λ-Calculi, *Theoretical Computer Science*, vol.4, 1–22, 1977.
20. P.D. Mosses. Programming Language Description Languages: From Christopher Strachey to Semantics Online, *Formal Methods: State of the Art and New Directions*, 249–273, Springer, 2010.
21. Gh. Paun. Membrane Computing. An Introduction. Springer, 2002.
22. G. Plotkin. A Powerdomain Construction, *SIAM Journal of Computing*, vol.5(3), 452-487, 1976.

23. J.C. Reynolds, The Discoveries of Continuations, *LISP and Symbolic Computation*, vol.6, 233–247, 1993.
24. J.J.M.M. Rutten. Semantic correctness for a parallel object-oriented language, *SIAM Journal of Computing*, vol.19(2), 341–383, 1990.
25. D. Sitaram, M. Felleisen. Reasoning with Continuations II: Full Abstraction for Models of Control, *Proc. ACM Conference on LISP and Functional Programming*, 161–175, 1990.
26. C. Strachey, C.P. Wadsworth. Continuations: A Mathematical Semantics for Handling Full Jumps, *Higher-Order and Symbolic Computation*, vol.13, 135–152, 2000.
27. E.N. Todoran. Metric Semantics for Synchronous and Asynchronous Communication: A Continuation-based Approach, *Electronic Notes in Theoretical Computer Science*, vol.28, 101–127, 2000.
28. E.N. Todoran, N. Papaspyrou. Continuations for Parallel Logic Programming, *Proceedings PPDP*, 257–267, ACM Press, 2000.
29. M. Wand. Continuation-Based Multiprocessing, *Higher-Order and Symbolic Computation*, vol.12(3), 285–299, 1999.
30. D.H.D. Warren. The Andorra Principle, Talk given at the Gigalips Workshop, SICS, Stockholm, 1988.

Bisimulations in many-valued modal logics

Denisa Diaconescu

Faculty of Mathematics and Computer Science
University of Bucharest
ddiaconescu@fmi.unibuc.ro

Modal logics are logics enriched with specific connectives (called *modalities*) capable of expressing and reasoning with statements containing the concepts of possibility and necessity [6]. Specialised modalities can capture various additional notions, e.g., temporal modalities can capture the idea of time, epistemic modalities the notion of knowledge, doxastic modalities the notion of belief [4]. Kripke semantics (or possible worlds semantics, relational semantics) is the standard semantics for modal logics: it consists of a set of worlds where each world is governed by the laws of propositional classical logic, the information about which facts are true at each world, and an accessibility relation between worlds. Since these kind of *transition systems* are frequent in computer science, modal logics prove themselves a valuable tool for studying various problems in computer science. For example, modal logics are widely used in program verification, distributed systems, and artificial intelligence [4, 6, 16, 19].

Many-valued modal logics bring one extra layer of expressivity to the table, allowing us to reason, for example, in the presence of vagueness or imprecission. Kripke semantics for many-valued modal logics is defined as in classical modal logic, except that now each world in governed by a many-valued logic (i.e., a logic with more than two possible truth values). We can consider frameworks in which the worlds are governed by different logics, and frameworks in which all worlds are governed by the same many-valued logic. Moreover, the accessibility relation between worlds can be just a binary relation as in the classical case (we call this case *crisp*) or can have a certain weight (we call this case *many-valued*). Such many-valued modal logics span fuzzy belief [12,15], fuzzy description logics [1,14, 25], many-valued tense logics [10], and spatial reasoning with vague predicates [24].

Bisimulation is a fundamental concept of equivalence between structures introduced in computer science by Park [23] and Hennessy and Milner [17], and independently in the context of model theory of modal logic by Van Benthem [3]. Bisimulation is intended to characterise those worlds in Kripke models with the same behaviour. The concept of bisimultation has applications, for example, in automata theory [20], verification of finite-state systems [9], and investigation of concurrent processes [18]. A model-theoretic viewpoint of bisimultation investigates the preservation of the truth of a modal formula under a class of structures. An important result in this direction is Van Benthem's theorem which shows that

the minimal modal logic K (i.e., the modal logics for which the accessibility relations in the Kripke semantics are just binary relations) may be viewed as the bisimulation-invariant fragment of first-order logic [3].

In the work reported here we investigate the notion of bisimulation in the framework of many-valued modal logics. We study bisimulations for many-valued modal logics with many-valued accessibility relations, generalising the results from [21,22] where the crisp case is treated. Our main result is a Hennessy-Milner property for the class of image-finite models. Expressivity of many-valued modal logics (with both crisp and many-valued accessibility relations) was previously studied via coalgebraic methods in [5]. Our results engage direct algebraic methods for investigating the expressivity of many-valued logics and therefore provide, in our opinion, insight on the characteristics of many-valued modal logics. Fuzzy bisimulations were previously engaged, e.g., for the study of quantitative systems where notions of behavioural distance are more adequate than two-valued bisimilarity [8], weighted automata [2], and social network analysis [11].

We deal with many-valued modal logics defined over complete MTL-chains. Formally, an *MTL-algebra* is an algebraic structures of the form $\mathbf{A} = \langle A, \wedge, \vee, \odot, \rightarrow, 0, 1\rangle$ such that

1. $\langle A, \wedge, \vee, 0, 1\rangle$ is a bounded lattice with an order defined by $a \leq b \Leftrightarrow a \wedge b = a$;
2. $\langle A, \odot, 1\rangle$ is a commutative monoid;
3. $\rightarrow$ is the residuum of $\odot$ (i.e., $a \odot b \leq c \Leftrightarrow a \leq b \rightarrow c$ for all $a, b, c \in A$);
4. the *prelinearity condition* holds (i.e., $(a \rightarrow b) \vee (b \rightarrow a) = 1$ for all $a, b \in A$).

If $\mathbf{A}$ is *complete* (i.e., $\bigwedge X$ and $\bigvee X$ exist in A for all $X \subseteq A$) and a *chain* (i.e., $\leq$ is a linear order on A) then is called a complete *MTL-chain*. MTL-algebras span the algebraic semantics of most notable fuzzy logics, e.g., Łukasiewicz logic, Gödel logic, and Product logic [13].

For any complete MTL-chain $\mathbf{A}$, we define a many-valued modal logic $\mathrm{K}(\mathbf{A})$ via its Kripke semantics following [7]. The modal logic $\mathrm{K}(\mathbf{A})$ has binary connectives $\wedge, \vee, \odot$, and $\rightarrow$, constants $\bar{0}, \bar{1}$, and unary modal connectives $\Box$ and $\Diamond$. A (many-valued) $\mathbf{A}$-*frame* is a pair $\mathfrak{F} = \langle W, R\rangle$ where W is a non-empty set of *worlds* and $R : W \times W \rightarrow A$ is an *($\mathbf{A}$-valued) accessibility relation*. For convenience, we write Rxy instead of $R(x, y)$. If $Rxy \in \{0, 1\}$ for any $x, y \in W$, R and $\mathfrak{F}$ are both called *crisp*. For any $w \in W$, let $R[x] = \{y \in W : Rxy > 0\}$.

A *(many-valued)* $\mathrm{K}(\mathbf{A})$*-model* $\mathfrak{M} = \langle W, R, V\rangle$ consists of an $\mathbf{A}$-frame $\langle W, R\rangle$ together with a *valuation* map for variables $V : \mathrm{Var} \times W \rightarrow A$ that is extended to formulas $V : \mathrm{Fm} \times W \rightarrow A$ by $V(\bar{0}, w) = 0$, $V(\bar{1}, w) = 1$, $V(\varphi \circ \psi, w) = V(\varphi, w) \circ V(\psi, w)$ for $\circ \in \{\wedge, \vee, \odot, \rightarrow\}$, and

$$V(\Box\varphi, w) = \bigwedge_{v \in R[w]} Rwv \rightarrow V(\varphi, v),$$

$$V(\Diamond\varphi, w) = \bigvee_{v \in R[w]} Rwv \odot V(\varphi, v).$$

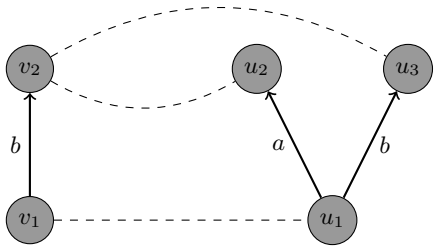


Fig. 1. Example of bisimilar models

Let us consider two K(**A**)-models $\mathfrak{M} = \langle W, R, V\rangle$ and $\mathfrak{M}' = \langle W', R', V'\rangle$. We say that $w \in W$ and $w' \in W'$ are *modally equivalent*, written $w \sim w'$, if $V(\varphi, w) = V'(\varphi, w')$ for all formulas $\varphi \in \mathrm{Fm}$. Furthermore, a non-empty binary relation $Z \subseteq W \times W'$ is called a *bisimulation* between $\mathfrak{M}$ and $\mathfrak{M}'$ if the following conditions are satisfied:

1. If wZw', then $V(p, w) = V'(p, w')$ for all $p \in \mathrm{Var}$.
2. If wZw' and $v \in R[w]$, then there exists $v' \in W'$ such that vZv' and $R'w'v' \geq Rwv$ *(the forth condition).*
3. If wZw' and $v' \in R'[w']$, then there exists $v \in W$ such that vZv' and $Rwv \geq R'w'v'$ *(the back condition).*

This notion of bisimulation is in agreement with the one proposed via coalgebraic methods in [5]; moreover, it coincides with the one defined in [21] for crisp models. We say that $w \in W'$ and $w \in W'$ are *bisimilar*, written $w \equiv w'$, if there exists a bisimulation Z between $\mathfrak{M}$ and $\mathfrak{M}'$ such that wZw'.

Example 1. Let **A** be a complete MTL-chain and $a, b \in A$ with $0 < a < b$. Consider the K(**A**)-models $\mathfrak{M} = \langle W, R, V\rangle$ and $\mathfrak{M}' = \langle W', R', V'\rangle$ displayed in Fig. 1 with $W = \{v_1, v_2\}$, $W' = \{u_1, u_2, u_3\}$, $Rv_1v_2 = b$, $R'u_1u_2 = a$, $R'u_1u_3 = b$, and V and V' making every propositional variable 1 in any world in W and W'. It is easy to check that $Z = \{(v_1, u_1), (v_2, u_2), (v_2, u_3)\}$ is a bisimulation between $\mathfrak{M}$ and $\mathfrak{M}'$.

We can easily show that bisimilarity implies modal equivalence in a similar way with the classical case (see, e.g. [6]). However, even in the classical case, modal equivalence does not imply, in general, the existence of a bisimulation between models. The Hennessy-Milner property captures exactly this phenomenon: we say that a class $\mathcal{K}$ of K(**A**)-models has the *Hennessy-Milner property* if for any models $\mathfrak{M} = \langle W, R, V\rangle$ and $\mathfrak{M}' = \langle W', R', V'\rangle$ in $\mathcal{K}$, if two states $w \in W$ and $w' \in W'$ are modally equivalent, then they are also bisimilar.

Our main result gives a necessary and sufficient algebraic condition on **A** such that the class of image-finite K(**A**)-models (i.e, models in which each world has finitely many successors) to admit the Hennessy-Milner property. Our algebraic characterisation is a generalisation of the one obtained in [21,22] for crisp models.

Let $n, m > 1$ be natural numbers and let $b \in A$, $\mathbf{d} \in A^m$, $\mathbf{a} \in A^n$, and $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_n) \in A^{n \times n}$. An MTL (propositional) formula $\psi(p_1, \ldots, p_n)$ is called $(b/\mathbf{d}, \mathbf{a}/\mathbf{C})$-*distinguishing formula* if

- either $b \to \psi[\mathbf{a}] < \psi[\mathbf{c}_i]$ and $b \to \psi[\mathbf{a}] < d_j \to \psi[\mathbf{a}]$ for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$,
- or $\psi[\mathbf{c}_i] < b \odot \psi[\mathbf{a}]$ and $d_j \odot \psi[\mathbf{a}] < b \odot \psi[\mathbf{a}]$ for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$.

We say that $\mathbf{A}$ has the *distinguishing formula property* if for all $n, m > 1$ natural numbers, $b \in A$, $\mathbf{d} \in A^m$ such that $b \neq 0$ and $d_j < b$ for all $j \in \{1, \ldots, m\}$, $\mathbf{a} \in A^n$, and $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_n) \in A^{n \times n}$ such that $\mathbf{a} \neq \mathbf{c}_i$ for all $i \in \{1, \ldots, n\}$, there exists an $(b/\mathbf{d}, \mathbf{a}/\mathbf{C})$-distinguishing formula. Note that for $b = 1$ and $d_j = 0$ for any $j = \{1, \ldots, m\}$, we obtain the notion of distinguishing formula property introduced in [21]. Now we can state our main result:

Theorem 1. *The following are equivalent for any complete MTL-chain* $\mathbf{A}$*:*

1. *The class of image-finite* $\mathrm{K}(\mathbf{A})$*-models has the Hennessy-Milner property.*
2. $\mathbf{A}$ *has the distinguishing formula property.*

Using McNaughton's theorem, we are able to apply our main result to the class of image-finite models of Łukasiewicz modal logic.

Corollary 1. *Let* $\mathbf{A}$ *be a finite or standard MV-chain. Then the class of image-finite* $\mathrm{K}(\mathbf{A})$*-models has the Hennessy-Milner property.*

References

1. Baader, F., Borgwardt, S., Peñaloza, R.: Decidability and complexity of fuzzy description logics. KI 31(1), 85–90 (2017)
2. Balle, B., Gourdeau, P., Panangaden, P.: Bisimulation metrics for weighted automata. In: ICALP 2017. vol. 103, pp. 1–14
3. van Benthem, J.: Correspondence theory. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. II. Reidel (1984)
4. van Benthem, J.: Modal logic for Open Minds. CSLI Lecture Notes 199 (2010)
5. Bílková, M., Dostál, M.: Expressivity of many-valued modal logics, coalgebraically. In: WoLLIC. pp. 109–124 (2016)
6. Blackburn, P., de Rijke, M., Venema, Y.: Modal logic. Cambridge University Press (2001)
7. Bou, F., Esteva, F., Godo, L., Rodríguez, R.: On the minimum many-valued logic over a finite residuated lattice. Journal of Logic and Computation 21(5), 739–790 (2011)
8. van Breugel, F., Worrell, J.: A behavioural pseudometric for probabilistic transition systems. Theoretical Computer Science 331, 115–142 (2005)
9. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000)
10. Diaconescu, D., Georgescu, G.: Tense operators on MV-algebras and Łukasiewicz-Moisil algebras. Fundamenta Informaticae 81(4), 379–408 (2007)

11. Fan, T., Liau, C.: Logical characterization of regular equivalence in weighted social networks. Artificial Intelligence 214, 66–88 (2014)
12. Godo, L., Hájek, P., Esteva, F.: A fuzzy modal logic for belief functions. Fundamenta Informaticae 57(2–4), 127–146 (2003)
13. Hájek, P.: Metamathematics of Fuzzy Logic. Kluwer, Dordrecht (1998)
14. Hájek, P.: Making fuzzy description logic more general. Fuzzy Sets and Systems 154(1), 1–15 (2005)
15. Hájek, P., Harmancová, D., Verbrugge, R.: A qualitative fuzzy possibilistic logic. International Journal of Approximate Reasoning 12, 1–19 (1995)
16. Halpern, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. Journal of the ACM 37(3), 549–587 (1990)
17. Hennessy, M., Milner, R.: On observing nondeterminism and concurrency. Lecture Notes in Computer Science 85, 295–309 (1980)
18. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. Journal of the ACM 32, 137–162 (1985)
19. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning About Systems. Cambridge University Press (2004)
20. Kozen, D.: Automata and Computability. Springer (1997)
21. Marti, M., Metcalfe, G.: Hennessy-Milner properties for many-valued modal logics. In: Proceedings of AiML 2014. pp. 407–420. King's College Publications (2014)
22. Metcalfe, G., Marti, M.: Expressivity in chain-based modal logics. Archive for Mathematical Logic (2017)
23. Park, D.: Concurrency and automata on infinite sequences. Lecture Notes in Computer Science 154, 561–572 (1981)
24. Schockaert, S., Cock, M.D., Kerre, E.: Spatial reasoning in a fuzzy region connection calculus. Artificial Intelligence 173(2), 258–298 (2009)
25. Straccia, U.: Reasoning within fuzzy description logics. Journal of Artificial Intelligence Research 14, 137–166 (2001)

A many-sorted polyadic modal logic

Ioana Leuştean and Natalia Moangă

Faculty of Mathematics and Computer Science,
University of Bucharest, Bucharest, Romania

[1] ioana@fmi.unibuc.ro
[2] natalia.moanga@drd.unibuc.ro

Abstract

In [3], we defined a many-sorted polyadic modal logic together with its corresponding algebraic theory. While the transition from the mono-sorted logic to many-sorted one is a smooth process, we see our system as a step towards deepening the connection between modal logic and program verification, since our system can be seen as the propositional fragment of Matching logic [4], a first-order logic for specifying and reasoning about programs. To our knowledge, the general framework presented in [3] is new, but we refer to [6, 8, 2, 5] for previous developments on many-sorted modal logic.

Our language is determined by a fixed, but arbitrary, many-sorted signature (S, Σ) and a set of many-sorted propositional variables $P = \{P_s\}_{s\in S}$ such that $P_s \neq \emptyset$ for any $s \in S$ and $P_{s_1} \cap P_{s_2} = \emptyset$ for any $s_1 \neq s_2$ in S. For any $n \in \mathbb{N}$ and $s, s_1, \ldots, s_n \in S$ we denote $\Sigma_{s_1\ldots s_n,s} = \{\sigma \in \Sigma \mid \sigma : s_1 \ldots s_n \to s\}$.

The transition from a mono-sorted to a many-sorted setting is a smooth process and we follow closely the developments from [1].

The set of formulas of $\mathcal{ML}_S$ is an S-indexed family $Form_S = \{Form_s\}_{s\in S}$ inductively defined as:

$$\phi_s ::= p \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid \sigma(\phi_{s_1}, \ldots, \phi_{s_n})$$

where $s \in S$, $p \in P_s$ and $\sigma \in \Sigma_{s_1\ldots s_n,s}$

We use the classical definitions of the derived logical connectors. For any $\sigma \in \Sigma_{s_1\ldots s_n,s}$ the *dual operation* is $\sigma^{\Box}(\phi_1, \ldots, \phi_n) := \neg\sigma(\neg\phi_1, \ldots, \neg\phi_n)$.

In order to define the semantics we introduce the (S, Σ)-*frames* and the (S, Σ)-*models*. An (S, Σ)-*frame* is a tuple $\mathcal{F} = (\mathcal{W}, \mathcal{R})$ such that:

- $\mathcal{W} = \{W_s\}_{s\in S}$ is an S-sorted set of worlds and $W_s \neq \emptyset$ for any $s \in S$,
- $\mathcal{R} = \{\mathcal{R}_\sigma\}_{\sigma\in\Sigma}$ such that $\mathcal{R}_\sigma \subseteq W_s \times W_{s_1} \times \ldots \times W_{s_n}$ for any $\sigma \in \Sigma_{s_1\ldots s_n,s}$.

If we consider $\mathcal{F}$ an (S, Σ)-frame, then an (S, Σ)-*model based on* $\mathcal{F}$ is a pair $\mathcal{M} = (\mathcal{F}, \rho)$ where $\rho : P \to \mathcal{P}(\mathcal{W})$ is an S-*sorted valuation function* such that $\rho_s : P_s \to \mathcal{P}(W_s)$ for any $s \in S$. The model $\mathcal{M} = (\mathcal{F}, \rho)$ will be simply denoted as $\mathcal{M} = (\mathcal{W}, \mathcal{R}, \rho)$. Following [1], if C is a set of frames then we say that a model $\mathcal{M}$ *is from* C if it is based on a frame from C.

Next we introduce a many-sorted *satisfaction relation*. If $\mathcal{M} = (\mathcal{W}, \mathcal{R}, \rho)$ is an (S, Σ)-model, $s \in S$, $w \in W_s$ and $\phi \in Form_s$, then the many-sorted *satisfaction relation* $\mathcal{M}, w \models^s \phi$ is inductively defined:

– $\mathcal{M}, w \models^s p$ iff $w \in \rho_s(p)$
– $\mathcal{M}, w \models^s \neg\psi$ iff $\mathcal{M}, w \not\models^s \psi$
– $\mathcal{M}, w \models^s \psi_1 \vee \psi_2$ iff $\mathcal{M}, w \models^s \psi_1$ or $\mathcal{M}, w \models^s \psi_2$
– if $\sigma \in \Sigma_{s_1 \dots s_n, s}$, then $\mathcal{M}, w \models^s \sigma(\phi_1, \dots, \phi_n)$ iff there exists $(w_1, \dots, w_n) \in W_{s_1} \times \dots \times W_{s_n}$ such that $\mathcal{R}_\sigma w w_1 \dots w_n$ and $\mathcal{M}, w_i \models^{s_i} \phi_i$ for any $i \in [n]$.

Consequently, we define our logic. Let $\mathbf{K}_{(S,\Sigma)} = \{\mathbf{K}_s\}_{s \in S}$ be the least S-sorted set of formulas with the following properties:

(a0) for any $s \in S$, if $\alpha \in Form_s$ is a theorem in classical logic, then $\alpha \in \mathbf{K}_s$,
(a1) the following formulas are in $\mathbf{K}_s$
(K^i_σ) $\sigma^\Box(\psi_1, \dots, \phi \to \chi, \dots, \psi_n) \to (\sigma^\Box(\psi_1, \dots, \phi, \dots, \psi_n) \to \sigma^\Box(\psi_1, \dots, \chi, \dots, \psi_n))$
$(Dual_\sigma)$ $\sigma(\psi_1, \dots, \psi_n) \leftrightarrow \neg\sigma^\Box(\neg\psi_1, \dots, \neg\psi_n)$
for any $n \geq 1$, $i \in [n]$, $\sigma \in \Sigma_{s_1 \dots s_n, s}$, ψ_{s_1}, …, $\psi_{s_n} \in Form_{s_n}$ and $\phi, \chi \in Form_{s_i}$.

It is straightforward that $\mathbf{K}_{(S,\Sigma)}$ is a generalization of the modal system $\mathbf{K}$ (see [1] for the mono-sorted version). If $\Lambda \subseteq Form_S$ is an S-sorted set of formulas, then the *normal modal logic* defined by Λ is $\mathbf{K}\Lambda = \{\mathbf{K}\Lambda_s\}_{s \in S}$ where

$$\mathbf{K}\Lambda_s := \mathbf{K}_s \cup \{\lambda' \in Form_s \mid \lambda' \text{ is an instance of a formula } \lambda \in \Lambda_s\}$$

Next, we assume $\Lambda \subseteq Form_s$ is an S-sorted set of formulas and we investigate the normal modal logic $\mathbf{K}\Lambda$.

In our approach, a logic is defined by its axioms. The deduction rules are *Modus Ponens* and *Universal Generalization*.

The distinction between local and global deduction from the mono-sorted setting is deepened in our version: *locally*, the conclusion and the hypotheses have the same sort, while *globally*, the set of hypotheses is a many-sorted set.

Definition 1. Assume that $n \geq 1$, $s_1, \dots, s_n \in S$ and $\phi_i \in Form_{s_i}$ for any $i \in [n]$. The sequence $\phi_1, \dots, \phi_n$ is a $\mathbf{K}\Lambda$*-proof for* ϕ_n if, for any $i \in [n]$, φ_i is in $\mathbf{K}\Lambda_{s_i}$ or φ_i is inferred from $\varphi_1, \dots, \varphi_{i-1}$ using *modus ponens* and *universal generalization*. If ϕ has a proof in $\mathbf{K}\Lambda$, then we say that ϕ is a *theorem* and we write $\vdash^s_{\mathbf{K}\Lambda} \phi$ where s is the sort of ϕ.

If $s \in S$, $\Phi_s \subseteq Form_s$ and $\phi \in Form_s$, then we say that ϕ is *locally provable from* Φ_s *in* $\mathbf{K}\Lambda$ and we write $\Phi_s \vdash^s_{\mathbf{K}\Lambda} \phi$, if there are $\phi_1, \dots, \phi_n \in \Phi_s$ such that $\vdash^s_{\mathbf{K}\Lambda} (\phi_1 \wedge \dots \wedge \phi_n) \to \phi$.

If $\Gamma \subseteq Form$ is an S-sorted set of formulas, we say that ϕ is a *globally provable* from Γ, and we write $\Gamma \vdash_{\mathbf{K}\Lambda} \phi$, if there exists a sequence $\phi_1, \dots, \phi_n$ such that $\phi_n = \phi$ and, for any $i \in [n]$, $\phi_i \in Form_{s_i}$ is an axiom or $\phi_i \in \Gamma_{s_i}$ or it is inferred from $\phi_1, \dots, \phi_{i-1}$ using *modus ponens* and *universal generalization*.

We further present the *local deduction*.

Theorem 1. *(Local deduction theorem for $\mathbf{K}\Lambda$) For any $s \in S$ and $\Phi_s \cup \{\varphi, \psi\} \subseteq Form_s$:*

$$\Phi_s \vdash^{s}_{\mathbf{K}\Lambda} \varphi \to \psi \text{ iff } \Phi_s \cup \{\varphi\}_s \vdash^{s}_{\mathbf{K}\Lambda} \psi.$$

Following closely the approach from [1], in order to study the canonical models and to prove the completeness theorem, we need to study the consistent sets. For any $s \in S$, we say that the set $\Phi_s \subseteq Form_s$ is (*locally*) $\mathbf{K}\Lambda$-*inconsistent* if $\Phi_s \vdash^{s}_{\mathbf{K}\Lambda} \bot_s$ and it is (*locally*) $\mathbf{K}\Lambda$-*consistent*, otherwise. Using the (locally) maximal consistent sets, we define the *canonical model* $\mathcal{M}_{\mathbf{K}\Lambda} = (\mathcal{W}^{\mathbf{K}\Lambda}, \{\mathcal{R}^{\mathbf{K}\Lambda}_{\sigma}\}_{\sigma \in \Sigma}, V^{\mathbf{K}\Lambda})$ as follows:

(1) for any $s \in S$, $\mathcal{W}^{\mathbf{K}\Lambda}_s = \{\Phi \subseteq Form_s \mid \Phi \text{ is maximal } \mathbf{K}\Lambda\text{-consistent}\}$,
(2) for any $\sigma \in \Sigma_{s_1 \ldots s_n, s}, w \in W^{\mathbf{K}\Lambda}_s, u_1 \in W^{\mathbf{K}\Lambda}_{s_1}, \ldots, u_n \in W^{\mathbf{K}\Lambda}_{s_n}$ we define

$$\mathcal{R}^{\mathbf{K}\Lambda}_{\sigma} w u_1 \ldots u_n \text{ iff } (\psi_1, \ldots, \psi_n) \in u_1 \times \cdots \times u_n \text{ implies } \sigma(\psi_1, \ldots, \psi_n) \in w$$

(3) $V^{\mathbf{K}\Lambda} = \{V^{\mathbf{K}\Lambda}_s\}_{s \in S}$ is the valuation defined by

$$V^{\mathbf{K}\Lambda}_s(p) = \{w \in W^{\mathbf{K}\Lambda}_s | p \in w\} \text{ for any } s \in S \text{ and } p \in P_s.$$

Our completeness results generalize the ones from [1].

Theorem 2. *For any many-sorted signature (S, Σ) the normal modal logic $\mathbf{K}_{(S,\Sigma)}$ is strongly complete with respect to the class of all (S, Σ)-frames. For any $\Lambda \subseteq Form$ the normal modal logic $\mathbf{K}\Lambda$ is complete with respect to the canonical model.*

We also investigate the *global deduction*: we relate the local and the global deduction, we prove a version of the deduction theorem in the global setting, we define the *globally consistent* sets and we prove that any globally consistent set has a model.

In order to develop an algebraic approach, we generalize the boolean algebras with operators as follows.

Definition 2. An (S, Σ)-*boolean algebra with operators* ((S, Σ)-*BAO*) is a structure

$$\mathfrak{A} = (\{\mathcal{A}_s\}_{s \in S}, \{f_\sigma\}_{\sigma \in \Sigma})$$

where $\mathcal{A}_s = (A_s, \vee_s, \neg_s, 0_s)$ is a boolean algebra for any sort $s \in S$ and, for any $\sigma \in \Sigma_{s_1 \ldots s_n, s}$, $f_\sigma : A_{s_1} \times \ldots \times A_{s_n} \to A_s$ satisfies the following properties:
(N) $f_\sigma(a_1, \ldots, a_n) = 0_s$ whenever $a_i = 0_{s_i}$ for some $i \in [n]$,
(A) $f_\sigma(a_1, \ldots, a_i \vee_{s_i} a'_i, \ldots, a_n) = f_\sigma(a_1, \ldots, a_i, \ldots, a_n) \vee_s f_\sigma(a_1, \ldots, a'_i, \ldots, a_n)$ for any $i \in [n]$.

We mention that similar structures were defined in [2], but in that case the operators are unary operations while, in our setting, they have arbitrary arities. We prove the analogue of Jónsson-Tarski theorem as well as the algebraic completeness for our systems of many-sorted modal logic.

Theorem 3. *(Algebraic completeness for $\mathbf{K}\Lambda$) If $s \in S$ and $\phi \in Form_s$ then $\vdash^s_{\mathbf{K}\Lambda} \phi$ if and only if $e_s(\phi) = 1_s$ in A_s for any (S, Σ)-BAO $\mathfrak{A}$ that is a model of $\mathbf{K}\Lambda$ and any assignment e in $\mathfrak{A}$.*

Finally, we relate our system with *matching logic* [4], a many-sorted first-order logic for program specification and verification. The completeness theorem for matching logic is proved using a interpretation in the first-order logic with equality. The starting point of our investigation was the representation of the (mono-sorted) polyadic modal logic as a particular system of matching logic in [4, Section 8]. Our initial goals were: to understand the propositional part of matching logic, to give a self-contained proof of the completeness theorem, to identify the algebraic theory and to investigate the relation with modal logic. The present system is an initial step in this direction.

Let (S, Σ) be a many-sorted signature. A *matching logic (S, Σ)-model* is

$$M = (\{M_s\}_{s \in S}, \{\sigma_M\}_{\sigma \in \Sigma})$$

where $\sigma_M : M_{s_1} \times \ldots \times M_{s_n} \to \mathcal{P}(M_s)$ for any $\sigma \in \Sigma_{s_1 \ldots s_n, s}$. To any matching logic model M we can associate an (S, Σ) BAO as follows:

- if we define $R_\sigma \subseteq M_{s_1} \times \cdots \times M_{s_n}$ by
$$R_\sigma w w_1 \ldots w_n \text{ iff } w \in \sigma_M(w_1, \ldots, w_n) \text{ for any } \sigma \in \Sigma_{s_1 \ldots s_n, s},$$
then $\mathcal{M} = (\{M_s\}_{s \in S}, \{R_\sigma\}_{\sigma \in \Sigma})$ is an (S, Σ)-frame;
- assuming that for any $s \in S$, $\mathcal{P}(M_s)$ has the usual structure of boolean algebra, then $\mathfrak{M} = (\{\mathcal{P}(M_s)\}_{s \in S}, \{m_\sigma\}_{\sigma \in \Sigma})$ is an (S, Σ)-BAO (the full complex algebra determined by $\mathcal{M}$).

With respect to the relation between matching logic and our general many-sorted polyadic modal logic, we can state the following:

- any matching logic model can be associated with an (S, Σ)-frame and vice versa,
- the algebraic structures of matching logic have a natural structure of (S, Σ)-boolean algebras with operators,
- the system $\mathbf{K}_{(S,\Sigma)}$ can be seen as the propositional counterpart of matching logic.

Even if Matching logic was the starting point of our research, one main issue was to connect our logic with already existing systems of many-sorted modal logic. In [8] the author defines a sound and complete two-sorted modal logic for projective planes. This system is a particular case of our general framework. We refer to [3] for more examples.

References

1. P. Blackburn, Y. Venema, M. de Rijke, Modal Logic, Cambridge University Press, 2002.
2. B. Jacobs, *Many-sorted coalgebraic modal logic: a model-theoretic study*, in: Theoretical Informatics and Applications Theoret. Informatics Appl. 35 (2001), pp. 31-59.

3. N. Moangă, I. Leuştean, *A many-sorted polyadic modal logic*, submitted (2018). arXiv:1803.09709
4. G. Roşu, *Matching logic* in: Logical Methods in Computer Science 13(4)(2017), pp. 1-61.
5. M. Rößiger, *From modal logic to terminal coalgebras*, in: Electronic Notes in Theoretical Computer Science 33 (2000), pp. 1-22.
6. J. van Benthem, *A Note on Dynamic Arrow Logic*, in: Tech Report, July 1992, Institute for Logic, Language and Computation,. University of Amsterdam; published in Jan van Eyck & Albert Visser, eds., 1994, Logic and Information Flow, The MIT Press, Cambridge (Mass.), pp. 15-29.
7. L. Schröder, D. Pattinson, *Modular algorithms for heterogeneous modal logics via multi-sorted coalgebra*, in: Mathematical Structures in Computer Science 21(2) (2011), pp. 235-266.
8. Y. Venema *Points, lines and diamonds: a two-sorted modal logic for projective planes*, in: Journal of Logic and Computation 9(5)(1999), pp. 601–621.

An operational-semantics-based approach to program verification using dynamic logic

Ioana Leuștean and Traian Florin Șerbănuță

Faculty of Mathematics and Computer Science,
University of Bucharest, Bucharest, Romania
ioana@fmi.unibuc.ro, traian.serbanuta@fmi.unibuc.ro

Traditionally, program verification within *modal logic*, as showcased by *dynamic logic* [4], is following the mainstream axiomatic approach proposed by Hoare/Floyd [6, 3]. More recently, Roşu [10] proposed *matching logic* and *reachability logic* as an alternative way to prove program correctness, using directly the (executable) operational semantics of a language.

In this paper, we take first steps in exploring the amenability of dynamic logic in particular, and of modal logic in general, to express operational semantics of languages (as axioms), and to make use of such semantics in program verification.

To do this, we use a general many-sorted polyadic modal logic, as defined in [7], which allows us to represent both the programs and their semantics in a uniform way.

Our language is determined by a fixed, but arbitrary, many-sorted signature (S, Σ) and a many-sorted set of propositional variables $P = \{P_s\}_{s\in S}$ such that $P_s \neq \emptyset$ for any $s \in S$ and $P_{s_1} \cap P_{s_2} = \emptyset$ for any $s_1 \neq s_2$ in S. For any $n \in \mathbb{N}$ and $s, s_1, \ldots, s_n \in S$ we denote $\Sigma_{s_1\ldots s_n,s} = \{\sigma \in \Sigma \mid \sigma : s_1 \ldots s_n \to s\}$.

The set of formulas of $\mathcal{ML}_S$ is an S-indexed family $Form_S = \{Form_s\}_{s\in S}$ inductively defined as follows:

$$\phi_s ::= p \mid \neg\phi_s \mid \phi_{1_s} \vee \phi_{2_s} \mid \sigma(\phi_{1_{s_1}}, \ldots, \phi_{n_{s_n}})$$

where $s \in S$, $p \in P_s$ and $\sigma \in \Sigma_{s_1\ldots s_n,s}$

We use the classical definitions of the derived logical connectives. For any $\sigma \in \Sigma_{s_1\ldots s_n,s}$ the *dual operation* is $\sigma^\Box(\phi_1, \ldots, \phi_n) := \neg\sigma(\neg\phi_1, \ldots, \neg\phi_n)$.

Let $\mathbf{K}_{(S,\Sigma)} = \{\mathbf{K}_s\}_{s\in S}$ be the least S-sorted set of formulas with the following properties:

(a0) for any $s \in S$, if $\alpha \in Form_s$ is a theorem in classical logic, then $\alpha \in \mathbf{K}_s$,
(a1) the following formulas are in $\mathbf{K}_s$
(K^i_σ) $\sigma^\Box(\psi_1, \ldots, \phi \to \chi, \ldots, \psi_n) \to$
$\to (\sigma^\Box(\psi_1, \ldots, \phi, \ldots, \psi_n) \to \sigma^\Box(\psi_1, \ldots, \chi, \ldots, \psi_n))$
$(Dual_\sigma)$ $\sigma(\psi_1, \ldots, \psi_n) \leftrightarrow \neg\sigma^\Box(\neg\psi_1, \ldots, \neg\psi_n)$
for any $n \geq 1$, $i \in [n]$, $\sigma \in \Sigma_{s_1\cdots s_n,s}$, $\psi_{s_1} \in Form_{s_1}$, ..., $\psi_{s_n} \in Form_{s_n}$ and $\phi, \chi \in Form_{s_i}$.

If $\Lambda \subseteq Form_S$ is an S-sorted set of formulas, then the *normal modal logic* defined by Λ is $\mathbf{K}\Lambda = \{\mathbf{K}\Lambda_s\}_{s\in S}$ where

$\mathbf{K}\Lambda_s := \mathbf{K}_s \cup \{\lambda' \in Form_s \mid \lambda'$ is obtained by substitution applied to a formula $\lambda \in \Lambda_s\}$

In the sequel we assume $\Lambda \subseteq Form_s$ is an S-sorted set of formulas and we investigate the normal modal logic $\mathbf{K}\Lambda$. In our approach, a logic is defined by its axioms. The deduction rules are *Modus Ponens* and *Universal Generalization*. Assume that $n \geq 1$, $s_1, \ldots, s_n \in S$ and $\phi_i \in Form_{s_i}$ for any $i \in [n]$. The sequence $\phi_1, \ldots, \phi_n$ is a $\mathbf{K}\Lambda$*-proof for* ϕ_n if, for any $i \in [n]$, φ_i is in $\mathbf{K}\Lambda_{s_i}$ or φ_i is inferred from $\varphi_1, \ldots, \varphi_{i-1}$ using *modus ponens* and *universal generalization*. If ϕ has a proof in $\mathbf{K}\Lambda$ then we say that ϕ is a *theorem* and we write $|\!\!\frac{s}{}_{\mathbf{K}\Lambda}\phi$ where s is the sort of ϕ.

In the remainder of the paper we develop a particular system $\mathbf{K}\Lambda$ which uses dynamic logic in a many-sorted setting. Our goal is to express operational semantics of languages as axioms in this logic, and to make use of such semantics in program verification. As a first step towards that goal, we consider here the SMC Machine described by Plotkin [8], we derive a Dynamic Logic set of axioms from its proposed transition semantics, and we argue that this set of axioms can be used to derive Hoare-like assertions regarding functional correctness of programs written in the SMC machine language.

The semantics of the *SMC machine* as laid out by Plotkin consists of a set of transition rules defined between configurations of the form $\langle S, M, C\rangle$ where S is a $ValueStack$ of intermediate results, M represents the $Memory$, mapping program identifiers to values, and C is the $ControlStack$ of commands representing the control flow of the program. As our target is to extend the Propositional Dynamic Logic (PDL) [4], we identify the $ControlStack$ with the notion of "programs" in dynamic logic, and use the ";" operator from dynamic logic to denote stack composition. We define our formulas to stand for *configurations* of the form $config(vs, mem)$ comprising only a value stack and a memory. Similarly to PDL, we use the modal operator $[\_]\_ : ControlStack \times Config \to Config$ to assert that a configuration formula must hold after executing the commands in the control stack. The axioms defining the dynamic logic semantics of the SMC machine are then formulas of the form $cfg \to [ctrl]cfg'$ saying that a configuration satisfying cfg must change to one satisfying cfg' after executing $ctrl$.

In Fig. 1 we introduce the signature of our logic as a context-free grammar (CFG) in a BNF-like form. We make use of the established equivalence between CFGs and algebraic signatures (see, e.g., [5]), mapping non-terminals to sorts and CFG productions to operation symbols. Note that, due to non-terminal renamings (e.g., Exp ::= Int), it may seem that our syntax relies on subsorting. However, this is done for readability reasons only. The renaming of non-terminals in syntax can be thought of as syntactic sugar for defining injection functions. For example, Exp ::= Int can be thought of as Exp ::= int2Exp(Int), and all occurrences of an integer term in a context in which an expression is expected could be wrapped by the int2Exp function.

The sorts `CtrlStack` and `Config` correspond to programs and formulas from PDL, respectively. Therefore the usual operations of dynamic logic ;, $\cup$, *, ?, $< \_ >$ and $[\_]$ are defined accordingly [4].

Next, we encode the transition system of the SMC machine as a set of axioms.

Syntax

```
 Nat ::=  natural numbers
 Var ::=  program variables
Bool ::= true | false
AExp ::= Nat | Var | AExp + AExp
BExp ::= AExp <= AExp
Stmt ::= x := AExp
       | if BExp
         then Stmt
         else Stmt
       | while BExp do Stmt
       | skip
       | Stmt ; Stmt
```

Semantics

```
     Val ::= n | b
ValStack ::= nil
           | Val . ValStack
     Mem ::= empty | set(Mem, x, n)
           | get(x,n)
CtrlStack ::= c(AExp)
           | c(BExp)
           | c(Stmt)
           | asgn(x)
           | plus   | leq
           | consumeVBool
           | c1 ; c2
  Config ::=  config(ValStack, Mem)
```

Fig. 1. Signature

Axioms

$(AMem0)$ $empty \to get(x, 0)$

$(AMem1)$ $set(mem, x, n) \to get(x, n)$

$(AMem2)$ $set(set(mem, x, n), y, m) \leftrightarrow set(set(mem, y, m), x, n)$ where x and y are distinct

$(AMem3)$ $set(set(mem, x, n), x, m) \leftrightarrow set(mem, x, m)$

(En) $config(vs, mem) \to [c(n)]config(n \cdot vs, mem)$ where n is an integer

(Ev) $config(vs, set(mem, x, n)) \to [c(x)]config(n \cdot vs, set(mem, x, n))$

$(E + I)$ $c(a1 + a2) \leftrightarrow c(a1); c(a2); plus$

$(E + E)$ $config(n2 \cdot n1 \cdot vs) \to [plus]config(n \cdot vs)$ where n is $n1 + n2$

$(B \leq I)$ $c(a1 <= a2) \leftrightarrow c(a2); c(a1); leq$

$(B \leq E)$ $config(n1 \cdot n2 \cdot vs, mem) \to [leq]config(truth(n1 <= n2) \cdot vs, mem)$

$(C$ **nil** $)$ $c(skip) \leftrightarrow \mathbf{1}?$

$(C := I)$ $c(x := a) \leftrightarrow c(a); asgn(x)$

$(C := E)$ $config(n \cdot vs, mem) \to [asgn(x)]config(vs, set(mem, x, n))$

$(C;)$ $c(s1; s2) \leftrightarrow c(s1); c(s2)$

$(C$ **if** $I)$ $c(if\ b\ then\ s1\ else\ s2) \leftrightarrow c(b); \mathbf{if}$
$config(true \cdot vs, mem) \to consumeVBool; c(s1)$
$\mid config(false \cdot vs, mem) \to consumeVBool; c(s2)$
$\mathbf{fi}$

$(C$ **while** $I)$ $c(while\ b\ do\ s) \leftrightarrow c(b); \mathbf{do}$
$config(true \cdot vs, mem) \to consumeVBool; c(s); c(b)$
$\mathbf{od}; consumeVBool$

$(AVBool)$ $config(t \cdot vs, mem) \to [consumeVBool]config(vs, mem)$

Note that, in order to keep as close to PDL as possible, and to be able to leverage their axioms in proofs, we make use in the axioms above of the PDL notations **if**..**fi** and **do**..**od** [4, Section 2.1]. Other than that, and the axioms for memory (which are straight-forward), we follow the rules of the SMC machine as closely as allowed by the formalism, using the same notation as in [8].

Finally, our logical system is $\mathbf{K}\Lambda$, where Λ contains the above axioms, as well as the axioms of PDL [4, Section 4.1].

The general theory from [7] provides two completeness results: the completeness with respect to the canonical frame [7, Theorem 2.19] and the completeness with respect to the class of many-sorted boolean algebras with operators that are models of Λ [7, Theorem 4.8]. The canonical model is a *non-standard* model with respect to the dynamic logic fragment of our system, meaning that the induction axioms hold on the sort `CtrlStack`, but the interpretation of the c^* operation might not be the reflexive and transitive closure of the interpretation of c, where c is a formula of sort `CtrlStack` (see [4, Section 3.2] for more details on standard and non-standard models).

We conclude this abstract by a simple example formalizing and stating a formula which can be proven by deduction in our logic.

Let pgm be the following program

```
i1:= 1; i2:= 2; if i1<=i2 then m:= i1 else m:= i2
```

Note that pgm is a formula of sort `Stmt` in our logic, m is a formula of sort `Var` and 1 is aformula of sort `Nat`. For this formula we can state and prove the following:

$$\begin{array}{c}\text{if } memi, memf \text{ are formulas of sort } \texttt{Mem} \text{ and} \\ vs \text{ is a formula of sort } \texttt{ValStack} \text{ then} \\ \vdash^{\texttt{Config}} config(vs, memi) \rightarrow [c(pgm)]config(vs', memf) \text{ implies} \\ \vdash^{\texttt{ValStack}} vs \leftrightarrow vs' \text{ and } \vdash^{\texttt{Mem}} memf \rightarrow get(m, 1).\end{array}$$

Which, can be read in plain English as: after executing pgm the value of the program variable m (in memory) will be 1, and the value stack will be the same as before the execution.

A sketch of the proof for the above statement as well as a more complex case-study involving loops and invariants will be presented at the workshop.

Related Work. It has been shown [2] that **Separation Logic** [9] (as branch-logic over the theory of maps) can be faithfully represented in modal logic; therefore one can employ dynamic logic reasoning directly on top of separation logic. Instead of following that approach, we here follow more closely the line of work of **Matching Logic** [10], using directly the operational semantics rather than the traditional FLoyd/Hoare axiomatic semantics. Similarly to matching logic, the use of configurations allows the description and "separation" of more than just heaps.

References

1. P. Blackburn, Y. Venema, M. de Rijke, Modal Logic, Cambridge University Press, 2002.

2. C. Calcagno, P. Gardner, U. Zarfaty. "Context logic as modal logic: completeness and parametric inexpressivity". In *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '07)*, 123-134. 2007. ACM, New York, NY, USA
3. R. W. Floyd,*Assigning meanings to programs.* Proceedings of the American Mathematical Society Symposia on Applied Mathematics. Vol. 19, pp. 19–31. 1967.
4. D. Harel, D. Kozen, J. Tiuryn, Dynamic logic, MIT Press, 2000.
5. J Heering, PRH Hendriks, P Klint, J Rekers, *The syntax definition formalism SDF —reference manual—*. ACM Sigplan Notices 24(11):43–75. 1989
6. C. A. R. Hoare, *An axiomatic basis for computer programming.* Communications of the ACM. 12 (10): 576–580 (1969).
7. N. Moangă, I. Leuştean, A many-sorted polyadic modal logic,submitted (2018). arXiv:1803.09709
8. G. D. Plotkin. A Structural Approach to Operational Semantics. (1981) Tech. Rep. DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark. (Reprinted with corrections in J. Log. Algebr. Program. 60-61: 17-139. 2004
9. Reynolds, John C. "Separation logic: A logic for shared mutable data structures." *Logic in Computer Science*: 55–74, 2002. Proceedings. 17th Annual IEEE Symposium on. IEEE, 2002.
10. G. Rosu, *Matching logic* in: Logical Methods in Computer Science 13(4)(2017), pp. 1-61.

Solving a variant of the 2-D pattern marching problem using Networks of Polarized Evolutionary Picture Processors with a restriction in polarity.

Ştefan Popescu

Department of Computer Science, University of Bucharest
popescustefan@fmi.unibuc.ro

Picture languages have been introduced in literature as a two-dimensional extension of traditional string languages [1]. Defining a picture as being a two-dimensional array of elements from a finite set of symbols can be considered as a formal model for image processing or pattern recognition. There are many computational models that try to generalize properties and operations from strings to 2-dimensional languages as well as to classify picture in a similar way as string languages: regular expressions [4], grammars [2, 7], automata [3]. One of the main focuses in this area is to define models for picture recognizability and pattern matching. A natural generalization of the classic pattern matching problem is the one where when given two 2-dimensional arrays (pictures) as input, we are tasked with determining if the first one occurs as a sub-array of the second. A classical approach would be the generalization of algorithms like KMP such as in [8]. We propose a different approach, using a bio-inspired computing model.

Mechanisms inspired from cell biology were considered: networks of interconnected node-processors which are able to perform only simple operations such as row/column deletion or symbol substitution. Moreover, the operations can affect only the outer rim of the picture. These nodes are assigned with communication protocols defined by some very simple context conditions. More simply said, each node processor acts on its local data set in accordance with some predefined evolutionary rules. This data set is then passed over the network according to some protocols based on compatibility and processor adjacency. All the nodes send and receive their data at once from the nodes they are connected to.

We begin by defining a bio-inspired computational model for deciding 2-dimensional (picture) languages that was first presented in [6] and is an extension of the one described in [5]. There are two main differences between the model presented here and the previous ones. Firstly, we have a slightly different accepting protocol for our current model based on an *Halting* node and a separate *Accepting* node. Secondly we have added a new type of operation inspired by biology, besides deletion and substitution, namely circular permutation. We show that by using this type of permutations and only two possible values for the polarity of a symbol, such networks can recognize any input that contains a given sub-picture as where our previous models limited one of the pattern dimensions to a size of at most 3 as was the case in [5] or we have used 3 possible values for the valence of each symbol as in the construction given in [6].

We briefly describe the computational process of such a network: Initially the input picture is placed in a special input node of the network. The computation itself is actually a succession of evolutionary and communication steps that take place one after another. In an evolutionary step, all the data sets assigned to each processor (or node) are affected by the rules specific to that processor (be it a substitution, deletion or circular permutation) as in the following example.

Example 1 *Let* $\pi = \begin{array}{|c|c|c|c|}\hline a&b&a&b\\\hline b&c&a&c\\\hline c&b&a&b\\\hline\end{array}$ *and the following evolutionary rules:*

$\sigma_1 \equiv a \to c \in Sub_V$ *and* $\sigma_2 \equiv a \to \epsilon \in Del_V$. *Then we have the following results:*

$$\sigma_1^{\uparrow}(\pi) = \left\{\begin{array}{|c|c|c|c|}\hline \boldsymbol{c}&b&a&b\\\hline b&c&a&c\\\hline c&b&a&b\\\hline\end{array}, \begin{array}{|c|c|c|c|}\hline a&b&\boldsymbol{c}&b\\\hline b&c&a&c\\\hline c&b&a&b\\\hline\end{array}\right\};\ \sigma_2^{\uparrow}(\pi) = \left\{\begin{array}{|c|c|c|c|}\hline b&c&a&c\\\hline c&b&a&b\\\hline\end{array}\right\};\ \sigma_1^{\rightarrow}(\pi) = \sigma_2^{\rightarrow}(\pi) = \left\{\begin{array}{|c|c|c|c|}\hline a&b&a&b\\\hline b&c&a&c\\\hline c&b&a&b\\\hline\end{array}\right\}$$

Applying $\sigma_1^{\uparrow}$ *on the picture* π *yields two results because there are two occurrences of* a *that can be substituted on the first row. In the case of* $\sigma_2^{\uparrow}$, *the top row is deleted because it contains at least one* a. *Both* $\sigma_1^{\rightarrow}(\pi)$ *and* $\sigma_2^{\rightarrow}(\pi)$ *have the same outcome, the unmodified picture* π, *because there is no occurrence of* a *on the rightmost column.*

The circular permutation is somewhat of a different nature, not depending on the presence of a certain symbol as in the case of previous point-mutation operations:

- $\curvearrowright(\pi)$ *returns the picture obtained from* π *by shifting its leftmost column after the rightmost column of* π. *Analogously,* $\curvearrowleft(\pi)$ *returns the picture obtained from* π *by shifting its rightmost column before the leftmost column of* π.
- $\circlearrowleft(\pi)$ *returns the picture obtained from* π *by shifting its upmost row below the downmost row of* π. *Analogously,* $\circlearrowright(\pi)$ *returns the picture obtained from* π *by shifting its downmost row above the upmost row of* π.

After an evolutionary step, a communication step takes place where, according to a protocol based on integer values associated with each symbol, adjacent nodes exchange information. Each picture is assigned a polarity based on the valence (integer number) associated to each symbol on the outer edges. The polarity of a picture is an element of the set $\{-,0,+\}^4$ depending on the sum of the valences of the symbols that are found on the top row, the rightmost column, the bottom row and the leftmost column.

We call the *valence of a symbol* of V as being the resulting value of a function $\varphi : V \to \mathbb{Z}$ that associates with every symbol from V an integer value.

The *polarity of a picture* is denoted by the function $\Phi : V_*^* \to \{-,0,+\}^4$ which associates a symbol from $\{-,0,+\}$ to the top and bottom rows, the leftmost and rightmost column. We define the polarity of a picture π of size (m,n) as:

$$\begin{aligned}\Phi(\pi) =& (sign(\sum_{i=1}^{n}\varphi(\pi(1,i))), sign(\sum_{i=1}^{m}\varphi(\pi(i,n))),\\ & sign(\sum_{i=1}^{n}\varphi(\pi(m,i))), sign(\sum_{i=1}^{m}\varphi(\pi(i,1)))).\end{aligned}$$

Example 2 *Let* $V = \{a, b, c, d\}$*, with the following symbol valences:* $\varphi(a) = 1$*;* $\varphi(b) = -2$ *and* $\varphi(c) = \varphi(d) = 0$*.*

For $\pi = \begin{array}{|c|c|c|c|}\hline a & b & c & d \\ \hline a & d & c & c \\ \hline b & a & a & a \\ \hline c & b & c & d \\ \hline \end{array}$ *the picture polarity of* π *is* $\Phi(\pi) = (-, +, -, 0)$*. As we can see, the sum of the valences of elements on the firs row is negative, while the valences of elements on the rightmost column yields a positive sum.*

The computation ends when something enters the network's $Halt$ processor. At this point the input is considered accepted if another special node, called the *Accept* node, is non-empty (or in the case of problem solving models, the answer is actually the information contained in the node itself).

We finally describe very succinctly how we solve our aforementioned pattern matching problem. Our main network can be broken down into three main parts. The first subnetwork's main action is deleting various rows and columns from the input picture. By deleting the outer rows and columns in all possible orders we obtain the set of all sub-pictures of our original input. This resulting set is continuously communicated to the second sub-network where it is checked in a massive parallel way if a given picture is actually identical to the pattern we are searching for. If there is such a picture, the *Accept* node will become non-void and will stay as such for the rest of the computation. The third sub-network is tasked with timing the whole process. Because the input picture is finite in size, the number of sub-pictures that need to be checked is also limited. According to the size of the input, this sub-network acts as a delay, giving time for all sub-pictures to be checked, before sending something to the $Halt$ node and ending the whole process.

References

1. D. Giammarresi, A. Restivo, Two-dimensional languages. In *Handbook of Formal Languages*, Springer–Verlag, Berlin, 1997, 215–267.
2. D. Giammarresi, A. Restivo, Recognizable picture languages, *Int. J. Pattern Recognition and Artificial Intelligence*, 6(1992) 241–256.
3. I. Inoue, I. Takanami, A survey of two-dimensional automata theory, *Proc. 5th Int. Meeting of Young Computer Scientists*, LNCS 381 Springer-Verlag, Berlin, 1990, pp. 72-91.
4. O. Matz, Regular expressions and context-free grammars for picture languages. In *Proc. 14th STACS*, Lecture Notes in Computer Science 1200, Springer-Verlag, 1997, pg 283-294.
5. Şt. Popescu, Networks of Polarized Evolutionary Picture Processors. In *Romanian Journal of Information Science and Technology*, Volume 18 Issue 1, 2015, pg 3–17.
6. Şt. Popescu, Solving 2-D Pattern Matching using Networks of Polarized Picture Processors With Circular Permutation; Presented at the *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, accepted for publication.

7. A. Rosenfeld, R. Siromoney, Picture languages – a survey, *Languages of design*, 1(1993) 229–245.
8. R.F. Zhu, T. Takaoka, A technique for two-dimensional pattern matching, *Communications of the ACM*, 32(1989) 1110–1120.

Compositional Verification of Reachability-Logic Properties on Reachability-Logic Specifications

Vlad Rusu* Inria, Lille, France Vlad.Rusu@inria.fr

1 Introduction

Reachability Logic (RL) [1] is both a formalism for specifying operational semantics of languages and a logic for specifying programs in the languages in question. RL has *Matching Logic (ML)* as a sublogic: matching and rewriting with RL formulas (denoting operational-semantics rules) can be used for executing programs. *Symbolic* program-execution and, beyond, symbolic program-verification w.r.t. RL formulas can also be achieved, but, in practice this requires complex operations such as unification modulo theories combined with SMT solving [2].

In this paper we present a version of RL not based on ML but on *Parameterised State Predicates (PSPs)*. Parameter instantiation plays the role of matching in ML-based RL, and symbolic execution becomes simpler, as it merely amounts to generating PSPs with existential quantifiers. The resulting RL version is best suited for being embedded in an interactive theorem prover such as Coq [3], where users can deal with simplifications such as existential-quantifier elimination.

A key design choice we made is *mixed embedding*: predicates are shallowly embedded in Coq's builtin type `Prop`, which has many dedicated proof commands users can benefit from. The rest, including a new proof system for RL, is deeply embedded in Coq's inductive types. As a result, an earlier version of the approach (differences with the present approach are given below) enabled us to prove both meta-properties of the logic (soundness and completeness of a proof system) and a nontrivial case study - the correctnes of a security hypervisor [4]. By contrast, *deep* embeddings [1, 6] of RL in Coq are not really practically usable in substantial applications because they are tailored to proving meta-properties of the logic.

In this paper we extend our previous approach as follows (1): the proof system now has more proof rules, making it easier and more flexible to use and leading to cleaner theoretical results; in particular, the *compositionality* rule allows one to prove formulas on sub-specifications called *components*; (2) RL formulas now have *parameters*, which are used for memorising previous values of state variabes and for abstracting away some of them when formulas are interpreted on the "poorer" state-spaces of components, and (3) specifications are also given as RL formulas that, for simplicity, only refer to the subset of state variables that are read or written, leaving all other (unmentioned) variables implicitly unchanged.

Finally, unlike the previous version [4] targeted at transition system models, our ambition is to apply the present approach to verifying *programs*, via the operational semantics of their languages defined in our embedding of RL in Coq.

2 Transition Systems and Parameterised State Predicates

Part of the success that RL has had for defining operational semantics of real languages (e.g., C and Java, among the languages implemented in the

* Partially supported by the ODSI project C2014/2-12, http://celticplus-odsi.org.

K-framework [5]) is due to the fact that RL formulas *only* mention what *part* of a programs's *state* (code, heap, stack, ...- depending on the programming language) is *accessed* (in reading and/or writing) when executing a given instruction. In this way, the complexity of writing "real" semantical rules remains manageable.

In our approach we emulate this useful feature by having a set of *state-spaces*, organised as a *complete upper semi-lattice*. Thus, state-spaces are ordered by a partial order $\sqsubset$, and any set $\{S_i\}_{i\in I}$ of state-spaces has a least upper bound $lub(\{S_i\}_{i\in I})$. In addition to these standard constructions, we require that for each S, S' such that $S \sqsubset S'$, there is a bijection $\alpha : S' \to C \times S$ for some *context* C.

We then write $S \sqsubset_\alpha S'$, meaning that the state-space S' is (up to the bijection α) a Cartesian product, and S is its projection on the first component. We let $\alpha_C : S' \to C$ (resp. $\alpha_S : S' \to S$) denote the composition of α with the projection of pairs on their context component (resp. on their state component).

A *transition system* is a pair $(S, \to)$, where S is a state-space and $\to \subseteq S \times S$. We write $s \to s'$ for $(s, s') \in \to$. A *path* is a finite sequence of states connected by the transition relation $\to$. We denote by $Paths(S, \to)$ the set of paths of $(S, \to)$. The *length* $len(\tau)$ is defined for each path $\tau \triangleq s_0 \to \cdots \to s_{n-1} \to s_n$ as $len(\tau) = n$. For $0 \leq i \leq len(\tau)$ we denote by $\tau(i)$ the i-th state in the path τ. A path is *complete* if its last state $s \triangleq \tau(len(\tau))$ is *final*, i.e., there is no transition $s \to s'$. We denote by $comPaths(S, \to)$ the set of complete paths of $(S, \to)$.

Given a transiton system $(S, \to)$ and a state-space S' such that $S \sqsubset_\alpha S'$, the *α-lifting of* $(S, \to)$ is the transition system $(S', \to')$ with $s'_1 \to' s'_2$ iff $\alpha_S(s'_1) \to \alpha_S(s'_2)$ and $\alpha_C(s'_1) = \alpha_C(s'_2)$. When this is the case we write $\to' = \uparrow_\alpha(\to)$. Given a set of transition systems $\{(S_i, \to_i)\}_{i\in I}$, we call $(S, \to)$ the *lifting* of $\{(S_i, \to_i)\}_{i\in I}$, and write $(S, \to) = \uparrow(\{(S_i, \to_i)\}_{i\in I})$, if $S = lub(\{(S_i\}_{i\in I})$ and, based on the fact there are bijections α_i such that $S_i \sqsubset_{\alpha_i} S$ for all $i \in I$, $\to = \bigcup_{i\in I}(\uparrow_{\alpha_i}(\to_i))$.

We say that a transition system $(S \to)$ is an *α-component* of a transition system $(S', \to')$, denoted by $(S, \to) \lhd_\alpha (S', \to')$, if (i) $S \sqsubset_\alpha S'$, (ii) for all states $s'_1, s'_2 \in S'$, $\alpha_S(s'_1) \to \alpha_S(s'_2)$ implies $s'_1 \to' s'_2$, and (iii) for all states $s', s'_1, s'_2 \in S'$, ($s' \to' s'_1$ and $\alpha_S(s') \not\to \alpha_S(s'_1)$ and $s' \to' s'_2$) implies $\alpha_S(s') \not\to \alpha_S(s'_2)$,

Explanations: (i), (ii) mean that (up to the bijection α) the component $(S \to)$ has a "poorer" state-space, and "fewer" transitions, than $(S' \to')$. The condition (iii) means that the component $(S \to)$ is obtained by "removing" transitions from $(S' \to')$, such that from each $s' \in S'$, either all transitions are removed, or none.

In Coq, transition systems are abstractly represented by declaring with `S:Type` the type of states together with the declaration of a transiton relation `S→S→Prop`. Concrete transition systems additionally define the type `S` and the transition relation. In this paper we define transition relations using sets of RL formulas.

Proving RL formulas requires some symbolic manipulations, and *Parameterised state predicates* (PSPs) are the symbolic representations chosen here.

We thus assume a finite set of *parameters spaces*. For a parameter-space Π and state-space S, a *PSP p of signature Π, S* (denoted by $p : \Pi, S$) is a predicate on $\Pi \times S$. We identify each PSP $\pi : \Pi, S$ with an anonymous $\{True, False\}$-valued function $\lambda(\pi : \Pi)(s : S)p\,\pi\,s$ that returns *True* on $\pi \in \Pi$ and $s \in S$ if $p\,\pi\,s$ and otherwise returns *False*. PSPs with the same signature are closed

under conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), implication ($\rightarrow$). The PSPs $\top$ (resp. $\bot$) are satisfied by all (resp. by none of) the states and parameters. For PSPs p, q, we write $p \Rightarrow q$ to denote the fact that for all states s and parameters π, if $p\,\pi\,s$ then $q\,\pi\,s$. Strictly speaking, PSP operations and constants are parameterised by signatures; we omit them for notation simplicity.

PSPs have a function type, written in Coq as `PSP:Type:= fun (Pi S:Type)⇒ Pi→S→Prop`. Then, the conjunction, disjunction, negation, implication, bottom, and top operations and constants on PSPs over the same signature are defined using Coq's predefined operations $\wedge$, $\vee$, $\neg$, $\rightarrow$, `False`, and `True`. Finally, $p \Rightarrow q$ translates to `imp(p q:PSP(Pi S:Type)):Prop:=`$\forall(\pi : \Pi)$ `(s:S),p` π `s` $\rightarrow$ `q` π `s`.

Since PSPs are defined over different signatures, in particular, over different state-spaces, it is useful to *raise* or *lower* them to other state-spaces in our state-space hierarchy. A PSP $p : \Pi, S$, when α-*raised* to a state space S' with $S \sqsubset_\alpha S'$, becomes the PSP $\uparrow_\alpha (p) : \Pi, S' \triangleq \lambda(\pi : \Pi)(s' : S').p\,\pi\,\alpha_S(s')$. Conversely, $p' : \Pi', S'$, when α-*lowered* to S such that $S \sqsubset_\alpha S'$ - i.e., there is a bijection $\alpha : S' \rightarrow C \times S$ - is $\downarrow_\alpha(p') : (\Pi' \times C), S \triangleq \lambda(\pi', c) : (\Pi' \times C)(s : S).p\,\pi'\,\alpha^{-1}(c, s)$.

Thus, raising a PSP ignores the additional structure in the "richer" state-space, wheras lowering a PSP "moves" a part of its state structure to parameters.

3 Reachability Logic

Definition 1. *An RL formula f over signature Π, S - denoted by $f : \Pi, S$, is a pair $f \triangleq l \lhd r$ where $l, r : \Pi, S$ are PSPs. We let $lhs(l \lhd r) \triangleq l$, $rhs(l \lhd r) \triangleq r$.*

Any RL formula $f : \Pi, S$ induces a transition system $\mathcal{T}(f) \triangleq (S, \rightarrow_f)$ with $s \rightarrow_f s'$ iff $lhs(f)\,\pi\,s$ and $rhs(f)\,\pi\,s'$ for some $\pi \in \Pi$. A set $\mathcal{S}$ of RL formulas $\{f_i : \Pi_i, S_i\}_{i \in I}$ induces the transition sistem $\uparrow(\{(\mathcal{T}(f_i)\}_{i \in I})$, cf. lifting operation $\uparrow$ in the previous section. When $\mathcal{S} \triangleq \{f_i : \Pi_i, S_i\}_{i \in I}$ we let $\uparrow(\mathcal{S}) \triangleq \uparrow(\{(\mathcal{T}(f_i)\}_{i \in I})$.

Let $\mathcal{S} \triangleq \{f_i : \Pi_i, S_i\}_{i \in I}$ be a formula set and $f \triangleq l \lhd r : \Pi, S$ be a formula. Then, f *is consistent with* $\mathcal{S}$, denoted by $\mathcal{S} \sqsupseteq f$, if $S \sqsubseteq lub(\bigcup_{i \in I} S_i)$. When $\mathcal{S} \sqsupseteq f$ we denote by $\uparrow_{\mathcal{S}}(f)$ the formula $(\uparrow_{\mathcal{S}}(l)) \lhd (\uparrow_{\mathcal{S}}(r))$ where $\uparrow_{\mathcal{S}}(l)$ is the raising of the PSP $l : \Pi, S$ to the state-space $lub(\bigcup_{i \in I} S_i)$ (and similarly for $\uparrow_{\mathcal{S}}(r)$).

Definition 2 (Semantical Entailment). *Consider a set $\mathcal{S}$ of RL formulas $\mathcal{S}$ and a formula $f : \Pi, S$ such that $\mathcal{S} \sqsupseteq f$. A path $\tau \in Paths(\uparrow(\mathcal{S}))$* satisfies *$f$ with parameters $\pi \in \Pi$, denoted by $\tau \models_\pi f$, if $lhs(\uparrow_{\mathcal{S}}(f))\,\pi\,\tau(0)$ and $rhs(\uparrow_{\mathcal{S}}(f))\,\pi\,\tau(k)$ for some $k \in \{0, \ldots, len(\tau)\}$. We say $\mathcal{S}$* semantically entails *f, denoted by $\mathcal{S} \models f$, if for all $\tau \in comPaths(\uparrow(\mathcal{S}))$ and all $\pi \in \Pi$, if $lhs(\uparrow_{\mathcal{S}}(f))\,\pi\tau(0)$ then $\tau \models_\pi f$.*

Apart from technicalities arising from formulas having parameters and different state-spaces, the definition of semantical entaiment $\mathcal{S} \models f$ just says that paths (in the transition system generated by $\mathcal{S}$) that "start" from $lhs(f)$ must "meet" $rhs(f)$ "along the way" to a final state. It is thus the usual definition of semantical entailment in *all-paths* RL [1], which is known to capture partial correctness.

The constraint "f consistent with $\mathcal{S}$" ($\mathcal{S} \sqsupseteq f$) just says that f, which is about the specification $\mathcal{S}$, may (naturally) only "talk about" a state-space known to $\mathcal{S}$.

As in most logics, semantical entailment $\models$ is not practical for proving formulas. We propose a syntactical entailment $\Vdash$, and show the two entailments coincide. The entailment $\Vdash$ is embodied in a proof system having the following ingredients.

Final states Hereafter we denote by ϕ the (parameterless) state predicate (for a given transition system $(\mathcal{S}, \rightarrow)$): $\phi \triangleq \lambda(s : S).\forall s', \neg(s \rightarrow s')$, i.e., s is final iff $\phi\, s$.

Derivatives Derivatives perform a "symbolic execution" of PSPs by RL formulas.

Definition 3 (Derivative). *Consider a PSP $p : \Pi, S$ an RL formula $l' \lhd r' : \Pi', S'$, with same state-spaces $S = S'$. The* derivative of p with respect to $l' \lhd r'$ *is defined by $\Delta_{l' \lhd r'}(p) \triangleq \lambda(\pi : \Pi).\lambda s.\exists(\pi' : \Pi').\exists s'.p\,\pi\,s' \wedge l'\pi' s' \wedge r'\pi' s$.*

Thus, $\Delta_{l' \lhd r'}(p)$ is a state predicate over the same signature as p. It is identified with a $\{True, False\}$-valued function with arguments $\pi : \Pi$ and states s, saying that there is a state s' satisfying $p\,\pi\,s'$ that "performs' a transition induced by the RL formula $l' \lhd r'$ to the current state s, i.e., $l'\,\pi' s' \wedge r'\,\pi'\,s$, for some value π' of the parameters Π' of $l' \lhd r'$. We note that we assumed for simplicity that the state-spaces of $l \lhd r$ and of p coincide. If this is not the case, state-spaces can first be lifted to their common *lub* state-space before the derivative is computed.

Compositional reasoning is based on α-components, α-lowering and this lemma:

Lemma 1 (Inferring Semantical Entailment). *Consider formula sets $\mathcal{S}', \mathcal{S}$ such that $(\uparrow(\mathcal{S}')) \lhd_\alpha (\uparrow(\mathcal{S}))$, and a formula $f \triangleq l \lhd r$ with $\mathcal{S} \sqsupseteq f$. Let $\downarrow_\alpha(f) \triangleq (\downarrow_\alpha(l)) \lhd (\downarrow_\alpha(r))$. Then, $\mathcal{S}' \sqsupseteq \downarrow_\alpha(f)$, and $\mathcal{S}' \models\!\downarrow_\alpha(f)$ implies $\mathcal{S} \models f$.*

We note that if $\mathcal{S}' \subseteq \mathcal{S}$, conditions (i) and (ii) from hypothesis $(\uparrow(\mathcal{S}')) \lhd_\alpha (\uparrow(\mathcal{S}))$ in the lemma already hold, thus, to use the lemma, only (iii) has to be checked.

Rules For technical reasons (related to the system's soundness) we shall consider *tagged* formulas $\langle b, i, l \lhd r\rangle$,i.e., triples consisting of a Boolean, a natural number and an RL formula. There are eight rules in our proof system (shown below).

Each rule transforms a *sequent* (of the form $\mathcal{S}, \mathcal{H} \vdash \mathcal{G}$, where $\mathcal{S}$, $\mathcal{H}, \mathcal{G}$ are sets of tagged formulas) into another sequent. Here, $\mathcal{S}$ is the current specification, $\mathcal{G}$ is the set of formulas currently under proof, and $\mathcal{H}$ is the set of current hypotheses.

$$\textsf{[Trv]}\ \frac{\mathcal{S}, \mathcal{H} \vdash \mathcal{G}}{\mathcal{S}, \mathcal{H} \vdash \{\langle \_, \_, l \lhd r\rangle\} \cup \mathcal{G}}\ \textit{if } l \Rightarrow r$$

$$\textsf{[Cmp]}\ \frac{\mathcal{S}, \mathcal{H} \vdash \mathcal{G}}{\mathcal{S}, \mathcal{H} \vdash \{\langle \_, \_, f\rangle\} \cup \mathcal{G}}\ \textit{if} (\uparrow(\mathcal{S}')) \lhd_\alpha (\uparrow(\mathcal{S})),\ \mathcal{S}' \Vdash\!\downarrow_\alpha(f)$$

$$\textsf{[Spl]}\ \frac{\mathcal{S}, \mathcal{H} \vdash \{\langle b, i+1, l_1 \lhd r\rangle, \langle b, i+1, l_2 \lhd r\rangle\} \cup \mathcal{G}}{\mathcal{S}, \mathcal{H} \vdash \{\langle b, i, l \lhd r\rangle\} \cup \mathcal{G}}\ \textit{if } l \Rightarrow (l_1 \vee l_2)$$

$$\textsf{[Stp]}\ \frac{\mathcal{S}, \mathcal{H} \vdash \bigcup_{f \in \mathcal{S}} \{\langle true, i+1, l_f \lhd r\rangle\} \cup \mathcal{G}}{\mathcal{S}, \mathcal{H} \vdash \{\langle b, i, l \lhd r\rangle\} \cup \mathcal{G}}\ \textit{if } l \wedge \lambda(\_ : \Pi)\phi \Rightarrow \bot, \forall f \in \mathcal{S}.\Delta_f(l) \Rightarrow l_f$$

$$\textsf{[Hyp]}\ \frac{\mathcal{S}, \mathcal{H} \cup \{\langle false, 0, l' \lhd r'\rangle\} \vdash \{\langle true, i+1, l'' \lhd r\rangle\} \cup \mathcal{G}}{\mathcal{S}, \mathcal{H} \cup \{\langle false, 0, l' \lhd r'\rangle\} \vdash \{\langle true, i, l \lhd r\rangle\} \cup \mathcal{G}}\ \textit{if } l \Rightarrow l', \Delta_{l' \lhd r'}(l) \Rightarrow l''$$

$$\textsf{[Sub]}\ \frac{\mathcal{S}, \mathcal{H} \vdash \{\langle b, max(i,j)+1, l' \lhd r\rangle\} \cup \mathcal{G}}{\mathcal{S}, \mathcal{H} \vdash \{\langle b, j, l \lhd r\rangle, \langle b, i, l' \lhd r\rangle\} \cup \mathcal{G}}\ \textit{if } l \Rightarrow l'$$

$$\textsf{[Add]}\ \frac{\mathcal{S}, \mathcal{H} \cup \{\langle false, 0, l \lhd r\rangle\} \vdash \mathcal{G} \cup \{\langle false, 0, l \lhd r\rangle\}}{\mathcal{S}, \mathcal{H} \vdash \mathcal{G}}$$

$$\textsf{[Rem]}\ \frac{\mathcal{S}, \mathcal{H} \vdash \mathcal{G}}{\mathcal{S}, \mathcal{H} \cup \mathcal{H}' \vdash \mathcal{G}}$$

Definition 4 (proof (fragment)). *A* proof fragment *is a sequence of sequents* $\mathcal{S}_0, \mathcal{H}_0 \vdash \mathcal{G}_0 \ \cdots \ \mathcal{S}_n, \mathcal{H}_n \vdash \mathcal{G}_n$ *such that for all* $i \in \{0 \dots n-1\}$, $\mathcal{S}_{i+1}, \mathcal{H}_{i+1} \vdash \mathcal{G}_{i+1}$ *is obtained from* $\mathcal{S}_i, \mathcal{H}_i \vdash \mathcal{G}_i$ *by applying (bottom-up) a rule of the proof system. A* proof of f in $\mathcal{S}$ *is a proof fragment where* $\mathcal{S} = \mathcal{S}_0$, $\mathcal{H}_0 = \mathcal{H}_n = \mathcal{G}_n = \emptyset$ *and* $\mathcal{G}_0 = \{\langle \mathit{false}, 0, f\rangle\}$. *We note by* $\mathcal{S} \Vdash f$ *the fact that* f *has a proof in* $\mathcal{S}$.

Theorem 1. *Assume* f *is consistent with* $\mathcal{S}$. *Then,* $\mathcal{S} \models f$ *if and only if* $\mathcal{S} \Vdash f$.

We show the proof's main idea. For soundness ($\Leftarrow$ implication), let $\mathcal{F} \triangleq \bigcup_{0 \le i \le n} \mathcal{G}_i$ where $\mathcal{S}_0, \mathcal{H}_0 \vdash \mathcal{G}_0 \cdots \mathcal{S}_n, \mathcal{H}_n \vdash \mathcal{G}_n$ is a proof of f in $\mathcal{S}_0$. Let the set $\mathcal{D} = \{(\mathcal{S}, \pi, \tau, \langle b, j, l \lhd r\rangle) \mid \tau \in \mathit{comPaths}(\uparrow(\mathcal{S})), \uparrow_{\mathcal{S}}(l)\, \pi\, \tau(0), l \lhd r \in \mathcal{F}\}$, and a relation $\prec$ over $\mathcal{D}$ such that $(\mathcal{S}, \pi, \tau, \langle b, j, l \lhd r\rangle) \prec (\mathcal{S}', \pi', \tau', \langle b', j', l' \lhd r'\rangle)$ if: either $(\uparrow(\mathcal{S}')) \lhd_\alpha (\uparrow(\mathcal{S}))$ occured in the proof of f; or $\mathcal{S}' = \mathcal{S}$ and $\mathit{len}(\tau) < \mathit{len}(\tau')$; or $\mathcal{S}' = \mathcal{S}, \mathit{len}(\tau) = \mathit{len}(\tau')$ and $b < b'$ where $<$ on Booleans is defined by $\mathit{false} < \mathit{true}$; or $\mathcal{S}' = \mathcal{S}, \mathit{len}(\tau) = \mathit{len}(\tau')$, $b = b'$, and the natural numbers i, i' (upper bounded by the proof's length) satisfy $i > i'$. $\prec$ is the lexicograhic product of four well-founded relations; we use its induction principle to prove soundness.

Regarding completeness (the $\Rightarrow$ implication), the proof is constructive: to establish $\mathcal{S} \Vdash f$ it suffices to find a PSP $\mathcal{I}$ with same signature Π, S as $f \triangleq l \lhd r$ such that $\mathcal{I} \wedge (\lambda(\_ : \Pi).\phi) \Rightarrow \bot$, $l \Rightarrow (\mathcal{I} \vee r)$, and $\Delta_{f'}(\mathcal{I}) \Rightarrow (\mathcal{I} \vee r)$ for all $f' \in \mathcal{S}$. One can systematically construct such a PSP for all formulas f such that $\mathcal{S} \models f$ (although other PSPs than the systematically constructed one are typically simpler to use in practice). Then, a certain sequence of rule applications (excluding the compositionality rule [Cmp]) in the proof system establishes $\mathcal{S} \Vdash f$.

Hence, in order to prove any valid RL formula users "only" need to produce an appropriate PSP $\mathcal{I}$. Using the [Cmp] rule is an optimisation: it calls the proof system on a smaller specification $\mathcal{S}'$, for proving auxiliary formulas introduced during the proof, and for which simpler PSPs $\mathcal{I}'$ can typically be found.

4 Potential Applications and Future Work

We expect to use the envisioned Coq-based environment for verifying programs. Some work remains ahead before we reach that point: the current implementation of the Coq proof system needs to be enhanced with parameterised state predicates (currently, it only has state predicates), the compositionality proof rule needs to be included, and the soundness and completeness proofs need to be redone.

References

1. A. Ştefănescu, Ş. Ciobâcă, R. Mereuţă, B. Moore, T. F. Şerbănuţă, and G. Roşu. All-path reachability logic. In *RTA 2014*, Springer LNCS 8560, pages 425–440.
2. Ş. Ciobâcă and D. Lucanu. A Coinductive Approach to Proving Reachability in Logically Constrained Term Rewriting Systems. In *IJCAR 2018*, to appear.
3. The Coq proof assistant reference manual. `http://coq.inria.fr`.
4. V. Rusu, M. Hauspie and G. Grimaud. Proving partial-correctness and invariance properties of transition-system models. In *TASE 2018*, to appear. Available at `https://project.inria.fr/rlase`.
5. The $\mathbb{K}$ semantic framework. `http://www.kframework.org`.
6. A. Arusoaie, D. Nowak, D. Lucanu, and V. Rusu, A Certified Procedure for RL Verification. In *SYNASC'17*, IEEE.

Machine Learning and Formal Methods or *the Ballad of East and West*

Ruxandra Stoean

Department of Computer Science, University of Craiova, Romania
rstoean@inf.ucv.ro

Abstract. Two fields - one bringing along mathematical rigorous principles to assure correctness in the operation of computational systems, the other generating autonomous intelligent behaviour for computing devices from data in real-world scenarios - have long followed separate paths. However, researchers from both sides have recently begun to understand that the two areas need to work together in order that each of them completely fulfills its goals. The current study aims to outline the points where one field may benefit from the support of the other and to present the current progress in these directions, eventually pointing to further possible combinations of strengths.

Keywords: machine learning · formal methods · program synthesis · theorem proving · deep learning · safety assurance.

1 Introduction

Formal methods (FM) have constantly offered the theoretical means to guarantee a faultless behaviour of computer software and hardware. Model checking and theorem proving are among their best-known artifacts. At the other end, machine learning (ML) has lately succeeded to achieve the creation of highly intelligent and increasingly independent computer systems. These are able to efficiently assist people in everyday duties as well as the experts in critical decision-making tasks. Autonomous car driving, computer-aided medical diagnosis are among its most acclaimed achievements.

The two have fields have so far taken successful separate roads. Or as Rudyard Kipling would have put it: ML is ML, FM are FM, and *never the twain shall meet*. Nevertheless, each of them would benefit from the strengths of the other, as researchers from both domains have recently come to acknowledge. Accordingly a special seminar entitled Machine Learning and Formal Methods was organized in the summer of 2017 within the Dagstuhl event [1] and in July 2018 the Summit on Machine Learning Meets Formal Methods [2] is entirely dedicated to the intensification of this collaboration. The aim of this contribution is therefore to put forward the emerging directions for this hybridization and

[1] www.dagstuhl.de/en/program/calendar/semhp/?semnr=17351
[2] www.floc2018.org/summit-on-machine-learning/

to identify the current open problems, in order to additionally contribute to bringing together the researchers working in FM with scientists performing ML.

From the ML perspective, whereas its latest accomplishments are indisputable in terms of the effective decision support, the lack of the safety guarantee within the use of its models poses a big question mark for the human counterpart. FM have always been the guardians of correctness for standard systems. But ML models often act far from a standard manner as well as in a non-deterministic environment and this gives rise to a challenge for FM. From the FM standpoint, learning from available data can aid in inductive program synthesis (i.e. automatically learning to program from a large number of examples) and automated proof generation (i.e. automatically learning to create or correct proofs from many given instances).

2 Formal Methods for Machine Learning

The latest star of ML is unquestionably deep learning (DL). Its capabilities for autonomous feature extraction and decision, together with the rise of big data, have made it present with unbeatable solutions in all fields of real-world applications e.g. from games outscoring the world champions and self-driving cars to devices supporting the medical decision. The deep neural network (DNN) systems are acting as or even surpassing humans, however one pressing issue regarding their conduct and decision still remains, which renders humans reluctant to trusting and using them. It is the important and recently disputed aspect regarding the correctness of their behaviour as well as the safety and ethics guarantee for their actions. Since all these concerns are unarguably vital in letting the intelligent machines act among us, the corresponding research effort has been considerably large in this direction.

One currently observed threat lies in the existence of adversarial examples that heavily disrupt the decision process of the DNN. These are small perturbations conducted on an input sample (e.g. typically an image) such that it gets misclassified by the system. This is a critical problem for self-driving vehicles, to just name an example. In this context, literature entries from the last years make significant steps toward the safety verification of DNN. The study in [5] develops a discrete framework that defines for a sample point a region consistent with its established class, as well as a set of manipulations that describe the allowed perturbations for the example that do not lead to changes into its label within that region. This approach is then applied layer-wise. The methodology is able to perform both the safety verification that no misclassifications occur at any layer level and falsification, where adversarial instance are used for fine-tuning the system. The paper of [4] presents itself as being able to handle more complex DL architectures. Towards this purpose, it uses a formal interpretation for the DNN, namely: an abstract numerical domain of logical formulas that represent shapes, an abstract input to hold all perturbed initial samples, abstract layers for the network and an abstract output that approximates the real outcome.

FM can also complement the ML-designed devices in specifying, verifying and controlling that they obey moral principles of acting. An ethical reasoning framework for an autonomous aircraft was defined in [3] through abstract ethical principles, policies and ordered rules of acting depending on the context. A formal verification that the designed model obeys the specified ethics was not straightforward, as a linear temporal logic over the possible plans cannot be specified in a system acting in an environment that is not deterministic. Model checking is consequently performed at the level of ethical principles. A complex formal verification of the manner of navigation and avoidance of obstacles by robots (such as cars) is performed in [7]. Safety, defined as the absence of a collision, is considered in multiple scenarios: having static obstacles, avoiding collisions but allowing for passive impact by other objects when the robot is stationary, with passive friendly safety when the previous type of impact is also avoided by stopping responsibly and passive orientation safety, when collision to other obstacles is prevented only for the areas surveyed by the sensors of the robot. The continuous motion of the robot given through differential equations and its discrete control actions are combined in a hybrid system, while differential dynamic logic performs the formal verification of the criteria defining safety.

3 Machine Learning for Formal Methods

Two main goals of computer scientists, i.e. to have computers write programs and give proofs by themselves, have been tackled by traditional search methodologies over time. ML has also lately attempted to find solutions for this task.

The simplest solution provided by a ML architecture would be to emulate the program that maps the input to the desired output based on a large collection of paired examples, in a classification-like formulation, without providing any line of code. But the main provocation is not to find a black box model that substitutes the program, but make the artificial system generate the program. The big data-based learners (namely DL) can aid in performing string transformations as done by MS Excel as well as generating the underlying program. This is achieved first by learning the input-output relationship from a large set of examples with a neural model and secondly by expanding partial programs through another DNN approach [8]. DL can alternatively support classical search algorithms by 'pre-digging' the optimal program parts, based on numerous previously seen examples of input/output data together with the corresponding program. A deep recurrent DNN can be used to predict the most likely programming parts (or labels, thinking again in terms of a classification formulation) in the form of probabilities, such as in the 2017 DeepCoder framework [2]. This DNN output can further help a traditional search technique in constructing the underlying program, based on those functions with a higher probability of appearance in its code. The limitation of this deep learning approach for program synthesis is given however by the possibility of application only to domain-specific languages (DSL) that contain a maximum of 5 instructions, with solely 10 first-order and 5 higher-order possible functions.

An alternative for attempting to address more elaborate program instances has been given by the evolutionary side of ML dealing with the generation of computer programs, i.e. genetic programming (GP). The method is shown to evolve more complex functions with variables, if-then instructions and loops for exact integer program synthesis in [9]. However, its output in prefix notation still needs to be further translated into a comprehensible program format. Moreover, GP performs an heuristic search, can get blocked into local optima, representation for the individuals can be difficult to decide and the approach is slow as all evolutionary algorithms for complex tasks. DL and GP can nevertheless complement each other for the task of program synthesis. Similar to [2], a DNN can now generate for a new input-output pair the probable program functions related to a chosen GP representation. The GP (alone or augmented by a classical search approach from programming languages) can further take those into account when evolving an optimal program. Or, as in [8], the DNN can expand partial programs created in the GP style.

As concerns automated theorem proving, there already exist several large libraries holding mathematical statements together with proofs given in a way that is understandable by computers. There are also acknowledged automated theorem provers (ATP) that are able to obtain a proof, given however an a priori selection of the useful premises for this. Given the sufficiently extensive available collections and its intrinsic feature extraction mechanisms, DL can be again the most appropriate choice for premise selection. Two such current approaches are represented by DeepMath [1] and the DL-guided proof search in [6]. The mathematical statements are encoded differently, as either first-order logic strings of variable length or as a fixed vector of symbols. Conjectures are fed to the DL (recurrent DNN, convolutional DNN or long-short term memory networks) together with given proofs, and the model outputs for each premise the probabilities of further being important in the proof of an ATP system. Following these results, future steps should be made into a higher level representation, deeper architectures and better time efficiency.

4 Conclusions

FM and ML are each leading salient fields of computer science, the former concerned with theoretical soundness, the latter targeting pioneering applications. Nevertheless, FM researchers can aid in providing model checking for the new applications of DL that are better able to handle the non-deterministic nature of the environment. On the other hand, ML scientists can help in tailoring further DL approaches for program synthesis and theorem proving on the base of the already available large collections of related data, with the aim of reaching a higher complexity in the program/proof representation and a more efficient running time. Consequently, although of distinct nature, or in fact due to this contrast, the two areas can greatly complete each other in the present and future of computer science. Or, as the ballad also ends, there should be *neither East nor West, Border, nor Breed, nor Birth.*

References

1. Alemi, A.A., Chollet, F., Een, N., Irving, G., Szegedy, C., Urban, J.: Deepmath - deep sequence models for premise selection. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. pp. 2243–2251. NIPS'16, Curran Associates Inc., USA (2016), http://dl.acm.org/citation.cfm?id=3157096.3157347
2. Balog, M., Gaunt, A.L., Brockschmidt, M., Nowozin, S., Tarlow, D.: Deepcoder: Learning to write programs. In: Proceedings International Conference on Learning Representations 2017. OpenReviews.net (Apr 2017), https://openreview.net/pdf?id=rkE3y85ee
3. Dennis, L., Fisher, M., Slavkovik, M., Webster, M.: Formal verification of ethical choices in autonomous systems. Robotics and Autonomous Systems **77**, 1 – 14 (2016). https://doi.org/https://doi.org/10.1016/j.robot.2015.11.012, http://www.sciencedirect.com/science/article/pii/S0921889015003000
4. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: Proceedings IEEE Symposium on Security and Privacy (2018)
5. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification. pp. 3–29. Springer International Publishing, Cham (2017)
6. Loos, S.M., Irving, G., Szegedy, C., Kaliszyk, C.: Deep network guided proof search. CoRR **abs/1701.06972** (2017), http://arxiv.org/abs/1701.06972
7. Mitsch, S., Ghorbal, K., Vogelbacher, D., Platzer, A.: Formal verification of obstacle avoidance and navigation of ground robots. The International Journal of Robotics Research **36**(12), 1312–1340 (2017). https://doi.org/10.1177/0278364917733549, https://doi.org/10.1177/0278364917733549
8. Parisotto, E., Mohamed, A., Singh, R., Li, L., Zhou, D., Kohli, P.: Neuro-symbolic program synthesis. CoRR **abs/1611.01855** (2016), http://arxiv.org/abs/1611.01855
9. Weise, T., Wan, M., Tang, K., Yao, X.: Evolving exact integer algorithms with genetic programming. In: 2014 IEEE Congress on Evolutionary Computation (CEC). pp. 1816–1823 (July 2014). https://doi.org/10.1109/CEC.2014.6900292

On coalgebras and 3/2-institutions

Adriana Balan

Department of Mathematical Methods and Models
University Politehnica of Bucharest, Romania
adriana.balan@mathem.pub.ro

Abstract. In a previous work, we have constructed a morphism of institutions to encode the connection between set-based coalgebras and poset-based coalgebras, which however does not fully capture the order-enriched features of the above connection. This abstract proposes a way of overcome the drawback of this morphism of institutions. To this aim, the talk will focus on signatures and models of the desired coalgebraic logic 3/2-institution.

Keywords: coalgebra · institution · partial function

The theory of institutions, introduced by Goguen and Burstall in [8], provides a categorical abstract framework formalising logical systems - syntax, semantics and satisfaction relation. Recently, Diaconescu introduced the concept of 3/2-institutions [6,7] in order to formalise Goguen's computational account of conceptual blending [9], with emphasis on partiality of signature morphisms and its syntactic and semantic consequences.

In computer science, coalgebras are models for state based systems. The carrier set represents the state space and the structure map gives the one-step transition to successor states, but also encodes various types of observations. Notorious examples are the Kripke frames of modal logic, the labelled transition systems, the deterministic automata of formal language theory, or the Markov chains used in statistics. Coalgebraic logic combines coalgebra and modal logic to study logics of systems in a uniform and modular style.

It is known that coalgebras based on sets and their coalgebraic logic can be formalised within the theory of institutions (see [1] and references therein). Coalgebras with extra structure also deserve attention: within Kleisli categories for monads, trace semantics can be captured, while coalgebras whose carrier set is an Eilenberg-Moore algebra for a monad are e.g. operational models of structural operational semantics. Of a totally different nature are topological and/or ordered coalgebras, among which we mention descriptive general frames and ordered automata. The ordered coalgebras (a.k.a. poset-based coalgebras) and their logic given by monotone predicate liftings can also be captured within institutions; this was done in [1], where the connection between set-based coalgebras and poset-based coalgebras was exhibited as a morphism of institutions. However, this morphism of institutions did not succeed to capture the full (ordered enriched) strength of our previous results on positive coalgebraic logic of [2]. Taking into account that 3/2-institutions are developed in an ordered setting [6,7], a natural question would be whether they can remedy the above missing features.

The present talk will only focus on signatures and models of the desired coalgebraic logic 3/2-institution. Recall the institution of coalgebraic logic $\mathsf{CoalgLog}_{\mathsf{wpb}}$ introduced in [1]:

- A signature is an endofunctor T on Set, the category of sets and functions, which preserves weak pullbacks, and a morphism of signatures $T \to T'$ is a weakly cartesian natural transformation $T' \to T$ (notice the change of direction!)
- A T-model is a T-coalgebra $X \xrightarrow{x} TX$ and a morphism of models is a coalgebra map.
- The sentences are given by the logic of all finitary predicate liftings (i.e., take the usual propositional logic, and add modalities from all finitary predicate liftings), and the induction principle gives translation of sentences.
- Finally, the semantics of the logic provides the satisfaction relation. For a detailed account we refer to [1].

It is well-known that set-endofunctors which preserve weak pullbacks have a well-behaved relation lifting [3]. In particular, they lift to the categories of sets and partial functions [5], respectively sets and relations [3,10] and as such, the associated categories of coalgebras can account for richer behaviour, like trace semantics, or cpo-enrichment and initial algebra - final coalgebra coincidence [11]. One step further, using the 2-categorical abstract machinery exhibited in [4], we shall see how to configure coalgebras (i.e. models of the 'future' 3/2-institution) and functors between their categories in the non-deterministic framework of [6].

References

1. A. Balan, A. Kurz, and J. Velebil. An institutional approach to positive coalgebraic logic. *J. Logic Comput.* 27(6):1799–1824 (2017).
2. A. Balan, A. Kurz and J. Velebil. Positive fragments of coalgebraic logics. *Logic. Meth. Comput. Sci.* 11(3:18):1–51 (2015).
3. M. Barr, Relational algebras. In: S. MacLane et al (eds.), *Reports of the Midwest Category Seminar IV*, Lect. Notes Math. 137:39–55, Springer, Heidelberg (1970).
4. J. Bourke. Codescent objects in 2-dimensional universal algebra. PhD Thesis, Univ. of Sydney (2010).
5. J. R. B. Cockett, S. Lack. Restriction categories I: categories of partial maps. *Theor. Comput. Sci.* 270(1-2):223–259 (2002).
6. R. Diaconescu. 3/2-Institutions: an institution theory for conceptual blending. arXiv:1708.09675 (2017).
7. R. Diaconescu. Generic partiality for $\frac{3}{2}$-institutions. arXiv:1711.04666 (2017).
8. J. A. Goguen and R. M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *J. Assoc. Comput. Mach.* 39(1):95–146 (1992).
9. J. Goguen. Mathematical Models of Cognitive Space and Time. In: D. Andler et al (eds.), *Reasoning and Cognition* (2006)
10. B. Jacobs. Trace Semantics for Coalgebras. *ENTCS* 106:167-184 (2004).
11. M. B. Smyth and G. D. Plotkin. The category theoretic solution of recursive domain equations. *SIAM J. Comput.* 11:761?783 (1982).

Automated dynamic deobfuscation of static obfuscated binaries

Vlad Constantin Craciun

Department of Computer Science
UAIC, IASI
vcraciun@info.uaic.ro

Keywords: cyber-threat · binary analysis · static analysis · dynamic analysis · obfuscation

1 Abstract

Binary analysis is a branch of program analysis, achievable through reverse engineering mechanisms and requires a very good cyber- threats skill set to reveal the hidden behaviors. Given the fact that in the last few years, the number of malign applications has significantly grown, researchers are forced to automate the process of extracting understandable pieces of code (not obfuscated) from mostly obfuscated applications, to reduce the time required to classify the binaries or to understand certain behaviors. We are already familiar with the fact that malware authors will make all efforts to impede or totally disable the reverse of their binaries, adding additional obfuscation layers. A threat scientist will do best in such cases when his efforts of digging through these obfuscation layers will not exceed a couple of minutes or hours at most. The obfuscation mechanisms, related to how seriously they affect the reversing process, can be divided into static and dynamic obfuscation. We will refer to static obfuscation as a post-processing mechanism of binaries through packing, encryption or code rearrangements (the execution will deobfuscate by itself the initial piece of code eventually) and to dynamic obfuscation as a function which is part of the execution itself (no more than a few deobfuscated code blocks are available at all time of execution). We must admit that static binary deobfuscation is inefficient with recent malign applications, but dynamic deobfuscation stands up to dig for deep encoded behaviors.

Obfuscation is an old problem which researchers tried to overcome, however it has so many faces that one cannot even think about until some concrete scenarios are given. Some of the work in this field deals with deobfuscation of hi-level pieces of source-code like java-script, flash, C#, Java, etc. for optimization reasons, while others try to deobfuscate binary traces in an attempt to help static symbolic execution or binary classification through static analysis means. There are still attempts to deobfuscate malign binaries to get hands on the core deobfuscated application which was packed by a known packer. This

type of research is presented in [1] by Jean-Yves Marion and Daniel Reynaud. The authors try to achieve unpacking using PIN for binary instrumentation and they succeed for certain known packers. Another relevant work in [2], where S. Hassen et al. propose a plugin for IDA Pro tool to deobfuscate and understand the communication protocol used by two binary threats back in 2009. Another research of Yoann Guillot and Alexandre Gazet in [3] deals with deobfuscation as a series of rewriting rules, based on the fact that the target code can be statically deobfuscated and we also have some similar work in [4] and [5]

In this research, we propose a way to automate the static deobfuscation of malign binaries which are obfuscated by uncommon packing or encryption algorithms. Standard deobfuscation mechanisms are not designed to handle a lot of cases but to target specific ones. We will study a case of static obfuscation which distributes the execution across multiple applications and memory areas. The binary is a well-known bot-net (Emotet) which obfuscates the execution flow using two different applications and a run-time built piece of code to hide its behavior. The deobfuscation purpose is not classification, but understanding how threat authors update the communication protocol, so we can replicate it to get fresh binary samples directly from bot-net owners cloud infrastructure. The deobfuscation's output is a fresh application which merges all the executable memory areas involved in execution process through a scripted virtual environment based on VIX API (VMWare). We also attempt to formalize the deobfuscation steps, such that our solution will possible scale to any malign binary using this approach (obfuscation by flow distribution). The presentation will start with the problems we are facing and will continue with an introduction to binary analysis and obfuscation mechanisms. Next we will see the advancement in this field and will try to formalize the problem and the solution in an attempt to scale the deobfuscation procedure to any type of application statically obfuscated. Finally we will draw some conclusions, problems and consequences of this research.

References

1. Jean-Yves Marion, Daniel Reynaud: Dynamic Binary Instrumentation for Deobfuscation and Unpacking November 18, 2009
2. H Sadi, P Porras, V Yegneswaran: Experiences in malware binary deobfuscation Virus Bulletin, 2010 - csl.sri.com
3. Guillot, Y., Gazet, A. J Comput Virol: Automatic binary deobfuscation, Springer-Verlag (2010) 6: 261. https://doi.org/10.1007/s11416-009-0126-4
4. S. K. Udupa, S. K. Debray and M. Madou, "Deobfuscation: reverse engineering obfuscated code," 12th Working Conference on Reverse Engineering (WCRE'05), 2005, pp. 10 pp.-. doi: 10.1109/WCRE.2005.13
5. B. Yadegari, B. Johannesmeyer, B. Whitely and S. Debray, "A Generic Approach to Automatic Deobfuscation of Executable Code," 2015 IEEE Symposium on Security and Privacy, San Jose, CA, 2015, pp. 674-691. doi: 10.1109/SP.2015.47

Search based Model in the Loop Testing for Cyber Physical Systems*

Ana Ţurlea[1], Felician Campean[2], and Raluca Lefticaru[3,1]

[1] Faculty of Mathematics and Computer Science and ICUB, University of Bucharest, Romania
[2] School of Engineering, University of Bradford, UK
[3] School of Electrical Engineering and Computer Science, University of Bradford, UK
ana.turlea@fmi.unibuc.com, f.campean@bradford.ac.uk, r.lefticaru@bradford.ac.uk

Testing at the model-in-the-loop level represents functional testing in early development phases in simulation environments. In this paper, we propose a search-based approach using a Genetic Algorithm (GA) to generate test cases for continuous controllers at the MiL level. We identify a set of requirements for the desired behaviour and search for test cases that break the requirements. We illustrate our approach on a cruise control system for a hybrid propulsion bicycle (e-Bike) case study [2]. The e-Bike system considered in this paper combines an electric drive system with regen capability with a conventional pedal drive. The hybrid system of the e-Bike affords a variety of ways of controlling the propulsion of the e-Bike, delivering enhanced control feature for the rider, including cruise control, smart pedal assist, and pedal recharge. We assume for our study that all parameters are observable through sensors, and aim to identify critical situations (e.g. lateral vehicle stability issues) when the CC needs to be switched off for safety reasons. Given the large number of variables, the state space control system is very large, and therefore exhaustive testing is not feasible. In this paper we consider as example of critical situation when the slip interval is greater than $20s$ (the amount of time, in seconds, when the bycicle slips).

The first step of the algorithm is the design of the requirements for the SUT, adding to the MiL different functions that measure the satisfiability of the requirements. Then, the genetic algorithm will generate input values for the parameters of the model, and uses the output of the simulation as objective function. GAs start with initialization of a population with random candidate solutions, evolve the population several times, until a solution is found or a stop condition is met. A solution represents a set of values that correspond to the input variables for the simulator and optimize the fitness function. The chromosome, $x = (x_1, x_2, ..., x_n)$, has fixed given length and it is composed of genes (values from given search space). Each variable has its definition domain (real values from given intervals) $\{D_1, D_2, ...D_n\}$. Our approach discretizes each D_i, choosing 16 values for each definition domain (ECU controllers commonly use lookup tables with grid dimension 16). The goal of discretization is to reduce

* Authors are supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-III-P4-ID-PCE-2016-0210.

the number of values a continuous variable assumes. For the e-Bike model, the simulator uses ten input variables and simulates the riding of the bike for 60 seconds. The simulation time is divided in 600 and, some parameters and metrics used in computing the objective function are recorded for each time moment. The genetic operators used in the algorithm are *Uniform Mutation*, *Single Point Crossover* and *Best Chromosome Selector*. When evaluating a chromosome, a simulation takes place and certain metrics are computed, for example the slip time, depending on the requirement that we want to check. In our experiments, the simulation computes the maximum slip interval for a simulation of riding the e-Bike for 60 seconds.

Using the Deap framework [1], we implemented in Python the algorithm and performed several experiments. In HeatMaps, each region has assigned the average value of the values of the points within that region [3]. The intervals of the region values are then mapped into different shades, generating shaded diagrams, that helped us to evaluate the results, coloring the diagrams such that the darkest cells represented regions in the search space containing points that were more likely to break the requirements. A HeatMap is effective if the region shades are stable and do not change on different runs of the algorithm. Comparing multiple generated HeatMaps we have been able to state that our algorithm is stable. To evaluate the discrete genetic algorithm we compared the results with the continuous genetic algorithm. We used the same configuration for the genetic algorithm, the only difference was the definition intervals for the input variables. After running both algorithms multiple times we computed the mean, the maximum and the minimum values for the fitness function (the same metric for both algorithms) using the last generated population for both algorithms. The aim of the algorithm is to maximize the maximum slip interval, generating worst case scenarios. So, comparing the results, we can state that the discrete algorithm finds better solutions.

In this paper, we proposed a search-based approach to automate generation of MiL level test cases for CPSs, using a discrete genetic algorithm. We evaluated our approach on a cruise control system for a hybrid propulsion bicycle case study. Our experiments showed that our approach automatically generates several test cases that indicate potential violations of the requirements at the MiL level, that can be further investigated by evaluating the test cases at the Hardware-in-the-loop level. As future work, we will improve the algorithm to produce populations with a better diversity between the individuals.

References

1. Distributed evolutionary algorithms in python, https://github.com/deap/deap
2. Lefticaru, R., Konur, S., Yildirim, U., Uddin, A., Campean, I.F., Gheorghe, M.: Towards an integrated approach to verification and model-based testing in system engineering (2017)
3. Matinnejad, R., Nejati, S., Briand, L., Bruckmann, T., Poull, C.: Automated model-in-the-loop testing of continuous controllers using search. In: International Symposium on Search Based Software Engineering. pp. 141–157. Springer (2013)