WADT 2016

23rd International Workshop on Algebraic Development Techniques

Submitted Abstracts

http://cs.swansea.ac.uk/wadt16
Preface

This volume contains the abstracts of the contributions to the 23rd International Workshop on Algebraic Development Techniques, WADT’16 held on September 20-23, 2016 in Gregynog, UK.

The algebraic approach to system specification encompasses many aspects of the formal design of software systems. Originally born as formal method for reasoning about abstract data types, it now covers new specification frameworks and programming paradigms (such as object-oriented, aspect-oriented, agent-oriented, logic and higher-order functional programming) as well as a wide range of application areas (including information systems, concurrent, distributed and mobile systems).

Typical, but not exclusive topics of interest are:

– Foundations of algebraic specification
– Other approaches to formal specification, including process calculi and models of concurrent, distributed and mobile computing
– Specification languages, methods, and environments
– Semantics of conceptual modelling methods and techniques
– Model-driven development
– Graph transformations, term rewriting and proof systems
– Integration of formal specification techniques
– Formal testing and quality assurance
– Validation and verification

The WADT steering committee consists of:

Andrea Corradini University of Pisa
Jose Fiadeiro Royal Holloway
Rolf Hennicker Ludwig-Maximilians-Universität München
Hans-Jörg Kreowski University of Bremen
Till Mossakowski Universitaet Magedeburg
Fernando Orejas Technical University of Catalonia
Francesco Parisi-Presicce Sapienza Università di Roma
Markus Roggenbach (Chair) Swansea University
Grigore Rosu University of Illinois at Urbana-Champaign
Andrzej Tarlecki Warsaw University

The workshop takes place under the auspices of IFIP WG 1.3, and is sponsored by IFIP TC1 and Swansea University. The event is organised by the Department of Computer Science at Swansea University.

September 6, 2016
Swansea

Markus Roggenbach
Phillip James
## Table of Contents

Paramaterised Verification for Multi-Agent Systems .......................... 1  
   Alessio Lomuscio (Invited Talk)

Theorising Monitoring and Surveillance ........................................ 3  
   John V Tucker, Victoria Wang and Kenneth Johnson (Invited Talk)

The Distributed Ontology, Model and Specification Language DOL ...... 5  
   Till Mossakowski (Invited Tutorial)

Signed Meadow Valued Probability Functions ............................... 10  
   Jan Bergstra and Alban Ponse

Towards a Formulation of the Modularization Theorem for  
Presentations in Default Logics ..................................................... 12  
   Valentin Cassano, Carlos Gustavo Lopez Pombo and Tom Maibaum

Creative Processes in Service-Oriented Computing .......................... 14  
   Claudia Elena Chirita and José Luiz Fiadeiro

Graph Models for Capital Markets .................................................. 16  
   Nneka Ene, Maribel Fernandez and Bruno Pinaud

Providing a Semantics and Modularisation Construots for Event-B  
using Institutions ........................................................................... 18  
   Marie Farrell, Rosemary Monahan and James Power

On the Most Suitable Axiomatization of Signed Integers Using Free  
Constructors ................................................................................... 20  
   Hubert Garavel

Behavioural Semantics for the Dynamic Logic with Binders ............... 22  
   Rolf Hennicker and Alexandre Madeira

Towards Critical Pair Analysis for the Graph Programming Language  
GP 2 ......................................................................................... 24  
   Ivaylo Hristakiev and Detlef Plump

A Calculus of Virtually Timed Ambients ......................................... 27  
   Einar Broch Johnsen, Martin Steffen and Johanna Beate Stumpf

On the Structural Link between Ontologies and Their Organised Data Sets  
Ridha Khedri and Alicia Marinache ................................................. 29

About partiality in institutions (co-)morphisms ............................... 31  
   Agustín Eloy Martínez Suáé, Carlos Gustavo Lopez Pombo, Fabio Gadducci and Tom Maibaum
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selecting Colimits for Parameterisation and Networks of Specifications</td>
<td>33</td>
</tr>
<tr>
<td>\textit{Till Mossakowski, Mihai Codescu and Florian Rabe}</td>
<td></td>
</tr>
<tr>
<td>Epistemic Reasoning about Operational Lies</td>
<td>35</td>
</tr>
<tr>
<td>\textit{Mohammadreza Mousavi and Mahsa Varshosaz}</td>
<td></td>
</tr>
<tr>
<td>The monad of continuous evolutions</td>
<td>38</td>
</tr>
<tr>
<td>\textit{Renato Neves}</td>
<td></td>
</tr>
<tr>
<td>(Asymmetric) combination of logics is functorial</td>
<td>40</td>
</tr>
<tr>
<td>\textit{Renato Neves, Alexandre Madeira, Luis Barbosa and Manuel A. Martins}</td>
<td></td>
</tr>
<tr>
<td>Generic Hoare Logic for Order-Enriched Effects with Exceptions</td>
<td>42</td>
</tr>
<tr>
<td>\textit{Christoph Rauch, Sergey Goncharov and Lutz Schröder}</td>
<td></td>
</tr>
<tr>
<td>Towards a formal framework for analyzing stream processing systems</td>
<td>44</td>
</tr>
<tr>
<td>\textit{Adrian Riesco, Miguel Palomino and Narciso Marti-Oliet}</td>
<td></td>
</tr>
<tr>
<td>Formal Specification of the Internet of Things</td>
<td>47</td>
</tr>
<tr>
<td>\textit{Edel Sherratt}</td>
<td></td>
</tr>
<tr>
<td>Foundations of Graph Transformation as a Logic-Programming Language</td>
<td>49</td>
</tr>
<tr>
<td>\textit{Ionut Tutu and José Luiz Fiadeiro}</td>
<td></td>
</tr>
<tr>
<td>Formalising Modular SOS Labels in Isabelle/HOL</td>
<td>51</td>
</tr>
<tr>
<td>\textit{Ferdinand Vesely}</td>
<td></td>
</tr>
<tr>
<td>Semantics for non-incremental reconfigurations of Asynchronous Relational Networks</td>
<td>53</td>
</tr>
<tr>
<td>\textit{Ignacio Vissani and Carlos Gustavo Lopez Pombo}</td>
<td></td>
</tr>
<tr>
<td>Algebraic Databases</td>
<td>55</td>
</tr>
<tr>
<td>\textit{Ryan Wisnesky}</td>
<td></td>
</tr>
<tr>
<td>Design and Validation of the P-Store Replicated Data Store in Maude</td>
<td>57</td>
</tr>
<tr>
<td>\textit{Peter Ölveczky}</td>
<td></td>
</tr>
</tbody>
</table>
Paramaterised Verification for Multi-Agent Systems

Alessio Lomuscio

Department of Computing, Imperial College London

Multi-agent systems (MAS) are distributed autonomous systems in which the components, or agents, act autonomously in order to reach private or common goals. Logic-based specifications for MAS typically do not refer only to the agents’ temporal evolution, but also their knowledge, intentions, strategic abilities, etc.

In this talk I will survey some of our work on model checking MAS against agent-based specifications. I will begin by reporting algorithms for symbolic model checking against epistemic and strategic specifications [1, 2]. I will then demonstrate MCMAS [3, 4], an open-source BDD-based model checker supporting these specification languages. A case study concerning the verification of diagnosability and fault-tolerance of an autonomous underwater vehicle will be discussed [5].

I will then consider the case of MAS where the number of agents is unbounded at design time and introduce the parameterised model checking problem for these systems. I will present recent results that enable us to establish sufficient conditions for determining a cut-off of a MAS [6], i.e., the number of agents that need to considered for verifying a MAS composed of any number of components. I will conclude by relating cut-offs to other techniques, notably counter-abstraction, to establish properties of swarm systems [7].

Acknowledgements: The work presented in this talk was partly funded by the EPSRC Research Project “Trusted Autonomous Systems”.

References

5. Ezekiel, J., Lomuscio, A., Molnar, L., Veres, S.: Verifying fault tolerance and self-
diagnosability of an autonomous underwater vehicle. In: Proceedings of the 22nd
International Joint Conference on Artificial Intelligence (IJCAI11), AAAI Press
(2011) 1659–1664
Artificial Intelligence 234 (2016) 152–189
Theorising Monitoring and Surveillance

John V Tucker¹, Victoria Wang², and Kenneth Johnson³

¹ College of Science, Swansea University, UK
² Institute of Criminal Justice Studies, University of Portsmouth, UK
³ School of Computer and Mathematical Sciences, Auckland University of Technology, New Zealand

Our machines, products, utilities, and environments have long been monitored by embedded software systems. Our professional, commercial, social and personal lives are also subject to monitoring as they are mediated by software systems. Data on nearly everything now exists, waiting to be collected and analysed for all sorts of reasons. The rising tide of data raise the questions:

What is monitoring? Do diverse and disparate monitoring software systems have anything in common?

We answer these questions by proposing an abstract conceptual framework for studying monitoring. We argue that it captures a structure common to many different monitoring practices, and that from it detailed formal models can be derived, customised to applications. The framework formalises the idea that monitoring is a process that observes the behaviour of people and objects in a context. The entities and their behaviours are represented by abstract data types and the observable attributes by logics. Since monitoring usually has a specific purpose, we extend the framework with protocols for detecting attributes or events that require interventions and, possibly, a change in behaviour. Our analysis is illustrated by examples of monitoring in computing and criminal justice.

The lecture reports on a programme of research designed to introduce formal models into theory making about social phenomena to do with digital society [3–5]. Our understanding of monitoring underpins answers to socio-technical questions such as

What is surveillance? Do diverse and disparate surveillance software systems have anything in common? What constitutes privacy relative to monitoring and surveillance?

We will touch on these questions with reference to the state of surveillance studies. The monitoring theory also complements research into analogue digital systems used in control [2] and physical experiments [1].

References


The Distributed Ontology, Model and Specification Language – DOL

Till Mossakowski

Institute of Cooperating Intelligent Systems, Otto-von-Guericke-University Magdeburg, Germany

Over the last decades, the WADT community has studied the formal specification of software (and hardware) in great detail [9, 1, 41]. One important aspect is the structuring of specifications in a modular way [42], which has been covered in specification languages like CLEAR [6], OBJ [17], ASL [45] and many others. Here, a powerful abstraction is the notion of institution, introduced by Goguen and Burstall [16]. It enables the study of concepts and languages for structured specifications in a way that is completely independent of the underlying logical system — the only condition being that the logical system is formalised as an institution, which is a rather mild requirement. Such an institution independent kernel language for structured specifications has been introduced in [40], and based on this, later the Common algebraic specification language Casl [3, 37] has been standardised.

While all these developments, including Casl, focus on formal specifications, the approach of providing an institution-independent language for the structuring of logical theories (or more precisely, finite presentations of these) can be applied to other areas as well.

In particular, in research on ontologies, the notion of conservative extension has been cited from the algebraic specification literature (e.g. [24]) und used for the notion of ontology module extraction in various description logics (see e.g. [20], and [18] for an institution-independent generalisation). The existing multitude of ontology languages like OWL and its sublogics, RDF, RDFS and their relations have been captured using institutions [22, 31].

Moreover, using the notion of heterogeneous multi-logic specification developed in [2, 11, 43, 13, 26, 27, 21, 36], a program for the institution-based formalisation of UML multi-viewpoint models has been formulated [8, 19, 7]. Note that model here is to be understood in the sense of model-driven engineering (MDE), to be distinguished from models in the sense of logical model theory (and institutional specification theory). In order to avoid confusion, we henceforth call the former MDE models.

Based on this observation of similarities between ontologies, MDE models and specifications, the Distributed Ontology, Model and Specification Language (DOL) has been proposed and adopted as an OMG standard [33, 32, 30]. Ontologies, MDE models and specifications are commonly abbreviated by the acronym OMS. Hence, DOL can be seen as a language for building OMS in a structured way and expressing their relations. Casl already provides several structuring constructs, e.g. (possibly conservative or definitional) extensions, unions, translations and hidings. DOL extends these in several ways:
theory-level semantics Casl uses a model-theoretic semantics, that is, a specification denotes a signature and a class of models over that signature. DOL adopts this, but also features theory-level semantics [39, 41] for certain constructs like module extraction or filtering.

reduction Casl features hiding of a specification (aka OMS) along a signature morphism, corresponding to the restriction to an export interface. DOL features three more similar operations:

module extraction extraction of a sub-OMS such that the original OMS is a conservative extension [20]. The extracted module may extend the given restriction signature.

approximation gives the theorems visible over the restriction signature and corresponds to the theory-level semantics of hiding [39, 41]. The problem of capturing this theory by a finite presentation has been studied for ontology languages under the terms forgetting and uniform interpolation [44, 23].

filtering extraction of a sub-OMS consisting of all sentences that actually are formed over the restricted signature [38].

minimization whereas free specifications in Casl allow the selection of the least interpretation of e.g. predicates, minimization allows the selection of all minimal interpretations, following McCarthy’s circumscription [25]. Also, the duals (cofree and maximal OMS) are included. Cofree OMS can be used for coinductive specification of process types, like in CoCasl [34].

refinement simple refinements are specification morphisms [41] (logically: interpretations of theories [14], in terms of OBJ [17] and Casl [3, 37]: views). The refinement language of [29] is included into DOL, that is, certain operation on refinements are available, like composition and extension. However, neither architectural specifications nor branching refinements are included, because their semantics is still subject of ongoing research.

equivalence OMS can be declared to equivalent, if they have a common definitional extension [35, 21]

alignment this notion is a relational generalisation of signature morphisms (which are typically functional in nature) [15, 12, 46]. Between a symbol from the source OMS and one from the target OMS, different relations can be specified.

networks networks generalise distributed specifications [35], networks of alignments [15] and distributed description logics [4]. They provide also a formal notion of viewpoint specifications, e.g. collections of UML diagrams providing different views on a system. A model of a network is a family of models of the involved OMS that is compatible along the mappings of the network. Networks can also be refined.

combination When alignments are normalised to spans or Ws of signature morphisms, networks correspond to diagrams (in the sense of category theory) of OMS [10]. A network can be combined into a single OMS by taking its colimit. Under suitable amalgamation conditions, the combination captures the model class of the network and thus can be used for reasoning about the networks.
entailments between OMS, or of an OMS by a network.

heterogeneity support for multiple logics (institutions) as discussed above:
OMS can be translated along institution comorphisms, be projected along
institution morphisms. Also, approximations, refinements and alignments
can be heterogeneous.

internet compatibility all names are full URLs resp. IRIs, and prefix maps
allow the convenient abbreviation of these.

This completes the overview of DOL, which is currently being finalised. The DOL
standard document and further information can be found at dol-omg.org. Tool
support for (an increasing part of) DOL is provided by the Heterogeneous Tool
Set (hets.eu) and Ontohub (ontohub.org). Sample DOL documents can be
found at onthub.org/dol-examples.

Future work will address the further extension of DOL, e.g. with queries and
architectural refinements. Also, the extension of proof support from standard
structured specifications [5, 28] to the whole of DOL is an important task.

Acknowledgements The author wishes to thank the community that has de-
developed DOL, in particular Mihai Codescu, Michael Gruninger, Maria Keet,
Alexander Knapp, Oliver Kutz, Christoph Lange and Fabian Neuhaus, as well
as those OMG members that have provided valuable feedback, in particular
Conrad Bock, Elisa Kendall, Pete Rivett and Ed Seidewitz.

References

3. M. Bidoit and Peter D. Mosses. CASL — the Common Algebraic Specification
5. Tomasz Borzyszkowski. Logical systems for structured specifications. Theoretical
6. Rod M. Burstall and Joseph A. Goguen. The semantics of CLEAR, a specification
7. Daniel Calegari, Till Mossakowski, and Nora Szasz. Heterogeneous verification in
31. Till Mossakowski and Oliver Kutz. The Onto-Logical Translation Graph. In Oliver Kutz and Thomas Schneider, editors, Modular Ontologies. IOS, 2011.
Signed Meadow Valued Probability Functions
(Extended abstract)

Jan A. Bergstra and Alban Ponse
Informatics Institute, University of Amsterdam
https://staff.fnwi.uva.nl/{j.a.bergstra/,a.ponse/}

1. Introduction

We provide an axiomatization of a probability function on a Boolean algebra.\(^1\) Meadows have been introduced in [4] as a finitely based variety that constitutes the basis of an approach to the algebraic specification of datatypes for numbers. For recent information on meadows we refer to [3].

In [2] the specification of meadows, named \(Md\) and consisting of ten axioms, has been extended with an additional function \(s(\_\_)\) that returns the sign (\(-1, 0,\) or \(1\)) of a number. Six axioms named \(Sign\) specify the sign function on top of \(Md\). A meadow equipped with a sign function that satisfies \(Sign\) is called a signed meadow, and in [3] it was shown that \(Md + Sign\) is complete for the equational theory of the signed meadow \((\mathbb{R}_0, s)\) of reals.

We apply these observations to the development of an equational theory of probability and we propose an equational loose specification of probability functions. The two-sorted structures serving as models of the specification consist of a signed meadow valued probability function defined on a Boolean algebra. The Boolean algebra serves as an event space and the probability function defined on it produces elements of (values in) a signed meadow that serve as probabilities. Special focus is on the case where values are chosen in \((\mathbb{R}_0, s)\).

2. Valuated Boolean algebras and probability functions

A Boolean algebra \((B, +, \cdot, \neg, 1, 0)\) can be defined as a system with at least two elements such that \(\forall x, y, z \in B\) the postulates of Boolean algebra are valid. Because we want to avoid overlap with the operations of a meadow, we will consider Boolean algebras with notation from propositional logic, thus consider \((B, \lor, \land, \neg, \top, \bot)\) and adopt the axioms given in [5], further referred to as \(BA\).

A Boolean algebra can be equipped with a valuation \(v\) that assigns to its elements values in a signed meadow. We consider the special case where the valuation function is a probability function by requiring that the valuation satisfies the Kolmogorov axioms for probability functions cast to the setting of signed meadows.

Table 1 provides equational axioms for a probability function with name \(P\) (\(PF_P\)). These axioms capture a version of Kolmogorov’s axioms in a setting of

---

\(^1\) This work is based on http://arxiv.org/abs/1307.5173v2.
Table 1. \(PF_P\), axioms for a probability function with name \(P\)

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) (P(\top) = 1)</td>
<td>(P) of (\top) is 1</td>
</tr>
<tr>
<td>(2) (P(\bot) = 0)</td>
<td>(P) of (\bot) is 0</td>
</tr>
<tr>
<td>(3) (P(x) = s(P(x)) \cdot P(x))</td>
<td>(P) satisfies the sign property</td>
</tr>
<tr>
<td>(4) (P(x \lor y) = P(x) + P(y) - P(x \land y))</td>
<td>(P) satisfies the addition property</td>
</tr>
<tr>
<td>(5) (P(x \land y) \cdot P(y) \cdot P(y)^{-1} = P(x \land y))</td>
<td>(P) satisfies the multiplication property</td>
</tr>
</tbody>
</table>

Boolean algebra. Axiom (3) expresses that the sign of \(P(x)\) is nonnegative, thus \(P(x) \geq 0\).

A valuated Boolean algebra equipped with a valuation \(P\) in some signed meadow \(\mathbb{M}\) that satisfies all axioms of \(BA + Md + Sign + PF_P\) is called a \(K(\mathbb{M}, P)\)-structure. In the case that \(\mathbb{M} = (\mathbb{R}_0, s)\) we write \(K(\mathbb{R}_0, P)\).

3. Results

A first result is that Bayes’ theorem follows from \(BA + Md + Sign + PF_P\). Next, on the basis of the completeness of \(Md + Sign\) for \((\mathbb{R}_0, s)\), we prove that \(BA + Md + Sign + PF_P\) axiomatises the equational theory of \(K(\mathbb{R}_0, P)\). For a specific choice for the event space \(E\), we provide a second completeness theorem that can be used in the setting for probabilistic reasoning about concrete examples, and we analyse an example derived from [1]. Finally, we discuss two generalisations of our approach. First we consider the notion of probability function space and investigate the possibility of updating a current probability function, corresponding to evidence being found that some event is certainly true. We find that repeated updates are possible only if the associated events are independent. Secondly, we consider multi-dimensional probability functions and the BCHS-inequalities, and discuss the relation with the one-dimensional case.

References

Towards a Formulation of the Modularization Theorem for Presentations in Default Logics

Valentín Cassano\textsuperscript{1}, Carlos G. Lopez Pombo\textsuperscript{2,3}, and Thomas S.E. Maibaum\textsuperscript{1}

\textsuperscript{1} Department of Computing and Software, McMaster University, Canada
\textsuperscript{2} Departamento de Computación, Universidad Nacional de Buenos Aires, Argentina
\textsuperscript{3} Consejo Nacional de Investigaciones Científicas y Tecnológicas (CONICET)

Abstract With an origin in Reiter’s seminal 1980 paper ‘A Logic for Default Reasoning’ (see [1]), default logics (DLs for short) are a sub-family of the logical approaches to nonmonotonic reasoning. DLs have received a great deal of attention and have accomplished much from the perspective of the study of nonmonotonic consequence (see [2]). In comparison, with some noteworthy exceptions (see [3,4,5,6]), the same cannot be said about the study of presentations of theories in DLs (default presentations for short) and of morphisms of said presentations (concepts analogous to those of algebraic specifications in classical logical systems). The latter opens an interesting niche for research if, light of the classics [7,8,9,10,11], we wish to design and construct default presentations that are not indivisible wholes but that result from “putting together” default presentations that are more tractable and easier to understand. Intending to fulfill this desideratum, we explore a formulation of default presentations, of their semantics, and of morphisms of default presentations. What we have in mind is the presentation of conditions that guarantee the existence of morphisms of default presentations, and a formulation and proof of a version of the so-called Modularization Theorem of [10,12] for default presentations. The use of DLs for reasoning about requirements (see [13]) provides an interesting area of application for our research efforts. We briefly explain our main ideas below.

The Modularization Theorem was first presented in [14] as the main result of an investigation into the conditions that need to be added to an entailment relation to support an account of specification, where specifications are viewed as presentations of theories over the entailment relation. These ideas were worked out in detail in [10] and [12] for specifications in First-Order Logic (FOL for short) and extended to specifications in general logical systems later on in [15].

The Modularization Theorem is interesting and relevant in the world of formal specifications because it is a basic tool for guaranteeing the preservation of modular structure in the stepwise development of specifications (see [10,12]). Borrowing an example from [12], in FOL, consider a situation in which: (i) a presentation \(\langle \Sigma_3, \Phi_3 \rangle\) is a conservative extension of a presentation \(\langle \Sigma_1, \Phi_1 \rangle\), indicated as \(\langle \Sigma_1, \Phi_1 \rangle \hookrightarrow \langle \Sigma_3, \Phi_3 \rangle\), and (ii) a presentation \(\langle \Sigma_2, \Phi_2 \rangle\) is an interpretation of \(\langle \Sigma_1, \Phi_1 \rangle\) along a morphism \(m : \Sigma_1 \to \Sigma_2\) of signatures, indicated as \(\langle \Sigma_1, \Phi_1 \rangle \overset{m}{\rightarrow} \langle \Sigma_2, \Phi_2 \rangle\) (see Fig. 1). In [12] it is argued that such a situation occurs naturally when composing implementations of presentations or instantiating parametrized presentations. The Modularization Theorem for FOL specifications allows us to materialize the top right corner of Fig. 1 by constructing
Fig. 1: The Modularization Theorem for FOL specifications

a presentation \((\Sigma_4, \Phi_4)\) that is a conservative extension of \((\Sigma_3, \Phi_3)\) and that is an interpretation of \((\Sigma_1, \Phi_1)\) along a morphism \(m' : \Sigma_3 \rightarrow \Sigma_4\) of signatures.

In the terminology of [12], the Modularization Theorem allows us to compose implementations of presentations and to instantiate parametrized presentations.

Against this background, and motivated by the use of DLs for reasoning about requirements: Is there a version of the Modularization Theorem for default presentations, where a default presentation is a theory presentation in a DL?

References

Creative Processes in Service-Oriented Computing

Claudia Elena Chirită and José Luiz Fiadeiro

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

Service-oriented computing is a recent computational paradigm that supports the development of complex software systems based on dynamic reconfigurations of networks [7]. These reconfigurations can be seen as cooperation-and-competition interactions between the entities of a system. They are triggered by requests of external resources or functionalities, and governed by specialised run-time mechanisms that enable applications to meet their business goals by discovering, selecting and binding to other entities, called service modules, which act as external providers.

Service-oriented systems are based on non-mereological composition, and hence should be seen as more than the sum of their parts [6]. One salient feature of these complex 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. Moreover, the dynamicity of the reconfigurations is endogenous, intrinsic to the systems, as their evolution is not driven by external factors such as the change of the environment, but originates from the design of the components themselves.

This kind of emergent behaviour is similar to what has been observed for computational creative systems, and in particular for music improvisations [1]. This leads us to explore the relationship between service-oriented computing and free-jazz performances. More precisely, we postulate that the dynamic reconfigurations of networks and free-jazz compositions are governed by the same principles, and that both can be regarded as instances of more general phenomena that can be formalized over many-valued logics.

We build on the idea that an improvisation can be seen as a collection of music phase spaces as in [2] that organise themselves through concept blending (see, for example, [9]) to emerge as the performed music. We recall from [3] how free-jazz performances can be formalized as service-oriented applications that evolve by requiring other music fragments, provided by dedicated service modules, to be added to the improvisation. Focusing on the way in which many-valued institutions [10] can be used to model these creative processes by means of specifications of service units (applications or service modules), we first define \( \mathbb{RL} \)-institutions, i.e. institutions endowed with residuated lattices [8] as truth spaces [4]. Unlike the many-valued \( \mathcal{L} \)-institutions presented in [5], \( \mathbb{RL} \)-institutions do not assume a fixed residuated lattice; instead, they allow the lattices to change through appropriate morphisms. We show how \( \mathcal{L} \)-institutions generalise
to \( \mathbb{R} \)-institutions via a Grothendieck construction. We then identify a class of logics that satisfy a set of requirements that make them suitable for dealing with improvisations and specifying music phase spaces, as presented in [4]. Finally, we prove that the logic \( BGN \) defined in [3] based on one of Anthony Braxton’s graphic notations for improvised music satisfies these conditions.

The last part of our study is dedicated to the investigation of a number of questions that are relevant both to free-jazz performances and service-oriented applications, and that can be answered through our formalization: for example, assessing the cohesiveness of a performance or determining the reliability of the system – to what extent can the user’s expectations be met.

References

Graph Models for Capital Markets

Nneka Chinelo Ene¹, Maribel Fernández¹, and Bruno Pinaud²

¹ King’s College London, UK
Nneka.Ene@kcl.ac.uk Maribel.Fernandez@kcl.ac.uk
² University of Bordeaux, France
Bruno.Pinaud@u-bordeaux.fr

Abstract. The sub-prime mortgage crisis of 2008 has heightened the need for more effective and transparent tools in the modelling of uncertainty of returns from expected cash flow streams within the capital markets. One of the key development needs is that of effective assurance methods. Graph Transformation Systems (GTS) are natural verification and validation tools given that dynamic graphs are able to provide mechanisms for explaining complex situations intuitively, and rule based systems are able to mathematically capture the dynamics of otherwise relatively difficult problems. In this paper we present a preliminary secondary market GTS model for validation purposes. We use port graphs to represent system states, and port graph rewriting rules to specify the dynamics of the system. The model is implemented within PORGY, a visual, interactive port-graph rewriting tool.

Introduction

The sub-prime mortgage crisis of 2008 has heightened the need for more effective and transparent tools in the modelling of uncertainty of returns from expected cash flow streams within the capital markets. Improper evaluation of new mortgage derivative products is believed to be the key driver behind the bubble. Given the failure of traditional top-down macro equilibrium models to predict the crisis, economists are gradually turning away from them in favour of more autonomous agent-based models (ABM), that, as a result of examining behaviour at a micro-level, are able to provide more realistic views [2].

‘Rational negligence’ has been noted as the behavioural pattern among agents in the Asset-Backed Securitization (ABS) space that has led to weakening in the market. Instead of performing a proper risk assessment, agents relied on ratings from agencies that were sometimes found to be inaccurate, especially in terms of underestimated default probabilities [1], details the DSGE (Dynamic Stochastic General Equilibrium) models were unable to capture or anticipate.

In this paper, we seek to formalise theories that have developed in the study of this area as a series of graph transformations rules, governed by suitable strategies. The resulting derivation tree describes the operational execution of the graph transformation system. It represents the whole process, right from origination to placement of securitized assets within a secondary market. By employing a graph transformational paradigm, inbuilt mechanisms that favour
accurate verification and validation, or assurance, can be more easily utilized. In addition, graph transformation systems provide a visual trace of the evolution of a dynamic system.

Contributions
We have built a hierarchical model of secondary capital markets, where the ‘rational negligence’ model sits at the top level of the hierarchy. Non-deterministic in nature, it highlights the strategy the market has adopted with regards to independent analysis. The model can handle different approaches to due diligence, operational costs, asset value and liquidity perceptions, among other variables. Below this system, able also to be executed to handle asset pricing and valuation issues, lie several more deterministic subsystems that model origination, structuring, SPV transfers and profitability of the sale, and therefore aid in enforcing internal checks as a result of complete system integration. Calculations are executed at rule level, such as in deciding whether or not propagation of an independent risk assessment of the asset separated from the influence of sentiment and market perception should occur. In addition to the graph transformation rules, the model includes strategies that not only control execution but also perform analytics.

All functionality is housed within the PORGY system, a graph transformation tool based on port graph rewriting, which provides a visual, interactive interface and an expressive strategy language [3]. Strategic port graph rewriting is a powerful mechanism to model dynamic behaviour. It has been used to model social networks, biochemical systems, interaction nets, etc.

Conclusion
We have used strategic port graph rewriting as a tool to model secondary capital markets. The derivation tree, which gives a visual account of the possible behaviours of the system, can be used as a debugging tool, or to experiment with different strategies and fine tune the rules. In future, we plan to develop predicate abstractions and state-based abstractions, as a technique to verify properties of the system.

References
Providing a Semantics and Modularisation Constructs for Event-B using Institutions

Marie Farrell*, Rosemary Monahan, and James F. Power
Maynooth University, Maynooth, Co. Kildare, Ireland
mfarrell@cs.nuim.ie

Event-B is an industrial-strength language for system-level modelling and verification combining event-based logic with basic set theory. In Event-B, machines model the dynamic parts of a system (variables, invariants and events) with contexts modelling the static parts (carrier sets and axioms). An event is composed of a guard (predicate) and an action which is represented as a before-after predicate relating the new values of the variables to the old. Events are triggered once their guard(s) evaluate to true [2]. An illustrative example of an Event-B machine describing a simple traffic lights system is given in Fig 1.

Formal refinement is central to Event-B, allowing a developer to write an abstract system specification and gradually add complexity. The Rodin Platform, an eclipse-based IDE for Event-B, ensures the safety of system specifications and refinement steps by generating appropriate proof-obligations, and then discharging these via support from various theorem provers [2]. Event-B has been used extensively in industrial projects, such as the Paris Métro Line 14.

Event-B is a mature formalism yet it has no well-defined semantics. This inhibits further development of the language particularly from the perspectives of adding modularisation constructs and facilitating interoperability with other formalisms. Current approaches to modularisation in Event-B include the decomposition [6] and modularity [1] plugins for Rodin. Both provide some degree of modularisation for Event-B but they do not directly enhance the formalism itself, nor is it clear how they will interact with other variants of Event-B. Interoperability is currently achieved via a range of Rodin plugins to translate to/from Event-B. Building plugins is the accepted approach to enhancing Event-B but these plugins often lack a formal mathematical foundation. Event-B is a formal language so any enhancements to it should be made in a formally defined way.

Our approach uses the theory of institutions to provide a framework defining a semantics, a rich set of modularisation constructs and promote interoperability for Event-B. $\mathcal{EVT}$, our institution for Event-B, is based on splitting an Event-B specification into two parts. First, a data part, which is defined using $\mathcal{FO\!P\!E\!Q}$, the standard institution for first order predicate logic with equality [5]. Secondly, an event part, defining a set of events in terms of formulae constraining their before- and after- states. $\mathcal{EVT}$ is motivated by the institution for UML state machines [3]. Since refinement is a key feature of Event-B, it is vital that any semantics of Event-B provide constructs for dealing with this. The theory of institutions is equipped with a richer notion of refinement than that found in Event-B [5], and this can now be used in Event-B specifications. By defining $\mathcal{EVT}$ and carrying out the associated proofs\(^1\), not only do we create a formal seman-

\* This work is funded by the Irish Research Council.

\(^1\) For an extended version and the proofs go to http://www.cs.nuim.ie/~mfarrell/
MACHINE mac1

VAR l a b o r a t r y : Bool

INVARIANTS
inv1: l a b o r a t r y ∈ BOOL

EVENTS

1. Initialisation
2. EVT
3. EVT
4. EVT
5. EVT
6. EVT
7. EVT
8. EVT
9. EVT
10. EVT
11. EVT
12. EVT
13. EVT
14. EVT
15. EVT
16. EVT
17. EVT
18. EVT
19. EVT
20. EVT
21. EVT

Fig. 1: Event-B machine specification for a traffic system, with each light controlled by boolean flags.

Fig. 2: A modular institution-based presentation corresponding to the abstract machine mac1 in Fig 1.

References

On the Most Suitable Axiomatization of Signed Integers Using Free Constructors
Extended Abstract

Hubert Garavel\textsuperscript{1,2}

\textsuperscript{1} Inria, Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
\textsuperscript{2} Saarland University, Saarbrücken, Germany

Problem statement

This lecture discusses how signed integers can be described using algebraic specifications. More precisely, one distinguishes, on the one hand, between (unsigned) natural numbers and (signed) integer numbers and, on the other hand, between arbitrary large numbers (i.e., the mathematical sets $\mathbb{N}$ and $\mathbb{Z}$), bounded numbers (i.e., finite subranges of $\mathbb{N}$ and $\mathbb{Z}$), and machine numbers (which obey modular arithmetic, as implemented in processor instruction sets). Our problem is the algebraic specification of $\mathbb{Z}$, i.e., of arbitrary large signed numbers.

The motivation for this work arised in 1996 when applying the LOTOS language (International Standard ISO/IEC 8807:1989) to industrial case studies, noticing that the predefined library of LOTOS, based on ACT-ONE abstract data types, provides only naturals (defined in the Peano style by means of two constructors $0 : \mathbb{N}$ and $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}$), but not integers. At that time, no obvious solution could be found in the litterature. During the past decades, various approaches have been devised for this problem and implemented in mainstream computer languages and provers. We review and compare these approaches with respect to their mathematical simplicity and computational efficiency.

Approaches based on set-theoretic concepts

One way to define integers relies on set inclusion: for instance, the predefined library of PVS [8, Chap. 7] specifies numeric types as (transitive) subtypes derived from a most general type named number; in this lattice, integers are defined as a subtype of rational numbers.

Another way, used in mathematical textbooks and in the definition of CASL [7, Sect. V:2, page 381], is to specify the set of integers as $\mathbb{Z} = (\mathbb{N} \times \mathbb{N})/\sim$, where $\sim$ is the equivalence relation defined by $(x, y) \sim (x', y') \iff x + y' = x' + y$. This approach based on cartesian product and set quotient has the merit of exhibiting a nice symmetry with rational numbers, whose set can be similarly defined as $\mathbb{Q} = (\mathbb{Z} \times (\mathbb{Z} \setminus \{0\}))/\sim$, where $\sim$ is another equivalence relation defined by $(x, y) \sim (x', y') \iff x.y' = x'.y$. However, this approach goes against the intuition as it builds a two-dimentional surface where a half line towards negative integers would be sufficient. Moreover, it is not computationally optimal, neither
in memory (the cartesian product suggests that it takes two naturals to build one integer) nor in CPU time (the quotient operation requires two additions and an equality test to compare two integers).

**Approaches based on algebraic specifications**

Rather than relying on concepts from Cantor’s set theory, it is worth examining how integers can be defined in Peano style (namely, “basic” algebraic specifications, without “higher-level” extensions), in the same way naturals are defined using the 0 and \textit{succ} constructors.

There exist intuitive approaches, such as completing 0 and \textit{succ} with a third operation \textit{pred} : \( x \mapsto x - 1 \), or defining \( \mathbb{Z} \) as \( \{+, -\} \times \mathbb{N} \). Unfortunately, these simple approaches result in non-free constructors, as \( \text{pred}(\text{succ}(x)) = x \) and \( (-,0) = (+,0) \), which is deemed unsuitable for computer-aided verification.

For this reason, mainstream tools rely on free constructors, but follow diverse approaches, which can be compared according to their complexity, measured in the number of sorts and constructors. For instance, mCRL2 [5] uses three sorts (\textit{Nat}, \textit{Pos}, and \textit{Int}) and two constructors. The Coq standard library [http://coq.inria.fr/library] uses two sorts (\textit{Positive} and \( \mathbb{Z} \)) but the \textit{Nat} sort also exists aside) and three constructors. Maude [2] and SMT-LIB [1], page 35 and Fig. 3.3 define only two sorts (\textit{Nat} and \textit{Int}) but respectively rely on subtyping and syntactic restrictions to obtain strictly positive naturals. The CADP toolbox [4] proposes a solution with only two sorts (\textit{Nat} and \textit{Int}) and two constructors. Finally, an approach with two sorts and a single constructor is discussed.

**References**

Behavioural Semantics for the Dynamic Logic with Binders $\mathcal{D}_\downarrow$

Rolf Hennicker$^1$ and Alexandre Madeira$^2$

$^1$ Ludwig-Maximilians-Universität München, Germany
$^2$ HASLab INESC TEC & Univ. Minho, Portugal

The quest for logics and formal methods for the rigorous development of reactive software, i.e., systems which interact with their environment along the whole computation, and not only in its starting and termination points [1], is an active topic of research in Computer Science. Typical approaches start from the construction of a concrete model (e.g. in the form of a transition system [8], or a process algebra expression [6, 2]) upon which the relevant properties are later formulated in a suitable (modal) logic and typically verified by some form of model-checking. Resorting to old software engineering jargon, most of these approaches proceed by inventing & verifying.

We intend, however, to contribute to the field following an alternative correct by construction perspective; loose specification has an important role to play, because they support the gradual addition of requirements and implementation decisions such that verification of the correctness of a complex system can be done piecewise in smaller steps. This entails the adoption of a logic suitable to express requirements on various levels of abstraction, from property specifications, concerning e.g. safety and liveness requirements, to constructive specifications representing concrete processes. In this view, we recently introduced $\mathcal{D}_\downarrow$-logic, a Dynamic Logic with Binders [7], combining in the same formalism binders of hybrid logic [4] with regular modalities of dynamic logics [5].

A signature $A$ of $\mathcal{D}_\downarrow$ is a set of atomic actions; and a set of variables is an infinite set of symbols $X$. For simplicity, the disjointness of variables and action sets is assumed. The set of formulas of $\mathcal{D}_\downarrow$, for a signature $A$ is defined by the grammar

$$\varphi ::= \mathsf{tt} \mid \mathsf{ff} \mid x \mid \downarrow x. \varphi \mid \mathsf{[]} \varphi \mid [\alpha] \varphi \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi$$

where $\alpha ::= a \mid \alpha ; \alpha \mid \alpha + \alpha \mid \alpha ^* \ | \ a \in A$ and $x \in X$. Semantics of $\mathcal{D}_\downarrow$ reflects our focus on effective computations: rather than the usual arbitrary relational structures, models $\mathcal{M} = (W, w_0, R)$ are transition structures over reachable states $W$, w.r.t. the initial state $w_0$ and accessibility relations $R = (R_a \subseteq W \times W)_{a \in A}$. Given an $A$-model $\mathcal{M} = (W, w_0, R)$, a state $w \in W$, a valuation $g : X \to W$ and a formula $\varphi$, we inductively define $\mathcal{M}, g, w \models \varphi$ as follows: $\mathcal{M}, g, w \models x \mid \mathsf{tt} \iff g(w) = w$; $\mathcal{M}, g, w \models \downarrow x. \varphi$ if $\mathcal{M}, g, g(x) \models \varphi$; $\mathcal{M}, g, w \models [\alpha] \varphi$ if there is a $w' \in W$ with $(w, w') \in R_\alpha$ and $\mathcal{M}, g, w' \models \varphi$; finally, the binder operator $\downarrow x. \varphi$ that evaluates $\varphi$ and assigns to variable $x$ the current state of evaluation:

$^*$ This work is financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 and by National Funds through the Portuguese funding agency FCT within project POCI-01-0145-FEDER-016692. A. Madeira is also supported by the FCT BPD grant SFRH/BPD/103004/2014.
\( M, g, w \models \downarrow \varphi \) if \( M, g[x \mapsto w], w \models \varphi \). The remaining cases are defined as expected. Hence, we have \( M \models \varphi \) if for any \( g : X \to W \), \( M, g, w_0 \models \varphi \). Note however that, as happens in the classic algebraic specification, this satisfaction notion can be too strict to be useful in the development practices.

For instance, it is easy to see that the left structure of Fig. 1 is a model of (flat) specification \( SP = \langle \{a\}, \{\downarrow x. (a)x\} \rangle \) and that the one on the right side it is not. But, in principle, since they are bisimilar, it should be irrelevant for implementation purposes to choose one or the other to implement \( SP \). From the modal logic perspective, this also shows that \( D_{\downarrow} \) has no modal invariance, a property verified in almost all modal logics.

Fig. 1. Two \( D_{\downarrow} \)-models

The goal of this work is to find a more relaxed interpretation of the satisfaction, on which, the model class of specifications are closed w.r.t. the bisimulation equivalence \( \equiv \). As in the classic algebraic specification (e.g. [3]), we can follow two approaches to overcome this situation: we can define abstractor semantics of a specification \( SP = \langle A, \Phi \rangle \) using an abstractor operator \( \text{Abs}_{\equiv}(SP) = \{ M | M \equiv N, N \in \text{Mod}(SP) \} \). Or, we can consider a new, behavioural satisfaction relation \( \models_{\sim} \), in order to take the semantics of the specification as \( \text{Mod}_{\sim}(SP) = \{ M \in \text{Mod}^{D_{\downarrow}}(A) | M \models_{\sim} \varphi, \varphi \in \Phi \} \) where, for each model \( M, \sim \) is the bisimilarity relation on the states of \( M \). The relation \( \models_{\sim} \) is exactly defined as \( \models \), except in the case of formulas \( M, g, w \models_{\sim} x \), where the condition \( g(x) \sim w \) is taken in the place of \( g(x) = w \).

We explore then, for this logic, some analogous results to the ones achieved on behavioural equivalence(s) in algebraic specifications. For instance, as expected, we have \( M \models_{\sim} \varphi \) if \( M/\sim \models \varphi \) where \( M/\sim \) is the quotient of \( M \) w.r.t. the bisimilarity \( \sim \) on (the states of) \( M \). We introduce a category of models and behavioural morphisms such that behavioural isomorphisms coincide with behavioural equivalences. On this setting, properties as “if \( M \equiv M' \) then for any \( \varphi, M \models_{\sim} \varphi \) iff \( M' \models_{\sim} \varphi \)” are studied. Finally, we obtain that \( \text{Mod}_{\sim}(SP) = \text{Abs}_{\equiv}(SP) \) iff \( \text{Mod}_{\sim}(SP) \) is closed under quotients.

References

7. A. Madeira, L. Barbosa, R. Hennicker, and M. Martins. Dynamic logic with binders and its application to the development of reactive systems. Accepted for ICTAC’16. LNCS. (in print).
Towards Critical Pair Analysis for the
Graph Programming Language GP 2

Ivaylo Hristakiev and Detlef Plump
University of York, United Kingdom

A common programming pattern in the graph programming language GP 2 [9] is to apply a set of attributed graph transformation rules as long as possible. To execute a set of rules \( \{r_1, r_2, \ldots, r_n\} \) as long as possible on a host graph, in each iteration an applicable rule is selected and applied. As rule selection and rule matching are non-deterministic, different graphs may result from the loop. Thus, if the programmer wants the loop to implement a function, a static analysis that establishes or refutes functional behaviour would be desirable.

GP 2 is based on the double-pushout approach to graph transformation with relabelling [4]. It facilitates computation over labels using rules labelled with expressions and host graphs labelled with concrete values. GP’s fixed algebra for labels consists of integers, character strings, and heterogeneous lists of strings and integers. Rule application can be seen as a two-stage process where rules are first instantiated, by replacing expressions with values, and then applied as usual. Hence rules are actually rule schemata.

Conventional confluence analysis in the double-pushout approach to graph transformation is based on the study of critical pairs which are conflicts in minimal context [8,2]. A conflict between two rule applications arises, roughly speaking, when one of the steps cannot be applied to the result of the other. In the presence of termination, one can check if all critical pairs are strongly joinable, and thus establish functional behaviour of the set of transformation rules.

However, the conventional notion of critical pairs is not directly applicable to GP 2 rule schemata. To construct such pairs, one needs to instantiate rule schemata to an (usually) infinite set of concrete rules [6], and thus the analysis cannot be automated as part of a confluence checker. Furthermore, when computing the labels of critical pairs, it has been observed [1] that syntactic unification is not sufficient in that the constructed set of critical pairs does not represent all conflicts. Instead, one has to take into account all equations valid in the attribute algebra. A severe restriction that avoids the need for unification altogether is to only allow rules over variables or variable-free expressions as done in [3].

In this work, we propose a solution to both problems by defining abstract critical pairs that are labelled with expressions, giving an algorithm for their construction, proving they are complete and that there are only finitely many of them. We avoid the need for the severe restriction that rules are labelled only with variables or variable-free expressions.

* Supported by a Doctoral Training Grant from the Engineering and Physical Sciences Research Council (EPSRC) in the UK.
As a simple example, let us consider a GP 2 program that recognizes series-parallel graphs by means of confluent graph reduction [10]. The essence of the program are two rule schemata, one of which is shown below.

\[
\text{series}(a, b, x, y, z: \text{list})
\]

\[
\begin{array}{c}
\text{a} & \text{b} \\
\text{x} & \text{y} & \text{z} \\
1 & 2 \rightarrow 1 & 2
\end{array}
\]

The schema is matched by finding an instance of the left-hand graph in a host graph. The schema replaces the middle node together with its incident edges by a single edge and relabels the remaining nodes with the empty list. The variables \(a, b, x, y, z\) are instantiated with variable-free lists during matching. An example abstract critical pair is shown below where the middle graph is obtained by overlaying the left-hand graph of the \text{series} schema with itself, and the graphs on either side are the results of applying the schema in conflicting ways. The conflict comes from the fact that either application deletes a common node (2 or 3) and also changes the label of a common node (3 or 2).

We give an algorithm for the construction of abstract critical pairs that respects the equations of GP’s label algebra, namely the associativity and unit laws of list concatenation. These critical pairs represent each conflict in the most abstract way. First, left-hand graphs of two given schemata are overlapped to produce an overlay graph. Next, the labels of the overlay are computed using the unification algorithm for GP 2 expressions [5]. The unification algorithm takes an equation \(s = ? t\) between two expressions and returns a set of substitutions (mappings from variables to expressions) that make them equal. Each substitution together with its overlay graph induces an abstract critical pair.

Completeness of critical pairs means that each pair of conflicting rule schema applications is an instance of an abstract critical pair. Formally, we state the fact in the following theorem.

**Theorem 1 (Completeness of Abstract Critical Pairs).** For each pair of conflicting rule schema applications \(G_1 \xleftarrow{r_1} G_0 \xrightarrow{r_2} G_2\) there exists an abstract critical pair \(P_1 \xleftarrow{r_1} P_0 \xrightarrow{r_2} P_2\) such that for \(i = 0, 1, 2\), there is an injective graph morphism \(P_i' \rightarrow G_i\) where \(P_i'\) is an instance of \(P_i\).

\[
\begin{array}{c}
P_0' \xleftarrow{r_1} P_0 \xrightarrow{r_2} P_2' \xrightarrow{r_1} P_1' \end{array}
\]

The proof is broken down into two main parts. First, we define concrete critical pairs (labelled with concrete data values) and prove that each pair of conflicting schema applications is an extension of a concrete critical pair. Note that the set of concrete critical pairs is usually infinite because of GP 2’s infinite label set. Next, we prove that each concrete critical pair is an instance of an abstract critical pair constructed by our algorithm.
Finally, we show that the set of abstract critical pairs is finite. This relies on the fact that the unification algorithm for GP 2 expressions terminates with a finite complete set of unifiers (meaning that each unifier of a problem $s \rightarrow t$ is an instance of a unifier from the finite set).

**Theorem 2 (Finiteness of Abstract Critical Pairs).** For each pair of left-linear rule schemata $r_1$ and $r_2$, the set of abstract critical pairs induced by $r_1$ and $r_2$ is finite.

The condition that schemata are left-linear means that list variables are not shared between labels. This ensures that the unification problem for GP 2 expressions has a finite complete set of unifiers.

We are currently working on proving the Local Confluence Theorem for GP 2, which establishes local confluence of sets of rule schemata for the case that all abstract critical pairs are strongly joinable. Another topic of future work are the critical pairs of [7] which are based on a relaxed notion of conflict.

**References**

A Calculus of Virtually Timed Ambients

Einar Broch Johnsen, Martin Steffen, and Johanna B. Stumpf

University of Oslo, Oslo, Norway
{einarj, msteffen, johanbst}@ifi.uio.no

Motivation. The ambient calculus is the process algebra of locations and domains, originally developed by Cardelli and Gordon [2] for distributed systems such as the Internet. In this paper, we extend the ambient calculus with a notion of virtual time as a resource. The resulting calculus can be used for instance to model aspects of virtualization in cloud computing, where different locations, barriers between locations, and barrier crossing are important features, as well as elasticity which allows to provision virtual resources on-demand.

Previous work on timed process algebras. Algebraic concurrency theories such as ACP, CCS and CSP have been extended to deal with time-dependent behaviour in various ways (e.g., [1, 5, 3]). All these approaches describe speed as the absolute duration of processes, while in our approach speed describes the relative processing power of an ambient.

Preliminaries on mobile ambients. An ambient represents the location or domain where a process is running. Ambients can be nested, such that a surrounding parental ambient contains subambients, and the nesting structure can change dynamically. This is specified by three basic capabilities. The input capability in n indicates the willingness of a process, respectively its containing ambient, to enter an ambient named n, running in parallel outside, e.g., k[in n.P] | n[Q] \rightarrow n[k[P] | Q]. The output capability out n enables an ambient to leave its surrounding ambient n, e.g., n[k[out n.P] | Q] \rightarrow k[P] | n[Q]. The third basic capability open n allows to open an ambient named n which is on the same level as the capability, e.g., k[open n.P | n[Q]] \rightarrow k[P | Q]. This syntax, as well as the semantics we consider, is based on [4] and largely unchanged compared to [2].

Virtually timed mobile ambients. We extend mobile ambients with notions of virtual time and resource consumption. Virtual time is a resource, which is made available to a location by its parental location, similar to time slices that an operating system provisions to its processes. Interpreting the locations of ambients as a place of deployment, each timed ambient is modelled to have a certain computing power, determined by its deployment. Thus, our model of timed ambients uses a local notion of time, which, however, is relative to the computing power of the embedding, parental ambients.

Timed systems. A timed ambient contains one local clock and possibly other timed ambients or classic untimed ambients and processes. A computing environment is a timed ambient which contains resources, as explained below.
Local clocks. To represent the outlined time model, each timed ambient is equipped with one local clock responsible for triggering timed behaviour and local resource consumption. Clocks have a speed, interpreted relative to the speed of the surrounding timed ambient. The speed $s$ of a clock is given by the rational number $p/q$, where $p$ is the number of local time slices emitted for a number $q$ of time slices received from the surrounding ambient. Time slices propagate from parental clocks to clocks in the subambients. Thus, the time in a nested ambient is relative to the global time, depending on the speeds of the clocks of the ambients it is nested in. We assume one universal outermost ambient with a global clock triggering the clocks of the local subambients recursively. When moving timed ambients, we must update the clocks to ensure a correct propagation of time slices, thus we use an update function and define timed capabilities in $n$, out $n$, and open $n$ for timed systems, corresponding to the similar untimed capabilities.

Computing resources. An ambient’s processing power is defined by a resource process which transforms the time slices of the local clock into locally consumable resources. Processes expend the processing power of the ambient they are contained in by consuming resources. An ambient with a higher local clock speed produces more resources per parental time slice which in turn allows more work to be done for each parental time slice.

Weak bisimulation for timed ambients. We define weak timed bisimulation for virtually timed ambients as a conservative extension of weak bisimulation for mobile ambients as defined by Merro and Zappa Nardelli [4]. We then define a bisimulation for specific classes of processes which relaxes the condition on timing. This way we can determine if a system is faster than another and give a worst case approximation for this timing difference. We finally show that weak timed bisimulation for ambients completely characterises reduction barbed congruence for virtually timed ambients, extending a result from [4].

References

On the Structural Link between Ontologies
and Their Organised Data Sets

Ridha Khedri and Alicia Marinache

Department of Computing and Software
Faculty of Engineering, McMaster University
Hamilton, Ontario Canada
{khedri, marinaam}@mcmaster.ca

Ontologies are commonly used to provide explicit specifications of a conceptualisation of a domain. They have been used for the representation of knowledge in many areas (e.g., [1–4]). Most of the existing formalisms define an ontology strictly as a hierarchical structure. The proposed work aims at capturing the structure of an ontology and relate it to a model for organised data sets. The ontology structure captures (Cartesian) mereological relationships on concepts, as well as other relationships relevant to the considered domain of application. The component related to organised datasets is modelled using diagonal-free cylindric algebra [5].

The domain knowledge system that we propose, which we call domain information system, can be separated into three distinct components. The first component is the structure of concepts and their relationships. We refer to it as abstract ontology. The second component serves for modelling the data stored in the information system. The first two components are used to form a domain information structure linking data to the concepts in the abstract ontology. The third component consists of the algebraic specifications that give more links between the two previous components of the domain information system. More details can also be found in [6].

Component 1 –Abstract Ontologies–. Ontologies capture the set of concepts in a domain, their attributes, and the relationships between the concepts. In our work, we focus only on the general relationships between concepts, such as the relations isA, memberOf, hasA, and partOf. We call such an ontology the abstract ontology, in the sense that it describes the most general, abstract view of an application domain.

On a set $C$ of concepts of a domain, we take $C = (C, \times, e_C)$ as a commutative idempotent monoid. Intuitively, the $e_C$ concept is a pseudo-concept that is neutral to the composition of concepts. For every $c_1, c_2 \in C$, we say that $c_1$ is a part of $c_2$, that we denote by $c_1 \subseteq c_2$ if $\exists c | c \in C \land c_1 \times c = c_2$. Then we consider a subset $L$ of $C$ on which the partOf forms a Boolean lattice of concepts $\mathcal{L} = (L, \subseteq_C)$. From a practical point of view, the set $L$ contains all the concepts that the considered data set directly represents. We also include in $L$ the concept $e_C$, which is the bottom of the lattice and represents the null concept. The top of the lattice is the main domain concept $\top_C$. The concepts in the lattice $L$ may be related to other concepts in $C$ through other domain specific relationships, which are represented as rooted graphs, where the root is always part of the
lattice. A rooted graph at $t_i$ is denoted by $G_{t_i} = (C_i, R_i, t_i)$, with $C_i \subseteq C$ and $R_i \subseteq C_i \times C_i$. It satisfies $t_i \in L \land (\forall c \mid c \in C_i \cdot c = t_i \lor (c, t_i) \in R_i^+)$. Let $\mathcal{G} = \{G_{t_i}\}_{t_i \in L}$ be a (possibly empty) family of rooted graphs. Then, an abstract ontology is the structure $\mathcal{O} = (C, L, \mathcal{G})$, where $\mathcal{C} = (C, \times, e_C)$ is a commutative idempotent monoid, $L = (L, \sqsubseteq)$ is a Boolean lattice, with $e_C \in L \subseteq C$, and $G = \{G_{t_i}\}_{t_i \in L}$ is a family of rooted graphs.

**Component 2 –Organised Data set**. In constructing the structure for the data, we need to relate it to the concepts of the lattice of the abstract ontology. For this purpose, we use diagonal-free cylindric algebra as model for organised data sets. It has been demonstrated that the relational model of data, as proposed by E.F. Codd in 1970, is a special case of the cylindric algebra model. We use the elements of the lattice $L$ to index the cylindrification operators, which provides a link between the lattice of concepts and the model for the data. We give the axioms that enable us to link the abstract ontology structure to data sets and we discuss the rationale and the intuition behind these axioms. We then present some additional constructions based on the above that enable us to extract more information from a data set and its ontology.

**Component 3 –Domain Non-Structural Specification**. The theories of the above structures provide the axioms pertinent to the structural parts of an information system. However, there is other domain-specific knowledge that is extremely useful in reasoning on data. This component is called the domain non-structural specification and it gives the definition of new concepts based on the concepts introduced by the abstract ontology (i.e., Component 1). We give results showing the need for the knowledge captured by this component of a domain information system.

Our work aims at expanding our current understanding of ontologies to give them better structure in order to support reasoning on structured data sets. It uses simple algebraic structures to construct a support system for reasoning on data. Our results encompass the main ideas in the area and reconcile some of them. We conjecture that it will enable us to scale reasoning on data sets of unprecedented large sizes and to generate better quality information from them. We show some evidence that supports the above conjecture.

**References**

About partiality in institutions (co-)morphisms

Carlos G. Lopez Pombo\textsuperscript{1,2}, Agustín E. Martínez Suñé\textsuperscript{4}, Fabio Gadducci\textsuperscript{3}, Thomas S.E. Maibaum\textsuperscript{4}
clpombo@dc.uba.ar, agusmartinez@dc.uba.ar, gadducci@di.unipi.it, and tom@maibaum.org

\textsuperscript{1} Departamento de Computación, FCEyN, Universidad de Buenos Aires, Argentina
\textsuperscript{2} Consejo Nacional de Investigaciones Científicas y Tecnológicas, Argentina
\textsuperscript{3} Dipartimento di Informatica, Università di Pisa, Italia
\textsuperscript{4} McMaster Centre for Software Certification, McMaster University, Canada

The nature of modern software is intrinsically heterogeneous. Such heterogeneity is derived from the diversity of properties and characteristics that software systems have, which is in turn witnessed by the diversity of specification formalisms and languages that are used to describe them. Institutions\cite{1} was introduced as a formalisation of the abstract model-theory of logical formalisms and has been proved adequate for taming such variety. The possibility to relate institutions via suitable mappings\cite{2,3}, which can e.g. interpreted as semantics-preserving translations, allowed to consider diagrams of institutions as heterogeneous logical environments\cite{4} and to use them as formal foundations for tools like HETS\cite{5}.

Following the original definitions of such connections between logical languages, those mappings relating signatures, sentences and models are required to be total. However, it is well-known that many logical formalisms relate in a partial way. Among others, the relation between CTL\cite{6} and LTL\cite{7} is a classical example of logics that share an equipollent proper fragment, while there is no total semantics-preserving mapping in either direction.

In this work we explore the notion of partial co-morphism (even if our considerations could be applied to morphisms as well) as a mean for providing this nuanced kind of relation between institutions. In analogy with partial functions from set-theory, a partial co-morphism \( \rho \) from institutions \( I \) and \( I' \) could be defined as a co-morphism \( \rho : \bar{I} \to I' \), where \( \bar{I} \) is a sub-institution of \( I \). More abstractly, in analogy with partial maps in category theory\cite{8}, we define partial co-morphisms as spans of co-morphisms \( I \xleftarrow{\sigma} \bar{I} \xrightarrow{\rho'} I' \), where \( \sigma : \bar{I} \hookrightarrow I \) is a monic in the associated category of institutions. In other terms, given the co-morphism \( \sigma = (F, \eta, \xi) \), the underlying functor \( F \) is faithful, and the natural transformations \( \eta \) and \( \xi \) have injective and surjective components, respectively.

The activity of software design and analysis assumes that the language in which the elements of the artifact are specified is fixed from the beginning of the process. Thus, any modification of the underlying language is a task that is considered offline and not part of the process. Our definition of partial co-morphism may enable the possibility of restructuring the underlying heterogeneous logical language via the use of the double pushout (DPO) construction. However, the

* The research of Carlos G. Lopez Pombo has been supported by the grants UBACyT 20020130200092BA, PICT 2013-2129, and PIP 11220130100148CO.
standard approach to DPO rewriting via adhesive categories [9] seems to be lacking here, since in the chosen category of institutions pushouts are not preserved along pullbacks (thus failing adhesiveness, according to [10, Theorem A]).

Heterogeneous tools mostly rely on the use of Grothendieck institutions [11] as a formal description of the heterogeneous language. Thus, our work requires to explore an alternative definition of flattening that is coping with partial mappings. Despite its flexibility, partial co-morphisms do not exhaust all the varieties of relations between logics. This phenomenon can be e.g. observed by recalling that the formula $\bigwedge_{i \in I} \alpha_i$ of $\mathcal{L}_{\omega \cdot \omega}$ can be translated to the theory presentation $\{\alpha_i\}_{i \in I}$ in $\text{FOL}_{\omega}$. Thus, we discuss how the advances hinted above concerning heterogeneity and adhesiveness can be recast when considering a generalisation of co-morphisms $\rho : I \rightarrow I'$ differing in the way in which sentences are mapped across different logical languages, that is, not resorting to a natural family of functions $\rho^\text{Sen}_\Sigma : \text{Sen}(\Sigma) \rightarrow \rho^\text{Sig} \circ \text{Sen}'(\Sigma)$, in which each $\Sigma$-sentence in $I$ is translated to a single $\rho^\text{Sig}(\Sigma)$-sentence in $I'$, but accepting that sets of sentences are translated to sets of sentences in a semantics-preserving way.

References
Selecting Colimits for Parameterisation and Networks of Specifications

Till Mossakowski\textsuperscript{1}, Mihai Codescu\textsuperscript{2}, and Florian Rabe\textsuperscript{3}

\textsuperscript{1} Otto-von-Guericke-University of Magdeburg, Germany
\textsuperscript{2} Free University of Bozen-Bolzano, Italy
\textsuperscript{3} Jacobs University Bremen, Germany

Abstract. Colimits are a powerful tool for the combination of objects in a category. In the context of modeling and specification, this is used, e.g., to define the semantics of instantiations of parameterised specifications or of combinations of specifications. Because colimits are only unique up to isomorphism, specification languages must usually select one element of the isomorphism class as the colimit. In particular, this selection is necessary so that the user can refer to the symbols in the colimit after constructing it. However, there is no obvious canonical way to make this selection. We develop and discuss criteria for evaluating colimit selections in specification languages. Then we provide selections for typical languages that enjoy desirable properties.

The notion of colimit provides a natural way to abstract the idea that some objects of interest, which can be e.g. logical theories, software specifications or semiotic systems, are combined while taking into account the way they are related. Specification languages whose semantics involves colimits are CASL (for instantiations of parameterised specifications) and its extension DOL (for combination of networks of specifications). Specware\cite{11} provides a tool computing colimits of specifications that has been successfully used in industrial applications;\cite{10} makes a strong case for the use of colimits in formal software development. The Heterogeneous Tool Set (HETS,\cite{7}) also supports computation of colimits, covering even the heterogeneous case\cite{2}. Recently, colimits have provided the base mechanism for concept creation by blending existing concepts\cite{4}. Moreover, colimits provide the basis for a good behaviour of parameterization in a specification language\cite{3}.

The problem that arises naturally when using colimits is that they are only unique up to isomorphism. By contrast, the semantics of a specification involves a specific signature, which must be selected from this isomorphism class. Similarly, any implementation of colimit computation must make such a selection. In most cases, this requires selecting a name for every symbol in the colimit. To be useful in practice, it is desirable that the selection appears natural to the user. In particular, whenever possible, original names should be reused, inclusions should be preserved, and different ways to specify the same colimit should yield identical (and not just isomorphic) results.

The semantics of CASL\cite{1,5} provides some method for computation of specific pushouts. However, the chosen institutional framework (institutions with...
a lot of extra infrastructure) is rather complicated. Moreover, desired properties of pushouts are only discussed casually. Rabe [9] discusses three desirable properties of selected pushouts and conjectures that they are not reconcilable.

We shed light on this conjecture and provide a total selection of pushouts, while [9] only provides a partial selection. The systematic investigation of selected colimits (i.e. beyond pushouts) is new to our knowledge.

In this paper, we present the fragments of the CASL [8] and DOL [6] languages that are related to colimits. We first develop criteria for colimit selections such that parameterisation and combination of networks enjoy good properties. Then we discuss in detail how to usefully select a colimit in a fairly general setting.

References

Motivation. A lie is an intentional announcement of (believed-to-be) incorrect information in order to deceive the audience [6]. A lie is an action, while (untrue) belief is a possible consequence of that action in agents’ state. The belief aspect, i.e., epistemics, of lies is often captured by modeling the effect of lies as belief revisions or updates. Modeling lies and their epistemics have attracted substantial attention in the recent literature concerning Dynamic Epistemic Logic (DEL, cf. [5, 6, 9, 12] for some recent examples; also see [11] for applications of DEL in security).

Communication protocols are often described in an operational and narrative style, describing the possible sequence of announcements (message exchanges) among the involved principals. Yet correctness properties of such protocols are often naturally expressed in terms of epistemic properties (e.g., whether a certain fact remains secret after running the protocol). Hence, it is beneficial to check epistemic properties on operational models of protocol specification [8], particularly in the presence of untruthful principals.

In this paper, we present an operational model in which lies can be told, i.e., a message can be communicated but announced to a certain audience as if something else (or nothing at all) has been communicated. In our model, we abstract away from the intention of the liar and focus on what credulous rational agents (involved in the communication or external observers) can infer about a particular execution of the protocol, if they know the protocol beforehand. (Credulous agents are those agents that are willing to accept what is being told to them as long as it does not lead to any epistemic inconsistency with the rest of their belief.) We focus on two types of logical properties: first, what an agent consistently believes and second, whether an agent can detect a particular lie in the course of a protocol execution. We express these properties in a rich extension of modal $\mu$-calculus with Dynamic Epistemic Logic constructs and define the semantics of our operational models in the semantic domain of our logic.
This paper integrates process algebra [2] as an operational framework and Temporal Dynamic Epistemic Logic [1] as a logical framework. It builds upon our earlier proposals [3, 4], in which only knowledge (thus, no lies and untrue beliefs) were incorporated.

**Related work.** The closest framework to ours are those based on Dynamic Epistemic Logic such as [5, 6, 9]. In the remainder, we focus on [5] as the point of reference for our comparison, because it is the most recent and the most developed example of such frameworks. In a nutshell, compared to [5], we put an emphasis on ease of modeling and provide a concrete and generic syntax for specifying various types of announcements. Our framework is much simpler than the action models of agent announcements in [5] in that we only specify the operational behavior of the protocol and derive the epistemic models automatically. This simplicity comes at a cost, namely, reduced expressiveness.

The semantics of our process algebra bears close resemblances to that of [5]. In [5, Section 3], a distinction has been made between the belief update for the speaker and the belief update for the addressee. In our framework, we have a set of intended audience (corresponding to the speaker in [5]) and the rest (corresponding to the addressee in [5]). In [4], we gave a translation from a similar process algebraic framework (excluding the aspect of lying) to the interpreted systems model [7, 10]. We expect a similar translation to be possible in our extended settings with lies.

**References**


The monad of continuous evolutions

Renato Neves
HASLab INESC TEC & Universidade do Minho
nevrenato@gmail.com

Even if pervasive in our daily lives, hybrid systems are still notoriously difficult to formally specify and analyse. This is due to the complexity that arises from the combination of discrete, and continuous behaviour, as well as from the heterogeneity in the devices that comprise the systems of today.

The latter issue is traditionally addressed within the component-based paradigm [6], in which a (complex) system is built from simpler components, the developer dictating how they should interact and coordinate between themselves.

A prime example of a formalisation of this paradigm is the work of Barbosa [1], which favours a view of components as black-boxes through the adoption of a coalgebraic perspective. For the sake of genericity, the calculus is parametrised by a strong monad [3], the latter being used to provide the desired behavioural model to the component calculus. For example, the powerset monad establishes a calculus of nondeterministic components, while the distribution monad gives rise to a probabilistic counterpart.

On the other hand, the complexity of combining continuous, and discrete behaviour has been addressed in a number of papers (e.g. [2, 5]), but much less frequently from a component-based perspective.

In this talk we introduce a (strong) monad [4] that encodes the typical continuous behaviour of hybrid systems, i.e. their influence over the (continuous) evolution of a physical process, like velocity, pressure, energy, or time. Intuitively, the monad allows for the control of an evolution to be passed along different systems, and provides a corresponding algebra as well — technically, this explores the structure of the associated Kleisli category.

Then, resorting to the coalgebraic calculus described in [1], we will show that an interesting framework for specifying and reasoning about hybrid components emerges. In brief, it includes several composition operators (e.g. parallelism, pipelining, feedback), wiring mechanisms, and synchronisation techniques. To illustrate the potentialities (and limitations) of the framework, we will analyse classic examples of hybrid systems under its light.

This monad parallels the role that (+1), powerset, and distribution monads have in the coalgebraic frameworks of faulty, nondeterministic, and probabilistic components, respectively.

Acknowledgements. This work is funded by ERDF - European Regional Development Fund, through the COMPETE Programme, and by National Funds through Fundação para a Ciência e Tecnologia (FCT) within project POCI-01-0145-FEDER-016692. The author was further supported by FCT grant SFRH/BD/52234/2013.
References


(Asymmetric) combination of logics is functorial

Renato Neves¹, Alexandre Madeira¹, Luis Soares Barbosa¹, and Manuel A. Martins²

¹ HASLab INESC TEC & Universidade do Minho
nevrenato@gmail.com, lsb@di.uminho.pt, madeira@ua.pt
² CIDMA - Dep. Mathematics, Universidade de Aveiro
martins@ua.pt

One of the standard practices in the development of critical software is the employment of logics to specify and verify behavioural properties. The ever increasing software complexity, however, renders this a difficult task, often demanding of the engineer the ability to handle different logics at the same time, in order to properly formalise the requirements of the system at hands.

Actually, this is contributing towards the adoption of a new methodology in software development: in order to formalise a given system, first note its distinguishing features; choose whatever logics are suitable to specify those features and then properly combine the selected logics. The result should be a single logic, tailored to the design of the whole system.

We will focus on three examples of what is known as asymmetric combination of logics, the qualifier asymmetric standing for the fact that specific features of a logic are developed ‘on top’ of another one. Hybridisation [4] is one of these examples — a systematic process along which the characteristic features of hybrid logic, both at the syntactic (i.e. modalities, nominals, etc.) and the semantic (i.e. possible worlds) levels, are built over an arbitrary logic. To be completely general this is framed in the theory of institutions [3].

The other two examples are ‘temporalisation’ of logics, introduced by M. Finger and D. Gabbay in [2], and the more recent approach to ‘probabilisation’ of logics, introduced by P. Baltazar in [1]. The former builds a temporal layer on top of a logic, the latter a probabilistic one.

In the talk we will argue that asymmetric combination of logics are often functorial, which brings a number of interesting possibilities: namely, the ability to lift the combination processes from logics to their translations, and moreover to characterise natural transformations between such asymmetric combinations. Another object of study concerns adjoints, and preservation of properties like conservativity, equivalence, and (co)limits.

The talk will recall hybridisation, temporalisation, and probabilisation of logics and introduce their uniform characterisation as endofunctors in a suitable category of institutions. In particular, we will see how to extend the examples above from combinations of logics to combination of logic translations, and that such an extension is functorial, preserves conservativity, and institutional equivalence. Natural transformations linking these examples will be discussed as well.
Acknowledgements. This work is funded by ERDF - European Regional Development Fund, through the COMPETE Programme, and by National Funds through Fundação para a Ciência e Tecnologia (FCT) within project POCI-01-0145-FEDER-016692.

The authors Renato Neves, Alexandre Madeira, and Luis Barbosa were further supported by FCT grants SFRH/BD/52234/2013, SFRH/BPD/103004/2014, and SFRH/BSAB/113890/2015, respectively.

Finally, Manuel Martins was supported by FCT within project UID/MAT/04106/2013 and the EU FP7 Marie Curie PIRSES-GA-2012-318986 project GeTFun: Generalizing Truth-Functionality.

References

Generic Hoare Logic for Order-Enriched Effects with Exceptions

Christoph Rauch, Sergey Goncharov, and Lutz Schröder

Friedrich-Alexander-Universität Erlangen-Nürnberg

Hoare Logic Made Exceptional. The use of monads as a means of modelling side-effecting computations [3] is well-established in the design and semantics of programming languages and has since been studied extensively. A broad range of computational effects is subsumed under this paradigm, e.g. store, non-determinism, exceptions, probabilities, and alternation.

In addition to just modeling side-effects, a matter of interest is how to reason about certain properties of such programs, one of the most important properties being functional correctness. Goncharov and Schröder [1] established a generic Hoare calculus for a monadic metalanguage. Generic here means that the monad \( T \) can be freely chosen, provided that it satisfies certain assumptions [1]:

- \( T \) is order-enriched, i.e. its Kleisli category carries a suitable cpo-enrichment
- \( T \) has a distinguished innocent submonad \( P \) abstractly capturing computations with only sufficiently harmless side-effects.

We call \((T, P)\) a predicated monad; \( \Omega = P1 \) serves as a type of truth values. We further demand that \((T, P)\) be sequentially compatible (see [1], Definition 28).

While being structurally rich enough for this framework, the exception monad \( TA = A + E \) (with \( E \) an object of exceptions) allows for an operation which does not fit into the scheme, namely catch : \( TA \rightarrow T(A + E) \), i.e. exception handling: The framework covers only operations that are algebraic, which catch is not. We explore a modified version of the generic Hoare calculus with explicit support for exception handling via abnormal postconditions [2]; we show that adding exceptions to a predicated monad again yields a predicated monad, and that the resulting Hoare calculus is also sound and relatively complete.

A similar calculus has been developed in [4], for which, however, no proof of relative completeness was given and which builds upon a different semantic framework: in particular, the treatment of loops in [4] relies on the base category of the monad being enriched over a suitable category of domains, while we require this only for the Kleisli category. Moreover, we generate the truth values, potentially intuitionistic, from the monad while in [4] they are assumed in the base category. Exceptions in our calculus are modeled using coproducts, hence it features sum types, case distinction, and a loop construction performing iterated case distinction. For coproducts, we add an operation \( ? : A + B \rightarrow PA \), which tests whether the argument is a left injection.

We extend the notion of Hoare triple given by \{\phi\} \( x \leftarrow p \) \{\psi\} \( \Leftrightarrow \) do \( \phi \); \( x \leftarrow p \); \( \psi \); ret \( x \) = do \( \phi \); \( p \) from [1] to quadruples containing an extra post-condition for abnormal termination [2,4]:
Definition (Hoare quadruple). Let $\phi, \psi : \Omega$ be assertions and let $\varepsilon : E \rightarrow \Omega$ be a predicate on exceptions. A Hoare quadruple with abnormal postcondition $\varepsilon$ is defined by the equivalence

$$\{ \phi \} x \leftarrow f(z) \{ \psi \mid \varepsilon \} (f \in \Sigma)$$

$$(\text{do } x \leftarrow e? ; \phi) \lor e?) x \leftarrow e? \{ \phi \mid \varepsilon \}$$

$$\{ \phi(t/x) \} x \leftarrow f(t) \{ \psi \mid \varepsilon \}$$

$$(\text{do } x \leftarrow p \{ \psi \mid \varepsilon \} \{ \psi \} y \leftarrow q \{ \chi \mid \varepsilon \}$$

$$\{ \phi \} y \leftarrow \text{do } x \leftarrow p; q \{ \chi \mid \varepsilon \}$$

$$(\phi \mid \varepsilon) x \leftarrow \text{ret}(t) \{ \phi \mid \varepsilon \}$$

$$(\phi) y \leftarrow \text{do } x \leftarrow p; q \{ \chi \mid \varepsilon \}$$

$$(\phi) x \leftarrow q \{ \psi \mid \varepsilon \} (\xi) x \leftarrow r \{ \psi \mid \varepsilon \}$$

$$(\phi \mid \varepsilon) x \leftarrow \text{case } c \text{ of } \text{inl } a \rightarrow q; \text{inr } b \rightarrow r \{ \psi \mid \varepsilon \}$$

$$\{ \phi \} x \leftarrow p \{ \psi \mid \varepsilon \} \psi \subseteq \psi' \subseteq \varepsilon' \{ \phi' \} x \leftarrow p \{ \psi' \mid \varepsilon' \}$$

$$\{ \varepsilon(e) \} x \leftarrow \text{raise } e \{ \bot \mid \varepsilon \}$$

$$\{ \phi \} y \leftarrow \text{catch } p \{ \text{case } y \text{ of } \text{inl } x \rightarrow \psi; \text{inr } e \rightarrow \varepsilon(e) \mid \bot \}$$

$$\{ \psi \} x \leftarrow q \{ \text{do } a \leftarrow e? ; \psi \lor \text{do } b \leftarrow e? ; \Xi \mid \varepsilon \} \{ \xi \} x \leftarrow r \{ \chi \mid \varepsilon \}$$

$$\{ \text{do } a \leftarrow e? ; \psi \lor \text{do } b \leftarrow e? ; \Xi \mid \varepsilon \} x \leftarrow \text{itcase } c \text{ of } \text{inl } a \rightarrow q; \text{inr } b \rightarrow r \{ \chi \mid \varepsilon \}$$

Fig. 1. Hoare calculus for order-enriched effects with exceptions.

Theorem. The Hoare calculus for order-enriched effects with exceptions in Figure 1 is sound. Furthermore, let $(T, P)$ be a sequentially compatible predicated monad and let the weakest preconditions for basic programs be expressible in the assertion language. Then the calculus is complete relative to the assertion language over $(T, P)$.

References

Towards a formal framework for analyzing stream processing systems

Adrián Riesco¹, Miguel Palomino¹, and Narciso Martí-Oliet¹

Universidad Complutense de Madrid, Spain
{ariesco,miguelpt,narciso}@ucm.es

With the rise of Big Data technologies [5], distributed stream processing systems (SPS) [1] have gained popularity in the last years. These systems are used to continuously process high volume streams of data, with applications including anomaly detection [1], low latency social media data aggregation [5], and the emergent Internet of Things (IoT) market [4]. Among them Spark Streaming [10] stands out as a particularly attractive option, with a growing adoption in the industry, so we will consider in particular some features of SPS in Spark Streaming. The core of Spark is a batch computing framework [9] that is based on manipulating so called Resilient Distributed Datasets (RDDs), which provide a fault-tolerant implementation of distributed multisets. Spark programmers are encouraged to define RDD transformations that are pure functions from RDD to RDD, and the set of predefined RDD transformations includes typical higher-order functions like map, filter, etc., as well as aggregations by key and joins for RDDs of key-value pairs. These notions of transformations are extended in Spark Streaming from RDDs to DStreams (Discretized Streams), which are series of RDDs corresponding to micro batches. These batches are generated at a fixed rate according to the configured batch interval. Spark Streaming is synchronous in the sense that given a collection of input and transformed DStreams, all the batches for each DStream are generated at the same time as the batch interval is met. Finally, it is also possible to consider window computations, in which we apply transformations over a sliding window of data.

Maude [3] is a high-level language and high-performance system supporting both equational and rewriting logic computation. Maude modules correspond to specifications in rewriting logic [6], a logic that allows the representation of many models of concurrent and distributed systems. This logic is an extension of membership equational logic [2], an equational logic that, in addition to equations, allows the statement of membership axioms characterizing the elements of a sort. Rewriting logic extends membership equational logic by adding rewrite rules that represent transitions in a concurrent system and can be nondeterministic. It is important to note that Maude specifications are executable and subject to multiple analysis, including LTL model checking. We observe an increased interest in defining various languages to cover many programming language paradigms in rewriting logic. This desideratum is stated in the rewriting logic semantics
project [7], where the programming languages semantics are defined as rewriting systems using Maude, and it is followed by the work in the K framework [8].

Since modern technologies such as Spark and Spark Streaming have evolved quickly, they lack in general the formal basis required to analyze them. In this paper we propose rewriting logic as a suitable framework for formalizing stream processing systems. In fact, it is possible to abstract the concept of RDD to a list and transformations in batches to functional transformations on these lists. Hence, the stream would be processed by distinguishing, in different rewrite rules, the possible transformations on DStreams. In this way we can perform LTL model-checking at DStream level. Moreover, it is also possible to extend the model checking procedure to windows by instrumenting the rules to keep track of the batches consumed thus far.

We illustrate our framework with a simple example. Assume a seismograph network that, every second, produces a list of natural numbers (our RDD) where every number indicates the vibration detected in that place. The center in charge of processing this information must generate true if there exists a number in the list greater than 5 and false otherwise. Hence, the center can just apply a filter function to remove those values smaller than 6 and then check whether the thus obtained list is nonempty. In this way, the input RDD would be a list of numbers and the output RDD would be a (singleton) list of Boolean values. The associated input DStream would consist of a list of these RDDs; consuming each RDD corresponds to a rewrite rule application. In this simple scenario we are interested in the property it is always the case that, if the input RDD contains a number greater than 5, then the output RDD contains a unique value true, and it only contains false otherwise, which would be just defined as usual on states. However, it might be worth to check that the seismograph is working appropriately, so we can also consider that earthquakes must last for several second and then a given seismograph cannot detect an earthquake in an isolated instant. Thus, we could ask the system to satisfy that, if an earthquake is detected in a window, then at least two consecutive RDD detect an earthquake (except possibly for the first and the last batch in the window). This second property would require a more complex definition, since it requires data from multiple states.

References


Towards a formal framework for stream processing systems


Abstract. Abstract state machines (ASMs), originally named evolving algebras, emerged in the mid 1990s [1,2] as a model of computation that built on the concepts embodied in algebraic specifications. Amongst the many and varied applications of abstract state machines [3,4], the formal semantics of SDL [5], the ITU standard specification and description language [6], is one of the most striking, both in scale and in practical application.

SDL has a graphical and a textual language, with semantics based on an ASM model of communicating agents. The SDL formal semantics has enabled development of tools to support creation of formal executable specifications, which provide a basis for simulation, automated testing, application generation and deployment. Current tool providers include Pragmadev\(^1\), Cinderella\(^2\), and, using a UML profile for SDL, IBM Rational tau\(^3\). SDL has a well-established record in telecommunications and embedded systems, and a history of revision and adaptation to meet changing real-world requirements. This evolution is facilitated by the ITU standardization process.

SDL is therefore well positioned to meet the challenges posed by the Internet of Things (IoT) [7]. These include creating formal specifications that capture the essential characteristics of different communicating devices operating in different physical environments, dealing with data of varying and uncertain quality and interfacing with a wide variety of data analysis applications. Reliability, security and resilience to unwanted interactions with external systems at every level must also be expressed.

Examples from smart homes, field robotics, and environmental sensing illustrate the benefits of formal specification and simulation in these areas, and indicate how the language and its underlying ASM model are likely to evolve in the future. The interplay between abstract mathematical models, language development, standardization and tool development is seen as key to motivating advances in all these areas.

Keywords: Internet of Things (IoT), abstract state machines, formal specification, SDL (Z.100), evolving algebras

---

1 http://www.pragmadev.com
2 http://cinderella.dk/
References

Foundations of Graph Transformation as a Logic-Programming Language

Ionuț Țuțu\textsuperscript{1,2} and José Luiz Fiadeiro\textsuperscript{1}

\textsuperscript{1} Department of Computer Science, Royal Holloway University of London
\textsuperscript{2} Institute of Mathematics of the Romanian Academy,
Research group of the project ID-3-0439
ittutu@gmail.com, jose.fiadeiro@rhul.ac.uk

The use of graphs, graph grammars, and of algebraic, rule-based graph transformations for the specification and analysis of software systems has a long tradition, dating back to the early seventies \cite{1}. Its focus is on those systems whose states can be formalized as graph-like structures, and whose evolution can be presented by means of discrete graph reconfigurations. This gives the graph-transformation paradigm a wide range of applicability, from formal language theory and compiler construction – the original motivation for studying graph grammars – to database design and modelling of concurrent and distributed systems. The volumes of the Handbook of Graph Grammars and Computing by Graph Transformation \cite{2} and the more recent monographs \cite{3,4} serve as evidence of the extensive attention that the field has received in the literature.

Despite having such a well-established theory, and in contrast with the comprehensive research on the operational semantics of graph transformation, aspects related to its denotational, model-theoretic semantics have yet to be sufficiently explored. This observation was originally made in \cite{5}, where the authors addressed the presentation of several algebraic approaches to graph transformation in the abstract framework of institutions \cite{6} – a framework that arose in the eighties in the field of algebraic specification out of the need to formalize the intuitive notion of logical system. We continue this line of research, but focus instead on the presentation of graph transformation in the context of substitution systems \cite{7}; these extend institutions by allowing for direct representations of variables and substitutions – together with appropriate abstractions of notions such as ground sentence, model, and satisfaction – enabling in this way the development of a logic-independent form of logic programming.

Our work and \cite{5} share similar objectives, but the means to achieve them, and the technical details involved, are quite different. For example, we regard graph-transformation rules as universally quantified sentences, much like the Horn clauses used in conventional term rewriting \cite{8}, whereas in \cite{5} they are formalized as atomic sentences whose semantics captures fully the application of the rules – the universal quantification is implicit. Moreover, because we aim to define a substitution system, we distinguish between ground models, reminiscent of the graph transition systems from \cite{9}, and valuations of variables, which correspond to potential matches for the rules. But the biggest difference lies in the way we define the translation of sentences (using pushouts instead of right-composition functors) and the reduction of valuations (using left-composition functors instead
of pullbacks) along substitutions. This allows us to prove that the satisfaction condition holds for substitutions regardless of the constructions used to define the reconfiguration of graphs: besides the classical double-pushout construction [1], we also consider faithful double-pullback transitions as in [10].

Building on the presentation of rules as universally quantified sentences, the proposed formalism makes possible the development of a graph-transformation variant of logic programming by following the general approach put forward in [7]; this departs from the more traditional view of logic programming as graph rewriting (see, e.g. [11]), which deals instead with the translation of relational programs into a particular kind of graph grammars. We show that every program (i.e. collection of rules) built over the substitution system of graph transformation admits an initial model, which corresponds to the free graph transition system generated by the rules. Together with the local sentences of the substitution system, this model is also shown to satisfy the hypotheses under which Herbrand’s theorem is known to hold, thus enabling us to establish the necessary link between the denotational and the operational semantics of graph transformation.

References

Formalising Modular SOS Labels in Isabelle/HOL

Ferdinand Vesely
Department of Computer Science, Swansea University
Swansea SA2 8PP, United Kingdom
csfvesely@swansea.ac.uk

Modular SOS (MSOS) [3] is a modular variant of Structural Operational Semantics, developed to address the modularity issues of ordinary SOS. In MSOS, auxiliary entities are moved from configurations into transition labels and label variables are used to propagate unmentioned entities throughout the rule. Configurations thus only contain program terms, including values resulting from computations. Labels are morphisms of a category, essentially representing how the entities change during a transition. The overall label is an indexed product (finite map or record) composed of label components. Each component has an underlying category, which dictates when two morphisms are composable. The product label is composable when it is composable component-wise. Labels on adjacent transitions have to be composable. Due to the nature of labels, unobservable (silent) transitions are simply those labelled by identity morphisms.

In practice, MSOS uses ML-like record notation for label patterns in rules with the convention that unprimed indices denote the readable and primed indices refer to the writable component of an entity. Unmentioned components of the label are matched by a label variable. For example, the pattern ‘\{sto = \sigma, sto’ = \sigma’[\alpha’ \mapsto 42], \ldots\}’ matches any label that updates the store \sigma with the mapping \alpha’ \mapsto 42, but it may contain further components. The label variable ‘\ldots’ matches any remaining unmentioned components of the label. Were it replaced with ‘\_’, only unobservable remaining components would be matched. As special cases, ‘\{\ldots\}’ matches labels that are fully unobservable, while ‘\{\ldots\}’ matches any label. In a single rule, a label variable always represents the same components, which allows propagation of entities throughout the rule. Here are example rules for an assignment command, using environments (\rho) to map identifiers (i) to store (\sigma) locations (l):

\[
\frac{e \xrightarrow{\ldots} e'}{i := e \xrightarrow{\ldots} i := e'} \quad \frac{e \in \text{dom}(\rho) \quad \rho(i) = l \quad \sigma’ = \sigma[l \mapsto v]}{i := v \xrightarrow{(\text{env} = \rho, \text{sto} = \sigma, \text{sto’} = \sigma’)} \text{skip}}
\]

The left rule allows the expression e to be gradually evaluated into a value. We use the label variable ‘\ldots’ to propagate any (potentially observable) components between the premise and the conclusion. The right rule applies when the right-hand side expression has been fully evaluated into a value v. The environment is used to look up the corresponding location in the store, which is updated to hold the new value. The label variable ‘\_’ asserts that the rest of the label is unobservable.
Here we present the first formalisation of MSOS labels in the theorem prover Isabelle/HOL. It follows an early exposition of the foundations of MSOS [2]. Our main motivation is to facilitate bisimulation proofs based on MSOS specifications [1,4]. Instead of using a concrete representation for labels, we develop a theory of MSOS labels abstractly, using locale and type classes to state axioms that should hold for any representation.

We avoid working with objects of label categories altogether. Morphisms are elements of any type, as long as they are equipped with a notion of composition and identity. With objects, the domain (dom) and co-domain (cod) of morphisms would be used to determine the left and right identity, and whether morphisms are composable. Instead, we equip the morphism type with a binary composable predicate \( \text{Comp} :: \alpha \Rightarrow \alpha \Rightarrow \text{bool} \) and operations \( \text{ld}_L, \text{ld}_R :: \alpha \Rightarrow \alpha \), which respectively assign the left and right identity to a morphism. We prefer to write composition in application order and use the infix symbol \( \circ \) (with type \( \alpha \Rightarrow \alpha \Rightarrow \alpha \)). The axioms for these operations are modified versions of category axioms. Because we only work with a single type, the type of morphisms, we can define morphisms as a type class \textit{label-morphism}, giving us the benefit of overloaded operations. Furthermore, we can use the class to constrain the type of variables representing labels: \( 'l :: \alpha :: \text{label-morphism} \).

A product label is defined through a family of basic operations \( \text{get}_\text{ent} :: \alpha \Rightarrow \beta \) and \( \text{set}_\text{ent} :: \alpha \Rightarrow \beta \Rightarrow \alpha \) where \text{ent} represents the index of the entity, \( \alpha \) is the type of the product label and \( \beta \) is the type of the label component. Both \( \alpha \) and \( \beta \) have to be in the \textit{label-morphism} type class. These operations should satisfy various axioms, such as \( \text{get}_\text{ent} (\text{set}_\text{ent} \ l \ c) = c \), \( \text{set}_\text{ent} \ l (\text{get}_\text{ent} \ l) = l \), and \( (\text{set}_\text{ent} \ l_1 \ c_1) \circ (\text{set}_\text{ent} \ l_2 \ c_2) = \text{set}_\text{ent} \ (l_1 \circ l_2) (c_1 \circ c_2) \). For each label entity type we define a type class \textit{ent} which defines the corresponding get and set operations. For example, a type class for environments \textit{env} will define \( \text{get}_\text{env} \) and \( \text{set}_\text{env} \). Then we can require a label \( l \) to include an environment by constraining its type: \( 'l :: \alpha :: \text{env} \).

In our development, we test this formalisation on a small specification and use it to prove a simple bisimulation. While the formalisation is currently cumbersome, we intend to use it as a basis for a fuller development which will provide a translation from the usual record notation, as well as better automation support.

References

Semantics for non-incremental reconfigurations of Asynchronous Relational Networks*

Ignacio Vissani\textsuperscript{1,2} and Carlos G. Lopez Pombo\textsuperscript{1,2}

\textsuperscript{1} Department of Computing, FCEyN, Universidad de Buenos Aires, Argentina
\textsuperscript{2} Consejo Nacional de Investigaciones Científicas y Tecnológicas, Argentina
ivissani@dc.uba.ar clpombo@dc.uba.ar

In the new paradigm of Service-Oriented Computing (SOC), the structure of software systems is intrinsically dynamic. In this paradigm, software artefacts run over globally available computational network infrastructures and they rely on external services they may need to procure and bind to at runtime, in order to collectively fulfil certain business goals. Therefore, development is no longer a process in which subsystems are developed and integrated by skilled engineers but a runtime composition of services discovered and bound by a dedicated middleware.

In [1] we provided a trace semantics for the service component algebra presented in [2] – Asynchronous Relational Networks (ARNs) – that accounted for the fact that, because of runtime discovery and binding, ARNs are reconfigured on demand as the execution reaches the point in which a component is required to continue executing.

Our work resorts to the formalisation of services introduced in [3], in terms of hypergraphs whose nodes correspond to structured sets of messages that can be exchanged between the network components attached to that point, and whose hyperedges capture those elements of networks that account for computation and/or communication, i.e. processes and communication channels.

The logic ruling the behaviour of components and communication channels, as given in [3,1], is formalised in terms of Muller automata [4], therefore the execution of both, components and communication channels, is prescribed to be infinite, as a consequence of the acceptance condition of this family of automata. That particular choice for the ARNs semantics forces the reconfiguration of a SOC software artefact to be necessarily incremental, thus bound services execute indefinitely accompanying the execution of the activity that triggered the reconfiguration.

In this work we explore non-incremental reconfigurations as the result of unbinding services. This situation may be the outcome of: a) the execution of a service whose goal is met in a finite number of steps, and b) the failure of a service to meet its goal.

Regarding the first situation, actual services are generally atomic and stateless; and, going further in the conceptual bases of the paradigm, each binding resulting

\* The research of Carlos G. Lopez Pombo has been supported by the grants UBACyT 2002/01302/00092BA, PICT 2013-2129, and PIP 11220130100148CO.
from a discovery triggered by the execution of an activity does not need to be
served by the same service as it may not be available at that particular time, or it
may not be the best suited for the task. In general, the existing formalisations
of services prescribe that services are reactive software artefacts, thus, once a service
is bound, it remains bound and there is no notion of termination. To overcome
this limitation we propose to model the computational aspects of terminating
services as finite state automata such that they coexist in a coherent way with
the non-terminating services whose computation is modelled by Muller automata.
We call this class of networks Hybrid ARNs.

The second situation emerges from the fact that the execution of communici-
tating software systems, running over a fallible communication infrastructure is
likely to experience errors derived from the temporal unavailability of a given
service bound to the executing software. SOC in the real world need to cope with
this kind of errors as an absolute minimum fault tolerance criterion. Our proposal
is to adapt the formal framework allowing the traces of a Hybrid ARN to include
spontaneous “structure-decreasing” transitions from one state to another.

These transitions are simple from the point of view of the transformation
suffered by the network but, at the same time, they give rise to the question of to
which state of the system it should transition to after the detection of the failure.
We explore several answers to this question determining a family of rollback
policies, depending on how aggressive the rollback is, and how computationally
complex is to calculate better solutions.

References

   for asynchronous relational networks. In Diaconescu, R., Codescu, M., Țuțu, I., eds.: Pro-
   Volume 9463 of Lecture Notes in Computer Science., Sinaia, Romania, Springer-Verlag (September 2015) 131–150
   Computer Science (503) (2013) 1–30
3. Țuțu, I., Fiadeiro, J.L.: A logic-programming semantics of services. In Heckel, R.,
   Milius, S., eds.: Proceedings of 5th International Conference Algebra and Coalgebra in
   Computer Science – CALCO 2013. Volume 8089 of Lecture Notes in Computer
   Science., Springer-Verlag (2013) 299–313
   Annual Symposium on Switching Circuit Theory and Logical Design,Chicago, Illinois,
Algebraic Databases - Extended Abstract

Patrick Schultz\textsuperscript{2}, David Spivak\textsuperscript{2}, Christina Vasilakopoulou\textsuperscript{2}, Ryan Wisnesky\textsuperscript{1}

\textsuperscript{1} Categorical Informatics
\textsuperscript{2} MIT

In this talk we present recent work on describing database schemas, instances, queries, and data-integration related artifacts using algebraic theories (i.e., theories in multi-sorted equational logic \cite{4}). This work is described in two papers currently under review: \cite{5} (which focuses on mathematics) and \cite{7} (which focuses on computer science). Both papers are available at the project website, http://categoricaldata.net/fql.html.

Our work extends a particular category-theoretic data model, the so-called “functorial data model”, which originated in the late 1990s \cite{2}. See \cite{8} and \cite{9} for a comparison of the functorial data model and the relational model, and \cite{6} for a description of a real-world use of the functorial data model. In this talk our focus is on building a bridge to the algebraic theory community by describing our work using only algebraic terminology.

In our formalism database schemas and instances are algebraic theories of a certain kind. The database instances on a schema $S$ constitute a category, denoted $S\text{–Inst}$, and a morphism of schemas \cite{1} $F : S \rightarrow T$ is defined as a morphism of algebraic theories obeying certain restrictions. Substitution along $F : S \rightarrow T$ denotes a functor $\downarrow F : S\text{–Inst} \rightarrow T\text{–Inst}$ with a right adjoint, $\Delta F : T\text{–Inst} \rightarrow S\text{–Inst}$, which in turn has a right adjoint, $\Pi F : S\text{–Inst} \rightarrow T\text{–Inst}$. The $\downarrow, \Delta, \Pi$ data migration functors provide a category-theoretic alternative to the traditional relational operations for both querying data (relational algebra) and migrating / integrating data (the chase \cite{1}), and have been implemented in an open-source data integration tool we call FQL, available at the project website. Note that the restrictions on algebraic theories are critical: in many logics, analogs of $\downarrow, \Delta, \Pi$ need not exist, or need not be adjoint \cite{3}.

We now describe our formalism. We first fix an algebraic theory, $Ty$, called the type side of our formalism. The type side represents the computational context within which our data resides. For example, the type side for SQL contains the function symbols and equations defined by the SQL standard. An example is:

\begin{align*}
\text{Sorts} & := \{\text{Nat, Char, String}\} \\
\text{Symbols} & := \{0 : \text{Nat}, \text{succ : Nat} \rightarrow \text{Nat}, \ A : \text{Char}, \ B : \text{Char}, \ldots, \ Z : \text{Char}, \ \text{nil : String, cons : Char} \times \text{String} \rightarrow \text{String}, + : \text{Nat} \rightarrow \text{Nat}\} \\
\text{Equations} & := \{\forall x. \ x + 0 = x, \ \forall x, y. \ x + \text{succ}(y) = \text{succ}(x + y)\}
\end{align*}

A schema is an algebraic theory extending $Ty$ obeying certain conditions which we omit here. For example (writing $x.f$ for $f(x)$):

\begin{align*}
\text{Sorts} & := \{\text{Emp, Dept}\}
\end{align*}
Symbols := \{ \text{mgr} : \text{Emp} \to \text{Emp}, \text{wrk} : \text{Emp} \to \text{Dept}, \text{secr} : \text{Dept} \to \text{Emp} \},
\text{dname} : \text{Dept} \to \text{String}, \text{ename} : \text{Emp} \to \text{String}

Equations := \{ \forall v. v.\text{mgr}.\text{wrk} = v.\text{wrk}, \forall v. v = v.\text{secr}.\text{wrk}, \forall v. v.\text{mgr}.\text{mgr} = v.\text{mgr} \}

An instance on schema $S$ is an algebraic theory extending $S$, obeying certain conditions which we omit here. The intended meaning of an instance is its term model, which we can visually present using a set of tables, with one table per entity with an ID column corresponding to the carrier set. For example,

\[
\text{Symbols} := \{ a, b, c : \text{Emp}, m, s : \text{Dept} \}
\]
\[
\text{Eqs} := \{ a.\text{ename} = \text{Al}, c.\text{ename} = \text{Carl}, m.\text{dname} = \text{Math}, a.\text{wrk} = m, b.\text{wrk} = m, s.\text{secr} = c, m.\text{secr} = b \}
\]

<table>
<thead>
<tr>
<th>Dept</th>
<th>ID</th>
<th>dname</th>
<th>secr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>m</td>
<td>Math</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>s.dname</td>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Emp</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
<tr>
<td>a.mgr</td>
</tr>
<tr>
<td>b.mgr</td>
</tr>
<tr>
<td>c.mgr</td>
</tr>
</tbody>
</table>

Using these definitions of schema and instance we define morphisms of schemas and morphisms of instances, arrange these artifacts into categories, and define the $\Sigma$, $\Delta$, $\Pi$ data migration functors.

References

Design and Validation of the P-Store
Replicated Data Store in Maude*

Peter Csaba Ölveczky

1 University of Oslo
2 University of Illinois at Urbana-Champaign

Introduction. Many large applications—such as Google search, Gmail, Facebook, Dropbox, eBay, online banking and card payment processing—are expected to be available continuously, even under peak load, congestion in parts of the network, server failures, and during scheduled hardware or software upgrades. Such applications also typically manage large amounts of data. To achieve the desired availability, the data must be replicated across geographically distributed sites, and to achieve the desired scalability and elasticity, the data store may have to be partitioned across multiple partitions.

It is well known [2] that it is hard or impossible to both guarantee strong correctness properties (such as serializability), high availability, and strong fault tolerance. The lack of strong correctness guarantees is acceptable for applications such as Google search, Facebook, and online newspapers, but is unacceptable in, e.g., online banking, online commerce (eBay and airplane reservation systems), and medical information systems. P-Store [6] is a well-known replicated and partitioned data store that provides both serializability and some fault tolerance (e.g., transactions can be validated even when some nodes participating in the validation are down).

Wide-area replicated data stores designs are typically evaluated on real implementations or by using simulation tools, both of which are laborious tasks which cannot check “corner cases” to guarantee the absence of errors. In this talk I will talk about the use of the rewriting-logic-based Maude language and tool [3] to formally specify and analyze P-Store.

Why is this interesting? Although cloud computing data stores are complex artifacts, members of the University of Illinois Assured Cloud Computing center have exploited the simplicity and expressiveness of Maude to formally define and analyze complex cloud computing data stores such as Google’s Megastore and Apache Cassandra [4,5]. So, why is formalizing P-Store interesting? First, it is a well-known data store in its own right with many good properties that combines both wide-area replication, data partition, some fault tolerance, serializability, and limited use of atomic multicast. Second, it uses atomic multicast to order concurrent transactions. Third, it uses “group communication” for atomic commitment. The point is that both atomic multicast and group communication commitment seem to be key generic building blocks in the design of cloud computing data stores (see, e.g., [1]) and that previous efforts have not formalized

* This work was partially supported by AFOSR/AFRL Grant FA8750-11-2-0084.
atomic multicast or group communication commitment. There are different protocols aimed at achieving atomic multicast. Instead of formalizing one of those, one of the main contributions of the P-Store formalization is an abstract Maude model of atomic multicast that allows any possible ordering of message reception consistent with atomic multicast.

Results. I have modeled both versions of P-Store, and performed model checking on small system configurations. Maude analysis showed some significant errors, like read-only transactions never getting validated in certain cases. An author of the original P-Store paper [6] confirmed that I had indeed found a nontrivial mistake in their algorithm and suggested a way of correcting the mistake. Maude analysis of the corrected algorithm did not show this error. I also found another error; the P-Store author said that this was not an error, but acknowledged that P-Store relied on a crucial assumption that was entirely missing from the paper. Finally, a key definition was very easy to misunderstand because of how it was phrased in English; fortunately, the P-Store author helped us clarify its meaning. However, it emphasizes the need for a formal specification in addition to the standard prose-and-pseudo-code description of the algorithm.

At the moment, I model check the system by defining interesting concurrent transactions and analyze the possible outcomes. In the near future, I should develop a systematic way of analyzing serializability, possibly by adding some kind of observer to the system state. A general technique for analyzing serializability of concurrent transactions would also be an interesting contribution. The executable Maude specifications of the different versions of P-Store, together with analysis commands, are available at http://folk.uio.no/peterol/WADT16.

References