Working Formal Methods Symposium
1st International Workshop
FROM 2017

Bucharest, Romania, July 5-8, 2017

Informal Proceedings

Dorel Lucanu and Viorica Sofronie-Stokkermans (Eds.)

July 5-8, 2017

Foreword

This volume contains the papers presented at FROM 2017, Working Formal Methods Symposium, held on July 5-8 in Bucharest.

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

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

Areas and formalisms of interest include: category theory in computer science, distributed systems and concurrency, formal languages and automata theory, formal modelling, verification and testing, logic in computer science, mathematical structures in computer science, models of computation, semantics of programming languages.

Methods of interest include: model checking, deductive verification, automated reasoning and model generation, automated induction, symbolic computation.

Applications of interest include: program analysis, verification and synthesis of software and hardware, computational logic, computer mathematics, knowledge representation, ontology reasoning, deductive databases, uncertainty reasoning and soft computing.

The program of the symposium included invited lectures and regular contributions. There were 10 submissions. Each submission was reviewed by at least two program committee members. The committee decided to accept 7 papers. The program also includes 16 invited talks. The extended abstracts of all of them are included in this volume.

We thank all invited speakers for their high-level presentations and for making the first edition of the FROM series possible. We also thank the authors who submitted their presentations to FROM 2017 and the Program Committee for its contribution in the evaluation and selection process. Lastly, we would like to thank the Organizing Committee for all the local arrangements and for creating and maintaining the FROM 2017 website.

We are grateful to the Research Institute of the University of Bucharest (ICUB) for the generous financial support offered, to the Faculty of Mathematics and Computer Science of the University of Bucharest and to the Faculty of Computer Science of the Alexandru Ioan Cuza University of Iasi for their valuable support in organizing this event. We benefited from the invaluable assistance of the EasyChair system through all the phases of submission, evaluation, and production of the proceedings.

June 25, 2017
Bucharest

Dorel Lucanu
Viorica Sofronie-Stokkermans

Program Committee

Dorel Lucanu	Alexandru Ioan Cuza University
Victor Mitrana	University of Bucharest
Traian Florin Şerbănuţă	University of Bucharest
Viorica Sofronie-Stokkermans	University Koblenz-Landau
Gheorghe Ştefănescu	University of Bucharest

Organizing Committee

Andrei Arusoaie	Alexandru Ioan Cuza University
Denisa Diaconescu	University of Bucharest
Ioana Leuştean	University of Bucharest
Oana Peiu	ICUB representative
Ştefan Popescu	University of Bucharest
Ana Ţurlea	University of Bucharest

Table of Contents

Multiparty Session Types for Mobility

Bogdan Aman and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science
Blvd. Carol I no.8, 700505 Iaşi, Romania
bogdan.aman@iit.academiaromana-is.ro, gabriel@info.uaic.ro

Mobility provides to the software applications the ability to move between locations and devices during their execution. This means that a running application migrates from one location to another depending on the proximity of the user. A mobile application is closely related to process migration in distributed systems modelled by process calculi such as distributed π-calculus [2] and TiMo [1]. Process migration represents the capability of a running process to be relocated from a location to another one in order to access and communicate data locally.

We propose session types for mobility to describe uninterrupted sequences of migrations for processes. We use a simple version of distributed π-calculus to define session types for mobility. The novelty of this approach is that we point out sequences of migrations as global types, and investigate scenarios in which processes are required to follow such a sequence of migrations along several locations.

Typing locations with mobility types is sufficient to detect subtle errors in the implementation of mobile applications and protocols. We do not insist on typing the local communication inside locations as it has been proposed by specific session types [4]. Communication session types are used to reason over communicating processes and their behaviour by

abstractly representing the trace of the usage of the channels as a structured sequences of types. A detailed discussion and analysis of several versions of the π-calculi with session types is given in [5].

Our approach is in contrast with the typing system for distributed π-calculus [2] which uses types of the form $T@p$ and dependent type techniques. The syntax for processes is based on distributed π-calculus [2] and user-defined processes [3]. The operational semantics is given by a reduction relation for which we present the most important rules for migration and communication:

$$k[[go\ l(a).P \mid Q]] \mid l[[(a).R]] \to k[[Q]] \mid l[[P \mid R]] \quad (\textsc{Migrate})$$
$$l[[a!\langle v\rangle.P \mid a?(u).Q]] \to l[[P \mid Q\{v/u\}]] \quad (\textsc{Comm})$$

Rule (Migrate) describes the migration of processes between locations by using a port available at the destination location where the parties can communicate in order to establish necessary local interactions (by using this port). (Comm) is the standard communication rule derived from the π-calculus where the received value v replaces the existing free occurrences of u in Q.

The global types (ranged over by $G, G', \ldots$) describe the global behaviour of the system. For example, type $l \to l' : a.G'$ says that from location l a process migrates to location l' through the access port a, and at l' the process behaves accordingly to the mobility type described by G'.

Local types (ranged over by $T, T', \ldots$) describe the local behaviour of processes, acting also as a link between global types and processes. Type $a!.T$ represents the behaviour of sending

A Multi-Valued Framework for Coalgebraic Logics over Generalised Metric Spaces

Adriana Balan*

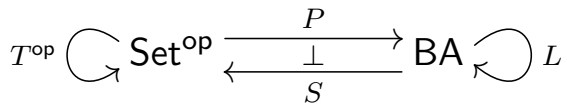
Department of Mathematical Methods and Models
University Politehnica of Bucharest, Romania
adriana.balan@mathem.pub.ro

It is by now generally acknowledged that the theory of universal coalgebra incorporates a wide variety of dynamic systems [18]. The classical study of their behaviour is based on qualitative reasoning – that is, Boolean, meaning that two systems (the systems' states) are bisimilar or not. This is best formalised within modal logics and the associated proof techniques. But in recent years there has been a growing interest in studying systems' behaviour in terms of quantity. There are situations where one behaviour is smaller than (or, is simulated by) another behaviour, or there is a measurable distance between behaviours in terms of real numbers, as it was done in [17,21]. It follows that a convenient framework for developing a theory that parallels the one of coalgebras over sets is given by coalgebras over $\mathscr{V}$-cat, the category of (small) enriched categories over a quantale $\mathscr{V}$. It has been recognised that $\mathscr{V}$-cat can be perceived as a category of "quasi-metric" spaces and nonexpanding maps, which subsumes both

* This work has been funded by University Politehnica of Bucharest, through the Excellence Research Grants Program, UPB–GEX, grant ID 252 UPB–EXCELENŢĂ–2016, research project no. 23/26.09.2016.

ordered sets and monotone mappings, and (generalised) metric spaces [12]. Several versions of many-valued modal logics for transition systems have been subsequently developed to incorporate such quantitative reasoning (e.g. [4,7,8,15,20]).

Recall that classical modal logics naturally fits in the uniform framework of (Boolean) coalgebraic logics [5], and that the latter can *abstractly* be described by the following diagram



where the functor P maps a set to the Boolean algebra of its subsets, while S associates to a Boolean algebra the set of its ultrafilters. There is a pair of endofunctors on top of this contravariant adjunction: T on sets giving the type of transition systems (coalgebras), and L on Boolean algebras whose algebras should be seen as the modal algebras of the logic of T-coalgebras. They are connected by a natural transformation $\delta : LP \to PT^{\mathsf{op}}$ which assigns to each modal operator its interpretation as a subset of acceptable successor states.

Therefore it seems natural to elaborate a similar uniform strong framework for reasoning about transition structures over $\mathscr{V}$-cat. The present talk will focus on the results obtained so far in this line of research.

Coalgebras over $\mathscr{V}$-cat. Formally, transition structures over (generalised) metric spaces are precisely the coalgebras for an endofunctor on $\mathscr{V}$-cat. But how to obtain such a functor? We have the plethora of Set-endofunctors as examples, hence the obviuous solution is to adapt them to quantale-enriched categories. Therefore, we shall explain how to extend functors $T : \mathsf{Set} \to \mathsf{Set}$ (and more generally Set-functors which

naturally carry a $\mathscr{V}$-metric structure) to $\mathscr{V}$-cat-functors $T^{\mathscr{V}}$: $\mathscr{V}$-$\mathsf{cat} \to \mathscr{V}$-$\mathsf{cat}$. This construction, called "$\mathscr{V}$-cat-ification", has been obtained by the author in [1], in collaboration with A. Kurz and J. Velebil. Using the density of the discrete functor $D : \mathsf{Set} \to \mathscr{V}$-$\mathsf{cat}$, we apply T to the "$\mathscr{V}$-nerve" of a $\mathscr{V}$-category, and then take an appropriate "quotient" in $\mathscr{V}$-cat. If T preserves weak pullbacks, then the $T^{\mathscr{V}}$ above can be obtained using Barr's relation lifting in a form of "lowest-cost paths" (see also [21, Ch. 4.3] and [10]). For example, the extension $\mathcal{P}^{\mathscr{V}}$ of the powerset functor $\mathcal{P}$ yields the familiar Pompeiu-Hausdorff metric, if the quantale is completely distributive.

Multivalued logical connection. The next step, following the well-established tradition in coalgebraic logics explained earlier (see also [14]), is to seek for a contravariant adjunction on top of which multi-valued coalgebraic logics is to be considered. This adjunction involves on one side a category of algebras Alg, and on the other side, a category of spaces Sp, obtained by conveniently restricting the adjunction $[-\mathscr{V}] \dashv [-, \mathscr{V}] : \mathscr{V}$-$\mathsf{cat}^{\mathsf{op}} \to \mathscr{V}$-$\mathsf{cat}$. Moreover, we look for $\mathscr{V}$ to live in both categories and to act as a "dualising" object, in the sense of [16]. A requirement for Alg is to be a variety[1] in the "world of $\mathscr{V}$-categories", at least monadic over $\mathscr{V}$-cat.

In *classical modal/coalgebraic logics* (no enrichment), this is achieved by taking Sp to be Set, and Alg to be the category of Boolean algebras, as detailed earlier. One step further, the case of the simplest two-elements quantale $\mathscr{V} = \mathbb{2}$ targets *positive modal/coalgebraic logics* (see [6,2] for a detailed exposition), from an order-enriched point of view, by choosing Sp to be

[1] This variety is not expected to be finitary unless $\mathscr{V}$ itself is finite.

the category of posets and monotone maps, and Alg to be the category of bounded distributive lattices DLat.[2] These in turn, are part of an adjunction $S' \dashv P' : \mathsf{Poset}^{\mathsf{op}} \to \mathsf{DLat}$ with P' taking uppersets and S' taking prime filters, or, equivalently, $P'X = \mathsf{Poset}(X, \mathbb{2})$ and $S'A = \mathsf{DLat}(A, \mathbb{2})$, where $\mathbb{2}$ plays the role of a "dualising object", being considered, depending on the context, either as a poset or as a distributive lattice.

In view of the above, the logical connection between Sp and Alg for $\mathscr{V}$-cat that we shall propose in the sequel starts from an adaptation of the Priestley duality [9]. The algebras of the logic will be built on distributive lattices endowed with a family of *adjoint pairs of operators* (shortly dlao($\mathscr{V}$)) indexed by the elements of the quantale $(r * - \dashv r \pitchfork - : A \to A)_{r \in \mathscr{V}}$, satisfying

$$1 * a = a \qquad (r \otimes r') * a = r * (r' * a)$$
$$0 * a = 0 \qquad (r \vee r') * a = (r * a) \vee (r' * a)$$

The morphisms between dlao($\mathscr{V}$) are distributive lattice maps preserving the adjoint operators. The resulting category, denoted $\mathsf{DLatAO}(\mathscr{V})$, is an algebraic category and will play in the sequel the role of Alg.[3] Each dlao($\mathscr{V}$) A becomes a $\mathscr{V}$-category [13] with $\mathscr{V}$-homs $A(a, a') = \bigvee\{r \in \mathscr{V} \mid r * a \leq a'\} = \bigvee\{r \in \mathscr{V} \mid a \leq r \pitchfork a'\}$, which is finitely complete and cocomplete [19], and each dlao($\mathscr{V}$)-morphism is also a $\mathscr{V}$-functor. Consequently, $\mathsf{DLatAO}(\mathscr{V})$ is a subcategory of $\mathscr{V}$-cat, monadic if the quantale $\mathscr{V}$ satisfies certain conditions.

By adapting the arguments in [9], we see that the dual category to

[2] Which is a finitary *ordered* variety [3].

[3] Observe that for $\mathscr{V} = \mathbb{2}$ the operators are trivial and we regain DLat.

Knowing Correlations

Alexandru Baltag

ILLC, University of Amsterdam
A.Baltag@uva.nl

Informationally, a question can be encoded as a variable, taking various values ("answers") in different possible worlds. If, in accordance to the recent trend towards an interrogative epistemology, "To know is to know the answer to a question" [6], then we are lead to paraphrasing Quine's motto: *To know is to know the value of a variable* [5].

In a paper with this title [2], presented at "Advances in Modal Logic" 2016, I introduced a logical account of this form of knowledge, by building on the work of Plaza [4], and Wang and Fan [7] on capturing 'knowledge de re' (knowledge of an object, "knowledge what") over Kripke models. In its turn, this research is motivated by work in Security on knowledge of keys and passwords. A *possible world* in these models is the same as a first-order assignment, i.e. an assignment $w : Var \to D$ of values (from some fixed domain D) to variables $x \in Var$. Plaza's semantics for 'knowledge de re' is given by putting: $w \models K_a x$ iff $\forall v \sim_a w\ (v(x)) = w(x))$. This is a natural analogue of the usual semantics of "knowledge that" in epistemic logic: an agent knows the value of x if *that value is the same in all her epistemic alternatives*. This is the sense in which may say that (s)he knows a password or an encryption key.

However, questions are never investigated in isolation: we answer questions by reducing them to other questions. This means that the proper object of knowledge is uncovering *correlations* between questions. *To know is to know a functional dependence between variables*. In the same paper, I investigated (and completely axiomatized) a Logic of Epistemic Dependency, that can express knowledge of functional dependencies between (the values of) variables, as well as dynamic modalities for learning new such dependencies. This dynamics captures the widespread view of knowledge acquisition as a process of learning correlations (with the goal of eventually tracking causal relationships in the actual world). This research is also clearly related to the study of dependencies in Database theory: indeed, the well-known Armstrong axioms [1] for dependency are provable in the Logic of Epistemic Dependency.

For a string $\mathbf{y} = (y_1, \ldots, y_n)$ of variables and a variable x, the *epistemic dependency formula* $K_a^{\mathbf{y}} x$ says that an agent knows the value of X *conditional* on being given the values of variables $\mathbf{y}$. The semantics is the obvious generalization of the above clause: if we use the abbreviation $w(\mathbf{y}) = v(\mathbf{y})$ for the conjunction $w(y_1) = v(y_1) \wedge \ldots w(y_n) = v(y_n)$, then we put

$$w \models K_a^{\mathbf{y}} x \quad \text{iff} \quad \forall v \sim_a w\ (w(\mathbf{y}) = v(\mathbf{y}) \Rightarrow v(x) = w(x))\,.$$

In words: an agent knows x given $\mathbf{y}$ if the value of x is the same in all the epistemic alternatives that agree with the actual world on the values of $\mathbf{y}$.

In this paper I propose an extension of the Epistemic Dependence Logic, called the Logic of Correlations (LC). Essentially, it uses van Benthem's Generalized-Assignment Semantics of First-Order Logic (on *dependency models*, also known as general assignment models). While the usual semantics of FOL allows all possible assignments of variables to objects, a dependency model restricts the range of available assignments to a given subset. Dependency models were proposed by van Benthem as a model for capturing dependencies and correlations between variables. It is known that FOL with this generalized semantics becomes decidable [3]. Unfortunately, the resulting logic is so weak that it can no longer express variable dependencies in an explicit manner (though the model still implicitly captures them).

Our Logic of Correlations proposes a remedy to this limitation, by adding to the syntax functional terms of the form $x(\mathbf{y}, \phi)$, denoting *the unique (value of) x determined by (the values of) $\mathbf{y}$ (in the current assignment) and by condition ϕ* (if such an x exists). For a given variable assignment w, this term has a denotation only if it is indeed the case that the current values $w(\mathbf{y})$ of variables $\mathbf{y}$, together with the information that ϕ, uniquely determine a specific value of x. So the precondition for denoting is a statement $x(\mathbf{y}, \phi)\downarrow$ (definable in our logic), saying that $\mathbf{y}$ *uniquely determines x conditional on ϕ*. When thinking of y as an agent (or group of agents), we may alternatively write this precondition as an "epistemic" statement $K^{\phi}_{\mathbf{y}}x$, saying that *agent (group) $\mathbf{y}$ 'knows' (the value of) x conditional on ϕ*. (This was the interpretation I adopted when introducing Epistemic Dependence Logic in my AiML paper.) By taking ϕ to be any tautological formula $\top$, we obtain a formula $K_{\mathbf{y}}x$ that captures a 'local' version of the so-called dependence atoms from Dependence Logic: all assignments (available in the model) that agree with the current assignment on $\mathbf{y}$ also agree with it on x. The usual "global" dependency statement $=(\mathbf{y}, x)$ from Dependence Logic (saying that every two available assignments that agree on $\mathbf{y}$ also agree on x) can be defined using the local version (as $KK_{\mathbf{y}}x$, where the first K is just the global modality, i.e. the universal quantifier over all the available variables).

Our first important result in this paper is that the Logic of Correlations is decidable. We show this by using the quasi-model method pioneered by van Benthem (and closely related to the mosaic method of Andreka and Nemeti). We also look at its embedding into fragments of FOL with the standard semantics. It is well known that FOL with generalized assignment semantics can be embedded in the Guarded Fragment (of FOL) with standard semantics, and moreover that the same quasi-model techniques can be used to show the decidability of the Guarded Fragment (Andreka et alia), and of further extensions (the so-called Packed Fragment). So it is natural to ask whether a similar embedding can be found for our Logic of Correlations.

The second main contribution in this paper is to embedd LC into a decidable extension of the Guarded Fragment (on standard first-order models), which we call the Guarded Definite-Description Fragment. This is obtained by adding *guarded definite descriptions* of the form $!x_i.\exists\mathbf{y}(P\mathbf{xy} \wedge \phi\mathbf{xy})$ (where x_i is one of the x's), denoting *the unique x_i satisfying the guardedly-quantified formula*

$\exists \mathbf{y}(P\mathbf{xy} \wedge \phi\mathbf{xy})$ (if such an x exists). Our quasi-model technique can be easily adapted to prove decidability of this fragment.

A natural open question is whether these decidability results can be further extended, say by adding "packed definite descriptions" to the Packed Fragment, and use the mozaic method. We conjecture that the answer is yes. Further fixed-point extensions are also worth considering. It is known that the extensions of Guarded and Packed fragments with monotonic fixed points are still decidable. What about the fixed-point extensions of the logics considered in this paper?

References

1. W. J. Armstrong, Dependency structures of data base relationships, IFIP Congress, vol. 74, pp. 580-583, 1974.
2. A. Baltag, To Know is to Know the Value of a Variable. In Proceedings of AiML '16 (Budapest, 2016), *Advances in Modal Logic* vol. 11, pp. 135-155, College Publications 2016.
3. H. Andreka, J. van Benthem, and I. Nemeti. Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic, vol. 27, nr. 3, pp 217274, 1998.
4. J. Plaza, Logics of public communication, Synthese, vol. 158, pp. 165-179, 2007.
5. W.V. Quine, On What There Is, Review of Metaphysics, vol. 3, nr. 5, pp 21-36, 1948.
6. J. Schaffer, Knowing the answer, Philosophy and Phenomenological Research, vol. 75, nr. 2, pp 383-403, Wiley Online Library 2007.
7. Wang, Y. and Fan, J., Conditionally knowing what, Advances in Modal Logic'14, pp. 569-587, College Publications, 2014.

A Formal Approach to Computational Creativity

Claudia Elena Chiriţă and José Luiz Fiadeiro

Department of Computer Science, Royal Holloway University of London, UK
claudia.elena.chirita@gmail.com, jose.fiadeiro@rhul.ac.uk

Computational creativity is a subdomain of AI developed in the last decades to explore the potential of computational systems to create original artefacts and ideas. Overlapping with the broader field of cognitive science, it encompasses "the philosophy, science and engineering of computational systems which [...] exhibit behaviours that unbiased observers would deem to be creative" [3].

In this work, we discuss computational creativity from an algebraic point of view, showing how we can give a mathematical formalization of creative systems and their components. We start from the tenet that creativity can be seen in essence as the identification or location of new conceptual objects in a conceptual space [1], and present creative systems in an institutional setting built around the notion of concept. We adopt Goguen's understanding of a conceptual space [8,7] as an algebraic specification [11], and develop our study based on institution theory [9], which formalizes logical systems by capturing their syntax, semantics, and the satisfaction relation between them in an algebraic manner. This allows us to maintain the generality of previous descriptions of creative systems [12], and at the same time to use formal definitions for concept abstraction, concretization, and blending [7] that enable reasoning about creative processes.

We first define creative systems by means of many-valued specifications written over institutions endowed with residuated lattices as truth spaces [2], and of abstract strategies for the discovery and evaluation of concepts based on the notion of graded semantic consequence between specifications [4].

We then focus on a subclass of creative systems modelled as complex dynamic systems. The idea that creative processes can be explained in terms of the evolution of complex systems has been extensively studied in works connecting creativity with biology-inspired algorithms (swarm optimization, genetic algorithms, etc.), nonlinear dynamical systems of equations and multi-agent systems. We continue this line of contributions by investigating a new connection between service-oriented architectures viewed as complex systems [5] and improvisation performances.

In service-oriented computing [6], software applications evolve dynamically to meet their goals through discrete reconfigurations triggered by the need for an external resource and realized through a three-fold process of discovery, selection and binding of service modules. A salient feature of these systems is the unpredictability of their actual architectural configuration (i.e. the entities and connectors through which entities exchange information) at design time. This is due to their openness to reconfiguration: binding new services could trigger subsequent processes of discovery, selection, and binding. The dynamic aspect of these reconfigurations is endogenous, intrinsic to the systems; their

evolution is not driven by external factors such as the change of the environment, but originates from the design of the components themselves. We argue that this gives rise to an emergent behaviour similar to what has been observed for computational creative systems that model, for example, improvisations.

In modelling creative processes as service-oriented computing, we regard concepts as modules and concept discovery as service discovery. In this context, we evaluate the usefulness of a concept through the mechanism of service selection, and recast concept blending in terms of service binding. This permits us to study properties of improvisation processes within the framework of service-oriented systems: by analysing service repositories from the perspective of quality-of-service profiling we can examine the system's reliability (to what extend the user's expectations of delivering an artefact can be met?), determine the reachability of some concepts (which modules are never or seldom used during the execution of an/any application?), identify relationships between concepts, such as redundancy, within the universe of concepts involved in the process (does the presence of certain service modules prohibit the use of other modules?), and assess the cohesion of a performance or creative act (via the soundness of resolution for many-valued logic programming).

While most of the current research in computational creativity seems to adopt a connectionist view on cognitive and in particular on creative processes, our approach adheres to the computational theory of mind. This opens naturally a series of questions related to the everlasting paradigm dispute between the subsymbolic and symbolic views on the philosophy of mind that one should not ignore. To answer the concern on the origin of concept specifications, we investigate the connection between the abstract representations of concepts in neural networks and their algebraic definition [10]. Establishing a formal relation between the two would provide a solution: the automatic learning from examples could alleviate the user's task of writing complex specifications of concepts.

References

1. Margaret A Boden. *The creative mind: Myths and mechanisms*. Psychology Press, 2004.
2. Claudia Elena Chiriţă, José Luiz Fiadeiro, and Fernando Orejas. Many-valued institutions for constraint specification. In *Fundamental Approaches to Software Engineering*, volume 9633 of *LNCS*, pages 359–376. Springer, 2016.
3. Simon Colton and Geraint A. Wiggins. Computational creativity: The final frontier? In Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, editors, *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 21–26. IOS Press, 2012.
4. Răzvan Diaconescu. Graded consequence: an institution theoretic study. *Soft Comput.*, 18(7):1247–1267, 2014.
5. José Luiz Fiadeiro. The many faces of complexity in software design. In *Conquering Complexity*, pages 3–47. Springer, 2012.

6. José Luiz Fiadeiro and Antónia Lopes. A model for dynamic reconfiguration in service-oriented architectures. *Software and System Modeling*, 12(2):349–367, 2013.
7. Joseph Goguen. An introduction to algebraic semiotics, with application to user interface design. In *Computation for metaphors, analogy, and agents*, pages 242–291. Springer, 1999.
8. Joseph A. Goguen. What is a concept? In Frithjof Dau, Marie-Laure Mugnier, and Gerd Stumme, editors, *Conceptual Structures: Common Semantics for Sharing Knowledge, 13th International Conference on Conceptual Structures, ICCS 2005, Kassel, Germany, July 17-22, 2005, Proceedings*, volume 3596 of *Lecture Notes in Computer Science*, pages 52–77. Springer, 2005.
9. Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
10. Michael John Healy and Thomas Preston Caudell. Ontologies and worlds in category theory: implications for neural systems. *Axiomathes*, 16(1):165–214, 2006.
11. Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Springer, 2012.
12. Geraint A. Wiggins. A preliminary framework for description, analysis and comparison of creative systems. *Knowl.-Based Syst.*, 19(7):449–458, 2006.

Scheduled Migration in Distributed Systems

Gabriel Ciobanu

Romanian Academy, Institute of Computer Science,
Blvd. Carol I no. 8, 700505 Iaşi, Romania
gabriel@info.uaic.ro

Coordination of mobile agents in distributed systems takes into account time scheduling, access to available resources, safe interaction among processes. Mobile and concurrent processes were described essentially in the π-calculus [18], a formalism working with communicating mobile processes, where the mobility is expressed by sending certain channel names as messages to other processes. The distributed π-calculus [17] is a variant of the π-calculus using explicit locations, explicit migration, replication and local communication among processes. After introducing and studying a timed extension of the distributed pi-calculus [14], we introduced a simple prototyping high-level programming language called TIMO to describe mobile agents using specific features as timed migration and timed communication in distributed networks [9]. TIMO is bridging the gap between the existing theoretical approaches and forthcoming realistic languages for bounded-time migrating agents in distributed systems. To understand the problems solved by such a language, just imagine the difficulties to find an optimal trip from Iaşi (Romania, 400km from Bucharest) to Kingston (Canada, half-way between Montreal and Toronto) according to several constraints and requirements on timing, routes, connections and price.

The standard notion of bisimilarity is extended in [5] to take into account the timed transitions and multisets of actions, and then to TIMO in [4]. Behavioural equivalences are based on the observable transitions of processes rather on their states (as in timed automata and timed Petri nets). The relationship between timed mobility in TIMO and Petri nets is presented in [10].

Several variants of TIMO were developed during the last years: a version with access permissions for mobile agents given by a type system [11], a real-time version RTIMO [1], a probabilistic extension PTIMO [15], a version perTIMO with safe access permissions [12]. Inspired by TIMO , a flexible software platform was introduced first in [7] and presented then in [8] to support the specification of agents allowing timed migration in a distributed environment.

Interesting properties of distributed networks described by TIMO refer to various time constraints over agents migration and communication, bounded liveness and optimal reachability. A verification tool called TIMO@PAT was developed by using an extensible platform for model checkers [16]. A probabilistic temporal logic called PLTM was introduced in [15] to verify properties of PTIMO processes making explicit reference to specific locations, and using temporal constraints over local clocks and multisets of actions. A formal relationship between RTIMO and timed automata allows us to use the model checking capabilities provided by the software tool UPPAAL [2]. TIMO was used to describe

a railway control system, and then use a new behavioural congruence over real-time systems (named strong open time-bounded bisimulation) to check which behaviours are closer to an optimal and safe behaviour [3]. In [6] it is defined a general framework for reasoning about systems specified in TiMo by using the Event-B modelling method as the target for translating TiMo specifications. Then the Rodin platform supporting Event-B is used to verify system properties using the embedded theorem-provers and model checkers.

In [13] it was developed a new semantic model for TiMo by using rewriting logic and strategies with the aim of providing a foundation for tool support. In particular, strategies are used to capture the locally maximal concurrent step of a TiMo specification. This model is then extended with access permissions in order to develop a new semantic model for perTiMo . These semantical models are formally proved to be sound and complete with respect to the original operational semantics on which they were based.

References

1. B. Aman, G. Ciobanu. Real-Time Migration Properties of rTiMo Verified in Uppaal . SEFM, *Lecture Notes in Computer Science* **8137**, 31–45 (2013).
2. B. Aman, G. Ciobanu. Timed Mobility and Timed Communication for Critical Systems. *Lecture Notes in Computer Science* **9128**, 146–161 (2015).
3. B. Aman, G. Ciobanu. Verification of Critical Systems Described in Real-Time TiMo . *Int'l Journal on Software Tools for Technology Transfer*, 1–14 (2016).
4. B. Aman, G. Ciobanu, M. Koutny. Behavioural Equivalences over Migrating Processes with Timers. *Lecture Notes in Computer Science* **7273**, 52–66 (2012).
5. G. Ciobanu. Behaviour Equivalences in Timed Distributed π-Calculus. In *Software-Intensive Systems and New Computing Paradigms*, *Lecture Notes in Computer Science* **5380**, 190–208 (2008).
6. G. Ciobanu, T.S. Hoang, A. Stefanescu. From TiMo to Event-B: Event-Driven Timed Mobility. *ICECCS 2014* best paper, IEEE Computer Society, 1–10 (2014).
7. G. Ciobanu, C. Juravle. A Software Platform for Timed Mobility and Timed Interaction. FORTE, *Lecture Notes in Computer Science* **5522**, 106–121 (2009).
8. G. Ciobanu, C. Juravle. Flexible Software Architecture and Language for Mobile Agents. *Concurrency and Comput. Practice and Experience* **24**, 559–571 (2012).
9. G. Ciobanu, M. Koutny. Modelling and Verification of Timed Interaction and Migration. FASE, *Lecture Notes in Computer Science* **4961**, 215–229 (2008).
10. G. Ciobanu, M. Koutny. Timed Mobility in Process Algebra and Petri Nets. *Journal of Logic and Algebraic Programming* **80**, 377–391 (2011).
11. G. Ciobanu, M. Koutny. Timed Migration and Interaction With Access Permissions. Symposium on Formal Methods (FM), *Lecture Notes in Computer Science* **6664**, 293–307 (2011).
12. G. Ciobanu, M. Koutny. PerTiMo: A Model of Spatial Migration with Safe Access Permissions. *The Computer Journal* **58**, 1041–1060 (2015).
13. G. Ciobanu, M. Koutny, L.J. Steggles. Strategy Based Semantics for Mobility with Time and Access Permissions. *Formal Aspects of Computing* **27**, 525–549 (2015).
14. G. Ciobanu, C. Prisacariu. Timers for Distributed Systems. *Electronic Notes in Theoretic Computer Science* **164**, 81–99 (2006).

Proving Reachability Modulo Theories

Ștefan Ciobâcă

Faculty of Computer Science
Alexandru Ioan Cuza University
stefan.ciobaca@info.uaic.ro

We consider transition systems generated by constrained rewrite rules of the form $l \twoheadrightarrow r \texttt{ if } \phi$, where l and r are terms and ϕ is a logical constraint. The terms l, r can contain both uninterpreted symbols and symbols interpreted in a builtin model such as the model of booleans and integers. The constraint ϕ is a first-order formula which limits the application of the rule. The intuitive meaning of a constrained rule $l \twoheadrightarrow r \texttt{ if } \phi$ is that any instance of l that satisfies ϕ transitions into the corresponding instance of r in one step.

Given a constrained rule system, which serves as a specification for a transition system, it is natural to define the notion of constrained term $\varphi = (t \,|\, \phi)$, where t is an ordinary term (with variables) and ϕ is a logical constraint. The intuitive meaning of such a term is the set of ground instances of t that satisfy ϕ.

A reachability formula is a pair of constrained terms $(t \,|\, \phi) \Rightarrow (t' \,|\, \phi')$. The intuitive meaning of a reachability formula is that any instance of $(t \,|\, \phi)$ reaches, along all terminating paths of the transition system, an instance of $(t' \,|\, \phi')$ that agrees with $(t \,|\, \phi)$ on the set of shared variables.

We provide a proof system for deriving valid reachability formulae from transition systems specified by a set of constrained rules. We present our proof system in two steps. In the first step, we provide a three-rule proof system for *symbolic execution of constrained terms*:

$$[\mathsf{axiom}]\ \frac{}{(t \,|\, \phi) \Rightarrow \varphi'}\ M^{\Sigma} \models \phi \iff \bot$$

$$[\mathsf{subs}]\ \frac{(t'' \,|\, \phi'' \wedge \neg\phi''') \Rightarrow (t' \,|\, \phi')}{(t \,|\, \phi) \Rightarrow (t' \,|\, \phi')}\quad \begin{array}{l} (t'' \,|\, \phi'') \Rightarrow \varphi' \equiv (t \,|\, \phi) \Rightarrow \varphi', \text{ and} \\ M^{\Sigma} \models \phi''' \iff (\exists X)(t'' =^{?} t' \wedge \phi'), \text{ and} \\ X \triangleq var(t', \phi') \setminus var(t'', \phi'') \end{array}$$

$$[\mathsf{der}^{\forall}]\ \frac{\{(t^j \mid \phi^j) \Rightarrow \varphi' \mid (t^j \mid \phi^j) \in \Delta_{\mathcal{R}}((t'' \mid \phi''))\}}{(t \mid \phi) \Rightarrow \varphi'}$$

$(t \mid \phi)$ $\mathcal{R}-$derivable, and
$(t'' \mid \phi'') \Rightarrow \varphi' \equiv (t \mid \phi) \Rightarrow \varphi'$, and
$\phi'' \wedge \bigwedge \left\{ \neg(\exists Y)\phi^j \;\middle|\; \begin{array}{l} (t^j \mid \phi^j) \in \Delta_{\mathcal{R}}((t'' \mid \phi'')), \\ Y \triangleq var(t^j, \phi^j) \setminus var(t'', \phi'') \end{array} \right\}$ not satisfiable

The set $\Delta_{\mathcal{R}}((t'' \mid \phi''))$ denotes the symbolic successors of the constrained term $(t'' \mid \phi'')$ and a constrained term $(t \mid \phi)$ is $\mathcal{R}$-derivable if it has at least such a successor.

When interpreting the proof system coinductively, its proof tress can be finite or infinite. The finite proof trees allow to derive reachability formulae $(t \mid \phi) \Rightarrow (t' \mid \phi')$ when there is a bounded number of steps between $(t \mid \phi)$ and $(t' \mid \phi')$. The infinite proof trees correspond to proofs of reachability formulae $(t \mid \phi) \Rightarrow (t' \mid \phi')$ that hold for an unbounded number of steps between $(t \mid \phi)$ and $(t' \mid \phi')$. Unfortunately, the infinite proof trees are not too useful in practice because they cannot be obtained in finite time.

In order to allow the derivation of reachability formulae that require an unbounded number of steps, we introduce a fourth proof rule to the system that we call *circularity*:

$$[\mathsf{circ}]\ \frac{\begin{array}{l}(t'_c \mid \phi'_c \wedge \phi \wedge \phi'')) \Rightarrow \varphi', \\ (t \mid \phi \wedge \neg\phi'') \Rightarrow \varphi'\end{array}}{(t \mid \phi) \Rightarrow \varphi'} \quad \begin{array}{l} M^{\Sigma} \models \phi'' \iff (\exists var(t_c, \phi_c))(t =^? t_c \wedge \phi_c), \\ (t_c \mid \phi_c) \Rightarrow (t'_c \mid \phi'_c) \in G \end{array}$$

The circularity proof rule can be used to compress infinite proof trees into finite proof trees by first identifying a set G of *circularities*, which are simply patterns of reachability formulae that are repeatedly used in the infinite proof tree. The set G includes the reachability formulas to be proved. The circularity proof rule allows to use G as axioms to prove the formulae in G, thereby reducing infinite trees to finite trees. As expected, using the circularity rule in an unrestricted fashion can quickly lead to unsoundness. We provide a *syntactic criterion for using circularity in a sound manner*, by requiring that each use of the rule is preceded by a proper step in the transition system (using the $[\mathsf{der}^{\forall}]$ rule). This syntactic criterion selects sound proof trees out of the entire set of trees. In order to validate the proof system, we have implemented it in the `RMT` tool and tested it on some non-trivial examples.

This is joint work with Dorel Lucanu. Partially, this work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI UEFISCDI, project number PN-III-P2-2.1-BG-2016-0394, within PNCDI III.

A Session Logic for Communication Protocols

Florin Craciun[1], Wei-Ngan Chin[2] and Andreea Costea[2]

[1] Faculty of Mathematics and Computer Science, Babes-Bolyai University
craciunf@cs.ubbcluj.ro
[2] School of Computing, National University of Singapore
chinwn@comp.nus.edu.sg, andreeac@comp.nus.edu.sg

Abstract. Communication-centered programming is one of the most challenging programming paradigms. Development of modern software applications requires expressive mechanisms to specify and verify the communications between different parties. In the last decade, many works have used session types to characterize the various aspects of structured communications. Different from session types, we propose a novel session logic with disjunctions to specify and verify the implementation of communication protocols. Our current logic is based on only two-party channel sessions, but it is capable of handling delegation naturally through the use of higher-order channels. Due to our use of disjunctions to model both internal and external choices, we rely solely on conditional statements to support such choices, as opposed to specialized switch constructs in prior proposals. Furthermore, since our proposal is based on an extension of separation logic, it also supports heap-manipulating programs and copyless message passing. We demonstrate the expressivity and applicability of our logic on a number of examples.

With the success of tools like [1], [14] and [4] for the development and verification of non-concurrent systems, there are now deeper desires for similar tools for distributed systems. Additionally, the communication requirement, which is ubiquitous in software systems from the information exchange between software entities to the communication of these systems with their environments, must be properly verified to affirm the system's correctness. Due to its importance, a number of researchers have focused on the problems of ensuring safe communication in the last few decades.

CSP (Communicating Sequential Processes) [9] and CCS (Calculus of Communicating Systems) [11] are among the earliest theories to address the communication problems. The most recent extensions for these works are based on session types, and their derivatives, such as contracts [15]. In the last decade, session types have been integrated into a number of programming languages and process calculi, including functional languages [13, 5], object-oriented languages [8, 6], calculi of mobile processes [7], and higher-order processes [12]. Recently, session types have also been extended with logic [3] to act as a contract between the communication entities. This extension allows a more precise verification of the involved parties by enabling a concise specification of the transmitted messages on what one party must ensure, and from which the other party can rely on it.

There was also a proposal for multi-party session logic [2], but this logic tries to also summarize the effects of processes involved in the protocol. In contrast, we propose a session logic which focuses entirely on the communication patterns, while the effects of the associated processes are summarized directly in each thread's pre- and postcondition.

Contributions: Different from previous approaches, we propose a session logic with a novel (and natural) use of disjunction to specify and verify the implementation of communication protocols. Even though the currently proposed logic is based on two-party channel sessions, it can also handle delegation through the use of higher-order channels. Unlike past solution on delegation [6], our proposal uses the same send/receive channel methods for sending values, data structures, and channels. For example, [6] requires a separate set of send/receive methods to support higher-order channels. Furthermore, due to our use of disjunctions to model both internal and external choices, we need only use conventional conditional statements to support both kinds of choices. In contrast, past proposals typically require the host languages to be extended with a set of specialized switch constructs to model both internal and external choices. Additionally, our proposal is based on an extension of separation logic, and thus it supports heap-manipulating programs and copyless message passing. Lately, Villard et al. [10] have designed a logic for copyless message passing communication. Their logic relies on state-based global contracts while our more general logic of session is built as an extension of separation logic with disjunction to support communication choices. The logical formulae on protocols can also be localised to each channel and may be freely passed through procedural boundaries. Villard et al. [10] currently use double-ended channels to solely support communication safety, but do not guarantee deadlock freedom. In contrast, a channel in our proposal is multi-ended with its complementary properties captured in local specifications, which are supplied via each of the channel's aliases. As channels can support a variety of messages, we can treat the read content as dynamically typed where conditionals are dispatched based on the received types. Alternatively, we may also guarantee type-safe casting via verifying communication safety. We can also go beyond such cast safety by ensuring that heap memory and properties of values passed into the channels are suitably captured. Lastly by using a subsumption relation on our communication proposal, we allow specifications on channels to differ between threads but would ensure that they remain compatible at each join point, in order to prevent intra-channel deadlocks. More realistically, we also assume the presence of asynchronous communication protocols, where send commands are non-blocking.

In this paper, we argue strongly on the simplicity, expressivity and applicability of our logic by demonstrating it through a number of examples.

References

1. Abdulla, P.A., Deneux, J., Stålmarck, G., Ågren, H., Åkerlund, O.: Designing safe, reliable systems using scade. In: Leveraging Applications of Formal Methods, pp. 115–129. Springer (2006)

2. Bocchi, L., Demangeon, R., Yoshida, N.: A multiparty multi-session logic. In: Trustworthy Global Computing, pp. 97–111. Springer (2013)
3. Bocchi, L., Honda, K., Tuosto, E., Yoshida, N.: A theory of design-by-contract for distributed multiparty interactions. In: CONCUR 2010-Concurrency Theory, pp. 162–176. Springer (2010)
4. Chin, W.N., David, C., Gherghina, C.: A HIP and SLEEK verification system. In: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. pp. 9–10. ACM (2011)
5. DeLine, R., Fähndrich, M.: Enforcing high-level protocols in low-level software. ACM SIGPLAN Notices 36(5), 59–69 (2001)
6. Dezani-Ciancaglini, M., Mostrous, D., Yoshida, N., Drossopoulou, S.: Session types for object-oriented languages. In: ECOOP 2006–Object-Oriented Programming, pp. 328–352. Springer (2006)
7. Gay, S., Hole, M.: Subtyping for session types in the π calculus. Acta Informatica 42(2-3), 191–225 (2005)
8. Gay, S.J., Vasconcelos, V.T., Ravara, A., Gesbert, N., Caldeira, A.Z.: Modular session types for distributed object-oriented programming. In: ACM Sigplan Notices. vol. 45, pp. 299–312. ACM (2010)
9. Hoare, C.A.R.: Communicating sequential processes. Communications of the ACM 21(8), 666–677 (1978)
10. Lozes, É., Villard, J.: Shared contract-obedient channels. Science of Computer Programming 100, 28–60 (2015)
11. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92 (1980)
12. Mostrous, D., Yoshida, N.: Session typing and asynchronous subtyping for the higher-order π-calculus. Information and Computation 241, 227–263 (2015)
13. Pucella, R., Tov, J.A.: Haskell Session Types with (Almost) No Class. SIGPLAN Not. 44(2) (Sep 2008)
14. Schneider, S.: The B-method: An introduction. Palgrave Oxford (2001)
15. Villard, J., Lozes, É., Calcagno, C.: Proving copyless message passing. In: Programming Languages and Systems, pp. 194–209. Springer (2009)

Automated Analysis of Possible Malware Applications

Vlad Crăciun

Faculty Computer Science, Alexandru Ioan Cuza University
vcraciun@info.uaic.ro

Static and dynamic program analysis are widely used by researchers to identify possible application issues or unwanted execution behaviors. Some researchers are working mostly at source level, making use of static analysis, whereas others are focused more on dynamic analysis and reverse engineering to understand how binary code flows through execution environments. There are already available a lot of intruments to help analysts dig for specific features when dealing with binary applications. JIT translators, decompilers, debuggers, emulators and virtualized environments create together a set of tools for malware researchers. However the number of malwares which security companies are dealing with, in the last few years, have already grown at a point where human intervention is desired only for the fierce ones. A lot of effort was pushed in the last decade, into automated analysis frameworks and tools, circleing around symbolic execution and taint analysis. However only few of them are able to bring light upon the most hidden execution paths. In an attempt to get some informations about execution paths encoded in a simple application, a simple symbolic execution tool would have been enough, but dealing with malware will require special features to face the hiding mechanisms like obfuscation, code-packing, code-encryption, polymorphism and the ability to render useless and exploit through targeted attacks, most of the public available analysis environments. These practices, enables them to know when they are subject to analysis and force the execution to finish. The challenge with this type of threats is to understand faster what they are up to and the same time to control their anti-analysis tricks.

Most of the automated analysis engines have their roots in a serie of articles published since 2007. A. Moser et al. in [5] uses dynamic symbolic execution (also known as concolic execution) to identify malicious behavior using QEMU. A more complex project, BitBlaze still based on QEMU is developed by D. Song et al. in [6], which uses dedicated analysis plugins to highlight certain features of binary applications and for example, the benefits of symbolic execution were available later by a plugin called Rudder. Another project started around 2010 by V. Chipounov et al. in [14] makes use of concolic execution to selective execute symbolic only the interest paths, thus filtering out symbolic space at system calls. Other kind of frameworks focus more on taint analysis or control flow graph interpretation like [7] to filter out obfuscation and polymorphism. Some others, are using virtualized environments in [8], [9], [10], [11], [13] and [12] to understand the application, outside the execution environment. More recent and improved frameworks are also available. They tend to use pools of execution trace creators like QEMU and PIN and to

make use of concolic execution to push the limits of path exploration. Some of them are presented in [1], [2], [3] and [4], however they all can be targetted or exploited as they are pulbic and free to use and study, by malware authors.

The main purpose of the research is to provide a fast way to understand malware actions and if possible, reverse the environmental changes or prevent the execution. We were able up to this point, to develop from scratch, a JIT for binary code translation to help us understand Windows malware faster without loosing additional meaning of the data flow. We achieved this, combining the power of dynamic analysis over a couple of plugins handled by the translator and the power of static analysis to bring back symbols onto the execution traces. In this work, we propose a JIT translator, similar to PIN from Intel along with a couple of plugins which could automate different types of analysis.

The advantage of using customized tools and frameworks will help us evade the anti-analysis tricks used by the malware authors, as they are unable to render them useless, unless they get to study them somehow. The types of analyses we can handle, expand also to emulators and virtual environments and cover bacward and forward taint analysis, dynamic symbolic execution, hooking and Control Flow Graphs. We are trying to put all these pieces together to bring binary analysis to the next level of speed and eficiency. Among the entire application, we will focus more on the translator engine and dynamic symbolic execution module, used to explore as much execution paths as possible.

The entire analysis engine is still in progress and we have a well working JIT, an almost finished dynamic symbolic execution engine with semantic for the most used arithmetic instructions and some dedicated hooking plugins to control the entire analysis context, thus distributing the analyzer with each new process, thread, callback or exception created by the target binary. We expect to finish soon the caching for environmental changes and full loop translations to optimize the time for trace creation. Caching changes will allow us to isolate what malware expects to find or modify into our environment by what it can actually do and it also speeds up the path exploration for dynamic symbolic execution, as the path shifting will resume to reversing memory and cached environment snapshots. We will also need to see how paths revealed by the dynamic symbolic execution could be labeled with symbols of the executed functions along with their parameters to help us discover cryptolocker behavior and not only, or how operating on control flow graphs may create strategies to clean binaries affected by file-infectors, or detecting certain graphs among merged execution traces.

The presentation will start with a short introduction to static and dynamic analysis benefits and will continue with the JIT design. In the second half we will try to see some differences between static symbolic execution and different ways to do its dynamic version, where we will also try to fit our own approach. In the end there will be highlighted possible extensions of the framework and some of their implications.

References

1. Shoshitaishvili, Yan and Wang, Ruoyu and Salls, Christopher and Stephens, Nick and Polino, Mario and Dutcher, Andrew and Grosen, John and Feng, Siji and Hauser, Christophe and Kruegel, Christopher and Vigna, Giovanni (State of) The Art of War: Offensive Techniques in Binary Analysis *2016 IEEE Symposium on Security and Privacy* http://angr.io/.
2. Reverse engineering framework in Python *2015*, https://github.com/cea-sec/miasm.
3. Jonathan Salwan, Pierrick Brunet, Florent Saudel, Romain Thomas A DBA Framework *2015* https://triton.quarkslab.com/.
4. A binary tracing tool for Windows *2016* https://github.com/K2/EhTrace.
5. Exploring Multiple Execution Paths for Malware Analysis *2007* http://analysis.seclab.tuwien.ac.at/papers/explore.pdf.
6. BitBlaze: A New Approach to Computer Security via Binary Analysis *2008* https://www0.comp.nus.edu/~liangzk/papers/iciss08.pdf.
7. MAIL: Malware Analysis Intermediate Language: a step towards automating and optimizing malware detection *2013* http://dl.acm.org/citation.cfm?id=2527006.
8. PyTrigger: A System to Trigger and Extract User-Activated Malware Behavior *2013* http://ieeexplore.ieee.org/abstract/document/6657230/.
9. MAVMM: Lightweight and Purpose Built VMM for Malware Analysis *2009* http://ieeexplore.ieee.org/abstract/document/5380697/.
10. AMAL: High-Fidelity, Behavior-based Automated Malware Analysis and Classification *2014* https://link.springer.com/chapter/10.1007/978-3-319-15087-1_9.
11. Ether: Malware Analysis via Hardware Virtualization Extensions *2008* http://dl.acm.org/citation.cfm?id=1455779.
12. Toward Automated Dynamic Malware Analysis Using CWSandbox *2007* http://ieeexplore.ieee.org/abstract/document/4140988/.
13. V2E: Combining Hardware Virtualization and Software Emulation for Transparent and Extensible Malware Analysis *2012* http://dl.acm.org/citation.cfm?id=2151053.
14. S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems *2011* dl.acm.org/citation.cfm?id=1950396.

Mathematical foundations for conceptual blending

Răzvan Diaconescu

Simion Stoilow Institute of Mathematics of the Romanian Academy
Razvan.Diaconescu@imar.ro

Conceptual blending

This work constitutes an effort to provide adequate mathematical foundations to *conceptual blending*, which is an important research problem in the area of *computational creativity*. This is a relatively recent multidisciplinary science, with contributions from/to artificial intelligence, cognitive sciences, philosophy and arts, going back at least until to the notion of *bisociation*, presented by Arthur Koestler. Its aims are not only to construct a program that is capable of human-level creativity, but also to achieve a better understanding and to provide better support for it. Conceptual blending was proposed by Fauconnier and Turner as a fundamental cognitive operation of language and common-sense, modelled as a process by which humans subconsciously combine particular elements of two possibly conceptually distinct notions, as well as their relations, into a unified concept in which new elements and relations emerge.

The structural aspects of this cognitive theory have been given rigorous mathematical grounds by Goguen based upon category theory. In this formal model, concepts are represented as logical theories giving their axiomatization. Goguen used the algebraic specification language OBJ to axiomatize the concepts, a language that is based upon a refined version of equational logic; but in fact the approach is independent of the logical formalism used (this is why category theory is involved).

In the above-mentioned work by Goguen there are convincing arguments, supported by examples, for the partiality of theory translations, which represents very much a departure to a different mathematical realm than that of logical theories (even when considered in a very general sense, as commonly done in modern computer science). In category-theoretic terms, this means that we need to consider there categories equipped with partial orders on the hom-sets that are preserved by the compositions of arrows/morphisms. These are special instances of 2-categories (a rather notorious concept), somehow half-way between ordinary categories

and 2-categories; according to Goguen, this is what motivates the term $\frac{3}{2}$-*category*. To summarise the main mathematical idea underlying theory blending as it stands now:

> *Theory blending is a cocone in a* $\frac{3}{2}$*-category in which objects represent logical theories and arrows correspond to partial mappings between logical theories.*

There is still a great deal of thinking on whether the cocone should actually be a colimit (in other words, a minimal cocone) or not necessarily. An understanding of this issue is that blending should not necessarily be thought as a colimit, but that colimits are related to a kind of *optimality* principle. Moreover, since $\frac{3}{2}$-category theory has several different concepts of colimits, there is still thinking about which of those is most appropriate for modelling the blending operation.

Goguen's ideas about theory blending benefited from an important boost with the European FP7 project COINVENT that has adopted them as its foundations. Based on this, a creative computational system has been implemented and demonstrated in fields like mathematics and music (although both use the strict rather than the $\frac{3}{2}$-version of category theory).

$\frac{3}{2}$-institutions

However, the COINVENT approach still lacks crucial theoretical features, especially a proper semantic dimension. Such a dimension is absolutely necessary when talking about concepts because meaning and interpretation are central to the idea of concept. For example, the idea of consistency of a concept depends on the semantics. If one considers also the abstraction level of Goguen's approach in its general form, of non-commitment to particular logical systems, then *the institution-theoretic dimension appears as inevitable*. In fact, Goguen argued for the role of institution theory in and so does the COINVENT project. However, institution theory cannot be used as such in a proper way because, as it stands now, it cannot capture the partiality of theory morphisms (which boils down to the partiality of signature morphisms).

Therefore we define a $\frac{3}{2}$-categorical extension of the concept of institution, called $\frac{3}{2}$-*institution*, that accommodates those aspects and that starts from an abstract $\frac{3}{2}$-category of signatures. Moreover, based on this, we unfold a theory of $\frac{3}{2}$-institutions aimed as a general institution theoretic foundations for conceptual blending.

Specification and Verification of Invariant Properties of Transitional Systems

Daniel Găină

Institute of Mathematics for Industry, Kyushu University
daniel@imi.kyushu-u.ac.jp

Abstract. The paper advances a verification method for transition systems whose reachable states are explicitly described by membership axioms. The proof technique is implemented in Constructor-based Inductive Theorem Prover (CITP), a proof management tool built on top of a variation of conditional equational logic enhanced with many modern features. This approach complements the so-called OTS/CafeOBJ method, a verification procedure for observational transitional systems that is already implemented in CITP.

1 Introduction

We propose a formal method for analyzing transition systems whose reachable states are described explicitly by membership axioms [8]. The specification and verification technique is supported by Constructor-based Inductive Theorem Prover (CITP) [7], which is implemented in Maude [1]. The formal language of CITP is based on a variation of conditional equational logic [3] with sub-sorting relations [4], membership axioms, transitions [2] and constructor operators [5], which makes Maude notation suitable to write specifications in the CITP language. The methodology described in the present contribution complements the one already supported by CITP [6] for analyzing invariant properties of Observational Transition Systems (OTS). In the latter technology, the system states are "black-boxes" that are distinguished only by observational functions.

Inductive theorem provers are interactive, in general. They require a trained user to direct the theorem prover towards discharging the goals which cannot be proved by automatic techniques. In many cases, the tool needs the user's help to perform trivial proofs. One major direction of research in inductive theorem proving is improving and reducing the user interaction. This issue is tackled in the present contribution from the following angles: (1) better strategies for generating induction schemes, (2) improved decision procedures to perform automated reasoning, and (3) improvements of the proof assistant interface to help the user understand the current state of the proof and interact with the tool in a more natural way.

The CITP's source code has been refactored to be more readable and improved to make it a better platform for future extensions. The new features consist of (a) an induction scheme based on constructors given as membership

axioms, (b) a tactic for computing and joining critical pairs, and (c) a new interface designed to improve the user's interaction with the tool and then implemented from scratch in Core Maude. Note that in the previous version the interface was implemented in Full Maude. The command parsing component of the interface was upgraded to generate better error messages. CITP can be downloaded from `http://imi.kyushu-u.ac.jp/~daniel/citp.html`.

2 Preliminaries

Note that a specification consists of a signature, a set of sentences and a class of models. The set of sentences gives the operational semantics since they form a term rewriting system which makes the specification executable by rewriting. The class of models gives the denotational semantics of the underlying specification. Our proof technique is intimately linked to the structure of the sentences and takes advantage of both denotational and operational semantics in order to improve the user's interaction with the tool: (a) it is equipped with specialized proof rules for the initial data types that are often used in practice such as sequences or Booleans, and (b) it uses term rewriting for verification.

A goal is a pair $\langle \mathtt{SP}, E \rangle$, where $\mathtt{SP}$ is a specification and E is a set of formulas to prove. A proof rule is a mapping from a goal $\langle \mathtt{SP}, E \rangle$ to a list of specifications $\langle \mathtt{SP}_1, E_1 \rangle \ldots \langle \mathtt{SP}_n, E_n \rangle$. The tactics are obtained by canonically extending the proof rules to mappings from lists of goals to lists of goals. In the current version of CITP, the user can give commands which may consist of lists of tactics rather than a single tactic. It follows that the proofs consist of sequences of lists of tactics instead of trees (whose nodes are goals and edges are lists of tactics). This has the advantage of simplifying the design of the tool interface and increase the automation level of the proof process by making all goals to be discharged available to the user simultaneously. However, if the user wants to apply a tactic list `tctList` to the current goal then the command `(. tctList)` can be fed to the tool. In addition, `(select n)` moves the goal `n` to the top of the goal list making it the current goal.

Assume that the initial goal is a pair $\langle \mathtt{SP}, \gamma \rangle$ consisting of a specification $\mathtt{SP}$ and a sentence $\gamma = (\forall X) \bigwedge H \Rightarrow C$, where X is a set of variables and $H \cup \{C\}$ is a set of atomic formulas given as equations, membership axioms or transitions. The idea is to develop tactics that decompose the sentence γ into simpler basic constituents, e.g. equational formulas, which can be discharged using term rewriting. There are two tactics designed to eliminate universal quantification: induction and theorem of constants. While the latter is applicable to all variables in X, the former can eliminate only variables of *constrained* sort [1]. It is the user's responsibility to separate variables of constrained sorts, which will be dealt by induction, from the rest of the variables, which will be handled by theorem of constants. Induction is applied first and then theorem of constants comes into play. The goals $\langle \mathtt{SP}_1, \gamma_1 \rangle \ldots \langle \mathtt{SP}_n, \gamma_n \rangle$ obtained by applying induction

[1] The sorts that have constructors are called constrained. The sorts with no constructors are called loose.

and theorem of constants have only quantifier free formulas, i.e. γ_i is of the form $\bigwedge H_i \Rightarrow C_i$ for all $i \in \{1, \ldots, n\}$. The tactic implication is designed to eliminate the logical implication. For example, by applying implication to a goal $\langle \mathtt{SP}_i, \gamma_i \rangle$ the result is $\langle \mathtt{SP}_i[H_i], C_i \rangle$, where $\mathtt{SP}_i[H_i]$ is the specification obtained from $\mathtt{SP}_i$ by adding the axioms in H_i. The goals $\langle \mathtt{SP}_i[H_i], C_i \rangle$ are discharged using term rewriting.

Consider the following specification of natural numbers with addition:

```
fmod PNAT is
 sorts PNat PNzNat .
 subsorts PNzNat < PNat .
 op 0 : -> PNat [ctor] .
 op s_ : PNat -> PNzNat [ctor].
 op _+_ : PNat PNat -> PNat [prec 33].
 vars M N : PNat .
 eq 0 + N = N          [metadata "1"].
 eq s M + N = s(M + N)[metadata "2"].
endfm
```

Suppose one wants to prove the associativity of addition. The goal is introduced by the command: `(goal PNAT |- eq (X:PNat + Y:PNat)+ Z:PNat = X:PNat + (Y:PNat + Z:PNat);)`. Variable `X` is chosen for induction and the tactic `(ind on X:PNat)` is applied. The tool will generate two subgoals, one for each constructor, zero `0` and successor `s`. By theorem of constants `(tc)`, the variables `Y` and `Z` are introduced as constants to the body of the specifications of the subgoals generated by induction. Since the logical implication does not occur in the formulas to prove, the goals can be discharged by the reduction tactic `(red)`. The proof of associativity consists of the tactic list `(ind on X:PNat tc red)`. The command `(show proof)` displays the current proof and can be used any time during the proof process. The command `(show goals)` display the remaining goals, and the command `(rollback)` returns the proof process to the previous state. In case of large specifications, it is undesirable to display all its sentences during the proof process. Therefore, only sentences with the attribute `metadata "n"` will be displayed, where `n` is a natural number. The axioms introduced in the proof process will get automatically a natural number as attribute.

The verification method described above is developed further in two directions to prove invariant properties of transitional systems: (1) one technique is designed for observational transitional systems, where the structure of the states is not accessible to the users and the states are distinguished only by some "observational" functions, and (2) the other approach is built for transitional systems where the states are described explicitly by membership equations. Therefore, the methods share common tactics such as theorem of constants, implication or reduction. The implementation of the tool is modular allowing to reuse the code for the shared tactics. Note that the present contribution focuses on the latter methodology.

References

1. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007.
2. R. Diaconescu and K. Futatsugi. *Cafeobj Report - The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.
3. J. A. Goguen and J. Meseguer. Completeness of many-sorted equational logic. *SIGPLAN Notices*, 17(1):9–17, 1982.
4. J. A. Goguen and J. Meseguer. Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. *Theor. Comput. Sci.*, 105(2):217–273, 1992.
5. D. Găină, K. Futatsugi, and K. Ogata. Constructor-based Logics. *J. UCS*, 18(16):2204–2233, 2012.
6. D. Găină, D. Lucanu, K. Ogata, and K. Futatsugi. On Automation of OTS/CafeOBJ Method. In S. Iida, J. Meseguer, and K. Ogata, editors, *Specification, Algebra, and Software*, volume 8373 of *LNCS*, pages 578–602. Springer, 2014.
7. D. Găină, M. Zhang, Y. Chiba, and Y. Arimoto. Constructor-based Inductive Theorem Prover. In R. Heckel and S. Milius, editors, *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013*, volume 8089 of *LNCS*, pages 328–333. Springer, 2013.
8. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97*, volume 1376 of *LNCS*, pages 18–61. Springer, 1997.

System Modelling, Validation and Testing using Kernel P Systems

Marian Gheorghe[1], Florentin Ipate[2]

[1] School of Electrical Engineering and Computer Science,
University of Bradford, West Yorkshire, Bradford BD7 1DP, UK
m.gheorghe@bradford.ac.uk
[2] Department of Computer Science, Faculty of Mathematics and Computer Science,
University of Bucharest, Str. Academiei nr. 14, 010014, Bucharest, Romania
florentin.ipate@ifsoft.ro

Nature inspired computational approaches have been on the focus of researchers for several decades. Membrane computing [14] is one of these paradigms that has recently been through significant developments leading to a broad spectrum of outcomes. The main computational models are called membrane systems or *P systems* and are inspired by the functioning and structure of the living cells as well as of more complex higher-oder biological entities, such as tissues and organs.

In recent years, various types or classes of P systems have been introduced and and studied, and in certain cases applied to different problems. While these variants provide more flexibility in modelling, this has inevitably resulted in a large pool of P system variants, which do not have a coherent integrating view and might be a drawback in analysing these models with some standard formal verification methods and tools.

Kernel P (kP) systems have been introduced to unify many variants of P system models, and combine a blend of various P system features and concepts, including (i) complex guards attached to rules, (ii) flexible ways to specify the system structure and dynamically change it and (iii) various execution strategies for rules and compartments.

The usability and efficiency of kP systems have been illustrated by a number of representative case studies, ranging from systems and synthetic biology, e.g. quorum sensing [12], genetic Boolean gates [15] and synthetic pulse generators [1], to some classical computational problems, e.g. sorting [6], broadcasting [9] and subset sum [5].

Kernel P system models are supported by an integrated software suite, namely kPWorkbench, which employs a set of simulation and formal verification tools and methods that permit simulating and verifying them. The models are expressed in a specification language, called *kP-Lingua*. The verification component of kPWorkbench [5] checks the correctness of kP system models by exhaustively analysing all possible behaviours. In order to facilitate the specification of system requirements, kPWorkbench features a property language, called *kP-Queries*, which comprises a list of property patterns written as natural language statements. The properties expressed in *kP-Queries* are verified using the Spin [11] and NuSMV [3] model checkers after being translated into correspond-

ing *Linear Temporal Logic (LTL)* and *Computation Tree Logic (CTL)* syntax. The simulation component features a native simulator [2, 13], which allows the users to simulate kP system models efficiently. In addition, kPWorkbench integrates the Flame simulator [4, 15], a general purpose large scale agent based simulation environment, based on a method that allows users to express kP systems as a set of communicating X-machines [10].

Significant progress has also been made in the area of testing applications modelled by kP systems. When testing a kP system model, an automata model needs to be constructed first, based on the computation tree of the kP system. As, in general, the computation tree may be infinite and cannot be modelled by a finite automaton, an approximation of the tree is used. This approximation is obtained by limiting the length of any computation to an upper bound k and considering only computations up to k transitions in length. This approximation is then used to construct a deterministic finite cover automaton (DFCA) of the model [6–8].

This paper reviews the main achievements in the area of kP systems modelling, verification and testing.

References

1. Bakir, M.E., Ipate, F., Konur, S., Mierla, L., Niculescu, I.: Extended simulation and verification platform for kernel P systems. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8961, pp. 158–178. Springer (2014), http://dx.doi.org/10.1007/978-3-319-14370-5_10
2. Bakir, M.E., Konur, S., Gheorghe, M., Niculescu, I., Ipate, F.: High performance simulations of kernel P systems. In: 2014 IEEE International Conference on High Performance Computing and Communications, 6th IEEE International Symposium on Cyberspace Safety and Security, 11th IEEE International Conference on Embedded Software and Systems, HPCC/CSS/ICESS 2014, Paris, France, August 20-22, 2014. pp. 409–412 (2014), http://dx.doi.org/10.1109/HPCC.2014.69
3. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification, 14th International Conference, CAV 2002, Copenhagen, Denmark, July 27-31, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2404, pp. 359–364. Springer (2002), http://dx.doi.org/10.1007/3-540-45657-0_29
4. Coakley, S., Gheorghe, M., Holcombe, M., Chin, S., Worth, D., Greenough, C.: Exploitation of high performance computing in the FLAME agent-based simulation framework. In: Min, G., Hu, J., Liu, L.C., Yang, L.T., Seelam, S., Lefèvre, L. (eds.) 14th IEEE International Conference on High Performance Computing and Communication & 9th IEEE International Conference on Embedded Software and Systems, HPCC-ICESS 2012, Liverpool, United Kingdom, June 25-27, 2012. pp. 538–545. IEEE Computer Society (2012), http://dx.doi.org/10.1109/HPCC.2012.79
5. Dragomir, C., Ipate, F., Konur, S., Lefticaru, R., Mierla, L.: Model checking kernel P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 14th International Confer-

ence, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8340, pp. 151–172. Springer (2013), http://dx.doi.org/10.1007/978-3-642-54239-8_12
6. Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S.: Kernel P systems modelling, testing and verification - sorting case study. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing - 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10105, pp. 233–250. Springer (2016), http://dx.doi.org/10.1007/978-3-319-54072-6_15
7. Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S., Lefticaru, R.: Kernel P systems: from modelling to verification and testing. Theoretical Computer Science (accepted for publication), http://hdl.handle.net/10454/11720
8. Gheorghe, M., Ipate, F.: On testing P systems. In: Corne, D.W., Frisco, P., Paun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 5391, pp. 204–216. Springer (2008), http://dx.doi.org/10.1007/978-3-540-95885-7_15
9. Gheorghe, M., Konur, S., Ipate, F., Mierla, L., Bakir, M.E., Stannett, M.: An integrated model checking toolset for kernel P systems. In: Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron, C. (eds.) Membrane Computing - 16th International Conference, CMC 2015, Valencia, Spain, August 17-21, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9504, pp. 153–170. Springer (2015), http://dx.doi.org/10.1007/978-3-319-28475-0_11
10. Holcombe, M.: X-machines as a basis for dynamic system specification. Software Engineering Journal 3(2), 69–76 (1988), http://dx.doi.org/10.1049/sej.1988.0009
11. Holzmann, G.J.: The model checker SPIN. IEEE Transactions on Software Engineering 23(5), 275–295 (1997)
12. Konur, S., Gheorghe, M., Dragomir, C., Mierla, L., Ipate, F., Krasnogor, N.: Qualitative and quantitative analysis of systems and synthetic biology constructs using P systems. ACS Synthetic Biology 4(1), 83–92 (2015), http://dx.doi.org/10.1021/sb500134w, pMID: 25090609
13. Konur, S., Kiran, M., Gheorghe, M., Burkitt, M., Ipate, F.: Agent-based high-performance simulation of biological systems on the GPU. In: 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, New York, NY, USA, August 24-26, 2015. pp. 84–89. IEEE (2015), http://dx.doi.org/10.1109/HPCC-CSS-ICESS.2015.253
14. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000), http://dx.doi.org/10.1006/jcss.1999.1693
15. Sanassy, D., Fellermann, H., Krasnogor, N., Konur, S., Mierla, L., Gheorghe, M., Ladroue, C., Kalvala, S.: Modelling and stochastic simulation of synthetic biological boolean gates. In: 2014 IEEE International Conference on High Performance Computing and Communications, 6th IEEE International Symposium on Cyberspace Safety and Security, 11th IEEE International Conference on Embedded Software and Systems, HPCC/CSS/ICESS 2014, Paris, France, August 20-22, 2014. pp. 404–408 (2014), http://dx.doi.org/10.1109/HPCC.2014.68

Parking with a Worm's Brain

Radu Grosu

Fakultät für Informatik, Technische Universität Wien, Austria
radu.grosu@tuwien.ac.at

The recent success of deep neural networks (DNN) in unsupervised and supervised learning, in particular the success of DNN autoencoders, convolutional DNN, and recurrent DNN, have lead to an unprecedented interest in DNN, and also to bold predictions of their role in taking over humanity and rendering us useless.

However, DNN are considerably simplified, artificial models, that only capture the functional behaviour of the neurone?s synapses, and completely ignore the dynamic dynamic behaviour of the neurones themselves. If DNN were the key to learning and to intelligence, why did nature create biological rather than artificial neurones? What are the advantages of biological neurones?

In this talk, I show that biological neurones are very well suited to devise, or automatically learn sophisticated nonlinear controllers. We use a dynamic model that captures with decent precision, the behaviour of neurones and their synapses, in the C.elegans nematode. This model is powerful enough for designing or learning a parallel parking algorithm with just 39 neurones, where the role of each neurone is well understood. Moreover, the model turns out to be very robust, and easily extensible to tolerate faults. In fact, the neural model we use represents a Turing complete formalism, where parallel composition, and not sequential composition, is the norm.

Proof Complexity and Satisfiability Solving for several Combinatorial Principles

Gabriel Istrate[1]

West University of Timişoara and the e-Austria Research Institute
Bd. V. Pârvan 4, cam 047, RO-300223, Timişoara, Romania
gabriel.istrate@acm.org

Keywords: proof complexity, Frege systems, the Kneser-Lovász theorem

1 Introduction

In this talk (based on papers [1, 2] and unpublished work performed with Cosmin Bonchiş and Adrian Crăciun, both affiliated with the West University of Timişoara) we discuss the use of statements from topological combinatorics as a source of interesting propositional formulas. Our main example is the Kneser-Lovász theorem and its strenghtening due to Schrijver. We first considered these formulas in conjunction with the open problem of separating the Frege and extended Frege propositional proof systems (see [3] for an in-depth presentation)

1.1 Theoretical Results

The Kneser-Lovász theorem is stated as follows:

Proposition 1. *Given $n \geq 2k \geq 1$ and a function $c : \binom{n}{k} \to [n-2k+1]$ there exist two disjoint sets A, B and a color $1 \leq l \leq n-2k+1$ with $c(A) = c(B) = l$.*

See [4, 5] for readable introductions. It was first established by Lovász, using methods rooted in Algebraic Topology.

We can map this statement to a class (denoted by Kneser_k^n) of propositional formulas in the obvious way, by encoding with a boolean variable $X_{A,i}$ the statement $c(A) = i$. For $k = 1$ we retrieve the wellknown class PHP_n of formulas encoding the pigeonhole principle.

The following results were proved in [1]:

Theorem 1. *For all $k \geq 1, n \geq 3$ there exists a variable substitution Φ_k, $\Phi_k : Var(\text{Kneser}_{k+1}^n) \longrightarrow Var(\text{Kneser}_k^{n-2})$ such that $\Phi_k(\text{Kneser}_{k+1}^n)$ is a formula consisting precisely of the clauses of Kneser_k^{n-2} (perhaps repeated and in a different order).*

Therefore, all known existing proof complexity lower bounds for the pigeonhole formulas extend to the Kneser_k^n formulas, and the class $\text{Kneser}_k^n + 1$ is "harder" than Kneser_k^n.

For $k = 2$ and $k = 3$, the cases where the Kneser-Lovász theorem has combinatorial proofs, we proved:

Theorem 2. *The following are true:*

- *(a) The class of formulas* Kneser_2^n *has polynomial size Frege proofs.*
- *(b) The class of formulas* Kneser_3^n *has polynomial size* extended *Frege proofs.*

We continued the investigation of Kneser formulae in [2], where the following surprising results were proved:

Theorem 3. *For fixed parameter* $k \geq 1$*, the propositional translations* Kneser_k^n *of the Kneser-Lovász theorem have polynomial size extended Frege proofs.*

Theorem 4. *For fixed parameter* $k \geq 1$*, the propositional translations* Kneser_k^n *of the Kneser-Lovász theorem have quasi-polynomial size Frege proofs.*

Remarkably, the proof of the previous two theorems **do not** use techniques from algebraic topology, but instead reduce each case of the Kneser-Lovász theorem with fixed $k \geq 1$ to the verification of a finite number of instances. Therefore each such fixed case has efficient combinatorial proofs.

2 Truncations of combinatorial results

A complementary perspective provided in paper [2] concerned the encoding of a combinatorial principle, the so-called *octahedral Tucker lemma*, a discrete version of the Borsuk-Ulam theorem that is strong enough to prove the Kneser-Lovász theorem. The propositional encoding of this principle is inefficient (leads to exponential-size formulas). However, in [2] we found a weaker but polynomial-size truncation. Therefore, our initial intuitions on the hardness of Kneser formulas may in fact hold for the harder truncated Tucker formulas.

In the talk we will present some complexity-theoretic results on the truncated Tucker formulae, as well as a new (unpublished) second example of the truncation of the Octahedral Tucker lemma that is strong enough to establish another combinatorial result, the so-called *necklace-splitting theorem* due to Noga Alon [6].

3 Experiments: solving Kneser formulae in practice

We will discuss experiments performed using the SAT solver *lingeling* and the Integer Programming solver *GUROBI* in solving various instances of the Schrijver formulas.

References

1. G. Istrate and A.Crãciun: Proof complexity and the Lovasz-Kneser Theorem. Lecture Notes in Computer Science, Springer Verlag, Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT'14), vol. 8561, 2014.
2. James Aisenberg, Maria Luisa Bonet, Sam Buss, Adrian Craciun and Gabriel Istrate: Short Proofs of the Kneser-Lovsz Coloring Principle. Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015), Lecture Notes in Computer Science vol. 9135, Springer Verlag, 2015.
3. Jan Krajicek: Bounded arithmetic, propositional logic and complexity theory. Cambridge University Press, 1995.
4. Jiri Matousek: Using the Borsuk-Ulam theorem: lectures on topological methods in combinatorics and geometry. Springer Science & Business Media, 2008.
5. Mark De Longueville. A course in topological combinatorics. Springer Science & Business Media, 2012.
6. Noga Alon: Splitting necklaces. Advances in Mathematics 63.3 (1987): 247-253.

Building Models for Verification of Security Properties

Marius Minea

Politehnica University of Timişoara, Romania
marius@cs.upt.ro

Security is a classical example of a domain where errors in system design are frequent, yet often hard to find before the system is deployed. Thus, applying formal methods for verification is much needed. To do this, it is crucial to have system models that are accurate enough to account for the targeted flaws, yet employ the right abstractions to keep them tractable by model checking tools.

We discuss how to obtain such models for verifying security properties of web applications: extracting a model by analyzing the code, if available, as well as inferring a model from application behavior.

Model extraction from web application code Web applications are prone to well-known specific flaws such as cross-site scripting or SQL injection, but also to broken authentication and business logic errors which are harder to pinpoint and thus test against. Model checking with its exhaustive state-space exploration capability is a natural candidate to apply, and numerous model checkers for security protocol exist; however, they are mostly used with hand-written models. Moreover, the ability of the Dolev-Yao attacker to generate new messages from known parts necessarily leads to very large state spaces.

Thus, the challenge is to build a tool that automatically extracts models that accurately capture the application workflow, yet employ suitable abstractions to keep the models small. We discuss the jMODEX tool [3] built in the context of the SPaCIoS project [4] to extract models from web applications written using Java ServerPages (JSP). The resulting models can be analyzed by the model checkers of the AVANTSSAR platform [2].

jMODEX works as one might expect by backward traversal of the control flow graph for each Java method, tracking dataflow and path conditions to build an extended finite state machine. Crucially however, jMODEX is not a general-purpose Java model checker. It has semantic knowledge about the API for interacting with HTTP requests and exploits the typical structure of such applications (built around a server loop). It does not handle full-fledged Java (e.g., recursion, polymorphic calls); on the other hand, it provides a database model and has support for a subset of SQL queries, which is crucial for handling practical applications. jMODEX is configurable and extensible through user-specified abstractions, which allow to specify the semantics of certain methods in terms of its meta-model and thus adapt the abstraction level to the security property of interest. In effect, one can view jMODEX as a framework for building custom model extractors. Analyzing extracted models with the CL-AtSe model checker, we have demonstrated finding a bug in an open-source bookstore application.

Model inference from application behavior Recently, model-learning [5], based on Angluin's automata learning algorithm [1] has emerged as a practical method for error detection. It can be used to reverse-engineer system models, check protocol conformance with a specification, compare two implementations, etc.

For a web application, model learning is in essence a form of 'smart crawling'. It creates a 'page graph' of the application, where nodes are pages and edges are transitions (through form submission) or links between pages. Inferring a crawled (black-box) model can be useful since a source-extracted model may be imprecise or have unwanted implementation detail. Again, employing the suitable abstraction is crucial: pages which differ just in the values of dynamical elements (e.g. books in a bookstore) are deemed equal, as are links with the same target but different URL parameter values. Crawling stops when no new page representatives are found, their successors being similar to a given depth.

Comparing white-box and black-box models While not identical, models extracted from code (white-box) and inferred from exercising the application (black-box) should represent the same behaviors. Since crawling only follows the interface (links and forms) available to the user, behaviors allowed by the code but not found by crawling are potential workflow vulnerabilities.

By composing transitions in the code model to macro-transitions from receiving a request to sending a response, the models become compatible at transition level. A path in the white-box model with a path condition that does not satisfy any crawled path would be an undesired execution (e.g. authentication bypass). These can be checked by feeding path conditions to an SMT solver such as Z3. Thus, the existence of both types of model can be exploited in verification.

Acknowledgments This is joint work with Petru Florin Mihancea, who developed the jMODEX tool, and with former students Alex Gyori, Cristian Iuga and Csongor Mátyás-Barta in the European FP7 project no. 257876 SPaCIoS.

References

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[2] A. Armando, W. Arsac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. Cuéllar, G. Erzse, S. Frau, M. Minea, S. Mödersheim, D. von Oheimb, G. Pellegrino, S. E. Ponta, M. Rocchetto, M. Rusinowitch, M. T. Dashti, M. Turuani, and L. Viganò. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference*, pages 267–282, 2012.

[3] P. F. Mihancea and M. Minea. jMODEX: Model extraction for verifying security properties of web applications. In *Proceedings of the IEEE CSME-WCRE Software Evolution Week*, pages 450–453, 2014.

[4] SPaCIoS Project. Deliverable 5.5: Final Tool Assessment, 2014. Available at http://spacios.eu/deliverables/spacios-d5.5.pdf.

[5] F. Vaandrager. Model learning. *Communications of the ACM*, 60(2):86–95, 2017.

Network controllability: theory and applications

Ion Petre

Computational Biomodeling Laboratory, Åbo Akademi University and Turku Centre for Computer Science, Turku 20500, Finland
ipetre@abo.fi

The intrinsic robustness of living systems against perturbations is a key factor that explains why many single-target drugs have been found to provide poor efficacy or to lead to significant side effects. Rather than trying to design selective ligands that target individual receptors only, network polypharmacology aims to modify multiple cellular targets to tackle the compensatory mechanisms and robustness of disease-associated cellular systems, as well as to control unwanted off-target side effects that often limit the clinical utility of many conventional drug treatments. However, the exponentially increasing number of potential drug target combinations makes the pure experimental approach quickly unfeasible, and translates into a need for algorithmic design principles to determine the most promising target combinations to effectively control complex disease systems, without causing drastic toxicity or other side-effects. Building on the increased availability of disease-specific essential genes, we concentrate on the target structural controllability problem, where the aim is to select a minimal set of driver/driven nodes which can control a given target within a network. That is, for every initial configuration of the system and any desired final configuration of the target nodes, there exists a finite sequence of input functions for the driver nodes such that the target nodes can be driven to the desired final configuration. We investigated this approach in some pilot studies linking FDA-approved drugs with cancer cell-line-specific essential genes, with some very promising results.

Proof Assistants as Smart Programming Languages

Andrei Popescu

[1] Department of Computer Science, Middlesex University London, UK
[2] Institute of Mathematics Simion Stoilow of the Romanian Academy, Bucharest, Romania
a.popescu@mdx.ac.uk

Abstract. Proof assistants are tools specialized in helping humans prove theorems. What is perhaps less known is that they are also useful as programming environments. In this short text, I give a flavor of how a proof assistant can help write more reliable programs (e.g., guaranteed to be terminating or productive) without requiring the user to write proofs. I also give pointers to the work of my collaborators and me on improving the programming experience in the proof assistant Isabelle/HOL

In a typical proof assistant, a user can write commands like:

$$\mathsf{datatype}\ \alpha\ \mathsf{list} = \mathsf{Nil} \mid \mathsf{Cons}\ \alpha\ (\alpha\ \mathsf{list})$$

$$\mathsf{length\ xs}\ =\ \mathsf{case\ xs\ of\ Nil} \Rightarrow 0 \mid \mathsf{Cons\ x\ ys} \Rightarrow 1 + \mathsf{length\ ys}$$

Such commands are familiar to functional programmers, looking like usual recursive datatype and function defined in a functional programming language such as ML or Haskell. However, a proof assistant has a different take at such commands.

1 Theorem-Based Programming

For example, higher-order logic (HOL), the underlying logic of many successful proof assistants,[3] supports natively neither datatypes nor recursion—but only plain, nonrecursive definitions.

So how are such recursive definitions bootstrapped? *By automatically performing several nonrecursive definitions and proving several theorems.* Thus, when the user writes a datatype command like the above, the system does the following in response:

- Starts with the functor $(\alpha, \beta)\ \mathsf{F} = \mathsf{unit} + \alpha \times \beta$
- Builds its least fixpoint (initial algebra) $\alpha\ \mathsf{list} \simeq (\alpha, \alpha\ \mathsf{list})\ \mathsf{F}$
- Splits the fixpoint bijection into constructors Nil and Cons
- Proves the familiar properties: constructor injectivity, constructor-based induction and recursion principles, etc.

[3] These include HOL4 [25], HOL Light [14], ProofPower/HOL [2], HOL Zero [1], as well as my favorite, Isabelle/HOL [18, 19]—which actually implements a slight extension of HOL [16, 17], enabling Haskell-style type classes [20].

All this happens automatically, in the background, without the user needing to know. In Isabelle/HOL, 22000 lines of ML implement this functionality, covering both inductive and coinductiove datatypes (the latter also known as codatatypes).

Moreover, in response to a recursive function specification like the above for length, the system reacts as follows:

- Analyzes the call graph and proves that it is well founded
- Defines length using a well-founded recursion combinator
- Proves the above recursive equation as a theorem

In short, everything is reduced to nonrecursive definitions in HOL. I call such a programming model *theorem-based programming*, because the reduction proceeds by proving theorems.

A benefit of theorem-based programming is the availability of a significant amount of structure and static information on the defined types and programs. In particular, users obtain for free many useful polytypic operations on datatypes, such as map functions and "forall" predicates, as well as termination or productivity knowledge about their programs. This knowledge base is highly flexible and extendable, especially if the user is willing to do not only programming, but also a bit of proving. However, it should be insisted that a proof assistant is already useful to "ordinary" programmers, as it provides many facts automatically.

2 Theorems Versus Axioms

But why should one prefer theorem-based to the lighter[4] *axiom*-based programming? For example, when the user writes a recursive equation such as that of list, why not simply accept it as a new axiom of the system, say, after some syntactic checks, e.g., that all recursive calls are guarded? Why go further and establish well-foundedness of the call graph and then produce a nonrecursive definition in the background? There are two reasons. First, because the axiomatic approach is risky: Getting the checks slightly wrong leads to inconsistencies in the logic. Second, and equally importantly, because the axiomatic approach is inflexible: When the syntactic checks fail, the users cannot do anything, even if they know their recursion is correct. Consider the following specification of quicksort:

$$\mathsf{qsort}\ (\mathsf{Cons}\ x\ xs)\ =\ \mathsf{qsort}\ (\mathsf{filter}\ (<x)\ xs)\ ++\ [x]\ ++\ \mathsf{qsort}\ (\mathsf{filter}\ (\geq x)\ xs)$$

Without knowing the semantics of filter, a typical syntactic check must reject this definition—for all we know, filter could increase the size of its input list. In theorem-based programming, the system can ask the user for a hint or employ an existing fact about filter from the knowledge base, and then accept the definition.

In summary, theorem-based programming is the safest way to achieve *reliable* and *flexible* executable specifications in a proof assistant. Existing proof assistant tools realize these desiderata to different degrees—as discussed in the excellent recent survey [12], covering some of the most important players in the field: Agda [11], Coq [3], and the HOL-based proof assistants.

[4] That is, lighter for the proof assistant designers and implementors, not for the end users.

3 Achieving Flexibility

The HOL-based proof assistants are famously reliable thanks to reducing everything to a minimalistic logic kernel. In Isabelle/HOL [18, 19], we have recently also achieved significant flexibility:

- The defined datatypes can be inductive or coinductive [26], free or permutative [13, 22–24], and can be freely mixed and nested [5, 7, 9, 10].
- The functions defined on these datatypes can flexibly recurse [15], corecurse [4, 8], and even mix recursion and corecursion [4, 8].

These features required state-of-the art category theory, as well as mechanisms for customizing the abstract results to concrete cases—our motto was "employ category theory in the background, but do not expose the end user to it." There are two concepts behind this flexibility (achieved without compromising safety):

- *rich datatypes*, stored not as mere "types," i.e., flat collections of elements, but as functors and relators with additional structure and theorems [26] and with controlled size [6]—this enables modular constructions, preserving natural abstraction barriers
- *intelligent (co)recursors*, learning from their interaction with the users—this enables the system to become increasingly permissive with its allowed (co)recursion patterns [4, 8] and associated (co)induction principles [4, 8, 21]

4 Conclusion

Proof assistants are smart programming languages, in that they analyze each new user-specified datatype and program and integrate them in a knowledge base, fueled by proving theorems. This knowledge base offers substantial services to the programmer: It guarantees that programs terminate or are productive, and automatically defines many useful functions for datatypes. A wealth of additional static knowledge is made available to programmers who are willing to reach out and prove some basic properties of their programs. I wholeheartedly invite programmers to try Isabelle/HOL, which is one of the smartest (and friendliest) proof assistants ever to walk the earth.

Acknowledgment I gratefully acknowledge support from the UK Engineering and Physical Sciences Research Council (EPSRC) starting grant "VOWS: Verification of Web-based Systems" (EP/N019547/1).

References

1. Adams, M.: Introducing HOL Zero (extended abstract). In: Fukuda, K., van der Hoeven, J., Joswig, M., Takayama, N. (eds.) ICMS 2010. LNCS, vol. 6327, pp. 142–143. Springer (2010)
2. Arthan, R.D.: Some mathematical case studies in ProofPower–HOL. In: Slind, K. (ed.) TPHOLs 2004 (Emerging Trends). pp. 1–16 (2004)
3. Bertot, Y., Casteran, P.: Interactive Theorem Proving and Program Development—Coq'Art: The Calculus of Inductive Constructions. Springer (2004)

4. Blanchette, J.C., Bouzy, A., Lochbihler, A., Popescu, A., Traytel, D.: Friends with benefits - implementing corecursion in foundational proof assistants. In: ESOP 2017. pp. 111–140 (2017)
5. Blanchette, J.C., Hölzl, J., Lochbihler, A., Panny, L., Popescu, A., Traytel, D.: Truly modular (co)datatypes for Isabelle/HOL. In: Klein, G., Gamboa, R. (eds.) ITP 2014. LNCS, vol. 8558, pp. 93–110. Springer (2014)
6. Blanchette, J.C., Popescu, A., Traytel, D.: Cardinals in Isabelle/HOL. In: Klein, G., Gamboa, R. (eds.) ITP 2014. LNCS, vol. 8558, pp. 111–127. Springer (2014)
7. Blanchette, J.C., Popescu, A., Traytel, D.: Unified classical logic completeness: A coinductive pearl. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS, vol. 8562, pp. 46–60. Springer (2014)
8. Blanchette, J.C., Popescu, A., Traytel, D.: Foundational extensible corecursion: A proof assistant perspective. In: Fisher, K., Reppy, J.H. (eds.) ICFP 2015. pp. 192–204. ACM (2015)
9. Blanchette, J.C., Popescu, A., Traytel, D.: Soundness and completeness proofs by coinductive methods. J. Autom. Reasoning 58(1), 149–179 (2017)
10. Blanchette, J.C., Traytel, D., Popescu, A.: Foundational nonuniform (co)datatypes for higher-order logic. In: LICS 2017. IEEE Computer Society (2017), to appear
11. Bove, A., Dybjer, P., Norell, U.: A brief overview of Agda—A functional language with dependent types. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 73–78. Springer (2009)
12. Bove, A., Krauss, A., Sozeau, M.: Partiality and recursion in interactive theorem provers - an overview. Mathematical Structures in Computer Science 26(1), 38–88 (2016)
13. Gheri, L., Popescu, A.: A general formalized theory of syntax with bindings. In: ITP 2017 (2017), to appear
14. Harrison, J.: HOL Light: An overview. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 60–66. Springer (2009)
15. Krauss, A.: Partial and nested recursive function definitions in higher-order logic. J. Autom. Reasoning 44(4), 303–336 (2010)
16. Kunčar, O., Popescu, A.: A consistent foundation for Isabelle/HOL. In: Urban, C., Zhang, X. (eds.) ITP 2015. LNCS, Springer (2015)
17. Kuncar, O., Popescu, A.: Comprehending Isabelle/HOL's consistency. In: ESOP 2017. pp. 724–749 (2017)
18. Nipkow, T., Klein, G.: Concrete Semantics: With Isabelle/HOL. Springer (2014)
19. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Springer (2002)
20. Nipkow, T., Snelting, G.: Type classes and overloading resolution via order-sorted unification. In: Functional Programming Languages and Computer Architecture, 5th ACM Conference, Cambridge, MA, USA, August 26-30, 1991, Proceedings. pp. 1–14 (1991)
21. Popescu, A., Gunter, E.L.: Incremental pattern-based coinduction for process algebra and its Isabelle formalization. In: Ong, C.H.L. (ed.) FoSSaCS 2010. pp. 109–127 (2010)
22. Popescu, A., Gunter, E.L.: Recursion principles for syntax with bindings and substitution. In: ICFP 2011. pp. 346–358 (2011)
23. Popescu, A., Gunter, E.L., Osborn, C.J.: Strong normalization of System F by HOAS on top of FOAS. In: LICS 2010. pp. 31–40. IEEE Computer Society (2010)
24. Schropp, A., Popescu, A.: Nonfree datatypes in Isabelle/HOL - animating a many-sorted metatheory. In: CPP. pp. 114–130 (2013)
25. Slind, K., Norrish, M.: A brief overview of HOL4. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 28–32. Springer (2008)
26. Traytel, D., Popescu, A., Blanchette, J.C.: Foundational, compositional (co)datatypes for higher-order logic: Category theory applied to theorem proving. In: LICS 2012, pp. 596–605. IEEE Computer Society (2012)

Matching Logic: Syntax and Semantics

Grigore Roşu[1] and Traian Florin Şerbănuţă[2]

[1] University of Illinois at Urbana-Champaign, USA
grosu@illinois.edu
[2] University of Bucharest, Romania
traian.serbanuta@unibuc.ro

Abstract. This paper recalls the syntax and semantics of Matching Logic [6], a first-order logic (FOL) variant for specifying and reasoning about structure by means of patterns and pattern matching. Its sentences, the *patterns*, are constructed using *variables*, *symbols*, *connectives* and *quantifiers*, but no difference is made between function and predicate symbols. In models, a pattern evaluates into a power-set domain (the set of values that *match* it), in contrast to FOL where functions and predicates map into a regular domain. Matching logic uniformly generalizes several logical frameworks important for program analysis, such as: propositional logic, algebraic specification, FOL with equality, modal logic, and separation logic. Patterns can specify separation requirements at any level in any program configuration, not only in the heaps or stores, without any special logical constructs for that: the very nature of pattern matching is that if two structures are matched as part of a pattern, then they can only be spatially separated. Like FOL, matching logic can also be translated into pure predicate logic with equality, at the same time admitting its own sound and complete proof system. A practical aspect of matching logic is that FOL reasoning with equality remains sound, so off-the-shelf provers and SMT solvers can be used for matching logic reasoning. Matching logic is particularly well-suited for reasoning about programs in programming languages that have an operational semantics, but it is not limited to this.

Introduction. In their simplest form, as term templates with variables, patterns abound in mathematics and computer science. They match a concrete, or ground, term if and only if there is some substitution applied to the pattern's variables that makes it equal to the concrete term, possibly via domain reasoning. This means, intuitively, that the concrete term obeys the structure specified by the pattern. We show that when combined with logical connectives and variable constraints and quantifiers, patterns provide a powerful means to specify and reason about the structure of states, or configurations, of a programming language.

Matching logic was born from our belief that programming languages must have formal definitions, and that tools for a given language, such as interpreters, compilers, state-space explorers, model checkers, deductive program verifiers, etc., can be derived from just *one* reference formal definition of the language, which is executable. No other semantics for the same language should be needed.

For example, [3] presents a program verification module of based on matching logic which takes the respective operational semantics of C [4], Java [1], and JavaScript [5] as input and yields automated program verifiers for these languages, capable of verifying challenging heap-manipulating programs at performance comparable to that of state-of-the-art verifiers specifically crafted for those languages.

Matching logic is particularly well-suited for reasoning about programs when their language has an operational semantics. That is because its patterns give us full access to all the details in a program configuration, at the same time allowing us to hide irrelevant detail using existential quantification or separately defined abstractions. Also, both the operational semantics of a language and its reachability properties can be encoded as rules $\varphi \Rightarrow \varphi'$ between patterns, called *reachability rules* in [3,2,7,8], and one generic, language-independent proof system can be used both for executing programs and for proving them correct. In both cases, the operational semantics rules are used to advance the computation. When executing programs the pattern to reduce is ground and the application of the semantic steps becomes conventional term rewriting. When verifying reachability properties, the pattern to reduce is symbolic and typically contains constraints and abstractions, so matching logic reasoning is used in-between semantic rewrite rule applications to re-arrange the configuration so that semantic rules match or assertions can be proved. We refer the interested reader to [3] for full details on our recommended verification approach using matching logic.

Although we favor the verification approach above, which led to the development of matching logic, there is nothing to limit the use of matching logic with other verification approaches, as an intuitive and succinct notation for encoding state properties. For example, one cantake an existing separation logic semantics of a language, regard it as a matching logic semantics and then extend it to also consider structures in the configuration that separation logic was not meant to directly reason about, such as function/exception/break-continue stacks, input/output buffers, etc. For this reason, we here present matching logic as a stand-alone logic, without favoring any particular use of it.

Syntax. Matching logic is a logic centered around the notion of patterns:

Definition 1. *Let (S, Σ) be a many-sorted signature of **symbols**. Matching logic (S, Σ)-**formulae**, also called (S, Σ)-**patterns**, or just (matching logic) **formulae** or **patterns** when (S, Σ) is understood from context, are inductively defined as follows for all sorts $s \in S$:*

$$
\begin{array}{lll}
\varphi_s ::= x \in \mathit{Var}_s & // & \textit{Variable} \\
\quad | \ \sigma(\varphi_{s_1}, ..., \varphi_{s_n}) \ \textit{ with } \sigma \in \Sigma_{s_1 \ldots s_n, s} \quad \textit{(written } \Sigma_{\lambda, s} \textit{ when } n = 0\textit{)} & // & \textit{Structure} \\
\quad | \ \neg \varphi_s & // & \textit{Complement} \\
\quad | \ \varphi_s \wedge \varphi_s & // & \textit{Intersection} \\
\quad | \ \exists x \,.\, \varphi_s \textit{ with } x \in \mathit{Var} \quad \textit{(of any sort)} & // & \textit{Binding}
\end{array}
$$

Let Pattern *be the S-sorted set of patterns. By abuse of language, we refer to the symbols in Σ also as patterns: think of $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ as the pattern $\sigma(x_1 : s_1, \ldots, x_n : s_n)$.*

We argue that the syntax of patterns above is necessary in order to express meaningful patterns, and at the same time it is minimal. Indeed, variable patterns allow us to extract the matched elements or structure and possibly use them in other places in more complex patterns. Forming new patterns from existing patterns by adding more structure/symbols to them is standard and the very basic operation used to construct terms, which are the simplest patterns. Complementing and intersecting patterns allows us to reason with patterns the same way we reason with logical propositions and formulae. Finally, the existential binder serves a dual role. On the one hand, it allows us to abstract away irrelevant parts of the matched structure, which is particularly useful when defining and reasoning about program invariants or structural framing. On the other hand, it allows us to define complex patterns with binders in them, such as λ-, μ-, or ν-bound terms/patterns (to be presented elsewhere).

Semantics. In their simplest form, as terms with variables, patterns are usually matched by other terms that have more structure, possibly by ground terms. However, sometimes we may need to do the matching modulo some background theories or modulo some existing domains, for example integers where addition is commutative or $2 + 3 = 1 + 4$, etc. For maximum generality, we prefer to impose no theoretical restrictions on the models in which patterns are interpreted, or matched, leaving such restrictions to be dealt with in implementations (for example, one may limit to free models, or to ones for which decision procedures exist, etc.).

Definition 2. *A **matching logic** (S, Σ)**-model** M, or just a Σ**-model** when S is understood, or simply a **model** when both S and Σ are understood, consists of:*

1. *An S-sorted set $\{M_s\}_{s \in S}$, where each set M_s, called the **carrier of sort** s **of** M, is assumed non-empty; and*
2. *A function $\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$ for each symbol $\sigma \in \Sigma_{s_1 \ldots s_n, s}$, called the **interpretation** of σ in M.*

Note that symbols are interpreted as relations, and that the usual (S, Σ)-algebra models are a special case of matching logic models, where $|\sigma_M(m_1, \ldots, m_n)| = 1$ for any $m_1 \in M_{s_1}, \ldots, m_n \in M_{s_n}$. Similarly, partial (S, Σ)-algebra models also fall as special case, where $|\sigma_M(m_1, \ldots, m_n)| \leq 1$, since we can capture the undefinedness of σ_M on $m_1, \ldots, m_n$ with $\sigma_M(m_1, \ldots, m_n) = \emptyset$. We tacitly use the same notation σ_M for its extension to argument sets, $\mathcal{P}(M_{s_1}) \times \cdots \times \mathcal{P}(M_{s_n}) \to \mathcal{P}(M_s)$, that is,

$$\sigma_M(A_1, \ldots, A_n) = \bigcup \{\sigma_M(a_1, \ldots, a_n) \mid a_1 \in A_1, \ldots, a_n \in A_n\}$$

where $A_1 \subseteq M_{s_1}, \ldots, A_n \subseteq M_{s_n}$.

Definition 3. *Given a model M and a map $\rho : Var \to M$, called an M-**valuation**, let its extension $\overline{\rho} :$ Pattern $\to \mathcal{P}(M)$ be inductively defined as follows:*

- $\overline{\rho}(x) = \{\rho(x)\}$, *for all* $x \in Var_s$
- $\overline{\rho}(\sigma(\varphi_1, \ldots, \varphi_n)) = \sigma_M(\overline{\rho}(\varphi_1), \ldots \overline{\rho}(\varphi_n))$ *for all* $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ *and appropriate* $\varphi_1, ..., \varphi_n$
- $\overline{\rho}(\neg\varphi) = M_s \setminus \overline{\rho}(\varphi)$ *for all* $\varphi \in \text{PATTERN}_s$
- $\overline{\rho}(\varphi_1 \wedge \varphi_2) = \overline{\rho}(\varphi_1) \cap \overline{\rho}(\varphi_2)$ *for all* φ_1, φ_2 *patterns of the same sort*
- $\overline{\rho}(\exists x.\varphi) = \bigcup\{\overline{\rho'}(\varphi) \mid \rho' : Var \to M,\ \rho'|_{Var\setminus\{x\}} = \rho|_{Var\setminus\{x\}}\} = \bigcup_{a \in M} \overline{\rho[a/x]}(\varphi)$

where " $\setminus$ *" is set difference, "*$\rho|_V$*" is* ρ *restricted to* $V \subseteq Var$*, and "*$\rho[a/x]$*" is map* ρ' *with* $\rho'(x) = a$ *and* $\rho'(y) = \rho(y)$ *if* $y \neq x$*. If* $a \in \overline{\rho}(\varphi)$ *then we say* a ***matches*** φ *(with witness* ρ*).*

It is easy to see that the usual notion of term matching is an instance of the above; indeed, if φ is a term with variables and M is the ground term model, then a ground term a matches φ iff there is some substitution ρ such that $\rho(\varphi) = a$. It may be insightful to note that patterns can also be regarded as predicates, when we think of "a matches pattern φ" as "predicate φ holds in a". But matching logic allows more complex patterns than terms or predicates, and models which are not necessarily conventional (term) algebras.

Note that property "if φ closed then $M \models \neg\varphi$ iff $M \not\models \varphi$", which holds in classical logics like FOL, does not hold in matching logic. This is because $M \models \neg\varphi$ means $\neg\varphi$ is matched by all elements, i.e., φ is matched by no element, while $M \not\models \varphi$ means φ is not matched by some elements. These two notions are different when patterns can have more than two interpretations, which happens when M can have more than one element.

Definition 4. *Pattern* φ *is* ***valid****, written* $\models \varphi$*, iff* $M \models \varphi$ *for all* M*. If* $F \subseteq$ PATTERN *then* $M \models F$ *iff* $M \models \varphi$ *for all* $\varphi \in F$*.* F ***entails*** φ*, written* $F \models \varphi$*, iff for each* M*,* $M \models F$ *implies* $M \models \varphi$*. A* ***matching logic specification*** *is a triple* (S, Σ, F) *with* $F \subseteq$ PATTERN*.*

The journal paper upon which this presentation is based [6] gives a more detailed introduction to the logic, its relation with existing logics, as well as a specialized (sound and complete) proof system for it.

References

1. Denis Bogdănaş and Grigore Roşu. K-Java: A Complete Semantics of Java. In *POPL'15*, pages 445–456. ACM, January 2015.
2. Andrei Ştefănescu, Ştefan Ciobâcă, Radu Mereuţă, Brandon M. Moore, Traian Florin Şerbănuţă, and Grigore Roşu. All-path reachability logic. In *RTA-TLCA'14*, volume 8560 of *LNCS*, pages 425–440. Springer, 2014.
3. Andrei Ştefănescu, Daejun Park, Shijiao Yuwen, Yilong Li, and Grigore Roşu. Semantics-based program verifiers for all languages. In *OOPSLA'16*, pages 74–91. ACM, 2016.
4. Chris Hathhorn, Chucky Ellison, and Grigore Roşu. Defining the undefinedness of C. In *PLDI'15*, pages 336–345. ACM, 2015.
5. Daejun Park, Andrei Ştefănescu, and Grigore Roşu. KJS: A complete formal semantics of JavaScript. In *PLDI'15*, pages 346–356. ACM, 2015.

6. Grigore Roşu. Matching logic. *Logical Methods in Computer Science*, to appear, 2017.
7. Grigore Roşu, Andrei Ştefănescu, Ştefan Ciobâcă, and Brandon M. Moore. One-path reachability logic. In *LICS'13*, pages 358–367. IEEE, 2013.
8. Grigore Rosu and Andrei Stefanescu. Checking reachability using matching logic. In *OOPSLA'12*, pages 555–574. ACM, 2012.

Proving Partial Correctness Beyond Programs

Vlad Rusu

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

Partial correctness is perhaps the most important functional property of algorithmic programs. It can be broadly stated as: on all terminating executions, a given relation holds between a program's inputs and outputs. It has been formalised in several logics, from, e.g, Hoare logics [1] to temporal logics [2].

Partial correctness is also a relevant property for any class of specification that has a notion of terminating execution. For example, communication protocols have both nonterminating executions (all messages are forever lost and re-sent) and terminating executions (all messages sent are properly received). Here, partial correctness may, for instance, require that on all terminating executions, the set of messages corresponding to the transmission of a given file are received *and* the reception of every message is acknowledged.

How can one naturally specify such *generic* partial-correctness properties, and how can one verify them in a *maximally trustworthy* manner? One possibility would be to use Hoare logics, but that solution is not optimal because Hoare logics intrinsically require *programs* (as their deduction rules focus on how program-instructions modify logical predicates), and we are targeting systems specified in formalisms that are *not* programs but more abstract *models*, e.g., one more naturally specifies communication protocols in some version of state-transition systems. Another possibility is to state partial-correctness properties in temporal logic and to use a model checker to prove the temporal formula on a state-transition system specification. This is a better solution, since it stays at a model-abstraction level; however, as we are aiming at trustworthy verification, this is not satisfactory as even in case of a successful verification, one's trust in the result is limited as one does not get independently-checkable verification certificates. Moreover, model checkers are limited to essentially finite-state systems (perhaps up to some data abstraction), a limitation we want to avoid.

Contribution We thus propose a generic approach implemented in the Coq proof assistant [24], a system trustworthy enough to be widely regarded as a *certification* tool, in the sense that Coq proofs are independently machine-checkable certificates. We define and implement therein a notion of *Abstract Symbolic Execution* (hereafter, ASE) to capture a generic notion of symbolic execution for otherwise arbitrary specifications. As property-specification language we adapt *Reachability Logic* [3–7] (hereafter, RL) to any system for which ASE is defined. We propose a new deductive system for this version of RL and prove its soundness both on paper and in the Coq proof assistant [24]; the latter provides us with a Coq-certified RL deductive system. We also prove a relative completeness result for our deductive system, which, although theoretical in nature, also has a practical value, since it amounts to a strategy for applying the proof system that does

succeed on valid RL formulas. Initial examples (proving the Needham-Schroeder-Lowe protocol [11][1]) suggest that the approach is applicable in practice.

Related Work Reachability Logic [3–7] is a formalism initially designed for expressing the operational semantics of programming languages and for specifying programs in the languages in question. Languages whose operational semantics is specified in (an implementation of) RL include those defined in the $\mathbb{K}$ framework [8], e.g., Java and C. Once a language is formally defined in this manner, programs in the language can be formally specified using RL formulas; the typical properties expressible in RL are partial-correctness properties. The verification of such formally-specified programs is performed by means of a sound deductive system, which is also complete relative to an oracle deciding certain first-order theories. Recently, it has been noted that RL is also relevant for other classes of systems, i.e., rewriting-logic specifications [9, 10]. In this paper we adapt RL to an even broader class of specifications - essentially, any specification for which an abstract notion of symbolic execution is defined.

The papers [3–7, 10], which describe several variants of RL (earlier known as *matching logic*[2]). The version of RL that we are here adapting is the *all-paths* version [6], suitable for concurrent nondeterministic systems, in contrast with the *one-path* version [5] for sequential programs.

We note that Coq soundness proofs have also been achieved for various proof systems for RL [5, 6]. Those proofs did not grow into practically usable RL interactive provers, however, because the resulting Coq frameworks require too much work in order to be instantiated even on the simplest programming languages[3]. By contrast, our ambition is to obtain a practically usable, certified RL prover within Coq, directly applicable to formalisms more abstract than programs.

Our approach is based on a generic notion of symbolic execution, an old technique that consists in executing programs with symbolic values rather than concrete ones [13]. Symbolic execution has more recently received renewed interest due to its applications in program analysis, testing, and verification [14–19]. Symbolic execution-like techniques have also been developped for rewriting logic, including rewriting modulo SMT [20] and narrowing-based analyses [21, 22].

Finally, abstract interpretation [23] provides us with a useful terminology (abstract and concrete states, abstract and concrete executions, simulations, etc) which we found most convenient for defining abstract symbolic execution.

References

1. C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12 (10): 576580, 1969.

[1] Extended paper and Coq code is available at `https://project.inria.fr/rlase`.

[2] Matching logic now designates a logic for reasoning on program configurations [12]. These changes in terminology are a side effect of the field being relatively recent.

[3] To our best understanding, obtaining certified RL interactive provers was not the goal of our colleagues; rather, they implemented automatic, non-certified provers.

2. Z. Manna and A. Pnueli The temporal logic of reactive and concurrent systems - specification Springer Verlag, 1992.
3. G. Roşu and A. Ştefănescu. Towards a unified theory of operational and axiomatic semantics. In *ICALP 2012*, Springer LNCS 7392, pages 351–363.
4. G. Roşu and A. Ştefănescu. Checking reachability using matching logic. In *OOPSLA 2012*, ACM, pages 555–574.
5. G. Roşu, A. Ştefănescu, Ş. Ciobâcă, and B. Moore. One-path reachability logic. In *LICS 2013*, IEEE, pages 358–367
6. 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.
7. A. Ştefănescu, D. Park, S. Yuwen, Y. Li, and G. Roşu. Semantics-based program verifiers for all languages. In *OOPSLA'16*, pages 74–91
8. The $\mathbb{K}$ semantic framework. `http://www.kframework.org`.
9. D. Lucanu, V. Rusu, A. Arusoaie, and D. Nowak. Verifying reachability-logic properties on rewriting-logic specifications. In *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer*, Springer LNCS 9200, pages 451–474.
10. Stephen Skeirik, Andrei Stefanescu, and Jose Meseguer. A constructor-based reachability logic for rewrite theories. Technical report, University of Illinois at Urbana-Champaign, 2017. Available at `http://hdl.handle.net/2142/95770`.
11. Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. *Software - Concepts and Tools*, 17(3):93–102, 1996.
12. G. Roşu. Matching logic. In *RTA 2015*, LIPICS volume 36, pages 5–21.
13. J. C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.
14. J. Jaffar, V. Murali, J. Navas, and A. Santosa. TRACER: A symbolic execution tool for verification. In *CAV, 2012*, Springer LNCS 7358, pages 758–766.
15. A. Coen-Porisini, G. Denaro, C. Ghezzi, and M. Pezzé. Using symbolic execution for verifying safety-critical systems. *SIGSOFT Software Engineering Notes*, 26(5):142–151, 2001.
16. C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. EXE: automatically generating inputs of death. In *ACM Conference on Computer and Communications Security 2006*, pages 322–335.
17. K. Sen, D. Marinov, and G. Agha. CUTE: a concolic unit testing engine for C. In *ESEC/FSE 2005*, ACM, pages 263–272,
18. C. Păsăreanu and N. Rungta. Symbolic PathFinder: symbolic execution of Java bytecode. In *ASE 2010*, ACM, pages 179–180.
19. D. Lucanu, V. Rusu, and A. Arusoaie. A generic framework for symbolic execution: A coinductive approach. *J. Symb. Comput.*, 80:125–163, 2017.
20. C. Rocha, J. Meseguer and C. Muñoz. Rewriting modulo SMT and open system analysis. *J. Log. Algebr. Meth. Program*, 86(1):269-297, 2017.
21. J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. Higher-Order and Symbolic Computation 20(1-2):23–160, 2007.
22. K. Bae, S. Escobar and J. Meseguer, Abstract Logical Model Checking of Infinite-State Systems Using Narrowing. *RTA 2013*, LIPICS volume 21, pages 81–96.
23. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL 1977*, ACM, pages 238–252.
24. The Coq proof assistant reference manual. `http://coq.inria.fr`.

Independence-Friendly Logic: Some Applications

Gabriel Sandu

University of Helsinki
gabriel.sandu@helsinki.fi

We shortly present an extension of first-order logic, IF (Independence-Friendly) logic, introduced by Hintikka and Sandu (1989). This logic allows for patterns of dependencies and independencies of quantifiers like

$$\forall x \exists y \forall x' (\exists y' / \{x, y\}) Q(x, x', y, y')$$

intended to express the fact that the last existential quantifier does not depend on the quantifiers $\forall x$ and $\exists y$ but only on the universal quantifier $\forall x'$. IF-formulae have received both a game-theoretical interpretation (games of imperfect information) and an alternative, compositional interpretation in terms of so-called teams (sets of assignments) (Hodges, 1997, Väänänen 2007). The second interpretation has led to various logics of dependence (DL) and independence that have flourished recently. I will shortly review in the first part of the presentation some of the main applications of IF logic and DL logic.

In the second part of the paper I will present a new application of DL logic to the modeling of causality and counterfactuals.

Several competing account of causal and counterfactual statements are known. One of them, the philosophical account, grounds the notion of causality on the notion of counterfactual dependence (David Hume, David Lewis). A different, more recent account coming from empirical sciences, is the so-called manipulationist or interventionist account: it attempts to reduce causality to (human) manipulations or interventions (Pearl, Woodward, etc). More to the point, this approach treats causality in terms of "interventionist counterfactuals", trying to replace intuitions about human interventions with a more general, non-anthropomorphic formal conditions on the notion of intervention.

The most basic formal objects in Pearl's and Woodward's approaches are variables which stand in some sort of dependency relations (causal diagrams). Here the word "variable" is understood as "something that may take different variables". I will show in the presentation how "interventionist counterfactuals" and various causal statements may be analyzed using teams of assignments. This is joint work with Fausto Barbero.

Representable functions in Moisil logic*

Andrei Sipoş[1,2]

[1] Simion Stoilow Institute of Mathematics of the Romanian Academy, P.O. Box 1-764, 014700 Bucharest, Romania,
Andrei.Sipos@imar.ro,
WWW home page: http://imar.ro/~asipos
[2] Faculty of Mathematics and Computer Science, University of Bucharest, Academiei 14, 010014 Bucharest, Romania

Gr. C. Moisil was the first who attempted an algebrization of the many-valued logic of Łukasiewicz's. In 1941, he introduced 3- and 4-valued *Łukasiewicz algebras*, and later generalized them to the n-valued [Moi41] and the ∞-valued case [Moi72]. Some years later, it was shown by A. Rose that this class of algebras is inadequate for the logic above. An alternative was devised by C. C. Chang, who introduced in 1958 [Cha58] the class of *MV-algebras*. Still, one may forcefully argue that Łukasiewicz algebras can still be considered algebras of logic, albeit for a different one, which is nowadays dubbed *Moisil logic*. The algebras themselves came to be known as *Łukasiewicz-Moisil algebras* or plainly *Moisil algebras* – a relatively short introduction is [Cig70], while an exhaustive monograph from the early 1990s is [BoiFilGeoRud91]. Later developments may be found in [Leu08] and [DiaLeu15].

Now, it is interesting to see what exactly does it mean that such algebras are or not adequate for some logic. We work here solely on the finite case, i.e. we fix an n and consider algebras of order $n+1$ with standard model consisting of "truth values" $0, \frac{1}{n}, \cdots, 1$. Since a classical result of Moisil's, paralleling Stone's celebrated result on Boolean algebras, states that any Moisil algebra is isomorphic to a subdirect product of the standard model, it is enough to focus on the structure of the standard model. This is exactly what Rose did: he showed that for $n \geq 4$ the elements $0, 1, \frac{n-1}{n}, 1$ form a Moisil subalgebra of the standard model which is not closed under Łukasiewicz implication, therefore that is not expressible in terms of the Moisil operations.

This raises the question: what functions from the rth power of the standard model to the standard model are represented by Moisil formulas? That is what we are going to answer here. In order to be able to express how we arrived at said answer, we shall firstly delve more deeply into how a Moisil algebra looks like.

A Moisil algebra is a de Morgan algebra endowed with n unary operations $\Delta_1, ..., \Delta_n$ called "nuances" or "Chrysippian endomorphisms" which are required to satisfy certain equational conditions. Generally (i.e. on the logic side), if φ is a formula, then $\Delta_i\varphi$ has the intuitive meaning that φ has the "truth value"

* This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS-UEFISCDI, project number PN-II-RU-TE-2014-4-0730.

greater than $\frac{n-i+1}{n}$. From this property, it may easily be seen that the nuances are progressively "contained", each one in the next. A natural question that we may consider is whether we may get rid altogether of these "classical" nuances and replace them with independent, mutually exclusive ones. Such operations $J_1, ..., J_n$ were introduced in [Cig82] and later used in [DiaLeu15] in order to obtain an alternative formulation and equational axiomatization of Moisil algebras. If we add the operation $J_0 := \Delta_1$, we obtain $n+1$ different mutually exclusive nuances, one for each truth value, satisfying the equation

$$J_i\left(\frac{j}{n}\right) = \delta_{ij}$$

on the standard model, where by δ_{ij} we mean the Kronecker delta.

With this tool now in our hands, we may derive our characterization of the representable functions. If $(a_1, ..., a_r)$ is a possible input of such a function f, then clearly $f(a_1, ..., a_r)$ is an element of a subalgebra generated by $\{a_1, ..., a_r\}$. But that subalgebra consists just of $\{0, 1, a_1, ..., a_r, 1-a_1, ..., 1-a_r\}$. So, if f is representable, then for any $(a_1, ..., a_r)$ we have that $f(a_1, ..., a_r) \in \{0, 1, a_1, ..., a_r, 1-a_1, ..., 1-a_r\}$. The challenge is to show that this condition is actually sufficient. We remember that in the Boolean case, any function is representable by a formula. The tool used to show this is the disjunctive normal form, i.e. from the truth table of the function we produce the disjunction between the possible cases that lead up to the value 1. This is also, in a way, what we are doing here: if we have a function f, we form the disjunction:

$$\bigvee_{(a_1,...,a_r)\in L_{n+1}^r} \big(J_{na_1}(x_1) \wedge \ldots \wedge J_{na_r}(x_r) \wedge s(a_1, ..., a_r, f(a_1, ..., a_r))\big),$$

where the J_i's serve to "separate" the lines of the truth table and the last term serves to identify the value on each such line. For example, if $n = 3$ and $r = 4$, then $s\left(1, 0, \frac{1}{3}, 0, \frac{2}{3}\right) = Nx_3$, where N is the de Morgan negation. The general construction, therefore, is given by:

$$s(a_1, ..., a_r, a) := \begin{cases} 1, & \text{if } a = 1; \\ 0, & \text{if } a = 0; \\ x_i, & \text{if } a \notin \{0, 1\} \text{ and } i = \min\{j \mid a = a_j\}; \\ Nx_i, & \text{if } a \notin \{0, 1, a_1, ..., a_r\} \text{ and } i = \min\{j \mid a = 1 - a_j\}. \end{cases}$$

These ideas help us further in obtaining alternate proofs for the cardinality of the free Moisil algebra of order $n+1$ and for characterizing the representable functions of Łukasiewicz logic.

These results are part of a joint work with Denisa Diaconescu and Ioana Leuştean.

References

[BoiFilGeoRud91] Boicescu, V., Filipoiu, A., Georgescu, G., Rudeanu, S.: Łukasiewicz-Moisil algebras. North-Holland (1991)

[Cha58] Chang, C.C.: Algebraic analysis of many-valued logics. Trans. Amer. Math. Soc., 88:467–490 (1958)
[Cig70] Cignoli, R.: Moisil Algebras. Notas de Logica Matematica, vol. 27, Inst. Mat., Univ. Nacional del Sur, Bahia-Blanca (1970)
[Cig82] Cignoli, R.: Proper n-valued Łukasiewicz algebras as S-algebras of Łukasiewicz n-valued propositional calculi. Studia Logica, 41:3–16 (1982)
[DiaLeu15] Diaconescu, D., Leuştean, I.: Mutually exclusive nuances of truth in Moisil logic. Sci. Ann. Comput. Sci. (special issue dedicated to Sergiu Rudeanu), vol. 25, pp. 69–88 (2015)
[Leu08] Leuştean, I.: A determination principle for algebras of n-valued Łukasiewicz logic. Journal of Algebra 320, 3694–3719 (2008)
[Moi41] Moisil, Gr.C.: Notes sur les logiques non-chryssipiennes. An. Ştiinţ. Univ. Al. I. Cuza Iaşi. Mat., vol. 27, pp. 86–98 (1941)
[Moi72] Moisil, Gr.C.: Essais sur les logiques non-chrysippiennes. Editions de l'Academie de la Republique Socialiste de Roumanie, Bucharest (1972)

A Two Steps Test Suite Generation Approach based on Extended Finite State Machines

Ana Ţurlea

Department of Computer Science, University of Bucharest,
turleaana@gmail.com

Abstract. Using extended finite state machines for test data generation can be a difficult process because we need to generate paths that are feasible and we also need to find input data that traverse a given path. This paper presents a two steps test generation algorithm for extended finite state machines. The first phase produces a set of feasible transition paths that cover all transitions and the second phase generates input sequences that trigger each path. The first step uses a modified multi-objective genetic algorithm (deleting redundant paths and shortening the paths) and the second phase uses a hybrid approach combining genetic algorithms with local search techniques. The multi-objective problem aims to optimize the transitions coverage and the paths feasibility, based on dataflow dependencies.

Keywords: genetic algorithm, extended finite state machines, multi objective genetic algorithm, hybrid genetic algorithm, local search, global search, transitions coverage, paths feasibility, test suite, test data generation

1 Introduction

Finite state machines (FSMs) are widely used in test data generation approaches. A FSM consists of a finite set of states and transitions between states. Each transition has a start state, and end state, an input and produces an output. An extended finite state machine (EFSM) extends the FSM with memory (context variables), guards for each transition and assignment operations. In FSMs all paths are feasible, but the existence of context variables in EFSMs can lead to infeasible paths. Using EFSMs in test data generation, we are dealing with feasibility problems. There are many coverage criteria used in EFSM testing. A set of inputs satisfies the transition coverage criterion if the execution of all test cases leads to all transitions being triggered. Our proposed approach is a two phases algorithm generating feasible paths based on transition coverage criterion and then generate input data for those paths. There are other similar methods, but our technique uses a modified multi-objective algorithm, having two objectives (transition coverage and path feasibility) and a solution shortening operator, for generating a test suite and a hybrid genetic algorithm for the test data generation.

2 Two steps algorithm for test data generation

Phase 1 - Multi-objective Algorithm: Paths generation: The first step uses a multi-objective algorithm and generates different collection of paths, taking into account data dependencies and transition coverage. We use two objectives, that may conflict.

Solution Encoding: a solution is a set of paths. A chromosome has variable length and is composed of genes (paths of fixed length). We put a limit on the total number of genes equal with the number of transitions (as we want to cover all transition we need a number of paths less or equal to the total number of transitions). To encode a path we adapt the encoding used by Kalaji et al. [2]. A path is represented by a sequence of integer values, each number defining a transition.

The Genetic Operators used are inspired from [1] and adapted to our problem. *The Mutation Operator* can change a chromosome in many ways, since we have different characteristics for the individuals. We will apply the following changes, with equal probability: add a random generated gene, replace a gene from a random point, remove a random selected gene, replace a value for a random gene from a random index, exchange materials between two random genes. *The Crossover Operator* creates two new chromosomes from the two existing parents chromosomes. Our algorithm applies two recombination methods with equal probability, as it follows: identify one gene in each parent at the same index (must be a valid index for both parents, since we have variable sized chromosomes) and interchange genes at this position; exchange materials between two genes from a valid random index.

Objective Functions: In this approach we use two objective functions, maximizing the path feasibility and transition coverage. To determine the feasibility of a path we adapt the metric described by Kalaji et al. in [2]. The feasibility metric guides the search towards transition paths that are likely to be feasible using dataflow dependencies among the transitions. The second objective function computes the transition coverage as follows: for each transition we count the occurrences in all paths and return this number if all transitions were covered or INF otherwise. In our experiments INF was equal to 10000.

The execution of the NSGAII-modified algorithm produces a population of solutions. Each solution represent a set of paths. After each evolution of the genetic algorithm we optimize each individual removing redundant paths and transitions as follows: sort the paths ascending according to the feasibility objective function; for each path we try to remove the last transitions if thy were covered already in the previous visited paths and mark as visited the remaining transitions. After modifying the chromosomes, we replace the current population and continue the evolution until the stopping condition is reached.

Phase 2: Hybrid Genetic Algorithm for Test Data Generation GAs tend to the global optimum, but may take a long time to converge to the optimal solution. To overcome this problem, in the second step of this method we use a Hybrid Genetic Algorithm (HGA) proposed in our previous work [4]. For each path generated at the first step, we generate input data that validates the guards and

triggers all transitions. This method uses a genetic algorithm combined with a local search method.

The algorithm ends when the stop criteria is reached or when the maximum number of evolutions is exceeded. After the selection step, a new generation is created using recombination, crossover and mutation. After each evolution of the genetic algorithm, the best chromosome is selected to be improved locally. Applying local search we may find a better individual. If local search returns a chromosome with better fitness function this new one will be added to the population and it will be used in the next evolution. If there is no improvement, the algorithm continues with the next evolution.

In our previous work [4] we used a Alternate Variables Method, as it follows: for each chromosome selected by search algorithm, we take all genes in the reverse order. For each gene we start exploratory moves (+1 and 1 for integer variables) and decide in which direction to search for better values. A successful move consists in improving the fitness value. If a +1 move was successful, then, with each iteration i of the search, we add = 2 i to that gene. Otherwise, if the 1 move was successful, we subtract from the gene. We iterate the search as long as the new chromosome (with the modified gene) has a better fitness function. The fitness function used in this approach is: $fitness = approach\_level + normalized\_branch\_level$ ($f = al + nbl$). $branch\_level$ computes, for the predicate that is not satisfied, how close the predicate was to being true, using Tracey's objective functions [3]. The normalization function is $norm : [0, 101] \rightarrow [0, 1], norm(d) = 1 - 1.05^{-d}$. More details about the algorithm ca be found in [4].

3 Conclusion

In this paper we propose a two steps algorithm for test data generation for EFSMs using a customized multi-objective algorithm to generate a transition coverage test suite and a hybrid genetic algorithm to generate input data for each path. This approach is different from the existing methods, customizing the genetic operators, optimizing the solutions by deleting redundant paths and path transitions and by using the combination of genetic algorithms and local search methods. The second phase of the algorithm may take a long time to find a solution, especially for paths with grater length. To overcome this problem, we decided to shorten the paths, to find paths with smaller complexity and, in the same time, to cover all transitions. Experiments have shown that there is easy to generate input data for the sets of paths found at the first step of the algorithm.

References

1. Asoudeh, N., Labiche, Y.: Multi-objective construction of an entire adequate test suite for an EFSM. In: Proceedings of the 2014 IEEE 25th International Symposium on Software Reliability Engineering. pp. 288–299. ISSRE '14, IEEE Computer Society, Washington, DC, USA (2014), http://dx.doi.org/10.1109/ISSRE.2014.14

2. Kalaji, A.S., Hierons, R.M., Swift, S.: An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models. Information & Software Technology 53(12), 1297–1318 (2011)
3. Tracey, N.J., Clark, J.A., Mander, K., McDermid, J.A.: An automated framework for structural test-data generation. In: The Thirteenth IEEE Conference on Automated Software Engineering, ASE 1998, Honolulu, Hawaii, USA, October 13-16, 1998. pp. 285–288 (1998), https://doi.org/10.1109/ASE.1998.732680
4. Turlea, A., Ipate, F., Lefticaru, R.: A hybrid test generation approach based on extended finite state machines. In: Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2016 18th International Symposium on. pp. 173–180. IEEE (2016)

A hybrid-logic approach to dynamic networks of interactions

Ionuţ Ţuţu and José Luiz Fiadeiro

Department of Computer Science, Royal Holloway University of London
ittutu@gmail.com, jose.fiadeiro@rhul.ac.uk

The advent of the Web and of new distributed frameworks such as service-oriented and cloud-computing architectures has led to a paradigm shift from human-based engineering of software systems to reconfigurable systems that are endowed – by middleware and network infrastructures – with the capability of organising themselves [5]. As distributed applications execute, they can trigger changes in the topology of the networks that support them: new links, nodes, or even entire subnetworks can be created at run time, often without human intervention, in response to the need for external resources or services [9,6].

Over the past few years, several research initiatives have proposed formal-methods approaches that address different aspects of the new generation of dynamic systems that are now operating in cyberspace. One of the most prominent is the mathematical model for the specification and verification of reconfigurable systems based on hybrid(ized) logics [1,2,11]. In their most basic form, these are logics obtained by enriching ordinary modal logics with nominals – symbols that name individual states (possible worlds) in Kripke models – and a dedicated satisfaction operator @ that enables a change of perspective from the current state to one that is named. The development of hybrid logics originated in Arthur Prior's work in the 1960s [14], and recently has been the subject of a renewed interest from the formal specification and verification community [13,3,8,7], part of which is by virtue of the use of hybrid logics in reasoning about reconfigurations.

In a nutshell, the hybrid-logic approach builds on the intuitive idea that system configurations (and the functionalities associated with them) can be regarded as local models of a Kripke structure; moreover, they can change simply by switching from one mode of operation to another via an accessibility relation. The key advancement here lies in understanding that the features characteristic to basic hybrid logic can be developed, through a process known as hybridization [12], on top of an arbitrary logical system, which is used for expressing configuration-specific requirements. This means that, depending on the base logical system, configurations can be captured, for example, as algebras, relational structures or, when the hybridization process is iterated, even as Kripke models.

In this work, we focus on the use of hybrid logics for specifying and reasoning about those reconfigurable systems whose configurations, or local models, are given by networks of interacting actors. These should be understood in the wider sense of Latour's actor-network theory [10]: actors are cyberphisical entities that have shared agency, from people, to objects, to locations; they interact through so-called channels, which account, for instance, of observations that an actor may

make of another, of control that an actor may exert on another, or of movement of an actor (e.g. a person) inside another (e.g. a location).

The ordinary hybridization process outlined above yields logical systems that are suitable for dealing with the structural aspects of actor networks. They can be naturally used, for example, to give faithful descriptions of the shapes of networks, of the (states of the) actors involved, or of the channels through which interactions can take place. Yet, in contrast to the general adequacy of hybrid logics to cope with reconfigurations, the challenge lies precisely in capturing the way networks evolve over time. This is because, for such dynamic networks of interactions, the higher-level reconfigurations of networks and the lower-level interactions between actors are closely intertwined: for instance, reconfigurations are always triggered by interactions, and they may result in new opportunities for interaction. In other words, dynamic networks of interactions do not exhibit the full orthogonality that the hybridization process requires between the details of the underlying logical system and the hybrid features to be developed. We therefore propose a new kind of hybridization that takes as input a hybrid logic, which can be used in specifying network states, and produces another logic with hybrid features, which can be used in specifying network reconfigurations. But the resulting logic is not hybrid: even though the accessibility relations model reconfigurations, they do not link networks directly; instead, they define inter-network connections between different actor states. We discuss what are the implications from a specification-theoretic perspective of this important distinction, and show that, much like ordinary hybrid logics [4], the ones that we propose here can also be encoded into first-order logic, assuming that the base logical system admits such an encoding. This provides preliminary proof-theoretic support for the hybrid specifications of dynamic networks of interactions.

References

1. Blackburn, P.: Representation, reasoning, and relational structures: a hybrid logic manifesto. Logic Journal of the IGPL 8(3), 339–365 (2000)
2. Braüner, T.: Hybrid logic and its Proof-Theory, Applied Logic Series, vol. 37. Springer (2011)
3. Diaconescu, R.: Quasi-varieties and initial semantics for hybridized institutions. Journal of Logic and Computation 26(3), 855–891 (2016)
4. Diaconescu, R., Madeira, A.: Encoding hybridized institutions into first-order logic. Mathematical Structures in Computer Science 26(5), 745–788 (2016)
5. Fiadeiro, J.L.: The many faces of complexity in software design. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity, pp. 3–47. Springer (2012)
6. Fiadeiro, J.L., Lopes, A., Bocchi, L.: An abstract model of service discovery and binding. Formal Aspects of Computing 23(4), 433–463 (2011)
7. Găină, D.: Birkhoff style calculi for hybrid logics. Formal Aspects of Computing pp. 1–28 (in press)
8. Găină, D.: Foundations of logic programming in hybrid logics with user-defined sharing. Theoretical Computer Science (in press)
9. Kon, F., Costa, F., Blair, G., Campbell, R.H.: The case for reflective middleware. Communications of the ACM 45(6), 33–38 (2002)

10. Latour, B.: Reassembling the Social: An Introduction to Actor-Network Theory. Oxford University Press (2005)
11. Madeira, A.: Foundations and techniques for software reconfigurability. PhD thesis, MAP-i (Minho, Aveiro, Porto) (2013)
12. Martins, M.A., Madeira, A., Diaconescu, R., Barbosa, L.S.: Hybridization of institutions. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) Algebra and Coalgebra in Computer Science. Lecture Notes in Computer Science, vol. 6859, pp. 283–297. Springer (2011)
13. Neves, R., Madeira, A., Martins, M.A., Barbosa, L.S.: Proof theory for hybrid(ised) logics. Science of Computer Programming 126, 73–93 (2016)
14. Prior, A.: Past, Present and Future. Oxford University Press (1967)

Author Index