LNCS 10644

Phillip James
Markus Roggenbach (Eds.)

Recent Trends in Algebraic Development Techniques

23rd IFIP WG 1.3 International Workshop, WADT 2016
Gregynog, UK, September 21–24, 2016
Revised Selected Papers

ifip

Springer

Lecture Notes in Computer Science 10644

Phillip James · Markus Roggenbach (Eds.)

Recent Trends in Algebraic Development Techniques

23rd IFIP WG 1.3 International Workshop, WADT 2016
Gregynog, UK, September 21–24, 2016
Revised Selected Papers

Springer

Editors

Phillip James
Swansea University
Swansea
UK

Markus Roggenbach
Swansea University
Swansea
UK

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-72043-2 ISBN 978-3-319-72044-9 (eBook)
https://doi.org/10.1007/978-3-319-72044-9

Library of Congress Control Number: 2017960875

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The 23rd International Workshop on Algebraic Development Techniques (WADT 2016) took place in Gregynog, Wales, UK, during September 21–24, 2016. The workshop took place under the auspices of IFIP WG 1.3 and was organized by the Department of Computer Science of Swansea University, UK. At the workshop, there were three invited talks and 20 contributed presentations, covering specification languages such as Event-B, CASL, and Maude, foundations of system specification such as institutions, monads, logics and their combinations, axiomatizations of data types, graph models and graph transformations, and applications including algebraic databases, service-oriented computing, ontologies, and the Internet of Things. Participants of the workshop travelled from Argentina, Canada, France, Germany, Ireland, Italy, The Netherlands, Norway, Poland, Portugal, Spain, Sweden, the USA, and the UK. This volume contains selected, peer-reviewed papers that were invited for submission after the workshop.

Fig. 1. Participants of WADT 2016 at Gregynog.

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). The workshop provided an opportunity to present recent and ongoing work, to meet colleagues, and to discuss new ideas and future trends. Typical 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, quality assurance, validation, and verification

The WADT can look back on a proud history of workshops. The first workshop took place in 1982 in Sorpesee, followed by Passau (1983), Bremen (1984), Braunschweig (1986), Gullane (1987), Berlin (1988), Wusterhausen (1990), Dourdan (1991), Caldes de Malavella (1992), S. Margherita (1994), Oslo (1995), Tarquinia (1997), Lisbon (1998), Chateau de Bonas (1999), Genoa (2001), Frauenchiemsee (2002), Barcelona (2004), La Roche en Ardenne (2006), Pisa (2008), Etelsen (2010), Salamanca (2012), and Sinaia (2014).

These proceedings collect selected contributions of varying nature:

- Kenneth Johnson, John Tucker, and Victoria Wang contribute a fully peer-reviewed paper based on their invited talk: “Theorizing Monitoring: Algebraic Models of Web Monitoring in Organisations.”
- Alessio Lomuscio and Till Mossakowski presented invited talks at the workshop. These proceedings include abstracts for these talks, “Advances in Verification of Multi-Agent System” and “The Distributed Ontology, Model and Specification Language – DOL”, respectively.
- Furthermore, this volume includes two fully peer-reviewed survey papers. Renato Neves, Alexandre Madeira, Luis Barbosa, and Manuel A. Martins, “Asymmetric Combination of Logics Is Functorial: A Survey”; and Ryan Wisnesky, David I. Spivak, and Patrick Schultz, “Algebraic Model Management.”
- Finally, the main body of this volume comprises nine peer-reviewed papers that present new results in the field of algebraic development techniques.

We hope that reading the contributions in this volume will bring as much joy as we had at our workshop in September 2016 in Gregynog.

June 2017

Phillip James
Markus Roggenbach

Organization

Steering Committee

Andrea Corradini	Università di Pisa, Italy
José Luiz Fiadeiro	Royal Holloway, University of London, UK
Rolf Hennicker	Ludwig-Maximilians-Universität, Germany
Hans-Jörg Kreowski	Universität Bremen, Germany
Till Mossakowski	Otto-Von-Guericke-Universität Magdeburg, Germany
Fernando Orejas	Universitat Politécnica de Catalunya, Spain
Francesco Parisi-Presicce	Università di Roma, Italy
Markus Roggenbach (Chair)	Swansea University, UK
Grigore Roşu	University of Illinois at Urbana-Champaign, USA
Andrzej Tarlecki	Warsaw University, Poland

Program Committee

Mihai Codescu	Libera Università di Bolzano, Italy
Andrea Corradini	Università di Pisa, Italy
José Luiz Fiadeiro	Royal Holloway, University of London, UK
Rolf Hennicker	Ludwig-Maximilians-Universität, Germany
Phillip James (Co-chair)	Swansea University, UK
Einar Broch Johnsen	Universitetet i Oslo, Norway
Alexander Knapp	Universität Augsburg, Germany
Narciso Martí-Oliet	Universidad Complutense de Madrid, Spain
Till Mossakowski	Otto-Von-Guericke-Universität Magdeburg, Germany
Mohammadreza Moussavi	Högskolan i Halmstad, Sweden
Peter Ölveczky	Universitetet i Oslo, Norway
Detlef Plump	University of York, UK
Florian Rabe	Jacobs University, Germany
Markus Roggenbach (Co-chair)	Swansea University, UK
Lutz Schröder	Friedrich-Alexander Universität, Germany
Ionut Tutu	Royal Holloway, University of London, UK

Additional Reviewers

Hubert Baumeister	Danmarks Tekniske Universitet, Denmark
Ferruccio Damiani	Università Degli Studi Di Torino, Italy
David Frutos Escrig	Universidad Complutense de Madrid, Spain
Martin Glauer	Otto Von Guericke Universität Magdeburg, Germany
Sergey Goncharov	Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Contents

Abstracts of Invited Talks

Advances in Verification of Multi-agent Systems

Alessio Lomuscio(✉)

Department of Computing, Imperial College London, London, UK
a.lomuscio@imperial.ac.uk

Abstract. I was honoured by the opportunity to share with the WADT 2016 attendees some of the recent work in our lab on verifying multi-agent systems (MAS) against agent-based specifications.

MAS are distributed autonomous systems in which the components, or agents, act autonomously in order to reach private or common goals. MAS have been used as a paradigm to realise a wide number of applications ranging from autonomous systems and robotics to services, electronic assistants, and beyond. Logic-based specifications for MAS typically do not refer only to the agents' temporal evolution, but also to their knowledge, strategic abilities, and other AI-inspired primitives.

I began by reporting algorithms for symbolic model checking against epistemic and strategic specifications [1,2]. I highlighted potential speedups of these techniques via a number of techniques including symmetry reduction [3], parallel approaches [4], and SAT-based methods [5].

I then demonstrated MCMAS [6,7], 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 was discussed [8,9] as well applications to the verification of artifact-based services [10,11].

I concluded by considering the case of MAS where the number of agents is unbounded and cannot be determined at design time. This is a typical assumption in robotic swarms and recent internet of things applications. In view of solving this, I reported our approach to the parameterised model checking problem. While this is generally undecidable, I presented results that establish sufficient conditions for determining a *cut-off* of a MAS [12–14], i.e., the number of agents that need to analysed for verifying a MAS composed of any number of components. I concluded by presenting applications to the verification of related notions, such as emergence [15–17].

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

References

1. Raimondi, F., Lomuscio, A.: Automatic verification of multi-agent systems by model checking via OBDDs. J. Appl. Logic **5**(2), 235–251 (2005)

P. James and M. Roggenbach (Eds.): WADT 2016, LNCS 10644, pp. 3–4, 2017.
https://doi.org/10.1007/978-3-319-72044-9_1

2. Cermák, P., Lomuscio, A., Mogavero, F., Murano, A.: Verifying and synthesising multi-agent systems against one-goal strategy logic specifications. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI15), pp. 2038–2044. AAAI Press (2015)
3. Cohen, M., Dam, M., Lomuscio, A., Qu, H.: A symmetry reduction technique for model checking temporal-epistemic logic. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI09), pp. 721–726 (2009)
4. Kwiatkowska, M., Lomuscio, A., Qu, H.: Parallel model checking for temporal epistemic logic. In: Proceedings of the 19th European Conference on Artificial Intelligence (ECAI10), pp. 543–548. IOS Press (2010)
5. Kacprzak, M., Lomuscio, A., Penczek, W.: From bounded to unbounded model checking for temporal epistemic logic. Fundamenta Informaticae **63**(2–3), 221–240 (2004)
6. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: a model checker for the verification of multi-agent systems. Softw. Tools Technolo. Trans. **19**(1), 9–30 (2017)
7. Čermák, P., Lomuscio, A., Mogavero, F., Murano, A.: MCMAS-SLK: a model checker for the verification of strategy logic specifications. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 525–532. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_34
8. 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), pp. 1659–1664. AAAI Press (2011)
9. Ezekiel, J., Lomuscio, A.: Combining fault injection and model checking to verify fault tolerance, recoverability, and diagnosability in multi-agent systems. Inf. Comput. **254**(2), 167–194 (2017)
10. Gonzalez, P., Griesmayer, A., Lomuscio, A.: Verification of GSM-based artifact-centric systems by predicate abstraction. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 253–268. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_16
11. Gonzalez, P., Griesmayer, A., Lomuscio, A.: Verifying GSM-based business artifacts. In: Proceedings of the 19th International Conference on Web Services (ICWS12), pp. 25–32. IEEE Press (2012)
12. Kouvaros, P., Lomuscio, A.: Automatic verification of parametrised interleaved multi-agent systems. In: Proceedings of the 12th International Conference on Autonomous Agents and Multi-agent systems (AAMAS13), IFAAMAS, pp. 861–868 (2013)
13. Kouvaros, P., Lomuscio, A.: A cutoff technique for the verification of parameterised interpreted systems with parameterised environments. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI13), pp. 2013–2019. AAAI Press (2013)
14. Kouvaros, P., Lomuscio, A.: Parameterised verification for multi-agent systems. Artif. Intell. **234**, 152–189 (2016)
15. Kouvaros, P., Lomuscio, A.: A counter abstraction technique for the verification of robot swarms. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI15), pp. 2081–2088. AAAI Press (2015)
16. Kouvaros, P., Lomuscio, A.: Verifying emergent properties of swarms. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15), pp. 1083–1089. AAAI Press (2015)
17. Kouvaros, P., Lomuscio, A.: Formal verification of opinion formation in swarms. In: Proceedings of the 15th International Conference on Autonomous Agents and Multi-agent systems (AAMAS16), IFAAMAS, pp. 1200–1209 (2016)

The Distributed Ontology, Model and Specification Language – DOL

Till Mossakowski(✉)

Institute of Intelligent Cooperating Systems,
Otto-von-Guericke-University Magdeburg, Magdeburg, Germany
till@iks.cs.uni-magdeburg.de

Over the last decades, the WADT community has studied the formal specification of software (and hardware) in great detail [1,9,42]. One important aspect is the structuring of specifications in a modular way [43], which has been covered in specification languages like CLEAR [6], OBJ [18], ASL [46] and many others. Here, a powerful abstraction is the notion of institution, introduced by Goguen and Burstall [17]. 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 [41], 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. [25]) und used for the notion of ontology module extraction in various description logics (see e.g. [21], and [19] 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 [23,32].

Moreover, using the notion of heterogeneous multi-logic specification developed in [2,12,14,22,27,28,36,44], a program for the institution-based formalisation of UML multi-viewpoint *models* has been formulated [7,8,20]. 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 [31,33,38]. **O**ntologies, MDE **m**odels and **s**pecifications 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

© IFIP International Federation for Information Processing 2017
Published by Springer International Publishing AG 2017. All Rights Reserved
P. James and M. Roggenbach (Eds.): WADT 2016, LNCS 10644, pp. 5–10, 2017.
https://doi.org/10.1007/978-3-319-72044-9_2

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 [40,42] 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 [21]. 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 [40,42]. The problem of capturing this theory by a finite presentation has been studied for ontology languages under the terms *forgetting* and *uniform interpolation* [24,45].
- **Filtering.** Extraction of a sub-OMS consisting of all sentences that actually are formed over the restricted signature [39].

Minimization. Whereas free specifications in CASL allow the selection of the least intepretation of e.g. predicates, minimization allows the selection of all minimal interpretations, following McCarthy's circumscription [26]. 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 [42] (logically: interpretations of theories [15], in terms of OBJ [18] and CASL [3,37]: views). The refinement language of [30] 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 ([11] had not been available when the DOL standard emerged).

Equivalence. OMS can be declared to equivalent, if they have a common definitional extension [22,35]

Alignment. This notion is a relational generalisation of signature morphisms (which are typically functional in nature) [13,16,47]. 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 [16] 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 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 is available at omg.org/spec/DOL; 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 ontohub.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 [5,29] to the whole of DOL is an important task.

Acknowledgements. The author wishes to thank the community that has 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

1. Astesiano, E., Kreowski, H.-J., Krieg-Brückner, B.: Algebraic Foundations of Systems Specification. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-642-59851-7
2. Bernot, G., Coudert, S., Le Gall, P.: Towards heterogeneous formal specifications. In: Wirsing, M., Nivat, M. (eds.) AMAST 1996. LNCS, vol. 1101, pp. 458–472. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0014333
3. Bidoit, M., Mosses, P.D.: CASL—The Common Algebraic Specification Language: User Manual. LNCS, vol. 2900. Springer, Heidelberg (2004). https://doi.org/10.1007/b11968
4. Borgida, A., Serafini, L.: Distributed description logics: assimilating information from peer sources. J. Data Semant. **1**, 153–184 (2003)
5. Borzyszkowski, T.: Logical systems for structured specifications. Theoret. Comput. Sci. **286**, 197–245 (2002)
6. Burstall, R.M., Goguen, J.A.: The semantics of clear, a specification language. In: Bjøorner, D. (ed.) Abstract Software Specifications. LNCS, vol. 86, pp. 292–332. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10007-5_41

7. Calegari, D., Mossakowski, T., Szasz, N.: Heterogeneous verification in the context of model driven engineering. Sci. Comput. Program. **126**, 3–30 (2016)
8. Cengarle, M.V., Knapp, A., Tarlecki, A., Wirsing, M.: A heterogeneous approach to UML semantics. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 383–402. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68679-8_23
9. Cerioli, M., Gogolla, M., Kirchner, H., Krieg-Brückner, B., Qian, Z., Wolf, M.: Algebraic System Specification and Development. A Survey and Annotated Bibliography, BISS monographs, vol. 3, 2nd edn. Shaker Verlag (1997)
10. Codescu, M., Mossakowski, T., Kutz, O.: A categorical approach to networks of aligned ontologies. J. Data Semant. **6**(4), 155–197 (2017). https://doi.org/10.1007/s13740-017-0080-0
11. Codescu, M., Mossakowski, T., Sannella, D., Tarlecki, A.: Specification refinements: calculi, tools, and applications. Sci. Comput. Program. **144**, 1–49 (2017). https://doi.org/10.1016/j.scico.2017.04.005
12. Coudert, S., Bernot, G., Le Gall, P.: Hierarchical heterogeneous specifications. In: Fiadeiro, J.L. (ed.) WADT 1998. LNCS, vol. 1589, pp. 107–121. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48483-3_8
13. David, J., Euzenat, J., Scharffe, F., Trojahn dos Santos, C.: The alignment API 4.0. Semant. Web **2**(1), 3–10 (2011)
14. Diaconescu, R.: Grothendieck institutions. Appl. Cat. Struct. **10**, 383–402 (2002)
15. Enderton, H.B.: A Mathematical Introduction to Logic. Academic Press, Cambridge (1972)
16. Euzenat, J., Shvaiko, P.: Ontology Matching, 2nd edn. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38721-0
17. Goguen, J.A., Burstall, R.M.: Institutions: abstract model theory for specification and programming. J. Assoc. Comput. Mach. **39**, 95–146 (1992). Predecessor in: LNCS 164, 221–256 (1984)
18. Goguen, J., Kirchner, C., Kirchner, H., Mégrelis, A., Meseguer, J., Winkler, T.: An introduction to OBJ 3. In: Kaplan, S., Jouannaud, J.-P. (eds.) CTRS 1987. LNCS, vol. 308, pp. 258–263. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-19242-5_22
19. Ibañez, Y.A., Mossakowski, T., Sannella, D., Tarlecki, A.: Modularity of ontologies in an arbitrary institution. In: Martí-Oliet, N., Ölveczky, P.C., Talcott, C. (eds.) Logic, Rewriting, and Concurrency. LNCS, vol. 9200, pp. 361–379. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23165-5_17
20. Knapp, A., Mossakowski, T., Roggenbach, M.: Towards an institutional framework for heterogeneous formal development in UML—a position paper. In: De Nicola, R., Hennicker, R. (eds.) Software, Services, and Systems. LNCS, vol. 8950, pp. 215–230. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15545-6_15
21. Konev, B., Lutz, C., Walther, D., Wolter, F.: Formal properties of modularisation. In: Stuckenschmidt, H., Parent, C., Spaccapietra, S. (eds.) Modular Ontologies. LNCS, vol. 5445, pp. 25–66. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01907-4_3
22. Kutz, O., Mossakowski, T., Lücke, D.: Carnap, Goguen, and the hyperontologies: logical pluralism and heterogeneous structuring in ontology design. Log. Univers. **4**(2), 255–333 (2010)
23. Lucanu, D., Li, Y.F., Dong, J.S.: Semantic web languages – towards an institutional perspective. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 99–123. Springer, Heidelberg (2006). https://doi.org/10.1007/11780274_6

24. Lutz, C., Wolter, F.: Foundations for uniform interpolation and forgetting in expressive description logics. In: Walsh, T. (ed.) IJCAI, pp. 989–995. IJCAI/AAAI (2011)
25. Maibaum, T.S.E.: Conservative extensions, interpretations between theories and all that!. In: Bidoit, M., Dauchet, M. (eds.) CAAP 1997. LNCS, vol. 1214, pp. 40–66. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0030588
26. McCarthy, J.: Circumscription - a form of non-monotonic reasoning. Artif. Intell. **13**(1–2), 27–39 (1980)
27. Mossakowski, T.: Comorphism-based Grothendieck logics. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 593–604. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45687-2_49
28. Mossakowski, T.: Heterogeneous specification and the heterogeneous tool set. Habilitation thesis, University of Bremen (2005)
29. Mossakowski, T., Autexier, S., Hutter, D.: Development graphs - proof management for structured specifications. J. Logic Algebraic Program. **67**(1–2), 114–145 (2006)
30. Mossakowski, T., Sannella, D., Tarlecki, A.: A simple refinement language for Casl. In: Fiadeiro, J.L., Mosses, P.D., Orejas, F. (eds.) WADT 2004. LNCS, vol. 3423, pp. 162–185. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31959-7_10
31. Mossakowski, T., Codescu, M., Neuhaus, F., Kutz, O.: The distributed ontology, modelling and specification language - DOL. In: Koslow, A., Buchsbaum, A. (eds.) The Road to Universal Logic-Festschrift for 50th birthday of Jean-Yves Beziau, Volume II, Studies in Universal Logic. Birkhäuser (2015)
32. Mossakowski, T., Kutz, O.: The onto-logical translation graph. In: Kutz, O., Schneider, T. (eds.) Modular Ontologies. IOS, Amsterdam (2011)
33. Mossakowski, T., Kutz, O., Codescu, M., Lange, C.: The distributed ontology, modeling and specification language. In: Del Vescovo, C., Hahmann, T., Pearce, D., Walther, D. (eds.) Proceedings of the 7th International Workshop on Modular Ontologies (WoMO-13), CEUR-WS 1081 (2013)
34. Mossakowski, T., Schröder, L., Roggenbach, M., Reichel, H.: Algebraic-co-algebraic specification in CoCasl. J. Logic Algebraic Program. **67**(1–2), 146–197 (2006)
35. Mossakowski, T., Tarlecki, A.: Heterogeneous logical environments for distributed specifications. In: Corradini, A., Montanari, U. (eds.) WADT 2008. LNCS, vol. 5486, pp. 266–289. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03429-9_18
36. Mossakowski, T., Tarlecki, A.: A relatively complete calculus for structured heterogeneous specifications. In: Muscholl, A. (ed.) FoSSaCS 2014. LNCS, vol. 8412, pp. 441–456. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54830-7_29
37. Mosses, P.D. (ed.): Casl Reference Manual. LNCS, vol. 2960. Springer, Heidelberg (2004). https://doi.org/10.1007/b96103
38. Object Management Group. The distributed ontology, modeling, and specification language (DOL) (2016). OMG standard http://www.omg.org/spec/DOL
39. Rabe, F., Kohlhase, M.: A scalable module system. Inf. Comput. **230**(1), 1–54 (2013)
40. Goguen, J., Roşu, G.: Composing hidden information modules over inclusive institutions. In: Owe, O., Krogdahl, S., Lyche, T. (eds.) From Object-Orientation to Formal Methods. LNCS, vol. 2635, pp. 96–123. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-39993-3_7

41. Sannella, D., Tarlecki, A.: Specifications in an arbitrary institution. Inf. Comput. **76**, 165–210 (1988)
42. Sannella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-17336-3
43. Sannella, D., Wirsing, M.: Specification languages. In: Algebraic Foundations of Systems Specification [1], pp. 243–272
44. Tarlecki, A.: Towards heterogeneous specifications. In: Gabbay, D., de Rijke, M. (eds.) Frontiers of Combining Systems 2, 1998, Studies in Logic and Computation, pp. 337–360. Research Studies Press (2000)
45. Wang, Z., Wang, K., Topor, R.W., Pan, J.Z.: Forgetting for knowledge bases in DL-Lite. Ann. Math. Artif. Intell. **58**(1–2), 117–151 (2010)
46. Wirsing, M.: Structured algebraic specifications: a kernel language. Theor. Comput. Sci. **42**, 123–249 (1986)
47. Zimmermann, A., Krötzsch, M., Euzenat, J., Hitzler, P.: Formalizing ontology alignment and its operations with category theory. In: Proceedings of FOIS-06, pp. 277–288 (2006)

Full Papers of Invited Talks

Theorising Monitoring: Algebraic Models of Web Monitoring in Organisations

Kenneth Johnson[1], John V. Tucker[2(✉)], and Victoria Wang[3]

[1] School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Private Bag 92006, Auckland 1142, New Zealand
kenneth.johnson@aut.ac.nz

[2] Department of Computer Science, Swansea University, Singleton Park, Swansea SA2 8PP, UK
j.v.tucker@swansea.ac.uk

[3] Institute of Criminal Justice Studies, University of Portsmouth, St George's Building, 141 High Street, Portsmouth PO1 2HY, UK
victoria.wang@portsmouth.ac.uk

Abstract. Our lives are facilitated and mediated by software. Thanks to software, data on nearly everything can be generated, accessed and analysed for all sorts of reasons. Software technologies, combined with political and commercial ideas and practices, have led to a wide range of our activities being *monitored*, which is the source of concerns about surveillance and privacy. We pose the questions: *What is monitoring? Do diverse and disparate monitoring systems have anything in common? What role does monitoring play in contested issues of surveillance and privacy?* We are developing an abstract theory for studying monitoring that begins by capturing structures common to many different monitoring practices. The theory formalises the idea that monitoring is a process that observes the behaviour of people and objects in a context. Such entities and their behaviours can be represented by abstract data types and their observable attributes by logics. In this paper, we give a formal model of monitoring based on the idea that behaviour is modelled by streams of data, and apply the model to a social context: the monitoring of web usage by staff and members of an organisation.

Keywords: Context · Monitoring · Records · Interventions
Surveillance · Organisation · Employee monitoring · Web monitoring
Abstract data types · Algebraic specification · Streams

1 Introduction

Our professional, economic, social and personal lives are facilitated and mediated by software running on computers and networks. Software exists to input, process and output data and, in particular, the software that drives computers and networks generate extensive data about their own operations. Given the diversity and ubiquity of software, computers and networks, data on nearly everything is being created – intentionally and unintentionally. Contemporary technologies allow a digital approximation of the lives of people and objects to be imagined, if not created in practice. Clearly, data is stored, accessed, analysed, and classified for all sorts of reasons, and is shared and used for all sorts of purposes. Combined with a wide spectrum of political and commercial needs and practices, software technologies have led to a wide range of our activities being *monitored*. An established example is the monitoring data collected by companies to improve their sales and customer service, through loyalty cards and recommender systems. Commercial monitoring designed to understand customer journeys and service personalisation is one among many sources of international concerns about surveillance and privacy [6,7]. Such opportunities for monitoring have expanded enormously with the growth of mobile devices and smart products.

Despite the fact that monitoring phenomena are ubiquitous, and monitoring is arguably the principal source of data that drives the development of data science, the nature of monitoring has been neglected theoretically. Earlier, in [4], we posed the general questions:

What is the nature and purpose of monitoring?
Do the diverse and apparently disparate monitoring systems have anything in common?
What role does monitoring play in understanding surveillance and privacy?

In [4], we began to answer the first two questions by proposing an abstract approach to monitoring that can identify and explore structures common to different monitoring systems. By reflecting on some monitoring examples, we have isolated some fundamental conceptual components of monitoring systems. However, monitoring abounds in *many* domains of science, engineering, manufacturing, infrastructure, and environment on the one hand and commerce, healthcare, management, security, and social and financial services on the other. Thus, we are at the beginning of our programme of exploration and theory building. Our immediate aims are to develop theoretical ideas about monitoring systems and their applications, formulate precise general questions, and make comparisons and useful classifications of monitoring systems. Hopefully, our theorising will help the analysis of both technical and sociological issues to do with the third question asked above about monitoring.

The theory introduced in [4] formalises the idea that monitoring is a process that observes the behaviour of people and objects in a particular context. Monitoring involves choosing data to represent entities and behaviour, properties to observe by testing the data, and a form of storage to record the results. Monitoring is all about data. Thus, entities and their behaviours are modelled naturally by abstract data types and their observable attributes by logical languages. In this paper we focus on a particular model of monitoring in which the behaviour of people and objects in a context are modelled by streams of data, i.e., sequences of data indexed by time. To illustrate and test the theory, we apply the model to a new social context, namely: the monitoring of web usage by staff and members of an organisation or company. This monitoring example is but one of hundreds to be found in the workplace, but it is easy to appreciate and reveals features that seem to be widely applicable.

The structure of the paper is as follows. In Sect. 2 we explain the basic concepts and principles of our general approach to monitoring: context, monitoring, storage and interventions. This introduces a conceptual framework for thinking about monitoring, one which can be formalised in a number of different ways. In Sect. 3 we give one such formal way: a general model of monitoring that takes behaviours to be modelled by streams of data.

Next, we turn to case studies. In Sect. 4 we discuss aspects of monitoring in organisations and companies. In Sects. 5 and 6 we use the general stream model in Sect. 3 to develop stream models of web monitoring in organisations. In Sect. 7 we consider storage as an abstract data type. In Sect. 8 we give examples of interventions. Thus, we will present our ideas about monitoring in three forms: as informal intuitions, formal definitions, and applications in a case study. Finally, in Sect. 9, we look back on the monitoring and intervention stack and we make some remarks on necessary further developments.

We assume that the reader is familiar with the basic algebraic concepts used to model data types: *signature*, *algebra*, *expansion*, *reduct*, *congruence*, *term*, *homomorphism*, *equational theory*, *first order theory*, etc. and, indeed, their relevant abstractions such as *institutions*. Whilst all are relevant only a few will appear in this short exposition of our theory, which we will develop using algebras. We have chosen to keep our algebraic techniques very simple to focus attention on monitoring and to present it as a new topic for theoretical investigation.

2 Concepts and Principles of Monitoring and Interventions

2.1 The Approach

Our conception of monitoring is based upon the following principle:

Principle. *Monitoring is confined to the collection, evaluation and recording of observational data about the behaviour of entities. The outputs of a monitoring system are simply records of observations.*

Thus, a key idea in our analysis is that it is *only* concerned with data. Thanks to this principle, our theory of monitoring is a theory of data and we can use the theory of abstract data types for its development.

The theory is based upon the idea that monitoring is a process that observes entities – people and objects, real and virtual – confined to a narrowly defined context wherein they and their behaviour can be represented by data that can be captured, queried and evaluated. A *context* consists of (i) *entities* and certain information about them called *characteristics*, and (ii) *behaviours*. Monitoring a context begins by choosing specific *attributes* of the data to be observed and a means of making *judgements* about the attributes.

To capture commonalities of diverse domains, and increase the generality of the analysis, we separate the acquisition of the monitoring data from its use.

However, monitoring usually has a specific purpose. The records are inspected and certain properties trigger actions that may change the behaviour of the entities. We call these checks and changes *interventions*.

The components are illustrated in Fig. 1. The monitoring system and the interventions are composed exclusively of data. We can design and combine abstract data types into what we call the *monitoring and intervention stack*.

Outside the monitoring and intervention stack we allow monitoring and intervention to employ any kind of technology and practice. To complete the description of monitoring, we introduce an informal notion of a

(i) *monitoring infrastructure*, for the technological and human systems that obtain and send the data representing the behaviour of entities to a monitoring system; and
(ii) an *intervention infrastructure* for the systems that receive the information from a monitoring system with interventions and initiate various responses and actions on the entities.

The details of these infrastructures are *not* part of the theory. Let us expand on the ideas introduced above.

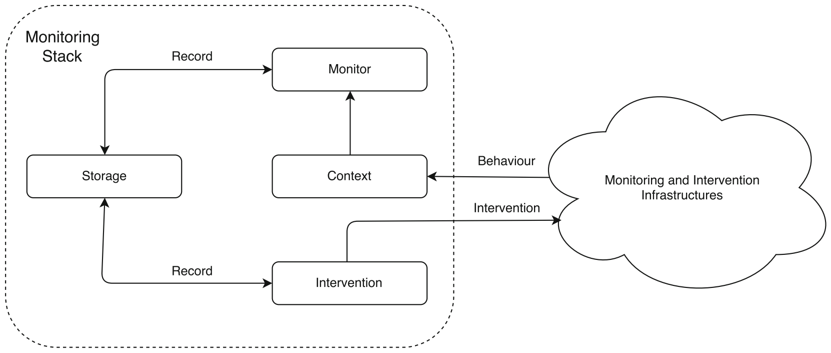


Fig. 1. The monitoring and intervention stack architecture

2.2 Conceptual Framework for Monitoring

The components that constitute the framework are these:

Context: Entities, Characteristics and Behaviour. Monitoring takes place in a context. A *context* is composed of *entities*. The entities have *characteristics* that define information that are relevant to the entities in the context. Entities have *behaviours* that can be observed. The behaviour of an entity depends upon the characteristics of the entity: characteristics are a parameter of entity behaviour.

Observation: Attributes and Judgements. Behaviours have *attributes* that can be observed. Observation involves making a query about, or testing, the behaviour of an entity for the presence of an attribute and making an evaluation. The evaluation produces a *judgement*.

In most cases the attribute will be a property that is assessed using a range of values on a scale; the evaluation will be a labelling, grading, measure, or probability. For example, physical measurements with error margins can give judgements that are bands of numerical values. Or a judgement may be a qualitative assessment based upon bands labelled metaphorically, such as the commonly used three-valued traffic light signifiers

$$\{green, amber, red\}.$$

In questionnaires and social surveys, five-valued assessments are used, such as:

$$\{strongly\ disagree, disagree, neutral, agree, strongly\ agree\}.$$

Monitoring and Records. The purpose of monitoring is to make an *observation* and a *record* of the observation. A record should contain the entity, its characteristics, the attributes observed and the judgements.

Interventions. To use the monitoring data, specific properties must be recognised in the records, noted and communicated to infrastructures *outside* the monitoring system, namely the intervention infrastructures. These communications with the outside we call *notifications*. The notification may initiate a series of physical or virtual actions that change an entity's characteristics and its behaviour.

Interventions are based on judgements. The judgements in a monitoring record are inspected: observations requiring actions are detected by *trigger conditions*. A trigger takes as input a judgement and returns a Boolean value that may lead to a notification for action. An *intervention* is a rule of the form

$$\textbf{if } trigger \textbf{ then } action \textbf{ else } do\ nothing.$$

Triggers decide what should happen. Actions change the information in the entity's characteristics.

These ideas together form a general conceptual framework for monitoring that can be developed in a number of ways. Clearly, there are different semantic models of behaviour, different logics to formalise attributes and judgements, and different models of computation to analyse monitoring and interventions – see Sect. 9.

3 Monitoring Behaviour Modelled by Streams

We suppose the behaviour of entities takes place in time. We model the behaviour of an entity over time by a *stream* of data from A,

$$\dots, a(t), \dots \in A \text{ for } t \in T,$$

where T is a set of data that mark points in time. There are a number of forms of stream behaviour, as time and data can be discrete or continuous, and be structured by orderings and topologies. Here is the formal model, component by component.

3.1 Monitoring Streams

Time. The choice of T is influential and will be called the *monitoring clock*. We choose time to be discrete and take $T = \{0, 1, 2, \ldots\}$.

Behaviour. Behaviour is characterised by some data from a set A – typically quantitative measurements, text, images, audio or video. The streams are sequences in discrete time that are infinite and always well-defined:

$$a(0), a(1), a(2), \ldots, a(t), \ldots \in A \text{ for } t \in T.$$

Thus, we define a *stream* of data to be a *total function* $a : T \to A$ mapping time points in T to data in A. The space of all behaviours is the set $[T \to A]$ of all streams.

Contexts: Entities, Characteristics and Behaviour. Let E be the set of entities and let C be the set of characteristics. We define the behaviour of an entity as a stream of data generated by the behaviour map

$$[\![-, -]\!] : E \times C \to [T \to A] \tag{1}$$

such that for entity $e \in E$, with characteristics $\chi \in C$, at time $t \in T$,

$$[\![e, \chi]\!](t) = \text{data characterising behaviour of entity } e \text{ with characteristics } \chi \text{ at time } t.$$

Observation: Attributes and Judgements. Behaviours have attributes that can be observed over time. Let *Attr* be a set of attributes of behaviours. Let J be a set of judgements; often, J is a finite set.

Whilst observation may take many forms, the act of observing the behaviour of an entity, and making an evaluation, *starts* with a map

$$Obs_0 : Attr \times [T \to A] \to J. \tag{2}$$

We will need some variations:

Individual Observation. Suppose an attribute may vary according to the entity and its characteristics, and is specified by

$$P : E \times C \to Attr. \tag{3}$$

In this case, on substituting into mapping (2), the observation map becomes

$$Obs : E \times C \times [T \to A] \to J, \tag{4}$$

which, given an entity e with characteristics χ, computes the extent or degree that $P(e, \chi)$ is a property of $a \in [T \to A]$:

$$Obs(e, \chi, a) = Obs_0(P(e, \chi), a). \tag{5}$$

Observation over intervals and at instants. Since behaviour is time dependent, attributes are likely to involve time. First, it is common to look at behaviour over, say, an interval $[t_1, t_2] \subset T$. This suggests further adaptations of mapping (2) of the form

$$Obs_0 : Attr \times [T \to A] \times T \times T \to J. \tag{6}$$

To observe the complete history of the entity up to time t we take the interval $[0, t]$. To observe behaviour at a particular time t we take the interval $[t, t]$. In both of these cases we adapt map (6) to one of this form

$$Obs_0 : Attr \times [T \to A] \times T \to J. \tag{7}$$

Secondly, in time dependent cases, the attribute to be checked may depend upon the entity and its characteristics (as above), and upon the time of inspection.

Monitoring. Now, we implement monitoring as follows. First, let

$$R = E \times C \times Attr \times J \tag{8}$$

be the set of *records*.

Individual Monitoring. If the attribute P to be observed depends upon entity e and characteristic χ then $P(e, \chi)$ is the attribute to be checked. Using mappings (1) and (5), the monitor function becomes

$$Monitor : E \times C \to R \tag{9}$$

defined by

$$Monitor(e, \chi) = (e, \chi, P(e, \chi), Obs(e, \chi, [\![e, \chi]\!])).$$

Monitoring over intervals and at instants. If the attribute P to be observed depends upon entity e and characteristic χ then using mappings (1), (5) and (6), the monitor function becomes

$$Monitor : E \times C \times T \times T \to R \tag{10}$$

defined by

$$Monitor(e, \chi, t_1, t_2) = (e, \chi, P(e, \chi), Obs(e, \chi, [\![e, \chi]\!], t_1, t_2)).$$

Monitoring Stream. Setting $t_1 = t_2 = t$, produces the mapping (7), which gives rise to a stream of records

$$monitor : E \times C \to [T \to R] \tag{11}$$

defined by

$$monitor(e, \chi)(t) = (e, \chi, P(e, \chi), Obs_0(P(e, \chi), [\![e, \chi]\!], t)).$$

3.2 Storage

The storage component is designed to collect and retain all, or some, of the records generated by monitoring. Thus, we assume that at any time the storage contains a finite list of records:

$$r_0, r_1, \ldots, r_n.$$

The index n is a function of time measured by the monitoring clock. For simplicity and brevity, we will assume that *all records generated by monitoring are stored.* The output of the monitoring component and the input of the storage component is a stream defined by map (11). It is easy to imagine cases where some filtering predicate chooses to store or not to store a record.

Clearly, there are many ways of designing a storage component, not least those of the theory of databases [5]. For our needs we may suppose there is an abstract data type for storage that comprises operations that input, organise, and output the records, and can support programs that can analyse the data in the store.

3.3 Interventions for Streams

Following the conceptual framework, interventions are based upon judgements, they do *not* involve behaviours directly, and so they are independent of the streams.

Triggers. Trigger conditions accept as input a judgement value $j \in J$, obtained as a result of the observations made by the function Obs, and outputs a truth value:

$$tc : J \to \mathbb{B}.$$

We denote the set of all trigger functions by $Trig = [J \to \mathbb{B}]$.

Actions. An action function

$$act : C \to C$$

performs an update $act(\chi)$ to the information $\chi \in C$. We denote the set of all action functions by Act.

Interventions. We use triggers and action functions to specify the intervention that results from the observation of an entity's behaviour. With both of these functions, we define an intervention of the form

$$tc \to act$$

where $(tc, act) \in Trig \times Act$. Mathematically, for $Intv = Trig \times Act$, we define the function

$$Int : R \times Intv \to E \times C \tag{12}$$

defined by

$$Int((e, \chi, P(e,\chi), j), tc \to act) = \begin{cases} (e, act(\chi)) & \text{if } tc(j) \\ (e, \chi) & \text{if } \neg tc(j). \end{cases}$$

4 Monitoring in Organisations

We now turn to organisational monitoring to illustrate the concepts of the theory.

4.1 Why Employee Monitoring?

Many people while at work are tempted to use the web for non-work purposes. Whilst most such activities are innocent and beneficial, some could be undesirable and even harmful to employees and to their organisations. For example, the use of pornographic and gambling sites would cause damage to the reputations of employees and organisations. Whether innocent or otherwise, employers are not pleased with the cultural effects of poor work ethics, or with financial losses associated with wasted time. Thus, employers have adopted surveillance technologies to monitor their employees' internet usage at work. The main aims for monitoring employee internet use during working hours are:

1. to deter unnecessary waste of contracted working hours on personal internet use;
2. to perform duty of care and protect employees from harmful exposure to the internet; and
3. to prevent any potential security breaches due to employee use of the internet.

(Compare [10].) Indeed, the use of monitoring across an increasingly wide range of behaviours is firmly established in organisational security management practices, and is commonly referred to as employee monitoring [18]. Of course, there are many issues associated with employee monitoring, especially legal and ethical. In the UK, as laid down by data protection laws, employers have the right to monitor employees' activities via different kinds of monitoring measures. These include opening mails and e-mails; using automated software to check e-mails; and checking logs of websites visited [2]. At the same time, the UK's data protection laws also set forth various rules that employers need to observe while monitoring, and the most important one is that – except in certain extreme cases – employers need to inform employees that they are monitored, what is being monitored, and why monitoring is undertaken (*ibid.*).

Of course, checking up on employees is nothing new [8]. Nevertheless, modern surveillance technologies present significant challenges because employee monitoring is now far more widespread, continuous, intense and secretive [3]. As a result, on the one hand, opponents of employee monitoring, such as labour unions, civil liberty groups, privacy advocates, and many employees themselves, have complained vehemently about employee monitoring [18]. The main charges include: increased levels of stress, decreased job satisfaction, decreased work life quality, lowered levels of customer service, creation of a hostile workplace, invasion of employee's privacy, lower morale, and unfair treatment of some employees (*ibid.*).

On the other hand, supporters of employee monitoring emphasise that employee monitoring not only increases productivity but also protects organisations from legal liability, data leakage and industrial espionage [12]. However, research on monitoring at the workplace also indicates that although a higher level of surveillance leads to more productivity on a task, it also leads to a lower level of quality of work on the task, and to a reduction of shared identity in the organisation's culture [9].

4.2 What Might Be Monitored and How?

Although the degree of employee monitoring is up to each employer, with advanced surveillance technologies, employee monitoring can be carried out very comprehensively. An employer can thoroughly monitor computer related activities of all of his/her employees down to the keystroke. Generally, the main aspects that are monitored include:

– applications;
– system settings;
– online surfing;
– connections to diverse devices and their locations;
– emails; and
– cloud activities.

The monitoring of these six aspects have objectives that are both facilitating and constraining. For applications, monitoring might be carried out to ensure appropriate versions of tools are in use, or to prevent the use of inappropriate tools, such as the Tor browser.[1] For system settings, monitoring might enhance access to peripherals (e.g., adding printers) or correct security vulnerabilities. For online surfing, monitoring might concern website visits, specific page views, downloads, and audio and video streaming. For connections to devices, monitoring might extend standards and good practices of the office to a diverse range of mobile laptops, tablets and phones. For emails, monitoring is concerned with the identity of correspondents and the content of their messages. For cloud activities, monitoring is concerned with the services used.

All these six aspects can be investigated by analysing a wide range of log files that record the activities of the system, the network and the user. These aspects can also be investigated by desktop monitoring tools [13]. Desktop monitoring is a form of monitoring that targets a specific computer and every single action taken by its user. Surveillance software can be installed, directly into the targeted computer, or remotely, to intercept signals emitted by this target computer. It allows the monitoring of a target computer's usage, both online and offline. In general, the system administrator of an organisation is responsible for observing the data obtained via desktop surveillance, who might be asked to look for particular inappropriate actions.

4.3 Organisational Web Monitoring

How does a simple case of web monitoring in an organisation fit the conceptual framework outlined in Sect. 2. Although employee monitoring is the monitoring

[1] https://www.torproject.org.

of people, it is actually the monitoring of data about people. To monitor their internet activity, the context is the behaviour of their computer accounts, and the devices they use to access them. Consider web monitoring as the real-time monitoring of an employee's online activity. A web monitoring system may have the following form:

- **Context:** Organisation/company;
- **Entities:** Employees via employee IT accounts;
- **Characteristics:** Account access permissions and status; monitoring status;
- **Behaviour:** Web activity;
- **Attributes:** Observation of sites visited classified as: *Unrestricted; Adult and Sexually Explicit; Gaming and Gambling; Personals and Dating; Proxies and Translators; and Intolerance and Hate*;
- **Record:** Staff account; data on websites visited, classified according to attribute categories;
- **Intervention:** In real time react to URL requests as follows: *allow; allow and notify employer; deny; deny and notify employer.*

Other categories that could be of interest to track are sites for Personal Email, Advertisements and Popups, Travel, P2P Downloads, or Social Networks. The categories need not have the same degree of interest for the organisation, e.g., shopping for a partner may be tolerated more than extremist politics.

5 Stream Model of Organisation Monitoring: Context

Using the general method given in Sect. 3, we construct algebraic models for monitoring web usage and data downloads.

5.1 Organisation: Entities, Identity and Characteristics

Entities. The *entities* we are monitoring are the computer accounts of people working at an organisation who require access to the internet.

Let *Empl* denote the set of employees. In many scenarios, each employee is assigned a personal account that uniquely identifies the employee and all the IT resources they access, many of which are communal. The employee's account stores or has links to a great deal of information about the employee, not all of which is available to the employee: for example, name and address; pay, insurance and tax details for employment; staff status and access permissions; professional development file; personnel file; and historical logs of recent activity.

Let *Acc* be the set of computing accounts. Define the injective function $h : \mathit{Empl} \to \mathit{Acc}$ that assigns a unique account $h(e) \in \mathit{Acc}$ to an employee $e \in \mathit{Empl}$ which allows access to the IT infrastructure. The gap between employee and account is important conceptually: in many case studies, *the objective is to monitor people but what is monitored is the behaviour of objects, namely devices and their use.*

Characteristics. In our case studies, we take the characteristics of an entity to be information about the employee's compliance with regulations. The first characteristic is *access status*, which refers to computing regulations that include or imply a classification of the content accessible via the internet in an *acceptable use policy* for their IT infrastructure. The second characteristic is the *monitoring status* specifying information about the current compliance of the employee's account.

Let τ denote the access status characteristic and μ the monitoring status. The characteristics of a monitored employee account is a pair

$$\chi = (\tau, \mu)$$

that will depend upon the account and what is monitored. Let C denote the set of all characteristics.

5.2 Web Examples

We will apply our monitoring stack to these monitoring scenarios.

Example 1: Monitoring Web Content. In this scenario, monitoring is used to determine employee compliance to the terms and conditions of their account. The access status is based on a specification that categorises web content, e.g., sites may be classified as in Sect. 4.3:

1. unrestricted,
2. adult/sexually explicit,
3. gaming and gambling,
4. personals and dating,
5. proxies and translators, and
6. intolerance and hate.

The access status defines numerical limits to the amount of web requests for content in each category. The employee's monitoring status records the number of times websites have been requested for each category. Over time, the status profiles the employee's (undesired) browsing habits.

Example 2: Monitoring Data Usage. The access status τ contains rules specifying data transfer limits applicable to downloadable content. The monitoring status μ is *green*, *amber* and *red*, respectively, corresponding to the situation where the employee has stayed within, or is close to, or has exceeded the data limit allowable for their account.

5.3 Modelling Web Behaviour

We will work through the components of the general stream model. The two monitoring contexts are completed by adding behaviour maps of the form

$$[\![-,-]\!] : Acc \times C \to [T \to A] \tag{13}$$

that deliver data from the monitoring infrastructure. The behaviour in time of account k with characteristics χ is a stream $[\![k, \chi]\!] = b$; what exactly is b? To model behaviour as a total function $b : T \to A$ we need to define the monitoring clock T and the data in the set A for the two examples.

Over time, the employee accesses the web, creating a stream of access requests. In both examples, time $T = \{0, 1, 2, \ldots, t, \ldots\}$ counts requests made by the account.

Web Content. To model web content as described in Example 1 in Sect. 5.2, we choose a finite alphabet W of symbols such that the set W^* of all finite words over W allows the formation of all possible uniform resource locators (URLs). We set $A = W^*$ and model account behaviour as a total function $b : T \to W^*$ such that $b(t) = w$ is the t^{th} webpage requested.[2]

Data Usage. Similarly, to model data usage in Example 2 in Sect. 5.2, we set $A = W^* \times \mathbb{N}$ and model behaviour as a total function $b : T \to W^* \times \mathbb{N}$ such that $b(t) = (w, n)$, where w is the web address requested and n represents the amount of data transferred as a result of the request (measured, say, in bytes).

Now, data is never without operations and our choice of data for A requires us to define a range of operations. Here is an operation to compute data usage over a specific time interval $[t_1, t_2]$, where $t_1, t_2 \in T$ and $t_1 < t_2$. Let $\pi : [T \to W^* \times \mathbb{N}] \to [T \to \mathbb{N}]$ be such that

$$\pi(b)(t) = \text{the data usage for behaviour } b \text{ at time } t.$$

Then, the aggregation operation $agg : [T \to W^* \times \mathbb{N}] \times T^2 \to \mathbb{N}$ is defined for $t_1 \leq t_2$ by

$$agg(b, t_1, t_2) = \sum_{t=t_1}^{t_2} \pi(b)(t). \tag{14}$$

6 Stream Model of Organisation Monitoring: Observation, Judgement, Monitoring

The next step is to specify the observation functions from which we derive the monitoring functions. Observation is based on a map of the form (2),

$$Obs_0 : Attr \times [T \to A] \to J,$$

and, since we have specified T and A already, to take this next step we must specify attributes Att and judgements J. We treat the two examples separately.

[2] URLs are definable by a context free grammar.

6.1 Web Content Observation

Attributes. Recalling Sects. 5.2 and 5.3, here we are to observe, classify and record web pages. Classification is the key concept: let

$$W_1, \ldots, W_n \text{ where } W_i \subseteq W^*$$

be a collection of sets that classify URLs along the lines of Sect. 5.2. Let $W_0 = W - \bigcup_{i=1}^{n} W_i$ be the unrestricted websites.

Let $l_1, \ldots, l_n$ in $\mathbb{N}$ specify the maximum number of web requests for content allowed for each category, respectively; we write $\mathbf{w} = (W_1, \ldots, W_n)$ and $\mathbf{l} = (l_1, \ldots, l_n)$ in $\mathbb{N}^n$.

These features make up the access status characteristic $\tau = (\mathbf{w}, \mathbf{l})$. The outcome of tests make up the monitoring status μ.

The *attribute* used determines compliance with the limits $\mathbf{l}$ in τ and is defined by the test

$$P_{comply}(x_1, \ldots, x_n) \equiv (x_1 < l_1) \wedge \cdots \wedge (x_n < l_n).$$

For this scenario, we choose the set

$$J = \{\mathbf{comply}, \mathbf{non\text{-}comply}\}$$

of judgements and interpret them as

- **comply**, if the employee account has conformed to its limits,
- **non-comply**, if the employee account has overstepped its account limits.

The monitoring status μ is a value from J.

To test behaviour b for the attribute, we need some operations: define the characteristic function $\chi_i : W^* \to \{0, 1\} \subset \mathbb{N}$ by

$$\chi_i(w) = \begin{cases} 1 & \text{if } w \in W_i, \\ 0 & \text{if } w \notin W_i. \end{cases}$$

Next, define the map $f : W^* \to \mathbb{N}^n$ for $w \in W^*$ by

$$f(w) = (\chi_1(w), \ldots, \chi_n(w)).$$

The function logs the number of times a website appears in each category; note that a website may fall into none, one or many categories. For an unrestricted website $w \in W_0$, $f(w) = (0, \ldots, 0)$.

The pointwise lifting of f leads to the stream operation $F : [T \to W^*] \to [T \to \mathbb{N}^n]$ defined by the equation

$$F(b)(t) = f(b(t)). \tag{15}$$

Observation and Judgement. We determine if the observed behaviour on b satisfies the attribute P_{comply} between t_1 and t_2 using the function $profile : [T \to W^*] \times T \times T \to \mathbb{N}^n$ defined using vector addition on $\mathbb{N}^n$ by

$$profile(b, t_1, t_2) = \sum_{t=t_1}^{t_2} F(b)(t), \tag{16}$$

which applies the classification and counts web requests over the interval $[t_1, t_2]$. Computing $profile(b, t_1, t_2)$, we can define the observation function:

$$Obs(P_{comply}, b, t_1, t_2) = \begin{cases} \textbf{comply} & \text{if } P_{comply}(profile(b, t_1, t_2)) = \mathbf{t}, \\ \textbf{non-comply} & \text{if } P_{comply}(profile(b, t_1, t_2)) = \mathbf{f}. \end{cases}$$

As remarked earlier, if the categories are viewed with different degrees of concern then more complicated attributes can be defined that weight the categories W_i and apply different thresholds; recall Sect. 4.3.

6.2 Data Usage Observation

Attributes. Again, recalling Sects. 5.2 and 5.3, here we are to observe, measure and record data downloads. In this scenario, both the employee's web requests and the amount of data are observed.

The access status characteristic contains a number $\tau \in \mathbb{N}$ representing the maximum amount of data that is allowed to be transferred over a period. The attributes are expressed by three tests on downloads:

$$\begin{aligned} P_{under}(x) &\equiv x < \tau - \epsilon \\ P_{near}(x) &\equiv \tau - \epsilon \leq x < \tau \\ P_{over}(x) &\equiv x \geq \tau \end{aligned}$$

where the constant $\epsilon \in \mathbb{N}$ defines an error margin around the account limit τ. Let

$$P_{usage} = (P_{under}, P_{near}, P_{over}).$$

We define the set of judgements $J = \{\textbf{green}, \textbf{amber}, \textbf{red}\}$ to specify compliance to the data limit usage as follows:

- **green** under the limit,
- **amber** near to exceeding the limit, or
- **red** over the limit.

The monitoring status μ is a value from J.

Observation and Judgement. Next, we define the *Obs* operation to observe and judge the amount of data transferred via the account over time. Using the summation operation *agg* defined by equation (14), the data usage of the account over the period $[t_1, t_2]$ is computed by $agg(b, t_1, t_2)$. Thus, we define

$$Obs(P_{usage}, b, t_1, t_2) = \begin{cases} \textbf{green} & \text{if } P_{under}(agg(b, t_1, t_2)), \\ \textbf{amber} & \text{if } P_{near}(agg(b, t_1, t_2)), \\ \textbf{red} & \text{if } P_{over}(agg(b, t_1, t_2)). \end{cases}$$

6.3 Records

The output of the monitoring component is a record of evaluating attributes over observed data. Mathematically, a record is an element

$$r = (k, \chi, P, j) \in R = Acc \times C \times Attr \times J$$

such that

- k is an employee account,
- χ is a characteristic of the employee account,
- P is a family of attributes,
- j is a judgement.

In our examples, records are created each time a web request is made, and a stream $\rho : T \to R$ of records may be obtained using map (11) in Sect. **??**.

6.4 Monitoring

Applying a general construction of the monitoring operation (9) from Sect. 3.1, we have the monitoring map

$$Monitor : Acc \times C \times T \times T \to R$$

that is defined for entity k, characteristics χ and a family $P(e, \chi)$ of attributes over $[t_1, t_2]$ by

$$Monitor(k, \chi, t_1, t_2) = (k, \chi, P(k, \chi), Obs(k, \chi, [\![e, \chi]\!], t_1, t_2)).$$

From these we can derive streams of records.

Web Content. For employee account k, characteristics $(\mathbf{w}, \mathbf{l}, \mu)$ and the family $P_{comply} = P_{comply}(k, \mathbf{w}, \mathbf{l}, \mu)$ of attributes:

$$Monitor(k, (\mathbf{w}, \mathbf{l}, \mu), t_1, t_2) =$$
$$(k, (\mathbf{w}, \mathbf{l}, \mu), P_{comply}, Obs(k, (\mathbf{w}, \mathbf{l}, \mu), [\![k, (\mathbf{w}, l, \mu)]\!], t_1, t_2)).$$

Data Usage. For employee account k, characteristics (τ, μ) and the family $P_{usage} = P_{usage}(k, \tau, \mu)$ of attributes.

$$Monitor(k, (\tau, \mu), t_1, t_2) = (k, (\tau, \mu), P_{usage}, Obs(k, (\tau, \mu), [\![k, (\tau, \mu)]\!], t_1, t_2)).$$

7 Stream Model of Organisation Monitoring: Storage

7.1 Histories, Thresholds and Queries

The records generated by monitoring all the accounts are collected and organised for retrieval by the storage component. Given the purpose of monitoring individuals, the organisation of the storage is based on accounts.

The storage is designed to preserve finite sequences

$$h = r_0, r_1, \ldots, r_s, \ldots, r_t$$

of records of the activity of an account $k \in Acc$, where r_s is the record received from the monitoring component at s within the monitoring period $[0, t]$. Each record in the list has k as its first component and such a sequence is called a monitoring *history* of the account k. The duration or length of the history h is $|h| = t + 1$. The set $History_k$ of all possible histories of the account k is the basis of the abstract data type for storage.

Also associated with an employee account k are finite sequences

$$r_{t_1}, \ldots, r_s, \ldots, r_{t_2}$$

of records over an arbitrary monitoring interval $[t_1, t_2]$ called monitoring *transcripts* for the account k, where r_s is the record received at time s from the monitoring component, for $s \in [t_1, t_2]$. Let $Transcript_k$ be the set of all transcripts of the account k. Note $History_k \subset Transcript_k$.

In monitoring all the accounts in the organisation, we can define

$$History = \bigcup_{k \in Acc} History_k \text{ and } Transcript = \bigcup_{k \in Acc} Transcript_k.$$

The storage component specification needs to support high-level queries about the records it stores, and such queries can be considered part of the monitoring process. Thinking about programs that compute queries gives us a way of testing what data and operations may be needed when specifying an abstract data type for storage. The abstract data types can be analysed formally by applying models of computation designed for abstract data types [11,14,17].

For example, consider the simple queries that return

Q1 *A transcript of the employee's account in the last two hours;*
Q2 *A list of employee accounts exceeding the web content category threshold at least once; and*
Q3 *The percentage of employee accounts who have exceeded their download limit.*

We sketch the types of data and operations required to program answers to the queries above.

7.2 Data and Operations for Storage

Database Types. At the heart of storage is some form of database DB containing histories of all the accounts. A state of the database during monitoring can be modelled in different ways, such as by a map

$$\sigma : Acc \to History \text{ such that } \sigma(k) \in History_k, \text{ for } k \in Acc.$$

In turn $History$, and $Transcript$, can be modelled as follows.

Let R_ϵ be the set R of records augmented by ϵ, a datum that denotes a *null record*. Let R^*_ϵ be the subset of the set $[T \to R_\epsilon]$ of all total maps, such that $h \in R^*_\epsilon$ has value ϵ almost always, i.e., $h : T \to R_\epsilon$ and $h(t) \neq \epsilon$ for only finitely many $t \in T$. Thus, formulating trivial conditions on the indexing, we have inclusions and embeddings:

$$History \subset Transcript \subset R^*_\epsilon \subset [T \to R_\epsilon].$$

At any moment, the number of accounts in the database σ is simply $|Acc|$.

Turning to the queries, Query Q1 involves three sorts: *employee accounts*, *transcripts*, and real-world *time*. Query Q2 uses *judgements* stored within records obtained by testing compliance with web content classifications. This query can return more than one matching employee account and so we make use of sets or lists Acc^* of accounts. The last query Q3 needs the *rational numbers* $\mathbb{Q}$ for quantitative operations; in this case, percentage.

In essence, we are sketching an abstract data type for the storage component based upon sets of data DB, Acc^*, $History$, $Transcript$, and $\mathbb{Q}$. Remember that R is comprised of four sets of data, Acc, C, $Attr$ and J, denoting employee accounts, characteristics, attributes and judgements, respectively. Other queries will create the need for other data types.

Database Operations. The basic operations on DB are about input and output. When a new record r is received from monitoring the account k, it is added to the database and the current history of account k is extended. We define $input : DB \times Acc \times R \to DB$ so that $input(\sigma, k, r)(k) \in History_k$ is updated with $input(\sigma, k, r)(k)(|\sigma(k)| + 1) = r$ where $|\sigma(k)|$ is the duration of the current history. The output operation $retrieve : DB \times Acc \to History$ is defined such that $retrieve(\sigma, k) = \sigma(k)$, the history associated with account k stored in database σ.

Transcripts are important in monitoring and can be extracted from histories by an operation $trans : DB \times Acc \times T \times T \to Transcript$ so that $trans(\sigma, k, t_1, t_2)$ is the sequence of records of account k stored for the period $[t_1, t_2]$.

Iteration on the Database. Many programs will involve searching the database for employee accounts matching a given property (e.g., to answer Query Q2). This requires a means of enumerating the accounts.

Operations on Clocks. Querying the database for observations occurring within a time frame is common; e.g., query Q1. Thus, it is necessary to have operations to support queries involving various measurements of real-world time. In our algebraic framework, we model clocks abstractly using the natural numbers $\mathbb{N}$ to count or mark events of interest. Specifically, the monitoring clock counts observations and indexes the histories and transcripts. To connect to real-world time requires operations that map between clocks. One method is to use a fast system clock to include a real-world time stamp in each record.

8 Interventions

In our framework, interventions formalise rules defined on records. They determine actions performed on characteristics, which may in turn impact on future observed behaviour. Here we give simple examples of interventions that can be applied to the models we developed in Sects. 5, 6 and 7. In practical scenarios, the intervention infrastructure involves social-technical mechanisms, such as complex hierarchical human structures within organisations (managers, committees and tribunals) that determine appropriate actions according to rules, practices and expediencies.

Web Categorisation Intervention. Consider an intervention to restrict an account's access to a category W_i of web pages when its most recent record is judged to have exceed the specified account limit l_i. Suppose this reduces the account limit so that future non-compliant behaviours are detected earlier or blocked altogether.

For the following, suppose that the database in the storage component has been queried to obtain the most recent record $r = (k, \chi, P, j)$ of a particular employee account k. The characteristic is of the form $\chi = (\tau, \mu)$ where $\tau = (\mathbf{w}, \mathbf{l})$ is the pair of web categories and their browsing limits, respectively. An intervention $tc \to a$ is applied to record r, using the function defined by (12), such that

- trigger $tc : J \to \mathbb{B}$ on judgements is defined by $tc(\textbf{non-comply}) = \mathbf{t}$ and $tc(\textbf{comply}) = \mathbf{f}$,
- action $act : C \to C$ on characteristics is defined by $act(\mathbf{w}, \mathbf{l}, \mu) = (\mathbf{w}, \mathbf{l}-\mathbf{b}, \mu')$, where $\mathbf{b} \in \mathbb{N}^n$ depends upon μ, and the status is reset to μ'.

Subtraction of $\mathbf{b}$ from $\mathbf{l}$ reduces the limits of web categories accessible by the account; if $l_i = b_i$ then the new limit is 0 and the category W_i is banned. The μ' leads to a notification to the intervention infrastucture.

Data Usage Intervention. Consider an intervention that updates a download limit. The characteristic χ contains the access status $\tau \in \mathbb{N}$, a number specifying the maximum amount of data to be transferred.

To illustrate our conceptual framework's flexibility, we extend the mathematical description of interventions to perform actions based on a collection of judgements. To this end, let J^* denote the set of all finite sequences of judgements from the set $J = \{green, amber, red\}$ and call a subset of J^* a *judgement pattern*. In the data usage scenario, consistent long term over-usage of data by an account can be expressed as a judgement pattern with many instances of *amber* and *red*; for example, the pattern could be expressed by a regular expression,

$$\mathbf{over} = (amber|red)^n,$$

for a fixed number $n \in \mathbb{N}$.

The intervention $Int : DB \times Intv \to E \times C$, mapping (12), can be extended to operate over transcripts, such as a record sequence r^* of daily observations of account k. An intervention $tc \to a$ is applied to r^* using the definition

- trigger condition $tc : J^* \to \mathbb{B}$ is defined by setting $tc(\mathbf{over}) = \mathbf{t}$ and $tc(J^* \setminus \{\mathbf{over}) = \mathbf{f}$,
- action $act : C \to C$ is defined for $b \in \mathbb{N}$ by $act(\tau, \mu) = (\tau + b, \mu')$.

This simply increases the limit set by τ by b and resets the status to μ'. There are many scenarios for which this form of intervention is relevant. For example, a customer of an internet service provider, such as a phone company, will analyse their data usage noting frequent limit breaches. These breaches can result in costly penalties or data transfer at premium rates. The customer needs to change their subscription to a tariff with improved terms and conditions for data usage, thus increasing their limit.

9 Concluding Remarks

Using a general conceptual framework for analysing monitoring, we have given a general mathematical model of monitoring based on modelling behaviour as streams of data, and applied it to monitoring web activities of members of an organisation. This is the second of our papers that aims to develop a general theory of monitoring; the first [4] introduced the conceptual framework and a simpler stream model, and applied the stream model to monitoring offenders for criminal justice jurisdictions. We know of no other attempts to establish a general theory of monitoring.

To end, we return to the general approach sketched in Sect. 2 and consider next steps in developing a theory of monitoring.

9.1 The Monitoring Stack

The monitoring and intervention stack consists of context, monitoring, storage and intervention. The core concepts of our approach informally described in Sect. 2.1 can be summarised by four signatures:

signature	**Context**
sorts	*entity*, *characteristic*, *behaviour*
operations	$[\![-,-]\!] : entity \times characteristic \to behaviour$

signature	**Monitoring**
imports	Context
sorts	*attribute*, *judgement*, *record*
operations	$obs : attributes \times behaviour \to judgement$ $monitor : entity \times characteristics \times attribute \to record$

signature	**Storage**
imports	Monitoring
sorts	*database*, *history*
operations	$input : database \times entity \times record \to database$ $retrieve : database \times entity \to history$

signature	**Intervention**
imports	Monitoring
sorts	*Intervention*
operations	$int : record \times intervention \to entity \times characteristic$

Each component can be specified as an abstract data type using signatures, equations and algebraic semantics. However, as the different organisational monitoring scenarios illustrate (not least the discussion of storage), there is a great deal more to these data types. To build the abstract data types of the monitoring stack, we need lots of operations and tests customised to applications, together with auxiliary types, operations and tests that act as a foundation for the construction; these latter can be grouped together into what we call the *platform algebra* for the stack. Some of these building blocks are generic such as algebras of infinite streams; others are made as the models develop.

9.2 Next Steps

Unsurprisingly at this early stage, the ideas in this paper require further analysis and application. Theoretically, there is much to be done on

(i) algebraic specifications of the monitoring stack to better understand the conceptual framework and stream models;
(ii) semantic models of behaviour, including formal models of time and space;
(iii) languages and logics for attribute specification and judgements;
(iv) storage models and their analytics;
(v) computability and complexity of monitoring.

In connection with (i), we have purposely simplified our presentation of the models, even avoiding signatures and equations, not to distract from the intuitions and ideas upon which the theory is based. In the matter of (ii), the notion of context seems to us simple and versatile, and capable of several formal approaches in which behaviour is analysed differently, e.g., by process algebras highlighting non-determinism and concurrency. Indeed, streams can be modelled in different ways. Time is an important component and leads to multiple clocks and streams. For (iii), the specification and evaluation of attributes in one form or another is core business for most formal design methods and will strengthen the theory and its relevance for thinking about tools for monitoring. In case (iv), the records managed by the storage component commonly meet the standard V^3 criteria for big data: volume, velocity and variety. There are many old and new technologies for effectively processing large quantities of data, including new technologies for graph databases [1]. In case (v), the computability and complexity of monitoring are important *a priori*, and will need to be developed when deploying the theory. The computational theory of data streams is rich [15,16].

There is also much more to be done on case studies. Our explorations of monitoring in criminal justice and organisations have been rewarding, helping to shape, test and refine the theory. However, each case study has many more details and variations to model, and there are huge new areas to enter, such as monitoring in manufacturing, retail, healthcare, and computer system administration, including cloud computing. As the case studies grow, the potential of these ideas to influence practice, through methods and tools, can be judged.

Our interest in monitoring was sparked by investigations into some social problems of surveillance and privacy. Our theory of monitoring needs to be expanded to analyse and classify degrees of surveillance and privacy. This would involve integrating a theory of identity for the entities.

The digital world we have created means that recording lives – professional and private; intentionally or accidentally – has become a widespread technical and social phenomena. The technologies have turned monitoring from a tool for understanding a complex problem rationally with the help of data, into an administrative and commercial addiction and an end in itself. It is the world's appetite for monitoring that drives its desire for data, the bigger the better. We believe monitoring systems to be a topic with considerable potential for general theories, diverse applications, and socio-technical insights. At this stage we are content to develop the theory for the pleasure of thinking.

Acknowledgments. These ideas were first presented publicly by one of us (JVT) at Gregynog in an invited lecture at WADT16; we thank the organisers for the invitation and colleagues attending WADT 16 for their encouraging and helpful comments. This work was partially supported by the EPSRC project *Data Release - Trust, Identity, Privacy and Security* (EP/N028139/1 and EP/N027825/1).

References

1. Angles, R.: A comparison of current graph database models. In: IEEE 28th International Conference on Data Engineering Workshops (ICDEW), pp. 171–177. IEEE (2012)

2. The National Association of Citizens Advice Bureaux: Monitoring at work (2016). www.citizensadvice.org.uk/work/rights-at-work/basic-rights-and-contracts/monitoring-at-work
3. Ford, M.: Surveillance and privacy at work (1998). www.ier.org.uk/publications/surveillance-and-privacy-work
4. Johnson, K., Tucker, J.V., Wang, V.: Monitoring and intervention: concepts and formal models. arXiv:1701.07484 (2016). https://arxiv.org/abs/1701.07484
5. Korth, H.F., Sudarshan, S., Silberschatz, A.: Database System Concepts. McGraw-Hill, New York City (2010)
6. Lyon, D.: Surveillance as Social Sorting: Privacy, Risk and Digital Discrimination. Routledge, Abingdon (2003)
7. Lyon, D.: Surveillance Studies: An Overview. Polity Press, Cambridge (2007)
8. Mishra, J., Crampton, S.: Employee monitoring: privacy in the workplace? SAM Adv. Manag. J. **63**(3), 4–11 (1998)
9. O'Donnell, A.T., Ryan, M.K., Jetten, J.: The hidden costs of surveillance for performance and helping behaviour. Group Process. Intergroup Relat. **16**, 246–256 (2012)
10. Paganini, P.: Employee monitoring, a controversial topic (2016). http://securityaffairs.co/wordpress/46814/digital-id/employee-monitoring.html
11. Stoltenberg-Hansen, V., Tucker, J.V.: Effective algebras. In: Abramsky, S., Gabbay, D., Maibaum, T. (eds.) Handbook of Logic in Computer Science. Volume IV: Semantic Modelling, pp. 357–526. Oxford University Press (1995)
12. Tavani, H.: Ethics and Technology: Controversies, Questions, and Strategies for Ethical Computing. Wiley, Hoboken (2010)
13. TechTarget: comprehensive guide to desktop monitoring tools. http://searchenterprisedesktop.techtarget.com/essentialguide/Comprehensive-guide-to-desktop-monitoring-tools
14. Tucker, J.V., Zucker, J.I.: Computable functions and semicomputable sets on many sorted algebras. In: Abramsky, S., Gabbay, D., Maibaum, T. (eds.) Handbook of Logic in Computer Science. Volume V: Logical and Algebraic Methods. Oxford University Press (2000)
15. Tucker, J.V., Zucker, J.I.: Continuity of operators on continuous and discrete time streams. Theor. Comput. Sci. **412**, 3378–3403 (2011)
16. Tucker, J.V., Zucker, J.I.: Computability of operators on continuous and discrete time streams. Computability **3**, 9–44 (2014)
17. Tucker, J.V., Zucker, J.I.: Generalizing computability to abstract algebra. In: Sommaruga, G., Strahm, T. (eds.) Turing's Revolution: The Impact of His Ideas About Computability. Birkhäuser, Springer Basel (2015)
18. Yerby, J.: Legal and ethical issues of employee monitoring. Online J. Appl. Knowl. Manag. **1**(2), 44–55 (2013)

Survey Papers

Asymmetric Combination of Logics is Functorial: A Survey

Renato Neves[1](✉), Alexandre Madeira[2], Luis S. Barbosa[2], and Manuel A. Martins[3]

[1] INESC TEC (HASLab), Universidade do Minho, Braga, Portugal
nevrenato@di.uminho.pt
[2] QuantaLab, INESC TEC (HASLab), Universidade do Minho, Braga, Portugal
amadeira@inesctec.pt, lsb@di.uminho.pt
[3] CIDMA – Department of Mathematics, Universidade de Aveiro, Aveiro, Portugal
martins@ua.pt

Abstract. Asymmetric combination of logics is a formal process that develops the characteristic features of a specific logic on top of another one. Typical examples include the development of temporal, hybrid, and probabilistic dimensions over a given base logic. These examples are surveyed in the paper under a particular perspective—that this sort of combination of logics possesses a functorial nature. Such a view gives rise to several interesting questions. They range from the problem of combining translations (between logics), to that of ensuring property preservation along the process, and the way different asymmetric combinations can be related through appropriate natural transformations.

Keywords: Institution · Hybridisation · Probabilisation
Temporalisation · Asymmetric combination

1 Introduction

1.1 Motivation and Context

It is well known that software's inherent high complexity renders formal design and analysis a difficult challenge, still largely unmet by the current engineering practices. Often, in fact, the formal specification of a non trivial software system calls for multiple logics so that specific types of requirements and design issues can be captured: if properties of data structures are typically encoded in an equational framework, behavioural issues will call for some sort of modal or temporal logic, whereas probabilistic reasoning will be required in order to predict or analyse faulty behaviour in distributed systems.

This fact explains the growing interest in the systematic combination of logics, an area whose overall aim can be summed up in a simple methodological principle: *identify the different natures of the requirements to be formalised, and combine whatever logics are suitable to handle them into a single logic for the* *whole system*. Its potential was already stressed in the eighties by Goguen and Meseguer, and the whole programme started to gain prominence in the following decade (*cf.* [3,18]).

The current paper surveys a specific type of combination of logics, called *asymmetric*, in which the characteristic features of a logic are developed on top of another one. Probably the most famous example is the process of *temporalisation* [12], in which the features of a temporal logic are added to another logic; the latter is often referred to as the *base logic* in order to distinguish the original machinery from the one added along the process. In brief, temporalisation adds a temporal dimension to the models of a given logic and syntactical machinery to suitably handle this added dimension. The *hybridisation* [20] and *probabilisation* [2] processes are more recent examples. The former develops a *hybrid* logic [1] on top of the base one whereas the latter adds probabilistic features. Other examples include *quantisation* [4] and *modalisation* [11], bringing into the picture features of quantum and modal logic, respectively.

Is there a common characterisation of these different combinations, able to provide a suitable setting to discuss their properties at a generic level? Such is the question addressed in this paper through the identification of their common *functorial nature*. This perspective structures the whole survey presented here.

Our approach is based on the theory of *institutions* [17], an abstract characterisation of logical systems that encompasses syntax, semantics, and satisfaction. Put forward by Goguen and Burstall in the late seventies, its original aim was to develop as much Computing Science as possible in a general, uniform way, independently of any particular logical system, in response to the *"population explosion among the logical systems used in Computing Science"* [17]. Since then this goal has been achieved to an extent even greater than originally thought. Indeed, institutions underlie the foundations of algebraic specification methods, and are most useful in handling and combining different sorts of logical systems. The universal character and resilience of institutions is witnessed by the wide set of logics formalised and subsequently explored within the framework. Examples go from standard classical logics, to more unconventional ones, typically capturing modern specification and programming paradigms—examples include *process algebras* [23], *temporal logics* [8], the Alloy language [25], coalgebraic logics [9], functional and imperative languages [30], among many others.

1.2 Contributions and Roadmap

Institutions are objects of a well known category **I** whose arrows are the so-called *institution comorphisms* (*cf.* [21,30]). In this setting we argue that an asymmetric combination of logics can, very often, be seen as an *endofunctor* over **I**. Three examples (*temporalisation*, *hybridisation*, and *probabilisation*) are discussed in detail, with their definitions (slightly) reworked to fit in the general picture. Such a functorial perspective has several advantages: an interesting one is the possibility to lift the combination process from logics to their translations, which allows for the characterisation of natural transformations between asymmetric combinations. Another interesting possibility is the study of adjoints, and preservation of properties such as conservativity, equivalence, and (co)limits.

We initiate this survey with a brief overview of common approaches to combination of logics, in Sect. 2. From there on, the focus is placed on asymmetric combinations and the characterisation of their functorial nature.

Thus, in Sect. 3 we recall the category of institutions **I** and revisit the three combinations of logics discussed in the paper. Then, in Sect. 4, these examples are made functorial. For the sake of simplicity and conciseness, we define an institutional notion of asymmetric combination and make, to a large extent, the necessary proofs at this level of abstraction. We stress, however, that the paper's main objective is not to introduce such a notion, but rather to survey the functorial nature of a number of asymmetric combinations and to show that the functorial perspective paves the way to several interesting mechanisms and research lines.

In the same section we study property preservation by these three (new) functors in what concerns conservativity (an important property in the validation of specifications) and the equivalence of institutions. We also discuss natural transformations between asymmetric combinations. Finally, in Sect. 5, we conclude and suggest future lines of research.

This paper assumes a basic knowledge of Category Theory. Whenever found suitable, we will omit subscripts in natural transformations and denote the underlying class of objects of a category **C** by $|\mathbf{C}|$ or just **C**. All proofs of the paper's results are detailed in [24].

2 Combination of Logics: A Brief Overview

The entry on *Combining Logics* in the *Stanford Encyclopedia of Philosophy* [7] stresses the role of Computing Science applications as a main driving force for research in obtaining new logical systems from old, integrating features and preserving properties to a reasonable extent: *"One of the main areas interested in the methods for combining logics is software specification. Certain techniques for combining logics were developed almost exclusively with the aim of applying them to this area."* The aforementioned hybridisation and temporalisation methods, for example, were originally developed with concrete applications to Computing Science in mind, but interestingly they can be more broadly understood as a specific way of combining logics at a model theoretical level.

As already mentioned, an asymmetric combination of logics develops specific features of a logic 'on top' of another one. This sort of combination was generalised by Caleiro et al. in [6], in a method called *parameterisation.* In brief, a logic is parametrised by another one if the atomic part of the former is replaced by the latter: thus, the method distinguishes a parameter to fill (the atomic part), a parametrised logic (the 'top' logic) and a parameter logic (the logic inserted within). More recently, Rasga *et al.* [29] proposed a method for importing logics by exploiting a graph-theoretic approach.

From a wider perspective, combination of logics is increasingly recognised as a relevant research domain, driven not only by philosophical enquiry on the nature of logics or strict mathematical questions, but also from applications in

Computing Science and Artificial Intelligence. The first methods appeared in the context of modal logics. This includes *fusion* of the underlying languages [33], pioneered by Fitting in a 1969 paper combining alethic and deontic modalities [13], and *product of logics* [31]. Both approaches can be characterised as *symmetric*. Product of logics, for example, amounts to pairing the Kripke semantics, *i.e.* the accessibility relations, of both logics. With a wider scope of application, *i.e.* beyond modal logics, *fibring* [14] was originally proposed by Gabbay, and contains fusion as a particular case. From a syntactic point of view the language of the resulting logic is freely generated from the signatures of the combined logics, symbols from both of them appearing intertwined in an arbitrary way.

Reference [5] offers an excellent roadmap for the several variants of fibring in the literature. A particularly relevant evolution was the work of A. Sernadas and his collaborators resorting to universal constructions from category theory to characterise different patterns of connective sharing, as documented in [32]. In the simplest case, where no constraint is imposed by sharing, fibring is the least extension of both logics over the coproduct of their signatures, which basically amounts to a coproduct of logics. This approach, usually referred to as *algebraic fibring*, makes heavy use of categorial constructions as a source of genericity to provide more general and wide applicable methods.

3 Asymmetric Combination of Logics (Institutionally)

3.1 Institutions

Let us recall the core notions of the theory of institutions and revisit the three working examples of combinations.

Definition 1. *An institution $\mathfrak{I}$ is a tuple $(Sign^{\mathfrak{I}}, Sen^{\mathfrak{I}}, Mod^{\mathfrak{I}}, (\models^{\mathfrak{I}}_{\Sigma})_{\Sigma \in |Sign^{\mathfrak{I}}|})$ where*

- *$Sign^{\mathfrak{I}}$ is a category whose objects are signatures and arrows signature morphisms.*
- *$Sen^{\mathfrak{I}} : Sign^{\mathfrak{I}} \to$ **Set**, is a functor that for each signature $\Sigma \in |Sign^{\mathfrak{I}}|$ returns a set of Σ-sentences,*
- *$Mod^{\mathfrak{I}} : (Sign^{\mathfrak{I}})^{op} \to$ **Cat**, is a functor that for each signature $\Sigma \in |Sign^{\mathfrak{I}}|$ returns a category whose objects are Σ-models and the arrows are Σ-model homomorphisms.*
- *$\models^{\mathfrak{I}}_{\Sigma} \subseteq |Mod^{\mathfrak{I}}(\Sigma)| \times Sen^{\mathfrak{I}}(\Sigma)$, is a satisfaction relation such that for each signature morphism $\varphi : \Sigma \to \Sigma'$ the following property holds*

$$Mod^{\mathfrak{I}}(\varphi)(M) \models^{\mathfrak{I}}_{\Sigma} \rho \text{ iff } M \models^{\mathfrak{I}}_{\Sigma'} Sen^{\mathfrak{I}}(\varphi)(\rho)$$

for any $M \in |Mod^{\mathfrak{I}}(\Sigma')|$, $\rho \in Sen^{\mathfrak{I}}(\Sigma)$. Diagrammatically,

$$\begin{tikzcd}
\Sigma \arrow[d, "\varphi"'] & Mod^{\mathfrak{I}}(\Sigma) \arrow[rr, dash, "\models^{\mathfrak{I}}_{\Sigma}"] & & Sen^{\mathfrak{I}}(\Sigma) \arrow[d, "Sen^{\mathfrak{I}}(\varphi)"] \\
\Sigma' & Mod^{\mathfrak{I}}(\Sigma') \arrow[u, "Mod^{\mathfrak{I}}(\varphi)"] \arrow[rr, dash, "\models^{\mathfrak{I}}_{\Sigma'}"'] & & Sen^{\mathfrak{I}}(\Sigma')
\end{tikzcd}$$

If the tuple does not necessarily respects the satisfaction condition above then we call it a pre-institution*.*

Notation 1. In the sequel we will refer to $Mod^{\mathfrak{I}}(\varphi)(M)$ as the φ-reduct of M and denote it by $M\restriction_{\varphi}$. When clear from the context, both the subscript and superscript in the satisfaction relation will be dropped.

Definition 2. *Consider two institutions* $\mathfrak{I}, \mathfrak{I}'$*. A* comorphism $(\Phi, \alpha, \beta) : \mathfrak{I} \to \mathfrak{I}'$ *is a triple such that*

- Φ*:* $Sign^{\mathfrak{I}} \to Sign^{\mathfrak{I}'}$ *is a functor,*
- α*:* $Sen^{\mathfrak{I}} \to Sen^{\mathfrak{I}'} \cdot \Phi$ *is a natural transformation,*
- β*:* $Mod^{\mathfrak{I}'} \cdot \Phi^{op} \to Mod^{\mathfrak{I}}$ *is a natural transformation*[1]*,*
- *and for any* $\Sigma \in |Sign^{\mathfrak{I}}|$*,* $M \in |Mod^{\mathfrak{I}'} \cdot \Phi^{op}(\Sigma)|$ *and* $\rho \in Sen^{\mathfrak{I}}(\Sigma)$

$$\beta_{\Sigma}(M) \models^{\mathfrak{I}}_{\Sigma} \rho \text{ iff } M \models^{\mathfrak{I}'}_{\Phi(\Sigma)} \alpha_{\Sigma}(\rho)$$

Diagrammatically, for each $\Sigma \in |Sign^{\mathfrak{I}}|$

$$\begin{tikzcd}
Mod^{\mathfrak{I}}(\Sigma) \arrow[r, dash, "\models^{\mathfrak{I}}_{\Sigma}"] & Sen^{\mathfrak{I}}(\Sigma) \arrow[d, "\alpha_{\Sigma}"] \\
Mod^{\mathfrak{I}'} \cdot \Phi^{op}(\Sigma) \arrow[u, "\beta_{\Sigma}"] \arrow[r, dash, "\models^{\mathfrak{I}'}_{\Phi(\Sigma)}"'] & Sen^{\mathfrak{I}'} \cdot \Phi(\Sigma)
\end{tikzcd}$$

Definition 3. *Let us consider two comorphisms* $(\Phi_1, \alpha_1, \beta_1) : \mathfrak{I} \to \mathfrak{I}'$*, and* $(\Phi_2, \alpha_2, \beta_2) : \mathfrak{I}' \to \mathfrak{I}''$*. Their composition* $(\Phi_2, \alpha_2, \beta_2) ; (\Phi_1, \alpha_1, \beta_1) : \mathfrak{I} \to \mathfrak{I}''$ *is defined as* $(\Phi_2, \alpha_2, \beta_2) ; (\Phi_1, \alpha_1, \beta_1) \triangleq (\Phi_2 \cdot \Phi_1,\ (\alpha_2 \circ 1_{\Phi_1}) \cdot \alpha_1,\ \beta_1 \cdot (\beta_2 \circ 1_{\Phi_1^{op}}))$ *where the white circle denotes the* Godement (horizontal) *composition of natural transformations. Thus,*

$$\begin{aligned}
&\Phi_2 \cdot \Phi_1 \ : \ Sign^{\mathfrak{I}} \to Sign^{\mathfrak{I}''}, \\
&(\alpha_2 \circ 1_{\Phi_1}) \cdot \alpha_1 \ : \ Sen^{\mathfrak{I}} \to Sen^{\mathfrak{I}''} \cdot \Phi_2 \cdot \Phi_1, \\
&\beta_1 \cdot (\beta_2 \circ 1_{\Phi_1^{op}}) \ : \ Mod^{\mathfrak{I}''} \cdot \Phi_2^{op} \cdot \Phi_1^{op} \to Mod^{\mathfrak{I}}.
\end{aligned}$$

Each institution $\mathfrak{I}$ has as the identity comorphism the triple $(1_{Sign^{\mathfrak{I}}}, 1_{Sen^{\mathfrak{I}}}, 1_{Mod^{\mathfrak{I}}})$.

As mentioned in the Introduction, institutions and respective comorphisms form a category **I**.

[1] $(\_)^{op}$ applied to a functor $F : \mathbf{C} \to \mathbf{D}$ induces a functor $F^{op} : \mathbf{C}^{op} \to \mathbf{D}^{op}$ such that for any object or arrow a in $\mathbf{C}$, $F^{op}(a) = F(a)$.

3.2 An Institutional Rendering of Asymmetric Combinations of Logics

Consider the following abstract characterisation of what is an asymmetric combination of logics. Start with arbitrary categories $Sign_1$, $Sign_2$, and two functors

$$M^{\mathcal{C}} : (Sign_1)^{op} \to \mathbf{Cat},\ \ M^{\mathcal{I}} : (Sign_2)^{op} \to \mathbf{Cat}.$$

Assume that, for each $\Delta \in |Sign_1|$, there is a functor $U_{(M^{\mathcal{C}},\Delta)} : M^{\mathcal{C}}(\Delta) \to \mathbf{Set}$. Whenever no ambiguities arise, we will drop the subscript of $U_{(M^{\mathcal{C}},\Delta)}$. Let us further assume that given a morphism $\varphi : \Delta \to \Delta'$ of $Sign_1$, the induced functor $M^{\mathcal{C}}(\varphi)$ makes the following diagram commute.

$$\begin{tikzcd} M^{\mathcal{C}}(\Delta') \arrow[rr, "M^{\mathcal{C}}(\varphi)"] \arrow[dr, "U"'] & & M^{\mathcal{C}}(\Delta) \arrow[dl, "U"] \\ & \mathbf{Set} & \end{tikzcd}$$

This leads to a functor $M^{\mathcal{C}}(M^{\mathcal{I}}) : (Sign_1 \times Sign_2)^{op} \to \mathbf{Cat}$ such that given a pair $(\Delta, \Sigma) \in Sign_1 \times Sign_2$, $M^{\mathcal{C}}(M^{\mathcal{I}})(\Delta, \Sigma)$ forms a *discrete* category whose objects are triples (S, R, m) where $R \in M^{\mathcal{C}}(\Delta)$, $U(R) = S$, and $m : S \to M^{\mathcal{I}}(\Sigma)$. Moreover, given a signature morphism $\varphi_1 \times \varphi_2 : (\Sigma, \Delta) \to (\Sigma', \Delta')$ we have $M^{\mathcal{C}}(M^{\mathcal{I}})(\varphi_1 \times \varphi_2)\ (S, R, m) \triangleq (S,\ M^{\mathcal{C}}(\varphi_1)(R),\ M^{\mathcal{I}}(\varphi_2) \cdot m)$.

Definition 4. *An asymmetric combination $\mathcal{C}$ is a tuple $(Sign^{\mathcal{C}}, Sen^{\mathcal{C}}, M^{\mathcal{C}}, \models^{\mathcal{C}})$ such that*

- *$Sign^{\mathcal{C}}$ is a category of signatures.*
- *$Sen^{\mathcal{C}}$ is a family of functions*

$$Sen^{\mathcal{C}}_{Sign} : (Sign \to \mathbf{Set}) \to (Sign^{\mathcal{C}} \times Sign \to \mathbf{Set})$$

  indexed by the categories $Sign$ in **Cat**.
- *$M^{\mathcal{C}}$ is a functor $M^{\mathcal{C}} : (Sign^{\mathcal{C}})^{op} \to \mathbf{Cat}$ as assumed above.*
- *Given functors $M^{\mathcal{I}} : Sign^{op} \to \mathbf{Cat}$, $Sen^{\mathcal{I}} : Sign \to \mathbf{Set}$, $\models^{\mathcal{C}}$ is a family of relation liftings $(\models^{\mathcal{C}}_{(\Delta,\Sigma)})_{(\Delta,\Sigma)\,\in\, Sign^{\mathcal{C}} \times Sign}$*

$$\models^{\mathcal{C}}_{(\Delta,\Sigma)} : |M^{\mathcal{I}}(\Sigma)| \times Sen^{\mathcal{I}}(\Sigma) \to |M^{\mathcal{C}}(M^{\mathcal{I}})\ (\Delta, \Sigma)| \times Sen^{\mathcal{C}}(Sen^{\mathcal{I}})(\Delta, \Sigma)$$

Given an institution $\mathcal{I}$, a pre-institution $\mathcal{CI}$, corresponding to a specific combination, is obtained as follows.

- *$Sign^{\mathcal{CI}} \triangleq Sign^{\mathcal{C}} \times Sign^{\mathcal{I}}$.*
- *$Sen^{\mathcal{CI}} \triangleq Sen^{\mathcal{C}}(Sen^{\mathcal{I}})$. We will assume that the sentences given by $Sen^{\mathcal{CI}}$ are inductively defined (*i.e. *are generated by a grammar) so that we can define recursive maps on them. Intuitively, their atoms include the sentences of the base logic.*
- *$Mod^{\mathcal{CI}} \triangleq M^{\mathcal{C}}(M^{\mathcal{I}})$.*
- *Given a signature $(\Delta, \Sigma) \in |Sign^{\mathcal{CI}}|$, $\models^{\mathcal{CI}}_{(\Delta,\Sigma)} \triangleq \models^{\mathcal{C}}_{(\Delta,\Sigma)} (\models^{\mathcal{I}}_{\Sigma})$.*

Temporalisation. We are now ready to recast the three aforementioned combinations of logics in the institutional setting. We start with temporalisation since it is the simplest of the three.

Definition 5. *Given an institution $\mathcal{I}$ the temporalisation process returns a pre-institution $\mathcal{LI} = (Sign^{\mathcal{LI}}, Sen^{\mathcal{LI}}, Mod^{\mathcal{LI}}, \models^{\mathcal{LI}})$ defined as*

- Signatures. *$Sign^{\mathcal{LI}} \triangleq Sign^{\mathcal{L}} \times Sign^{\mathcal{I}}$, where $Sign^{\mathcal{L}}$ is the one object category* 1*. Since $Sign^{\mathcal{LI}} \cong Sign^{\mathcal{I}}$, no distinction will be made, unless stated otherwise, between the two signature categories.*
- Sentences. *Given a signature $\Sigma \in |Sign^{\mathcal{LI}}|$, $Sen^{\mathcal{LI}}(\Sigma)$ is the smallest set generated by grammar*

$$\rho \ni \ \psi \mid \neg\rho \mid \rho \wedge \rho \mid X\rho \mid \rho\, U \rho$$

  where $\psi \in Sen^{\mathcal{I}}(\Sigma)$. For a signature morphism $\varphi : \Sigma \to \Sigma'$, $Sen^{\mathcal{LI}}(\varphi)$ is a function that, provided a sentence $\rho \in Sen^{\mathcal{LI}}(\Sigma)$, replaces the base sentences ψ (i.e. *elements of $Sen^{\mathcal{I}}(\Sigma)$) occurring in ρ by $Sen^{\mathcal{I}}(\varphi)(\psi)$; in symbols $Sen^{\mathcal{LI}}(\varphi)(\rho) = \rho[\psi \in Sen^{\mathcal{I}}(\Sigma) \ / \ Sen^{\mathcal{I}}(\varphi)(\psi)\ ]$ (recall that sentences are assumed to be inductively defined).*
- Models. *Given the object $\star \in |1|$, $M^{\mathcal{L}}(\star)$ is the category whose (unique) element is the pair $(\mathbb{N}, \mathrm{suc} : \mathbb{N} \to \mathbb{N})$ ($\mathbb{N}$ denotes the set of natural numbers) and U $(\mathbb{N}, \mathrm{suc} : \mathbb{N} \to \mathbb{N})$ is $\mathbb{N}$. Hence, the elements of category $Mod^{\mathcal{LI}}(\Sigma)$ are triples $(\mathbb{N}, \mathrm{suc} : \mathbb{N} \to \mathbb{N}, m)$ (often denoted by letter M) where $m : \mathbb{N} \to |Mod^{\mathcal{I}}(\Sigma)|$. We will often denote $m\,(n)$ by M_n.*
- Satisfaction. *Given a signature $\Sigma \in |Sign^{\mathcal{LI}}|$, $M \in |Mod^{\mathcal{LI}}(\Sigma)|$, $\rho \in Sen^{\mathcal{LI}}(\Sigma)$, $M \models \rho$ iff $M \models^0 \rho$ where*
  $M \models^j \psi$ *iff* $M_j \models \psi$ *for* $\psi \in Sen^{\mathcal{I}}(\Sigma)$
  $M \models^j \rho \wedge \rho'$ *iff* $M \models^j \rho$ *and* $M \models^j \rho'$
  $M \models^j \neg\rho$ *iff* $M \not\models^j \rho$
  $M \models^j X\rho$ *iff* $M \models^{j+1} \rho$
  $M \models^j \rho\, U\, \rho'$ *iff for some* $k \geq j$, $M \models^k \rho'$ *and for all* $j \leq i < k$, $M \models^i \rho$

Note that *temporalised* propositional logic coincides with the classic *linear temporal logic* (*cf.* [12]).

Theorem 1. *Temporalised $\mathcal{I}$ (*i.e. *$\mathcal{LI}$) is an institution.*

In the sequel we show that the other two asymmetric combinations enjoy the same property, which is essential for their characterisation as endofunctors. Of course, this also entails the possibility of combining a logic an arbitrary number of times, using any of these three processes.

Probabilisation. In order to handle probabilistic systems (*e.g. Markov chains*) probabilisation [2] adds a probabilistic dimension to logics. In institutional terms,

Definition 6. *Consider an arbitrary institution $\mathcal{I}$. Its probabilised version $\mathcal{PI} = (Sign^{\mathcal{PI}}, Sen^{\mathcal{PI}}, Mod^{\mathcal{PI}}, \models^{\mathcal{PI}})$ is defined as follows*

– Signatures. *$Sign^{\mathcal{PI}} \triangleq Sign^{\mathcal{P}} \times Sign^{\mathcal{I}}$, where $Sign^{\mathcal{P}}$ is the one object category* 1. *Since $Sign^{\mathcal{PI}} \cong Sign^{\mathcal{I}}$, no distinction will be made, unless stated otherwise, between the two signature categories.*
– Sentences. *For a signature $\Sigma \in |Sign^{\mathcal{PI}}|$, $Sen^{\mathcal{PI}}(\Sigma)$ is the smallest set generated by grammar*

$$\rho \ni \ t < t \mid \neg\rho \mid \rho \wedge \rho$$

for $t \in \mathrm{T}(\Sigma)$ $(\mathrm{T} : Sign^{\mathcal{PI}} \to \mathbf{Set})$. *$\mathrm{T}(\Sigma)$ is generated by grammar*

$$t \ni \ r \mid \int \psi \mid t + t \mid t \,.\, t$$

where $r \in \mathbb{R}$ is a real number, and $\psi \in Sen^{\mathcal{I}}(\Sigma)$. Also, we have

$$\begin{aligned} &Sen^{\mathcal{PI}}(\varphi)(\rho) \triangleq \rho[t \in \mathrm{T}(\Sigma) \,/\, \mathrm{T}(\varphi)(t)\,], \textit{ where}\\ &\mathrm{T}(\varphi)(t) \triangleq t[\,\psi \in Sen^{\mathcal{I}}(\Sigma) \,/\, Sen^{\mathcal{I}}(\varphi)(\psi)\,]\end{aligned}$$

– Models. *$Mod^{\mathcal{P}}(\star)$ is the discrete category whose elements are probability spaces $(S, p : 2^S \to [0,1])$. Functor U returns the carrier set. Hence, models in $Mod^{\mathcal{PI}}(\Sigma)$ are triples (S, p, m) where $m : S \to Mod^{\mathcal{I}}(\Sigma)$. For each sentence $\psi \in Sen^{\mathcal{I}}(\Sigma)$ we set $m^{-1}[\psi] \triangleq \{s \in S : m(s) \models \psi\}$.*
– Satisfaction. *Finally, given a signature $\Sigma \in |Sign^{\mathcal{PI}}|$, a model $M \in |Mod^{\mathcal{PI}}(\Sigma)|$, and $\rho \in Sen^{\mathcal{PI}}(\Sigma)$, define*

$$\begin{array}{ll@{\qquad\qquad}lll} M_r & = r & & & \\ M_{(\int\psi)} & = p(m^{-1}[\psi]) & M \models t < t' & \textit{iff} & M_t < M_{t'} \\ M_{(t+t')} & = M_t + M_{t'} & M \models \neg\rho & \textit{iff} & M \not\models \rho \\ M_{(t.t')} & = M_t \,.\, M_{t'} & M \models \rho \wedge \rho' & \textit{iff} & M \models \rho \textit{ and } M \models \rho' \end{array}$$

Theorem 2. *Probabilised* $\mathcal{I}$ (i.e. $\mathcal{PI}$) *is an institution.*

Example 1. Probabilised propositional logic *($\mathcal{P}PL$). The probabilisation of propositional logic is the following logic:*

– Signatures. *Signatures are sets of propositional symbols P.*
– Sentences. *Sentences are generated by grammar $\rho \ni \ t < t \mid \neg\rho \mid \rho \wedge \rho$ where t is a term generated by grammar $t \ni \ r \mid \int\psi \mid t + t \mid t \,.\, t$ for $r \in \mathbb{R}$ and ψ a propositional sentence.*
– Models. *Models are probability spaces equipped with a function whose domain is the set of outcomes and the codomain the universe of propositional models.*

Intuitively, $\mathcal{P}$PL offers a probabilistic 'flavour' to propositions. For instance, one may say that the probability of p holding is less than probability of q holding, $\int p < \int q$. Other examples of probabilised logics are discussed in [2].

Hybridisation. Hybridisation [20] (and its variations *e.g.* [15]) provides the foundations for handling different kinds of *reconfigurable systems* (*i.e.* computational systems that change their execution modes throughout their lifetime) in a systematic manner: in brief, the hybrid machinery relates and pinpoints the different execution modes while the base logic specifies the properties that are supposed to hold in each particular mode.

Since hybridisation was originally defined in institutional terms we will just recall here its definition but without nominal quantification, which yields an asymmetric fragment of the process. Such a fragment is adopted in [20] to define parametrised translations from hybridised institutions into first-order logic—the authors of [10] extended this work to accommodate nominal quantification as well. The same fragment is the one adopted in [19] to provide a general characterisation of equivalence and refinement for hybridised logics.

Definition 7. *Given an institution* $\mathcal{I}$, $\mathcal{HI} = (Sign^{\mathcal{HI}}, Sen^{\mathcal{HI}}, Mod^{\mathcal{HI}}, \models^{\mathcal{HI}})$ *is defined as*

- Signatures. $Sign^{\mathcal{HI}} \triangleq Sign^{\mathcal{H}} \times Sign^{\mathcal{I}}$, *where* $Sign^{\mathcal{H}}$ *is the category* **Set** $\times$ **Set** *whose objects are pairs of sets* (Nom, Λ). Nom *denotes a set of nominal symbols, and* Λ *a set of modality symbols.*
- Sentences. *For a signature* $(\Delta, \Sigma) \in |Sign^{\mathcal{HI}}|$ *(with* $\Delta = (Nom, \Lambda)$*),* $Sen^{\mathcal{HI}}(\Delta, \Sigma)$ *is the smallest set generated by grammar*

$$\rho \ni\ i \mid \psi \mid \neg\rho \mid \rho \wedge \rho \mid @_i\rho \mid \langle\lambda\rangle\rho$$

where $i \in Nom$, $\psi \in Sen^{\mathcal{I}}(\Sigma)$, $\lambda \in \Lambda$. *For a signature morphism* $\varphi_1 \times \varphi_2 : (\Delta, \Sigma) \to (\Delta', \Sigma')$, *nominals, modalities, and base sentences of* $\rho \in Sen^{\mathcal{HI}}(\Delta, \Sigma)$ *are replaced according to* $\varphi_1 \times \varphi_2$ *by* $Sen^{\mathcal{HI}}(\varphi_1 \times \varphi_2)$.
- Models. *Given a signature* $\Delta \in |Sign^{\mathcal{H}}|$, $M^{\mathcal{H}}(\Delta)$ *is the discrete category whose elements are triples* $(S, (R_i)_{i \in Nom}, (R_\lambda)_{\lambda \in \Lambda})$ *such that* $R_i \in S$, *and* $R_\lambda \subseteq S \times S$. *Functor* U *forgets the last two elements, keeping just the carrier set. For any signature morphism* $(\varphi_1, \varphi_2) : (Nom, \Lambda) \to (Nom', \Lambda')$, *we have* $M^{\mathcal{H}}(\varphi_1, \varphi_2)(S, (R'_i)_{i \in Nom'}, (R'_\lambda)_{\lambda \in \Lambda'}) \triangleq (S, (R_i)_{i \in Nom}, (R_\lambda)_{\lambda \in \Lambda})$, *where*

$$R_i = R'_{\varphi_1(i)} \text{ and } R_\lambda = R'_{\varphi_2(\lambda)}$$

- Satisfaction. *Given* $(\Delta, \Sigma) \in |Sign^{\mathcal{HI}}|$, *a model* $M \in |Mod^{\mathcal{HI}}(\Delta, \Sigma)|$ *and a sentence* $\rho \in Sen^{\mathcal{HI}}(\Sigma)$, *the satisfaction relation is defined as*

$$M \models \rho \text{ iff } M \models^w \rho \text{ forall } w \in S$$

where

$M \models^w i$ *iff* $R_i = w$ *for* $i \in Nom$
$M \models^w \psi$ *iff* $m(w) \models \psi$ *for* $\psi \in Sen^{\mathcal{I}}(\Sigma)$
$M \models^w \neg\rho$ *iff* $M \not\models^w \rho$
$M \models^w \rho \wedge \rho'$ *iff* $M \models^w \rho$ *and* $M \models^w \rho'$
$M \models^w @_i\rho$ *iff* $M \models^{R_i} \rho$
$M \models^w \langle\lambda\rangle\rho$ *iff there is some* $w' \in W$ *such that* $(w, w') \in R_\lambda$ *and* $M \models^{w'} \rho$

The proof that, for any institution $\mathcal{I}$, hybridisation yields another institution is given in reference [20].

Example 2. Hybridised propositional logic ($\mathcal{H}PL$). *Hybridisation of propositional logic returns the following logic.*

- Signatures. *Signatures are triples (Nom, Λ, P) where Nom is a set of nominal symbols, Λ a set of modality symbols, and P a set of propositional symbols.*
- Sentences. *Sentences are generated by grammar*

$$\rho \ni \; i \mid \psi \mid \neg\rho \mid \rho \wedge \rho \mid @_i\rho \mid \langle\lambda\rangle\rho$$

where i is a nominal, λ is a modality, and ψ a propositional sentence. Note that we have two levels of Boolean connectives: the ones from propositional logic, and the ones introduced by hybridisation. One can, however, 'collapse' them since they semantically coincide.
- Models. *Models are triples (W, R, m) such that W defines the set of worlds, R describes the transitions between worlds and names states. Moreover each world $w \in W$ points to a propositional model $m(w)$.*

4 Asymmetric Combinations of Logics as Functors

4.1 Lifting Comorphisms

In the previous section three combinations of logics were revisited under the light of the theory of institutions. We intend now to discuss them as translations between logics. We will do this at the level of the abstract definition of a combination of logics given above, leading thus to more powerful results, applicable not only to the three combinations discussed, but also to any other fitting the characterisation.

Formally, given a comorphism $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ a combination process maps (Φ, α, β) into $\mathcal{C}(\Phi, \alpha, \beta) : \mathcal{C}\mathcal{I} \to \mathcal{C}\mathcal{I}'$. The strategy for such a lifting is simple: when transforming signatures, sentences or models, we keep the top level structure and change the bottom level according to the base comorphism. Thus,

Definition 8. *A comorphism $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ is lifted to a mapping $(\mathcal{C}\Phi, \mathcal{C}\alpha, \mathcal{C}\beta) : \mathcal{C}\mathcal{I} \to \mathcal{C}\mathcal{I}'$ as follows:*

- Signatures. $\mathcal{C}\Phi : Sign^{\mathcal{C}\mathcal{I}} \to Sign^{\mathcal{C}\mathcal{I}'}$,

$$\mathcal{C}\Phi \triangleq 1_{Sign^{\mathcal{C}}} \times \Phi.$$

- Sentences. $\mathcal{C}\alpha : Sen^{\mathcal{C}\mathcal{I}} \to Sen^{\mathcal{C}\mathcal{I}'} \cdot \mathcal{C}\Phi$,

$$(\mathcal{C}\alpha)_{(\Delta,\Sigma)}(\rho) \triangleq \rho\,[\,\psi \in Sen^{\mathcal{I}}(\Sigma) \,/\, \alpha_{\Sigma}(\psi)\,],$$

for any $(\Delta, \Sigma) \in |Sign^{\mathcal{C}\mathcal{I}}|$.

– Models. $\mathcal{C}\beta : Mod^{\mathcal{C}\mathcal{I}'} \cdot \mathcal{C}\Phi^{op} \to Mod^{\mathcal{C}\mathcal{I}}$,

$$(\mathcal{C}\beta)_{(\Delta,\Sigma)} \triangleq id \times id \times (\beta_{\Sigma}\cdot),$$

for any $(\Delta, \Sigma) \in |Sign^{\mathcal{C}\mathcal{I}}|$.

Clearly, $\mathcal{C}\Phi$ is a functor and both $\mathcal{C}\alpha$, and $\mathcal{C}\beta$ are natural transformations.

Lemma 1. *The lifting process, as defined above, preserves identities and distributes over composition.*

To conclude that the three combinations are endofunctors one step still remains: to show that the lifted arrows are comorphisms. This, however, entails the need to inspect each specific combination on its own, as they all lift the satisfaction relation in different ways. Certainly a fully generic definition would be an interesting result. However, this turned out to be a surprisingly complex issue, which furthermore is not essential for the message that we want this paper to convey.

Theorem 3. *If* (Φ, α, β) *is a comorphism then, for any of the three combinations* $\mathcal{C}$ *discussed above,* $\mathcal{C}(\Phi, \alpha, \beta)$ *is a comorphism as well.*

4.2 Property Preservation (Conservativity and Equivalence)

The characterisation of asymmetric combinations as endofunctors over the category of institutions **I** provides a sound basis for the study of property preservation by them. Such a study is illustrated in this section in which it is shown that temporalisation, probabilisation, and hybridisation preserve conservativity and equivalence. We start with the former case.

In Computing Science a main reason to study under what conditions a logic may be translated into another is to seek for the existence of (better) computational proof support. In the institutional setting, suitable translations are often defined by comorphisms, which in many cases should obey the following condition: whenever completeness is required, *i.e.* whenever one demands the validation of the specification against all possible scenarios (models), then the comorphisms involved must be conservative. Formally,

Definition 9. *A comorphism* (Φ, α, β) *is conservative whenever, for each signature* $\Sigma \in |Sign^{\mathcal{I}}|$, β_{Σ} *is surjective on objects.*

Let us describe in more detail the relevance of conservativity for validation. Recall the satisfaction condition placed upon comorphisms. For a signature $\Sigma \in |Sign^{\mathcal{I}}|$, $M \in |Mod^{\mathcal{I}'} \cdot \Phi^{op}(\Sigma)|$, and $\rho \in Sen^{\mathcal{I}}(\Sigma)$ we have $\beta_{\Sigma}(M) \models^{\mathcal{I}}_{\Sigma} \rho$ iff $M \models^{\mathcal{I}'}_{\Phi(\Sigma)} \alpha_{\Sigma}(\rho)$. Graphically, for each $\Sigma \in |Sign^{\mathcal{I}}|$

$$\begin{tikzcd}
Mod^{\mathcal{I}}(\Sigma) \arrow[r, dash, "\models^{\mathcal{I}}_{\Sigma}"] & Sen^{\mathcal{I}}(\Sigma) \arrow[d, "\alpha_{\Sigma}"] \\
Mod^{\mathcal{I}'} \cdot \Phi^{op}(\Sigma) \arrow[u, "\beta_{\Sigma}"] \arrow[r, dash, "\models^{\mathcal{I}'}_{\Phi(\Sigma)}"'] & Sen^{\mathcal{I}'} \cdot \Phi(\Sigma)
\end{tikzcd}$$

Suppose we want to verify that a sentence $\rho \in Sen^{\mathcal{I}}(\Sigma)$ is satisfied by all models $M \in |Mod^{\mathcal{I}}(\Sigma)|$. For this we resort to the comorphism by translating the sentence (through α) into the target logic. The satisfaction condition, once verified, ensures that if the sentence is satisfied by all models there, then all models in the image of β_Σ will satisfy the original sentence. Of course, if β_Σ is surjective on objects its image will coincide with $|Mod^{\mathcal{I}}(\Sigma)|$, thus proving that the original sentence is satisfied by all models in $|Mod^{\mathcal{I}}(\Sigma)|$.

Theorem 4. *A lifted conservative comorphism is still conservative.*

Next we show that the application of temporalisation, probabilisation, and hybridisation to two equivalent logics yields again two equivalent logics. First, recall the definition of equivalence of categories.

Definition 10. *Two categories* $\mathbf{C}, \mathbf{D}$ *are equivalent if there are two functors* $F : \mathbf{C} \to \mathbf{D}, G : \mathbf{D} \to \mathbf{C}$ *and two natural isomorphisms* $\epsilon : FG \to 1_{\mathbf{D}}$, $\eta : 1_{\mathbf{C}} \to GF$. *In these circumstances, an equivalence of categories,* G *(resp.* F*) is the inverse up to isomorphism of* F *(resp.* G*)*

Definition 11. *A comorphism* (Φ, α, β) *is an equivalence of institutions if the following conditions hold.*

- Signatures. Φ *forms an equivalence of categories.*
- Sentences. α *has an inverse up to semantical equivalence,* i.e.*a natural transformation* $\alpha^{-1} : Sen^{\mathcal{I}'} \cdot \Phi \to Sen^{\mathcal{I}}$ *such that for any sentence* $\rho \in Sen^{\mathcal{I}}(\Sigma)$,

$$(\alpha^{-1} \cdot \alpha)(\rho) \models \rho, \quad \rho \models (\alpha^{-1} \cdot \alpha)(\rho)$$

  or more concisely, $(\alpha^{-1} \cdot \alpha)(\rho) \models\mid \rho$.
  Moreover, for any sentence $\rho \in Sen^{\mathcal{I}'} \cdot \Phi(\Sigma)$, $(\alpha \cdot \alpha^{-1})(\rho) \models\mid \rho$.
- Models. β *has an inverse up to isomorphism,* i.e., *a natural transformation* β^{-1} *such that for any* $\Sigma \in |Sign^{\mathcal{I}}|$, *functor* β^{-1}_Σ *is the inverse up to isomorphism of* β_Σ.

More about equivalence of institutions can be found in *e.g.* document [21].

Theorem 5. *A lifted equivalence of institutions is still an equivalence of institutions.*

4.3 Natural Transformations

We consider now natural transformations between asymmetric combinations of logics, which seem to fit nicely into the picture: while lifted comorphisms map the bottom level and keep the top one, such natural transformations map the top and keep the bottom. For example, take a natural transformation $\tau : \mathcal{L} \to \mathcal{H}$. It is clear that each institution $\mathcal{I}$, induces a comorphism $\tau_{\mathcal{I}} : \mathcal{L}\mathcal{I} \to \mathcal{H}\mathcal{I}$. Furthermore, naturality expresses the commutativity of the diagram below

$$\begin{tikzcd}
\mathcal{L}\mathcal{I} \arrow[r, "\mathcal{L}(\Phi,\alpha,\beta)"] \arrow[d, "\tau_{\mathcal{I}}"'] & \mathcal{L}\mathcal{I}' \arrow[d, "\tau_{\mathcal{I}'}"] \\
\mathcal{H}\mathcal{I} \arrow[r, "\mathcal{H}(\Phi,\alpha,\beta)"'] & \mathcal{H}\mathcal{I}'
\end{tikzcd}$$

for each comorphism (Φ, α, β). This means that when translating a logic whose levels are both mapped by a composition of natural transformations and lifted comorphisms, it does not matter which one of the top or bottom levels is taken first.

Let us illustrate this construction through the natural transformation $\tau : \mathcal{L} \to \mathcal{H}$, which relates temporalisation to hybridisation. We will, for now, disregard the *until* (U) constructor associated with $\mathcal{L}$, in order to keep the construction simple. First consider a signature $N \in |Sign^{\mathcal{H}}|$ such that $N \triangleq (\{Init\}, \{After, After^{\star}, Next\})$. Then for any signature $(N, \Sigma) \in |Sign^{\mathcal{H}I}|$ define the full subcategory of $Mod^{\mathcal{H}\mathcal{I}}(N, \Sigma)$ (denoted in the sequel by $M^{\mathcal{N}\mathcal{I}}(N, \Sigma)$) whose objects are triples (S, R, m) subjected to the following rules:

$$\begin{aligned}
&S = \mathbb{N} && \\
&R_{Init} = 0 && (a, b) \in R_{After} \text{iff}\, a < b \\
&(a, b) \in R_{Next} \,\text{iff}\, b = \text{suc}(a) && (a, b) \in R_{After^{\star}} \,\text{iff}\, a \leq b.
\end{aligned}$$

Definition 12. *Given an institution $\mathcal{I}$, define an arrow $\tau_{\mathcal{I}} = (\tau_{\mathcal{I}}\Phi, \tau_{\mathcal{I}}\alpha, \tau_{\mathcal{I}}\beta)$ where*

– Signatures. *$\tau\Phi : Sign^{\mathcal{L}\mathcal{I}} \to Sign^{\mathcal{H}\mathcal{I}}$ is a functor such that $\tau\Phi(\Sigma) \triangleq (N, \Sigma)$ and, for any signature morphism $\varphi : \Sigma \to \Sigma'$*

$$\tau\Phi\,(\varphi) : (N,\, \Sigma) \to (N,\, \Sigma'), \quad \tau\Phi\,(\varphi) \triangleq id \times \varphi$$

– Sentences. *Given a signature $\Sigma \in |Sign^{\mathcal{L}\mathcal{I}}|$, $\tau\alpha : Sen^{\mathcal{L}\mathcal{I}}(\Sigma) \to Sen^{\mathcal{H}\mathcal{I}} \cdot \tau\Phi(\Sigma)$ is a function such that $\tau\alpha(\rho) \triangleq @_{Init}\sigma(\rho)$ where*

$$\begin{aligned}
&\sigma(\psi) = \psi, \text{ for } \psi \in Sen^{\mathcal{I}}(\Sigma) && \sigma(\rho \wedge \rho') = \sigma(\rho) \wedge \sigma(\rho') \\
&\sigma(\neg\rho) = \neg\sigma(\rho) && \sigma(X\rho) = [Next]\, \sigma(\rho)
\end{aligned}$$

The proof that $\tau\alpha$ is a natural transformation follows through routine calculation.

– *Finally, given a signature $\Sigma \in |Sign^{\mathcal{L}\mathcal{I}}|$, arrow $\tau\beta : Mod^{\mathcal{H}\mathcal{I}} \cdot (\tau\Phi)^{op} \to Mod^{\mathcal{L}\mathcal{I}}$ is a functor such that*

$$\tau\beta\,(S, R, m) \triangleq (\mathbb{N}, \text{suc} : \mathbb{N} \to \mathbb{N}, m)$$

Clearly, $\tau\beta$ is a natural transformation.

Theorem 6. *$\tau : \mathcal{L} \to \mathcal{H}$ forms a natural transformation whenever $Mod^{\mathcal{H}\mathcal{I}}$ (for any institution $\mathcal{I}$) is equal to $Mod^{\mathcal{N}\mathcal{I}}$.*

In order to include the *until* constructor we need to add *nominal quantification* to hybridisation, which would yield the translation

$$\sigma(\rho U \rho') = \exists x \,.\, \langle After^{\star} \rangle(\, x \wedge \sigma(\rho')) \wedge [After^{\star}](\langle After \rangle x \Rightarrow \sigma(\rho))$$

Actually, the proof that hybridisation with nominal quantification is also an endofunctor (and the satisfaction condition for *until* associated with τ holds) boils down to a routine calculation. This means that the theorem above can be replicated, taking care of the *until* operator, in a straightforward manner.

5 Conclusions and Future Work

Asymmetric combination of logics is a promising tool for the (formal) development of complex, heterogeneous software systems. This justifies their study at an abstract level, paving the way to general results on, for example, property preservation along the combination process. Often such a study has been made on a case-by-case basis *e.g.* [11,27,28]. This paper, on the other hand, surveys a more general, functorial perspective using three different asymmetric combinations of logics as case-studies. In particular, it provided their characterisation as endofunctors over the category of institutions by showing how to lift comorphisms and proving that the lifted arrows obey the functorial laws. This made clear that not only logics, but also their translations can be combined.

The development of an institutional, abstract notion of asymmetric combination of logics proposed in the paper, hints at a set of directions for future research. For example, we saw at the abstract level that conservativity (an important property for safely ‘borrowing’ a theorem prover) and equivalence are preserved by combination. However, a full study is still to be done in what regards preservation of (co)limits, *e.g.* to discuss whether *the combination of the product of two logics is equivalent to the product of their respective combinations.*

Another research direction was set by J. Goguen in his Categorial Manifest [16]: *“if you have found an interesting functor, you might be well advised to investigate its adjoints”.* We studied natural transformations between such functors and showed that they nicely complement the lifting of comorphisms: while the latter maps the bottom level and keeps the top one, the former maps the top and keeps the bottom. We gave an example of a natural transformation between temporalisation and hybridisation, but others deserve to be studied as well. For example, in document [20] it is shown how, given a comorphism from an institution $\mathcal{I}$ to FOL, a comorphism from $\mathcal{H}\mathcal{I}$ to FOL can be obtained. More generally, the current paper shows that comorphisms can be built by lifting the original comorphism and then composing it with the ‘flat’ natural transformation $E : \mathcal{C} \to 1_{\mathcal{I}}$ (whenever it exists). Diagrammatically,

$$\begin{tikzcd}
\mathcal{I} \arrow[r, "(\Phi{,}\alpha{,}\beta)"] \arrow[d, squiggly, "\mathcal{C}"'] & \mathcal{I}' \arrow[d, squiggly, "\mathcal{C}"] \\
\mathcal{C}\mathcal{I} \arrow[r, "\mathcal{C}(\Phi{,}\alpha{,}\beta)"'] & \mathcal{C}\mathcal{I}' \arrow[u, bend right=60, "E"']
\end{tikzcd}$$

On a more speculative note, the perspective taken in this paper also suggests to look at 'trivial' asymmetric combinations. For example, it is straightforward to define *identisation*, in which the added layer has a trivial structure, but also *trivialisation* ($\mathcal{T}$), which turns a logic into the trivial one (technically, the initial object in the category **I** of institutions). The latter case implies that there is a (unique) natural transformation $\mathcal{T} \rightarrow \mathcal{C}$ to any combination $\mathcal{C}$.

From a pragmatic point of view, the incorporation of these ideas into the Hets platform [22] paves the way for its effective use in Software Engineering. Hets is often described as a "motherboard" of logics where different "expansion cards" can be plugged in. These refer to individual logics (with their particular analysers and proof tools) as well as to logic translations. To make them compatible, logics are formalised as institutions and translations as comorphisms. Therefore Hets provides an interesting setting for the implementation of the theory developed in this paper. Again, a specific case—that of *hybridisation*—was already implemented in the Hets platform [26].

Acknowledgments. This work is financed by the ERDF - European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within projects POCI-01-0145-FEDER-016692 and UID/MAT/04106/2013. Further support was provided by Norte Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement through the ERFD in the context of project NORTE-01-0145-FEDER-000037. Renato Neves was also sponsored by FCT grant SFRH/BD/52234/2013, and Alexandre Madeira by FCT grant SFRH/BPD/103004/2014.

References

1. Areces, C., ten Cate, B.: Hybrid logics. In: Blackburn, P., Wolter, F., van Benthem, J. (eds.) Handbook of Modal Logics. Elsevier, Amsterdam (2006)
2. Baltazar, P.: Probabilization of logics: completeness and decidability. Log. Univers. **7**(4), 403–440 (2013)
3. Blackburn, P., de Rijke, M.: Why combine logics? Stud. Logica **59**(1), 5–27 (1997)
4. Caleiro, C., Mateus, P., Sernadas, A., Sernadas, C.: Quantum institutions. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 50–64. Springer, Heidelberg (2006). https://doi.org/10.1007/11780274_4
5. Caleiro, C., Sernadas, A., Sernadas, C.: Fibring logics: past, present and future. In: Artëmov, S.N., Barringer, H., d'Avila Garcez, A.S., Lamb, L.C., Woods, J. (eds.) We Will Show Them! Essays in Honour of Dov Gabbay, vol. 1, pp. 363–388. College Publications (2005)

6. Caleiro, C., Sernadas, C., Sernadas, A.: Parameterisation of logics. In: Fiadeiro, J.L. (ed.) WADT 1998. LNCS, vol. 1589, pp. 48–63. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48483-3_4
7. Carnielli, W., Coniglio, M.E.: Combining logics. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy, Winter 2011 edn. (2011)
8. Cengarle, M.V.: The temporal logic institution. Technical report 9805, Ludwig-Maximilians-Universität München, Institut für Informatik, November 1998
9. Cîrstea, C.: An institution of modal logics for coalgebras. J. Log. Algebraic Program. **67**(1–2), 87–113 (2006)
10. Diaconescu, R., Madeira, A.: Encoding hybridized institutions into first-order logic. Math. Struct. Comput. Sci. **26**(5), 745–788 (2016)
11. Diaconescu, R., Stefaneas, P.: Ultraproducts and possible worlds semantics in institutions. Theoret. Comput. Sci. **379**(1–2), 210–230 (2007)
12. Finger, M., Gabbay, D.: Adding a temporal dimension to a logic system. J. Logic Lang. Inform. **1**(3), 203–233 (1992)
13. Fitting, M.: Logics with several modal operators. Theoria **35**, 259–266 (1969)
14. Gabbay, D.: Fibred semantics and the weaving of logics: part 1. J. Symb. Log. **61**(4), 1057–1120 (1996)
15. Găină, D.: Birkhoff style calculi for hybrid logics. Formal Asp. Comput. **29**, 1–28 (2017)
16. Goguen, J.A.: A categorical manifesto. Math. Struct. Comput. Sci. **1**(1), 49–67 (1991)
17. Goguen, J.A., Burstall, R.M.: Institutions: abstract model theory for specification and programming. J. ACM **39**, 95–146 (1992)
18. Goguen, J.A., Meseguer, J.: Models and equality for logical programming. In: Ehrig, H., Kowalski, R., Levi, G., Montanari, U. (eds.) TAPSOFT 1987. LNCS, vol. 250, pp. 1–22. Springer, Heidelberg (1987). https://doi.org/10.1007/BFb0014969
19. Madeira, A., Martins, M.A., Barbosa, L.S., Hennicker, R.: Refinement in hybridised institutions. Formal Asp. Comput. **27**(2), 375–395 (2015)
20. Martins, M.A., Madeira, A., Diaconescu, R., Barbosa, L.S.: Hybridization of institutions. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 283–297. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22944-2_20
21. Mossakowski, T., Goguen, J., Diaconescu, R., Tarlecki, A.: What is a logic? In: Beziau, J.Y. (ed.) Logica Universalis, pp. 111–133. Birkhäuser Basel (2007)
22. Mossakowski, T., Maeder, C., Lüttich, K.: The heterogeneous tool set, Hets. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_40
23. Mossakowski, T., Roggenbach, M.: Structured CSP – a process algebra as an institution. In: Fiadeiro, J.L., Schobbens, P.-Y. (eds.) WADT 2006. LNCS, vol. 4409, pp. 92–110. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71998-4_6
24. Neves, R., Madeira, A., Barbosa, L.S., Martins, M.A.: Asymmetric combination of logics is functorial: a survey (extended version). arXiv preprint arXiv:1611.04170 (2017)
25. Neves, R., Madeira, A., Martins, M., Barbosa, L.: An institution for alloy and its translation to second-order logic. In: Bouabana-Tebibel, T., Rubin, S.H. (eds.) Integration of Reusable Systems. AISC, vol. 263, pp. 45–75. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04717-1_3

26. Neves, R., Madeira, A., Martins, M.A., Barbosa, L.S.: Hybridisation at work. In: Heckel, R., Milius, S. (eds.) CALCO 2013. LNCS, vol. 8089, pp. 340–345. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40206-7_28
27. Neves, R., Madeira, A., Martins, M.A., Barbosa, L.S.: Proof theory for hybrid(ised) logics. Sci. Comput. Program. **126**, 73–93 (2016)
28. Neves, R., Martins, M.A., Barbosa, L.S.: Completeness and decidability results for hybrid(ised) logics. In: Braga, C., Martí-Oliet, N. (eds.) SBMF 2014. LNCS, vol. 8941, pp. 146–161. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15075-8_10
29. Rasga, J., Sernadas, A., Sernadas, C.: Importing logics: soundness and completeness preservation. Stud. Logica **101**(1), 117–155 (2013)
30. Sannella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. EATCS Monographs on Theoretical Computer Science. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-17336-3
31. Segerberg, K.: Two-dimensional modal logic. J. Philos. Log. **2**(1), 77–96 (1973)
32. Sernadas, A., Sernadas, C., Caleiro, C.: Fibring of logics as a categorial construction. J. Log. Comput. **9**(2), 149–179 (1999)
33. Thomason, R.H.: Combinations of tense and modality. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic: Volume II: Extensions of Classical Logic, pp. 135–165. Reidel, Dordrecht (1984)

Algebraic Model Management: A Survey

Patrick Schultz[1], David I. Spivak[1], and Ryan Wisnesky[2(✉)]

[1] Massachusetts Institute of Technology, Cambridge, MA, USA
[2] Categorical Informatics, Inc., Cambridge, MA, USA
ryan@catinf.com

Abstract. We survey the field of model management and describe a new model management approach based on algebraic specification.

Keywords: Model management · Algebraic databases
Data integration · Functorial data migration · Category theory
Algebraic specification

1 Introduction

In this paper we survey the field of *model management* and describe a new model management approach based on techniques from the field of *algebraic specification*, with the hope of establishing an interlingua between the two fields. By "model management" we mean "meta-data intensive" database management in the sense of Bernstein & Melnik [4], which we define in Sect. 2. By "a new algebraic model management approach" we mean our particular way [19,20] of specifying database schemas and instances using algebraic (equational) theories.

We first noticed a connection between model management and algebraic specification while investigating applications of *category theory* [3] to *data integration* [5]. These investigations are described in [19,20], and we present no substantial new results in this paper. We assume readers have basic proficiency with category theory [3], algebraic specification [17], and SQL.

Outline. In Sect. 2 we describe the traditional approach to model management and in Sect. 3 we describe our algebraic approach. Also in Sect. 3 we describe the open-source AQL (Algebraic Query Language) tool, available for download at http://categoricaldata.net/aql.html, which implements our approach in software. We conclude in Sect. 4 by comparing our approach with the traditional approach.

2 Model Management

To quote from Melnik [16]:

> Many challenging problems facing information systems engineering involve the manipulation of complex metadata artifacts, or *models*, such as

database schemas, interface specifications, or object diagrams, and mappings between models. The applications that solve metadata manipulation problems are complex and hard to build. The goal of generic model management is to reduce the amount of programming needed to develop such applications by providing a database infrastructure in which a set of high-level algebraic operators, such as Match, Merge, and Compose, are applied to models and mappings as a whole rather than to their individual building blocks.

In the paragraph above the word “model” is defined to mean a metadata artifact such as a schema, which conflicts with the definition of the word “model” as a structure satisfying a theory. In this paper, we use the phrase “model management” to mean the field identified above, and use the word “model” to mean a structure satisfying a theory.

Today model management is a large sub-field of information management with a research literature containing hundreds of published articles [4]. There is a consensus in that literature [4] that model management is concerned with at least the problems described in the next sections.

2.1 Schema Mapping

Given two database schemas S and T, the *schema mapping* problem [7] is to construct a “mapping” $F : S \to T$ that captures some user-specified relationship between S and T. Different model management systems use different notions of schema, including SQL, XML, and RDF [4]. The most common mapping formalism studied in the literature is that of “embedded dependencies” (EDs) [5]: formulae in a fragment of first-order logic with useful computational properties.

We will use SQL schemas and EDs in our examples in this section. Consider the following SQL schema S, consisting of two tables connected by a foreign key:

```
CREATE TABLE N2(ID INT PRIMARY KEY, age INT)

CREATE TABLE N1(ID INT PRIMARY KEY, name STRING, salary INT,
     f INT FOREIGN KEY REFERENCES N2(ID))
```

and the following SQL schema T, consisting of one table:

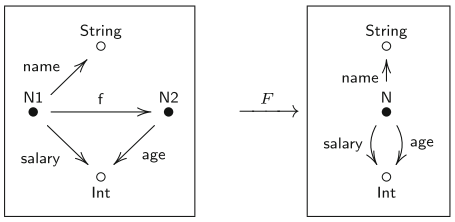
```
CREATE TABLE N(ID INT PRIMARY KEY, age INT,
     name STRING, salary INT).
```

These two SQL schemas are displayed graphically in Fig. 1.

An example schema mapping $F : S \to T$ expressing that the target table N is the join of source tables $\mathsf{N1}$ and $\mathsf{N2}$ along the column f is:

$$\forall id_1, id_2, a, n, s.\ \mathsf{N1}(id_1, n, s, id_2) \wedge \mathsf{N2}(id_2, a) \to \mathsf{N}(id_1, a, n, s).$$

Two instances satisfying the above ED are shown in Fig. 1. In general, many EDs can map between two SQL schemas.



N1

ID	name	salary	f
1	Alice	\$100	1
2	Bob	\$250	2
3	Sue	\$300	3

N2

ID	age
1	20
2	20
3	30

$$\begin{tikzcd} \mathsf{N1},\mathsf{N2} \arrow[r, shift left, "{[\![\Pi_F]\!],[\![\Sigma_F]\!]}"'] & \mathsf{N} \arrow[l, shift left, "{[\![\Delta_F]\!]}"'] \end{tikzcd}$$

N

ID	name	salary	age
1	Alice	\$100	20
2	Bob	\$250	20
3	Sue	\$300	30

Fig. 1. Example data migrations, with foreign keys (see Sects. 2.1 and 3.2)

2.2 Query Generation

Given a schema mapping $F : S \to T$, the *query generation* problem [4] is to construct a query which converts databases on S to databases on T in a way that satisfies F. The query languages typically studied include SQL, XQuery, and various comprehension- and λ-calculi [4].

A SQL query to implement the example mapping from Sect. 2.1 is:

```
INSERT INTO N
SELECT N1.ID, N1.age, N2.name, N1.sal
FROM N1, N2
WHERE N1.f = N2.ID
```

Technically, the `INSERT` portion of the above SQL code is not a "query", but rather an "update", and in practice the code generated from a query generation task will often store the results of the query. An example of running the above SQL is shown as the left-to-right direction of Fig. 1. In general, many or no SQL queries may implement a set of EDs [5]. EDs can also be directly executed by an algorithm called "the chase" [5].

2.3 Mapping Inversion

Given a schema mapping $F : S \to T$, the *mapping inversion* problem [10] is to construct a schema mapping $F^{-1} : T \to S$ that undoes F with respect to query generation (i.e. the queries generated from F and F^{-1} should be inverses).

The natural candidate ED to invert the schema mapping of Sect. 2.1 expresses that N projects onto N1 and N2:

$$\forall id_1, a, n, s.\ \mathsf{N}(id_1, a, n, s) \to \exists id_2.\ \mathsf{N1}(id, n, s, id_2) \wedge \mathsf{N2}(id_2, a)$$

and a possible SQL implementation of this ED is:

```
INSERT INTO N1                 | INSERT INTO N2
SELECT ID, name, sal, ID       | SELECT ID, age
FROM N                         | FROM N
```

However, the above ED is *not* an inverse to the ED of Sect. 2.1, as is seen by taking $\emptyset = \mathsf{N1} \neq \mathsf{N2}$. Indeed, it is rare for an ED, or set of EDs, to be invertible, and weaker notions of inverse, such as "quasi-inverse" [10], are common in the literature [10]. An example of running the above SQL is shown as the right-to-left direction of Fig. 1.

2.4 Mapping Composition

Given schema mappings $F : S \to T$ and $G : T \to U$, the *mapping composition* problem [8] is to construct a schema mapping $G \circ F : S \to U$ that is equivalent with respect to the query generation problem (i.e. running the query generated from $G \circ F$ should have the same effect as running the query generated from G on the results of the query generated from F).

The composition of the ED from Sect. 2.1 with the ED from Sect. 2.3 is

$$\forall id_1, id_2, n, s, a.\ \mathsf{N1}(id_1, n, s, id_2) \wedge \mathsf{N2}(id_2, a) \to \exists x.\ \mathsf{N1}'(id, n, s, x) \wedge \mathsf{N2}'(x, a)$$

where $\mathsf{N1}'$, $\mathsf{N2}'$ are target "copies" of source tables $\mathsf{N1}$, $\mathsf{N2}$. This composed ED is not the identity, thereby showing that the ED from Sect. 2.3 does not invert the ED from Sect. 2.1. In the case of EDs, composed mappings may not exist [8], but some restrictions and extensions of EDs are closed under composition [8].

2.5 Schema Matching

Given two database schemas S and T, the *schema matching* problem [5] is to automatically find "correspondences" between S and T and to automatically infer schema mappings $S \to T$ from these correspondences. In general, inference of entire mappings cannot be fully automated and the focus of the matching problem is to reduce the human effort required to construct a schema mapping by e.g., suggesting partial mappings that can be completed by users. There are many techniques for schema matching ranging from comparison of column names by string similarity to machine learning algorithms; for an overview, see [5]. In the example from Sect. 2.1, two correspondences that are easy to automatically find are $(\mathsf{N1}, \mathsf{N})$ and $(\mathsf{N2}, \mathsf{N})$ and tools such as Clio [14] can create the ED from Sect. 2.1 from these two correspondences.

2.6 Further References

In this paper we will focus on the problems described in the previous sections, but many other problems are studied in the model management literature [4], and many of these problems are related to algebraic specification. For example, *schema/instance merge* problems [4], which arise often in data integration scenarios [2], can be formalized as pushouts in suitable categories of schemas/instances [20], and such pushouts are related to model-theoretic concepts such as *model amalgamation* [15].

Many software products solve model management problems [4], including *ETL (Extract, Transform, Load)* tools [5], which extract data from separate databases, apply user-specified transformations, and then load the result into a target system such a data warehouse; *query mediators* [5], which answer queries about a "virtual" integrated database by combining queries about separate source databases; and *visual schema mapping tools* [14] which allow users to create schema mappings by visually connecting related schema elements with lines, as shown in Fig. 2.

There have been at least two attempts to provide a "meta semantics" for model management operations. In [16] Melnik gives a "state based" meta semantics to some of the above operations by defining a schema mapping $S \rightarrow T$ to be an arbitrary binary relation between instances on S and instances on T; the ED-based semantics described above is an instantiation of this meta semantics. In [2,13] the authors give an "institution theoretic" meta semantics to some of the above operations by defining a schema mapping $S \rightarrow T$ to be a morphism in a suitable category of schemas; AQL's semantics is an instantiation of this meta semantics.

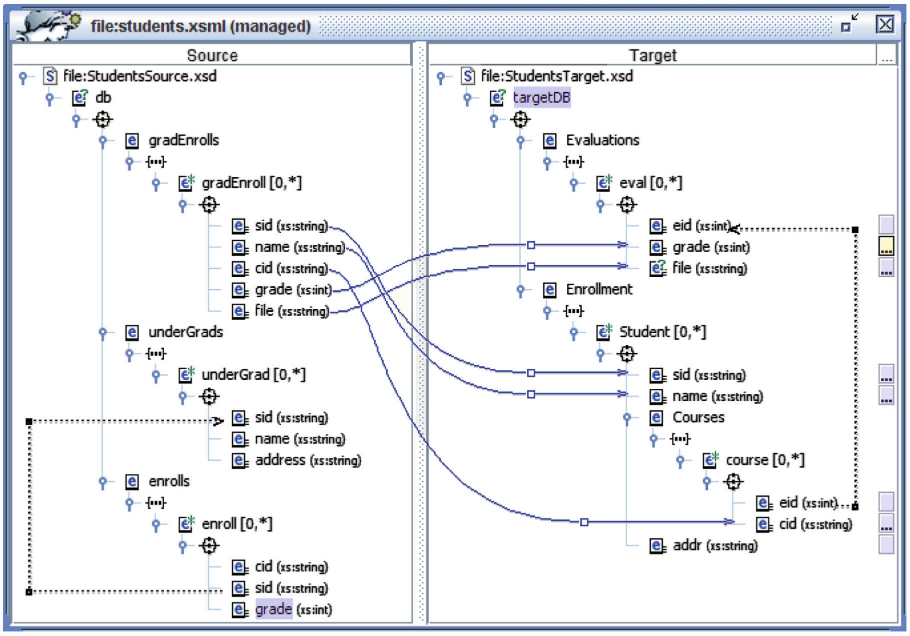


Fig. 2. A schema mapping in Clio [14]

3 Algebraic Model Management

Our approach to model management is based on the algebraic approach to databases, data migration, and data integration we describe in [19,20]. Those works, and hence this work, extend a particular category-theoretic data model that originated in the late 1990s [11] and was later extended in [21,23] and implemented in AQL (http://categoricaldata.net/aql.html).

In the next section we describe our formalism for database schemas and instances and introduce AQL. The subsequent sections implement the model management operations from Sect. 2 using our formalism. In this section we abbreviate "algebraic theory" as "theory".

3.1 Algebraic Databases

In our formalism [20], database schemas and instances are defined as theories of a certain kind, which we describe in the next sections. For ease of exposition, we will sometimes conflate schemas and instances as defined in our formalism with their AQL equivalents.

Type Sides. We first fix a theory, *Ty*, called the *type side* of our formalism. The sorts of *Ty* are called *types* and the functions of *Ty* are the functions that can appear in schemas and instances.

AQL allows arbitrary theories to be used as type sides. But we have found that in practice, AQL users almost always want to use the theory of an existing programming language, say java, for their type side. The ability to "bind" AQL to an existing language is particularly important in model management because input data may only be accessible through, e.g., a java API. For this reason, AQL allows a type side to be defined by specifying, for each sort s, a java class C_s and a java function `String` $\to C_s$ that tells AQL how to interpret the strings it encounters in AQL programs as objects of C_s.

An example AQL type side about integers and strings is shown in Fig. 3. This type side defines a theory with two sorts and infinitely many constants – all the java strings and integers – and no equations. The java code for `Int` says that whenever a string x is encountered in an AQL program and a term of sort `Int` is required, that java's `parseInt` function should be applied to x to yield the desired `Int`. The keyword `literal`, used in many places in AQL, indicates a literal (user-defined constant) definition.

Schemas. A *schema* on type side *Ty* is a theory extending *Ty* with new sorts (called *entities*), new unary functions from entities to types (called *attributes*), new unary functions from entities to entities (called *foreign keys*), and new equations (called *data integrity constraints*) of the form $\forall v : s.\ t = t'$, where s is an entity and t, t' are terms of the same type, each containing a single free variable v. The restrictions in the preceding sentence (e.g., no functions from types to entities) are necessary to use our formalism for model management purposes [19,20].

Figure 4 shows the AQL schemas corresponding to Fig. 1. These schemas contain no equations and are both on the type side *Ty* defined in Fig. 3.

```
typeside Ty = literal {
    java_types
         String = "java.lang.String"
         Int = "java.lang.Integer"
    java_constants
         String = "return input[0]"
         Int = "return java.lang.Integer.parseInt(input[0])"
}
```

Fig. 3. AQL type side *Ty*

```
schema S = literal : Ty {
    entities
         N1
         N2
    foreign_keys
         f : N1 -> N2
    attributes
         name : N1 -> String
         salary : N1 -> Int
         age : N2 -> Int
}
```

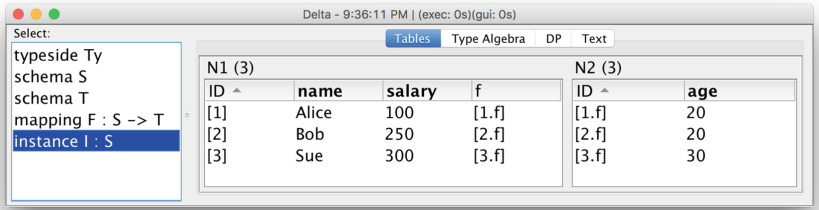
```
schema T = literal : Ty {
    entities
         N
    attributes
         name : N -> String
         salary : N -> Int
         age : N -> Int
}
```

Fig. 4. AQL schemas S and T on type side *Ty*

```
instance I = literal : S {
    generators
         1 2 3 : N1
    equations
         name(1) = Alice  salary(1) = 100   age(f(1)) = 20
         name(2) = Bob    salary(2) = 250   age(f(2)) = 20
         name(3) = Sue    salary(3) = 300   age(f(3)) = 30
}
```

Fig. 5. AQL instance I on schema S

Instances. An *instance* I on schema S is a theory extending S with new 0-ary function (constant) symbols called *generators* and non-quantified equations. An example AQL instance on schema S (Fig. 4) is shown in Fig. 5.



N1 (3)

ID	name	salary	f
[1]	Alice	100	[1.f]
[2]	Bob	250	[2.f]
[3]	Sue	300	[3.f]

N2 (3)

ID	age
[1.f]	20
[2.f]	20
[3.f]	30

Fig. 6. Initial algebra for AQL instance I

The intended meaning of an instance I, written $[\![I]\!]$, is the *term model* (i.e., *initial algebra*) for I which contains, for each sort s, a *carrier set* consisting of the closed terms of sort s modulo provability in I. A morphism of instances $I \to J$ is a homomorphism (natural transformation) of algebras $[\![I]\!] \to [\![J]\!]$.

Figure 6 shows the meaning of the instance I from Fig. 5 in the AQL tool. The AQL tool visually displays term models as sets of tables, one per entity e, each with an ID column corresponding to the carrier set for e. The tables in Fig. 6 are isomorphic to the left tables in Fig. 1.

In the following sections we implement the model management operations from Sect. 2 using the preceding definitions of schema and instance.

3.2 Schema Mapping

Given schemas S, T, the *schema mapping* problem (Sect. 2.1) is to construct a "mapping" $F : S \to T$ that captures some relationship between S and T.

Let S and T be AQL schemas on the same type side Ty. An AQL schema mapping $F : S \to T$ is defined as a "derived signature morphism" [18] from S to T that is the identity on Ty. That is, $F : S \to T$ assigns to each entity $e \in S$ an entity $F(e) \in T$, and to each attribute/foreign key $f : s \to s'$ a term $F(f)$, of type $F(s')$ and with one free variable of type $F(s)$, in a way that respects equality: if $S \vdash t = t'$, then $T \vdash F(t) = F(t')$. We have found that many mappings arising in practice cannot be expressed using plain signature morphisms and require the more general notion of "derived" signature morphism.

Whereas a schema mapping in Sect. 2.1 was an ED (formula in a fragment of first-order logic), which induces a single binary satisfaction relation between instances, AQL schema mappings are derived signature morphisms and induce three relations between instances, which we will describe in the next section.

An example AQL schema mapping $F : S \to T$ is shown in Fig. 7, where AQL schemas S and T are defined in Fig. 4. This mapping is also shown graphically in Fig. 1.

```
mapping F = literal : S -> T {
    entities
        N1 -> N
        N2 -> N
    foreign_keys
        f -> lambda x:N. x
    attributes
        name -> lambda x:N. name(x)
        salary -> lambda x:N. salary(x)
        age -> lambda x:N. age(x)
}
```

Fig. 7. AQL schema mapping $F : S \to T$

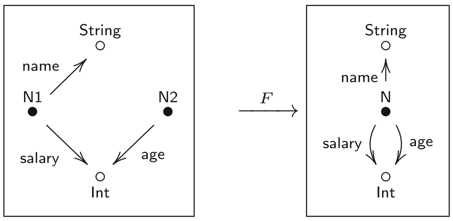
3.3 Query Generation

Given a mapping $F : S \to T$, the *query generation* problem (Sect. 2.2) is to use F to construct a query which converts databases on S to databases on T.

In our formalism, the database instances and morphisms on a schema S constitute a category, denoted S–**Inst**, and a schema mapping $F : S \to T$ induces a functor $\Sigma_F : S\text{–}\mathbf{Inst} \to T\text{–}\mathbf{Inst}$ defined by substitution. The functor Σ_F has a right adjoint, $\Delta_F : T\text{–}\mathbf{Inst} \to S\text{–}\mathbf{Inst}$, which corresponds to the "model reduct functor" when our formalism is described in institution-theoretic terms [2]. The functor Δ_F has a right adjoint, $\Pi_F : S\text{–}\mathbf{Inst} \to T\text{–}\mathbf{Inst}$. See [19] for proof that Δ_F always has left and right adjoints. As adjoints, Δ_F, Π_F preserve limits and Δ_F, Σ_F preserve colimits, implying many useful properties; for example, $\Sigma_F(I + J) \cong \Sigma_F(I) + \Sigma_F(J)$ and $\Pi_F(I \times J) \cong \Pi_F(I) \times \Pi_F(J)$.

Note that unlike Sect. 2.1, where there was a single query associated with a schema mapping (ED), in our algebraic approach there are three queries, one for each of Δ_F, Σ_F, Π_F. The conditions under which Δ_F,Σ_F, Π_F can be expressed in SQL and vice-versa are characterized in [23].

Although it is possible to give explicit formulae to define $\Delta_F, \Sigma_F, \Pi_F$ [19] we instead give examples in Figs. 1 and 8. Note that in these examples we are not showing instances (theories) as defined in Sect. 3.1; we are showing term models. For this reason, we surround $\Delta_F, \Sigma_F, \Pi_F$ with denotation brackets $[\![]\!]$ in these examples. In addition, as adjoints Δ, Σ, Π are only defined up to unique isomorphism, so we arbitrarily make up names for IDs and in these examples. Figures 1 and 8 show an AQL schema mapping F which takes two distinct source entities, N1 and N2, to the target entity N. The $[\![\Delta_F]\!]$ functor projects in the opposite direction of F: it projects columns from the single table for N to two separate tables for N1 and N2, similar to `FROM N AS N1` and `FROM N AS N2` in SQL. When there is a foreign key from N1 to N2, the $[\![\Delta_F]\!]$ functor populates it so that N can be recovered by joining N1 and N2. The $[\![\Pi_F]\!]$ functor takes the cartesian product of N1 and N2 when there is no foreign key between N1 and



N1

ID	name	salary
1	Alice	\$100
2	Bob	\$250
3	Sue	\$300

N2

ID	age
4	20
5	20
6	30

$\xleftarrow{[\![\Delta_F]\!]}$

N

ID	name	salary	age
1	Alice	\$100	20
2	Bob	\$250	20
3	Sue	\$300	30

N1

ID	name	salary
1	Alice	\$100
2	Bob	\$250
3	Sue	\$300

N2

ID	age
4	20
5	20
6	30

$\xrightarrow{[\![\Sigma_F]\!]}$

N

ID	name	salary	age
1	Alice	\$100	age(1)
2	Bob	\$250	age(2)
3	Sue	\$300	age(3)
4	name(4)	salary(4)	20
5	name(5)	salary(5)	20
6	name(6)	salary(6)	30

N1

ID	name	salary
1	Alice	\$100
2	Bob	\$250
3	Sue	\$300

N2

ID	age
4	20
5	20
6	30

$\xrightarrow{[\![\Pi_F]\!]}$

N

ID	name	salary	age
1	Alice	\$100	20
2	Bob	\$250	20
3	Sue	\$300	20
4	Alice	\$100	20
5	Bob	\$250	20
6	Sue	\$300	20
7	Alice	\$100	30
8	Bob	\$250	30
9	Sue	\$300	30

Fig. 8. Example data migrations (see Sect. 3.2)

N2, and joins N1 and N2 along the foreign key when there is. The $[\![\Sigma_F]\!]$ functor disjointly unions N1 and N2; because N1 and N2 are not union compatible (have different columns), $[\![\Sigma_F]\!]$ creates null values. When there is a foreign key between N1 and N2, $[\![\Sigma_F]\!]$ merges the tuples that are related by the foreign key, resulting in a join. As these examples illustrate, Δ_F can be thought of as projection, Π_F can be thought of as a product followed by a filter (which can result in a join), and Σ_F can be thought of as a disjoint union (which does not require union-compatibility) followed by a merge (which can also result in a join).

3.4 Mapping Composition

Given schema mappings $F : S \to T$ and $G : T \to U$, the *mapping composition* problem (Sect. 2.4) is to construct a schema mapping $G \circ F : S \to U$ that is equivalent with respect to query generation.

In one sense, the mapping composition problem is trivial [19] for our formalism: $\Delta_{F \circ G} \cong \Delta_G \circ \Delta_F$, $\Pi_{F \circ G} \cong \Pi_F \circ \Pi_G$, and $\Sigma_{F \circ G} \cong \Sigma_F \circ \Sigma_G$. But this solution is not wholly satisfactory because in practice a mixture of Δ, Σ, Π functors may be needed to accomplish any particular task (similarly, in SQL a mixture of joins and unions may be needed to accomplish any particular task). The following results are proved in [19,23]:

– Every composition $\Sigma_F \circ \Delta_G$ is isomorphic to $\Delta_{F'} \circ \Sigma_{G'}$ for some F', G'. This statement is also true if Σ_F is replaced with Π_F.
– Pairs of the form (F, G), denoting $\Sigma_F \circ \Delta_G$, are closed under composition. This statement is also true if Σ_F is replaced with Π_F. Such pairs can be specified in an intuitive "select-from-where" syntax, described in [19,20].
– Triples of the form (F, G, H), denoting $\Sigma_F \circ \Pi_G \circ \Delta_H$, are closed under composition, provided that F is a *discrete op-fibration* [3], which is exactly the "union compatibility" condition [5] that Σ_F performs unions over tables whose columns match; Fig. 1 is not a discrete op-fibration.

3.5 Mapping Inversion

Given a schema mapping $F : S \to T$, the *mapping inversion* problem (Sect. 2.3) is to construct a mapping $F^{-1} : T \to S$ that somehow "undoes" F.

Our formalism has strong inversion properties but does not have inverses per se. When there exists $F^{-1} : T \to S$ such that $F \circ F^{-1} = id$ and $F^{-1} \circ F = id$, then $\Delta_F \circ \Delta_{F^{-1}} \cong id$, $\Sigma_F \circ \Sigma_{F^{-1}} \cong id$, and $\Pi_F \circ \Pi_{F^{-1}} \cong id$. In general F need not have an inverse, but when S and T have finite initial algebras/term models (which is a priori undecidable, and implies decidability of S and T) it is possible to construct F^{-1} whenever it exists by considering all possible functors $T \to S$. When F has a right adjoint $G : T \to S$, a weaker condition than having an inverse, there are canonical morphisms $\Sigma_F \to \Delta_G$ and $\Delta_F \to \Pi_G$.

In practice "round-tripping" [5] of data is desirable even when inverses do not exist. For example, projection, because it forgets information, typically cannot be inverted, but we may want to remember where the projected data originated. In our formalism the adjunctions between Σ,Δ,Π provide round-tripping. For example, for every $F : S \to T$ and S-instance I there is a canonical morphism $I \to \Delta_F(\Sigma_F(I))$, the *unit* of the $\Sigma_F \dashv \Delta_F$ adjunction, which describes where each ID in I is sent to by Σ_F (and similarly for Π_F). Dually, for every T-instance J there is a canonical morphism $\Sigma_F(\Delta_F(J)) \to J$, the *co-unit* of the $\Sigma_F \dashv \Delta_F$ adjunction, which describes where the IDs in $\Delta_F(J)$ originate (and similarly for Π_F). The unit and co-unit can be used to obtain, for every morphism $h : \Sigma_F(I) \to J$, a mate $h' : I \to \Delta_F(J)$ and vice-versa (and similarly for Π_F). Relating adjointness to existing relaxed notions of inverse such as quasi-inverse [9] is an important area for future work.

3.6 Schema Matching

Given database schemas S and T, the *schema matching* problem (Sect. 2.5) is to automatically suggest schema mappings $S \to T$ to the user.

In this section, we define two schema matching techniques used by AQL. Our techniques compare entities, and foreign keys and attributes ("symbols") by name, as strings, and so our techniques depend on having (probably user-provided) names whose similarity as strings reflects their semantic similarity. Let $\sigma : \mathsf{String}, \mathsf{String} \to [0, 1]$ be any string similarity function [5] where a value of 1 indicates a "good" match and a value of 0 indicates a "bad" match.

- The first technique attempts to infer a schema mapping $F : S \to T$. For each entity $s \in S$, we define $F(s) := t$ where $t \in T$ is an entity that maximizes $\sigma(s, t)$. For each symbol $f : s \to s' \in S$, we then consider the set X of symbols $F(s) \to F(s')$. If X is non-empty, we choose a symbol $g \in X$ that maximizes $\sigma(f, g)$ and set $F(f) := g$. If X is empty but there is a shortest path p from $F(s)$ to $F(s')$, we set $F(f) := p$. If no shortest path exists, the match fails. The F so constructed is only a candidate schema mapping: AQL must verify that F preserves provable equality in S.
- The second technique attempts to infer a schema A and schema mappings $F : A \to S$ and $G : A \to T$. Such a span of mappings can be interpreted as a query of the form $\Sigma_F \circ \Delta_G$ or $\Pi_F \circ \Delta_G$. Let c be some user-provided string similarity cutoff. The entities of A are those pairs of S-entities and T-entities (s, t) such that $\sigma(s, t) > c$. The symbols $(s, t) \to (s', t')$ of A are those pairs of S-symbols and T-symbols $(f : s \to s', g : t \to t')$ such that $\sigma(f, g) > c$. The mappings F and G are projections.

4 Conclusion

When comparing our algebraic approach to model management with other approaches originating in relational database theory [1] it is important to note that our databases are "deductive databases" [1]. That is, we define databases "intensionally", as sets of equations, rather than as sets of tables. As such, care must be taken when mediating between our definitions and relational definitions. For example, our instances can be "inconsistent" in the sense that an instance can prove $1 = 2$ for two distinct constant symbols 1 and 2. Such situations are often, but not always [12], errors, and the AQL tool checks for such situations using standard techniques based on "conservative theory extensions" [12]. In addition, our schemas do not define a set of constants (a "domain") that all the instances on that schema share, as is customary in relational database theory [7]. Hence our approach is closer in spirit to traditional logic [6] than database theory [1].

There are many connections between our algebraic approach to model management and the ED-based approach described in Sect. 2. EDs are more expressive than our purely equational data integrity constraints and can be added to our formalism in a simple way, described in [22] (although in [22], EDs are called "lifting problems"). In ED-based approaches the "chase" [5] operation has a similar semantics to our Σ operation, and a formal comparison between the chase and Σ is forthcoming.

Acknowledgements. The authors thank Lucian Popa, Eswaran Subrahmanian, and Peter Gates and were supported by NIST SBIR grant 70NANB 16H178, AFOSR grant FA9550–14–1–0031 and NASA grant NNL14AA05C.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley-Longman, Boston (1995)
2. Alagić, S., Bernstein, P.A.: A model theory for generic schema management. In: Ghelli, G., Grahne, G. (eds.) DBPL 2001. LNCS, vol. 2397, pp. 228–246. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46093-4_14
3. Barr, M., Wells, C.: Category Theory for Computing Science. Prentice Hall International (1995)
4. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: ICMD (2007)
5. Doan, H., Halevy, A., Ives, Z.: Principles of Data Integration. Morgan Kaufmann, Burlington (2012)
6. Enderton, H.B.: A Mathematical introduction to logic. Academic Press, Cambridge (2001)
7. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theor. Comput. Sci. **336**(1), 89–124 (2005)
8. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.: Composing schema mappings: second-order dependencies to the rescue. TODS **30**(4), 994–1055 (2005)
9. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.: Quasi-inverses of schema mappings. TODS **33**(2), 11 (2008)
10. Fagin, R.: Inverting schema mappings. TODS **32**(4), 25 (2007)
11. Fleming, M., Gunther, R., Rosebrugh, R.: A database of categories. J. Symbolic Comput. **35**(2), 127–135 (2003)
12. Ghilardi, S., Lutz, C., Wolter, F.: Did I damage my ontology? Principles of knowledge representation and reasoning (2006)
13. Goguen, J.: Information integration in institutions (unpublished) (2004). http://cseweb.ucsd.edu/~goguen/pps/ifi04.pdf
14. Haas, L.M., Hernández, M.A., Ho, H., Popa, L., Roth, M.: Clio grows up: from research prototype to industrial tool. In: ICMD (2005)
15. Hodges, W.: A Shorter Model Theory. Cambridge University Press, Cambridge (1997)
16. Melnik, S.: Generic Model Management: Concepts and Algorithms. LNCS. Springer, Heidelberg (2004)
17. Mitchell, J.C.: Foundations of Programming Languages. MIT Press, Cambridge (1996)
18. Mossakowski, T., Krumnack, U., Maibaum, T.: What is a derived signature morphism? In: Codescu, M., Diaconescu, R., Ţuţu, I. (eds.) WADT 2015. LNCS, vol. 9463, pp. 90–109. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28114-8_6

19. Schultz, P., Spivak, D.I., Vasilakopoulou, C., Wisnesky, R.: Algebraic databases. Theory and Applications of Categories (2017)
20. Schultz, P., Wisnesky, R.: Algebraic data integration. J. Funct. Program. **27** (2016). Cambridge University Press. https://doi.org/10.1017/S0956796817000168
21. Spivak, D.I.: Functorial data migration. Inf. Comput. **217**, 31–51 (2012)
22. Spivak, D.I.: Database queries and constraints via lifting problems. Math. Struct. Comput. Sci. **6**(24) (2014)
23. Spivak, D.I., Wisnesky, R.: Relational foundations for functorial data migration. In: DBPL (2015)

Regular Papers

Probability Functions in the Context of Signed Involutive Meadows (Extended Abstract)

Jan A. Bergstra and Alban Ponse(✉)

Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
A.Ponse@uva.nl
https://staff.science.uva.nl/j.a.bergstra/,
https://staff.science.uva.nl/a.ponse/

Abstract. The Kolmogorov axioms for probability functions are placed in the context of signed meadows. A completeness theorem is stated and proven for the resulting equational theory of probability calculus. Elementary definitions of probability theory are restated in this framework.

Keywords: Meadow · Bayes' theorem · Bayesian reasoning

1 Introduction

The Kolmogorov axioms for probability functions may be considered a module that can be included in a variety of more or less formalized contexts. We will propose and investigate some consequences of these axioms when placed in the context of involutive meadows, that is meadows where inverse is an involution following the terminology of [7].

In particular we will discuss an axiomatization of a probability function (PF) on a Boolean algebra. The Boolean algebra serves as an event space, the PF 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 the signed meadow of real numbers. The following objectives motivate the line of development in this paper.

1. To develop an approach towards strictly equational reasoning about probability.
2. To provide a finite loose equational specification of probability functions.
3. To provide a useful completeness result for equational axioms of probability functions.
4. To investigate some total versions of the conditional probability operator.
5. To initiate the development of an application for the theory of signed meadows as outlined in [4,5].

This paper is a shortened version of http://arxiv.org/abs/1307.5173v4.

P. James and M. Roggenbach (Eds.): WADT 2016, LNCS 10644, pp. 73–87, 2017.
https://doi.org/10.1007/978-3-319-72044-9_6

We will produce an axiom system consisting of twenty-six equational axioms covering Boolean algebra, meadows, the sign function, and the PF. Then we will introduce several derived operators and prove a number of simple facts, including Bayes' theorem.

These axioms constitute a finite equational basis for the class of Boolean algebra based, real-valued PFs. In other words, the completeness results of [4,5] extend to the case with Boolean algebra based PFs. We understand this result to convey that the set of twenty-six axioms is complete in a reasonable sense.

The paper is structured as follows: in the remainder of this section we discuss the concept of a meadow in more detail and provide a survey of relevant design options. In Sect. 2 we introduce some preliminaries. In Sect. 3 we provide equational axioms for a PF, and in Sect. 4 we discuss completeness. In Sect. 5 we consider multi-dimensional probability functions, and Sect. 6 contains some concluding remarks.

1.1 A Survey of Design Options for the Inverse of 0

A meadow is a ring-like structure equipped with an inverse function. A ring based meadow expands a ring with a one place inverse function (inversive notation), or a two place division function (divisive notation). The terms 'inversive notation' and 'divisive notation' were coined in [6].

The key design choice that needs to be made when contemplating a meadow concerns the way it handles the inverse of 0. In a rather scattered literature on the subject a plurality of different options has been developed and studied, though in varying levels of detail. A brief survey of these endeavours sets the stage for the plan of this paper. The listing below is incomplete, but it contains all proposals for which we have been able to find an unambiguous description. As a criterion regarding this judgement we have required that (i) it must be possible to find out when a closed expressions written using $0, 1, +, -, \cdot, (-)^{-1}$ is considered to have a value in the mathematical structure at hand, (ii) for two closed expressions both having a value it must be possible to determine equality in the same structure, and (iii) the relation between inverse and division must be transparent. We will distinguish three design options for ring based meadows and three design options for non-ring based meadows. We will first survey design options for non-ring based meadows.

Non-ring Based Meadows
Three options for setting the inverse of zero in a non-ring based meadow stand out, each involving an error value which fails to meet the requirements of a ring. Distinguishing these options is facilitated by making use of a uniform terminology.

Natural inverse. If 0^{-1} is equated with an unsigned infinite value, often denoted by ∞, then 0 is said to have a natural inverse. The use of natural inverse in mathematics dates back to Riemann at least. Wheels are the prominent instance of meadows with natural inverse, see [9].

Signed natural inverse. If the inverse of zero is equated with a signed infinite value (written say as $+\infty = \infty$, which differs from $-\infty$) we propose to speak of a signed natural inverse. This design choice underlies the transreals and transrationals, see [16].

Common inverse. If the inverse of zero is equated with an error value $\mathbf{a}$ then, following [8], zero is said to have a common inverse. Common meadows are meadows based on common inverse. The error value $\mathbf{a}$ satisfies $x + \mathbf{a} = x \cdot \mathbf{a} = -\mathbf{a} = \mathbf{a}$ and for that reason fails to comply with the requirements for a ring $(0 \cdot x = 0)$. Moreover, the error value is unique.

Also in the case of natural inverse and signed natural inverse, the error value(s) fail to comply with the requirements for a ring $(0 \cdot x = 0)$.

Ring Based Meadows
For ring based meadows three options may be distinguished.

Partial inverse. The most prominent ring based meadow leaves the inverse of 0 undefined and considers inverse to be a partial function.
Working with partial inverse deviates from mathematical practice to the extent that questions like whether or not $1/0 = 2/0$ must be taken seriously. When dealing with partial inverse there are no semantic questions about it, but the choice of a logic of partial functions leaves substantial room for design variation, beginning with a choice between three ways of looking at the truth value of say $1/0 = 1/0$: is it considered as being true, or as being false in an overarching two-valued logic, or as not being true in an overarching logic which is not two-valued.

Symmetric inverse. If the meadow is based on a regular ring and the value of 0^{-1} is taken to be 0, 0 is said to have a symmetric inverse. The meadows of [4,5] and several preceding papers are ring based meadows with symmetric inverse. Alternatively this case is referred to as featuring an involutive inverse, and such meadows are referred to as involutive meadows.

Non-involutive inverse. If the inverse of 0^{-1} is taken to be different from 0, $(x^{-1})^{-1} = x$ cannot hold, that is inverse is not an involution, and inverse is said to be non-involutive. The non-involutive meadows discussed in [7] that satisfy $0^{-1} = 1$ are ring based meadows with an asymmetric inverse. If the inverse of 0^{-1} is taken to be say 17 or any (rational or real) number different from 0 and 1, 0 is said to have an ad hoc non-involutive inverse. Ad hoc non-involutive inverses come into play when formalizing the theory of fields in first order logic in the presence of a function symbol for either inverse or division (or both).

1.2 Working with Involutive Ring Based Meadows

In this paper we will work exclusively with ring based involutive meadows, which will be referred to simply as meadows. The motivation for this choice is that it appears to be a most straightforward way to pursue the objectives that were listed above. However, we do not claim that for the purpose of developing an equational approach to probability working with ring based meadows is the best option, neither do we claim that among the three options for ring based meadows working with a symmetric inverse is best suited to this objective.

2 Boolean Algebras and Meadows

In this section we specify the mathematical context on which our axiomatization is based. In particular, we provide specifications for Boolean algebras (Sect. 2.2), and for events and (signed) meadows (Sect. 2.3).

2.1 Boolean Algebras

A Boolean algebra $(B, +, \cdot, ', 1, 0)$ may be defined as a system with at least two elements such that $\forall x, y, z \in B$ the well-know 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 $(B, \vee, \wedge, \neg, \top, \bot)$ and adopt the axioms in Table 1. In [14] it was shown that the axioms in Table 1 constitute an equational basis.

Table 1. *BA*, a self-dual equational basis for Boolean algebras

$(x \vee y) \wedge y = y$	(1)	$x \vee (y \wedge z) = (y \vee x) \wedge (z \vee x)$	(4)
$(x \wedge y) \vee y = y$	(2)	$x \wedge \neg x = \bot$	(5)
$x \wedge (y \vee z) = (y \wedge x) \vee (z \wedge x)$	(3)	$x \vee \neg x = \top$	(6)

2.2 Valuated Boolean Algebras and Some Naming Conventions

A Boolean algebra can be equipped with a valuation v that assigns to its elements values in a signed meadow.

In this paper we will investigate the special case where the valuation function of a valuated Boolean algebra is a probability function by requiring that the valuation satisfies the Kolmogorov axioms for probability functions cast to the setting of signed meadows.

By way of notational convention we will from now on assume that E (for events) is the name of the carrier of a Boolean algebra, and that V (for values) names the carrier of the meadow in a valuated Boolean algebra.

2.3 Events and Signed Meadows

The set of axioms in Table 2 specifies the class of meadows. In the setting of probability functions the elements of the underlying Boolean algebra are referred to as events.[1] We will use "value" to refer to an element of a meadow,[2] and a probability function is a valuation (from events to the values in a signed meadow).[3]

Table 2. *Md*, a set of axioms for meadows

$(x+y)+z = x+(y+z)$	(7)	$x \cdot y = y \cdot x$	(12)
$x+y = y+x$	(8)	$1 \cdot x = x$	(13)
$x+0 = x$	(9)	$x \cdot (y+z) = x \cdot y + x \cdot z$	(14)
$x+(-x) = 0$	(10)	$(x^{-1})^{-1} = x$	(15)
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	(11)	$x \cdot (x \cdot x^{-1}) = x$	(16)

An expression of type E is an event expression or an event term, an expression of type V is a value expression or equivalently a value term. In the signature of a valuated Boolean algebra there is just one notation for a probability function, the function symbol P.[4]

In a meadow equipped with an ordering $<$, the sign function $\mathbf{s}(\_)$ is defined by

$$\mathbf{s}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } 0 < x. \end{cases}$$

The axioms in Table 3 specify the sign function in an equational manner. Before commenting on these axioms, we define the conditional $p \lhd q \rhd r$ expressing a form of "if-then-else", notations for a division operator, absolute value, and orderings. Furthermore, we adopt the convention to write $x - y$ for $x + (-y)$. Let

Table 3. *Sign*, a set of axioms for the sign operator

$\mathbf{s}(1_x) = 1_x$	(17)	$\mathbf{s}(x^{-1}) = \mathbf{s}(x)$	(20)
$\mathbf{s}(0_x) = 0_x$	(18)	$\mathbf{s}(x \cdot y) = \mathbf{s}(x) \cdot \mathbf{s}(y)$	(21)
$\mathbf{s}(-1) = -1$	(19)	$0_{\mathbf{s}(x)-\mathbf{s}(y)} \cdot (\mathbf{s}(x+y) - \mathbf{s}(x)) = 0$	(22)

[1] Events are closed under $- \vee -$, which represents alternative occurrence and $- \wedge -$, which represents simultaneous occurrence, and under negation.

[2] Rational numbers and real numbers are instances of values.

[3] We will exclude probability functions with negative values, a phenomenon known in non-commutative probability theory, leaving the exploration of that kind of generalization to future work.

[4] In some cases the restriction to a single probability function P is impractical and providing a dedicated sort for such functions brings more flexibility and expressive power. This expansion may be achieved in different ways.

p, q and r range over V, the carrier of the signed meadow in a valuated Boolean algebra, then

$$
\begin{aligned}
1_p &=_{\texttt{def}} p \cdot p^{-1}, & p/q &=_{\texttt{def}} \frac{p}{q},\\
0_p &=_{\texttt{def}} 1 - 1_p, & |p| &=_{\texttt{def}} \mathbf{s}(p) \cdot p,\\
p \triangleleft q \triangleright r &=_{\texttt{def}} 1_q \cdot p + 0_q \cdot r, & p < q &=_{\texttt{def}} \mathbf{s}(q-p) = 1,\\
\frac{p}{q} &=_{\texttt{def}} p \cdot q^{-1}, & p \leq q &=_{\texttt{def}} \mathbf{s}(\mathbf{s}(q-p)+1) = 1.
\end{aligned}
$$

In Table 3, axiom (22) is an equational representation of the conditional equation $\mathbf{s}(x) = \mathbf{s}(y) \;\rightarrow\; \mathbf{s}(x+y) = \mathbf{s}(x)$. Finally, the equivalences

$$p \geq 0 \iff \mathbf{s}(\mathbf{s}(p)+1) = 1 \iff p = \mathbf{s}(p) \cdot p = |p|$$

are provable from $Md + Sign$ (this follows from Theorem 4.1.1 below).

We will also consider the subclass of signed *cancellation meadows*. A cancellation meadow satisfies the Inverse Law (IL) of Table 4.

Table 4. Inverse law (IL)

$$x \neq 0 \;\rightarrow\; x \cdot x^{-1} = 1$$

3 Signed Meadow Based Probability Calculus

In Sect. 3.1 we formulate axioms for a probability function. Following the methods of abstract data type specification we will focus on axioms in equational form. Then, we discuss a plurality of versions of the conditional probability operator (Sect. 3.2) and some properties thereof, in particular versions of Bayes' theorem. Finally, we consider independent events (Sect. 3.3).

3.1 Equational Axioms for a Probability Function

In Table 5 we define the set PF_P of axioms for a probability function. These axioms represent Kolmogorov's axioms in the context of a Boolean algebra (rather than a universe of sets) and a signed meadow (instead of a field). Axiom (25) expresses that the sign of $P(x)$ is nonnegative. Axiom (26) distributes P over finite unions. In the absence of an infinitary version of axiom (26) we consider these axioms to constitute an axiomatization for the restricted concept of probability functions only, rather than for probability measures in general.

In combination with the axioms $BA + Md$, the two axioms (24) and (26) in Table 5 can be replaced by the single axiom

$$P(x) = P(x \wedge y) + P(x \wedge \neg y) \tag{†}$$

Table 5. PF_P, a set of axioms for a probability function with name P

$P(\top) = 1$	(23)	$P(x) = \|P(x)\|$	(25)
$P(\bot) = 0$	(24)	$P(x \vee y) = P(x) + P(y) - P(x \wedge y)$	(26)

where the expressions $x \wedge y$ and $x \wedge \neg y$ characterize two disjoint (mutually exclusive) events: axiom (24) follows from $P(x) = P(x \wedge x) + P(x \wedge \neg x)$, thus $P(x \wedge \neg x) = P(\bot) = 0$, and axiom (26) follows from

$$P(x \vee y) \overset{\dagger}{=} P((x \vee y) \wedge x) + P((x \vee y) \wedge \neg x) = P(x) + P(y \wedge \neg x)$$

and $P(y) \overset{\dagger}{=} P(y \wedge x) + P(y \wedge \neg x)$, thus $P(y \wedge \neg x) = P(y) - P(x \wedge y)$. Conversely, axiom (†) follows from (24) and (26):

$$P(x) = P((x \wedge y) \vee (x \wedge \neg y)) = P(x \wedge y) + P(x \wedge \neg y) - P(\bot). \quad (\ddagger)$$

Theorem 3.1.1 (Disjoint event factorization). *$BA + Md + PF_P \vdash P(x) = P(x \wedge y) + P(x \wedge \neg y)$.*

Proof. This is (‡), which is shown above. $\square$

Theorem 3.1.2 (Probability upper bound). *$BA + Md + Sign + PF_P \vdash P(x) \leq 1$.*

Proof. First notice $1 = P(\top) = P(x \vee \neg x) = (P(x) + P(\neg x)) - P(\bot) = P(x) + P(\neg x)$, so $P(x) = 1 - P(\neg x)$. Because $P(\neg x) \geq 0$ we conclude $P(x) \leq 1$. $\square$

The following theorem asserts in equational form the conditional equation $P(y) = 0 \to P(x \wedge y) = 0$, using inversive notation.

Theorem 3.1.3 *$BA + Md + Sign + PF_P \vdash P(x \wedge y) \cdot P(y) \cdot P(y)^{-1} = P(x \wedge y)$.*

Proof. With help of Theorem 4.1.1, see there. $\square$

3.2 Conditional Probability as a Total Operator: Four Options

Conditional probability $P(x \mid y)$ of event x relative to event y is conventionally understood as a partial function of x and y, defined only if $P(y)$ is nonzero. The objective of developing an equational logic for probability theory suggests that total versions of the conditional probability operator ought to be contemplated.

Conditional probability defined according to Kolmogorov is written below as $P^\star(x \mid y)$, where variables x and y range over E, and is defined by

$$P^\star(x \mid y) =_{\texttt{def}} \frac{P(x \wedge y)}{P(y)} \lhd P(y) \rhd \uparrow.$$

Here $\uparrow$ denotes that the result is undefined.[5] The key advantage of partial conditional probability is that one does not introduce a value for, say $P(x \mid \bot)$ which might be subsequently disputed. Four ways of making conditional probability defined on all inputs will now be distinguished.

Definition 3.2.1 (Zero-totalized conditional probability).

$$P^0(x \mid y) =_{\texttt{def}} \frac{P(x \wedge y)}{P(y)}.$$

We notice that $P^0(\top \mid \bot) = 0$, a choice for which no convincing philosophical motivation can be put forward. Two advantages can be put forward in favour of $P^0(-|-)$: the logical simplicity that comes with it being total and the calculational simplicity that comes with choosing 0 as a value for $P^0(x \mid y)$ when $P(y) = 0$. The following properties are immediate:

$$P^0(x \mid x) = \frac{P(x)}{P(x)} \quad \text{and} \quad P(x) = P(x) \cdot P^0(x \mid x).$$

Moreover we have 'joint probability factorization':

$$P(x \wedge y) = P(x \wedge y) \cdot P(y) \cdot P(y)^{-1} = (P(x \wedge y)/P(y)) \cdot P(y) = P^0(x \mid y) \cdot P(y),$$

and 'total probability':

$$\begin{aligned} P(x) &= P(x \wedge y) + P(x \wedge \neg y) \\ &= P(x \wedge y) \cdot P(y) \cdot P(y)^{-1} + P(x \wedge \neg y) \cdot P(\neg y) \cdot P(\neg y)^{-1} \\ &= P^0(x \mid y) \cdot P(y) + P^0(x \mid \neg y) \cdot P(\neg y). \end{aligned}$$

Another illustration of the latter advantage is the derivability of Bayes' theorem in its simplest form (see Theorem 3.2.5.1).

Definition 3.2.2 (One-totalized conditional probability).

$$P^1(x \mid y) =_{\texttt{def}} \frac{P(x \wedge y)}{P(y)} \lhd P(y) \rhd 1.$$

[5] We assume that in a context of partial functions an identity $t = r$ is valid if either both sides are undefined or both sides are defined and equal. This convention, however, leaves room for alternative readings of the expressions at hand. In particular the definition given for $x \lhd y \rhd z$ implies that whenever t is undefined, so is $t \lhd r \rhd s$. That is not a very plausible feature of the conditional and in the presence of partial operations the conditional operator requires a different definition. These complications are to some extent avoided, or rather made entirely explicit, when working with total functions. The use of the notation $P^\star(-|-)$ instead of the common notation $P(-|-)$ is justified by the fact that unavoidably $P^\star(-|-)$ inherits properties from the equational specification of the functions from which it has been made up. Such properties need not not coincide with what is expected from $P(-|-)$.

We will write $x \to y$ for $\neg x \vee y$. The principal advantage of one-totalized conditional probability over zero-totalized conditional probability is the validity of the following rule, which provides some intrinsic motivation for this design of conditional probability:

$$(x \to y) = \top \;\Rightarrow\; P^1(x \mid y) = 1.$$

If $\alpha \in \{\star, 0, 1\}$ then the function $P \circ^\alpha y =_{\texttt{def}} \lambda x \in E.P^\alpha(x|y)$ is not a probability function for each y. In particular, if $P(y) = 0$, $P \circ^\alpha y$ will fail to comply with either $P \circ^\alpha y(\top) = 1$ or with $P \circ^\alpha y(\bot) = 0$. Now $\lambda P \in PF.P \circ^\alpha y$ being the well-known update operator that goes with some applications of Bayes' theorem, it is a reasonable requirement that this very operator becomes total as well. We will introduce two options for conditionalization which achieve this requirement.

Definition 3.2.3 (Safe conditional probability).

$$P^s(x \mid y) =_{\texttt{def}} \frac{P(x \wedge y)}{P(y)} \lhd P(y) \rhd P(x).$$

We find that $P \circ^s y = P$ if $P(y) = 0$, which allows the view that $\lambda P.P \circ^s y$ is an operator mapping probability functions to probability functions for all events y, or stated differently that $\lambda y.(\lambda P.P \circ^s y)$ is a total mapping from events to probability function transformations. $P \circ^s$ is safe because it enforces no update when an inconsistency is observed.

Yet another way to achieve this property of a conditional update is to return an exceptional value, in this case the canonical probability function for an atomic event. An atom in E is an event $a \in E$ which satisfies $\texttt{atom}(a) =_{\texttt{def}} \forall x \in E.(x \wedge a = a \texttt{ OR } x \wedge a = \bot)$. For an atom $a \in E$ the probability function $\texttt{pf}_a$ is defined by:

$$\texttt{pf}_a(x) =_{\texttt{def}} \begin{cases} 1 & \text{if}\, x \wedge a = a, \\ 0 & \text{if}\, x \wedge a = \bot. \end{cases}$$

Definition 3.2.4 (Exception raising conditional probability for atom $a \in E$).

$$P^{e/a}(x \mid y) =_{\texttt{def}} \frac{P(x \wedge y)}{P(y)} \lhd P(y) \rhd \texttt{pf}_a(x).$$

For $P^0(-|-), P^s(-|-)$, and $P^{e/a}(-|-)$ we are not aware of earlier definitions, whereas $P^1(-|-)$ has been considered by Adams in [1], and in subsequent literature. For a survey of conditional logic and conditional probabilities we refer to [12].

Of particular importance given its ubiquitous use is Bayes' theorem. Bayes' theorem takes different forms for different versions of conditional probability, and in each of these cases it appears as a consequence of $BA + Md + Sign + PF_P$.

Theorem 3.2.5 (Versions of Bayes' theorem). *In $BA + Md + Sign + PF_P$ the following equations are derivable:*

1. $P^0(x \mid y) = \dfrac{P^0(y \mid x) \cdot P(x)}{P(y)}$ $\quad (Bayes' \, theorem \, for P^0(-|-))$,
2. $P^1(x \mid y) = \dfrac{P^1(y \mid x) \cdot P(x)}{P(y)} \lhd P(y) \rhd 1$ $\quad (Bayes' \, theorem \, for P^1(-|-))$,
3. $P^s(x \mid y) = \dfrac{P^s(y \mid x) \cdot P(x)}{P(y)} \lhd P(y) \rhd P(x)$ $\quad (Bayes' \, theorem \, for P^s(-|-))$,
4. $P^{e/a}(x \mid y) = \dfrac{P^{e/a}(y \mid x) \cdot P(x)}{P(y)} \lhd P(y) \rhd \mathtt{pf}_a(x)$ $\quad (Bayes' \, theorem \, for$ $P^{e/a}(-|-))$.

Proof. Version 1: derive $P^0(x \mid y) = P(x \wedge y)/P(y) \overset{3.1.3}{=} (P(y \wedge x)/P(y)) \cdot (P(x)/P(x)) = (P(y \wedge x)/P(x)) \cdot (P(x)/P(y)) = (P^0(y \mid x) \cdot P(x))/P(y)$.

Version 2-4: see http://arxiv.org/abs/1307.5173v4. □

3.3 Independence of Events

A valuated Boolean algebra equipped with a valuation P in some signed meadow $\mathbb{M}$ that satisfies all axioms of $BA + Md + Sign + PF_P$ will be called a $K(\mathbb{M}, P)$-*structure*. Given a $K(\mathbb{M}, P)$-structure, two events x and y are said to be independent relative to that structure if $P(x \wedge y) = P(x) \cdot P(y)$ is valid.

Theorem 3.3.1. *Events x and y are independent if and only if $P^0(x \mid y) = P(x) \cdot P^0(y \mid y)$ and equivalently if and only if $P^0(y \mid x) = P(y) \cdot P^0(x \mid x)$.*

Proof. If x and y are independent, then $P^0(x \mid y) = P(x \wedge y)/P(y) = (P(x) \cdot P(y))/P(y) = P(x) \cdot P^0(y \mid y)$, and similarly one finds $P^0(y \mid x) = P(y) \cdot P^0(x \mid x)$.

Conversely, from $P^0(x \mid y) = P(x) \cdot P^0(y \mid y)$ one finds $P(x \wedge y)/P(y) = P(x) \cdot (P(y)/P(y))$, so multiplying both sides by $P(y)$ yields $P(x \wedge y) \cdot (P(y)/P(y)) = P(x) \cdot (P(y)/P(y)) \cdot P(y)$, which implies $P(x \wedge y) = P(x) \cdot P(y)$ by Theorem 3.1.3. □

4 Logical Aspects of Equations for Probability Functions

In this section we provide a completeness result for $BA + Md + Sign + PF_P$ (Sect. 4.1) and discuss the use of a free Boolean algebra as an event space (Sect. 4.2).

4.1 Completeness of $BA + Md + Sign + PF_P$

In [4] it is shown that $Md+Sign$ constitutes a finite basis for the equational theory of signed cancellation meadows. Stated differently: for each equation $t = r$, if $Md + Sign + PF_P + IL \models t = r$ then also $Md + Sign + PF_P \vdash t = r$, where IL is the inverse law defined in Table 4. This fact is understood as a completeness result because a stronger set of axioms would necessarily exclude some meadows that are expansions of ordered fields. In a preceding version of this paper[6] it was shown that the basis theorem extends to the setting with probability functions: if $BA + Md + Sign + PF_P + IL \models t = r$ then also $BA + Md + Sign + PF_P \vdash t = r$.

For the purposes of this paper we prefer to make use of a different completeness result for the same equational theory that allows us to obtain a more intuitively appealing completeness result for the axiom system $BA+Md+Sign+PF_P$. This second completeness result is given in terms of validity of equations relative to a single signed meadow rather than in an elementary class of structures.

We recall the following result [5, Theorem 3.14], where we write $\mathbb{R}_0$ for the meadow that is the expansion of the field of real numbers $\mathbb{R}$ with total inverse operator and $0^{-1} = 0$, and $(\mathbb{R}_0, \mathbf{s})$ for $\mathbb{R}_0$ expanded with the sign function $\mathbf{s}(\_)$.

Theorem 4.1.1. *For an equation $t = r$ in the signature of signed meadows: $(\mathbb{R}_0, \mathbf{s}) \models t = r$ if and only if $Md + Sign \vdash t = r$.*

One can apply this theorem to obtain a simple proof of Theorem 3.1.3: let $\phi(u, v) = 0_{|u|+|v|} \cdot u$. Then $(\mathbb{R}_0, \mathbf{s}) \models \phi(u, v) = 0$, so by Theorem 4.1.1 one obtains $Md + Sign \vdash \phi(u, v) = 0$. Substituting $P(y \wedge x)$ for u and $P(y \wedge \neg x)$ for v and applying Theorem 3.1.1, one derives

$$\begin{aligned} BA + Md + Sign + PF_P \vdash 0 &= \Big(1 - \frac{|P(y \wedge x)| + |P(y \wedge \neg x)|}{|P(y \wedge x)| + |P(y \wedge \neg x)|}\Big) \cdot P(y \wedge x) \\ &= \Big(1 - \frac{P(y)}{P(y)}\Big) \cdot P(y \wedge x), \end{aligned}$$

from which the required result follows immediately.

The same completeness result as Theorem 4.1.1 works for conditional equations (for a proof see http://arxiv.org/abs/1307.5173v4).

Theorem 4.1.2. *For a conditional equation $t_1 = r_1 \wedge \ldots \wedge t_n = r_n \to t = r$ in the signature of signed meadows: $(\mathbb{R}_0, \mathbf{s}) \models t_1 = r_1 \wedge \ldots \wedge t_n = r_n \to t = r$ if and only if $Md + Sign \vdash t_1 = r_1 \wedge \ldots \wedge t_n = r_n \to t = r$.*

A $K(\mathbb{R}_0, P)$*-structure* is a model of $BA + Md + Sign + PF_P$ that contains the meadow of signed reals, $(\mathbb{R}_0, \mathbf{s})$, as the domain of its values. We will write $K(\mathbb{R}_0, P)$ for the class of $K(\mathbb{R}_0, P)$-structures.

Theorem 4.1.1 can be extended to the setting of $K(\mathbb{R}_0, P)$-structures, thus obtaining a satisfactory completeness result for $BA + Md + Sign + PF_P$ (see http://arxiv.org/abs/1307.5173v4 for a proof that depends on Theorem 4.1.2).

Theorem 4.1.3. *The axiom system $BA + Md + Sign + PF_P$ is sound and complete for the equational theory of $K(\mathbb{R}_0, P)$.*[7]

[6] http://arxiv.org/abs/1307.5173v1.

[7] More generally, $BA + Md + Sign + PF_P$ is sound for the class of $K(\mathbb{M}, P)$-structures with $\mathbb{M}$ a signed cancellation meadow.

4.2 Using Free Boolean Algebras as Event Spaces

For the purpose of reformulating some elementary aspects of probability theory and statistics the generality of working with arbitrary Boolean algebras is inessential, at least at this initial stage in the development of an equational calculus of probabilities. For that reason we will now introduce several simplifying assumptions:

– A finite set C of constants for events is provided. Elements of C are called primitive events. We will only consider free Boolean algebras generated by the primitive events.
– With BA_C we will denote the equations for Boolean algebra in a signature which is expanded with the constants in C.
– The class of models of $BA_C + Md + Sign + PF_P$ with a free event space over C, $(\mathbb{R}_0, \mathbf{s})$ as its meadow of values, and a probability function P is denoted $K_C(\mathbb{R}_0, P)$. Different structures in $K_C(\mathbb{R}_0, P)$ only differ in the choice (interpretation) of the probability function P.

These assumptions correspond to what is needed for the specification of examples of probabilistic reasoning.

Theorem 4.2.1. *$Md + Sign + BA_C + PF_P$ is sound and complete for the equations of type V that are true in all structures in $K_C(\mathbb{R}_0, P)$. In other words, for t and r terms of sort V: $Md+Sign+BA_C+PF_P \vdash t = r \iff K_C(\mathbb{R}_0, P) \models t = r$.*

Proof. The proof is merely a reformulation of the proof of Theorem 4.1.3. □

5 Multi-dimensional Probability Functions

In this section we provide axioms for multi-dimensional PFs (Sect. 5.1), and discuss a condition for the existence of a particular universal PF (Sect. 5.2).

5.1 Equational Axioms for a Probability Function Family

Let $D = \{a_1, \ldots, a_d\}$ be a finite, non-empty set. The elements of D are referred to as dimensions. With A^f_D we denote the set of finite non-empty sequences of elements of D in which each dimension occurs at most once, and with $\ell(w)$ we denote the length of $w \in A^f_D$. Note that A^f_D is finite. Elements of A^f_D serve as arities of probability functions on a multi-dimensional event space of dimension $\ell(w)$. If $\ell(w) > 1$, then w is written as a comma-separated sequence, e.g. $\ell(a_1, a_3) = 2$ and we write $(a_1, a_3) \in A^f_D$.

Given an event space E and a name P for a probability function, an arity family for D is a subset W of A^f_D that is closed under permutation and under taking non-empty subsequences. Given an arity family W for D, a function family for W consists of a function $P^w : E^{\ell(w)} \to V$ for each arity $w \in W$. A function family for dimension set D, arity family W and function name P is a *probability function family* (PFF) if it satisfies the axioms of Table 6. Because in an arity repetition of dimensions is disallowed, these axioms reduce to what we had already in the case of a single dimension.

Table 6. *$PFF_{W,P}$*, axioms for a PFF with arity family W and name P, where $a \in D$, $k \in \mathbb{N}$, $\boldsymbol{x} = x_1, \ldots, x_k$ and $P(y, \boldsymbol{x}) = P(y)$ if $k = 0$, and $w = (a, u) \in W$ with $\ell(w) = k + 1$

$P^{a,v,b,v'}(y_1, x_1, \ldots, x_m, y_2, z_1, \ldots, z_n) = P^{b,v,a,v'}(y_2, x_1, \ldots, x_m, y_1, z_1, \ldots, z_n)$ for all $a, b \in D$ and $(a, v, b, v') \in W$, where v, v' can be empty (thus $m = 0, n = 0$)	(27)
$P^a(\top) = 1$	(28)
$P^{a,v}(\top, x_1, \ldots, x_{k+1}) = P^v(x_1, \ldots, x_{k+1})$	(29)
$P^w(\bot, \boldsymbol{x}) = 0$	(30)
$P^w(y, \boldsymbol{x}) = \lvert P^w(y, \boldsymbol{x})\rvert$	(31)
$P^w(y \vee z, \boldsymbol{x}) = P^w(y, \boldsymbol{x}) + P^w(z, \boldsymbol{x}) - P^w(y \wedge z, \boldsymbol{x})$	(32)

5.2 Existence of a Universal Probability Function

A subset W of A^f_D may or may not have a maximal element under inclusion. If W has a maximal element $\overline{w}$ and if we have a probability function family $(P^w)_{w \in W}$ for W, then $P^{\overline{w}}$ serves as a universal element for the family of probability functions because all other members of it can be found via successive application of the axioms (27)–(30).

As it turns out some PFFs cannot be extended with a universal PF. In the notation of our specification of probability families we will state a specific result that may serve as a necessary condition for the possibility to extend a PFF with a universal element.

Theorem 5.2.1. *Given a set of dimensions $D = \{a, b, c, d\}$, an arity family W for D that satisfies $W \supset \{(b, c),\ (b, d),\ (a, d),\ (a, c)\}$, and a PFF $(P^w)_{w \in W}$, let t be the following term:*

$$t = P^{b,c}(y, z) + P^{b,d}(y, u) + P^{a,d}(x, u) - P^{a,c}(x, z) - P^b(y) - P^d(u).$$

Then, if W has a maximal element, then $-1 \leq t \leq 0$, that is, the following two inequalities must hold for $G_{W,P} = BA + Md + Sign + PFF_{W,P}$:

$$G_{W,P} \vdash t + 1 = \mathbf{s}(t + 1) \cdot (t + 1) \quad \textit{and} \quad G_{W,P} \vdash -t = \mathbf{s}(-t) \cdot -t.$$

Clearly if a PFF for D contains all of $P^{b,c}, P^{b,d}, P^{a,d}, P^{a,c}$ and fails to meet either one of the mentioned inequalities on t, then a universal PF cannot be found for it.

These facts are known as the BCHS (Bell, Clauser, Horne, Shimony) inequalities. Both were formulated and shown in a set theoretic framework for probability theory in [15] and [10,11], and a straightforward proof is given in [13, Sect. 9.2],[8] which we repeat here.

[8] From this pair of inequalities one can derive the original Bell inequalities from [3]. The key observation of Bell was that quantum mechanics gives rise to the hypothesis that a 4-dimensional event space exists in which a family of joint probabilities for at most two dimensions can be found that violates the inequalities from the theorem.

Proof (of Theorem 5.2.1, taken from [13]).

$$\begin{aligned}
P^{b,c,d}(y,z,u) &= P^{a,b,c,d}(x,y,z,u) + P^{a,b,c,d}(\neg x,y,z,u)\\
&\leq P^{a,c}(x,z) + P^{a,d}(\neg x,u)\\
&= P^{a,c}(x,z) + P^{d}(u) - P^{a,d}(x,u), \qquad (33)\\
P^{b,c,d}(\neg y,z,u) &= P^{a,b,c,d}(x,\neg y,z,u) + P^{a,b,c,d}(\neg x,\neg y,z,u)\\
&\leq P^{a,d}(x,u) + P^{a,c}(\neg x,z)\\
&= P^{a,d}(x,u) + P^{c}(z) - P^{a,c}(x,z), \qquad (34)\\
0 \leq P^{b,c,d}(y,\neg z,\neg u) &= P^{b,c}(y,\neg z) - P^{b,c,d}(y,\neg z,u)\\
&= P^{b}(y) - P^{b,c}(y,z) - P^{b,d}(y,u) + P^{b,c,d}(y,z,u). \qquad (35)
\end{aligned}$$

Combining (33) and (35) yields

$$0 \leq P^{b}(y) - P^{b,c}(y,z) - P^{b,d}(y,u) + P^{a,c}(x,z) + P^{d}(u) - P^{a,d}(x,u). \qquad (36)$$

By (35) and the equality $-P^{c,d}(z,u) + P^{c,d}(\neg z,\neg u) = 1 - P^{c}(z) - P^{d}(u)$,

$$\begin{aligned}
0 &\leq P^{b,c,d}(\neg y,\neg z,\neg u) = P^{c,d}(\neg z,\neg u) - P^{b,c,d}(y,\neg z,\neg u)\\
&= 1 - P^{b}(y) - P^{c}(z) - P^{d}(u) + P^{b,c}(y,z) + P^{b,d}(y,u) + P^{b,c,d}(\neg y,z,u). \qquad (37)
\end{aligned}$$

Then from (34) and (37) we get

$$0 \leq 1 - P^{b}(y) - P^{d}(u) + P^{b,c}(y,z) + P^{b,d}(y,u) + P^{a,d}(x,u) - P^{a,c}(x,z). \qquad (38)$$

Inequalities (36) and (38) prove the theorem. □

6 Concluding Remarks

The incentive for this work came from a talk given by professor Ian Evett on the occasion of the retirement of dr. Huub Hardy as a driving force behind the MSc Forensic Science at the University of Amsterdam.[9] That talk illustrated the headway that the Bayesian approach to reasoning in forensic matters has made in recent years. However, Evett also highlighted the conceptual and political problems that may still lie ahead of its universal adoption in the legal process.

In order to improve the understanding of these issues an elementary logical formalization of reasoning with probabilities might be useful. With that perspective in mind we came to the conclusion that in spite of the abundance of introductory texts to probability theory, the development of an axiomatic approach from first principles may yet cover new ground. The formalization of probabilities in terms of equational logic outlined above is intended to serve as a point of departure from which to develop presentations of probability theory that

[9] This meeting took place at Science Park Amsterdam, Friday June 7, 2013 under the heading "Frontiers of Forensic Science", and was organized by Andrea Haker.

may be be helpful when a formal and logically precise perspective on reasoning with probabilities is aimed at.

We acknowledge many discussions with Andrea Haker (University of Amsterdam) regarding the relevance of logically grounded reasoning methodologies in forensic science. We thank both reviewers for their constructive comments.

References

1. Adams, E.: Probability and the logic of conditionals. In: Hintikka, J., Suppes, P. (eds.) Aspects of Inductive Logic, pp. 265–316. North-Holland (1966)
2. Barber, D.: Bayesian Reasoning and Machine Learning. Cambridge University Press, Cambridge (2012). http://web4.cs.ucl.ac.uk/staff/D.Barber/pmwiki/pmwiki.php?n=Brml.Online. ISBN 0521518148, 9780521518147. On-line version 18 November 2015
3. Bell, J.S.: On the Einstein Podolsky Rosen paradox. Physics **1**(3), 195–200 (1964)
4. Bergstra, J.A., Bethke, I., Ponse, A.: Cancellation meadows: a generic basis theorem and some applications. Comput. J. **56**(1), 3–14 (2013). https://doi.org/10.1093/comjnl/bxs028. https://arxiv.org/abs/0803.3969v3
5. Bergstra, J.A., Bethke, I., Ponse, A.: Equations for formally real meadows. J. Appl. Log. **13**(2, Part B), 1–23 (2015). https://doi.org/10.1016/j.jal.2015.01.004. https://arxiv.org/abs/1310.5011v4
6. Bergstra, J.A., Middelburg, C.A.: Inversive meadows and divisive meadows. J. Appl. Log. **9**(3), 203–220 (2011). https://arxiv.org/abs/0907.0540v3
7. Bergstra, J.A., Middelburg, C.A.: Division by zero in involutive meadows. J. Appl. Log. **13**(1), 1–12 (2015). https://arxiv.org/abs/1406.2092
8. Bergstra, J.A., Ponse, A.: Fracpairs and fractions over a reduced commutative ring. Indag. Math. **27**, 727–748 (2016). https://doi.org/10.1016/j.indag.2016.01.007. https://arxiv.org/abs/1411.4410v2
9. Carlström, J.: Wheels - on division by zero. Math. Struct. Comput. Sci. **14**(1), 143–184 (2004). https://doi.org/10.1017/S0960129503004110
10. Fine, A.: Joint distributions, quantum correlations, and commuting observables. J. Math. Phys. **23**, 1306–1310 (1982). https://doi.org/10.1063/1.525514
11. Fine, A.: Hidden variables, joint probability, and the Bell inequalities. Phys. Rev. Lett. **48**(5), 291–295 (1982). https://doi.org/10.1103/PhysRevLett.48.291
12. Milne, P.: Bruno de Finetti and the logic of conditional events. Br. J. Philos. Sci. **48**, 195–232 (1997)
13. de Muynck, W.M.: Foundations of quantum mechanics, an empiricist approach. In: Fundamental Theories of Physics, vol. 127. Kluwer Academic Publishers (2002). Pre-publication version: http://www.phys.tue.nl/ktn/Wim/Kluwerbookprepversion.pdf
14. Padmanabhan, H.: A self-dual equational basis for Boolean algebras. Canad. Math. Bull. **26**(1), 9–12 (1983)
15. Rastall, P.: The Bell inequalities. Found. Phys. **13**(6), 555–570 (1983)
16. dos Reis, T.S., Gomide, W., Anderson, J.A.D.W.: Construction of the transreal numbers and algebraic transfields. IAENG Int. J. Appl. Math. **46**(1), 11–23 (2016). http://www.iaeng.org/IJAM/issues_v46/issue_1/IJAM_46_1_03.pdf

A Calculus of Virtually Timed Ambients

Einar Broch Johnsen, Martin Steffen, and Johanna Beate Stumpf(✉)

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

Abstract. A virtual machine, which is a software layer representing an execution environment, can be placed inside another virtual machine. As virtual machines at every level in a location hierarchy compete with other processes for processing time, the computing power of a virtual machine depends on its position in this hierarchy and may change if the virtual machine moves. These effects of nested virtualization motivate the calculus of virtually timed ambients, a formal model of hierarchical locations for execution with explicit resource provisioning, introduced in this paper. Resource provisioning in this model is based on virtual time slices as a local resource. To reason about timed behavior in this setting, weak timed bisimulation for virtually timed ambients is defined as an extension of bisimulation for mobile ambients. We show that the equivalence of contextual bisimulation and reduction barbed congruence is preserved by weak timed bisimulation. The calculus of virtually timed ambients is illustrated by examples.

1 Introduction

Virtualization technology enables the resources of an execution environment to be represented as a software layer, a so-called *virtual machine*. Application-level processes are agnostic to whether they run on such a virtual machine or directly on physical hardware. Since a virtual machine is a process, it can be executed on another virtual machine. Technologies such as VirtualBox, VMWare ESXi, Ravello HVX, Microsoft Hyper-V, and the open-source Xen hypervisor increasingly support running virtual machines inside each other in this way. This *nested virtualization*, originally introduced by Goldberg [12], is necessary to host virtual machines with operating systems which themselves support virtualization [5], such as Microsoft Windows 7 and Linux KVM. Nested virtualization has many uses, for example for end-user virtualization for guests, for development, and for deployment testing. Nested virtualization is also a crucial technology to support the hybrid cloud, as it enables virtual machines to migrate between different cloud providers [26].

To study the logical behavior of virtual machines in the context of nested virtualization, this paper develops a calculus of virtually timed ambients with explicit resource provisioning. Whereas previous work on process algebra with resources typically focusses on binary resources such as locks (e.g., [16,20]) and previous work on process algebra with time typically focusses on time-outs (e.g., [4,6,13,19,22]), time and resources in virtually timed ambients are *quantitative* notions: a process which gets *more resources* typically executes *faster*. Interpreting virtually timed ambients as locations for the deployment of processes, the resource requirements of processes executing at a location are matched by resources made available by the virtually timed ambient. The number of resources made available by the virtually timed ambient constitutes its computing power. This number is determined by the time slices it receives from its parent ambient. A virtually timed ambient that shares the time slices of its parent ambient with another process has less available time slices to execute its own processes. The model of resource provisioning in virtually timed ambients is inspired by Real-Time ABS [15], but extended to address nested virtualization in our calculus.

We call the corresponding time model for virtually timed ambients *virtual time*. Virtual time is provided to a virtually timed ambient by its parent ambient, similar to the time slices that an operating system provisions to its processes. When we consider many levels of nested virtualization, virtual time becomes a local notion of time which depends on a virtually timed ambient's position in the location hierarchy. Virtually timed ambients are mobile, reflecting that virtual machines may migrate between host virtual machines.

To formalize nested virtualization, notions of mobility and nesting are essential. The calculus of mobile ambients, originally developed by Cardelli and Gordon [8], captures processes executing at distributed locations in networks such as the Internet. Mobile ambients model both location mobility and nested locations, which makes this calculus well-suited as a starting point for our work. Combining these notions from the ambient calculus with the concepts of virtual time and resource provisioning, the calculus of virtually timed ambients can be seen as a model of nested virtualization, where different locations, barriers between locations, barrier crossing, and their relation to virtual time and resource provisioning are important, and where the number and position of virtually timed ambients available for processing tasks influences the overall processing time of a program. This allows the effects of, e.g., load balancing and scaling to be observed using weak timed bisimulation.

Contributions. To study the effects of nested virtualization, the main contributions of this paper can be summarized as follows:

- we define a calculus of *virtually timed ambients*, to the best of our knowledge the first process algebra capturing notions of virtual time and resource provisioning for nested virtualization;
- we define *weak timed bisimulation* for the calculus, and show that weak timed bisimulation is equivalent to reduction barbed congruence with time.

2 Virtually Timed Ambients

Mobile ambients [8] are located processes, arranged in a hierarchy which may change dynamically. Interpreting the location as a place of deployment, virtually timed ambients extend mobile ambients with notions of virtual time and resource consumption. The timed behavior depends on the one hand on the *local* timed behavior, but on the other hand on the placement or deployment of the component in the hierarchical ambient structure. Virtually timed ambients use a notion of time which is *local* to each ambient, but at the same time *relative* to the computing power of the surrounding ambients.

Before considering the details of virtually timed ambients, we briefly recall the syntax and basic ideas of mobile ambients [8]. This syntax, and the semantics we consider, is based on [18] and largely unchanged compared to [8]. The main difference to [8] lies in a separation of processes into two levels: *processes* and *systems*. Systems characterize the outermost layer of an ambient structure. This distinction is used to simplify proofs of bisimulation in Sect. 3.

Table 1. Syntax of the virtually timed ambient calculus.

	n	name
	`tick`	virtual time slice
Global systems:		
$G ::=$	$\mathbf{0}$	inactive system
	$G \mid G$	parallel composition
	$n[\textsc{Source} \mid M]$	virtually timed root ambient with a source clock
Timed systems:		
$M, N ::=$	$\mathbf{0}$	inactive system
	$M \mid N$	parallel composition
	$(\nu n)M$	restriction
	$n[\textsc{Clock} \mid P]$	virtually timed ambient with a local clock
Timed processes:		
$P, Q ::=$	$\mathbf{0}$	inactive process
	$P \mid Q$	parallel composition
	$(\nu n)P$	restriction
	$!C.P$	replication
	$C.P$	prefixing
	$n[\textsc{Clock} \mid P]$	virtually timed ambient with a local clock
Timed capabilities:		
$C ::=$	**in** n	can enter n and adjust the local clock there
	out n	can exit n and adjust the local clock on the outside
	open n	can open n and adjust own local clock
	consume	consumes one resource

Mobile Ambients. Mobile ambients [8], originally introduced to represent "administrative domains" for processes, are defined as follows. The inactive process $\mathbf{0}$ does nothing. The parallel composition $P \mid Q$ allows both processes P and Q to proceed concurrently, where the binary operator $\mid$ is commutative and associative. The restriction operator $(\nu m)P$ creates a new and unique name with process P as its scope. In the calculus, the administrative domains for processes, called *ambients*, are represented by names. A process P located in an ambient named m is written $m[P]$. Ambients can be nested, and the nesting structure can change dynamically. A change of the nesting structure is specified by prefixing a process with a *capability*. There are 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 with its own ambient; e.g., $k[\mathit{in}\, n.P] \mid n[Q] \rightarrowtriangle n[k[P] \mid Q]$. The output capability $\mathit{out}\, n$ enables an ambient to leave its surrounding (or parental) ambient n; e.g., $n[k[\mathit{out}\, n.P] \mid Q] \rightarrowtriangle k[P] \mid n[Q]$. The open capability $\mathit{open}\, n$ allows an ambient named n at the same level as the capability to be opened; e.g., $k[\mathit{open}\, n.P \mid n[Q]] \rightarrowtriangle k[P \mid Q]$. The semantics is given as a reduction semantics which combines structural congruence with reduction rules.

Virtual Time and Local Clocks. Virtually timed ambients combine timed processes and timed capabilities with the features of the calculus for mobile ambients summarized above. In Table 1 we can see that in the calculus of virtually timed ambients every closed system of ambients must be contained in a root ambient with a *source clock* triggering the clocks of the local subambients recursively. Timed systems and processes are defined analogously to systems and processes in mobile ambients, with the difference that each virtually timed ambient contains a *local clock* and other virtually timed ambients or processes. The timed capabilities of virtually timed ambients extend the capabilities of mobile ambients with additional time management, explained below. In order to define computing power, a capacity **consume** for resource consumption, of processes is added.

Definition 1 (Virtually timed ambients). *Virtually timed ambients are defined by the syntax in Table 1.*

The semantics is given as a reduction semantics which combines structural congruence with reduction rules and can be found in Tables 2 and 3. In Table 3 we make use of the notion of *observables*.

Definition 2 (Observables). *An* observable*, also known as a* barb*, is the presence of a top-level ambient whose name is not restricted. The observation predicate $P \downarrow n$ captures this observable. Thus, $P \downarrow n$ if $P \equiv (\nu \bar{m})(n[P_1] \mid P_2)$, where $n \notin \{\bar{m}\}$. We write $P \Downarrow n$ if there exists P' such that $P \Rightarrow P'$ and $P' \downarrow n$.*

To represent the outlined time model the local clock contained in each virtually timed ambient is responsible for triggering timed behavior and local resource consumption. Each time slice emitted by a local clock triggers the clock of one of

Table 2. Reduction rules.

$P \rightharpoonup Q \Rightarrow (\nu n)P \rightharpoonup (\nu n)Q$	(R-Res)
$P \rightharpoonup Q \Rightarrow P \mid R \rightharpoonup Q \mid R$	(R-Par)
$P \rightharpoonup Q \Rightarrow n[P] \rightharpoonup n[Q]$	(R-Amb)
$P' \equiv P, P \rightharpoonup Q, Q \equiv Q' \Rightarrow P' \rightharpoonup Q'$	(R-Red $\equiv$)

its subambients in a round-robin way or is consumed by a process as a resource. This corresponds to a simple form of fair, *preemptive scheduling*, which makes the system's behavior sensitive to the number of co-located virtually timed ambients and resource consuming processes. Clocks have a *speed*, interpreted *relative* to the speed of the surrounding virtually timed ambient. The speed of a clock is given by the pair (p, q), where p is the number of local time slices emitted for a number q of time slices received from the surrounding ambient, $p, q \in \mathbb{N}^0$. Thus, time in a nested ambient is relative to the global time, and depends on the speed of the clocks of the ambients in which it is contained and on its number of siblings. The speed of the source clocks is defined as $(n, 0), n \in \mathbb{N}^0$, as the sources do not need any input, while for the speed of a local clock it holds that an input of $q = 0$ is only valid if $p = 0$, too. As those ambients with speed $(0, 0)$ do not require any times slices from their parental ambient and do not show any timed behavior, they are not considered *time consuming.* However, processes which are prefixed with the resource consumption capability **consume**.P are considered time consuming. Note, that mobile ambients can be represented as virtually timed ambients with a clock with speed $(0, 0)$.

Definition 3 (Local clocks). *A local clock contains a* counter *to record the number of received time slices, its own* speed*, and two* sets*:*

$$\text{Clock}\{counter, (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}.$$

The first set contains the names of time consuming processes running in the ambient as well as time consuming virtually timed subambients in the surrounding ambient which have not yet received a time slice in the current cycle and the second set those which have.

When a clock receives a time slice, denoted `tick`, from its surrounding ambient, one of the following actions occurs: If $counter + 1 < q$, then the clock records this time slice and continues waiting (i.e., $\text{Clock}\{counter := counter + 1, (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}$); if $counter + 1 = q$, then the input number is reached, the counter is set to 0 and the clock emits time slices to p subambients of the first set and puts them in the second set (i.e., $:= \text{Clock}\{counter := 0, (p, q), \{a_{p+1} \ldots, a_k\}, \{a_{k+1}, \ldots a_n, a_1, a_2, \ldots, a_p\}\}$). As soon as the first of the two sets is empty, the first and second set are switched. Thus, no ambient receives a second time slice before every other subambient has received the first one. In the sequel, we omit the representation of the counter and the sets of subambients. For a better overview in the examples we denote the speed of the clocks as superscript $\text{Clock}^{p,q}$. If an ambient is not time consuming, i.e., it has a clock with speed $(0, 0)$, we do not mention the clock. For

Table 3. Timed reduction rules for timed capabilities, where a_k and b_i are time consuming virtually timed ambients and processes in R and Q, respectively.

$$\frac{\begin{array}{l}\textsc{Clock}_k = \textsc{Clock}\{c_k, (p_k, q_k), \{m, n\}, \emptyset\}\\ \textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, \dots, a_k\}, \{a_{k+1}, \dots a_n\}\}\\ \textsc{Clock}_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \dots, b_i\}, \{b_{i+1}, \dots b_j\}\}\\ \textsc{Clock}^*_k = \textsc{Clock}\{c_k, (p_k, q_k), \{m\}, \emptyset\}\\ \textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, \dots, a_k, n\}, \{a_{k+1}, \dots a_n\}\}\\ \textsc{Clock}^*_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \dots, b_i\} \cup \bar{o}, \{b_{i+1}, \dots b_j\}\}\end{array}}{\begin{array}{l}k[\textsc{Clock}_k \mid n[\textsc{Clock}_n \mid \mathbf{in}\ m.P \mid Q] \mid m[\textsc{Clock}_m \mid R]]\\ \quad \rightarrowtriangle k[\textsc{Clock}^*_k \mid m[\textsc{Clock}^*_m \mid R \mid n[\textsc{Clock}_n \mid P \mid Q]]]\end{array}} \quad (\text{TR-}\textsc{In})$$

$$\frac{\begin{array}{l}\textsc{Clock}_k = \textsc{Clock}\{c_k, (p_k, q_k), \emptyset, \{m\}\}\\ \textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \dots, a_k, n\}, \{a_{k+1}, \dots a_n\}\}\\ \textsc{Clock}_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \dots, b_i\}, \{b_{i+1}, \dots b_j\}\}\\ \textsc{Clock}^*_k = \textsc{Clock}\{c_k, (p_k, q_k), \{n\}, \{m\}\}\\ \textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \dots, a_k\}, \{a_{k+1}, \dots a_n\}\}\\ \textsc{Clock}^*_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \dots, b_i\} \cup \bar{o}, \{b_{i+1}, \dots b_j\}\}\end{array}}{\begin{array}{l}k[\textsc{Clock}_k \mid m[\textsc{Clock}_m \mid n[\textsc{Clock}_n \mid \mathbf{out}\ m.P \mid Q] \mid R]]\\ \quad \rightarrowtriangle k[\textsc{Clock}^*_k \mid n[\textsc{Clock}_n \mid P \mid Q] \mid m[\textsc{Clock}^*_m \mid R]]\end{array}} \quad (\text{TR-}\textsc{Out})$$

$$\frac{\begin{array}{l}\textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{n\}, \emptyset\}\\ \textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \dots, a_n\} \cup \bar{o}, \emptyset\}\end{array}}{m[\textsc{Clock}_m \mid \mathbf{open}\ n.P \mid n[\textsc{Clock}_n \mid R] \mid Q] \rightarrowtriangle m[\textsc{Clock}^*_m \mid P \mid R \mid Q]} \quad (\text{TR-}\textsc{Open})$$

$$\frac{\begin{array}{l}\textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \dots, a_n\}, \emptyset\},\ p_m > 0\\ \textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \dots, a_n, \mathbf{consume}.P\}, \emptyset\}\end{array}}{m[\textsc{Clock}_m \mid \mathbf{consume}.P \mid R] \rightarrowtriangle m[\textsc{Clock}^*_m \mid R]} \quad (\text{TR-}\textsc{Res})$$

$$\frac{\begin{array}{l}\textsc{Clock} = \textsc{Clock}\{c, (p, q), \{a_1, a_2, \dots, a_k\}, \{a_{k+1}, \dots a_n\}\},\ c + 1 < q\\ \textsc{Clock}^* = \textsc{Clock}\{c + 1, (p, q), \{a_1, a_2, \dots, a_k\}, \{a_{k+1}, \dots a_n\}\}\end{array}}{m[\mathtt{tick} \mid \textsc{Clock} \mid R] \rightarrowtriangle m[\textsc{Clock}^* \mid R]} \quad (\text{TR-}\textsc{Tick}_1)$$

$$\frac{\textsc{Clock} = \textsc{Clock}\{c, (p, q), \{a_1, a_2, \dots, a_k\}, \{a_{k+1}, \dots a_n\}\},\ c + 1 = q}{m[\mathtt{tick} \mid \textsc{Clock} \mid R] \rightarrowtriangle m[\mathrm{RR}(\textsc{Clock} \mid R)]} \quad (\text{TR-}\textsc{Tick}_2)$$

$$\frac{\textsc{Source} = \textsc{Source}\{0, (n, 0), \{a_1, a_2, \dots, a_k\}, \{a_{k+1}, \dots a_n\}\}}{\textsc{Source} \mid R \rightarrowtriangle \mathrm{RR}(\textsc{Source} \mid R)} \quad (\text{TR-}\textsc{Source})$$

RR: Round-based distribution function

```
input: $\textsc{Clock}\{counter, (p,q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots, a_n\}\} \mid R$
$S := \{a_1, a_2, \ldots, a_k\}$
$T := \{a_{k+1}, \ldots, a_n\}$
**if** $S = \emptyset$ **then**
**return** $\textsc{Clock}\{0, (p,q), \emptyset, \emptyset\}$
**else**
    **while** $p \geq |S|$ **do**
        **for all** $a_i \in S$ **do**
            **if** $a_i = \mathbf{consume}.P$ **then** $S := S \setminus a_i$; $R := R \mid P$
                Let $P \equiv (\nu\widetilde{n})P'$, $P'$ is $\nu$-binder free, $\overline{o} = \{o \mid P' \downarrow o\}$:
                $S := S \cup \overline{o}$
            **else** $R := a_1 \mid \cdots \mid a_i[\texttt{tick} \mid \cdots] \mid \cdots \mid a_n$
            **end if**
        **end for**
        $p := p - |S|$; $S := S \cup T$; $T := \emptyset$
    **end while**
    Choose a subset $S' \subset S$ such that $|S'| = p$.
    **for all** $a_i \in S'$ **do**
        **if** $a_i = \mathbf{consume}.P$ **then** $S' := S \setminus a_i$, $R := R \mid P$
            Let $P \equiv (\nu\widetilde{n})P'$, $P'$ is $\nu$-binder free, $\overline{o} = \{o \mid P' \downarrow o\}$:
            $S := S \cup \overline{o}$
        **else** $R := a_1 \mid \cdots \mid a_i[\texttt{tick} \mid \cdots] \mid \cdots \mid a_n$
        **end if**
    **end for**
    $S := S \setminus S'$; $T := T \cup S'$
**end if**
**return** $\textsc{Clock}\{0, (p,q), S, T\} \mid R$
```

actions which do not require time we assume *maximal progress*, meaning that actions which can be executed immediately will not be delayed.

Timed Capabilities. The timed capabilities **in** n, **out** n, and **open** n enable virtually timed ambients to move in a timed system. When moving virtually timed ambients, we must consider that the clocks need to know about their current subambients, therefore their list of subambients need to be adjusted.

We now explain the reduction rules for virtually timed ambients, which are given in Tables 2 and 3. Observe that if we would not adjust the clocks then a moving subambient would not receive time slices from its new parental clock. In (TR-In) and (TR-Out), the clocks of the old and new parental ambient of the moving ambient have to be updated. Here we let $P \equiv (\nu\widetilde{n})P'$, where P' is ν-binder free, such that $\overline{o} = \{o \mid P' \downarrow o\}$. In (TR-Open) the clock of the opening ambient itself is updated. Note also that here the clock of the opened ambient is deleted. For virtually timed ambients with a clock with speed $(0,0)$, the timed capabilities are equivalent to the capabilities for mobile ambients, as ambients, which are not time consuming, are not considered in the time management of the clocks. In (TR-Res) the time consuming process is moved into the clock, where it awaits the distribution of a time slice as resource before it can continue. This reduction can only happen in virtually timed ambients with $p > 0$, meaning ambients which actually emit resources. In (TR-Tick$_1$) the required number q_m of input time slices to trigger the local clock is not reached, thus the incoming time slice, denoted as `tick`, is only registered in the counter. In (TR-Tick$_2$) the local clock releases p time slices to its subambients and potentially to time consuming process. This is denoted with the function RR for round based distribution of time slices. The function takes as input the distributing Clock and distributes time slices `tick` to the subambients and processes in the given sets, thereby adjusting the sets of remaining and of served ambients. The source clocks Source can reduce without parental time slices as given in (TR-Source). The following example shows the encoding of a system with a load balancer in virtually timed ambients.

Example 1. A system with a load balancer can be defined as follows:

load balancer system: $(\nu\ lb, a, b)\ lbs[\textsc{Clock}^{2,1} \mid lb \mid a \mid b]$

incoming request: $request[P.done\_signal \mid in\ lbs.enter\_signal.\mathbf{open}\ move]$

load balancer: $lb[!\mathbf{open}\ start.wait\_for\_enter.\mathbf{open}\ lock_a.$
$wait\_for\_enter.\mathbf{open}\ lock_b.start[]\ |!lock_a[x[]] \mid$
$!lock_b[y[]]\ |!(\mathbf{open}\ x.move[\mathbf{out}\ lb.\mathbf{in}\ request.\mathbf{in}\ a] \mid$
$\mathbf{open}\ y.move[\mathbf{out}\ lb.\mathbf{in}\ request.\mathbf{in}\ b])]$

ambient a: $a[\textsc{Clock}^{1,1}\ |!\mathbf{open}\ request.wait\_for\_done.done[\mathbf{out}\ a.\mathbf{out}\ lbs]]$

ambient b: $b[\textsc{Clock}^{1,1}\ |!\mathbf{open}\ request.wait\_for\_done.done[\mathbf{out}\ b.\mathbf{out}\ lbs]]$.

Here, the untimed load balancer creates a *move* ambient which moves incoming requests alternately into the virtually timed ambients a and b. For each time slice it receives from the source clock of the surrounding root ambient, the local clock of *lbs* distributes two time slices. Therefore, both subambients a and b receive one time slice. When a request has been executed, it releases an ambient *done* which emerges to the outside of the system and becomes observable.

Resource Consumption. Processes expend the processing power of the ambient they are contained in by consuming the local time slices as resources. Thus, time consuming processes and time consuming subambients in a virtually timed ambient compete for the same resource. The consumption of a computing resource is defined as the capability **consume**. An ambient with a higher local clock speed produces more time slices and therefore also more resources for each parental time slice, which in turn allows more work to be done for each parental time slice. We consider resource consumption by a request which was sent to the system of Example 1.

Example 2. Consider the virtually timed system with a load balancer from Example 1, with an incoming request.

$lbs[\cdots] \mid$

$request[\mathbf{consume}.\mathbf{consume}.done\_signal \mid \mathbf{in}\ lbs.enter\_signal.\mathbf{open}\ move]$

The request enters the system and is transferred by the load balancer into a, where it is opened during the reduction and awaits resource consumption. After one time signal of the source clock, the virtually timed ambient a emits one resource, which is consumed by the request:

$$\begin{aligned} a[\textsc{Clock}^{1,1}\ |!\mathit{open\ request.wait\_for\_done.done}[\mathbf{out}\ a.\mathbf{out}\ \mathit{lbs}] \\ |\ \mathit{wait\_for\_done.done}[\mathbf{out}\ a.\mathbf{out}\ \mathit{lbs}] \mid \mathbf{consume}.\mathit{done\_signal}]. \end{aligned}$$

After another time signal from the source clock the ambient *done* can emerge:

$$a[\textsc{Clock}^{1,1}\ |!\mathit{open\ request.wait\_for\_done.done}[\mathbf{out}\ a.\mathit{out\ lbs}]] \mid \mathit{done}[\mathbf{out}\ \mathit{lbs}].$$

Table 4. Rules for timed labeled transition systems, where in (Co-Enter) given $\textsc{Clock} = \textsc{Clock}\{c_k, (p_k, q_k), \{a_1, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}$ the updated clock is denoted by $\textsc{Clock}^* = \textsc{Clock}\{c_k, (p_k, q_k), \{a_1, \ldots, a_k, n\}, \{a_{k+1}, \ldots a_n\}\}$ as seen in Table 3.

$(\nu\widetilde{m})(m[\textsc{Clock} \mid \mathbf{in}\ n.P \mid Q] \mid M), m \in \widetilde{m}$ $\xrightarrow{*.\mathtt{enter}\_n} (\nu\widetilde{m})(n[m[(\textsc{Clock} \mid P) \mid Q] \mid \circ\ ] \mid M)$	(Enter Shh)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid \mathbf{in}\ n.P \mid Q] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\mathtt{enter}\_n} (\nu\widetilde{m})(n[k[(\textsc{Clock} \mid P) \mid Q] \mid \circ\ ] \mid M)$	(Enter)
$(\nu\widetilde{m})(m[\textsc{Clock} \mid \mathbf{out}\ n.P \mid Q] \mid M), m \in \widetilde{m}$ $\xrightarrow{*.\mathtt{exit}\_n} (\nu\widetilde{m})(m[(\textsc{Clock} \mid P) \mid Q] \mid n[M \mid \circ\ ])$	(Exit Shh)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid \mathbf{out}\ n.P \mid Q] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\mathtt{exit}\_n} (\nu\widetilde{m})(k[(\textsc{Clock} \mid P) \mid Q] \mid n[M \mid \circ\ ])$	(Exit)
$(\nu\widetilde{m})(k[(\textsc{Clock} \mid P)] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\overline{\mathtt{enter}}\_n} (\nu\widetilde{m})(k[\textsc{Clock}^* \mid n[\circ] \mid P] \mid M)$	(Co-Enter)
$(\nu\widetilde{m})(k[(\textsc{Clock} \mid P)] \mid M)$ $\xrightarrow{n.\mathtt{open}\_k} n[\ \circ \mid (\nu\widetilde{m})(P \mid M)]$	(Open)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid Q] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\mathtt{tick}} (\nu\widetilde{m})(k[\textsc{Clock} \mid \mathtt{tick} \mid Q] \mid M)$	(Tick)

3 Comparing Virtually Timed Ambients

When comparing virtually timed ambients, e.g., in terms of bisimulation, we need to consider time as a factor. For this purpose, we define a labeled transition system which contains an observable action capturing global time steps.

Weak Bisimulation for Virtually Timed Ambients. The reduction semantics from Sect. 2 captures the behavior of *closed* or global systems. To define bisimulation, we first formalize an *open* version of the operational semantics, using a labelled transition relation. To express interaction with a surrounding *context* in the open setting, transitions will have *labels* which capture interaction with an environment.

Definition 4 (Labels). *Let the set of labels Lab, with typical element α, be given as follows: $\alpha \in Lab ::= \tau \,|\, k.\mathtt{enter\_}n \,|\, k.\mathtt{exit\_}n \,|\, k.\overline{\mathtt{enter}}\mathtt{\_}n \,|\, n.\mathtt{open\_}k \,|\, *.\mathtt{exit\_}n \,|\, *.\mathtt{enter\_}n \,|\, k.\mathtt{tick}$, where k and n represent names of ambients. The* internal label *is τ, the rest are called* observable *labels. We refer to labels of the forms $*.\mathtt{exit\_}n$ and $*.\mathtt{enter\_}n$ as* anonymous *and other labels as* non-anonymous*, and let the* untimed *labels exclude the tick labels.*

The definition of the labels is based on the formalization for mobile ambients in [18]. The corresponding timed labelled transition system is given in Table 4. In the rules (Enter) and (Exit), an ambient k enters, respectively exits, from an ambient n provided by the environment. The rules (Enter Shh) and (Exit Shh) model the same behavior for ambients with private names. In the rule (Co-Enter), an ambient n provided by the environment enters an ambient k of the process. In the rule (Open), the environment provides an ambient n in which the ambient k of the process is opened. In the rule (Tick), $M \xrightarrow{k.\mathtt{tick}} M'$ expresses that the top-level ambient k of the system M receives one time slice $\mathtt{tick}$ from the source clock. Note that the transition semantics contains the symbol $\circ$ as a placeholder variable for the body of the context ambient, containing an arbitrary process and an arbitrary local clock. The process $P := \circ$ must be instantiated in the bisimulation. The replacement of the placeholder by a process and local clock is written as $P \bullet (\textsc{Clock} \mid Q)$ and defined as expected. The reduction semantics of a process can be encoded in the labelled transition system, because a reduction step can be seen as an interaction with an empty context. We are interested in bisimulations that abstract from τ-actions and use the notion of *weak actions*; let $\Rightarrow$ denote the reflexive and transitive closure of $\xrightarrow{\tau}$, let $\overset{\alpha}{\Rightarrow}$ denote $\Rightarrow\xrightarrow{\alpha}\Rightarrow$, and let $\overset{\hat{\alpha}}{\Rightarrow}$ denote $\Rightarrow$ if $\alpha = \tau$ and $\overset{\alpha}{\Rightarrow}$ otherwise. An example of a system consuming one parental $\mathtt{tick}$ and performing the subsequent τ-actions is given below:

Example 3. We reconsider Example 2. After one time signal of the source clock, the subambient a emits one resource, which is consumed by the request:

$$\begin{aligned} a[P'] := a[&\textsc{Clock}^{1,1} \mid !\mathbf{open}\ request.wait\_for\_done.done[\mathbf{out}\ a.\mathbf{out}\ lbs] \\ &\mid wait\_for\_done.done[\mathbf{out}\ a.\mathbf{out}\ lbs] \mid \mathbf{consume}.done\_signal].\end{aligned}$$

After another time signal from the source clock and some internal τ steps the ambient *done* can emerge, thus here it holds that:

$$\begin{aligned} &lbs[\textsc{Clock}^{2,1} \mid lb \mid a[P'] \mid b] \xrightarrow{lbs.\mathtt{tick}} lbs[\textsc{Clock}^{2,1} \mid \mathtt{tick} \mid lb \mid a[P'] \mid b] \\ &\Rightarrow lbs[\textsc{Clock}^{2,1} \mid lb \mid a \mid b] \mid done[].\end{aligned}$$

Considering timed systems as defined in Table 1, we can now define weak timed bisimulation as follows:

Definition 5 (Weak timed bisimulation). *A symmetric relation $\mathcal{R}$ over timed systems is a* weak timed bisimulation *if $M \,\mathcal{R}\, N$ implies:*

- *if $M \xrightarrow{\alpha} M'$, $\alpha \in \{\tau, k.\mathtt{enter}\_n, k.\mathtt{exit}\_n, k.\overline{\mathtt{enter}}\_n, n.\mathtt{open}\_k, k.\mathtt{tick}\}$, then there is a system N' such that $N \overset{\hat{\alpha}}{\Rightarrow} N'$ and for all clocks* Clock *and processes P it holds that $M' \bullet(\text{Clock} \mid P) \;\mathcal{R}\; N' \bullet(\text{Clock} \mid P)$;*
- *if $M \xrightarrow{*.\mathtt{enter}\_n} M'$ then there is a system N' such that $N \mid n[\circ] \Rightarrow N'$ and for all clocks* Clock *and processes P it holds that $M' \bullet(\text{Clock} \mid P)$ $\mathcal{R}$ $N' \bullet(\text{Clock} \mid P)$;*
- *if $M \xrightarrow{*.\mathtt{exit}\_n} M'$ then there is a system N' such that $n[\circ \mid N] \Rightarrow N'$ and for all clocks* Clock *and processes P it holds that $M' \bullet(\text{Clock} \mid P)$ $\mathcal{R}$ $N' \bullet(\text{Clock} \mid P)$.*

Systems M and N are *weakly timed bisimilar*, written $M \approx^t N$, if $M \mathcal{R} N$ for some weak timed bisimulation $\mathcal{R}$. If two systems are weakly timed bisimilar in a timed setting where we observe the ticking of the source clock, then it follows from the definition of weak timed bisimulation that they are weakly bisimilar in a setting where we do not observe the ticking of the clocks and instead interpret all `tick`-actions as τ-actions.

Lemma 1 (Consistency). *$M \approx^t N$ implies that M and N are weakly bisimilar, $M \approx N$.*

Note that for virtually timed ambients which are not time consuming, i.e. with a speed of $(0, 0)$, weak timed bisimulation and weak bisimulation coincide.

Example 4. We compare the behavior of the system *lbs* from Example 1 with a second system called lbs_2, which is defined as follows:

load balancer system: $(\nu\ lb, a)\ lbs_2[\text{Clock}^{1,1} \mid lb \mid a]$

incoming request: $request[P.done\_signal \mid \mathbf{in}\ lbs_2.enter\_signal.\mathbf{open}\ move]$

load balancer: $lb[\ !wait\_for\_enter.move[\mathbf{out}\ lb.\mathbf{in}\ request.\mathbf{in}\ a]]$

ambient a: $a[\text{Clock}^{2,1} \mid !\mathbf{open}\ request.wait\_for\_done.done[\mathbf{out}\ a.\mathbf{out}\ lbs_2]]$

In contrast to *lbs*, system lbs_2 only contains one virtually timed ambient, which receives all requests. If we do not observe time, the systems behave the same, $lbs \approx lbs_2$, as they both answer requests by emitting an observable *done*-signal. However, the systems are not weakly timed bisimilar, $lbs \not\approx^t lbs_2$.

Reduction Barbed Congruence. Honda and Yoshida's method [14] can be used to define weak reduction barbed congruence for mobile ambients [18]. This approach can be extended to virtually timed ambients.

Definition 6 (Reduction barbed congruence over timed systems). Reduction barbed congruence over timed systems $\simeq_s$ *is the largest symmetrical relation over timed systems which is preserved by all system contexts, reduction closed and barb preserving.*

Theorem 1. *Weak timed bisimilarity and reduction barbed congruence coincide over timed systems.*

We first introduce some concepts related to reduction barbed congruence over timed systems before approaching the proof.

Definition 7. *A* context *is a process with a hole. A* system context *is a context that transforms systems into systems. System contexts are generated by the following grammar:* $\mathcal{C}[-] := - \mid (\mathcal{C}[-] \mid M) \mid (M \mid \mathcal{C}[-]) \mid (\nu n)\mathcal{C}[-] \mid n[\mathcal{C}[-] \mid P] \mid n[P \mid \mathcal{C}[-]]$*, where* M *is an arbitrary system and* P *is an arbitrary process.*

Definition 8. *A relation* $\mathcal{R}$ *is* preserved by system contexts *if* $M\mathcal{R}N$ *implies* $\mathcal{C}[M]\mathcal{R}\mathcal{C}[N]$ *for all system contexts* $\mathcal{C}[-]$.

Theorem 2. *Weak timed bisimilarity is preserved by system contexts.*

Proof. We extend the proof of Merro and Zappa Nardelli [18] for the $k.\texttt{tick}$ action. As the $k.\texttt{tick}$ action can be seen as a special case of the $k.\overline{\texttt{enter\_n}}$ action, the same proof method can be used here.

Definition 9. *A relation* $\mathcal{R}$ *over processes is* barb preserving *if* $P\ \mathcal{R}\ Q$ *and* $P \downarrow n$ *implies* $Q \Downarrow n$.

Weak timed bisimilarity carries over the properties of being reduction closed and barb-preserving from weak bismilarity for mobile ambients [18]. Thus, weak timed bisimilarity is contained in reduction barbed congruence, $\approx^t \subseteq \simeq_s$. To show that the other direction holds as well, we need to define a system context that observes the action $k.\texttt{tick}$. Contexts to observe the other actions of the labeled transition system are defined in [18]. Again we can consider $k.\texttt{tick}$ as a special case of the $k.\overline{\texttt{enter\_n}}$ action and can therefore define the context equivalently:

$$\mathcal{C}_{\texttt{tick}}[-] = (\nu a, b)a[\textbf{in}\ n.\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\textbf{out}\ n.done[\textbf{out}\ b]]]] \mid -.$$

Note that that all top level ambients of the system which is entered in the context will receive the time via τ-actions. With this context, we can adjust Lemma 3.8 of Merro and Zappa Nardelli [18] to virtually timed ambients.

Lemma 2. *Let* $\alpha \in \{k.\texttt{enter\_}n, k.\texttt{exit\_}n.k.\overline{\texttt{enter\_n}}, n.\texttt{open\_k}, n.\texttt{tick}\}$ *and let* M *be a system. For all clocks* Clock *and processes* P*, if* $M \xrightarrow{\alpha} M'$ *then* $\mathcal{C}_\alpha[M] \bullet (\textsc{Clock} \mid P) \Rightarrow\simeq_s (M' \bullet (\textsc{Clock} \mid P)) \mid done[]$.

Proof. We will only consider $\alpha = n.\mathtt{tick}$. All other cases are detailed in [18]. Let P be a process. We know that $M \xrightarrow{n.\mathtt{tick}} M'$. Then $M' \equiv n[\mathtt{tick} \mid Q] \mid M''$.

$$\begin{aligned}
&\mathcal{C}_{\mathtt{tick}}[M] \bullet (\textsc{Clock} \mid P)\\
\equiv&((\nu a,b)a[\mathbf{in}\ n.\mathtt{tick}[\mathbf{out}\ a.b[\mathbf{out}\ \mathtt{tick}.\mathbf{out}\ n.\mathit{done}[\mathbf{out}\ b]])]] \mid M)\\
&\bullet (\textsc{Clock} \mid P)\\
\twoheadrightarrow&((\nu a,b)n[a[\mathtt{tick}[\mathbf{out}\ a.b[\mathbf{out}\ \mathtt{tick}.\mathbf{out}\ n.\mathit{done}[\mathbf{out}\ b]]]] \mid Q] \mid M'')\\
&\bullet (\textsc{Clock} \mid P)\\
\twoheadrightarrow&((\nu a,b)n[a[] \mid \mathtt{tick}[b[\mathbf{out}\ \mathtt{tick}.\mathbf{out}\ n.\mathit{done}[\mathbf{out}\ b]]] \mid Q] \mid M'') \bullet (\textsc{Clock} \mid P)\\
\twoheadrightarrow&((\nu a,b)n[a[] \mid \mathtt{tick} \mid Q] \mid b[\mathit{done}[\mathbf{out}\ b]] \mid M'') \bullet (\textsc{Clock} \mid P)\\
\twoheadrightarrow&((\nu a,b)n[a[] \mid \mathtt{tick} \mid Q] \mid b[] \mid \mathit{done}[] \mid M'') \bullet (\textsc{Clock} \mid P)\\
\equiv&(M' \bullet (\textsc{Clock} \mid P)) \mid \mathit{done}[].
\end{aligned}$$

To prove the correspondence between actions α and their contexts $\mathcal{C}_\alpha[-]$, we have to prove the converse of the lemma above as well. The proof of this result uses particular contexts $\mathtt{spy}_\alpha\langle i,j,-\rangle$ as a technical tool to guarantee that the process P provided by the environment does not perform any action. We define the context $\mathtt{spy}_{\mathtt{tick}}\langle i,j,-\rangle := (i[] \mid -) \oplus (j[] \mid -)$, where i,j are fresh ambient names and $\oplus$ is an encoding of internal choice in the ambient calculus. We can now adjust the proof of Lemma 3.12 in [18], making use of Lemmas 3.9, 3.10 and 3.11 in [18].

Lemma 3. *Let* $\alpha \in \{k.\mathtt{enter\_}n, k.\mathtt{exit\_}n.k.\overline{\mathtt{enter\_n}}, n.\mathtt{open\_k}, n.\mathtt{tick}\}$ *and let* M *be a system. Let* i,j *be fresh names for* M*. For all processes* P *with* $\{i,j\} \cap \mathit{fn}(P) = \emptyset$*, if* $C_\alpha[M] \bullet (\textsc{Clock} \mid \mathit{spy}_\alpha\langle i,j,P\rangle) \Rightarrow\equiv N \mid \mathit{done}[]$ *and* $N \Downarrow_{i,j}$ *there exists a system* M' *such that* $M \overset{\alpha}{\Rightarrow} M'$ *and* $M' \bullet (\textsc{Clock} \mid \mathit{spy}_\alpha\langle i,j,P\rangle) \simeq_s N$*.*

Proof. We consider the case of $\alpha = n.\mathtt{tick}$, all other cases are detailed in [18].

$$\begin{aligned}
&\mathcal{C}_\alpha[M] \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i,j,P\rangle)\\
\equiv\ &(\nu a,b)a[\mathbf{in}\ n.\mathtt{tick}[\mathbf{out}\ a.b[\mathbf{out}\ \mathtt{tick}.\mathbf{out}\ n.\mathit{done}[\mathbf{out}\ b]]]] \mid M\\
&\quad\bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i,j,P\rangle)\\
\twoheadrightarrow\ &(\nu a,b)n[a[] \mid \mathtt{tick} \mid Q] \mid b[] \mid M''\\
&\quad\bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i,j,P\rangle) \mid \mathit{done}[] \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i,j,P\rangle)\\
\equiv\ &M' \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i,j,P\rangle) \mid \mathit{done}[] \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i,j,P\rangle)\\
\equiv\ &N \mid \mathit{done}[]\ .
\end{aligned}$$

We conclude that $M \overset{n.\mathtt{tick}}{\Longrightarrow} M'$. It holds that $M' \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i,j,P\rangle) \simeq_s N$.

It now follows using Theorem 3.14 in [18] that reduction barbed congruence is contained in weak timed bisimilarity, i.e. $M \approx^t N$ iff $M \simeq_s N$.

4 Related Work

As our work is building on the ambient calculus, we focus on timed process algebras based on the closely related π-calculus [23], which originated from CCS. Related work building on ACP and CSP can be found in, e.g., [3,4,22].

A special action for time was introduced in an early timed extension [19] of CCS, without committing to a discrete or continuous time domain. A related idling action that needs exactly one time unit to be processed is proposed in [13], where time is discrete and processes synchronized via a global clock. The notion of local time proposed for CCS in [24] is closer to our local clocks, but uses a time-out oriented model. Timers, which are introduced to express the possibility of a time-out and are controlled by a global clock, have been studied for mobile ambients [1,2,10]. Modeling time-out is a straightforward extension of our work. However, the high-level idea of these works is very different from ours: they all focus on speed as the *duration* of processes, while in our approach with local clocks speed describes the *processing power* of a virtually timed ambient.

Cardelli and Gordon defined a labeled transition system for mobile ambients [9], but no bisimulation. A bisimulation relation for a restricted version of mobile ambients, called mobile safe ambients, is defined in [17] and provides the basis for later work. Barbed congruence for the same fragment of mobile ambients is defined in [25]. It is shown in [11] that name matching reduction barbed congruence and bisimulation coincide in the π-calculus. A bisimulation relation with contextual labels for the ambient calculus is defined in [21], but this approach is not suitable for providing a simple proof method. A labelled bisimulation for mobile ambients is defined by Merro and Zappa Nardelli [18], who prove that this bisimulation is equivalent to reduction barbed congruence and develop up-to-proof techniques. The weak timed bisimulation defined in this paper is a conservative extension of this approach.

A process algebra with resources as primitives is studied in [16], in which priorities are used to make processes sensitive to scheduling. A similar approach with explicit scheduling is studied in [20]. In contrast to these works, scheduling in our approach is determined by the implementation of the resource distribution. A core language for defining cloud services and their deployment on cloud platforms is introduced in [7] to enable statically safe service composition and custom implementations of cloud services. However, in contrast to our approach, time and performance are not taken into consideration.

5 Concluding Remarks

We believe that virtualization opens for new and interesting foundational models of computation by explicitly emphasizing deployment and resource allocation. This paper introduces virtually timed ambients, a formal model of hierarchical locations of execution with explicit resource provisioning. In the proposed model resource provisioning is based on virtual time, a local notion of time reminiscent of time slices for virtual machines in the context of nested virtualization. This way, the computing power of an ambient depends on its location in the deployment hierarchy. To reason about timed behavior in this setting, we define weak timed bisimulation for virtually timed ambients as a conservative extension of bisimulation for mobile ambients, and show that the equivalence of bisimulation to reduction barbed congruence is preserved by this extension.

The calculus of virtually timed ambients opens up opportunities for further interesting research questions. One line of research is in statically controlling resource management, for example by means of behavioral types. Another line of research is in dynamically controlling resource management by means of resource awareness. This line of work is suggested by examples in this paper such as load balancers but could be enhanced by reflective resource capabilities allowing a process to influence its own deployment similar to virtualization APIs found in the context of cloud computing.

References

1. Aman, B., Ciobanu, G.: Mobile ambients with timers and types. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) ICTAC 2007. LNCS, vol. 4711, pp. 50–63. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75292-9_4
2. Aman, B., Ciobanu, G.: Timers and proximities for mobile ambients. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 33–43. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74510-5_7
3. Baeten, J.C.M., Bergstra, J.A.: Real time process algebra. Technical report CS-R 9053, Centrum voor Wiskunde en Informatica (CWI) (1990)
4. Baeten, J.C.M., Middelburg, C.A.: Process Algebra with Timing. Monographs in Computer Science. An EATSC series. Springer, Heidelberg (2002). https://doi.org/10.1007/978-3-662-04995-2
5. Ben-Yehuda, M., Day, M.D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., Yassour, B.: The turtles project: design and implementation of nested virtualization. In: Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI 2010), pp. 423–436. USENIX (2010)
6. Berger, M.: Towards abstractions for distributed systems. Ph.D. thesis, University of London, Imperial College, November 2004
7. Bračevac, O., Erdweg, S., Salvaneschi, G., Mezini, M.: CPL: a core language for cloud computing. In: Proceedings of the 15th International Conference on Modularity, pp. 94–105. ACM (2016)
8. Cardelli, L., Gordon, A.D.: Mobile ambients. Theoret. Comput. Sci. **240**(1), 177–213 (2000)
9. Cardelli, L., Gordon, A.D.: Equational properties of mobile ambients. Math. Struct. Comput. Sci. **13**(3), 371–408 (2003)
10. Ciobanu, G.: Interaction in time and space. ENTSC **203**(3), 5–18 (2008)
11. Fournet, C., Gonthier, G.: A hierarchy of equivalences for asynchronous calculi. J. Logic Algebraic Program. **63**(1), 131–173 (2005)
12. Goldberg, R.P.: Survey of virtual machine research. IEEE Comput. **7**(6), 34–45 (1974)
13. Hennessy, M., Regan, T.: A process algebra for timed systems. Inf. Comput. **117**(2), 221–239 (1995)

14. Honda, K., Yoshida, N.: On reduction-based process semantics. Theoret. Comput. Sci. **151**(2), 437–486 (1995)
15. Johnsen, E.B., Schlatte, R., Tapia Tarifa, S.L.: Integrating deployment architectures and resource consumption in timed object-oriented models. J. Log. Algebraic Methods Program. **84**(1), 67–91 (2015)
16. Lee, I., Philippou, A., Sokolsky, O.: Resources in process algebra. J. Log. Algebraic Programming **72**(1), 98–122 (2007)
17. Merro, M., Hennessy, M.: A bisimulation-based semantic theory of safe ambients. ACM Trans. Program. Lang. Syst. **28**(2), 290–330 (2006)
18. Merro, M., Zappa Nardelli, F.: Behavioral theory for mobile ambients. J. ACM **52**(6), 961–1023 (2005)
19. Moller, F., Tofts, C.: A temporal calculus of communicating systems. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 401–415. Springer, Heidelberg (1990). https://doi.org/10.1007/BFb0039073
20. Mousavi, M.R., Reniers, M.A., Basten, T., Chaudron, M.R.V.: PARS: a process algebraic approach to resources and schedulers. In: Process Algebra for Parallel and Distributed Processing. Chapman and Hall/CRC (2008)
21. Murakami, M.: Congruent bisimulation equivalence of ambient calculus based on contextual transition system. In: Proceedings of the 7th International Symposium on Theoretical Aspects of Software Engineering (TASE 2013), pp. 149–152. IEEE Computer Society (2013)
22. Nicollin, X., Sifakis, J.: The algebra of timed processes, ATP: theory and application. Inf. Comput. **114**(1), 131–178 (1994)
23. Sangiorgi, D., Walker, D.: The Pi-Calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
24. Satoh, I., Tokoro, M.: A timed calculus for distributed objects with clocks. In: Nierstrasz, O.M. (ed.) ECOOP 1993. LNCS, vol. 707, pp. 326–345. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-47910-4_17
25. Vigliotti, M., Phillips, I.: Barbs and congruences for safe mobile ambients. ENTCS **66**(3), 37–51 (2007)
26. Williams, D., Jamjoom, H., Weatherspoon, H.: The Xen-Blanket: virtualize once, run everywhere. In: Proceedings of the 7th European Conference on Computer Systems (EuroSys 2012), pp. 113–126. ACM (2012)

An Institution for Event-B

Marie Farrell[(✉)], Rosemary Monahan, and James F. Power

Department of Computer Science, Maynooth University,
Maynooth, Co. Kildare, Ireland
mfarrell@cs.nuim.ie

Abstract. This paper presents a formalisation of the Event-B formal specification language in terms of the theory of institutions. The main objective of this paper is to provide: (1) a mathematically sound semantics and (2) modularisation constructs for Event-B using the specification-building operations of the theory of institutions. Many formalisms have been improved in this way and our aim is thus to define an appropriate institution for Event-B, which we call $\mathcal{EVT}$. We provide a definition of $\mathcal{EVT}$ and the proof of its satisfaction condition. A motivating example of a traffic-light simulation is presented to illustrate our approach.

Keywords: Event-B · Institutions · Refinement · Formal methods
Modular specification · Formal specification

1 Introduction and Motivation

Event-B is an industrial-strength, state-based formalism for system-level modelling and verification, combining set theoretic notation with event-driven modelling. However, Event-B lacks well-developed modularisation constructs and it is not easy to combine specifications in Event-B with those written in other formalisms [6]. Our thesis, presented in this paper, is that the theory of institutions can provide a framework for defining a rich set of modularisation operations and promoting interoperability and heterogeneity for Event-B.

This paper is centered around an illustrative example of a specification written in Event-B, inspired by one in the *Rodin Handbook* [7], which we present in the remainder of Sect. 1. We define our institution for Event-B, called $\mathcal{EVT}$, in Sect. 2, prove that it is a valid institution, and define a comorphism between the institution for first-order predicate logic with equality and $\mathcal{EVT}$ in Sect. 3. In Section 4 we use this institution to recast our Event-B example in modular form using specification-building operators and address refinement, since this is of central importance in Event-B. We summarise our contributions and outline future directions in Sect. 5.

M. Farrell—This work is funded by Government of Ireland Postgraduate Grant from the Irish Research Council.

1.1 Formal Specification of a Traffic-Lights System in Event-B

Figure 1 presents an Event-B machine for a traffic-lights system with one light signalling cars and one signalling pedestrians [2]. The goal of the specification is to ensure that it is never the case that both cars and pedestrians receive the "go" signal at the same time (represented by boolean flags on line 2). Machine specifications typically contain variable declarations (line 2), a variant expression (none in this example), invariants (lines 3–6) and event specifications (lines 7–21). *Contexts* in Event-B can be used to model the static properties of a system (constants, axioms and carrier sets). Figure 2 provides a context giving a specification for the data-type *COLOURS*. The axiom on line 5 explicitly restricts the set to only contain the constants *red*, *green* and *orange*.

Figure 1 specifies five different events (including a starting event called `Initialisation` defined on lines 8–10). Each event has a guard, specifying when it can be activated, and an action, specifying what happens when the event is activated. For example, the `set_peds_go` event as specified on lines 11–13, has one guard expressed as a boolean expression (line 12), and one action, expressed as an assignment statement (line 13). Moreover, each event has a status, which can be either `ordinary`, `convergent`, or `anticipated`. If the status is different from `ordinary`, then the event is concerned with the variant expression, i.e. with a natural-number expression used in proving termination properties. Our example has no variant so all events have the status `ordinary`.

Figure 3 shows an Event-B machine specification for `mac2` that refines the machine `mac1` (Fig. 1). The machine `mac1` is refined by first introducing the new context on line 1 and then by replacing the truth values used in the abstract machine with new values from the carrier set *COLOURS*. This new data type is included into `mac2` using the `SEES` construct on line 1 of Fig. 3. During refinement, the user typically supplies a *gluing invariant* relating properties of the abstract machine to their counterparts in the concrete machine [2]. The gluing invariants in Fig. 3 (lines 6 and 8) define a one-to-one mapping between the concrete variables introduced in `mac2` and the abstract variables of `mac1`. The concrete variables (*peds_colour* and *cars_colour*) can be assigned either *red* or *green*, thus the gluing invariants map *true* to *green* and $false$ to *red*.

Event-B permits the addition of new variables and events: `button_pushed` (line 2) and `press_button` (lines 30–31). The existing events from `mac1` are renamed to reflect refinement; for example, the event `set_peds_green` is declared to refine `set_peds_go` (lines 14–15). This event has also been altered via the addition of a guard (line 16) and an action (line 18) that incorporate the functionality of a button-controlled pedestrian light. This example highlights features of the Event-B language, but notice how the same specification has to be provided twice in Fig. 1. The events `set_peds_go` and `set_peds_stop` are equivalent, modulo renaming of variables, to `set_cars_go` and `set_cars_stop`. Ideally, writing and proving the specification for these events should only be required once. Our approach addresses these issues as will be seen in Sect. 4.

```
MACHINE mac1
VARIABLES  cars_go,  peds_go
INVARIANTS
   inv1: cars_go  ∈  BOOL
   inv2: peds_go  ∈  BOOL
   inv3: ¬ (peds_go = true
               ∧  cars_go = true)
EVENTS
  Initialisation ordinary
    then  act1: cars_go  :=  false
          act2: peds_go  :=  false
  Event set_peds_go ≙ ordinary
    when  grd1: cars_go  =  false
    then  act1: peds_go  :=  true
  Event set_peds_stop ≙ ordinary
    then act1: peds_go  :=  false
  Event set_cars_go ≙ ordinary
    when  grd1: peds_go  =  false
    then  act1: cars_go  :=  true
  Event set_cars_stop ≙ ordinary
    then act1: cars_go  :=  false
END
```

Fig. 1. Event-B machine specification for a traffic system.

```
CONTEXT ctx1
SETS  COLOURS
CONSTANTS  red,  green,  orange
AXIOMS
   axm1: partition(COLOURS,
                 {red},  {green}, {orange})
END
```

Fig. 2. Event-B context specification for the colours of a set of traffic–lights.

```
MACHINE mac2  refines mac1   SEES ctx1
VARIABLES  cars_colour,  peds_colour,
           buttonpushed
INVARIANTS
   inv1: peds_colour  ∈  {red, green}
   inv2: (peds_go = true)
                          ⇔ (peds_colour = green)
   inv3: cars_colour  ∈  {red, green}
   inv4: (cars_go = true)
                          ⇔  (cars_colour = green)
   inv5: buttonpushed  ∈   BOOL
EVENTS
  Initialisation ordinary
    then  act1: cars_colour  :=  red
          act2: peds_colour  :=  red
  Event set_peds_green ≙ ordinary
  refines   set_peds_go
    when  grd1: cars_colour  =  red
          grd2: buttonpushed  =  true
    then  act1: peds_colour  :=  green
          act2: buttonpushed  :=  false
  Event set_peds_red ≙ ordinary
  refines   set_peds_stop
    then  act1: peds_colour  :=   red
  Event set_cars_green ≙ ordinary
  refines   set_cars_go
    when  grd1: peds_colour  =  red
    then  act1: cars_colour  :=  green
  Event set_cars_red ≙ ordinary
  refines   set_cars_stop
    then  act1: cars_colour  :=  red
  Event press_button ≙ ordinary
    then  act1: buttonpushed :=  true
END
```

Fig. 3. A refined Event-B machine specification for a traffic system.

1.2 Related Work: Institutions and Modularisation

Originally, Event-B was not equipped with any modularisation constructs. Because of this, several approaches have been suggested for modularising Event-B specifications. Abrial first proposed two styles of decomposition based on identifying shared variables and shared events [3]. Elaborating these approaches, approximately 8 modularisation plugins have been developed for various versions of Rodin, each offering a different perspective on implementing modularisation. By defining an institution for the Event-B formalism, we can modularise Event-B specifications using specification-building operators [11], and thus provide an approach to developing modular specifications that is consistent with the state of the art in formal specification.

An attempt was previously made to provide an institution and corresponding morphisms for Event-B and UML [4]. However, the definitions of Event-B sentences and models were vague, making it difficult to evaluate their semantics in a meaningful way. Also, the models described resemble the set-theoretic foundations of B specifications, whereas here we concentrate on event-based models.

Our presentation of an illustrative example in both Event-B and its modular institutional version is an important element of developing this work.

Our approach provides scope for the interoperability of Event-B and other formalisms via institution (co)morphisms. Those familiar with the institution for UML state machines, $\mathcal{UML}$, may notice that we have based the construction of our institution for Event-B, $\mathcal{EVT}$, on $\mathcal{UML}$ [8]. Both institutions describe state-based formalisms so, by keeping $\mathcal{UML}$ in mind during the development of $\mathcal{EVT}$, it will be possible to design meaningful translations between them in the future.

2 An Institution for Event-B

The theory of institutions, originally developed by Goguen and Burstall in a series of papers originating from their work on algebraic specification, provides a general framework for defining a logical system [5].

Definition 1 (Institution). An **institution** $\mathcal{INS}$ for some given formalism will consist of definitions for:

Vocabulary: a category **Sign** whose objects are called signatures and whose arrows are called signature morphisms.

Syntax: a functor $\mathbf{Sen} : \mathbf{Sign} \to \mathbf{Set}$ giving a set $\mathbf{Sen}(\Sigma)$ of Σ-*sentences* for each signature Σ and a function $\mathbf{Sen}(\sigma) : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$ for each signature morphism $\sigma : \Sigma \to \Sigma'$.

Semantics: a functor $\mathbf{Mod} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ giving a category $\mathbf{Mod}(\Sigma)$ of Σ-*models* for each signature Σ and a functor $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ for each signature morphism $\sigma : \Sigma \to \Sigma'$.

Satisfaction: for every signature Σ, a satisfaction relation $\models_{\mathcal{INS},\Sigma}$ between Σ-models and Σ-sentences.

An institution must uphold the **satisfaction condition:** for any signature morphism $\sigma : \Sigma \to \Sigma'$ and translations $\mathbf{Mod}(\sigma)$ of models and $\mathbf{Sen}(\sigma)$ of sentences we have for any $\phi \in \mathbf{Sen}(\Sigma)$ and $M' \in |\mathbf{Mod}(\Sigma')|$.

$$M' \models_{\mathcal{INS},\Sigma'} \mathbf{Sen}(\sigma)(\phi) \quad \iff \quad \mathbf{Mod}(\sigma)(M') \models_{\mathcal{INS},\Sigma} \phi$$

There are two basic languages within the Event-B language. The first one is the Event-B mathematical language (propositional/predicate logic, set-theory and arithmetic) and the second is the Event-B modelling language [1]. To represent the latter, we propose a new custom solution; for the former, however, we can use $\mathcal{FOPEQ}$, the institution of first-order logic with equality. Thus, our institution for Event-B is built on $\mathcal{FOPEQ}$.

Definition 2 ($\mathcal{FOPEQ}$-Signature). A **signature** in $\mathcal{FOPEQ}$, $\Sigma_{\mathcal{FOPEQ}} = \langle S, \Omega, \Pi \rangle$, is a tuple where S is a set of sort names, Ω is a set of operation names indexed by arity and sort, and Π is a set of predicate names indexed by arity.

Definition 3 ($\Sigma_{\mathcal{FOPEQ}}$-Sentence). For any $\Sigma_{\mathcal{FOPEQ}} = \langle S, \Omega, \Pi \rangle$, $\Sigma_{\mathcal{FOPEQ}}$-**sentences** are closed first-order formulae built out of atomic formulae using $\wedge, \vee, \neg, \Rightarrow, \iff, \exists, \forall$. Atomic formulae are equalities between $\langle S, \Omega \rangle$-terms, predicate formulae of the form $p(t_1, \ldots, t_n)$ where $p \in \Pi$ and $t_1, \ldots, t_n$ are terms (with variables), and the logical constants `true` and `false`.

Definition 4 ($\Sigma_{\mathcal{FOPEQ}}$-Model). Given a signature $\Sigma_{\mathcal{FOPEQ}} = \langle S, \Omega, \Pi \rangle$, a model over $\mathcal{FOPEQ}$ consists of a carrier set $|A|_s$ for each sort name $s \in S$, a function $f_A : |A|_{s_1} \times \cdots \times |A|_{s_n} \rightarrow |A|_s$ for each operation name $f \in \Omega_{s_1 \ldots s_n, s}$ and a relation $p_A \subseteq |A|_{s_1} \times \cdots \times |A|_{s_n}$ for each predicate name $p \in \Pi_{s_1 \cdots s_n}$, where $s_1, \ldots, s_n$, and s are sort names.

The satisfaction relation in $\mathcal{FOPEQ}$ is the usual satisfaction of first-order sentences by first-order structures.

2.1 Defining $\mathcal{EVT}$

Definition 5 ($\mathcal{EVT}$-Signature). A **signature** in $\mathcal{EVT}$ is a five-tuple $\Sigma_{\mathcal{EVT}} = \langle S, \Omega, \Pi, E, V \rangle$ where $\langle S, \Omega, \Pi \rangle$ is a standard $\mathcal{FOPEQ}$-signature as described above, E is a set of events, i.e. of pairs $\langle$*event name*, *status*$\rangle$ where status belongs to the poset $\{$`ordinary` $<$ `anticipated` $<$ `convergent`$\}$, and V is a set of sorted variables. We assume that every signature has an initial event, called `Init`, whose status is always `ordinary`.

Notation: We write Σ in place of $\Sigma_{\mathcal{EVT}}$ when describing a signature over our institution for Event-B. For signatures over other institutions than $\mathcal{EVT}$ we will use the subscript notation; e.g. a signature over $\mathcal{FOPEQ}$ is denoted by $\Sigma_{\mathcal{FOPEQ}}$. For a given signature Σ, we access its individual components using a dot-notation, e.g. $\Sigma.V$ for the set V in the tuple Σ.

Definition 6 ($\mathcal{EVT}$-Signature Morphism). A **signature morphism** $\sigma : \Sigma \rightarrow \Sigma'$ is a five-tuple containing σ_S, σ_Ω, σ_Π, σ_E and σ_V. Here σ_S, σ_Ω, σ_Π are the mappings taken from the corresponding signature morphism in $\mathcal{FOPEQ}$.

- $\sigma_E : \Sigma.E \rightarrow \Sigma'.E$ is a function such that for any mapping $\sigma_E\langle e, st\rangle = \langle e', st'\rangle$ we have $st \leq st'$; in addition, σ_E preserves the initial event: in symbols, we have that $\sigma_E\langle \texttt{Init}, \texttt{ordinary}\rangle = \langle \texttt{Init}, \texttt{ordinary}\rangle$.
- $\sigma_V : \Sigma.V \rightarrow \Sigma'.V$ is a sort-preserving function on sets of variable names, working similarly to the sort-preserving mapping for constant symbols, σ_Ω.

Definition 7 ($\Sigma_{\mathcal{EVT}}$-Sentence). A sentence over $\mathcal{EVT}$ is a pair $\langle e, \phi(\overline{x}, \overline{x}')\rangle$ where e is an event name in the domain of $\Sigma.E$ and $\phi(\overline{x}, \overline{x}')$ is an open $\mathcal{FOPEQ}$-formula over the variables $\overline{x}$ from $\Sigma.V$ and their primed versions $\overline{x}'$.

In the *Rodin Platform*, Event-B machines are presented (syntactically sugared) as can be seen below, where $I(\overline{x})$ represents the invariant over $\overline{x}$.

```
MACHINE m SEES ctx  refines a
  VARIABLES x̄
  INVARIANTS I(x̄)
  VARIANT n(x̄)
  EVENTS
  Initialisation ordinary
    then  act-name: BA(x̄, x̄′)
  Event e ≙ status
    any p̄
    when  guard-name: G(x̄, p̄)
    with  witness-name: W(x̄, p̄)
    then  act-name: BA(x̄, p̄, x̄′)
        ⋮
END
```

The variant expression, denoted by $n(\overline{x})$, is used for proving termination properties. Events that have a status of `anticipated` or `convergent` must not increase and strictly decrease the variant expression respectively. Events can have parameter(s) as given by $\overline{p}$. $G(\overline{x}, \overline{p})$ and $W(\overline{x}, \overline{p})$ represent the guard(s) and witness(es) respectively over the variables and parameter(s). Actions are interpreted as before-after predicates i.e. $x := x + 1$ is interpreted as $x' = x + 1$. Thus, $BA(\overline{x}, \overline{p}, \overline{x}')$ represents the action(s) over the parameter(s) $\overline{p}$ and the sets of variables $\overline{x}$ and $\overline{x}'$.

Sentences written in the mathematical language (such as axioms) are interpreted as sentences over $\mathcal{FOPEQ}$. We can include these in specifications over $\mathcal{EVT}$ using the comorphism which will be defined in Sect. 3. We represent the Event-B event, variant and invariant sentences as sentences over $\mathcal{EVT}$.

For each Event-B invariant sentence $I(\overline{x})$ we form the open $\mathcal{FOPEQ}$-sentence $I(\overline{x}) \wedge I(\overline{x}')$. Since invariants must hold for all events in a machine, each invariant sentence is paired with each event name e for all $\langle e, s\rangle \in \Sigma.E$, where s is an event status. Thus, we form the $\mathcal{EVT}$ sentence $\langle e, I(\overline{x}) \wedge I(\overline{x}')\rangle$.

The variant expression applies to specific events, so we pair it with an event name in order to meaningfully evaluate it. This expression can be translated into an open $\mathcal{FOPEQ}$-term, which we denote by $n(\overline{x})$, and we use this to construct a formula based on the status of the event(s) in the signature Σ.

- For each $\langle e, \texttt{anticipated}\rangle \in \Sigma.E$ we form the sentence $\langle e, n(\overline{x}') \leq n(\overline{x})\rangle$.
- For each $\langle e, \texttt{convergent}\rangle \in \Sigma.E$ we form the sentence $\langle e, n(\overline{x}') < n(\overline{x})\rangle$.

Note that we are assuming the existence of a suitable type for variant expressions and the usual arithmetic interpretation of the predicates $<$ and $\leq$.

Event guard(s) and witnesses are also labelled predicates that can be translated into open $\mathcal{FOPEQ}$-formulae over the variables $\overline{x}$ in V and parameters $\overline{p}$. These are denoted by $G(\overline{x}, \overline{p})$ and $W(\overline{x}, \overline{p})$ respectively. In Event-B, actions are interpreted as before-after predicates, and so they can be translated into open $\mathcal{FOPEQ}$-formulae denoted by $BA(\overline{x}, \overline{p}, \overline{x}')$. Thus for each event we form the formula $\phi(\overline{x}, \overline{x}') = \exists \overline{p} \cdot G(\overline{x}, \overline{p}) \wedge W(\overline{x}, \overline{p}) \wedge BA(\overline{x}, \overline{p}, \overline{x}')$ where $\overline{p}$ are the event parameters. This generates an $\mathcal{EVT}$-sentence of the form $\langle e, \phi(\overline{x}, \overline{x}')\rangle$. The `Init` event, which is an Event-B sentence over only the after variables denoted by $\overline{x}'$, is a special case. In this case, we form the $\mathcal{EVT}$-sentence $\langle \texttt{Init}, \phi(\overline{x}')\rangle$.

There is no formal semantics for Event-B defined in the literature as such. Therefore, we have based our construction of $\mathcal{EVT}$-models on the notion of a mathematical model as described by Abrial [1, Ch. 14]. In these models the state is represented as a sequence of variable-values and models are defined over before and after states. We interpret these states as sets of variable-to-value mappings in our definition of $\mathcal{EVT}$-models.

Definition 8 (Σ-***State***$_A$)**.** For any given $\mathcal{EVT}$-signature Σ we define a **Σ-state** of an algebra A as a set of (sort appropriate) variable-to-value mappings whose domain is the set of sort-indexed variable names $\Sigma.V$. We define the set $State_A$ as the set of all such Σ-states. By "sort appropriate" we mean that for any variable x of sort s in V, the corresponding value for x should be drawn from $|A|_s$, the carrier set of s given by the $\mathcal{FOPEQ}$-model A.

Definition 9 ($\Sigma_{\mathcal{EVT}}$**-Model).** Given $\Sigma = \langle S, \Omega, \Pi, E, V\rangle$, **Mod**$(\Sigma)$ provides a category of models, where a **model** over Σ is a tuple $\langle A, L, R\rangle$. A is a $\Sigma_{\mathcal{FOPEQ}}$-model, and the non-empty initialising set $L \subseteq State_A$ provides the states after the `Init` event. Then for every event name $e \in dom(E)$, other than `Init`, we define $R.e \subseteq State_A \times State_A$ where for each pair of states $\langle s, s'\rangle$ in $R.e$, s provides values for the variables x in V, and s' provides values for their primed versions x'. Then $R = \{R.e \mid e \in dom(E) \ and \ e \neq \texttt{Init}\}$.

Intuitively, a model over Σ maps every event name $e \in dom(\Sigma.E)$ to a set of variable-to-value mappings over the carriers corresponding to the sorts of each of the variables $x \in \Sigma.V$ and their primed versions x'. In cases where there are no variables in $\Sigma.V$, L is the singleton $\{\{\}\}$.

For example, given the event e on the right, with natural number variable x and boolean variable y we construct the variable to value mappings:

```
Event e ≙
  when  grd1:    x<2
  then  act1:    x := x + 1
        act2:    y := false
```

$$R_e = \left\{ \begin{array}{ll} \{x \mapsto 0, y \mapsto false, x' \mapsto 1, y' \mapsto false\}, & \{x \mapsto 0, y \mapsto true, x' \mapsto 1, y' \mapsto false\}, \\ \{x \mapsto 1, y \mapsto false, x' \mapsto 2, y' \mapsto false\}, & \{x \mapsto 1, y \mapsto true, x' \mapsto 2, y' \mapsto false\} \end{array} \right\}$$

The notation used above is interpreted as *variable name* $\mapsto$ *value* where the value is drawn from the carrier set corresponding to the sort of the variable name given in $\Sigma.V$. We note that trivial models be excluded as the initialising set L is never empty. In cases where there are no variables in $\Sigma.V$, L is the singleton $L = \{\{\}\}$.

The reduct of an $\mathcal{EVT}$-model $M = \langle A, L, R\rangle$ along an $\mathcal{EVT}$-signature morphism $\sigma : \Sigma \to \Sigma'$ is given by $M|_\sigma = \langle A|_\sigma, L|_\sigma, R|_\sigma\rangle$. Here $A|_\sigma$ is the reduct of the $\mathcal{FOPEQ}$-component of the $\mathcal{EVT}$-model along the $\mathcal{FOPEQ}$-components of σ. $L|_\sigma$ and $R|_\sigma$ are based on the reduction of the states of A along σ, i.e. for every Σ'-state s of A, that is for every sorted map $s : \Sigma'.V \to |A|$, $s|_\sigma$ is the map $\Sigma'.V \to |A|$ given by the composition $\sigma_V; s$. This extends in the usual manner from states to sets of states and to relations on states.

Satisfaction: In order to define the satisfaction relation for $\mathcal{EVT}$, we describe an embedding from $\mathcal{EVT}$-signatures and models to $\mathcal{FOPEQ}$-signatures and models.

Given an $\mathcal{EVT}$-signature $\Sigma = \langle S, \Omega, \Pi, E, V\rangle$ we form the following two $\mathcal{FOPEQ}$-signatures:

– $\Sigma^{(V,V')}_{\mathcal{FOPEQ}} = \langle S, \Omega \cup V \cup V', \Pi\rangle$ where V and V' are the variables and their primed versions, respectively, that are drawn from the $\mathcal{EVT}$-signature, and represented as 0-ary operators with unchanged sort. The intuition here is

that the set of variable-to-value mappings for the free variables in an $\mathcal{EVT}$-signature Σ are represented by adding a distinguished 0-ary operation symbol to the corresponding $\mathcal{FOPEQ}$-signature for each of the variables $x \in V$ and their primed versions.
- Similarly, for the initial state and its variables, we construct the signature $\Sigma^{(V')}_{\mathcal{FOPEQ}} = \langle S, \Omega \cup V', \Pi \rangle$.

Given the $\mathcal{EVT}$ Σ-model $\langle A, L, R \rangle$, we construct the $\mathcal{FOPEQ}$-models:

- For every pair of states $\langle s, s' \rangle$, we form the $\Sigma^{(V,V')}_{\mathcal{FOPEQ}}$-model expansion $A^{(s,s')}$, which is the $\mathcal{FOPEQ}$-component A of the $\mathcal{EVT}$-model, with s and s' added as interpretations for the new operators that correspond to the variables from V and V' respectively.
- For each initial state $s' \in L$ we construct the $\Sigma^{(V')}_{\mathcal{FOPEQ}}$-model expansion $A^{(s')}$ analogously.

For any $\mathcal{EVT}$-sentence over Σ of the form $\langle e, \phi(\overline{x}, \overline{x}') \rangle$ we create a corresponding $\mathcal{FOPEQ}$-formula by replacing the free variables with their corresponding operator symbols. We write this (closed) formula as $\phi(\overline{x}, \overline{x}')$.

Definition 10 (Satisfaction Relation). For any $\mathcal{EVT}$-model $\langle A, L, R \rangle$ and $\mathcal{EVT}$-sentence $\langle e, \phi(\overline{x}, \overline{x}') \rangle$, where e is an event name other than `Init`, we define:

$$\langle A, L, R \rangle \models_{\Sigma} \langle e, \phi(\overline{x}, \overline{x}') \rangle \iff \forall \langle s, s' \rangle \in R.e \cdot A^{(s,s')} \models_{\Sigma^{(V,V')}_{\mathcal{FOPEQ}}} \phi(\overline{x}, \overline{x}')$$

Similarly, we evaluate the satisfaction condition of $\mathcal{EVT}$-sentences of the form $\langle \texttt{Init}, \phi(\overline{x}') \rangle$ as follows:

$$\langle A, L, R \rangle \models_{\Sigma} \langle \texttt{Init}, \phi(\overline{x}') \rangle \iff \forall s' \in L \cdot A^{(s')} \models_{\Sigma^{(V')}_{\mathcal{FOPEQ}}} \phi(\overline{x}')$$

Theorem 1 (Satisfaction Condition). *Given $\mathcal{EVT}$ signatures Σ_1 and Σ_2, a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$, a Σ_2-model M_2 and a Σ_1-sentence ψ_1, the following satisfaction condition holds:*

$$\mathbf{Mod}(\sigma)(M_2) \models_{\mathcal{EVT}_{\Sigma_1}} \psi_1 \quad \iff \quad M_2 \models_{\mathcal{EVT}_{\Sigma_2}} \mathbf{Sen}(\sigma)(\psi_1)$$

Proof. Let M_2 be the model $\langle A_2, L_2, R_2 \rangle$, and ψ_1 the sentence $\langle e, \phi(\overline{x}, \overline{x}') \rangle$. Then the satisfaction condition is equivalent to

$$\forall \langle s, s' \rangle \in R_2|_{\sigma}.e \cdot (A_2|_{\sigma})^{(s,s')}|_{\sigma} \models_{\mathcal{FOPEQ}_{\Sigma^{(V_1,V_1')}_{\mathcal{FOPEQ}}}} \phi(\overline{x}, \overline{x}')$$
$$\iff \forall \langle s, s' \rangle \in R_2.\sigma_E(e) \cdot A_2^{(s,s')} \models_{\mathcal{FOPEQ}_{\Sigma^{(V_2,V_2')}_{\mathcal{FOPEQ}}}} Sen(\sigma)(\phi(\overline{x}, \overline{x}'))$$

Here, validity follows from the validity of satisfaction in $\mathcal{FOPEQ}$. We prove a similar result for initial events in the same way. □

Pragmatics of Specification Building in $\mathcal{EVT}$: We represent an Event-B specification, such as that for `mac1` in Fig. 1, as a presentation over $\mathcal{EVT}$. For any signature Σ, a Σ-presentation is a set of Σ-sentences. A model of a Σ-presentation is a Σ-model that satisfies all of the sentences in the presentation [5]. Thus, for a presentation in $\mathcal{EVT}$, model components corresponding to an event must satisfy all of the sentences specifying that event. This incorporates the standard semantics of the *extends* operator for events in Event-B where the extending event implicitly has all the parameters, guards and actions of the extended event but can have additional parameters, guards and actions [3].

An interesting aspect is that if a variable is not assigned to within an action, then a model for the event may associate a new value with this variable. Some languages deal with this using a *frame condition*, asserting implicitly that values for unmodified variables do not change. In Event-B such a condition would cause complications when combining presentations, since variables unreferenced in one event will be constrained not to change, and this may contradict an action for them in the other event. As far as we can tell, the informal semantics for the Event-B language do not require a frame condition, and we have not included one in our definition.

3 Relating $\mathcal{FOPEQ}$ and $\mathcal{EVT}$

Initially, we defined the relationship between $\mathcal{FOPEQ}$ and $\mathcal{EVT}$ to be a duplex institution formed from a restricted version of $\mathcal{EVT}$ ($\mathcal{EVT}_{res}$) and $\mathcal{FOPEQ}$ where $\mathcal{EVT}_{res}$ is the institution $\mathcal{EVT}$ but does not contain any $\mathcal{FOPEQ}$ components. Duplex institutions are constructed by enriching one institution, in this case $\mathcal{EVT}_{res}$, by the sentences of another, in this case $\mathcal{FOPEQ}$, using an institution semi-morphism [5,11]. This approach would allow us to constrain $\mathcal{EVT}_{res}$ by $\mathcal{FOPEQ}$ and thus facilitate the use of $\mathcal{FOPEQ}$-sentences in an elegant way. However, duplex institutions are not supported in HETS [9], and therefore we opt for a comorphism which embeds the simpler institution $\mathcal{FOPEQ}$ into the more complex institution $\mathcal{EVT}$ [11].

Definition 11. (The institution comorphism ρ). We define $\rho : \mathcal{FOPEQ} \to \mathcal{EVT}$ to be an institution comorphism composed of:

- The functor $\rho^{Sign} : \mathbf{Sign}_{\mathcal{FOPEQ}} \to \mathbf{Sign}_{\mathcal{EVT}}$ which takes as input a $\mathcal{FOPEQ}$-signature of the form $\langle S, \Omega, \Pi \rangle$ and extends it with the set $E = \{\langle \texttt{Init ordinary} \rangle\}$ and an empty set of variable names V. $\rho^{Sign}(\sigma)$ works as σ on S, Ω and Π, it is the identity on the `Init` event and the empty function on the empty set of variable names.
- The natural transformation $\rho^{Sen} : \mathbf{Sen}_{\mathcal{FOPEQ}} \to \rho^{Sign}; \mathbf{Sen}_{\mathcal{EVT}}$ which pairs any closed $\mathcal{FOPEQ}$-sentence (given by ϕ) with the `Init` event name to form the $\mathcal{EVT}$-sentence $\langle \texttt{Init}, \phi \rangle$. As there are no variables in the signature, we do not require ϕ to be over the variables $\overline{x}$ and $\overline{x}'$.
- The natural transformation $\rho^{Mod} : (\rho^{Sign})^{op}; \mathbf{Mod}_{\mathcal{EVT}} \to \mathbf{Mod}_{\mathcal{FOPEQ}}$ is such that for any $\mathcal{FOPEQ}$-signature Σ,
$$\rho^{Mod}_{\Sigma}(Mod(\rho^{Sign}(\Sigma))) = \rho^{Mod}_{\Sigma}(\langle A, L, \emptyset \rangle) = A$$

Theorem 2. *The institution comorphism ρ is defined such that for any $\Sigma \in |\mathbf{Sign}_{\mathcal{FOPEQ}}|$, the translations $\rho^{Sen}_{\Sigma} : \mathbf{Sen}_{\mathcal{FOPEQ}}(\Sigma) \rightarrow \mathbf{Sen}_{\mathcal{EVT}}(\rho^{Sign}(\Sigma))$ and $\rho^{Mod}_{\Sigma} : \mathbf{Mod}_{\mathcal{EVT}}(\rho^{Sign}(\Sigma)) \rightarrow \mathbf{Mod}_{\mathcal{FOPEQ}}(\Sigma)$ preserve the satisfaction relation. That is, for any $\psi \in \mathbf{Sen}_{\mathcal{FOPEQ}}(\Sigma)$ and $M' \in |\mathbf{Mod}_{\mathcal{EVT}}(\rho^{Sign}(\Sigma))|$*

$$\rho^{Mod}_{\Sigma}(M') \models_{\mathcal{FOPEQ}_{\Sigma}} \psi \iff M' \models_{\mathcal{EVT}_{\rho^{Sign}(\Sigma)}} \rho^{Sen}_{\Sigma}(\psi) \quad (*)$$

Proof. By Definition 11, $M' = \langle A, L, \emptyset \rangle$, $\rho^{Mod}_{\Sigma}(M') = A$ and $\rho^{Sen}_{\Sigma}(\psi) = \langle \mathtt{Init}, \psi \rangle$. Therefore, we transform (*) into

$$A \models_{\mathcal{FOPEQ}_{\Sigma}} \psi \iff M' \models_{\mathcal{EVT}_{\rho^{Sign}(\Sigma)}} \langle \mathtt{Init}, \psi \rangle$$

Then, by the definition of satisfaction in $\mathcal{EVT}$ (Definition 10)

$$A \models_{\mathcal{FOPEQ}_{\Sigma}} \psi \iff A^{(s')} \models_{\mathcal{FOPEQ}_{(\rho^{Sign}(\Sigma))^{(V')}_{\mathcal{FOPEQ}}}} \psi$$

We deduce that $\Sigma = (\rho^{Sign}(\Sigma))^{V'}_{\mathcal{FOPEQ}}$, since there are no variable names in V' and thus no new operator symbols are added to the signature. As there are no variable names in V', $L = \{\{\}\}$, so we can conclude that $A^{(s')} = A$. Thus the satisfaction condition holds. □

For a Σ-specification written over $\mathcal{FOPEQ}$ we can use the specification building operator $\_$ `with` $\rho : Spec_{\mathcal{FOPEQ}}(\Sigma) \rightarrow Spec_{\mathcal{EVT}}(\rho^{Sign}(\Sigma))$ to interpret this as a specification over $\mathcal{EVT}$ [11]. This results in a specification with just the `Init` event and no variables, containing $\mathcal{FOPEQ}$-sentences that hold in the initial state. This process is used to represent contexts, specifically their axioms, which are written over $\mathcal{FOPEQ}$ as sentences over $\mathcal{EVT}$.

In cases where a specification is enriched with new events, then the axioms and invariants should also apply to these new events. One approach to this would require a new kind of $\mathcal{EVT}$-sentence for invariants, which we denote by $\langle inv, \phi(\overline{x}, \overline{x}') \rangle$, these are applied to all events in the specification when evaluating the satisfaction condition. We do not present these details fully here due to space concerns.

3.1 Pushouts and Amalgamation

We ensure that the institution $\mathcal{EVT}$ has good modularity properties by proving that $\mathcal{EVT}$ admits the amalgamation property: all pushouts in $\mathbf{Sign}_{\mathcal{EVT}}$ exist and every pushout diagram in $\mathbf{Sign}_{\mathcal{EVT}}$ admits *weak* model amalgamation [11].

Proposition 1. *Pushouts exist in $\mathbf{Sign}_{\mathcal{EVT}}$.*

Proof. Given two signature morphisms $\sigma_1 : \Sigma \rightarrow \Sigma_1$ and $\sigma_2 : \Sigma \rightarrow \Sigma_2$ a pushout is a triple $(\Sigma', \sigma'_1, \sigma'_2)$ that satisfies the universal property: for all triples $(\Sigma'', \sigma''_1, \sigma''_2)$ there exists a unique morphism $u : \Sigma' \rightarrow \Sigma''$ such that the diagram on the left below commutes. Our pushout construction follows $\mathcal{FOPEQ}$ for the elements that $\mathcal{FOPEQ}$ has in common with $\mathcal{EVT}$. In $\mathbf{Sign}_{\mathcal{EVT}}$ the additional elements are E and V.

$$\begin{tikzcd}
 & \Sigma \arrow[dl, "\sigma_1"'] \arrow[dr, "\sigma_2"] & \\
\Sigma_1 \arrow[dr, "\sigma_1'"] \arrow[ddr, "\sigma_1''"'] & & \Sigma_2 \arrow[dl, "\sigma_2'"'] \arrow[ddl, "\sigma_2''"] \\
 & \Sigma' \arrow[d, dashed, "u"] & \\
 & \Sigma'' &
\end{tikzcd}$$

– *Set of* ⟨event name, status⟩ *pairs* E: The set of all event names in the pushout is the pushout in **Set** on event names only. Then, the status of an event in the pushout is the supremum of all statuses of all events that are mapped to it. Since signature morphisms map $\langle$`Init,ordinary`$\rangle$ to $\langle$`Init,ordinary`$\rangle$ the pushout does likewise. The universality property for E follows from that of **Set**.

– *Set of sort-indexed variable names* V: The set of sort-indexed variable names in the pushout is the pushout in $\mathcal{FOPEQ}$ for the sort components and the pushout in **Set** for the variable names. This is a similar construction to the pushout for operation names in $\mathcal{FOPEQ}$ as these also have to follow the sort pushout. Thus, the universality property for V follows from that of **Set** and the $\mathcal{FOPEQ}$ pushout for sorts. □

Proposition 2. *Every pushout diagram in* $\boldsymbol{Sign}_{\mathcal{EVT}}$ *admits weak model amalgamation.*

We decompose this proposition into two further subpropositions:

Proposition 2(a). *For* $M_1 \in |\boldsymbol{Mod}(\Sigma_1)|$ *and* $M_2 \in |\boldsymbol{Mod}(\Sigma_2)|$ *such that* $M_1|_{\sigma_1} = M_2|_{\sigma_2}$*, there exists a model (the amalgamation of* M_1 *and* M_2*)* $M' \in |\boldsymbol{Mod}(\Sigma')|$ *such that* $M'|_{\sigma_1'} = M_1$ *and* $M'|_{\sigma_2'} = M_2$*.*

Proof. Consider the commutative diagram with signature morphisms $\sigma_1, \sigma_2, \sigma_1'$ and σ_2' below:

$$\begin{tikzcd}
 & M' = \langle A', L', R' \rangle \arrow[dl, "Mod(\sigma_1')"'] \arrow[dr, "Mod(\sigma_2')"] & \\
M_1 = \langle A_1, L_1, R_1 \rangle \arrow[dr, "Mod(\sigma_1)"'] & & M_2 = \langle A_2, L_2, R_2 \rangle \arrow[dl, "Mod(\sigma_2)"] \\
 & M = \langle A, L, R \rangle &
\end{tikzcd}$$

We construct $M' = \langle A', L', R' \rangle$ as follows. A' is the $\mathcal{FOPEQ}$-model (amalgamation of A_1 and A_2) over $\mathcal{FOPEQ}$. We construct the initialising set L' by amalgamating L_1 and L_2 to get the set of all possible combinations of variable mappings, while respecting the amalgamations induced on variable names via the pushout V'. We construct the relation R', which is the amalgamation of R_1 and R_2, in a similar manner. □

Proposition 2(b). *For any two model morphisms* $f_1 : M_{11} \rightarrow M_{12}$ *in* $\boldsymbol{Mod}(\Sigma_1)$ *and* $f_2 : M_{21} \rightarrow M_{22}$ *in* $\boldsymbol{Mod}(\Sigma_2)$ *such that* $f_1|_{\sigma_1} = f_2|_{\sigma_2}$*, there exists a model morphism (the amalgamation of* f_1 *and* f_2*) called* $f' : M_1' \rightarrow M_2'$ *in* $\boldsymbol{Mod}(\Sigma')$*, such that* $f'|_{\sigma_1'} = f_1$ *and* $f'|_{\sigma_2'} = f_2$*.*

We have omitted this proof but it can be found on our webpage[1].

[1] http://www.cs.nuim.ie/~mfarrell/extended.pdf.

4 Modularising Event-B Specifications

Our definition of $\mathcal{EVT}$ allows the restructuring of Event-B specifications using the standard specification-building operators for institutions [11]. Thus $\mathcal{EVT}$ provides a means for writing down and splitting up the components of an Event-B system, facilitating increased modularity for Event-B specifications. Figure 4 contains heterogeneous structured specifications corresponding to the Event-B machine `mac1` defined in Fig. 1. Since Hets is our target platform, where each institution is represented as a "logic", we use its notation and implementation of the logic for $\mathcal{CASL}$ to represent the $\mathcal{FOPEQ}$ components of our specifications.

Lines 1–6: TwoBools can be presented as a pure $\mathcal{CASL}$ specification, declaring two boolean variables constrained to have different values.

Lines 7–17: LightAbstract is a specification in the $\mathcal{EVT}$ logic for a single traffic light that extends (using keyword then) TwoBools which is first translated via the comorphism ρ into a specification over $\mathcal{EVT}$. It contains the events `set_go` and `set_stop`, with the constraint that a light can only be set to "go" if its opposite light is not set to "go". We use "thenAct" in place of the "then" Event-B keyword to distinguish from the "then" specification-building operator.

Lines 18–32: The specification mac1 combines (using keyword and) two versions of LightAbstract, each with a different signature morphism (σ_1 and σ_2) mapping the specification variables and event names to those in the Event-B machine. The where notation used on lines 22–32 is just a convenient presentation of the signature morphisms, it is not part of the syntax of the specification language that we use in Hets.

We get a presentation over the institution $\mathcal{EVT}$ for `mac1` by flattening out the structuring. Notice that the specification for each individual light had to be explicitly written down twice in the Event-B machine in Fig. 1 (lines 11–15 and lines 16–20). In our modular institution-based presentation we only need one light specification and simply supply the required variable and event mappings. In this way, $\mathcal{EVT}$ provides a more flexible degree of modularity than is currently present in Event-B.

4.1 Refinement in the $\mathcal{EVT}$ Institution

Event-B supports three forms of machine refinement: the refinement of event internals (guards and actions) and invariants; the addition of new events; and the decomposition of an event into several events [2]. It is therefore essential for any formalisation of Event-B to be capable of capturing refinement.

In general for institutions, a refinement from an abstract specification A to some concrete specification C is defined using model-class inclusion as $|Mod(C)| \subseteq |Mod(A)|$ when $Sig[A] = Sig[C]$. In Event-B, new variable or event names cannot be added if the signatures stay the same. This provides only one option: strengthen the formulae in event definitions, which will result in at most the same number of models. This accounts for the first form of refinement in Event-B. Both of the other forms of refinement in Event-B cause the signatures to change i.e. the set of events will get larger when adding or decomposing events. In the case when the signatures are different, we can define a signature morphism $\sigma : Sig[A] \rightarrow Sig[C]$ from which we can construct the model reduct $Mod(\sigma) : Mod(C) \rightarrow Mod(A)$. We can thus restrict the concrete model to only contain elements of the abstract signatures by applying the model reduct before evaluating the subset relation defined above.

4.2 A Modular, Refined Specification

Figure 5 contains a presentation over $\mathcal{EVT}$ corresponding to the main elements of the Event-B specification MAC2 presented in Figs. 2 and 3. Here, we present three $\mathcal{CASL}$ specifications and three $\mathcal{EVT}$ specifications.

Lines 1–10: We specify the *Colours* data type with a standard $\mathcal{CASL}$ specification, as can be seen in Fig. 2. The specification TwoColours describes two variables of type *Colours* constrained to be not both green at the same time. This corresponds to the gluing invariants on lines 5 and 7 of Fig. 3. The specification modularisation constructs used in Fig. 5, allow these properties to be handled distinctly and in a manner that facilitates comparison with the TwoBools specification on lines 1–6 of Fig. 4.

Lines 15–25: A specification for a single light is provided in LightRefined which uses TwoColours to describe the colour of the lights. As was the case with LightAbstract in Fig. 4, the specification makes clear how a single light operates. An added benefit here is that a direct comparison with the abstract specification can be done on a per-light basis.

Lines 11–14, 26–34: The specifications BoolButton and ButtonSpec account for the part of the MAC2 specification that requires a button. These details were woven through the code in Fig. 3 (lines 2, 8, 16, 18, 29, 30) but the specification-building operators allow us to modularise the specification and group these related definitions together, clarifying how the button actually operates.

Lines 35–51: Finally, to bring this all together we combine a copy of LightRefined with a specification corresponding to the sum (and) of LightRefined and ButtonSpec with appropriate signature morphisms. This second specification combines the event `gobutton` in ButtonSpec with the event `set_green` in LightRefined thus accounting for `set_peds_green` in Fig. 3. One small issue involves making sure that the name replacements are done correctly, and in the correct order, hence the bracketing on lines 37–38 is important.

The combination of these specifications involves merging two events with different names: `gobutton` from ButtonSpec with the event `set_green` from LightRefined. To ensure that these differently-named events are combined into an event of the same name we use the signature morphism σ_5 to give `gobutton` the same name as `set_green` before combining them. Ensuring that the events

```
logic CASL
spec  TwoBools =
  Bool
  then
    ops i_go, u_go : Bool
    . ¬ (i_go = true ∧ u_go = true)

logic EVT
spec  LightAbstract =
  TwoBools with ρ
  then
    Initialisation ordinary
      thenAct  act1 : i_go := false
    Event set_go ≙ ordinary
      when  grd1: u_go = false
      thenAct  act1: i_go := true
    Event set_stop ≙ ordinary
      thenAct  act1: i_go := false
```

```
logic EVT
spec  mac1 =
  (LightAbstract with σ1)
   and (LightAbstract with σ2)
  where
   σ1 = {i_go ↦ cars_go, u_go ↦ peds_go,
         ⟨set_go, ordinary⟩
           ↦ ⟨set_cars_go, ordinary⟩,
         ⟨set_stop, ordinary⟩
           ↦ ⟨set_cars_stop, ordinary⟩}
   σ2 = {i_go ↦ peds_go, u_go ↦ cars_go,
         ⟨set_go, ordinary⟩
           ↦ ⟨set_peds_go, ordinary⟩,
         ⟨set_stop, ordinary⟩
           ↦ ⟨set_peds_stop, ordinary⟩}
```

Fig. 4. A modular institution-based presentation corresponding to the abstract machine `mac1` in Fig. 1.

```
logic CASL
spec  Colours =
  then
    sorts
    free type Colours ::= red|green|
                          orange

spec  TwoColours =
    Colours
    then
      ops icol, ucol : Colours
      . ¬(icol = green ∧ ucol = green)

spec  BoolButton =
  Bool
  then
    ops button : Bool

logic EVT
spec  LightRefined =
  TwoColours with ρ
  then
    Initialisation ordinary
      thenAct  act1: icol := red
    Event set_green ≙ ordinary
      when  grd1: ucol = red
      thenAct  act1: icol := green
    Event set_red ≙ ordinary
      thenAct  act1: icol := red
```

```
logic EVT
spec  ButtonSpec =
  BoolButton with ρ
  then
    Event gobutton ≙ ordinary
      when  grd1: button = true
      thenAct  act1: button := false
    Event pushbutton ≙ ordinary
      thenAct  act1: button := true

spec  mac2 =
  (LightRefined  with σ3)
  and (LightRefined and
         (ButtonSpec with σ5)with σ4)

where
  σ3 = {i_col ↦ cars_colour, u_col ↦ peds_colour,
        ⟨set_green, ordinary⟩
          ↦ ⟨set_cars_green, ordinary⟩,
        ⟨set_red, ordinary⟩
          ↦ ⟨set_cars_red, ordinary⟩}
  σ4 = {i_col ↦ peds_colour, u_col ↦ cars_colour,
        ⟨set_green, ordinary⟩
          ↦ ⟨set_peds_green, ordinary⟩,
        ⟨set_red, ordinary⟩
          ↦ ⟨set_peds_red, ordinary⟩}
  σ5 = {⟨gobutton, ordinary⟩
          ↦ ⟨set_green, ordinary⟩}
```

Fig. 5. A modular institution-based presentation corresponding to the refined machine `mac2` specified in Fig. 3.

have the same name allows the and operator to combine both events' guards and actions and the morphism σ_4 to name the resulting event `set_peds_green`. The resulting specification also contains the event `pushbutton`. The labels given to guards/actions are syntactic sugar to make the specification aesthetically resemble the usual Event-B notation for guards/actions.

Figure 6 uses the refinement syntax available in Hets to specify each of the refinements in the specification of the concrete machine `mac2`:

```
1  refinement REF : Bool to Colours =
2      Bool  ↦ Colours,
3      true  ↦ green,
4      false ↦ red
5      i_go ↦ icol,
6      u_go ↦ ucol,
7      ⟨set_peds_go, ordinary⟩
8         ↦  ⟨set_peds_green, ordinary⟩,
9      ⟨set_peds_stop, ordinary⟩
10        ↦  ⟨set_peds_red, ordinary⟩,
11     ⟨set_cars_go, ordinary⟩
12        ↦  ⟨set_cars_green, ordinary⟩,
13     ⟨set_cars_stop, ordinary⟩
14        ↦  ⟨set_cars_red, ordinary⟩
15 end
```

Fig. 6. Defining the refinement relationships between the concrete and abstract presentations.

Lines 2–4: define the data refinement of *Bool* into *Colours*, with an appropriate mapping for the values.

Lines 5–6: define the refinement of the two boolean variables into their corresponding variables of type *Colour*. In combination with lines 2–4, this corresponds to the gluing invariants on lines 5 and 7 of Fig. 3.

Lines 7–14: define the refinement relation between the four events: this corresponds to the refines statements on lines 14, 20, 23 and 27 of Fig. 3.

5 Conclusion and Future Work

Currently, the core benefit of $\mathcal{EVT}$, our institution for Event-B, is the increased modularity of Event-B specifications via the use of specification-building operators. The concept of refinement, central to Event-B, is also well-developed in the theory of institutions, and we have shown how this can be applied here. Devising meaningful institutions and corresponding morphisms to/from Event-B provides a mechanism not only for ensuring the safety of a particular specification but also, via morphisms, a potential for integration with other formalisms. Interoperability and heterogeneity are significant goals in the field of software engineering, and we believe that the work presented in this paper provides a basis for the integration of Event-B with other formalisms defined in this way.

The Heterogeneous Tool-Set HETS provides a framework for heterogeneous specifications where each formalism is represented as a logic and understood in the theory of institutions [9]. Our logic for $\mathcal{EVT}$ utilises the already existing institution $\mathcal{CASL}$ [10] to account for the $\mathcal{FOPEQ}$ parts of the $\mathcal{EVT}$ institution thus taking advantage of the interoperability/heterogeneity supplied by HETS. $\mathcal{CASL}$ provides sorts and predicates like those written in lines 4–6 from Fig. 4.

At present we can parse, statically analyse and combine specifications written over $\mathcal{EVT}$. Future work includes developing comorphisms to translate between $\mathcal{EVT}$ and other logics in HETS as well as integrating with the provers currently available in HETS (e.g. Isabelle). Comorphisms between these theorem provers and $\mathcal{EVT}$ will allow us to prove our specifications correct in HETS. We envisage that development should take place here to fully take advantage of the prospects for interoperability. A translation from Event-B to $\mathcal{EVT}$ in the future will not only enable us to fully utilise both the *Rodin Platform* and HETS, but will also provide a translational semantics for Event-B using the theory of institutions.

Acknowledgements. The authors would like to acknowledge the reviewers for their helpful comments and Ionut Tutu for his assistance with the presentation of the technical details of our institution for Event-B.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, New York (2010)
2. Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. Int. J. Softw. Tools Technol. Transf. **12**(6), 447–466 (2010)
3. Abrial, J.-R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: application to Event-B. Fundamenta Informaticae **77**(1–2), 1–28 (2007)
4. Achouri, A., Jemni Ben Ayed, L.: UML activity diagram to Event-B: a model transformation approach based on the institution theory. In: Information Reuse and Integration, pp. 823–829, August 2014
5. Goguen, J.A., Burstall, R.M.: Institutions: abstract model theory for specification and programming. J. ACM **39**(1), 95–146 (1992)
6. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D., Latvala, T.: Supporting reuse in Event B development: modularisation approach. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) ABZ 2010. LNCS, vol. 5977, pp. 174–188. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11811-1_14
7. Jastram, M., Butler, P.M.: Rodin User's Handbook: Covers Rodin V.2.8. CreateSpace Independent Publishing Platform (2014)
8. Knapp, A., Mossakowski, T., Roggenbach, M., Glauer, M.: An Institution for simple UML state machines. In: Egyed, A., Schaefer, I. (eds.) FASE 2015. LNCS, vol. 9033, pp. 3–18. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46675-9_1
9. Mossakowski, T., Maeder, C., Lüttich, K.: The heterogeneous tool set, Hets. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_40
10. Mosses, P.D. (ed.): CASL Reference Manual. LNCS, vol. 2960. Springer, Heidelberg (2004). https://doi.org/10.1007/b96103
11. Sanella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. Springer, Heidelberg (2012)

On the Most Suitable Axiomatization of Signed Integers

Hubert Garavel[1,2,3,4](✉)

[1] Inria, Grenoble, France
[2] Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
[3] CNRS, LIG, F-38000 Grenoble, France
[4] Saarland University, Saarbrücken, Germany
hubert.garavel@inria.fr
http://convecs.inria.fr

Abstract. The standard mathematical definition of signed integers, based on set theory, is not well-adapted to the needs of computer science. For this reason, many formal specification languages and theorem provers have designed alternative definitions of signed integers based on term algebras, by extending the Peano-style construction of unsigned naturals using "zero" and "succ" to the case of signed integers. We compare the various approaches used in CADP, CASL, Coq, Isabelle/HOL, KIV, Maude, mCRL2, PSF, SMT-LIB, TLA+, etc. according to objective criteria and suggest an "optimal" definition of signed integers.

1 Introduction

It took a few millennia to properly formalize number theory but, at present, mathematics has sound and well-established concepts for numbers.

In computer science, the situation is different. Following a tradition initiated by Fortran and Algol 60, most programming languages rely on a set of predefined data types, among which numerical types (integers, reals, etc.) have finite domains and usually map to the machine words provided by the underlying hardware. In many cases, the semantics of these types is not defined formally, as it depends on the implementation. Even languages with strong semantic foundations may be incompletely defined if they import predefined types; this is the case, for instance, with the synchronous languages Lustre [31, Sects. 3.1 and 3.2], Esterel [5, Sect. 4.3.1], and Signal [13, Sect. 2.3], which assume the existence of predefined signed integers and floating-point reals, presumably imported from the C language.

Some specification languages use a similar approach, by assuming the existence of numerical types rather than defining them formally. Because specification languages are expected to be more abstract and higher-level than programming languages, such predefined numerical types are usually infinite and the

© IFIP International Federation for Information Processing 2017
Published by Springer International Publishing AG 2017. All Rights Reserved
P. James and M. Roggenbach (Eds.): WADT 2016, LNCS 10644, pp. 120–134, 2017.
https://doi.org/10.1007/978-3-319-72044-9_9

mapping to hardware implementation is often left aside. These are four examples of specification languages in which numbers are taken for granted[1]:

- The VDM language [15] [19, Sect. 3.1.2] has five predefined numerical types, real numbers being the most general one (`nat1` $\subset$ `nat` $\subset$ `int` $\subset$ `rat` $\subset$ `real`).
- The predefined library of PVS [25, Chap. 7] assumes the existence of a universal `number` type, of which the usual numerical types are subsets (`naturalnumber` $\subset$ `integer` $\subset$ `rational` $\subset$ `real` $\subset$ `number`). Actually, PVS defines many more types (e.g., `posnat`, `nonneg_int`, `nzrat`, etc.) that form a lattice, the elements of which are related by PVS judgments and properties.
- The Z notation [16, Sects. 11.2 and B.7] assumes the existence of an unspecified number type $\mathbb{A}$ ("arithmos") that contains naturals and integers ($\mathbb{N}_1 \subset \mathbb{N} \subset \mathbb{Z} \subset \mathbb{A}$). These sets and their operations are defined using high-level statements such as: "*multiplication on integers is characterised by the unique operation under which the integers become a commutative ring with identity element 1*" [16, Sect. B.7.11].
- The B language [20, Sects. 3.4, 5.3, 5.6, and 7.25.2] assumes the existence of the set $\mathbb{Z}$, of which $\mathbb{N}_1$ and $\mathbb{N}$ are subsets ($\mathbb{N}_1 \subset \mathbb{N} \subset \mathbb{Z}$). The language also defines a set `INT` of "concrete integers" that belong to a finite range `MININT`...`MAXINT`, the bounds of which are implementation-dependent, e.g., $(-2^{31}) \ldots (2^{31}-1)$, as well as two subsets $\texttt{NAT}_1$ and `NAT` of `INT` without negative values ($\texttt{NAT}_1 \subset \texttt{NAT} \subset \texttt{INT} \subset \mathbb{Z} \wedge \texttt{NAT}_1 \subset \mathbb{N}_1 \wedge \texttt{NAT} \subset \mathbb{N}$).

Relying on undefined or externally-defined numerical types makes these languages closer to programming languages than formal methods. Their semantics is not entirely defined and properties involving numbers cannot be proven within these languages, unless some specific theories are imported. We believe that a unified semantic definition that encompasses numbers is highly preferable.

The present article addresses the following problem: *what is the best way to define the set $\mathbb{Z}$ of integers formally?* An ideal definition should be mathematically elegant and compatible with the needs of computer-aided verification.

Our motivation for this question arose in 1996 when applying the LOTOS language [14] to an industrial case study that required signed integers: the predefined library of LOTOS, based on ACT-ONE [9] abstract data types, provided only natural numbers, but no signed integers. At that time, no obvious solution could be found in the literature. During the past decades, various approaches have been devised for this problem and implemented in mainstream specification languages and theorem provers. The present article reviews and compares these approaches. We restrict our study to arbitrary large naturals and integers (i.e., the mathematical sets $\mathbb{N}$ and $\mathbb{Z}$), thus excluding finite subranges of $\mathbb{N}$ and $\mathbb{Z}$, especially machine numbers (e.g., 32-bit integers) and modular arithmetic.

[1] In the present article, we distinguish between *naturals* (also: *natural numbers*), which are the elements of $\mathbb{N}$ (i.e., unsigned), and *integers*, which are the elements of $\mathbb{Z}$ (i.e., signed). We assume that $0 \in \mathbb{N}$ and we write $\mathbb{N}_1 =_{def} \mathbb{N} \setminus \{0\}$.

2 Definitions of $\mathbb{N}$

In mathematics, natural numbers can be constructed in two main ways:

- *Construction based on set theory*: assuming that the concept of set preexists, we know from the works of Zermelo, Fraenkel, and Von Neumann that natural numbers can be defined as the following sequence of sets:

$$\begin{aligned} 0 &=_{def} \varnothing \\ 1 &=_{def} 0 \cup \{0\} = \{0\} = \{\varnothing\} \\ 2 &=_{def} 1 \cup \{1\} = \{0, 1\} = \{\varnothing, \{\varnothing\}\} \\ 3 &=_{def} 2 \cup \{2\} = \{0, 1, 2\} = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\} \end{aligned}$$

  and so on, where number $n+1$ is defined as the set $n \cup \{n\}$, i.e., $\{0, 1, \ldots, n\}$. We are not aware of any computer language that uses this approach to define natural numbers. Even formal methods based on set theory, such as VDM, Z, B, or TLA+ [18] do not define their naturals this way.
- *Construction based on algebraic terms:* we know from Peano axioms that natural numbers can be represented as algebraic terms built with two constructors [$\texttt{zero} : \rightarrow \mathbb{N}$] and [$\texttt{succ} : \mathbb{N} \rightarrow \mathbb{N}$]. Because this approach lends itself well to machine execution and automated reasoning, it has been adopted by most computer languages that formally define their numbers.

  The simplicity of this approach fits theoretical needs well but faces practical limitations: Peano-style naturals are encoded in base one (*unary* representation), which is cumbersome when writing large numbers, and inefficient when directly executing algebraic specifications by giving them as input to some rewrite engine or encoding them into some language (e.g., a functional programming language) that supports inductive types and pattern matching; in practice, unary representation often causes stack overflow for naturals larger than 10^6.

  For this reason, refined approaches have been proposed to represent naturals more compactly and reduce the amount of rewrite steps; this is usually done by encoding numbers in a base different from one, e.g., two [2,4], three [8], four [7], ten [2,4,17,29,33], sixteen [2], or in some arbitrary base [32, Systems DA and JP]. In this article, we stick to the unary representation, since it is used by a majority of specification languages and software tools.

The choice between set theory and algebraic terms for defining unsigned naturals can also be found in the construction of signed integers. In the sequel, we examine each approach in turn.

3 Approach 1: Definition of $\mathbb{Z}$ Using Set Theory

In mathematical textbooks, the set of integers is defined as $\mathbb{Z} = (\mathbb{N} \times \mathbb{N})/\!\sim$, where $\times$ is the cartesian product and $\sim$ the equivalence relation such that $(x, y) \sim (x', y') \Longleftrightarrow x + y' = x' + y$.

This approach has been retained for defining integers in CASL [23, Sect. V:2, p. 381] and Isabelle/HOL [28, Sect. 52 p. 586]. It has the merit of exhibiting a nice analogy 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 such that $(x, y) \sim (x', y') \iff x \cdot y' = x' \cdot y$ — but the analogy does not hold beyond this point, as the set $\mathbb{R}$ of real numbers cannot be defined as a quotient set of $\mathbb{Q} \times \mathbb{Q}$.

Such a set-theoretic definition of integers has various drawbacks: (i) it relies on cartesian product and quotient set, which are involved notions, compared to the simplicity of Peano axioms; (ii) it makes the definition of integers very different from that of naturals; (iii) it goes against the intuition as it builds a two-dimensional surface where a half line towards negative integers would be sufficient; (iv) it does not support reasoning by induction on integers, as pointed out, e.g., in [24, Sect. 8.4 p. 165]; (v) it is not optimal computationally, neither in memory (the cartesian product suggests that it takes two naturals to build one integer) nor in time (the quotient operation requires two additions and an equality test to compare two integers).

Having mentioned this approach, we now consider, in the remainder of this article, alternative definitions based upon algebraic terms rather than set theory.

4 Formal Definitions

4.1 Syntactic Notations

Following the established terminology of algebraic specification, we define a *sort*[2] to be a collection of algebraic terms. These terms should be well-typed, meaning all the usual constraints arising from the arity of operations, the sorts of operation arguments, and the sorts of operation results must be taken into account.

If t is a term of sort S, if f is a unary operation $[f : S \to S]$, and if n is a natural number, let $f^n(t)$ be the term defined inductively by $f^0(t) =_{def} t$ and $f^{n+1}(t) =_{def} f(f^n(t))$. For instance, $\texttt{succ}^3(\texttt{zero}) = \texttt{succ}(\texttt{succ}(\texttt{succ}(\texttt{zero})))$.

As often as possible, we try to split the set of operations into *constructors*, which determine the set of possible values of a sort, and *non-constructors*, which are defined functions specified using algebraic equations or term rewrite rules. We define the *constructors of a sort S* to be all constructors that return a result of sort S. For instance, the constructors of natural numbers defined in the Peano style are `zero` and `succ`; all other operations also returning a natural number (e.g., addition, subtraction, multiplication, etc.) are seen as non-constructors.

We define a *ground normal form* to be any well-typed term built using constructors only; consequently, a ground normal form cannot contain free variables or non-constructors. We define the *domain* of a sort S to be the set, noted $dom(S)$, of all ground normal forms of sort S. We define the *image* of a constructor f to be the set, noted $img(f)$, of all ground normal forms whose top-level constructor is f. For instance, if `nat` is the sort defined by the two constructors $[\texttt{zero} : \to \texttt{nat}]$ and $[\texttt{succ} : \texttt{nat} \to \texttt{nat}]$, its domain

[2] We prefer using *sort* rather than *type*, which is often given a different meaning.

is $dom(S) = \{\texttt{succ}^n(\texttt{zero}) \mid n \in \mathbb{N}\}$ and the images of its constructors are $img(\texttt{zero}) = \{\texttt{zero}\}$ and $img(\texttt{succ}) = \{\texttt{succ}^n(\texttt{zero}) \mid n \in \mathbb{N} \setminus \{0\}\}$.

4.2 Semantic Denotations

Our goal is to study how mathematical integers can be conveniently represented by sorts, constructors, and algebraic terms. We thus carefully distinguish between *notations* (i.e., terms) and *denotations* (i.e., numbers from $\mathbb{Z}$).

If f is a constructor, we write $[\![f]\!]$ its intended denotation, which we conveniently formulate as a λ-expression, in which all the trivial conversions between sorts or types (i.e., from Peano-like terms to $\mathbb{N}$, or from $\mathbb{N}$ to $\mathbb{Z}$) are implicitly performed. For instance, $[\![\texttt{zero}]\!] = 0$ and $[\![\texttt{succ}]\!] = \lambda n.(n + 1)$.

Having defined the denotation $[\![f]\!]$ of a constructor f, we extend this notion to define the denotation $[\![t]\!]$ of a ground normal form t by induction on the syntax of terms: $[\![f(t_1, \ldots, t_n)]\!] =_{def} [\![f]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$. For instance, $[\![\texttt{succ}(\texttt{zero})]\!] = [\![\texttt{succ}]\!]([\![\texttt{zero}]\!]) = (\lambda n.(n + 1))(0) = 1$.

A necessary condition for a sort S to represent $\mathbb{Z}$ is that $[\![.]\!]$ is a surjection from dom(S) to $\mathbb{Z}$, i.e., $(\forall n \in \mathbb{Z})\ (\exists\, t \in dom(S) \mid [\![t]\!] = n)$, meaning that each integer can be denoted by at least one ground normal form of sort S.

Additionally, we say that the constructors of sort S are *free* if $[\![.]\!]$ is an injection[3] from dom(S) to $\mathbb{Z}$, i.e., $(\forall t_1, t_2 \in dom(S))\ ([\![t_1]\!] = [\![t_2]\!] \Rightarrow t_1 = t_2)$, where $t_1 = t_2$ means the syntactic identity of terms t_1 and t_2. Thus, if the constructors are free, each integer is denoted by exactly one ground normal form of S. This definition remains compatible with the usual, more general definition stating that constructors are free if any two syntactically different ground normal forms cannot be proven equal (or be rewritten one into the other, or both into a common third term).

Let S be a sort intended to represent $\mathbb{Z}$, and $f_1, \ldots, f_n$ its constructors (with $n \geq 1$). It follows from the above definitions that these constructors are free iff all functions $[\![f_i]\!]$ are injective and the sets of denotations of the images $img(f_1)$, ..., $img(f_n)$ form a partition[4] of $\mathbb{Z}$, i.e., $\mathbb{Z} = \{[\![t]\!] \mid t \in img(f_1)\} \uplus \ldots \uplus \{[\![t]\!] \mid t \in img(f_n)\}$, where $\uplus$ denotes the disjoint union. Notice that if a sort has a single constructor, this constructor is not necessarily free, as it may be non-injective; various examples will be seen below.

To compare the various definitions of $\mathbb{Z}$ based on algebraic terms, we quantify the complexity of each approach using two natural numbers (m, n) defined as follows: given a sort S that represents $\mathbb{Z}$, n is the number of constructors (noted f_1, ..., f_n) of S, and m is the number of different sorts that occur in the arguments of f_1, ..., f_n; in particular, m is equal to one if sort S is defined directly, without referencing another sort. The number n of constructors adversely impacts the conciseness of non-constructor operations having arguments of sort S: a unary function is likely to require n equations or rewrite rules; a binary function is likely to require n^2 equations or rewrite rules, etc. We write $\mathbf{NF}_n^m$ (resp. $\mathbf{F}_n^m$) an approach based on m sorts and n non-free (resp. free) constructors.

[3] And, hence, a bijection.
[4] In particular, are pairwise disjoint.

5 Definitions of $\mathbb{Z}$ Using Non-free Constructors

In this section, we review four approaches that quickly come to mind when trying to define integers using algebraic terms. However, these approaches lead to non-free constructors which, we believe, are not optimal. The merit of an approach is inversely proportional to its number of *collisions*, i.e., the maximal number of distinct ground normal forms that can denote the same integer value.

5.1 Approach 2: $\mathbf{NF}_1^2$ – Two Sorts and One Non-free Constructor

Interestingly, the set-theoretic approach of Sect. 3 can be reformulated in algebraic style by defining a sort `Int` built upon the preexisting sort `Nat` using a single constructor [$\texttt{pair} : \texttt{Nat}, \texttt{Nat} \rightarrow \texttt{Int}$] such that $[\![\texttt{pair}]\!] = \lambda m.\lambda n.(m - n)$. Clearly, this unique constructor is not free: for instance, $[\![\texttt{pair}(\texttt{succ}(\texttt{zero}), \texttt{zero})]\!] = [\![\texttt{pair}(\texttt{succ}(\texttt{succ}(\texttt{zero})), \texttt{succ}(\texttt{zero})]\!] = 1$. More generally, each integer $n \geq 0$ can be represented by an infinite number of terms $\{\texttt{pair}(\texttt{succ}^{k+n}(\texttt{zero}), \texttt{succ}^k(\texttt{zero})) \mid k \in \mathbb{N}\}$ and each integer $n \leq 0$ can be represented by an infinite number of terms $\{\texttt{pair}(\texttt{succ}^k(\texttt{zero}), \texttt{succ}^{k-n}(\texttt{zero})) \mid k \in \mathbb{N}\}$; the number of collisions is thus $\aleph_0$, which is a most undesirable property.

5.2 Approach 3: $\mathbf{NF}_3^1$ – One Sort and Three Non-free Constructors

To define a sort `Int` representing $\mathbb{Z}$, an intuitive idea, used in the library[5] of the KIV tool [10, p. 8 and Sect. 5.1 p. 42], is to take the Peano system [$\texttt{zero} :\rightarrow \texttt{Int}$] and [$\texttt{succ} : \texttt{Int} \rightarrow \texttt{Int}$] and extend it with a third constructor [$\texttt{pred} : \texttt{Int} \rightarrow \texttt{Int}$] such that $[\![\texttt{pred}]\!] = \lambda n.(n - 1)$, while `zero` and `succ` keep their usual denotations: $[\![\texttt{zero}]\!] = 0$ and $[\![\texttt{succ}]\!] = \lambda n.(n+1)$. Definitions of integers involving such a predecessor operation have been studied in [32, System SP] [29], and also in [3,4], where the predecessor operation is a non-constructor. The fact that $[\![\texttt{pred}]\!] = [\![\texttt{succ}]\!]^{-1}$ gives an appealing symmetry, as `pred` and `succ` progress in opposite directions.

Unfortunately, these constructors are not free, the simplest counterexample being $[\![\texttt{pred}(\texttt{succ}(\texttt{zero}))]\!] = [\![\texttt{succ}(\texttt{pred}(\texttt{zero}))]\!] = 0$. Each integer $n \geq 0$ can be represented by an infinite number of terms, e.g., $\{\texttt{pred}^k(\texttt{succ}^{n+k}(\texttt{zero})) \mid k \in \mathbb{N}\}$ or, more generally, any combination (in any order) of k applications of `pred` mixed with $(k + n)$ applications of `succ` — a dual remark holds if $n \leq 0$. The number of collisions is thus $\aleph_0$.

5.3 Approach 4: $\mathbf{NF}_1^3$ – Three Sorts and One Non-free Constructor

Another approach, used in the type library of PSF [30, Sect. 8] [21, Sect. 2.5], is based on the idea that an integer is a natural with a sign. Assuming the existence of a sort `Sign` with two free constructors [$\texttt{plus} :\rightarrow \texttt{Sign}$] and [$\texttt{minus} :\rightarrow \texttt{Sign}$],

[5] https://swt.informatik.uni-augsburg.de/swt/projects/lib/basic/specs/int-basic1/export/unit.xml.

the sort `Int` can be defined using a constructor [$\texttt{pair} : \texttt{Sign}, \texttt{Nat} \rightarrow \texttt{Int}$] such that $[\![\texttt{pair}]\!] = \lambda s.\lambda n.(\text{if } s = \texttt{plus} \text{ then } n \text{ else } -n)$.

This unique constructor is not free, because of the collision $[\![\texttt{pair}(\texttt{plus}, \texttt{zero})]\!] = [\![\texttt{pair}(\texttt{minus}, \texttt{zero})]\!] = 0$, a situation that we will refer to as the *double-zero issue*[6]. This is the only collision in this approach.

5.4 Approach 5: $\mathbf{NF}_2^2$ – Two Sorts and Two Non-free Constructors

A variant of the previous approach $\mathbf{NF}_1^3$ of Sect. 5.3 does not rely on the sort `Sign` but uses instead two constructors [$\texttt{pos} : \texttt{Nat} \rightarrow \texttt{Int}$] and [$\texttt{neg} : \texttt{Nat} \rightarrow \texttt{Int}$] such that $[\![\texttt{pos}]\!] = \lambda n.n$ and $[\![\texttt{neg}]\!] = \lambda n.(-n)$. This approach is used in TLA+ [18, Sect. 18.4 p. 347], where *Int* is defined as $Nat \cup \{Zero - n \mid n \in Nat\}$. Other variants in which `neg` is the unary-minus operator [$\texttt{neg} : \texttt{Int} \rightarrow \texttt{Int}$] have been studied in, e.g., [3,4,7,17,33,34].

Again, these constructors are not free, since $[\![\texttt{pos}(\texttt{zero})]\!] = [\![\texttt{neg}(\texttt{zero})]\!] = 0$. Some tool designers are aware of this double-zero issue and propose to address it in various ways:

– *Dependent types:* One may wish to rule out undesirable terms such as `pair(minus, zero)` or `neg(zero)`. This is the approach followed in the SMT-LIB standard [1, Fig. 3.3 p. 34 and 35], which states: "*The set of values for the* `Int` *sort consists of all numerals and all terms of the form* $(-n)$ *where* n *is a numeral other than 0*". Prohibiting the algebraically-closed term (-0) can be done trivially, at the level of syntax, but things are far less easy with general terms of the form $(-e)$, where e is an expression containing free variables and/or arbitrary user-defined functions. Deciding whether such terms belong to `Int` is equivalent to answer the question $e = 0$, which is undecidable in the general case and would anyway require involved proofs in each particular case. Alternative approaches with efficient decision procedures are thus preferable.
– *Equations relating constructors:* Other approaches use equations or rewrite rules in order to formalize the relations that may exist between constructors. For instance, [3,4] handle the double-zero issue by adding two equations $-0 = 0$ and $- - n = n$, which eliminate unwanted terms by bringing them under a suitable normal form. However, this approach weakens the difference between constructors and non-constructors and often raises termination issues. There exists a classical technique (see [27, Sect. 3], [11, Sect. 3.3], etc.) to eliminate non-free constructors by replacing each of them with two distinct operations: a free constructor and a non-constructor. Unfortunately, such a dissociation of roles does not give exploitable results when applied to the construction of $\mathbb{Z}$.

[6] Notice that the IEEE 754 standard for floating-point numbers also defines two zeros, -0.0 and $+0.0$, which are equivalent in most cases but can still be distinguished, e.g., using the `signbit` primitive.

– *Subtypes:* Another approach relies on subtyping and operator overloading to avoid the double-zero issue. For instance, the predefined type library of Maude [6, pp. 45–46, 116, and 248–251] makes plain use of order-sorted specifications to define integers as follows: (i) it first defines the sort `Nat` of naturals, together with the subsort `NzNat` of `Nat`, which contains all naturals different from zero; (ii) it then defines the sort `Int` of integers, together with the subsort `NzInt` of `Int`, which contains all integers different from zero and is also a supersort of `NzNat`; (iii) a "unary-minus" constructor $[- : \texttt{NzNat} \to \texttt{NzInt}]$ is defined, such that $[\![-]\!] = \lambda n.(-n)$; (iv) this constructor is extended to integer sorts by introducing two subsort-overloaded non-constructors $[- : \texttt{NzInt} \to \texttt{NzInt}]$ and $[- : \texttt{Int} \to \texttt{Int}]$; (v) these non-constructors are defined by adding two equations $-0 = 0$ and $--n = n$. This approach is the only one in which values of sort `Nat` are also values of sort `Int`; such implicit type conversion closely reflects the mathematical fact that $\mathbb{N} \subset \mathbb{Z}$, but might become a nuisance when turning an algebraic specification with arbitrary large numbers into a concrete implementation written in a programming language with bounded numbers (e.g., the C language with its `int` and `unsigned int` types): to avoid silent, yet unsafe numeric overflows, the programmer will have to detect all implicit type conversions in the formal specification and make them explicit in the program.

These various approaches, even if correct, are heavy. Lighter approaches using only the same basic concepts as for the Peano-style definition of naturals are desirable. All in one, we believe that definitions based on non-free constructors are sub-optimal. Therefore, in the next section, we investigate simpler approaches genuinely based on free constructors.

6 Definitions of $\mathbb{Z}$ Using Free Constructors

Free constructors obviously lead to simpler mathematics. On the practical side too, free constructors are desirable, for at least three reasons: (i) terms defined using free constructors have a unique representation, so that, in principle, no memory bit is wasted due to the existence of multiple representations of the same value; (ii) because each term has a unique representation, comparison of values relies upon syntactic identity, which can be efficiently implemented using bit-string comparison and/or hashing; no extra computation is required to compare multiple representations of the same value or to bring terms under a canonical form first; (iii) increasingly many computer languages (e.g., functional, object-oriented, etc.) are supporting pattern matching with free constructors, whereas the number of tools (e.g., term rewrite engines, tools for algebraic specifications, etc.) that can handle non-free constructors is shrinking, as many tools are no longer maintained[7].

[7] See http://rewriting.loria.fr/systems.html to learn about such halted tools.

6.1 Approach 6: $\mathbf{F}_2^3$ – Three Sorts and Two Free Constructors

The approaches $\mathbf{NF}_1^3$ of Sect. 5.3 and $\mathbf{NF}_2^2$ of Sect. 5.4 (i.e., defining an integer as the combination of a sign and a natural) can be reused and adapted to a free-constructor setting. This requires to introduce a new sort and to break the symmetry between the negative and positive cases, so as to forbid, by means of statically-decidable type checking, one of the two zero values.

For instance, mCRL2 [12, Appendices B.2, B.3, B.4, and D.5] has three predefined sorts: `Pos`, which denotes $\mathbb{N} \setminus \{0\}$ and is encoded in binary form, `Nat`, which denotes $\mathbb{N}$ and is defined in Peano style using two free constructors [$@\texttt{c0} :\rightarrow \texttt{Nat}$] and [$@\texttt{cNat} : \texttt{Pos} \rightarrow \texttt{Nat}$], and `Int`, which denotes $\mathbb{Z}$ and is also defined using two free constructors [$@\texttt{cInt} : \texttt{Nat} \rightarrow \texttt{Int}$] and [$@\texttt{cNeg} : \texttt{Pos} \rightarrow \texttt{Int}$] such that $[\![@\texttt{cInt}]\!] = \lambda n.n$ and $[\![@\texttt{cNeg}]\!] = \lambda n.(-n)$. The elements of $\mathbb{Z}$ are thus encoded as follows: $n \geq 0 \mapsto @\texttt{cInt}(n)$ and $n < 0 \mapsto @\texttt{cNeg}(-n)$.

As with Maude, the double-zero issue is avoided by a deliberate dissymmetry between both constructors `@cInt` and `@cNeg`, whose arguments have sorts `Nat` and `Pos`, respectively. But, contrary to Maude, `Nat` and `Pos` are distinct sorts, not subtypes; this requires explicit type conversions and may create confusion for users, who must carefully distinguish between natural and positive values.

6.2 Approach 7: $\mathbf{F}_3^2$ – Two Sorts and Three Free Constructors

A different approach can be found in the standard library of the Coq theorem prover[8] (see [26] for confirmation). To construct natural numbers, this library defines a sort[9] `nat` built in Peano style using two constructors [$\texttt{O} :\rightarrow \texttt{nat}$] and [$\texttt{S} : \texttt{nat} \rightarrow \texttt{nat}$]. In an alternative approach, the Coq library also contains a sort `positive` that represents $\mathbb{N} \setminus \{0\}$, i.e., all natural numbers greater or equal to one — such numbers are encoded in binary form, as unbounded strings of bits inductively defined by three free constructors [$\texttt{xH} :\rightarrow \texttt{positive}$], [$\texttt{xO} : \texttt{positive} \rightarrow \texttt{positive}$], and [$\texttt{xI} : \texttt{positive} \rightarrow \texttt{positive}$]. Finally, Coq defines a sort `N` (also intended to represent $\mathbb{N}$) as the union of all `positive` values and of a constant `N0` denoting 0; this is done using two constructors [$\texttt{Npos} : \texttt{positive} \rightarrow \texttt{N}$] and [$\texttt{N0} :\rightarrow \texttt{N}$].

To construct integer numbers, the Coq library defines a sort `Z` that only uses the sort `positive`, but neither `nat` nor `N`. The sort `Z` is built using three free constructors [$\texttt{Z0} :\rightarrow \texttt{Z}$], [$\texttt{ZPos} : \texttt{positive} \rightarrow \texttt{Z}$], and [$\texttt{ZNeg} : \texttt{positive} \rightarrow \texttt{Z}$] such that $[\![\texttt{Z0}]\!] = 0$, $[\![\texttt{ZPos}]\!] = \lambda n.n$, and $[\![\texttt{ZNeg}]\!] = \lambda n.(-n)$. The elements of $\mathbb{Z}$ are thus encoded as follows: $0 \mapsto \texttt{Z0}$, $n > 0 \mapsto \texttt{ZPos}(n)$, and $n < 0 \mapsto \texttt{ZNeg}(-n)$.

Notice that this approach could be slightly adapted to define `Z` using sort `N` (or `nat`) rather than `positive`. In such case, the three constructors would become [$\texttt{Z0} :\rightarrow \texttt{Z}$], [$\texttt{ZPos} : \texttt{N} \rightarrow \texttt{Z}$], and [$\texttt{ZNeg} : \texttt{N} \rightarrow \texttt{Z}$] such that $[\![\texttt{Z0}]\!] = 0$, $[\![\texttt{ZPos}]\!] = \lambda n.(n+1)$, and $[\![\texttt{ZNeg}]\!] = \lambda n.(-n-1)$. The elements of $\mathbb{Z}$ would be thus encoded as follows: $0 \mapsto \texttt{Z0}$, $n > 0 \mapsto \texttt{ZPos}(n-1)$, and $n < 0 \mapsto \texttt{ZNeg}(-n-1)$.

[8] http://coq.inria.fr/library.

[9] I.e., a *datatype* in the terminology of Coq.

Contrary to some aforementioned approaches, the Coq library handles negative and positive numbers symmetrically. Even if this approach is modified (as explained in the previous paragraph) to use only two sorts, it still relies upon three constructors, which increases the length of definitions and proofs for most operations on integers; in particular, the usual binary operators are likely to require nine equations (rather than four when only two constructors are used).

6.3 Approach 8: $\mathbf{F}_2^2$ – Two Sorts and Two Free Constructors

We have seen so far two definitions of $\mathbb{Z}$ based on free constructors: one with three sorts and two constructors (i.e., $\mathbf{F}_2^3$ of Sect. 6.1), another one with two sorts and three constructors (i.e., $\mathbf{F}_3^2$ of Sect. 6.2). At this point, a natural question is: *is there a simpler definition with two sorts and two constructors only?*

The answer is: yes. Such a solution was found in 1996 when trying to extend the standard library of LOTOS with signed integers, and it has been distributed as part of the CADP toolbox since February 1997.

This approach uses two sorts `Nat`, which denotes $\mathbb{N}$, and `Int`, which denotes $\mathbb{Z}$. The sort `Int` is built from `Nat`, without the need for a third sort, using two free constructors [`pos` : `Nat` $\rightarrow$ `Int`] and [`neg` : `Nat` $\rightarrow$ `Int`] such that $[\![\texttt{pos}]\!] = \lambda n.n$ and $[\![\texttt{neg}]\!] = \lambda n.(-n-1)$. The elements of $\mathbb{Z}$ are thus encoded as follows: $n \geq 0 \mapsto \texttt{pos}(n)$ and $n < 0 \mapsto \texttt{neg}(-n-1)$.

Although this approach does not handle negative and positive numbers symmetrically, it enjoys a nice symmetry property, as the denotations of constructors are both involutive functions, i.e., $[\![\texttt{pos}]\!] = [\![\texttt{pos}]\!]^{-1}$ and $[\![\texttt{neg}]\!] = [\![\texttt{neg}]\!]^{-1}$, or also $[\![\texttt{pos}]\!]([\![\texttt{pos}]\!](n)) = n$ and $[\![\texttt{neg}]\!]([\![\texttt{neg}]\!](n)) = n$, which seems a counterpart of the algebraical identities $+(+n) = n$ and $-(-n) = n$.

The constructor pair (`pos`, `neg`) is unique in this respect, and there is no simpler solution: consider the set Φ of involutive functions φ over $\mathbb{Z}$, i.e., $(\forall n)\ (\varphi(\varphi(n)) = n)$; consider the "simple" elements of Φ, namely affine functions such that $(\forall n)\ (\varphi(n) =_{\scriptscriptstyle def} an+b)$, where a and b are constants; the "simple" involutive solutions are either $\lambda n.n$ or all functions of the form $\lambda n.(-n+b)$; among the restrictions to $\mathbb{N}$ of these solutions, the only pair of free constructors is `pos` (which corresponds to $a = 1$ and $b = 0$) and `neg` (which corresponds to $a = -1$ and $b = -1$), thus ensuring that $\mathbb{Z} = \{[\![\texttt{pos}]\!](n) \mid n \in \mathbb{N}\} \uplus \{[\![\texttt{neg}]\!](n) \mid n \in \mathbb{N}\}$.

It is worth noting that this approach supports straightforward induction on integers, which is lacking in some other approaches — see, e.g., [24, Sects. 8.4 and 8.4.3] for a discussion concerning Isabelle/HOL. Using the `pos` and `neg` constructors, induction on integers can be achieved by two inductions on naturals: firstly, one proves that the property P hold for `pos`(0) and that, if P holds for `pos`(n), it also holds for `pos`$(n+1)$; secondly, one proves that P holds for `neg`(0) and that, if P holds for `neg`(n), it also holds for `neg`$(n+1)$.

6.4 Approach 9: $\mathbf{F}_1^2$ – Two Sorts and One Constructor

Given that $\mathbb{Z}$ can be defined as $\mathbb{Z} =_{\scriptscriptstyle def} (\mathbb{N} \times \mathbb{N})/\!\sim$ and that there exist bijections from $\mathbb{N}^2$ to $\mathbb{N}$ (e.g., diagonal enumeration), it is possible to define the sort `int` using a single constructor [$\texttt{map} : \texttt{nat} \to \texttt{int}$] that is a bijection from $\mathbb{N}$ to $\mathbb{Z}$. There are various choices for map; the simplest definition is likely to be the following one: $[\![\texttt{map}]\!] =_{def} \lambda n.(\text{if } \texttt{even}(n) \text{ then } n/2 \text{ else } -(n+1)/2)$, where [$\texttt{even} : \texttt{nat} \to \texttt{bool}$] is the predicate characterizing even natural numbers. This definition gives, as n increases, the following sequence of values for $\texttt{map}(n)$: 0, -1, 1, -2, 2, -3, 3, -4, 4, etc. The elements of $\mathbb{Z}$ are thus encoded as follows: $n \geq 0 \mapsto \texttt{map}(2n)$ and $n < 0 \mapsto \texttt{map}(-2n-1)$.

This approach $\mathbf{F}_1^2$ is related to the approach $\mathbf{F}_2^2$ of Sect. 6.3 by two identities: $\texttt{pos}(n) = \texttt{map}(2n)$ and $\texttt{neg}(n) = \texttt{map}(2n+1)$. But, even if having a single constructor map is a form of minimality, it does not make the definitions of the usual non-constructors (+, <, etc.) more concise than using the two constructors of approach $\mathbf{F}_2^2$. Indeed, most definitions still need to distinguish two cases, depending whether some argument is odd or even; approach $\mathbf{F}_1^2$ checks this using conditional equations, whereas approach $\mathbf{F}_2^2$ uses pattern matching. For instance, the incrementation function [$\texttt{incr} : \texttt{int} \to \texttt{int}$] requires two conditional equations:

```
incr (map (x)) = map (x + 2)    if even (x) = true
incr (map (x)) = map (x - 2)    if even (x) = false
```

Such a systematic reliance on conditional equations, parity tests, and divisions by two has, at least, three drawbacks: (i) the definitions are neither elegant nor intuitive; (ii) reasoning by induction is not easy; (iii) direct implementation in algebraic or functional languages is not efficient, since parity tests cost $O(n)$ in time (before deciding if a number is odd or even, one needs to visit all its cons subterms).

6.5 Approach 10: $\mathbf{F}_3^1$ – One Sort and Three Free Constructors

After the WADT'2016 presentation in Gregynog, Lutz Schröder suggested to the author yet another approach, in which the sort int is defined using three free constructors: [$\texttt{zero} :\to \texttt{int}$], [$\texttt{nego} :\to \texttt{int}$][10], and [$\texttt{succ} : \texttt{int} \to \texttt{int}$] such that $[\![\texttt{zero}]\!] = 0$, $[\![\texttt{nego}]\!] = -1$, and $[\![\texttt{succ}]\!] = \lambda n.(\text{if } n \geq 0 \text{ then } n+1 \text{ else } n-1)$. The elements of $\mathbb{Z}$ are thus encoded as follows: $0 \mapsto \texttt{zero}$, $n > 0 \mapsto \texttt{succ}^n(\texttt{zero})$, $-1 \mapsto \texttt{nego}$, and $n < -1 \mapsto \texttt{succ}^{-n-1}(\texttt{nego})$.

At first sight, this approach looks truly beautiful, as it is the simplest possible extension of the Peano system, to which only one constructor nego of null arity is added. Moreover, the definition of sort int is self-contained and does not rely on the existence of a sort nat. A similar approach appears in [8, Sect. 4], where integers are written in base 2 starting from two constants 0 and -1.

However, the definitions of non-constructors can be algorithmically involved with this approach, due to the dual nature of succ (which means either incrementation on positive numbers or decrementation on negative numbers, and might thus complicate induction proofs) and because the sign of a number cannot be

[10] We name so this constructor, as a shorthand for *NEGative One*.

immediately determined without traversing all `succ` constructors until a terminal constant (`zero` or `nego`) is reached. Some operations are nevertheless easy to express; this is the case, for instance, of the two predicates [`isneg` : `int` $\rightarrow$ `bool`] and [`ispos` : `int` $\rightarrow$ `bool`] that check whether an integer is strictly negative or positive-or-null:

```
isneg (zero) = false                   ispos (zero) = true
isneg (nego) = true                    ispos (nego) = false
isneg (succ (x)) = isneg (x)           ispos (succ (x)) = ispos (x)
```

But other operations, even the simplest ones, are much less intuitive. For instance, incrementation [`incr` : `int` $\rightarrow$ `int`] is tricky because it requires either to insert one `succ` if the argument is positive or to delete one `succ` if the argument is negative. We believe that this cannot be done without introducing an auxiliary operation [`buff` : `int` $\rightarrow$ `int`] that keeps one `succ` in a virtual buffer until the terminal constant gets known, an information that is required to take an insertion-or-deletion decision:

```
incr (zero) = succ (zero)              buff (zero) = succ (succ (zero))
incr (nego) = zero                     buff (nego) = nego
incr (succ (x)) = buff (x)             buff (succ (x)) = succ (buff (x))
```

To avoid, such intricacies, one can resort to conditional equations, the premises of which use the aforementioned `isneg` and `ispos` predicates. This way, incrementation could be defined more concisely:

```
incr (x) = succ (x)     if ispos (x)
incr (nego) = zero
incr (succ (x)) = x     if isneg (x)
```

Such equations would be very similar to those of approach $\mathbf{F}_2^2$ of Sect. 6.3. Actually, both approaches are related by the two identities $\texttt{pos}(\texttt{succ}^n(\texttt{zero})) = \texttt{succ}^n(\texttt{zero}) = n$ and $\texttt{neg}(\texttt{succ}^n(\texttt{zero})) = \texttt{succ}^n(\texttt{nego}) = -n-1$. The approach $\mathbf{F}_2^2$ is yet simpler: it uses normal equations rather than conditional ones, and determines the sign of a number in time $O(1)$ by pattern matching on the top-level constructor of the term, rather than in time $O(n)$ by invoking the predicates `isneg` and `ispos` that visit all subterms to reach an innermost terminal constant.

As a final remark, any sort S defined by $(k+1)$ constructors consisting of k constants $[f_i :\longrightarrow S]_{i\in\{0,\ldots,k-1\}}$ and one successor function [`succ` : $S \rightarrow S$][11] also represents $\mathbb{N}$, which can be seen by taking $(\forall i \in \{0,\ldots,k-1\})$ $(\llbracket f_i \rrbracket =_{def} i)$ and $\llbracket \texttt{succ} \rrbracket =_{def} \lambda n.(n+k)$; under these definitions, the constructors of S are free.

7 Conclusion

We have shown that the standard definition of signed integers found in mathematical textbooks (namely, $\mathbb{Z} =_{def} (\mathbb{N}\times\mathbb{N})/\!\sim$) is not well-adapted to the needs

[11] This definition generalizes the Peano system (for $k = 1$) and the approach presented in this section (for $k = 2$).

of computer science. It has been argued, e.g. in [22, p. 86], that this standard definition, although less straightforward and natural than a term-algebra approach[12], should nevertheless be preferred because it leads to shorter definitions and proofs by avoiding subdivision into a large number of cases. This argument predates the computer era: it might have been valid when all proofs had to be done manually, but is less relevant today, as interactive or automated theorem provers play an ever-increasing role and handle case disjunctions much better than humans. Moreover, it is unsure that the definitions of arithmetic and relational operators are significantly longer when integers are defined in Peano style with only a few constructors.

In computer science, there is consensus to specify naturals using the Peano constructors `zero` and `succ`, but no consensus at all on how integers should be specified. Leaving aside languages that assume the existence of predefined integers, we reviewed ten different ways of defining integers formally: the set-theoretical approach, four approaches based on non-free constructors ($\mathbf{NF}_1^2$, $\mathbf{NF}_3^1$, $\mathbf{NF}_1^3$, and $\mathbf{NF}_2^2$), and five approaches based on free constructors ($\mathbf{F}_2^3$, $\mathbf{F}_3^2$, $\mathbf{F}_2^2$, $\mathbf{F}_1^2$, and $\mathbf{F}_3^1$).

Such a diversity is enjoyable, but too many diverging approaches can be a nuisance. Given the practical usefulness of integers, a common approach would be desirable, so as to ease tool interoperability and enable formal specifications and proofs to be reused. In this respect, the approach $\mathbf{F}_2^2$ of Sect. 6.3 would be the best candidate: it is simple, enjoys elegant mathematical properties, allows induction over integers, leads to concise definitions for the usual operations on integers, and has been implemented in the CADP toolbox since 1997.

Finally, considering the alternative approaches that propose more efficient representations for unsigned naturals than the Peano-style unary representation, $\mathbf{F}_2^2$ is orthogonal to some of these approaches, e.g., [2], and could be combined with them to provide efficient representations for signed integers as well.

Acknowledgements. The author is grateful to Radu Mateescu and Mihaela Sighireanu who, in 1997 at INRIA Grenoble, helped writing LOTOS algebraic equations to show that the approach $\mathbf{F}_2^2$ was feasible. Acknowledgements are also due to Holger Hermanns and Gilles Nies (Saarland University), and to Jan Friso Groote (Technical University of Eindhoven) for their valuable remarks. Jan Bergstra and Alban Ponse (University of Amsterdam) provided the author with recent bibliographic references. This article benefited from the lively WADT'2016 discussions in Gregynog, especially with Alexander Knapp (Augsburg University), who mentioned KIV, with Narciso Marti-Oliet (Conplutence University of Madrid), who gave informed comments about Maude, and with Lutz Schröder (Friedrich-Alexander-Universität Erlangen-Nürnberg), who suggested the approach $\mathbf{F}_3^1$. Frédéric Lang, Wendelin Serwe, and Hugues Evrard (INRIA Grenoble) proof-read earlier versions of this article.

[12] [22] suggests here the approach $\mathbf{NF}_3^1$ of Sect. 5.2 based on three non-free constructors.

References

1. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard - Version 2.5, 93 p., June 2015
2. Bergstra, J.A.: Four complete datatype defining rewrite systems for an abstract datatype of natural numbers (version 3). Report TCS1407v3, University of Amsterdam (UvA), The Netherlands, June 2014
3. Bergstra, J.A., Ponse, A.: Datatype defining rewrite systems for the ring of integers, and for natural and integer arithmetic in unary view. The Computing Research Repository (CoRR) abs/1608.06212, August 2016. http://arxiv.org/abs/1608.06212
4. Bergstra, J.A., Ponse, A.: Three datatype defining rewrite systems for datatypes of integers each extending a datatype of naturals (version 3). Report TCS1409v3, University of Amsterdam (UvA), The Netherlands, February 2016. http://arxiv.org/abs/1406.3280
5. Berry, G.: The Esterel V5 Language Primer – Version v5_91. INRIA Sophia-Antipolis, France, June 2000
6. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude – A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71999-1
7. Cohen, D., Watson, P.: An efficient representation of arithmetic for term rewriting. In: Book, R.V. (ed.) RTA 1991. LNCS, vol. 488, pp. 240–251. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-53904-2_100
8. Contejean, E., Marché, C., Rabehasaina, L.: Rewrite systems for natural, integral, and rational arithmetic. In: Comon, H. (ed.) RTA 1997. LNCS, vol. 1232, pp. 98–112. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-62950-5_64
9. Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification 1 - Equations and Initial Semantics. EATCS Monographs on Theoretical Computer Science, vol. 6. Springer, Heidelberg (1985). https://doi.org/10.1007/978-3-642-69962-7
10. Ernst, G., Haneberg, D., Pfähler, J., Reif, W., Schellhorn, G., Stenzel, K., Tofan., B.: A practical course on KIV. Technical report, 152 p., Institute for Software and Systems Engineering, University of Augsburg, Germany (2015)
11. Garavel, H.: Compilation of LOTOS abstract data types. In: Vuong, S.T. (ed.) Proceedings of the 2nd International Conference on Formal Description Techniques, Vancouver B.C., Canada, FORTE 1989, pp. 147–162. North-Holland, December 1989
12. Groote, J., Mousavi, M.: Modeling and Analysis of Communicating Systems. The MIT Press, Cambridge (2014)
13. Houssais, B.: The Synchronous Programming Language SIGNAL – A Tutorial. INRIA Rennes, France, September 2004
14. ISO/IEC: LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, Geneva, September 1989
15. ISO/IEC: Vienna Development Method – Specification Language – Part 1: Base Language. Internation Standard 13817-1:1996, International Organization for Standardization – Programming Languages, their Environments and System Software Interfaces, Geneva, December 1995
16. ISO/IEC: Z Formal Specification Notation – Syntax, Type System and Semantics. International Standard 13568:2002, International Organization for Standardization – Information Technology, Geneva, July 2002

17. Kennaway, J.R.: Complete term rewrite systems for decimal arithmetic and other total recursive functions. In: Proceedings of the 2nd International Workshop on Termination, La Bresse, France, May 1995
18. Lamport, L.: Specifying Systems – The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, Boston (2002)
19. Larsen, P.G., Lausdahl, K., Battle, N.: VDM-10 language manual. Technical report TR-2010-06, Overture – Open-source Tools for Formal Modelling, April 2010
20. Lecomte, T.: B Language Reference Manual (Version 1.8.7). ClearSy System Engineering, Aix-en-Provence, France (2010)
21. Mauw, S., Mulder, J.: A PSF library of data types. In: van den Brand, M.G.J., van Deursen, A., Dinesh, T.B., Kamperman, J., Visser, E. (eds.) Proceedings of ASF+SDF 1995: A Workshop on Generating Tools From Algebraic Specifications, May 1995. http://ivi.fnwi.uva.nl/tcs/pub/reports/1995/P9504/5.html. Technical report P9504-5, Programming Research Group, University of Amsterdam, The Netherlands
22. Mendelson, E.: Number Systems and the Foundations of Analysis. Dover Books on Mathematics Series. Courier Corporation, Chelmsford (1973)
23. Mosses, P.D. (ed.): CASL Reference Manual – The Complete Documentation of the Common Algebraic Specification Language. LNCS, vol. 2960. Springer, Heidelberg (2004). https://doi.org/10.1007/b96103
24. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL – A Proof Assistant for Higher-Order Logic (2016). http://isabelle.in.tum.de/doc/tutorial.pdf (Early Version: Lecture Notes in Computer Science, vol. 2283, Springer, 2002)
25. Owre, S., Shankar, N.: The PVS prelude library. Technical report CSL-03-01, SRI International, Computer Science Laboratory, March 2003
26. Paulin-Mohring, C.: Basics of COQ (Sep 2011), Lecture Notes of the LASER 2011 Summer School
27. Thompson, S.J.: Laws in Miranda. In: Proceedings of the ACM Conference on LISP and Functional Programming (LFP 1986), pp. 1–12 (1986)
28. Isabelle/HOL – Higher-Order Logic. University of Cambridge, Technische Universität München, February 2016. http://isabelle.in.tum.de/website-Isabelle2016/dist/library/HOL/HOL/document.pdf
29. van der Kamp, L.: A Term Rewrite System for Decimal Integer Arithmetic. Bachelor Informatica, University of Amsterdam (UvA), The Netherlands, June 2016
30. van Wamel, J.J.: A Library for PSF. Report PR9301, Programming Research Group, University of Amsterdam, The Netherlands (1993)
31. Verimag: Lustre Language Reference Manual – Versions 3, 3+, 4, 5. Grenoble, France, September 1997
32. Walters, H.R., Zantema, H.: Rewrite systems for integer arithmetic. In: Hsiang, J. (ed.) RTA 1995. LNCS, vol. 914, pp. 324–338. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59200-8_67
33. Walters, H.R.: A complete term rewriting system for decimal integer arithmetic. Technical report CS-R9435, CWI, Amsterdam, The Netherlands (1994)
34. Zantema, H.: Basic Arithmetic by Rewriting and its Complexity, Technical University of Eindhoven, The Netherlands, December 2003. http://www.win.tue.nl/~hzantema/aritm.pdf

Observational Semantics for Dynamic Logic with Binders

Rolf Hennicker[1] and Alexandre Madeira[2,3](✉)

[1] Ludwig-Maximilians-Universität München, München, Germany
[2] QuantaLab and HASLab INESC TEC, University of Minho, Braga, Portugal
[3] CIDMA - Department of Mathematics, University of Aveiro, Aveiro, Portugal
madeira@ua.pt

Abstract. The dynamic logic with binders $\mathcal{D}^{\downarrow}$ was recently introduced as a suitable formalism to support a rigorous stepwise development method for reactive software. The commitment of this logic concerning bisimulation equivalence is, however, not satisfactory: the model class semantics of specifications in $\mathcal{D}^{\downarrow}$ is not closed under bisimulation equivalence; there are $\mathcal{D}^{\downarrow}$-sentences that distinguish bisimulation equivalent models, i.e., $\mathcal{D}^{\downarrow}$ does not enjoy the modal invariance property. This paper improves on these limitations by providing an observational semantics for dynamic logic with binders. This involves the definition of a new model category and of a more relaxed satisfaction relation. We show that the new logic $\mathcal{D}^{\downarrow}_{\sim}$ enjoys modal invariance and even the Hennessy-Milner property. Moreover, the new model category provides a categorical characterisation of bisimulation equivalence by observational isomorphism. Finally, we consider abstractor semantics obtained by closing the model class of a specification SP in $\mathcal{D}^{\downarrow}$ under bisimulation equivalence. We show that, under mild conditions, abstractor semantics of SP in $\mathcal{D}^{\downarrow}$ is the same as observational semantics of SP in $\mathcal{D}^{\downarrow}_{\sim}$.

1 Introduction

The study of logics and formal methods for rigorous development of reactive systems, i.e. systems which interact with their environment during the computation [1], is an active topic of research. Dynamic logic with binders, called $\mathcal{D}^{\downarrow}$-logic, has been introduced in [7] as a logical framework which allows to express properties of reactive systems, from abstract safety and liveness requirements down to concrete specifications of the (recursive) structure of executable processes.

A. Madeira—This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within projects POCI-01-0145-FEDER-016692 and UID/MAT/04106/2013 and also with its Operational Programme NORTE 2020, trough the project NORTE-01-0145-FEDER-000037. The second author is also supported by the individual grant SFRH/BPD/103004/2014.

P. James and M. Roggenbach (Eds.): WADT 2016, LNCS 10644, pp. 135–152, 2017.
https://doi.org/10.1007/978-3-319-72044-9_10

$\mathcal{D}^{\downarrow}$-logic combines in the same formalism modalities indexed by regular expressions of actions, as in Dynamic Logic [6], with binders of Hybrid Logic [4], which bind state variables to particular states and thus allow us to specify concrete processes. We have shown in [7] how the whole development process of reactive systems can be supported by stepwise refinement of $\mathcal{D}^{\downarrow}$-specifications whose models are labelled transition systems with initial state.

However, the satisfaction relation used in $\mathcal{D}^{\downarrow}$ and its notion of isomorphism, the categorical formalisation of identity among objects, are too strict to allow proper behavioural abstraction. As it is well known, bisimulation equivalence is usually adopted to identify behaviourally equivalent systems. However, this is not reflected in the model category of $\mathcal{D}^{\downarrow}$ where model classes are closed under isomorphism but, in general, not under bisimulation equivalence. Thus $\mathcal{D}^{\downarrow}$-logic does not enjoy the *modal invariance property* which requires that bisimilar models satisfy exactly the same logical sentences.

To find a solution, we draw an analogy to algebraic specifications of data types: Equational and first-order logic specifications do generally not support abstraction w.r.t. behaviourally equivalent data structures. This fact led to a significant number of studies proposing different solutions; see Chap. 8 in [10] for a summary. One idea, originally proposed by Reichel [9], was to relax the satisfaction relation of first-order logic such that equations are not necessarily interpreted by the set-theoretic equality but by observational equality of elements; see, e.g., [2,5]. We take up this idea and propose, in Sect. 3, a new logic, called $\mathcal{D}^{\downarrow}_{\sim}$, which has the same sentences and models as $\mathcal{D}^{\downarrow}$ but more relaxed notions of satisfaction and model morphism. The idea of satisfaction in $\mathcal{D}^{\downarrow}_{\sim}$, called *observational satisfaction*, is that state variables x occurring in a formula can be interpreted by arbitrary states as long as they are bisimilar to the state to which x was bound before. This leads to *observational semantics* of a specification SP consisting of all models which observationally satisfy the axioms of SP. Model morphisms in $\mathcal{D}^{\downarrow}_{\sim}$, called *observational morphisms*, capture the idea of simulation. We show that observational satisfaction of positive sentences is preserved by observational morphisms. Moreover, we show that models which are observationally isomorphic satisfy observationally the same sentences, i.e. we get modal invariance of sentences w.r.t. satisfaction and isomorphism in $\mathcal{D}^{\downarrow}_{\sim}$.

In Sect. 4, we study relationships between isomorphism in $\mathcal{D}^{\downarrow}_{\sim}$ and bisimulation equivalence and prove that both concepts are indeed equivalent. Thus, we get (i) a categorical characterisation of bisimulation equivalence and (ii) the modal invariance property w.r.t. observational satisfaction and bisimulation equivalence, which solves our problem discussed above. But the new logic $\mathcal{D}^{\downarrow}_{\sim}$ allows us to go even a step further: We prove a Hennessy-Milner Theorem which shows that two image finite models satisfy in $\mathcal{D}^{\downarrow}_{\sim}$ the same sentences if and only if they are bisimilar - which in turn is equivalent to being isomorphic in $\mathcal{D}^{\downarrow}_{\sim}$.

In Sect. 5, we compare observational semantics of specifications in $\mathcal{D}^{\downarrow}_{\sim}$ with another possibility for behavioural abstraction called *abstractor semantics*. The idea of abstractor semantics goes again back to algebraic specifications where

Sannella and Tarlecki have proposed to abstract from the "standard" model class of a specification by taking its closure under an appropriate equivalence relation; see [10]. For reactive system specifications this means that we consider our original $\mathcal{D}^{\downarrow}$-logic, specifications over $\mathcal{D}^{\downarrow}$ and their model classes (in terms of satisfaction in $\mathcal{D}^{\downarrow}$) but then abstract from a specification's model class by closing it under bisimulation equivalence. We investigate that observational semantics and abstractor semantics of reactive system specifications can be related completely analogously as it has been done for algebraic specifications of data types in [3]. We show that both semantics coincide if and only if any model of a specification SP interpreted in $\mathcal{D}^{\downarrow}$ is also a model when SP is interpreted in $\mathcal{D}^{\downarrow}_{\sim}$.

2 $\mathcal{D}^{\downarrow}$-Logic: Background and Motivations

2.1 Overview on $\mathcal{D}^{\downarrow}$

This section reviews $\mathcal{D}^{\downarrow}$-logic introduced in [7] and proves additionally that satisfaction in $\mathcal{D}^{\downarrow}$ is preserved by isomorphism. $\mathcal{D}^{\downarrow}$-logic is designed to express properties of reactive systems, from abstract safety and liveness properties down to concrete ones specifying the (recursive) structure of processes. It thus combines modalities indexed by regular expressions of actions, as in Dynamic Logic [6], and state variables with binders, as in Hybrid Logic [4]. These motivations are reflected in its semantics. Differently from what is usual in modal logics, whose semantics is given by Kripke structures and satisfaction of formulas is evaluated globally, $\mathcal{D}^{\downarrow}$ models are reachable, labelled transition systems with initial states where satisfaction is evaluated. This reflects our focus on computations, i.e. on effective processes. In modal logic this corresponds to submodels of Kripke structures generated by a given point, which represents the initial state of computations.

Definition 1 (Models and model morphisms). *Let A be a set of atomic actions. An A-model is triple (W, w_0, R) where W is a* set of states, *$w_0 \in W$ is the* initial state *and $R = (R_a \subseteq W \times W)_{a \in A}$ is a family of* transition relations *such that, for each $w \in W$, there is a finite sequence of transitions $R_{a^k}(w^{k-1}, w^k), 1 \leq k \leq n$, with $w^k \in W, a^k \in A$, such that $w^0 = w_0$ and $w^n = w$.*

Given two A-models $\mathcal{M} = (W, w_0, R)$ and $\mathcal{M}' = (W', w'_0, R')$, a model morphism *$h : \mathcal{M} \to \mathcal{M}'$ is a function $h : W \to W'$ such that $h(w_0) = w'_0$ and, for each $a \in A$, if $(w_1, w_2) \in R_a$ then $(h(w_1), h(w_2)) \in R'_a$.*

Lemma 1. *The class of A-models and A-model morphisms define a category denoted by $\mathrm{Mod}^{\mathcal{D}^{\downarrow}}(A)$. The identity morphisms $id_{\mathcal{M}}$ are the identity functions.*

As usual, we say that two models $\mathcal{M}, \mathcal{M}' \in \mathrm{Mod}^{\mathcal{D}^{\downarrow}}(A)$ are *isomorphic*, in symbols $\mathcal{M}$ **iso** $\mathcal{M}'$, if there is a pair of morphisms $h : \mathcal{M} \to \mathcal{M}'$ and $h^{-1} : \mathcal{M}' \to \mathcal{M}$ such that $h \cdot h^{-1} = id_{\mathcal{M}}$ and $h^{-1} \cdot h = id_{\mathcal{M}'}$.

The *set of (composed) actions*, $\mathrm{Act}(A)$, induced by a set of atomic actions A is given by

$$\alpha ::= a \mid \alpha;\alpha \mid \alpha+\alpha \mid \alpha^*$$

where $a \in A$. In the context of a finite set of atomic actions $A = \{a_1, \ldots, a_n\}$, we may briefly write A for the complex action $a_1 + \ldots + a_n$. For a set X of variables and an A-model $\mathcal{M} = (W, w_0, R)$, a *valuation* is a function $g : X \to W$. Given such a valuation g, a variable $x \in X$ and a state $w \in W, g[x \mapsto w]$ denotes the valuation with $g[x \mapsto w](x) = w$ and $g[x \mapsto w](y) = g(y)$ for any $y \in X, y \neq x$.

Definition 2 (Formulas and sentences). *The set of A-*formulas *is given by*

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid x \mid \ \downarrow x.\, \varphi \mid @_x\varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

*where $x \in X$ and $\alpha \in \mathrm{Act}(A)$. An A-formula φ is called A-*sentence *if φ contains no free variables. Free variables are defined as usual with $\downarrow$, the only operator binding variables.*

The binder operator $\downarrow x.\varphi$ assigns to variable x the current state of evaluation and evaluates φ. The operator $@_x\varphi$ evaluates φ in the state assigned to x. To define the satisfaction relation formally we need to clarify how composed actions are interpreted in models. Let $\alpha \in \mathrm{Act}(A)$ and $\mathcal{M} \in \mathrm{Mod}^{\mathcal{D}^{\downarrow}}(A)$. The interpretation of α in $\mathcal{M}$ extends the interpretation of atomic actions by $R_{\alpha;\alpha'} = R_\alpha \cdot R_{\alpha'}, R_{\alpha+\alpha'} = R_\alpha \cup R_{\alpha'}$ and $R_{\alpha^*} = (R_\alpha)^\star$, with the operations $\cdot, \cup$ and $\star$ standing for relational composition, union and reflexive-transitive closure. Given an A-model $\mathcal{M} = (W, w_0, R), w \in W$ and $g : X \to W$,

- $\mathcal{M}, g, w \models \mathbf{tt}$ is true; $\mathcal{M}, s \models \mathbf{ff}$ is false;
- $\mathcal{M}, g, w \models x$ iff $g(x) = w$;
- $\mathcal{M}, g, w \models \downarrow x.\, \varphi$ iff $\mathcal{M}, g[x \mapsto w], w \models \varphi$;
- $\mathcal{M}, g, w \models @_x\varphi$ iff $\mathcal{M}, g, g(x) \models \varphi$;
- $\mathcal{M}, g, w \models \langle\alpha\rangle\varphi$ iff there is a $v \in W$ with $(w, v) \in R_\alpha$ and $\mathcal{M}, g, v \models \varphi$;
- $\mathcal{M}, g, w \models [\alpha]\varphi$ iff for any $v \in W$ with $(w, v) \in R_\alpha$ it holds $\mathcal{M}, g, v \models \varphi$;
- $\mathcal{M}, g, w \models \neg\varphi$ iff it is false that $\mathcal{M}, g, w \models \varphi$;
- $\mathcal{M}, g, w \models \varphi \wedge \varphi'$ iff $\mathcal{M}, g, w \models \varphi$ and $\mathcal{M}, g, w \models \varphi'$;
- $\mathcal{M}, g, w \models \varphi \vee \varphi'$ iff $\mathcal{M}, g, w \models \varphi$ or $\mathcal{M}, g, w \models \varphi'$.

We write $\mathcal{M}, w \models \varphi$ if, for any valuation $g : X \to W$, we have $\mathcal{M}, g, w \models \varphi$. If φ is an A-sentence, then the valuation is irrelevant, i.e., $\mathcal{M}, g, w \models \varphi$ iff $\mathcal{M}, w \models \varphi$. $\mathcal{M}$ *satisfies* an A-sentence φ, written $\mathcal{M} \models \varphi$, if $\mathcal{M}, w_0 \models \varphi$.

Hence, $\mathcal{D}^{\downarrow}$-logic expresses properties of states reachable from the initial one. For instance, if A is finite, $\mathcal{D}^{\downarrow}$ is able to express liveness requirements such as "*after the occurrence of an action a, an action b can be eventually realised*" with $[A^*; a]\langle A^*; b\rangle\mathbf{tt}$, safety properties by sentences of the form $[A^*]\varphi$, in particular, deadlock freeness by $[A^*]\langle A\rangle\mathbf{tt}$. $\mathcal{D}^{\downarrow}$-logic is also suited to express process structures and, thus, the implementation of abstract requirements. The binder operator is crucial for this. The ability to give names to visited states together with the modal features allows to express recursive process patterns. For instance, the following sentence captures processes with two states and alternating a and b transitions.

$$\downarrow x_0.\big(\langle a\rangle \downarrow x_1.(\langle b\rangle x_0)\big)$$

Definition 3 (Specification). *A* specification *SP is a pair $SP = (A, \Phi)$ where A is a set of atomic actions and Φ is a set of A-sentences.*

Definition 4 (Semantics). *The* semantics *of a specification $SP = (A, \Phi)$ in $\mathcal{D}^{\downarrow}$ is given by the class of models*

$$Mod(SP) = \{\mathcal{M} \in \mathrm{Mod}^{\mathcal{D}^{\downarrow}}(A) \mid \mathcal{M} \models \varphi \textit{ for all } \varphi \in \Phi\}.$$

Lemma 2. *Let $\mathcal{M} = (W, w_0, R)$ and $\mathcal{M}' = (W', w_0', R')$ be two A-models and $h : \mathcal{M} \to \mathcal{M}'$ an isomorphism. Then for any $w \in W$, valuation $g : X \to W$ and A-formula φ, we have*

$$\mathcal{M}, g, w \models \varphi \textit{ iff } \mathcal{M}', g \circ h, h(w) \models \varphi.$$

Proof. The proof is performed by induction on the structure of A-formulas. The base cases $\varphi = \mathbf{tt}$ and $\varphi = \mathbf{ff}$ are trivial.
Case $\varphi = x$**:**

$$\begin{array}{ll|ll}
 & \mathcal{M}, g, w \models x & & h(g(x)) = h(w) \\
\Leftrightarrow & \quad \{ \models \text{defn}\} & \Leftrightarrow & \quad \{ \circ \text{ composition}\} \\
 & g(x) = w & & (g \cdot h)(x) = h(w) \\
\Leftrightarrow & \quad \{ h \text{ injective}\} & \Leftrightarrow & \quad \{ \models \text{defn}\} \\
 & & & \mathcal{M}', g \circ h, h(w) \models x
\end{array}$$

Case $\varphi = \downarrow x.\, \phi$**:**

$$\begin{array}{ll|ll}
 & \mathcal{M}, g, w \models \downarrow x.\phi & & \mathcal{M}', g[x \mapsto w] \circ h, h(w) \models \phi \\
\Leftrightarrow & \quad \{ \models \text{defn}\} & \Leftrightarrow & \quad \{ \text{ since } g[x \mapsto w] \circ h = (g \circ h)[x \mapsto h(w)]\} \\
 & \mathcal{M}, g[x \mapsto w], w \models \phi & & \mathcal{M}', (g \circ h)[x \mapsto h(w)], h(w) \models \phi \\
\Leftrightarrow & \quad \{ \text{ I.H. } \} & \Leftrightarrow & \quad \{ \models \text{defn}\} \\
 & & & \mathcal{M}', g \circ h, h(w) \models \downarrow x.\phi
\end{array}$$

Case $\varphi = \langle \alpha \rangle \phi$**:**

$$\begin{array}{ll|ll}
 & \mathcal{M}, g, w \models \langle\alpha\rangle\phi & & \mathcal{M}', g \circ h, h(v) \models \phi \\
\Leftrightarrow & \quad \{ \models \text{defn}\} & & \text{ for some } v \in W, (h(w), h(v)) \in R'_\alpha \\
 & \mathcal{M}, g, v \models \phi \text{ for some } v \in W, (w, v) \in R_\alpha & \Leftrightarrow & \quad \{ \models \text{defn} + h \text{ surjective}\} \\
\Leftrightarrow & \quad \{ \text{ I.H.} + h \text{ iso} + \star\} & & \mathcal{M}', g \circ h, h(w) \models \langle\alpha\rangle\phi
\end{array}$$

$\star$: We use the fact, that morphisms also satisfy $(w_1, w_2) \in R_\alpha$ then $(h(w_1), h(w_2)) \in R'_\alpha$ for composed actions $\alpha \in \mathrm{Act}(A)$.
The proof for the remaining cases is straightforward. □

Theorem 1. *Let $\mathcal{M}$ and $\mathcal{M}'$ be A-models such that $\mathcal{M}$* **iso** *$\mathcal{M}'$. Then, for any A-sentence φ, we have*

$$\mathcal{M} \models \varphi \textit{ iff } \mathcal{M}' \models \varphi.$$

Proof. Since φ has no free variables, it follows from Lemma 2, that for any $w \in W$, we have $\mathcal{M}, w \models \varphi$ iff $\mathcal{M}', h(w) \models \varphi$ where h is an isomorphism between $\mathcal{M}$ and $\mathcal{M}'$. In particular, since $h(w_0) = w_0'$, we have $\mathcal{M}, w_0 \models \varphi$ iff $\mathcal{M}', w_0' \models \varphi$, i.e., $\mathcal{M} \models \varphi$ iff $\mathcal{M}' \models \varphi$. □

Corollary 1. *For any specification SP, $Mod(SP)$ is closed under* **iso** .

2.2 Motivations

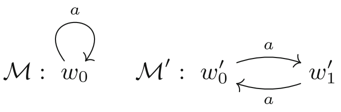
Let us recall the well-known notion of bisimulation between transition systems:

Definition 5 (Bisimulation). *Let $\mathcal{M} = (W, w_0, R)$ and $\mathcal{M}' = (W', w_0', R')$ be two A-models. A* bisimulation *between $\mathcal{M}$ and $\mathcal{M}'$ is a relation $S \subseteq W \times W'$ that contains (w_0, w_0') and satisfies*

(zig) *for any $a \in A, w, v \in W, w' \in W'$ such that $(w, w') \in S$:*
if $(w, v) \in R_a$, then there is a $v' \in W'$ such that $(w', v') \in R_a'$ and $(v, v') \in S$;
(zag) *for any $a \in A, w \in W, w', v' \in W'$ such that $(w, w') \in S$:*
if $(w', v') \in R_a'$, then there is a $v \in W$ such that $(w, v) \in R_a$ and $(v, v') \in S$.

Two A-models $\mathcal{M}$ and $\mathcal{M}'$ are called *bisimulation equivalent*, denoted by $\mathcal{M} \equiv \mathcal{M}'$, if there exists a bisimulation between $\mathcal{M}$ and $\mathcal{M}'$. It is well known that bisimulation equivalence is indeed an equivalence relation on the class of A-models. Moreover, if $\mathcal{M} \equiv \mathcal{M}'$, then there exists a greatest bisimulation between $\mathcal{M}$ and $\mathcal{M}'$, which we denote by $\sim_{\mathcal{M}'}^{\mathcal{M}}$.

Bisimulation equivalence plays a central role in the analysis and development of reactive systems. It can be taken as the standard behavioural equivalence between processes in the sense that, given two bisimulation equivalent processes, it should be irrelevant for the correctness of an implementation which one is chosen to realise a given system specification. The notion of bisimulation equivalence plays also an important role in the theory of modal logics: the satisfaction in most of modal logics is invariant w.r.t. bisimulation equivalence, i.e. bisimulation equivalent models satisfy the same sentences. However, this is not the case for the logic $\mathcal{D}^{\downarrow}$. In order to see that, let us consider the two $\{a\}$-models $\mathcal{M}$ and $\mathcal{M}'$ presented in Fig. 1 and the specification $SP = (\{a\}, \{\downarrow x.\langle a\rangle x\})$. It is easy to see that $\mathcal{M} \in Mod(SP)$ and $\mathcal{M}' \notin Mod(SP)$. However, $\mathcal{M} \equiv \mathcal{M}'$



$$\begin{tikzcd} \mathcal{M} : w_0 \arrow[loop above, "a"] & \mathcal{M}' : w_0' \arrow[r, bend left, "a"] & w_1' \arrow[l, bend left, "a"] \end{tikzcd}$$

Fig. 1. Bisimilar models

which shows that $\mathcal{D}^{\downarrow}$ does not obey the implementation principle from above. From the logic, point of view it illustrates that $\mathcal{D}^{\downarrow}$ does not enjoy of the modal invariance property.

3 $\mathcal{D}^{\downarrow}_{\sim}$-Logic

In this section we introduce a new logic, called $\mathcal{D}^{\downarrow}_{\sim}$, which generalises $\mathcal{D}^{\downarrow}$-logic by supporting abstraction w.r.t. observationally indistinguishable states. The formulas and sentences of $\mathcal{D}^{\downarrow}_{\sim}$ are the same as in $\mathcal{D}^{\downarrow}$. The essential difference lies in the definition of model morphisms and in a relaxation of the satisfaction relation which is adjusted to the observational paradigm. As a central result we will show that in the new category $\mathcal{D}^{\downarrow}_{\sim}$ observationally isomorphic models satisfy observationally the same sentences; i.e. we get modal invariance w.r.t. observational isomorphism and the relaxed (observational) satisfaction relation.

3.1 Observational Models Category

We introduce a new category of models for a set A of atomic actions. The objects of this category are, as in $\mathcal{D}^{\downarrow}$, reachable (labelled) transition systems with initial states. However, we introduce a new kind of model morphism, called observational morphism. Such morphisms are not functions but relations which abstract away the difference between states with an observationally equal behaviour. For this purpose, we consider for any A-model $\mathcal{M} = (W, w_0, R)$ the *observational equality* relation $\sim_{\mathcal{M}} \subseteq W \times W$, which is defined as the greatest bisimulation $\sim^{\mathcal{M}}_{\mathcal{M}}$ between $\mathcal{M}$ and $\mathcal{M}$[1]. Then an observational morphism $h : \mathcal{M} \to \mathcal{M}'$ is a relation between the state spaces of two A-models $\mathcal{M}$ and $\mathcal{M}'$ containing their initial states which has the following properties: (1) h is a simulation relation such that any transition in $\mathcal{M}$ is simulated by a transition in $\mathcal{M}'$ with the same label (i.e. observational morphisms satisfy the "zig" condition of a bisimulation), (2) h preserves observational equality of states from $\mathcal{M}$ to $\mathcal{M}'$ and (3) h is closed under the observational equalities $\sim_{\mathcal{M}}$ and $\sim'_{\mathcal{M}}$ of $\mathcal{M}$ and $\mathcal{M}'$ resp. These properties are expressed by the three conditions in the subsequent definition. We note that observational morphisms could be equivalently defined by morphisms between the quotient structures of $\mathcal{M}$ and $\mathcal{M}'$ considered later on in Definition 10. We prefer, however, to give a direct definition on the state spaces of $\mathcal{M}$ and $\mathcal{M}'$ since those models are actually the representations of concrete implementations and not their quotient structures.

Definition 6 (Observational morphisms). *Let $\mathcal{M} = (W, w_0, R)$ and $\mathcal{M}' = (W', w'_0, R')$ be two A-models. An* observational morphism *$h : \mathcal{M} \to \mathcal{M}'$ is a relation $h \subseteq W \times W'$ containing (w_0, w'_0) such that the following conditions are satisfied:*

[1] It exists since bisimulation equivalence is reflexive and it is an equivalence relation on the states of $\mathcal{M}$.

1. *For any $a \in A, w, v \in W, w' \in W'$ such that $(w, w') \in h$:*
   if $(w, v) \in R_a$, then there is a $v' \in W'$ such that $(w', v') \in R'_a$ and $(v, v') \in h$.

$$\begin{tikzcd} w \arrow[r, dash, "R_a"] \arrow[d, "h"'] & v \\ w' & \end{tikzcd} \quad \Rightarrow \exists v' \in W : \begin{tikzcd} w \arrow[r, dash, "R_a"] \arrow[d, "h"'] & v \arrow[d, "h"] \\ w' \arrow[r, dash, "R'_a"'] & v' \end{tikzcd}$$

2. *For any $w, v \in W, w', v' \in W'$ such that $(w, w') \in h$ and $(v, v') \in h$:*
   if $w \sim_{\mathcal{M}} v$, then $w' \sim_{\mathcal{M}'} v'$.

$$\begin{tikzcd} w \arrow[r, dash, "\sim_{\mathcal{M}}"] \arrow[d, "h"'] & v \arrow[d, "h"] \\ w' & v' \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} w \arrow[r, dash, "\sim_{\mathcal{M}}"] \arrow[d, "h"'] & v \arrow[d, "h"] \\ w' \arrow[r, dash, "\sim_{\mathcal{M}'}"'] & v' \end{tikzcd}$$

3. *For any $w, v \in W, w', v' \in W'$ such that $(w, w') \in h$:*
   if $w \sim_{\mathcal{M}} v$ and $w' \sim_{\mathcal{M}} v'$, then $(v, v') \in h$.

$$\begin{tikzcd} w \arrow[r, dash, "\sim_{\mathcal{M}}"] \arrow[d, "h"'] & v \\ w' \arrow[r, dash, "\sim_{\mathcal{M}'}"'] & v' \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} w \arrow[r, dash, "\sim_{\mathcal{M}}"] \arrow[d, "h"'] & v \arrow[d, "h"] \\ w' \arrow[r, dash, "\sim_{\mathcal{M}'}"'] & v' \end{tikzcd}$$

By the definition of composed actions and their interpretation as relations the simulation condition 1 of Definition 6 can be lifted to composed actions:

Remark 1. Condition 1 of Definition 6 implies that for any $\alpha \in \mathrm{Act}(A)$ and any $w, v \in W, w' \in W'$ such that $(w, w') \in h$:
if $(w, v) \in R_\alpha$, then there is a $v' \in W'$ such that $(w', v') \in R'_\alpha$ and $(v, v') \in h$.

Lemma 3. *Observational morphisms are total relations.*

Proof. This is a direct consequence of the reachability of states. On the one hand, we have $(w_0, w'_0) \in h$. The induction step corresponds to 1 of Definition 6. □

Theorem 2. *The class of A-models together with observational morphisms form a category, denoted by $\mathrm{Mod}^{\mathcal{D}^{\downarrow}_{\sim}}(A)$. For each $\mathcal{M} \in \mathrm{Mod}^{\mathcal{D}^{\downarrow}_{\sim}}(A)$, the identity morphism $1_{\mathcal{M}}$ is the observational equality $\sim_{\mathcal{M}}$.*

Proof. Observational morphisms are closed under composition of relations: Given two observational morphisms $h : \mathcal{M} \to \mathcal{M}'$ and $h' : \mathcal{M}' \to \mathcal{M}''$, their composition $h \cdot h' : \mathcal{M} \to \mathcal{M}''$ is the relation $\{(w, w'')|$ there exists w' s.t. $(w, w') \in h$ and $(w', w'') \in h'\}$. It is straightforward to show, by standard set-theoretic reasoning, that $h \cdot h'$ satisfies the conditions 1–3 of Definition 6 since h and h' do so. Also it is clear that relational composition is associative.

For each A-model $\mathcal{M}, 1_{\mathcal{M}} = \sim_{\mathcal{M}}$ is an observational morphism $\mathcal{M} \to \mathcal{M}$: Since $\sim_{\mathcal{M}}$ is a bisimulation it satisfies 1 of Definition 6. Since $\sim_{\mathcal{M}}$ is the greatest bisimulation on $\mathcal{M}$ it is closed under composition and therefore, taking into account that $\sim_{\mathcal{M}}$ is an equivalence relation, it satisfies 2 and 3. Finally, because of the closure property 3 of Definition 6, it is obvious that, for any observational morphism $\mathcal{M} \xrightarrow{h} \mathcal{M}'$, we have $1_{\mathcal{M}} \cdot h = h$ and $h \cdot 1_{\mathcal{M}'} = h$. □

For A-models $\mathcal{M}$ and $\mathcal{M}'$ we write $\mathcal{M}$ $\mathbf{iso}_{\sim}$ $\mathcal{M}'$ whenever $\mathcal{M}$ and $\mathcal{M}'$ are observationally isomorphic in the category $\mathrm{Mod}^{\mathcal{D}^{\downarrow}_{\sim}}(A)$. The next lemma states a useful property which shows that the inverse of an observational isomorphism $h : \mathcal{M} \to \mathcal{M}'$ in the category $\mathrm{Mod}^{\mathcal{D}^{\downarrow}_{\sim}}(A)$ is just the inverse relation of h.

Lemma 4. *Let $\mathcal{M} = (W, w_0, R)$ and $\mathcal{M}' = (W', w_0', R')$ be two A-models and $h : \mathcal{M} \to \mathcal{M}'$ an observational isomorphism with inverse $h^{-1} : \mathcal{M}' \to \mathcal{M}$. Then for all $w \in W$ and $w' \in W'$ the following holds: $(w, w') \in h$ if and only if $(w', w) \in h^{-1}$.*

Proof. For the proof we use Lemma 3 and condition 3 of Definition 6. Assume $(w, w') \in h$. Since $h^{-1} : \mathcal{M}' \to \mathcal{M}$ is an observational morphism it is total, by Lemma 3. Hence, there exists $v \in W$ such that $(w', v) \in h^{-1}$. By the isomorphism property we have $h \cdot h^{-1} = 1_{\mathcal{M}}$. Since $(w, v) \in h \cdot h^{-1}$, we get $(w, v) \in 1_{\mathcal{M}}$, i.e. $w \sim_{\mathcal{M}} v$. Since $h^{-1} : \mathcal{M}' \to \mathcal{M}$ satisfies 3 of Definition 6, $(w', v) \in h^{-1}$ and $v \sim_{\mathcal{M}} w$ implies $(w', w) \in h^{-1}$. The converse direction is proved analogously by using again condition 3 of Definition 6. □

As a consequence of Lemma 4, we can show that observational isomorphisms satisfy the "zag" condition of a bisimulation.

Lemma 5. *Let $\mathcal{M} = (W, w_0, R)$ and $\mathcal{M}' = (W', w_0', R')$ be two A-models and $h : \mathcal{M} \to \mathcal{M}'$ an observational isomorphism. Then the following holds:*
For any $a \in A, w \in W, w', v' \in W'$ such that $(w, w') \in h$:
if $(w', v') \in R'_a$, then there is a $v \in W$ such that $(w, v) \in R_a$ and $(v, v') \in h$.

Proof. Assume $(w, w') \in h$ and $(w', v') \in R'_a$. Let h^{-1} be the inverse of h. Then, by Lemma 4, $(w', w) \in h^{-1}$. Since h^{-1} satisfies condition 1 of Definition 6, there is a $v \in W$ such that $(w, v) \in R_a$ and $(v', v) \in h^{-1}$. By Lemma 4, $(v, v') \in h$ and we are done. □

As an example, consider the two $\{a\}$-models $\mathcal{M}$ and $\mathcal{M}'$ in Fig. 1. The relation $h = \{(w_0, w_0'), (w_0, w_1')\}$ is an observational isomorphism between $\mathcal{M}$ and $\mathcal{M}'$. We have also seen in Sect. 2.2 that $\mathcal{M}$ and $\mathcal{M}'$ are bisimilar. In fact, we will show later, in Sect. 4, that observational isomorphism coincides with bisimulation equivalence.

3.2 Observational Satisfaction

We are now ready to generalise the satisfaction relation of $\mathcal{D}^{\downarrow}$-logic to take into account observational abstraction. We use the same formulas as in $\mathcal{D}^{\downarrow}$, which were called A-formulas for a given set A of atomic actions. But now, in the logic $\mathcal{D}^{\downarrow}_{\sim}$, the observational satisfaction of an A-formula allows to interpret variables x by states which are not identical but only observationally equal to the current valuation of x.

Definition 7 (Observational satisfaction). *Let $\mathcal{M} = (W, w_0, R)$ be an A-model, $w \in W$ and $g : X \to W$ a valuation. The* observational satisfaction *of an A-formula φ in state w of $\mathcal{M}$ w.r.t. valuation g, denoted by $\mathcal{M}, g, w \models_{\sim} \varphi$, is defined analogously to the satisfaction as shown in Sect. 2.1, with the exception of*

$$\mathcal{M}, g, w \models_{\sim} x \text{ iff } g(x) \sim_{\mathcal{M}} w.$$

For each A-sentence φ, the valuation is irrelevant and $\mathcal{M}$ satisfies observationally *φ, denoted by $\mathcal{M} \models_{\sim} \varphi$, if $\mathcal{M}, w_0 \models_{\sim} \varphi$.*

As an example, we consider the $\{a\}$-model $\mathcal{M}'$ in Fig. 1 for which we have: $\mathcal{M}' \models_{\sim} \downarrow x.\langle a \rangle x$. This is true since the a-transition reaches state w_1' which is observationally equal to state w_0'.

Using the observational satisfaction relation we can equip specifications, as defined in Definition 3, with an observational semantics.

Definition 8 (Observational semantics). *The* observational semantics *of a specification $SP = (A, \Phi)$ is given by the class of models*

$$Mod_{\sim}(SP) = \{\mathcal{M} \in \mathrm{Mod}^{\mathcal{D}^{\downarrow}_{\sim}}(A) \mid \mathcal{M} \models_{\sim} \varphi \textit{ for all } \varphi \in \Phi\}.$$

In the following we want to analyse relationships between observational satisfaction and observational morphisms. First, we show that observational satisfaction of positive A-sentences is preserved by observational morphisms; see Theorem 3. Then we show that observational satisfaction of arbitrary A-sentences is preserved and reflected in the case of observational isomorphisms; see Theorem 4.

Definition 9 (Positive formulas and sentences). *An A-formula (A-sentence) φ is a* positive A-formula (A-sentence)*, if it does not contain negation $\neg$ and the box operator* $[.]$.

Lemma 6. *Let $\mathcal{M} = (W, w_0, R)$ and $\mathcal{M}' = (W', w_0', R')$ be two A-models and $h : \mathcal{M} \to \mathcal{M}'$ an observational morphism. Then for any $w \in W, w' \in W'$ with $(w, w') \in h$, for any valuations $g : X \to W, g' : X \to W'$ with $(g(x), g'(x)) \in h$ for all $x \in X$, and for any positive A-formula φ, we have*

$$\mathcal{M}, g, w \models_{\sim} \varphi \textit{ implies } \mathcal{M}', g', w' \models_{\sim} \varphi.$$

Proof. The proof is performed by induction on the structure of positive A-formulas.

The base cases $\varphi = \mathbf{tt}$ and $\varphi = \mathbf{ff}$ are trivial.

Case $\varphi = x$:

$$\begin{array}{cl|cl}
 & \mathcal{M}, g, w \models_{\sim} x & & \\
\Leftrightarrow & \{ \models_{\sim} \text{defn} \} & & g'(x) \sim_{\mathcal{M}'} w' \\
 & g(x) \sim_{\mathcal{M}} w & \Leftrightarrow & \{ \models_{\sim} \text{defn} \} \\
\Rightarrow & \{ \text{ step } \star \ \} & & \mathcal{M}', g', w' \models_{\sim} x
\end{array}$$

Step $\star$ follows from condition 2 of Definition 6 and the assumptions $(g(x), g'(x)) \in h$ and $(w, w') \in h$.

Case $\varphi = \downarrow x.\, \phi$:

$$\begin{array}{cl|cl}
 & \mathcal{M}, g, w \models_{\sim} \downarrow x.\, \phi & & \\
\Leftrightarrow & \{ \models_{\sim} \text{defn} \} & & \mathcal{M}', g'[x \mapsto w'], w' \models_{\sim} \phi \\
 & \mathcal{M}, g[x \mapsto w], w \models_{\sim} \phi & \Leftrightarrow & \{ \models_{\sim} \text{defn} \} \\
\Rightarrow & \{ \text{ step } \star\star \ \} & & \mathcal{M}', g', w' \models_{\sim} \downarrow x.\, \phi
\end{array}$$

Step $\star\star$ follows from the Induction Hypothesis, since $(g(y), g'(y)) \in h$ for all $y \in X$ and $(w, w') \in h$ implies $(g[x \mapsto w](y), g'[x \mapsto w'](y)) \in h$ for all $y \in X$.

Case $\varphi = @_x \phi$:

$$\begin{array}{cl|cl}
 & \mathcal{M}, g, w \models_{\sim} @_x \phi & & \\
\Leftrightarrow & \{ \models_{\sim} \text{defn} \} & & \mathcal{M}', g', g'(x) \models_{\sim} \phi \\
 & \mathcal{M}, g, g(x) \models_{\sim} \phi & \Leftrightarrow & \{ \models_{\sim} \text{defn} \} \\
\Rightarrow & \{ \text{ by I.H. since } (g(x), g'(x)) \in h \ \} & & \mathcal{M}', g', w' \models_{\sim} @_x \phi
\end{array}$$

Case $\varphi = \langle \alpha \rangle \phi$:

$$\begin{array}{cl}
 & \mathcal{M}, g, w \models_{\sim} \langle \alpha \rangle \phi \\
\Leftrightarrow & \{ \models_{\sim} \text{defn} \} \\
 & \mathcal{M}, g, v \models_{\sim} \phi \text{ for some } v \in W \text{ with } (w, v) \in R_\alpha \\
\Rightarrow & \{ \text{ Remark 1 + I.H. } \} \\
 & \mathcal{M}', g', v' \models_{\sim} \phi \text{ for some } v' \in W' \text{ with } (w', v') \in R'_\alpha \\
\Leftrightarrow & \{ \models_{\sim} \text{defn} \} \\
 & \mathcal{M}', g', w' \models_{\sim} \langle \alpha \rangle \phi
\end{array}$$

The cases $\varphi = \phi \wedge \phi'$ and $\varphi = \phi \vee \phi'$ are straightforward by Induction Hypothesis. $\square$

Theorem 3. *Let $\mathcal{M}$ and $\mathcal{M}'$ be two A-models and $h : \mathcal{M} \to \mathcal{M}'$ an observational morphism. Then, for any positive A-sentence φ, we have*

$$\mathcal{M} \models_{\sim} \varphi \textit{ implies } \mathcal{M}' \models_{\sim} \varphi.$$

Proof. Since φ is a sentence, it follows from Lemma 6, that for any $w \in W, w' \in W'$ with $(w, w') \in h$, we have: $\mathcal{M}, w \models_{\sim} \varphi$ implies $\mathcal{M}', w' \models_{\sim} \varphi$. In particular, since $(w_0, w_0') \in h$, $\mathcal{M}, w_0 \models_{\sim} \varphi$ implies $\mathcal{M}', w_0' \models_{\sim} \varphi$, i.e., $\mathcal{M} \models_{\sim} \varphi$ *implies* $\mathcal{M}' \models_{\sim} \varphi$. □

Let us now consider the case in which h is an observational isomorphism.

Lemma 7. *Let $\mathcal{M} = (W, w_0, R)$ and $\mathcal{M}' = (W', w_0', R')$ be two A-models and $h : \mathcal{M} \to \mathcal{M}'$ an observational isomorphism. Then for any $w \in W, w' \in W'$ with $(w, w') \in h$, for any valuations $g : X \to W, g' : X \to W'$ with $(g(x), g'(x)) \in h$ for all $x \in X$, and for any A-formula φ, we have*

$$\mathcal{M}, g, w \models_{\sim} \varphi \textit{ iff } \mathcal{M}', g', w' \models_{\sim} \varphi.$$

Proof. The proof is performed by induction on the structure of the formulas. The base case $\varphi = \mathbf{tt}$ is trivial and for $\varphi = \mathbf{ff}$ we note that neither $\mathcal{M}, g, w \models_{\sim} \mathbf{ff}$ nor $\mathcal{M}, g, w \models_{\sim} \mathbf{ff}$ holds.

Case $\varphi = x$: The proof is performed as for Lemma 6 with the addition that the "$\Rightarrow$" step (step $\star$) holds also in the opposite direction for the following reason: Let h^{-1} be the inverse of h. Since $(g(x), g'(x)) \in h$ and $(w, w') \in h$ we obtain, by Lemma 4, that $(g'(x), g(x)) \in h^{-1}$ and $(w', w) \in h^{-1}$. Now we can apply condition 2 of Definition 6 for h^{-1} such that $g'(x) \sim_{\mathcal{M}'} w'$ implies $g(x) \sim_{\mathcal{M}} w$.

Cases $\varphi = \downarrow x.\, \phi$ and $\varphi = @_x\phi$: The proof is performed as for Lemma 6 with the addition that the "$\Rightarrow$" steps hold also in the opposite direction since now the Induction Hypothesis holds also in the other direction.

Case $\varphi = \langle\alpha\rangle\phi$: The proof is performed as for Lemma 6 with the addition that the "$\Rightarrow$" step holds also in the opposite direction. To see this, we know by Lemma 5 that the "zag" condition of a bisimulation holds for h and for atomic actions $a \in A$. It is straightforward to prove that then the "zag" condition holds also for structured actions $\alpha \in \mathrm{Act}(A)$. Taking into account the I.H. we are done.

The cases $\varphi = \neg\phi, \varphi = \phi \wedge \phi'$ and $\varphi = \phi \vee \phi'$ are straightforward by Induction Hypothesis. The case $\varphi = [\alpha]\phi$ can be shown either by using the I.H. or by taking into account that the box operator can be expressed by negation and diamond. □

Theorem 4. *Let $\mathcal{M}, \mathcal{M}'$ be two A-models such that $\mathcal{M}$ $\mathbf{iso}_{\sim}$ $\mathcal{M}'$. Then, for any A-sentence φ, we have*

$$\mathcal{M} \models_{\sim} \varphi \textit{ iff } \mathcal{M}' \models_{\sim} \varphi.$$

Proof. The proof is completely analogous to the proof of Theorem 3, using Lemma 7 instead of Lemma 6. □

Corollary 2. *For any specification SP, its observational semantics $Mod_{\sim}(SP)$ is closed under* $\mathbf{iso}_{\sim}$.

The next theorem establishes a connection between the observational satisfaction in $\mathcal{D}^{\downarrow}_{\sim}$ and the satisfaction in $\mathcal{D}^{\downarrow}$. It relies on the construction of the quotient $\mathcal{M}/{\sim}$ of an A-model $\mathcal{M}$ that identifies observationally equal (i.e. bisimilar) states.

Definition 10. *Let $\mathcal{M} = (W, w_0, R)$ be an A-model. The* quotient of $\mathcal{M}$ *w.r.t. $\sim_{\mathcal{M}}$ is the A-model $\mathcal{M}/{\sim} = (W/{\sim}, [w_0], R/{\sim})$, where*

- *$W/{\sim} = \{[w] \mid w \in W\}$ with $[w] = \{w' \mid w \sim_{\mathcal{M}} w'\}$, and for all $a \in A$,*
- *$(R/{\sim})_a = \{([w],[v]) \mid$ there exist $w' \in [w]$ and $v' \in [v]$ s.t. $(w,v) \in R_a\}$.*

Remark 2. For any $a \in A$ and $w, v \in W$, if $([w],[v]) \in (R/{\sim})_a$ then there exists $\hat{v} \in [v]$ such that $(w, \hat{v}) \in R_a$. This follows from the (zig) property of $\sim_{\mathcal{M}}$. This fact can be generalised to composed actions $\alpha \in \mathrm{Act}(A)$.

Sentences are observationally satisfied by an A-model $\mathcal{M}$, if and only if they are satisfied by its quotient $\mathcal{M}/{\sim}$:

Theorem 5. *For any A-model $\mathcal{M}$ and for any A-sentence φ,*

$$\mathcal{M} \models_{\sim} \varphi \textit{ iff } \mathcal{M}/{\sim} \models \varphi.$$

Proof. For the proof we show, more generally, that for any $w \in W$, valuation $g : X \to W$ and A-formula φ,

$$\mathcal{M}, g, w \models_{\sim} \varphi \text{ iff } \mathcal{M}/{\sim}, g/{\sim}, [w] \models \varphi$$

where $g/{\sim}\colon X \to W$ is defined by $(g/{\sim})(x) = [g(x)]$. The proof can be performed by induction over the structure of A-formulas. For the base formulas $\varphi = x$, we have:

$$\begin{array}{ll|ll}
 & \mathcal{M}, g, w \models_{\sim} x & & [g(x)] = [w] \\
\Leftrightarrow & \quad \{ \models_{\sim} \text{defn}\} & \Leftrightarrow & \quad \{ [g(x)] = (g/{\sim})(x) + \models \text{defn}\} \\
 & g(x) \sim_{\mathcal{M}} w & & \mathcal{M}/{\sim}, g/{\sim}, [w] \models x \\
\Leftrightarrow & \quad \{ \text{ equivalence classes defn}\} & &
\end{array}$$

For the case $\varphi = \langle\alpha\rangle\phi$, we have:

$$\begin{array}{ll}
 & \mathcal{M}, g, w \models_{\sim} \langle\alpha\rangle\phi \\
\Leftrightarrow & \quad \{ \models_{\sim} \text{defn}\} \\
 & \text{there exists } v \in W \text{ with } (w,v) \in R_\alpha \text{ and } \mathcal{M}, g, v \models_{\sim} \phi \\
\Leftrightarrow & \quad \{ \text{ step } \star \} \\
 & \text{there exists } [v'] \in W/{\sim} \text{ with } ([w],[v']) \in (R/{\sim})_\alpha \text{ and } \mathcal{M}/{\sim}, g/{\sim}, [v'] \models_{\sim} \phi \\
\Leftrightarrow & \quad \{ \models_{\sim} \text{defn}\} \\
 & \mathcal{M}/{\sim}, g/{\sim}, [w] \models_{\sim} \langle\alpha\rangle\phi
\end{array}$$

Step $\star$: The direction "$\Rightarrow$" is trivial using $v' = v$ and the Induction Hypothesis. For the direction "$\Leftarrow$" assume $([w],[v']) \in (R/{\sim})_\alpha$ for some v'. By Remark 2 we know that there exists $\hat{v} \in [v']$ such that $(w,\hat{v}) \in R_\alpha$. From $\mathcal{M}/{\sim}, g/{\sim}, [v'] \models_\sim \phi$ it follows that $\mathcal{M}/\sim, g/\sim, [\hat{v}] \models_\sim \phi$ (since $[\hat{v}] = [v']$). By Ind. Hyp. we get $\mathcal{M}, g, \hat{v} \models_\sim \phi$. Since $(w,\hat{v}) \in R_\alpha$, we have $\mathcal{M}, g, w \models_\sim \langle\alpha\rangle\phi$.

The remaining cases are straightforward. $\square$

4 Recovering Modal Invariance for Bisimulation

Theorem 4 of the last section shows modal invariance of sentences in the $\mathcal{D}^{\downarrow}_{\sim}$-logic. In this section we will transfer this result to the case in which bisimulation equivalence is used instead of an observational isomorphism. In fact, this is a consequence of our general result (Theorem 6) that bisimulation equivalence can be characterised as an isomorphism in the category $\mathrm{Mod}^{\mathcal{D}^{\downarrow}_{\sim}}(A)$. Finally, we can even prove a Hennessy-Milner-Theorem for observational satisfaction; see Theorem 7.

Lemma 8. *Let $\mathcal{M} = (W, w_0, R)$ and $\mathcal{M}' = (W', w_0', R')$ be two A-models. If $\mathcal{M} \equiv \mathcal{M}'$, then $\mathcal{M}$ $\mathbf{iso}_\sim$ $\mathcal{M}'$.*

Proof. Since $\mathcal{M} \equiv \mathcal{M}'$ we can consider the greatest bisimulation relation $\sim^{\mathcal{M}}_{\mathcal{M}'} \subseteq W \times W'$ between $\mathcal{M}$ and $\mathcal{M}'$. We show that $\sim^{\mathcal{M}}_{\mathcal{M}'}$ is an isomorphism in the category $\mathrm{Mod}^{\mathcal{D}^{\downarrow}_{\sim}}(A)$. First, we note that $\sim^{\mathcal{M}}_{\mathcal{M}'}$ contains (w_0, w_0'). Then we show that $\sim^{\mathcal{M}}_{\mathcal{M}'}$ is an observational morphism. This is proved by using two simple properties of greatest bisimulations: The inverse of $\sim^{\mathcal{M}}_{\mathcal{M}'}$ is $\sim^{\mathcal{M}'}_{\mathcal{M}}$ and the composition of $\sim^{\mathcal{M}}_{\mathcal{M}'}$ and $\sim^{\mathcal{M}'}_{\mathcal{M}''}$ is $\sim^{\mathcal{M}}_{\mathcal{M}''}$.

- Condition 1 of Definition 6 holds, since $\sim^{\mathcal{M}}_{\mathcal{M}'}$ is a bisimulation.
- For 2 of Definition 6, let us suppose $(w,w') \in \sim^{\mathcal{M}}_{\mathcal{M}'}$ and $(v,v') \in \sim^{\mathcal{M}}_{\mathcal{M}'}$ and $(w,v) \in \sim^{\mathcal{M}}_{\mathcal{M}}$. Hence, we have $(w',w) \in \sim^{\mathcal{M}'}_{\mathcal{M}}$ and by composition of bisimulation relations and the fact that $\sim^{\mathcal{M}'}_{\mathcal{M}'}$ is the greatest bisimulation we get $(w',v') \in \sim^{\mathcal{M}'}_{\mathcal{M}'}$.
- For 3 of Definition 6, let us suppose $(w,w') \in \sim^{\mathcal{M}}_{\mathcal{M}'}$, $(w,v) \in \sim^{\mathcal{M}}_{\mathcal{M}}$ and $(w',v') \in \sim^{\mathcal{M}'}_{\mathcal{M}'}$ Hence, $(v,w) \in \sim^{\mathcal{M}}_{\mathcal{M}}$ and by composition of bisimulation relations and the fact that $\sim^{\mathcal{M}}_{\mathcal{M}'}$ is the greatest bisimulation we get $(v,v') \in \sim^{\mathcal{M}}_{\mathcal{M}'}$.

Finally, $\sim^{\mathcal{M}}_{\mathcal{M}'}$ is an isomorphism, since $(\sim^{\mathcal{M}}_{\mathcal{M}'} \cdot \sim^{\mathcal{M}'}_{\mathcal{M}}) = \sim^{\mathcal{M}}_{\mathcal{M}} = 1_{\mathcal{M}}$ and, conversely, $(\sim^{\mathcal{M}'}_{\mathcal{M}} \cdot \sim^{\mathcal{M}}_{\mathcal{M}'}) = \sim^{\mathcal{M}'}_{\mathcal{M}'} = 1_{\mathcal{M}'}$. $\square$

Theorem 6. *For any two A-models $\mathcal{M}$ and $\mathcal{M}'$, we have:*

$$\mathcal{M}\ \mathbf{iso}_\sim\ \mathcal{M}' \textit{ iff } \mathcal{M} \equiv \mathcal{M}'.$$

Proof. The direction "$\Rightarrow$" follows from condition 1 in Definition 6 and from Lemma 5. The direction "$\Leftarrow$" follows from Lemma 8. $\square$

As a consequence of Theorem 6 and the modal invariance for $\mathcal{D}^{\downarrow}_{\sim}$-logic (Theorem 4), we get modal invariance for bisimulation equivalence.

Corollary 3. *Let $\mathcal{M}, \mathcal{M}'$ be two A-models such that $\mathcal{M} \equiv \mathcal{M}'$. Then for any A-sentence φ, we have*

$$\mathcal{M} \models_{\sim} \varphi \textit{ iff } \mathcal{M}' \models_{\sim} \varphi.$$

As an example, we consider the two bisimilar $\{a\}$-models $\mathcal{M}$ and $\mathcal{M}'$ in Fig. 1 for which we have: $\mathcal{M} \models_{\sim} \downarrow x.\langle a \rangle x$ and $\mathcal{M}' \models_{\sim} \downarrow x.\langle a \rangle x$.

Corollary 4. *For any specification SP, its observational semantics $Mod_{\sim}(SP)$ is closed under $\equiv$.*

Proof. Direct consequence of Corollary 3. □

The next lemma provides the basis for proving the converse of Corollary 3 which will lead to a Hennessy-Milner Theorem w.r.t. $\mathcal{D}^{\downarrow}_{\sim}$-logic (if models are image finite).

Lemma 9. *Let $\mathcal{M}, \mathcal{M}'$ be two image finite[2] A-models and $w \in W, w' \in W'$ two states such that, for any A-sentence φ,*

$$\mathcal{M}, w \models_{\sim} \varphi \textit{ iff } \mathcal{M}', w' \models_{\sim} \varphi.$$

Then, there is a relation $h \subseteq W \times W'$ such that $(w, w') \in h$ and h satisfies the conditions "zig" and "zag" of a bisimulation; cf. Definition 5.

Proof. Let us consider the relation

$$h := \{(u, u') \mid \mathcal{M}, u \models_{\sim} \varphi \text{ iff } \mathcal{M}', u' \models_{\sim} \varphi,\ \varphi \text{ is an } A\text{-sentence}\}.$$

Obviously, $(w, w') \in h$. In order to prove "zig" we follow the strategy adopted in [8] for the proof of the so-called Hennessy-Milner Theorem. Planning to derive a contradiction, let us suppose there exists $(u, u') \in h, a \in A$ and $v \in W$ with $(u, v) \in R_a$, for which

$$\text{there is not a } v' \in W' \text{ such that } (u', v') \in R'_a \textit{ and } (v, v') \in h. \tag{1}$$

By assumption, $\mathcal{M}'$ is image finite and hence the set $R'_a[u'] := \{v'_1, \ldots, v'_k\}$ of a-successors of u' in $\mathcal{M}'$ is finite. It is also not empty since $(u, u') \in h$. By (1), for each $i \in \{1, \ldots, k\}$ there is a formula φ_i such that

$$\mathcal{M}, v \models_{\sim} \varphi_i \text{ and } \mathcal{M}', v'_i \not\models_{\sim} \varphi_i. \tag{2}$$

Hence, we have $\mathcal{M}, u \models_{\sim} \langle a \rangle(\varphi_1 \wedge \cdots \wedge \varphi_k)$ and $\mathcal{M}', u' \not\models_{\sim} \langle a \rangle(\varphi_1 \wedge \cdots \wedge \varphi_k)$, contradicting the assumption $\mathcal{M}, u \models_{\sim} \varphi$ iff $\mathcal{M}', u' \models_{\sim} \varphi$ for all A-sentences φ. Therefore h satisfies "zig". One can show analogously that h satisfies "zag". □

[2] i.e. in any state there are at most finitely many outgoing transitions labelled with the same atomic action.

Theorem 7. *Let $\mathcal{M}, \mathcal{M}'$ be two image finite A-models. Then the following properties are equivalent:*

1. $\mathcal{M}$ $\mathbf{iso}_\sim$ $\mathcal{M}'$,
2. $\mathcal{M} \equiv \mathcal{M}'$,
3. *for any A-sentence $\varphi, \mathcal{M} \models_\sim \varphi$ iff $\mathcal{M}' \models_\sim \varphi$.*

Proof. 1. $\Leftrightarrow$ 2.: Theorem 6.
2. $\Rightarrow$ 3.: Corollary 3.
2. $\Leftarrow$ 3.: Follows from Lemma 9 by taking for w and w' the initial states w_0 and w_0' of $\mathcal{M}$ and $\mathcal{M}'$ resp. $\square$

5 Relating Abstractor and Observational Semantics

Another possibility to provide an abstract semantics for a specification SP is to consider all models that are bisimulation equivalent to a "standard" model of SP, i.e. to a model of SP in the logic $\mathcal{D}^{\downarrow}$. This semantics is called *abstractor semantics*. In this section we investigate relationships between abstractor semantics and observational semantics. It turns out that results obtained in the framework of algebraic specifications, see [3], can be transferred to our logics $\mathcal{D}^{\downarrow}$ and $\mathcal{D}^{\downarrow}_{\sim}$ for reactive systems' specifications as well.

Definition 11 (Abstractor semantics). *The* abstractor semantics *of a specification $SP = (A, \Phi)$ is given by the class of models*

$$Abs_{\equiv}(SP) = \{\mathcal{M} \in \mathrm{Mod}^{\mathcal{D}^{\downarrow}}(A) \mid \mathcal{M} \equiv \mathcal{N} \textit{ for some } \mathcal{N} \in Mod(SP)\}.$$

Part 1 of the next theorem shows that observational semantics is a subclass of abstractor semantics. The converse does, in general, not hold. It may even be the case that standard models of a specification, which always belong to the abstractor semantics, do not belong to the observational semantics. This happens, if axioms of a specification contradict the observational equality between states. In order to illustrate this, let us consider the specification $SP = \langle \{a\}, \{\downarrow x.\langle a\rangle\neg x\}\rangle$. If we consider the model $\mathcal{M}'$ with two states depicted in Fig. 1, we have that $\mathcal{M}' \models \downarrow x.\langle a\rangle\neg x$ but $\mathcal{M}' \not\models_\sim \downarrow x.\langle a\rangle\neg x$ since the state w_1' reached by the a-transition from w_0' is observationally equal to w_0' but the negation $\neg x$ would forbid this. Hence, $\mathcal{M}' \in Mod(SP)$ but $\mathcal{M}' \notin Mod_\sim(SP)$. If, however, the axioms of a specification SP have the form that all models of SP in $\mathcal{D}^{\downarrow}$ belong to the observational semantics of SP in $\mathcal{D}^{\downarrow}_{\sim}$, then Part 2 of the next theorem shows that abstractor and observational semantics coincide.

Theorem 8. *Let $SP = (A, \Phi)$ be a specification.*

1. $Mod_\sim(SP) \subseteq Abs_{\equiv}(SP)$.
2. *$Mod(SP) \subseteq Mod_\sim(SP)$ if and only if $Mod_\sim(SP) = Abs_{\equiv}(SP)$.*

Proof. Part 1: Let $\mathcal{M} \in Mod_{\sim}(SP)$ and $\mathcal{M}/{\sim}$ its quotient according to Definition 10. By Theorem 5, we have that $\mathcal{M} \models_{\sim} \varphi$ iff $\mathcal{M}/{\sim} \models \varphi$ for all A-sentences φ and hence for all $\varphi \in \Phi$. Since $\mathcal{M} \in Mod_{\sim}(SP)$, we get $\mathcal{M}/{\sim} \in Mod(SP)$. Moreover, it is straightforward to show that $\mathcal{M} \equiv \mathcal{M}/{\sim}$, since the definition of $R/{\sim}$ entails that the relation $B \subseteq W \times W/{\sim}$ with $B = \{(w, [w]) \mid w \in W\}$ is a bisimulation. The (zig) condition of a bisimulation is obvious. For the (zag) condition assume that $([w], [v]) \in (R/{\sim})_a$. By Remark 2 we know that there exists $\hat{v} \in [v]$ such that $(w, \hat{v}) \in R_a$. Since $[\hat{v}] = [v]$ and $(\hat{v}, [\hat{v}]) \in B$ we have $(\hat{v}, [v]) \in B$. Finally, from $\mathcal{M} \equiv \mathcal{M}/{\sim}$ and $\mathcal{M}/{\sim} \in Mod(SP)$ we get $\mathcal{M} \in Abs_{\equiv}(SP)$.
Part 2: "$\Rightarrow$:" Assume $Mod(SP) \subseteq Mod_{\sim}(SP)$. By 1. we have $Mod_{\sim}(SP) \subseteq Abs_{\equiv}(SP)$. Let $\mathcal{M} \in Abs_{\equiv}(SP)$, i.e. there is a model $\mathcal{N} \in Mod(SP)$ such that $\mathcal{M} \equiv \mathcal{N}$. By assumption $\mathcal{N} \in Mod_{\sim}(SP)$, i.e., $\mathcal{N} \models_{\sim} \Phi$. By Corollary 3, $\mathcal{M} \models_{\sim} \Phi$, and hence $\mathcal{M} \in Mod_{\sim}(SP)$.

"$\Leftarrow$:" For this direction, assume $\mathcal{M} \in Mod(SP)$. Hence, $\mathcal{M} \in Abs_{\equiv}(SP)$. By assumption $Mod_{\sim}(SP) = Abs_{\equiv}(SP)$ and hence $\mathcal{M} \in Mod_{\sim}(SP)$. □

Finally we want to discuss the relationship of observational semantics with abstractor semantics in the context of fully abstract models. An A-model $\mathcal{M}$ is *fully abstract* if the observational equality $\sim_{\mathcal{M}}$ coincides with the set-theoretic equality of states. The *fully abstract semantics* of a specification $SP = (A, \Phi)$ in $\mathcal{D}^{\downarrow}$ is given by the class of its fully abstract models

$$Mod^{fa}(SP) = \{\mathcal{M} \in Mod(SP) \mid \mathcal{M} \textit{ is fully abstract}\}.$$

If we consider all A-models which are bisimulation equivalent to some fully abstract model of a specification we get the class

$$Abs^{fa}_{\equiv}(SP) = \{\mathcal{M} \in \mathrm{Mod}^{\mathcal{D}^{\downarrow}}(A) \mid \mathcal{M} \equiv \mathcal{N} \text{ for some } \mathcal{N} \in Mod^{fa}(SP)\}.$$

Our final result shows that this class coincides with the observational semantics of a specification. A similar result has been obtained for algebraic specifications in [3].

Theorem 9. *For any specification $SP = (A, \Phi)$, $Mod_{\sim}(SP) = Abs^{fa}_{\equiv}(SP)$.*

Proof. The proof of the inclusion "$\subseteq$" is the same as for part 1 in Theorem 8 taking into account that $\mathcal{M}/{\sim}$ is fully abstract. It remains to show $Abs^{fa}_{\equiv}(SP) \subseteq Mod_{\sim}(SP)$. Let $\mathcal{M} \in Abs^{fa}_{\equiv}(SP)$. Then $\mathcal{M} \equiv \mathcal{N}$ for some $\mathcal{N} \in Mod^{fa}(SP)$. Since $\mathcal{N} \models \Phi$ and $\mathcal{N}$ is fully abstract, we have $\mathcal{N} \models_{\sim} \Phi$. Since $\mathcal{M} \equiv \mathcal{N}$ we get, by Corollary 3, that $\mathcal{M} \models_{\sim} \Phi$. Hence $\mathcal{M} \in Mod_{\sim}(SP)$. □

6 Conclusion

This paper follows the motivations of [7] on the definition of a logic to develop reactive systems in a stepwise manner from abstract requirements specifications to concrete specifications of processes. In this context, the quest for a more liberal semantics appeared that is closed under behavioural equivalence. Following ideas from algebraic specifications of data structures, we have proposed a new logic for specifications of reactive systems, called $\mathcal{D}^{\downarrow}_{\sim}$, which satisfies both the modal invariance property and a Hennessy-Milner Theorem. The key to achieve this was a new, relaxed satisfaction relation, which allows interpreting state variables up to bisimilarity.

There are several interesting research questions to be pursued on the basis of $\mathcal{D}^{\downarrow}_{\sim}$. For instance, we want to investigate how $\mathcal{D}^{\downarrow}_{\sim}$ can be extended to an institution. A preliminary study shows that a straightforward extension using functions $\sigma : A \to A'$ between action sets as signature morphisms would not work. The reason is that A' may introduce new actions that distinguish, in some A'-models, states which are observationally equal when using only actions in A. Then the satisfaction condition of an institution would not be valid. Therefore we must investigate adjustments on signatures, signature morphisms and models to establish the satisfaction condition. Another interesting extension to follow concerns the incorporation of weak bisimulations which would allow further behavioural abstraction w.r.t. silent transitions.

Acknowledgement. We would like to thank the anonymous reviewers of this paper for their careful reviews with many useful comments and suggestions.

References

1. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press, Cambridge (2007)
2. Bidoit, M., Hennicker, R.: Constructor-based observational logic. J. Log. Algebr. Program. **67**(1–2), 3–51 (2006)
3. Bidoit, M., Hennicker, R., Wirsing, M.: Behavioural and abstractor specifications. Sci. Comput. Program. **25**(2–3), 149–186 (1995)
4. Bräuner, T.: Hybrid Logic and Its Proof-Theory. Applied Logic Series. Springer, Dordrecht (2010). https://doi.org/10.1007/978-94-007-0002-4
5. Goguen, J.A., Malcolm, G.: A hidden agenda. Theor. Comput. Sci. **245**(1), 55–101 (2000)
6. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge (2000)
7. Madeira, A., Barbosa, L.S., Hennicker, R., Martins, M.A.: Dynamic logic with binders and its application to the development of reactive systems. In: Sampaio, A., Wang, F. (eds.) ICTAC 2016. LNCS, vol. 9965, pp. 422–440. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46750-4_24
8. Milner, R.: Communication and Concurrency. PHI Series in Computer Science. Prentice Hall, Upper Saddle River (1989)
9. Reichel, H.: Behavioural validity of conditional equations in abstract data types. In: Proceedings of the Vienna Conference on Contributions to General Algebra 3, pp. 301–324. B. G. Teubner Verlag (1985)
10. Sannella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-17336-3

Towards Critical Pair Analysis for the Graph Programming Language GP 2

Ivaylo Hristakiev and Detlef Plump(✉)

University of York, York, UK
{ish503,detlef.plump}@york.ac.uk

Abstract. We present the foundations of critical pair analysis for the graph programming language GP 2. Our goal is to develop a static checker that can prove or refute confluence (functional behaviour) for a large class of graph programs. In this paper, we introduce *symbolic* critical pairs of GP 2 rule schemata, which are labelled with expressions, and establish the completeness and finiteness of the set of symbolic critical pairs over a finite set of rule schemata. We give a procedure for their construction.

1 Introduction

A common programming pattern in the graph programming language GP 2 [16] is to apply a set of attributed graph transformation rules as long as possible. To execute a set of rules $\{r_1, \ldots, r_n\}$ for 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 such an iteration. 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 [8]. Programs can perform computations on labels by using rules labelled with expressions (also known as attributed rules). GP 2's label algebra consists of integers, character strings, and heterogeneous lists of integers and strings. Rule application can be seen as a two-stage process where rules are first instantiated, by replacing variables with values and evaluating the resulting expressions, 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 *critical pairs*, which represent conflicts in minimal context [5,15]. A conflict between two rule applications arises when one of the steps deletes or relabels an item matched by the other. In the presence of termination, one can check if all critical pairs are *strongly joinable*, and thus establish that the set of transformation rules is confluent.

I. Hristakiev—Supported by a Doctoral Training Grant from the Engineering and Physical Sciences Research Council (EPSRC) in the UK.

P. James and M. Roggenbach (Eds.): WADT 2016, LNCS 10644, pp. 153–169, 2017.
https://doi.org/10.1007/978-3-319-72044-9_11

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 conventional graph transformation rules [12], and thus the analysis cannot be automated as part of a confluence checker. Furthermore, when constructing the labels of critical pairs, it has been observed [4, p. 198] that syntactic unification of the labels of overlapping graphs is not sufficient (as proposed by [10]). This is because the constructed set of critical pairs need not represent all conflicts. Instead, one has to take into account all equations valid in the attribute algebra. This problem is circumvented in [6,7] by imposing a severe restriction that avoids the need for unification altogether, namely to only allow rules labelled with variables or variable-free expressions.

In this paper, we do not use such restrictions. We rather define *symbolic* critical pairs which are labelled with expressions, and give an algorithm for their construction based on our unification algorithm for GP 2 expressions [11]. As a by-product of this construction, it is easy to show that a finite set of rule schemata gives rise to a finite set of symbolic critical pairs. We then prove that the generated critical pairs are complete in that they represent all conflicts of the given set of rule schemata. This proof is based on the completeness of the GP 2 unification algorithm.

We assume the reader to be familiar with basic notions of the double-pushout approach to graph transformation (see [4]).

Related Work. The approach of [14] also defines symbolic critical pairs in the context of symbolic graph transformation where symbolic graphs are transformed via symbolic rules (rules equipped with first-order logical formulas). Symbolic critical pairs represent all possible conflicts between such symbolic rules. However, it is important to stress the differences with our approach. No construction algorithm is given for these critical pairs whereas we give a construction for the GP 2 setting. In fact, that approach treats attribute algebras as a parameter, and thus a general construction algorithm cannot be given. Even so, a topic of future work is the relaxed notion of conflict where a minimal pair of derivations is critical if the pair does not commute when attribute semantics are taken into account.

The differences with critical pairs in the attributed setting of [4] are similar to the above. In this setting, graph attributes are represented via special *data* nodes and linked to ordinary graph nodes/edges via attribution edges, giving rise to infinite graphs. Attributed rules contain a data node for each term in the term algebra $T(X)$. The critical pair construction however is restricted to rules whose attributes are variables or variable-free, e.g. see [6]. An earlier version of the construction was based on computing a most general unifier [10], which renders the critical pairs incomplete. As above, attribute algebras are treated as parameters.

2 Graphs and Graph Programs

In this section, we present the approach of GP 2 [2,16], a domain-specific language for rule-based graph manipulation. The principal programming units of GP 2 are rule schemata $\langle L \leftarrow K \rightarrow R \rangle$ labelled with expressions that operate on host graphs labelled with concrete values. The language allows to combine schemata into programs. The definition of GP 2's latest version, together with a formal operational semantics, can be found in [2].

2.1 Background

Labelled graphs. We start by recalling the basic notions of partially labelled graphs and their morphisms.

A (partially) labelled graph G consists of finite sets V_G and E_G of nodes and edges, source and target functions for edges $s_G, t_G\colon E_G \rightarrow V_G$, and a partial node/edge labelling function $l_G\colon V_G + E_G \rightarrow \mathcal{L}$ over a (possibly infinite) label set $\mathcal{L}$. Given a node or edge x, $l_G(x) = \perp$ expresses that $l_G(x)$ is undefined[1]. The graph G is totally labelled if l_G is a total function. The classes of partially and totally labelled graphs over a label set $\mathcal{L}$ are denoted by $\mathcal{G}_\perp(\mathcal{L})$ and $\mathcal{G}(\mathcal{L})$.

A premorphism $g\colon G \rightarrow H$ consists of two functions $g_V\colon V_G \rightarrow V_H$ and $g_E\colon E_G \rightarrow E_H$ that preserve sources and targets. A graph morphism g is a premorphism that preserves labels of nodes and edges, that is $l_H\big(g(x)\big) = l_G(x)$ for all $x \in \mathit{Dom}(l_G)$. A morphism g preserves undefinedness if it maps unlabelled items of G to unlabelled items in H. Morphism g is an inclusion if $g(x) = x$ for all items x in G. Note that inclusions need not preserve undefinedness. Morphism g is injective (surjective) if g_V and g_E are injective (surjective), and is an isomoprhism (denoted by $\cong$) if it is injective, surjective and preserves undefinedness. The class of injective label preserving morphisms is denoted as $\mathcal{M}$ for short, and the class of injective label and undefinedness preserving morphisms is denoted as $\mathcal{N}$.

Partially labelled graphs and label-preserving morphisms constitute a category [8,9]. Composition of morphisms is defined componentwise. What is special about this category is that pushouts need not always exist, and not all pushouts along $\mathcal{M}$-morphisms are natural[2].

GP 2 labels. The types `int` and `string` represent integers and character strings. The type `atom` is the union of `int` and `string`, and `list` represents lists of atoms. Given lists l_1 and l_2, we write $l_1 : l_2$ for the concatenation of l_1 and l_2 (not to be confused with the list-cons operator in Haskell). Atoms are lists of length one. The empty list is denoted by `empty`. Variables may appear in labels in rules and are typed over the above categories. Labels in rule schemata are built up from constant values, variables, and operators - the standard arithmetic operators

[1] We do not distinguish between nodes and edges in statements that hold analogously for both sets.

[2] A pushout is natural if it is also a pullback.

for integer expressions (including the unary minus), string concatenation for string expressions, `length` operator for list and string expressions, `indegree` and `outdegree` operators for nodes. In pictures of graphs, nodes or edges that are shown without a label are implicitly labelled with the empty list, while unlabelled items in interfaces are labelled with $\perp$ to avoid confusion.

Additionally, a label may contain an optional *mark* which is one of `red`, `green`, `blue`, `grey` and `dashed` (where `grey` and `dashed` are reserved for nodes and edges, respectively). The mark component of labels is represented graphically rather than textually. For example, all edges of the rule schema `series` in Fig. 2 have the label (`empty`, `dashed`).

Rule schemata and direct derivations. In order to compute with labels, it is crucial that nodes and edges can be relabelled. The double-pushout approach with partially labelled interface graphs is used as a formal basis [8].

To apply a rule schema to a graph, the schema is first instantiated by evaluating its labels according to some assignment α. An assignment α maps each variable occurring in a given schema to a value in GP 2's label algebra. Its unique extension α^* evaluates the schema's label expressions according to α. For short, we denote GP 2's label algebra as A. Its corresponding term algebra over the same signature is denoted as $T(X)$, and its terms are used as graph labels in rule schemata. Here X is the set of variables occurring in schemata. To avoid an inflation of symbols, we sometimes equate A or $T(X)$ with the union of its carrier sets.

A GP 2 rule schema $r = \langle L \leftarrow K \rightarrow R \rangle$ consists of two inclusions $K \rightarrow L$ and $K \rightarrow R$ such that L and R are graphs in $\mathcal{G}(T(X))$ and K is a graph in $\mathcal{G}_\perp(T(X))$. Consider a graph G in $\mathcal{G}_\perp(T(X))$ and an assignment $\alpha\colon X \rightarrow A$. The instance G^α is the graph in $\mathcal{G}_\perp(A)$ obtained from G by replacing each label l with $\alpha^*(l)$. The instance of a rule schema $r = \langle L \leftarrow K \rightarrow R \rangle$ is the rule $r^\alpha = \langle L^\alpha \leftarrow K^\alpha \rightarrow R^\alpha \rangle$.

A direct derivation via rule schema r and assignment α between host graphs $G, H \in \mathcal{G}(A)$ consists of two natural pushouts as in Fig. 1. We denote such a derivation by $G \overset{r,g,\alpha}{\Longrightarrow} H$. Rules may also be applied to graphs in $\mathcal{G}(T(X))$. In this case assignments become substitutions $\sigma : X \rightarrow T(X)$. This will be useful later for critical pairs which are labelled over $T(X)$.

$$\begin{tikzcd}
L \arrow[d, mapsto] & K \arrow[l] \arrow[r] \arrow[d, mapsto] & R \arrow[d, mapsto] \\
L^\alpha \arrow[d, "g"'] \arrow[dr, phantom, "\text{NPO}"] & K^\alpha \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "\text{NPO}"] & R^\alpha \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

Fig. 1. A direct derivation

In [8] it is shown that in case the interface graph K has unlabelled items, their images in the intermediate graph D are also unlabelled by the condition that the pushouts are natural. Given a rule r and a graph G together with an injective match $g\colon L \to G$ satisfying the *dangling condition* (no node in $g(L) - g(K)$ is incident to an edge in $G - g(L)$), there exists a unique double natural pushout [8, Theorem 1].

When a rule schema is graphically declared as done in Fig. 2, the interface is represented by the node numbers in L and R. Nodes without numbers in L are to be deleted and nodes without numbers in R are to be created. All variables in R have to occur in L so that for a given match of L in a host graph, applying the rule schema produces a graph that is unique up to isomorphism.

Program constructs. A GP 2 program consists of declarations of rule schemata and macros, and a main command sequence which controls their application order. The language offers several operators for combining subprograms - the postfix operator '!' iterates a program as long as possible; sequential composition 'P; Q'; a rule set $\{r_1, \ldots, r_n\}$ tries to non-deterministically apply any of the schemata (failing if none are applicable); `if` C `then` P `else` Q allows for conditional branching (*C, P, Q* are arbitrary command sequences) meaning that if the program C succeeds on a copy of the host graph then P is executed on the original, if C fails then Q is executed on the original host graph.

Simple lists. The values of rule schema variables at execution time are determined by graph matching. To ensure that matches induce unique "actual parameters", expressions on the left-hand side of a rule schema must have a simple shape. A simple list expression [2] contains no arithmetic, length or degree operators, at most one occurrence of a list variable, at most one occurrence of a string variable per string expression. For example, `a:x` and `y:n:n` are simple expressions (`a, n` are atom variables; `x, y` are list variables) whereas `n` $*$ `2` or `x:y` are not simple.

Assumptions. In this paper, we make several assumptions. First, we further restrict simple lists to not contain string concatenation (`.`) and unary minus (`-`) operators. These operators inflate the unification algorithm of [11] which we use for the construction of critical pairs, without posing a substantial challenge. Second, a proper treatment of GP 2 *conditional* rule schemata requires extra technicalities, and hence we consider unconditional schemata only. Third, we assume rule schemata to be *left-linear* (see Sect. 3).

2.2 Example: Recognition of Series-Parallel Graphs

As a motivating example, consider a GP program that recognizes *series-parallel graphs*. These graphs have been introduced as models of electrical networks [3], and are interesting from a complexity point of view since many graph problems, some of which NP-complete, are solvable in linear time for these graphs e.g. maximum matching, maximum independent set, Hamiltonian completion.

```
Main = unlabel!; Reduce!; delete; if nonempty then fail
Reduce = {series, parallel}
```

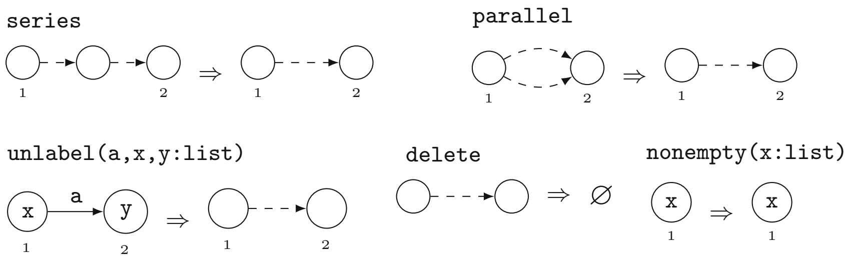


Fig. 2. GP 2 program recognizing series-parallel graphs.

These graphs are recognized by means of graph reduction: for a host graph G, apply a set of size-reducing rules `Reduce` $=$ {`series`, `parallel`} as long as possible, obtaining a result graph H, then check whether H is isomorphic to ○⟶○ (ignoring labels) to decide whether the original graph G is series-parallel.

A GP 2 program implementing the above algorithm is presented in Fig. 2. Given a host graph G, the program works as follows. First, it removes all labels by applying the `unlabel` rule as long as possible (labels do not play a role in whether a graph is series-parallel or not). Applying `unlabel` amounts to non-deterministically selecting a subgraph of the host graph that matches `unlabel`'s left graph, relabelling the matched nodes with the empty list and recreating the connecting edge as `dashed` to avoid non-termination.

Afterwards, the `Reduce` rules are applied as long as possible. To determine whether the resulting graph has the correct shape, the program first attempts to delete the correct result graph (see above) then checks whether this yields the empty graph. If either the deletion or the non-empty check fails, then the program fails. In this context, termination of the program with a proper graph means the host graph G is series-parallel, and failure means G is not series-parallel.

However, if the non-deterministic reduction results in a graph other than ○- - ▸○, we need to be sure that no other reduction sequence ends in that graph. Therefore, the correctness of the above recognition algorithm depends on the *confluence* of the loops `unlabel!` and `Reduce!`.

Confluence [15] is a property of a rewrite system that ensures that any pair of derivations on the same host graph can be joined again thus leading to the same result, and is an important property for many kinds of graph transformation systems. A confluent computation is globally deterministic despite possible local non-determinism. The main technique for confluence analysis is based on the study of critical pairs which are conflicts in minimal context. However, the previous results in critical pair analysis do not cover rule schemata and GP. This raises the question of how to check whether loops such as `unlabel!` and `Reduce!` are confluent or not.

Infinity of conventional critical pairs. To construct critical pairs for the above case, we can consider the infinite set of all rules obtained by arbitrary instantiations of the schemata and compute conventional critical pairs over those (see, for example, [4]). Since there would be an infinite number of rule instances to consider, the set of conventional critical pairs would also be infinite.

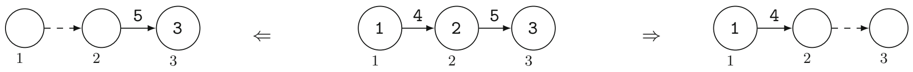


Fig. 3. A conventional critical pair of `unlabel` with itself.

For example, consider the conflicting pair of derivations in Fig. 3. The middle graph is obtained by overlapping the left-hand graph of `unlabel` with itself, and the graphs on either side are the results of applying the schema in conflicting ways. The pair is in conflict because both derivations relabel a common node (2) to the empty list. The instantiating assignment of `unlabel` and its copy is $\alpha = \{\mathtt{x1} \to 1, \mathtt{y1} \to 2, \mathtt{x2} \to 2, \mathtt{y2} \to 3, \mathtt{a1} \to 4, \mathtt{a2} \to 5\}$ where the variables are indexed to signify from which `unlabel` instance they originate.

3 Unification of GP 2 List Expressions

Below we review the problem of unifying GP 2 list expressions. The problem arises when having to overlap graphs labelled with expressions to compute critical pairs. Unification has a long history in the automated deduction community, see for instance [1] for an introduction. We use our AU-unification algorithm [11] as a solution, and its properties - namely completeness and termination. As mentioned in the Introduction, the use of our algorithm is motivated by the need to respect the axioms valid in the label algebra. (See Sect. 6 for more on the relation between critical pairs and unification.)

A substitution maps GP variables to expressions $\sigma : X \to T(X)$. For example, we write $\sigma = \{\mathtt{x} \mapsto \mathtt{x} + \mathtt{1}\}$ for the substitution that maps $\mathtt{x}$ (an integer variable) to $\mathtt{x} + \mathtt{1}$ and every other variable to itself. Applying a substitution σ to an expression t, denoted by $t\sigma$, means to replace every variable x in t by $\sigma(x)$ simultaneously. In the above example, $(\mathtt{x} : -\mathtt{x})\sigma = (\mathtt{x} + \mathtt{1}) : -(\mathtt{x} + \mathtt{1})$. Composition of substitutions λ and σ is written as $\lambda \circ \sigma$ (λ *after* σ). Given substitutions $\sigma_1, \ldots, \sigma_n$ with pairwise disjoint domains, their composition $\sigma_1 \circ \ldots \circ \sigma_n$ is commutative. A substitution σ is *more general* on a set of variables X than a substitution θ if there exists a substitution λ such that $x\theta =_{\mathrm{AU}} (x\sigma)\lambda$ for all $x \in X$. In this case we write $\sigma \leqq_X \theta$ and say that θ is an instance of σ on X. Here $=_{\mathrm{AU}}$ is the equivalence relation on expressions generated by the axioms of associativity and unity of list concatenation $\mathrm{AU} = \{\mathtt{x} : (\mathtt{y} : \mathtt{z}) = (\mathtt{x} : \mathtt{y}) : \mathtt{z},\ \mathtt{empty} : \mathtt{x} = \mathtt{x},\ \mathtt{x} : \mathtt{empty} = \mathtt{x}\}$, where $\mathtt{x}, \mathtt{y}, \mathtt{z}$ are list variables. If one considers ordinary equality $=$, then the unification is called *syntactic*.

A *unification problem* is an equation P of the form $s =^? t$ where s and t are simple list expressions without common variables. A *unifier* of P is a substitution σ over the set of variables occurring in P (denoted as $\mathrm{Var}(P)$) such that $s\sigma =_{\mathrm{AU}} t\sigma$.

A set $\mathcal{C}$ of unifiers is a *complete set of unifiers* of a unification problem P if for each unifier θ there exists $\sigma \in \mathcal{C}$ such that $\sigma \leqq_{\mathrm{Var}(P)} \theta$. This essentially means that any substitution that is a unifier is an instance of some unifier in $\mathcal{C}$. The set $\mathcal{C}$ is also *minimal* if each pair of distinct unifiers in $\mathcal{C}$ are incomparable w.r.t. $\leqq_{\mathrm{Var}(P)}$. If a unification problem is not unifiable, then by convention $\varnothing$ is its minimal complete set of unifiers.

The paper [11] gives an algorithm for solving unification problems between GP 2 list expressions. The algorithm produces a finite complete set of unifiers for a given problem. The assumptions of the algorithm are: (1) simple expressions, as presented in Sect. 2; (2) *left-linearity* (see below); (3) infinite pool of fresh variables. We can summarize the results of [11] as the following theorem.

Theorem 1 (Unification algorithm). *There exists an algorithm solving the following problem:*

Input: *A unification problem $s =^? t$ between simple list expressions s and t without common variables*

Output: *A finite complete set of unifiers* $\mathrm{UNIF}(s =^? t)$

We write $\mathrm{UNIF}(P)$ for the set of unifiers returned by the unification algorithm. For a finite system of independent unification problems $(P_1, \ldots, P_n)$[3], the extension of UNIF is defined to be the set of unifiers obtained by combining the unifiers of each individual unification problem:

$$\mathrm{UNIF}(P_1, \ldots, P_n) = \big\{\sigma_1 \circ \ldots \circ \sigma_n \mid \sigma_i \in \mathrm{UNIF}(P_i),\ 1 \leq i \leq n\big\}$$

For example, if $\mathrm{UNIF}(P_1) = \{\alpha, \beta\}$ and $\mathrm{UNIF}(P_2) = \{\lambda\}$, then $\mathrm{UNIF}(P_1, P_2) = \{\alpha \circ \lambda,\ \beta \circ \lambda\}$. By the above result, this set is finite. It is also complete since it contains all combinations of unifiers.

Left-linearity. The left-linearity assumption states that no list variables are shared between items in a left-hand graph of a rule schema. This is sufficient to ensure that we can apply our generalized algorithm in the construction of critical pairs (Theorem 2) as the system of equations resulting from overlapping left-hand graphs will have a finite set of solutions. Without this assumption it is easy to construct two rule schemata that induce the system of equations $\{\mathtt{x} : \mathtt{1} =^? \mathtt{y},\ \mathtt{1} : \mathtt{x} =^? \mathtt{y}\}$. The system is not independent as the list variables $\mathtt{x}$ and $\mathtt{y}$ are shared between the two equations. We can solve each equation separately, but the composition of their unifiers does not produce a unifier for the system. In fact, this system has an *infinite* minimal complete set of solutions $\{\mathtt{x} \mapsto \mathtt{empty}, \mathtt{y} \mapsto \mathtt{1}\}$, $\{\mathtt{x} \mapsto \mathtt{1}, \mathtt{y} \mapsto \mathtt{1} : \mathtt{1}\}$, $\{\mathtt{x} \mapsto \mathtt{1} : \mathtt{1}, \mathtt{y} \mapsto \mathtt{1} : \mathtt{1} : \mathtt{1}\}, \ldots$.

[3] Two unification problems are independent if they do not share list variables.

Unification example. The minimal complete set of unifiers of the problem $\langle \mathtt{a} : \mathtt{x} =^? \mathtt{y} : 2\rangle$ (where $\mathtt{a}$ is an atom variable and $\mathtt{x}$, $\mathtt{y}$ are list variables) is $\{\sigma_1, \sigma_2\}$ with $\sigma_1 = \{\mathtt{a} \mapsto 2, \mathtt{x} \mapsto \mathtt{empty}, \mathtt{y} \mapsto \mathtt{empty}\}$ and $\sigma_2 = \{\mathtt{x} \mapsto \mathtt{z} : 2, \mathtt{y} \mapsto \mathtt{a} : \mathtt{z}\}$. We have $(\mathtt{a} : \mathtt{x})\sigma_1 = 2 : \mathtt{empty} =_{\mathrm{AU}} 2 =_{\mathrm{AU}} \mathtt{empty} : 2 = (\mathtt{y} : 2)\sigma_1$ and $(\mathtt{a} : \mathtt{x})\sigma_2 = \mathtt{a} : (\mathtt{z} : 2) =_{\mathrm{AU}} (\mathtt{a} : \mathtt{z}) : 2 = (\mathtt{y} : 2)\sigma_2$. Other unifiers such as $\sigma_3 = \{\mathtt{x} \mapsto 2, \mathtt{y} \mapsto \mathtt{a}\}$ are instances of σ_2, hence set of unifiers $\{\sigma_1, \sigma_2, \sigma_3\}$ is complete but not minimal.

4 Symbolic Critical Pairs

In this section, we define the notions of independence and conflicts for rule schema rewriting as done in [12]. Then we develop the notion of symbolic critical pairs describing conflicts in minimal context. Symbolic critical pairs allow for the realization of a static confluence checker.

Independence of schema derivations. Two schema derivations are independent if neither derivation deletes or relabels any common item. This can be expressed as an 'existence-of-morphisms' condition. Independent derivations can be interchanged, leading to the same result. This property is known as the Local Church-Rosser Theorem, shown in [12] for the case of rule schemata.

In the rest of the paper we assume that the variables occurring in different rule schemata are distinct, which can always be achieved by variable renaming.

Definition 1 (Independence of derivations). Two rule schema direct derivations $G \overset{r_1,m_1,\alpha}{\Longrightarrow} H_1$ and $G \overset{r_2,m_2,\alpha}{\Longrightarrow} H_2$ are *independent* if the plain derivations with relabelling $G \overset{r_1^\alpha,m_1}{\Longrightarrow} H_1$ and $G \overset{r_2^\alpha,m_2}{\Longrightarrow} H_2$ are independent, meaning that there exist morphisms $i : L_1^\alpha \to D_2$ and $j : L_2^\alpha \to D_1$ such that $f_2 \circ i = m_1$ and $f_1 \circ j = m_2$.

$$\begin{tikzcd}
R_1 \arrow[d, mapsto] & K_1 \arrow[l] \arrow[r] \arrow[d, mapsto] & L_1 \arrow[d, mapsto] & & L_2 \arrow[d, mapsto] & K_2 \arrow[l] \arrow[r] \arrow[d, mapsto] & R_2 \arrow[d, mapsto] \\
R_1^\alpha \arrow[d] & K_1^\alpha \arrow[l] \arrow[r] \arrow[d] & L_1^\alpha \arrow[dr, hook, "m_1"] \arrow[drrr, dotted, "i"] & & L_2^\alpha \arrow[dl, hook', "m_2"'] \arrow[dlll, dotted, "j"'] & K_2^\alpha \arrow[l] \arrow[r] \arrow[d] & R_2^\alpha \arrow[d] \\
H_1 & D_1 \arrow[l] \arrow[rr, "f_1"] & & G & & D_2 \arrow[ll, "f_2"'] \arrow[r] & H_2
\end{tikzcd}$$

Two direct derivations are in *conflict* if they are not independent. There are different types of conflict that can arise between two direct derivations. One option is to have that either derivation deletes graph elements which are used by the other (delete-use conflict). The other option is that one derivation relabels graph elements used by the other (relabelling conflict).

Example of conflict. Fig. 4 shows two direct derivations $H_1 \Leftarrow G \Rightarrow H_2$ that use instances of the rule schema `unlabel`. The derivations are in conflict - there are no morphisms $L_1 \to D_2$ and $L_2 \to D_1$ with the desired properties. The problem is that node 2 gets relabelled. Note that the edge from node 1 to 3 is never matched and is therefore preserved during both derivations.

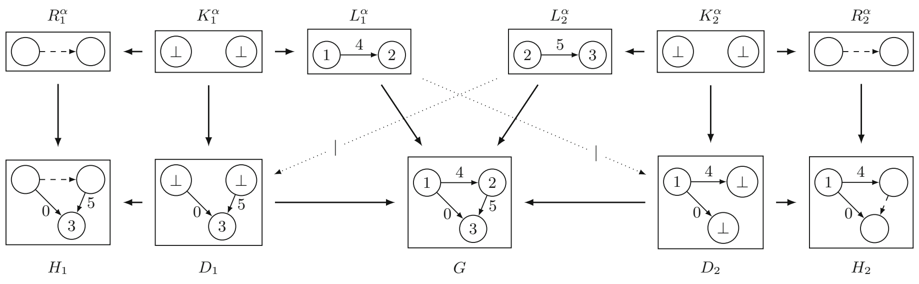


Fig. 4. Conflict due to relabelling.

Symbolic critical pairs

Critical pairs allow for the static confluence analysis of rule schema rewriting. Each conflict that may occur during the graph transformation is represented by a critical pair. Hence, it is possible to foresee each conflict by computing all critical pairs statically. Each pair of rule schemata induces a set of critical pairs.

We define critical pairs that are labelled with expressions rather than from a concrete data domain. Each symbolic critical pair represents a possibly infinite set of conflicting host graph derivations. What is special about our critical pairs is that they show the conflict in the most abstract way.

A pair of derivations $T_1 \overset{r_1,m_1,\sigma}{\Longleftarrow} S \overset{r_2,m_2,\sigma}{\Longrightarrow} T_2$ between graphs labelled with expressions is a critical pair if it is in conflict and minimal. Minimality means the pair of matches (m_1, m_2) is jointly surjective – the graph S can be considered as a suitable overlap of L_1^σ and L_2^σ. Two items $x \in L_1$ and $y \in L_2$ are overlapped if $m_1(x) = m_2(y)$, which induces a unification problem $l(x) =^? l(y)$ between their labels. Formally, overlapping graphs L_1^σ and L_2^σ induces a system of unification problems:

$$\mathrm{EQ}(L_1^\sigma \xrightarrow{m_1} S \xleftarrow{m_2} L_2^\sigma) = \{l_{L_1^\sigma}(a) \overset{?}{=} l_{L_2^\sigma}(b) \mid (a,b) \in L_1^\sigma \times L_2^\sigma \text{ with } m_1(a) = m_2(b)\}$$

The substitution σ is taken from a complete set of unifiers of the above system of problems and is used to instantiate the schemata. To avoid a circular definition, the system can instead be constructed over the induced premorphisms $L_i \to S$ rather than over $L_i^\sigma \to S$, $i = 1, 2$.

To disambiguate between the critical pairs of our approach and conventional critical pairs found in literature (e.g. [15]), we introduce them as *symbolic*[4].

Definition 2 (Symbolic Critical Pair). *A symbolic critical pair is a pair of direct derivations* $T_1 \overset{r_1,m_1,\sigma}{\Longleftarrow} S \overset{r_2,m_2,\sigma}{\Longrightarrow} T_2$ *on graphs labelled with expressions such that:*

(1) σ *is a substitution in* $\mathrm{UNIF}(\mathrm{EQ}(L_1 \xrightarrow{m_1} S \xleftarrow{m_2} L_2))$ *where* L_1 *and* L_2 *are the left-hand graphs of* r_1 *and* r_2*,* m_1 *and* m_2 *are premorphisms, and*

[4] The paper [14] introduces symbolic critical pairs in the setting of symbolic graph transformation where graphs are combined with first-order logic formulas.

(2) the pair of derivations is in conflict, and
(3) $S = m_1(L_1^\sigma) \cup m_2(L_2^\sigma)$, meaning S is minimal, and
(4) $r_1^\sigma = r_2^\sigma$ implies $m_1 \neq m_2$. □

We assume the derivations are via left-linear rule schemata for UNIF to return a finite set of unifiers. Terms appearing in left-hand graphs are restricted to simple lists with an optional mark component as presented in Sect. 2.

Critical pairs of the Series-Parallel Program. An example symbolic critical pair of the rules in Fig. 2 is shown in Fig. 5 where the middle graph is obtained by overlapping the left-hand graph of the `unlabel` schema with itself, and the graphs on either side are the results of applying the schema in conflicting ways.

The pair is in conflict because both derivations relabel a common node (2) to the empty list. Note that this symbolic critical pair looks very similar to the pair of conflicting derivations in Fig. 3. In fact, they are related by the instantiation $\lambda = \{\mathtt{x1} \to \mathtt{1}, \mathtt{y1} \to \mathtt{2}, \mathtt{y2} \to \mathtt{3}, \mathtt{a1} \to \mathtt{4}, \mathtt{a2} \to \mathtt{5}\}$ where the variables are indexed (as usual) to signify from which `unlabel` instance they originate.

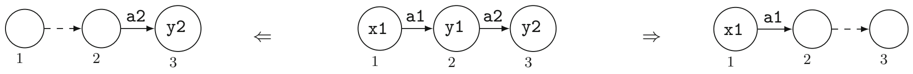


Fig. 5. A symbolic critical pair of `unlabel` with itself.

There are 4 more symbolic critical pairs obtained by self-overlapping `unlabel`. In addition, the set `Reduce` gives rise to 3 symbolic critical pairs. Both loops `unlabel!` and `Reduce!` can be shown to be locally confluent by analysing these critical pairs under a suitable notion of critical pair joinability (to be published elsewhere). It follows that the program in Fig. 2 is correct.

5 Construction and Finiteness of Symbolic Critical Pairs

We give an algorithm for the construction of symbolic critical pairs that respects the equations of GP's label algebra, namely the associativity and unit laws of list concatenation. The construction is given as the following theorem.

Theorem 2 (Construction of Symbolic Critical Pairs). *Given left-linear rule schemata $r_1 = \langle L_1 \leftarrow K_1 \to R_1 \rangle$ and $r_2 = \langle L_2 \leftarrow K_2 \to R_2 \rangle$, the following construction computes all symbolic critical pairs of r_1 and r_2:*

1. *Compute all overlaps of L_1 and L_2, giving rise to pairs of jointly surjective premorphsisms (m_1, m_2) into an unlabelled graph S.*
2. *For each overlap check that m_1 and m_2 satisfy the dangling condition w.r.t. r_1 and r_2.*
3. *For each overlap compute the set of unifiers* UNIF(EQ($L_1 \overset{m_1}{\to} S \overset{m_2}{\leftarrow} L_2$)).

4. For each unifier σ from the above set and its overlap do the following:
   (a) Instantiate r_1 and r_2 via σ to obtain the rules r_1^σ and r_2^σ, and if $r_1^\sigma = r_2^\sigma$ check that $m_1 \neq m_2$.
   (b) Define the labelling function of S as

$$l_S(x) = \begin{cases} l_{L_1^\sigma}(x') & \textit{if } \exists x' \in L_1^\sigma \textit{ such that } m_1(x') = x \\ l_{L_2^\sigma}(x') & \textit{if } \exists x' \in L_2^\sigma \textit{ such that } m_2(x') = x \end{cases}$$

   (c) Construct the derivations $T_1 \overset{r_1,m_1,\sigma}{\Longleftarrow} S \overset{r_2,m_2,\sigma}{\Longrightarrow} T_2$.
   (d) If the pair of derivations is in conflict, then it is a symbolic critical pair.

Proof. We show that the above construction produces exactly all symbolic critical pairs according to Definition 2. The construction computes only symbolic critical pairs - when Step 4.d is reached, the pair of derivations exists, is minimal and in conflict, and is labelled using one of the substitutions returned by UNIF. The construction computes all critical pairs since all overlaps and all unifiers per overlap are considered. □

The construction of symbolic critical pairs is similar to that of conventional critical pairs. The most important difference occurs when overlapping graph nodes or edges since unification needs to be considered. This process terminates in finite time - overlapping finite graphs produces a finite number of overlaps (Step 1), left-linearity allows for UNIF to produce a finite set of substitutions (Step 3), all other components are either finite checks or constructing a pair of direct derivations. Recall that the number of conventional critical pairs is infinite in general due to the infinite number of rules that a schema represents. Consequently, the computation of symbolic critical pairs is much more suitable for automation as part of a confluence checker.

Corollary 1 (Finiteness of Symbolic Critical Pairs). *For each pair of left-linear rule schemata r_1 and r_2, the set of symbolic critical pairs induced by r_1 and r_2 is finite.*

Proof sketch. Since the above construction computes all critical pairs and terminates, then the set of symbolic critical pairs must be finite. □

6 Completeness of Symbolic Critical Pairs

Completeness of critical pairs means that each pair of conflicting direct derivations is an instance of a symbolic critical pair. Formally, we state the result as a theorem. We start by discussing the link between unification of expressions and completeness of critical pairs.

Critical Pairs and Unification. As shown in Sect. 2.2, one needs to consider critical pairs labelled with expressions rather than concrete values. This means one needs an algorithm to compute their labels which are expressions resulting from overlapping left-hand graphs of rules. If one does not severely restrict the shape of labels, this computation involves unification. However, the type of unification becomes crucial when considering whether the constructed critical pairs are complete, as we show below.

Consider two rule schemata with left-hand sides $\boxed{\texttt{1:x}}$ and $\boxed{\texttt{y:1}}$ where `x` and `y` are `list` variables. Overlapping these graphs induces the unification problem $P = \langle \texttt{1} : \texttt{x} =^? \texttt{y} : \texttt{1}\rangle$. This problem can be syntactically unified via its most general unifier $\sigma = \{\texttt{x}, \texttt{y} \to \texttt{1}\}$. However, the problem has an infinite number of AU-unifiers: $\{\texttt{x}, \texttt{y} \to \texttt{empty}\}$, $\{\texttt{x}, \texttt{y} \to \texttt{1}\}$, $\{\texttt{x}, \texttt{y} \to \texttt{1} : \texttt{1}\}$, $\ldots$, none of which are instances of σ except one. As a consequence, critical pairs labelled using most general unifiers are incomplete. To our knowledge this problem has been first observed in [4, p. 198] in the context of attributed graph transformation. Their solution is to restrict the shape of labels/attributes to variable-free or variable-only terms which avoids the need for unification.

Our solution to this problem involves using our complete AU-unification algorithm - solving P produces the AU-unifiers $\{\texttt{x}, \texttt{y} \to \texttt{empty}\}$ and $\{\texttt{x} \to \texttt{x}' : \texttt{1}, \texttt{y} \to \texttt{1} : \texttt{x}'\}$ where $\texttt{x}'$ is a fresh list variable. (Our algorithm also produces σ making the result set non-minimal.) Consider any assignment α such that $(\texttt{1} : \texttt{x})\alpha =_{\mathrm{AU}} (\texttt{y} : \texttt{1})\alpha$. By Theorem 1, there exists a unifier $\sigma \in \mathrm{UNIF}(P)$ and instantiating substitution λ such that $\alpha = \lambda \circ \sigma$. Thus, a symbolic critical pair labelled using σ can be instantiated via λ to a critical pair of host graph derivations. Consequently, completeness of our AU-unification algorithm allows for greater representational power when it comes to critical pairs.

Restriction Lemma. In the following, we present a restriction construction, formulated only for direct derivations, which is in some sense the inverse of extending a derivation to a larger context. This construction is necessary for the proof of Theorem 3.

Lemma 1 (Restriction). *Given a direct derivation* $G \overset{r,m,\alpha}{\Longrightarrow} H$, *a morphism* $e : P \to G \in \mathcal{N}$, *and a match* $m' : L^\alpha \to P \in \mathcal{N}$ *such that* $m = e \circ m'$, *then there is a direct derivation* $P \overset{r,m',\alpha}{\Longrightarrow} Q$ *leading to the (extension) diagram below.*

$$\begin{tikzcd}
L^\alpha \arrow[d, "m'"] \arrow[dd, bend right, "m"'] \arrow[dr, phantom, "(2)"] & K^\alpha \arrow[l] \arrow[r] \arrow[d] \arrow[dd, bend left] \arrow[dr, phantom, "(3)"] & R^\alpha \arrow[d] \arrow[dd, bend left] \\
P \arrow[d, "e"] \arrow[dr, phantom, "(1)"] & N \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(4)"] & Q \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

Proof. See the long version of this paper [13].

Completeness of symbolic critical pairs. Next we state our Completeness Theorem. The technical aspects of its proof (see [13]) are concerned with the properties of partially labelled graphs $\mathcal{G}_\perp$ and the classes of horizontal and vertical morphisms in direct derivations ($\mathcal{M}$ and $\mathcal{N}$). These basic properties have already been studied in [8,9]. Below we give a proof sketch, including only of the important steps.

Theorem 3. (Completeness of Symbolic Critical Pairs). *For each pair of conflicting rule schema applications $H_1 \overset{r_1,m_1,\alpha}{\Longleftarrow} G \overset{r_2,m_2,\alpha}{\Longrightarrow} H_2$ between left-linear schemata r_1 and r_2 there exists a symbolic critical pair $T_1 \overset{r_1}{\Longleftarrow} S \overset{r_2}{\Longrightarrow} T_2$ with (extension) diagrams between $H_1 \Leftarrow G \Rightarrow H_2$ and an instance of $T_1 \Leftarrow S \Rightarrow T_2$.*

$$\begin{tikzcd}
T_1 \arrow[d, mapsto] & S \arrow[l, Leftarrow] \arrow[r, Rightarrow] \arrow[d, mapsto] & T_2 \arrow[d, mapsto] \\
Q_1 \arrow[d] & P \arrow[l, Leftarrow] \arrow[r, Rightarrow] \arrow[d] & Q_2 \arrow[d] \\
H_1 & G \arrow[l, Leftarrow] \arrow[r, Rightarrow] & H_2
\end{tikzcd}$$

Proof sketch. We start by decomposing the pair of matches $(m_1 : L_1^\alpha \to G, m_2 : L_2^\alpha \to G)$ (Fig. 6) to obtain a graph $P = m_1(L_1^\alpha) \cup m_2(L_2^\alpha) = m_1'(L_1^\alpha) \cup m_2'(L_2^\alpha)$ together with jointly surjective matches $(m_1' : L_1^\alpha \to P, m_2' : L_2^\alpha \to P)$ and morphism $e : P \to G \in \mathcal{N}$.

$$\begin{tikzcd}
R_1^\alpha \arrow[d] \arrow[dd, bend right] \arrow[dr, phantom, "(3)"] & K_1^\alpha \arrow[l] \arrow[r] \arrow[d] \arrow[dd, bend left] \arrow[drr, phantom, "(2)"] & L_1^\alpha \arrow[dr, "m_1'"] \arrow[ddr, bend right, "m_1"'] & & L_2^\alpha \arrow[dl, "m_2'"'] \arrow[ddl, bend left, "m_2"] & K_2^\alpha \arrow[l] \arrow[r] \arrow[d] \arrow[dd, bend right] \arrow[dr, phantom, "(7)"] & R_2^\alpha \arrow[d] \arrow[dd, bend left] \\
Q_1 \arrow[d] \arrow[dr, phantom, "(4)"] & N_1 \arrow[l] \arrow[rr] \arrow[d] \arrow[drr, phantom, "(1)"] & & P \arrow[d, "e"] & & N_2 \arrow[ll] \arrow[r] \arrow[d] \arrow[ul, phantom, "(6)"] \arrow[dll, phantom, "(5)"] \arrow[dr, phantom, "(8)"] & Q_2 \arrow[d] \\
H_1 & D_1 \arrow[l] \arrow[rr] & & G & & D_2 \arrow[ll] \arrow[r] & H_2
\end{tikzcd}$$

Fig. 6. Decomposed pushouts

Next we apply Lemma 1 twice to obtain the restricted derivations $P \Rightarrow Q_1$ and $P \Rightarrow Q_2$. It is not difficult to show that $Q_1 \Leftarrow P \Rightarrow Q_2$ is minimal and in conflict using the commutativity of (1), the properties of (m_1', m_2'), and Definition 1 (e.g. see the proof of Lemma 6.22 in [4]). This concludes the first part of the proof.

For the second part we will show that $Q_1 \Leftarrow P \Rightarrow Q_2$ is an instance of a symbolic critical pair $T_1 \Leftarrow S \Rightarrow T_2$. We use the fact that the assignment α is an AU-unifier for the system of equations $\mathrm{EQ}(L_1, L_2, m_1, m_2)$ and therefore, by Theorem 1 (r_1 and r_2 are left-linear), α is an instance of a unifier $\sigma \in \mathrm{UNIF}(\mathrm{EQ}(L_1, L_2, m_1', m_2'))$ such that $\alpha = \lambda \circ \sigma$ where λ is some assignment.

Next we construct the symbolic critical pair $T_1 \Leftarrow S \Rightarrow T_2$. The graphs have the same node/edge sets as $Q_1 \Leftarrow P \Rightarrow Q_2$ but different labels. First, instantiate L_1 and L_2 via σ to obtain graphs L_1^σ and L_2^σ. Then define $S = m_1'(L_1^\sigma) \cup m_2'(L_2^\sigma)$. This definition is sound because σ is a unifier. It is easy to show that $P \cong S^\lambda$ using $\alpha = \lambda \circ \sigma$.

$$\begin{tikzcd}
R_1 \arrow[dd, mapsto, "\sigma"] & K_1 \arrow[l] \arrow[r] \arrow[dd, mapsto, "\sigma"] & L_1 \arrow[dd, mapsto, "\sigma"] \arrow[dr, mapsto, "\alpha"] & \\
 & & & L_1^\alpha \arrow[ddd, bend left, "m_1'"] \\
R_1^\sigma \arrow[d] & K_1^\sigma \arrow[l] \arrow[r] \arrow[d] \arrow[dl, phantom, "(10)"] \arrow[drr, phantom, "(9)"] & L_1^\sigma \arrow[ur, mapsto, "\lambda"] \arrow[dr] & \\
T_1 \arrow[d, mapsto, "\lambda"] & O_1 \arrow[l] \arrow[rr] \arrow[d, mapsto, "\lambda"] & & S \arrow[d, mapsto, "\lambda"] \\
Q_1 & N_1 \arrow[l] \arrow[rr] & & S^\lambda \cong P
\end{tikzcd}$$

Fig. 7. Construction of $S \Rightarrow T_1$.

We proceed by constructing the derivation $S \overset{r_1, m_1', \sigma}{\Longrightarrow} T_1$ - the double-pushout is $(9+10)$ of Fig. 7 together with the instantiation squares right above it. The same construction is applied to obtain $S \Rightarrow T_2$. By Definition 2, it follows that $T_1 \overset{r_1, m_1', \sigma}{\Longleftarrow} S \overset{r_2, m_2', \sigma}{\Longrightarrow} T_2$ is a symbolic critical pair – (m_1', m_2') are jointly surjective, σ is a unifier, and it can be shown the derivations are in conflict since $Q_1 \Leftarrow P \Rightarrow Q_2$ is in conflict. $\square$

Example of completeness. Consider the pairs of derivations in Figs. 3, 4 and 5. They form the layers of the diagram in Theorem 3. The morphism $e : P \to G$ is an inclusion where G contains the extra edge from node 1 to 3. The assignment λ linking the symbolic critical pair to its instance is $\lambda = \{\mathtt{x1} \to 1, \mathtt{y1} \to 2, \mathtt{y2} \to 3, \mathtt{a1} \to 4, \mathtt{a2} \to 5\}$.

7 Conclusion and Future Work

We have presented the foundations of critical pair analysis for the graph programming language GP 2. Our goal is to develop a static checker that can verify or refute confluence (functional behaviour) for a large class of graph programs. We have introduced *symbolic* critical pairs of GP 2 rule schemata, which are labelled with expressions, and established the completeness and finiteness of the set of symbolic critical pairs over a finite set of rule schemata. We have given a procedure for constructing that set.

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 symbolic critical pairs are strongly joinable. The precise definition of joinability is an interesting problem from an algorithmic point of view, and so is the development of a procedure that determines program confluence by analysing critical pairs. Another interesting topic is the role of SMT solvers for deciding label equivalences and implications in the context of isomorphism checking and joinability analysis.

References

1. Baader, F., Snyder, W.: Unification theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. 2, pp. 445–532. Elsevier and MIT Press, Amsterdam and Cambridge (2001)
2. Bak, C.: GP 2: efficient implementation of a graph programming language. Ph.D. thesis, University of York (2015). http://etheses.whiterose.ac.uk/id/eprint/12586
3. Duffin, R.J.: Topology of series-parallel networks. J. Math. Anal. Appl. **10**(2), 303–318 (1965). https://doi.org/10.1016/0022-247X(65)90125-3
4. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-31188-2
5. Ehrig, H., Golas, U., Habel, A., Lambers, L., Orejas, F.: $\mathcal{M}$-adhesive transformation systems with nested application conditions: part 2: embedding, critical pairs and local confluence. Fundamenta Informaticae **118**(1–2), 35–63 (2012). https://doi.org/10.3233/FI-2012-705
6. Ehrig, H., Prange, U., Taentzer, G.: Fundamental theory for typed attributed graph transformation. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 161–177. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30203-2_13
7. Golas, U., Lambers, L., Ehrig, H., Orejas, F.: Attributed graph transformation with inheritance: efficient conflict detection and local confluence analysis using abstract critical pairs. TCS **424**, 46–68 (2012). https://doi.org/10.1016/j.tcs.2012.01.032
8. Habel, A., Plump, D.: Relabelling in graph transformation. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 135–147. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45832-8_12
9. Habel, A., Plump, D.: $\mathcal{M},\mathcal{N}$-adhesive transformation systems. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2012. LNCS, vol. 7562, pp. 218–233. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33654-6_15
10. Heckel, R., Küster, J.M., Taentzer, G.: Confluence of typed attributed graph transformation systems. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 161–176. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45832-8_14
11. Hristakiev, I., Plump, D.: A unification algorithm for GP 2. In: Graph Computation Models (GCM 2014), Revised Selected Papers. Electronic Communications of the EASST, vol. 71 (2015). http://journal.ub.tu-berlin.de/eceasst/article/view/1002, https://doi.org/10.14279/tuj.eceasst.71.1002
12. Hristakiev, I., Plump, D.: Attributed graph transformation via rule schemata: Church-Rosser theorem. In: Milazzo, P., Varró, D., Wimmer, M. (eds.) STAF 2016 Collocated Workshops, Revised Selected Papers. LNCS, vol. 9946, pp. 145–160. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50230-4_11

13. Hristakiev, I., Plump, D.: Towards critical pair analysis for the graph programming language GP 2 (long version) (2017). https://www.cs.york.ac.uk/plasma/publications/pdf/HristakievPlump.WADT16.Long.pdf
14. Kulcsár, G., Deckwerth, F., Lochau, M., Varró, G., Schürr, A.: Improved conflict detection for graph transformation with attributes. In: Proceedings of Graphs as Models, GaM 2015, EPTCS, vol. 181, pp. 97–112 (2015). https://doi.org/10.4204/EPTCS.181.7
15. Plump, D.: Confluence of graph transformation revisited. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 280–308. Springer, Heidelberg (2005). https://doi.org/10.1007/11601548_16
16. Plump, D.: The design of GP 2. In: Proceedings of International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2011, EPTCS, vol. 82, pp. 1–16 (2012). https://doi.org/10.4204/EPTCS.82.1

Canonical Selection of Colimits

Till Mossakowski[1(✉)], Florian Rabe[2(✉)], and Mihai Codescu[3(✉)]

[1] Otto-von-Guericke-University of Magdeburg, Magdeburg, Germany
till@iws.cs.uni-magdeburg.de
[2] Jacobs University Bremen, Bremen, Germany
f.rabe@jacobs-university.de
[3] Free University of Bozen-Bolzano, Bolzano, Italy
Mihai.Codescu@unibz.it

Abstract. Colimits are a powerful tool for the combination of objects in a category. In the context of modeling and specification, they are used in the institution-independent semantics (1) of instantiations of parameterised specifications (e.g. in the specification language CASL), and (2) of combinations of networks of specifications (in the OMG standardised language DOL).

The problem of using colimits as the semantics of certain language constructs is that they are defined only up to isomorphism. However, the semantics of a complex specification in these languages is given by a signature and a class of models over that signature – not by an isomorphism class of signatures. This is particularly relevant when a specification with colimit semantics is further translated or refined. The user needs to know the symbols of a signature for writing a correct refinement.

Therefore, we study how to usefully choose one representative of the isomorphism class of all colimits of a given diagram. We develop criteria that colimit selections should meet. We work over arbitrary inclusive categories, but start the study how the criteria can be met with $\mathbb{S}et$-like categories, which are often used as signature categories for institutions.

1 Introduction

"Given a species of structure, say widgets, then the result of interconnecting a system of widgets to form a super-widget corresponds to taking the *colimit* of the diagram of widgets in which the morphisms show how they are interconnected." [7]

Motivation. 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 [16] (for instantiations of parameterised specifications) and its extension DOL (see [14,17] and http://dol-omg.org) (for combination of networks of specifications). Specware [24] provides a tool computing colimits of specifications

P. James and M. Roggenbach (Eds.): WADT 2016, LNCS 10644, pp. 170–188, 2017.
https://doi.org/10.1007/978-3-319-72044-9_12

that has been successfully used in industrial applications; [22] makes a strong case for the use of colimits in formal software development. The Heterogeneous Tool Set (HETS, [13]) also supports the computation of colimits, covering even the heterogeneous case [4]. Colimits have been used for ontology alignment [25] and database integration [21]. Recently, colimits have provided the base mechanism for concept creation by blending existing concepts [10]. Moreover, colimits provide the basis for a good behaviour of parameterisation in a specification language [6].[1]

The problem that arises naturally when using colimits is that they are not unique, but only unique up to isomorphism. By contrast, the semantics of a specification involves a specific signature, which must be selected from this isomorphism class. Also, any implementation of colimit computation in a tool must make an according choice of how the colimiting object actually looks, in particular when it comes to the names of its symbols. Otherwise, users have no control over the well-formedness of further specifications built from the colimit: Referring to symbols of the colimit is only possible with knowledge about the actual symbol names appearing in the colimit. To be useful in practice, it is desirable that such a choice appears natural to the user. For example, the names of original symbols should be preserved whenever possible.

Contribution. Our contribution is two-fold. Firstly, we develop a suite of properties that can be used to evaluate and classify different colimit selections. All of these are motivated by the desire that parameterisation and combination of networks enjoy good properties. We show that these properties, although all desirable, cannot be realised at once. Secondly, we give solutions for systematically selecting colimits in various signature categories that provide good trade-offs between these conflicting properties.

Related work. The semantics of CASL [1,12] provides some method for the computation of specific pushouts. However, the chosen institutional framework (institutions with a lot of extra infrastructure) is rather complicated, while we use the much more natural framework of inclusive categories. Moreover, desirable properties of pushouts are only discussed casually. Rabe [18] 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 [18] only provides a partial selection. In the context of Specware, colimits are computed as equivalence classes [22]. The systematic investigation of selected colimits (i.e. beyond pushouts) is new to our knowledge.

Overview. In Sect. 2, we recall some preliminaries as well as language constructs from CASL and DOL that involve colimits in their semantics. Then we develop criteria for elegant colimit selection in Sect. 3. In Sect. 4, we give colimit selections for various categories. We will also see that not every category admits a selection

[1] When we use the term *specification*, our theory applies equally to *ontologies* and *models*, provided these have a formal semantics as theories of some institution.

that satisfies all desirable properties. Therefore, we pursue a second goal in Sect. 5, namely to find useful categories for which we can give particularly elegant selections.

Some proofs have been omitted here. They were available during review. The full version of the paper can be found under https://arxiv.org/abs/1705.09363.

2 Preliminaries

2.1 Categories with Symbols

The large variety of logical languages in use can be captured at an abstract level using the concept of *signature categories.* The objects of such a category are signatures which introduce syntax for the domain of interest, and the signature morphisms capture relations between signatures such as changes of notation, extensions, or translations. For example, signature categories feature heavily in the framework of *institutions* [8], where they are the starting point for abstractly capturing the semantics of logical systems and developing results independently of the specific features of a logical system.

In institutions and similar frameworks, the signature category is abstract, i.e., it is an arbitrary category. In practice, some properties of signature categories have emerged that are satisfied by the over-whelming majority of logical systems, and that are very helpful for establishing generic results.

For colimits, two properties are particularly important:

Definition 1 [3]. *An **inclusive category** consists of a category* $\mathbf{C}$ *with a broad subcategory*[2] *that is a partially ordered class.*

*The morphisms of the broad subcategory are called **inclusions**, and we write* $A \hookrightarrow B$ *if there is an inclusion from* A *to* B*. We denote the (quasi-)category of inclusive categories and inclusion preserving functors by* $\mathbb{IC}at$*.*

In particular, $\mathbb{S}et$ is an inclusive category via the standard inclusions $A \hookrightarrow B$ iff $A \subseteq B$. Arbitrary categories can be recovered by using the identity relation as the partial order.

Definition 2. *An (inclusive)* category with symbols *consists of an (inclusive) category* $\mathbf{C}$ *and an (inclusion-preserving) functor* $|\_| : \mathbf{C} \to \mathbb{S}et$ *We call* $|A|$ *the set of symbols of* A*.*

In particular, $\mathbb{S}et$ is an inclusive category with symbols via $|A| = A$.

The intuition behind these definitions is that very often signatures can be seen as sets of named declarations. Then the subset relation defines the inclusion relation, and the names of the declarations define the set of symbols.

Signature categories are usually such that signatures that differ only in the choice of names are isomorphic. Then a key difficulty about colimits lies in selecting the set of names to be used in the colimit.[3]

[2] That is, with the same objects as $\mathbf{C}$.

[3] For inclusive categories for which the symbol functor uniquely lifts colimits, solving colimit selection for $\mathbb{S}et$ already suffices. However, the inclusive categories studied in Sect. 4.3 typically do not enjoy this property.

2.2 Specification Operators with Colimit Semantics

The power of the abstraction provided by institutions and related systems is best illustrated by the fact that languages like CASL and DOL provide syntax and semantics of specifications *in an arbitrary institution*. This is done by defining operators on specifications and morphisms.

A *basic specification* consists of a signature and a set of sentences[4]—called the *axioms*—over it. A kernel language of specification operators has been introduced in [19]. It includes union, renaming and hiding. CASL and DOL provide many further constructs.

The semantics of many of these operators can be defined as the colimit of a certain diagram. Therefore, such operators are often defined only up to isomorphism. In the sequel we recall important examples from CASL (parameterisation) and DOL (combination of networks).

Parametrisation. Many specification languages, including CASL, allow specifications to be generic. A generic specification consists of a (formal) parameter specification P and a body specification B extending the formal parameter, i.e. $P \hookrightarrow B$. We make P explicit by writing $B[P]$.

A typical example is the specification $List[Elem]$ for lists parametrised by the specification $Elem$ which declares a sort $elem$.

Given an actual parameter specification A and a specification morphism $\sigma : P \to A$, we write the instantiation of $B[P]$ with A via σ as $B[A\mathbf{fit}\sigma]$. Its semantics is given by the pushout on the left below:

$$\begin{tikzcd}
P \arrow[r, hook] \arrow[d, "\sigma"] & B \arrow[d] \\
A \arrow[r, hook] & B[A\ \mathbf{fit}\ \sigma]
\end{tikzcd}
\quad \text{e.g.} \quad
\begin{tikzcd}
Elem \arrow[r, hook] \arrow[d, "\sigma"] & List[Elem] \arrow[d] \\
Nat \arrow[r, hook] & List[Nat\ \mathbf{fit}\ elem \mapsto nat]
\end{tikzcd}$$

The right hand side above gives a typical example where the specification Nat declares a sort nat that is used to instantiate the sort $elem$. Note that the above is also an example of how a pushout of an inclusion can often be selected as an inclusion again. We will get back to that in Definition 12.

A natural requirement is that the instantiated body $B[A\ \mathbf{fit}\ \sigma]$ extends the actual parameter A in much the same way as the body B extends the formal parameter P. For example, a sort *list* introduced in the specification *List* should be kept (and not renamed) within the instantiation $List[Nat\ \mathbf{fit}\ elem \mapsto nat]$. Technically, this means that the semantics should not be an *arbitrary* colimit. Similarly, the user would expect that any symbols declared in the body should appear verbatim in the instantiated body, unless they have been renamed by σ.

[4] It is straightforward but not essential here to make the notion of sentence precise.

Networks of Specifications. In DOL, a network of specifications (called distributed specification in [15]) is a graph. Its nodes are labelled with pairs (O, SP) where SP is a specification and O its name. The edges are theory morphisms $(O_1, SP_1) \xrightarrow{\sigma} (O_2, SP_2)$, either induced by the import structure of the specifications, or by refinements.

A network is specified by giving a list of specifications O_i, morphisms M_i between them and sub-networks N_i, with the intuition that the graph of the network is the union of the graphs of all its elements.

```
network N =
  N_1,...,N_m,
  O_1,...,O_n,
  M_1,...,M_p
```

Now the operator **combine** takes a network and produces the specification given by the colimit of the graph.

Example 1. In the example below, the network N3 consists of the nodes S, T2, and U2 and two automatically added edges, which are the inclusions from S to T2 and U2. Thus, N3 is a span, and **combine N3** yields its pushout. Indeed, both occurrences of sort s from S are identified in the pushout.

In the network N4, we exclude one of the automatically added inclusions. Thus, N4 is a graph with one isolated node for T2 and one inclusion edge from S to U2. **combine N4** yields the disjoint union of T2 and U2. That means that the two occurrences of sort s from S are kept seperate.

```
spec S = sort s end
spec T2 = S then sort t end
spec U2 = S then sort u end
network N3 = S, T2, U2 end
network N4 = N3 excluding S -> T2 end
```

3 Desirable Properties of Colimit Selections

The central definition regarding colimit selection is the following:

Definition 3. *Given a category* **C***, a* selection of colimits *is a partial function* ***sel*** *from* **C***-diagrams D to cocones on D such that* ***sel***(D)*, if defined, is a colimit for D. If* ***sel*** *is only defined for pushouts, we speak of a selection of pushouts, and so on.*

While it is trivial to give *some* selection of colimits (e.g., by using the axiom of choice or by randomly generating names), it turns out that selecting colimits *elegantly* is a non-trivial task. For example, selecting a colimit may require inventing new names, or there can be multiple conflicting strategies for selecting names. [18] conjectures that it is not possible to select pushouts in a way that the selected pushouts are total, coherent, and enjoy natural names.

In this section, we introduce a suite of criteria for colimit selection. We work with an arbitrary inclusive category **C** with symbols.

3.1 Symbols of a Diagram

We are interested in selecting a colimit (C, μ_i) for a diagram $D : \mathbf{I} \to \mathbf{C}$. In most practically relevant signature categories, the construction of a colimit can be reduced to the construction of the colimit in $\mathbb{S}et$ of the corresponding sets of symbols. Because the colimit in $\mathbb{S}et$ amounts to taking a quotient of a disjoint union, we introduce the following auxiliary concept:

Definition 4 (Symbols of a Diagram). *Given a diagram $D : \mathbf{I} \to \mathbf{C}$, we define the set $\mathbf{Sym}(D)$ by*

$$\mathbf{Sym}(D) := \biguplus_{i \in |\mathbf{I}|} |D(i)| := \{(i, x) \mid i \in \mathbf{I}, x \in |D(i)|\}.$$

Moreover, we define the preorder $\leq_D$ on $\mathbf{Sym}(D)$ by

$$(i, x) \ \leq_D \ (j, |D(m)|(x)) \ for\ any\ m : i \to j \in \mathbf{I}.$$

and we define $\sim_D$ to be the least equivalence relation containing $\leq_D$.

Given any colimit (C, μ_i) of D, we embed $\mathbf{Sym}(D)$ into $|C|$ by defining

$$\mu_D(i, x) := |\mu_i|(x)$$

Intuitively, $\mathbf{Sym}(D)$ contains the symbols of all nodes of D. $\sim_D$ defines which symbols must definitely be identified in the colimit:

Proposition 1. *$\sim_D$ is a subset of the kernel of μ_D.*

Proof. This follows from μ being a cocone. □

In some categories such as $\mathbb{S}et$, we even have $\sim_D = \ker(\mu_D)$.

In principle, a natural property to desire of the selected colimit is that $|\mathbf{sel}(D)|$ is a quotient of $\mathbf{Sym}(D)$, in particular $|\mathbf{sel}(D)| = \mathbf{Sym}(D)/\sim_D$ if $\sim_D = \ker(\mu_D)$. However, that is often impractical, e.g., in the typical case where $|\Sigma|$ is intended to be a set of strings that serve as user-friendly names. In particular, we do not want to see the indices $i \in I$ creep into the symbol names in $|\mathbf{sel}(D)|$. Therefore, we define:

Definition 5 (Names and Name-Clashes). *For every equivalence class $X \in \mathbf{Sym}(D)/\sim_D$, let $\boldsymbol{Nam}(X) = \{x | (i, x) \in X\}$.*

We say that D is name-clash-free *if the sets $\boldsymbol{Nam}(X)$ are pairwise disjoint for all X. We say that D is* fully-sharing *if additionally all sets $\boldsymbol{Nam}(X)$ have size* 1.

Intuitively, name-clash-freeness means that whenever two nodes use the same symbol x, the diagram requires these two symbols to be shared in the colimit. An example will be given in Example 2 below. A particularly common special case arises when both nodes import x from the same node. The following makes that precise:

Proposition 2. *Consider a diagram $D : \mathbf{I} \to \mathbf{C}$. Assume that for all $(i, x), (j, x) \in \mathbf{Sym}(D)$ there are $(k, y) \in \mathbf{Sym}(D)$ and $m : k \to i$ and $n : k \to j$ in $\mathbf{I}$ such that $|m|(y) = |n|(y) = x$.*

Then D is name-clash-free. If additionally all edges in D are inclusions, D is fully-sharing.

The value of name-clash-freeness is the following: for the colimit, we can pick symbols that were already present in D. This allows selecting a colimit whose symbol names are inherited from the diagram (and thus already known to the user who requested the colimit). Moreover, if D is fully-sharing, these representatives are uniquely determined.[5]

3.2 Properties of Colimit Selections

Being thus prepared, we can now define a number of desirable properties that make a particular selection **sel** of colimits elegant.

The most obviously desirable property is that we select a colimit whenever we can:

Definition 6 (Completeness). ***sel*** *is complete if it is defined for every diagram that has a colimit.*

Choosing Symbols. Typically, we cannot simply choose $|\mathbf{sel}(D)| = \mathbf{Sym}(D)/\sim_D$ because the choice of symbols is restricted:

Definition 7 (Name-Compliance). *Let* **Symbols** *be some subcategory of* $\mathbb{S}et$. *We call an object Σ* **Symbols***-compliant if $|\Sigma| \in$* **Symbols***. A diagram is* **Symbols***-compliant if all involved objects are.*

sel preserves **Symbols***-compliance if* ***sel***(D) *is* **Symbols***-compliant whenever D is.*

In practical systems, symbols must be chosen from a fixed set S, e.g., the set of alphanumeric strings. In that case, **Symbols** contains all sets that are subsets of S. If we want a compliance-preserving colimit selection, we have to pick names from S—that can be much more difficult to do canonically than to pick arbitrary symbols.

It is easy to select colimits by picking arbitrary symbols, e.g., by generating a fresh string as the name of any new declaration. But that is undesirable—it is preferable that the symbols of **sel**(D) are inherited from D in the following sense:

[5] CASL has a mechanism of "compound identifiers" that ensures name-clash-freeness in multiple instantiations of parametrised specifications, such as *List*[*List*[*Elem*]], see [16], p.47f. and p.224f.

Definition 8 (Natural Names). ***sel*** *has* natural names *if for every name-clash-free diagram D, the selected colimit* $\boldsymbol{sel}(D) = (C, \mu_i)$ *is such that*

– $|C|$ *contains exactly one representative* $r \in \boldsymbol{Nam}(X)$ *for every equivalence class* X*,*
– $|\mu_i|$ *maps every* x *to the respective representative* r*.*

Note that if D is fully-sharing, natural names fully determine $|C|$. For the general case, we have to choose some r for each equivalence class. There are multiple options for making that choice canonical. For example:

Definition 9 (Origin-Based Names). *Let* ***sel*** *have natural names.*

sel *has* origin-*based symbol names if for every class X the chosen representative r is such that there is some i such that (i, r) is minimal in X with respect to $\leq_D$.*

Definition 10 (Majority-Based Names). *Let* ***sel*** *have natural names.*

sel *has* majority-based *symbol names if for every class X the chosen representative x maximizes the cardinality of* $\{i \in I | (i, x) \in X\}$.

Accordingly, ***sel*** *has* majority-origin-based *symbol names if the above cardinality function is used to choose among multiple minimal elements.*

Example 2. Consider a span D consisting of $A \xleftarrow{\alpha} P \xrightarrow{\beta} B$. We consider multiple situations given by the rows of the following table:

	$\lvert P\rvert$	$\lvert A\rvert$	$\lvert B\rvert$	$\lvert\alpha\rvert$	$\lvert\beta\rvert$
1	$\{\}$	$\{x\}$	$\{x\}$		
2	$\{p\}$	$\{a, a'\}$	$\{b, b'\}$	$p \mapsto a$	$p \mapsto b$
3	$\{p, p'\}$	$\{a\}$	$\{p, p'\}$	$p \mapsto a, p' \mapsto a$	$p \mapsto p, p' \mapsto p'$
4	$\{elem\}$	$\{nat, +\}$	$\{elem, list\}$	$elem \mapsto nat$	$elem \mapsto elem$

Depending on the situation, different colimit selections are possible:

1. The diagram is not name-clash-free, and we cannot inherit names.
2. The diagram is name-clash-free but not fully sharing. The sets $\mathbf{Nam}(\_)$ are $\{p, a, b\}$, $\{a'\}$, and $\{b'\}$. Thus, there are three possible colimits that have natural names. All three satisfy the majority condition. The origin condition allows uniquely selecting $|\mathbf{sel}(D)| = \{p, a', b'\}$.
3. The only set $\mathbf{Nam}(\_)$ is $\{p, p', a, p, p'\}$ (where we repeat elements to indicate how often they occur in the corresponding equivalence class). We can have natural names, but neither majority nor origin yield a unique choice.
4. This is a typical case of instantiating a parametric specification (here: lists with a parameter for the type of elements) with an actual parameter (here: the set of natural numbers). The sets $\mathbf{Nam}(-)$ are $\{elem, elem, nat\}$, $\{list\}$, and

$\{+\}$. We can have natural names, and both origin and majority uniquely yield $|\mathbf{sel}(D)| = \{elem, +, list\}$. However, neither is elegant: The desired choice would be $\{nat, +, list\}$.

Pushouts Along Inclusions. Pushouts along inclusions are of particular importance because they provide the semantics of parametrization. As in Sect. 2.2, D is a diagram as given on the right.

$$\begin{tikzcd} P \arrow[r, hook] \arrow[d, "\sigma"] & B \\ A & \end{tikzcd}$$

The following property is motivated by the desire that instantiating parameterised specifications should always be defined:

Definition 11 (Total pushouts). ***sel*** *has* total pushouts *if it is defined for all spans where one arrow is an inclusion.*

Moreover, it is desirable that the instantiation extends A in the same way in which P extends B. The following definitions make this precise:

Definition 12 (Pushout-Stable Inclusions). *Let* ***sel*** *have total pushouts.*

sel *has* pushout-stable *inclusions if the pushout selection preserves the inclusion, i.e.,* ***sel***(D) *is of the form*

$$\begin{tikzcd} P \arrow[r, hook] \arrow[d, "\sigma"] & B \arrow[d, "\sigma^B"] \\ A \arrow[r, hook] & \sigma(B) \end{tikzcd}$$

Definition 13 (Pushout-Stable Names). *Let* ***sel*** *have pushout-stable inclusions.*

sel *has* pushout-stable names *if for every selected pushout*

$$\begin{tikzcd} P \arrow[r, hook] \arrow[d, "\sigma"] & B \arrow[d, "\sigma^B"] \\ A \arrow[r, hook] & \sigma(B) \end{tikzcd} \qquad \begin{tikzcd} {|P|} \arrow[r, hook] \arrow[d, "|\sigma|"] & {|B|} \arrow[d, "|\sigma^B|"] \\ {|A|} \arrow[r, hook] & {|\sigma(B)|} \end{tikzcd}$$

we have $|\sigma(B)| \setminus |A| = |B| \setminus |P|$ *and* $|\sigma^B|$ *is the identity on that set.*

The aim of pushout-stable inclusions is that we can have

- (vertically) $\_^B$ as a functor $(P \downarrow \mathbf{C}) \to (B \downarrow \mathbf{C})$,
- (horizontally) $\sigma(\_)$ as functor $(P \downarrow \mathbf{C}) \to (A \downarrow \mathbf{C})$ mapping extensions of P to extensions of A.

However, in general, the functoriality laws only hold up to isomorphism. Therefore, we want to impose an additional condition, which is adapted from [18]:

Definition 14 (Coherent Pushouts). *Let **sel** have pushout-stable inclusions. Then **sel** has* coherent pushouts *if the following coherence conditions hold:*

1. $id_P(B) = B$ *and* $id_P^B = id_B$,
2. $\sigma(P) = A$ *and* $\sigma^P = \sigma$,
3. $(\sigma_1;\sigma_2)(B) = \sigma_2(\sigma_1(B))$ *and* $(\sigma_1;\sigma_2)^B = \sigma_1^B;\sigma_2^{\sigma_1(B)}$ *and finally*
4. *for* $P \hookrightarrow B_1 \hookrightarrow B_2$, $\sigma(B_2) = \sigma^{B_1}(B_2)$ *and* $\sigma^{B_2} = (\sigma^{B_1})^{B_2}$

where two conditions refer to the following diagrams

$$\begin{tikzcd}
P \arrow[r, hook] \arrow[d, "\sigma_1"'] & B \arrow[d, "\sigma_1^B"'] \arrow[ddr, "(\sigma_1;\sigma_2)^B"] & \\
A \arrow[r, hook] \arrow[d, "\sigma_2"'] & \sigma_1(B) \arrow[d, "\sigma_2^{\sigma_1(B)}"'] & \\
A' \arrow[r, hook] & \sigma_2(\sigma_1(B)) \arrow[r, equal] & (\sigma_1;\sigma_2)(B)
\end{tikzcd}$$

$$\begin{tikzcd}
P \arrow[r, hook] \arrow[d, "\sigma"'] & B_1 \arrow[r, hook] \arrow[d, "\sigma^{B_1}"'] & B_2 \arrow[d, "\sigma^{B_2}"'] \arrow[dr, "(\sigma^{B_1})^{B_2}"] & \\
A \arrow[r, hook] & \sigma(B_1) \arrow[r, hook] & \sigma(B_2) \arrow[r, equal] & \sigma^{B_1}(B_2)
\end{tikzcd}$$

and ensure that pushouts compose vertically and horizontally.

Coherence. The coherence conditions for pushouts can be generalized to arbitrary diagrams. The general idea is that if there are multiple ways to construct a colimit step-by-step, then it should not matter in which order the construction proceeds. Here step-by-step means that we first construct a colimit of a subdiagram of D and then add that colimit to D and construct a colimit of the resulting bigger diagram, and so on.

A formal definition for the general case is rather difficult. The following special case is adapted from [2]:

Definition 15 (Interchange). ***sel*** *has* interchange *if given a name-clash-free diagram* $D : \mathbf{I} \times \mathbf{J} \to \mathbf{C}$ *(seen as a bifunctor) involving inclusions only*

$$\boldsymbol{sel}_{i\in\mathbf{I}}(\boldsymbol{sel}_{j\in\mathbf{J}} D(i,j)) = \boldsymbol{sel}_{j\in\mathbf{J}}(\boldsymbol{sel}_{i\in\mathbf{I}} D(i,j))$$

With an isomorphism instead of equality, this condition always holds.

To state the coherence condition in full generality, we need a few auxiliary definitions:

Definition 16. *Consider a category I with an object i such that every I-object has at most one arrow into i.*

We write $I\backslash i$ for the subcategory of I formed by removing i. We write $I^{\to i}$ for the subcategory of I formed by removing i and all nodes that have no arrow into i. For a diagram $D : I \to \mathbf{C}$, we write $D\backslash i$ and $D^{\to i}$ for the corresponding restrictions of D.

We say that i is a colimit node *of D if $D(i)$ and the set of all morphisms $D(m)$ for I-arrows m into i are a colimit of $D^{\to i}$. If additionally that colimit is equal to **sel**$(D^{\to i})$, we call i a **sel**-colimit node.*

The intuition behind colimit nodes is that they arise by taking a colimit of a subdiagram and can be ignored when forming a colimit of the entire diagram. For example, in the two commuting diagrams of Definition 14, the nodes $\sigma_1(B)$ and $\sigma(B_1)$ are colimit nodes. They arise as the intermediate results of constructing the pushout in two steps. In general, they arise when constructing a colimit step-by-step:

Proposition 3. *Consider a diagram $D : I \to \mathbf{C}$ with a colimit node i. Then D and $D\backslash i$ have the same colimits.*

Proof. For every D-colimit we obtain a $D\backslash i$-cone by removing the injection from i. Vice versa, every $D\backslash i$-colimit (C, μ) can be uniquely extended to a D-cone with the unique factorization $\mu_i : D(i) \to C$ for the colimit $D(i)$.

In both cases, the colimit properties are shown by diagram chase. □

Now we can define that coherence means that we can indeed ignore colimit nodes when selecting a colimit:

Definition 17. ***sel*** *is* coherent *for the diagram D if for every **sel**-colimit node i we have that **sel**(D) and **sel**$(D\backslash i)$ are equal (apart from the former additionally containing the uniquely determined injection μ_i).*

By iterating the coherence property, we can remove or add **sel**-colimit nodes from/to a diagram without affecting the selected colimit.

4 Colimit Selections for Typical Signature Categories

4.1 Sets

As the simplest possible signature category, we consider the category $\mathbb{S}et$ (with standard inclusions and the identity symbol functor).

We first provide a positive result that gives a large sets of desirable properties that can be realised at once:

Theorem 1. *$\mathbb{S}et$ has a selection of colimits that has completeness, pushout-stable inclusions, total pushouts, interchange, and majority-origin-based names.*

Moreover, for name-clash-free diagrams, this selection has natural names, pushout-stable names, coherent pushouts.

Proof. (Sketch) If name-clash-freeness is satisfied and the diagram consists of inclusions only, just take the union as colimit, which ensures that interchange holds.

Given a span $B \overset{\iota}{\hookleftarrow} P \xrightarrow{\sigma} A$ with σ not an inclusion, let $\sigma(B) := A \cup (B\backslash A) \cup B'$, where $\kappa : B' \cong (B \cap A)\backslash P$ such that $B' \cap (A \cup (B\backslash A)) = \emptyset$. Define

$$\begin{tikzcd}
P \arrow[r, hook, "\iota"] \arrow[d, "\sigma"] & B = P \cup (B\backslash P) \arrow[d, "\sigma^B = \sigma \cup \theta"] \\
A \arrow[r, hook] & \sigma(B) = A \cup ((B\backslash(P \cup A)) \cup B')
\end{tikzcd}$$

where $\theta : B\backslash P \to (B \setminus (P \cup A)) \cup B'$ is given by

$$\theta(x) = \begin{cases} \kappa^{-1}(x), & \text{if } x \in (B \cap A)\backslash P \\ x, & \text{if } x \in B\backslash(P \cup A) \end{cases}.$$

If D is a name-clash-free diagram not of the above forms, define its colimit $(C, (\mu_i)_{i\in|I|})$ as follows. C is defined by selecting from each equivalence class $X \in \mathbf{Sym}(D)/\sim_D$ a representative $r(X) \in \mathbf{Nam}(X)$ and for each index i, and each $(i, x) \in X$, we define $\mu_i(x) = r(X)$. Use the majority-origin principle. If that does not determine a representative, select one of the candidates randomly.

Finally, for an arbitrary non name-clash-free diagram, select an arbitrary colimit, ensuring completeness. □

Second, we provide a negative result that gives a small set of desirable properties that cannot be realised at once:

Theorem 2. *$\mathbb{S}et$ does not have a selection that has total pushouts, pushout-stable inclusions and names, and coherent pushouts.*

Theorem 1 shows that in $\mathbb{S}et$, we can realise several criteria for colimit selection we have defined so far.

Regarding the choice of names in Theorem 1, we cannot expect to achieve origin-based and majority-based names. In fact, one can show that pushout-stable inclusions and names contradict the origin and majority properties. Moreover, it is evident that origin and majority can contradict each other. Consider e.g.

$$\begin{tikzcd}
\{a\} \arrow[r] \arrow[d] & \{b\} \arrow[d] \\
\{b\} \arrow[r] & \{x\}
\end{tikzcd}$$

Origin would lead to $x = a$, while majority would lead to $x = b$.

Nevertheless, the property of origin-majority-based names is useful to guide the pushout selection in cases where the other properties do not determine names uniquely.

4.2 Product Categories

Signatures of many logical systems of practical interest are often tuples of sets of symbols of different kind. For example, OWL signatures consist of sets of atomic classes, individuals, object and data properties. To be able to transfer the selection of colimits and its properties defined for $\mathbb{S}et$ to categories of tuples of sets, we make use of a more general result that ensures that the selection of colimits and its properties are stable under products.

Theorem 3. *Let $(\mathbf{C}_j)_{j\in J}$ be a family of inclusive categories with symbols and assume selections of colimits sel_j that have the properties in Theorem 1 or Theorem 1. Then the product $\Pi_{j\in J} C_j$ can be canonically turned into an inclusive category with symbols that also has a selection of colimits* ***sel*** *with the same properties.*

Example 3. In the case of multi-sorted logics with function or predicate symbols, we can define a selection function for colimits in a step-wise manner. Let us consider the case of multi-sorted equational logic, that we denote EQL. If we fix a set of sorts S, let $\mathbf{Sign}_S^{EQL}$ be the category of multi-sorted algebraic signatures with sort set S. We can express it as

$$\mathbf{Sign}_S^{EQL} = \Pi_{w\in S^*, s\in S}\ \mathbb{S}et.$$

Objects of this category provide a set of operation symbols $F_{w,s}$ for each string of argument sorts w and result sort s. With the canonical lifting of the symbol functors of the factors (all of which are the identity on $\mathbb{S}et$) to this product, we obtain the symbol functor on $\mathbf{Sign}_S^{EQL}$ given by $|\_| = \biguplus_{i\in J} |\pi_j(\_)|$, which decorates each operation symbol with argument and result sorts. We write $f : w \to s \in |F|$ instead of $((w,s), f) \in |F|$.

4.3 Split Fibrations

Theorem 3 gives us a selection of colimits for $\mathbf{Sign}_S^{EQL}$. However, our overall goal is to provide such a selection for $\mathbf{Sign}^{EQL}$. Now $\mathbf{Sign}^{EQL}$ is a split fibration $\mathbf{Sign}^{EQL} \to \mathbb{S}et$, with fibres $\mathbf{Sign}_S^{EQL}$. It is well-known that a split fibration can be obtained as Grothendieck construction (flattening) of an indexed category indexing the fibres. Hence, we will construct such an indexed category for EQL. This is achieved by observing that each function $u : S \to S'$ leads to a functor $B_u : \mathbf{Sign}_{S'}^{EQL} \to \mathbf{Sign}_S^{EQL}$ defined as $B_u(F') = F$, where $F_{w,s} = F'_{u(w),u(s)}$. This functor has a left adjoint denoted $L_u : \mathbf{Sign}_S^{EQL} \to \mathbf{Sign}_{S'}^{EQL}$ defined as $L_u(F) = F'$, where $F'_{w',s'} = \uplus_{w\in S^*, s\in S, u(w)=w', u(s)=s'} F_{w,s}$.

We thus obtain an indexed inclusive category $B : \mathbb{S}et^{op} \to \mathbb{IC}at$, and it suffices to show that the selection of colimits and its properties are stable under the Grothendieck construction (flattening, see [23]).

Theorem 4. *Let $B : \mathbf{Ind}^{op} \to \mathbb{IC}at$ be an indexed inclusive category (where* **Ind** *is inclusive itself) such that*

- *B is* locally reversible*, i.e. for each $u : i \to j$ in* **Ind***, $B_u : B_j \to B_i$ has a selected left adjoint $F_u : B_i \to B_j$ (note that we do not require coherence of the F_u),*
- **Ind** *has a selection of colimits $\boldsymbol{sel}_{\mathbf{Ind}}$,*
- *each category B_i has a selection of colimits $\boldsymbol{sel}^i$, for $i \in |\mathbf{Ind}|$.*

Then the Grothendieck category $B^\#$ is itself an inclusive category.[6]

[6] Note that this construction extends to institutions, yielding Grothendieck institutions, see [5].

Theorem 5. *Under the assumptions of Theorem 4, let* $(|\_|\theta) : B \to \mathbb{I}ndSet$ *be a (faithful inclusive) oplax indexed functor (where* $\mathbb{I}ndSet : \mathbf{Ind}^{op} \to \mathbb{IC}at$ *is the constant functor delivering* $\mathbb{S}et$*).*

This amounts to, for each B_i*, a (faithful inclusive) symbol functor* $|\_|_i : B_i \to \mathbb{S}et$*, and for each* $u : i \to j$*,* $\theta_u : B_u; |\_|_i \to |\_|_j$ *a natural transformation, such that the* θ_u *are coherent.*

Then $B^{\#}$ *can be equipped with a symbol functor as well.*

Proof. Define $|(i, A_i)| = |i| \uplus |A_i|_i$, and $|(u : i \to j, \sigma)| = |u| \uplus (|\sigma|_i; (\theta_u)_{A_j})$. □

Theorem 6. *Under the assumptions of Theorems 4 and 5, extended by:*

- F_u *preserves inclusions, and moreover,*
- *the unit and counit of the adjunction are inclusions.*

If **Ind** *and each* B_i *have colimit selections enjoying the properties of Theorem 1, then so does* $B^{\#}$*.*

We can apply Theorem 6 to $B : \mathbb{S}et^{op} \to \mathbb{IC}at$ as defined above to obtain a selection of colimits $\mathbf{sel}^{EQL}$ for EQL signatures. By the theorem, $\mathbf{sel}^{EQL}$ has the properties in Theorem 1.

Example 4. We apply these result to EQL, where $B_S = \mathbf{Sign}_S^{EQL}$, using the symbol functors $|\_|_S : \mathbf{Sign}_S^{EQL} \to \mathbb{S}et$ $(S \in |\mathbb{S}et|)$ defined above. Given $u : S \to S'$, $\theta_u : B_u; |\_|_S \to |\_|_{S'}$ is defined as $(\theta_u)_{F'} : |B_u(F')| \to |F'|$, acting as $(\theta_u)_{F'}(f : u(w) \to u(s)) = f : w \to s$. Using Theorem 5, we obtain the usual symbol functor for many-sorted signatures, which for any signature delivers the set of sorts plus the set of typed function symbols of form $f : w \to s$.

Again, the symbol selection principles of Theorem 1 carry over.

5 Categories for Improved Colimit Selection

5.1 Named Specifications

An important technique for avoiding name clashes is to use bipartite IRIs as symbols. IRIs are Internationalized Resource Identifiers for identification per IETF/RFC 3987:2005. Symbols using bipartite IRIs consist of

- **namespace**: an IRI that identifies the containing specification, usually ending with `/`[7]
- **local name**: a name (not containing `/`) that identifies a non-logical symbol within a specification.

Let **IRI** be the subcategory of $\mathbb{S}et$ containing only the sets of bipartite IRIs.

[7] In some languages, `#` is used instead of/. But this has the disadvantage that, when used as an IRL, the fragment following the `#` is not transmitted to servers.

For most practical purposes, it is acceptable to restrict attention to **IRI**-compliant signatures. For example, DOL (in accordance with many other languages) strongly recommends using bipartite IRIs.

Note that in an **IRI**-compliant signature Σ, the symbols in $|\Sigma|$ may have different namespaces. For example, in DOL, namespaces M serve as the identifiers of basic specifications Σ, and then symbols in $|\Sigma|$ are of the form M/sym. But when a specification N imports M, (see Sect. 2.2), the namespace M of the imported symbols is retained and only new symbols declared in N use the namespace N.

The main advantage of using IRIs is that specifications (and thus the symbols in them) have globally unique names [9]. That makes name clashes much less common:

Proposition 4. *Consider a set of basic signatures with pairwise different namespaces. Then diagrams generated by networks consisting only of* **IRI**-*compliant basic specifications and imports are fully-sharing.*

Proof. Because basic specifications have unique identifiers, the result follows immediately from Proposition 2. □

In practice, the assumptions of Proposition 4 quite often hold, because networks to be combined often consist of import links only.

Proposition 5. *Consider* $\mathbb{S}et$ *with standard inclusions and the identity symbol functor. The selection constructed in Theorem 1 can be modified to a selection that additionally preserves* **IRI**-*compliance.*

Proof. We just need to ensure that new symbols in the colimit are of the form N/sym for some fresh namespace N. □

However, generating fresh namespaces interacts poorly with coherence.

5.2 Structured Symbol Names

There are essentially two problems when trying to select colimits canonically: name clashes and ambiguous names. Intuitively, name clashes arise if we have one name for multiple symbols. And ambiguity arises if we have multiple names for one symbol. If neither is the case, named specifications are usually sufficient to obtain canonical colimits.

Our goal now is to handle name clashes and ambiguity. We introduce a subcategory **Vocab** of $\mathbb{S}et$ and focus on **Vocab**-compliance-preserving colimits. We want to pick **Vocab** in such a way that we can select canonical colimits elegantly.

To motivate the following definition of **Vocab**, let us look again at the causes behind name clashes and ambiguity. Name clashes arise if the same node name occurs multiple times in a diagram. For example, consider two nodes i and j (without any arrows) and $|D(i)| = \{a\}$ and $|D(j)| = \{a\}$. (This occurs, for example, when taking the disjoint union of the set $\{a\}$ with itself.) Because this diagram is not name-clash-free, we cannot have natural names in the colimit. Our solution below introduces qualifiers that create two copies p/a and q/a of the clashing name a.

Ambiguity arises if a diagram contains a non-inclusion arrow. For example, consider $m : i \to j$, and $|D(i)| = \{a\}$ and $|D(j)| = \{b\}$ and $|D(m)|(a) = b$. $\sim_D$ has one equivalence class, which contains (i, a) and (j, b). In Sect. 3.2, we focused on choosing either a or b as a natural name in the colimit. Our solution here retains both names and chooses the set $\{a, b\}$ as a symbol in the colimit.

Because colimits can be iterated, **Vocab** must allow for any combination of those two constructions. That yields the following definition:

Definition 18 (Structured Symbols). *We assume a fixed set* **Name** *of strings (which we call* ***names****).*

We write **QualName** *for the set of lists of names (which we call* ***qualified names****). We assume $Name \subset QualName$, and we write* **nil** *for the empty list and p/q for the concatenation of lists.*

A ***structured symbol*** *is a set of qualified names.*

A ***vocabulary*** *V is a set of pairwise disjoint structured symbols. We write $\overline{V}$ for $\bigcup_{S \in V} S$, and for every $s \in \overline{V}$ we write $[s]$ for the unique $S \in V$ such that $s \in S$.*

We write **Vocab** *for the full subcategory of $\mathbb{S}et$ containing only the vocabularies.*

The operation $[s]$ is crucial: It allows us to use any $s \in S \in V$ as a representative for S. Thus, in order to use structured symbols, we do not have to change our external (human-facing) syntax: Users can still write and read s. We only have to change our internal (machine-facing) syntax by maintaining the set S.

Vocab is an inclusive category with symbols (using the same symbol functor as for $\mathbb{S}et$). As we see below, it allows for good colimit selections. But the symbols used in the symbol functor cannot be strings anymore: they are sets of lists of strings.

Above we left open the question where the qualifiers come from that we use to disambiguate name clashes. It would not be acceptable to use the indices from I as qualifiers because they are arbitrary and not visible to the user. Instead, we assume that the user has provided qualifiers by assigning labels to some nodes in the diagram:

Definition 19 (Labeled Diagram). *A labeled* **C***-diagram (D, L) consists of a diagram $D : \mathbf{I} \to \mathbf{C}$ and a function L from* **I***-objects to* $\mathbf{Name} \cup \{\mathbf{nil}\}$.

L can be a partial function because we only need to label those nodes that are involved in name clashes. However, it is more convenient to make L a total function by assuming that all unlabeled nodes are labeled with the empty list **nil**.

Similar to Definition 4, we define the symbols of a labeled diagram:

Definition 20 (Symbols of a Labeled Diagram). *Let* $(D : \mathbf{I} \to \mathbf{Vocab}, L)$ *be a labeled diagram over* **Vocab**. *We define:*

$$\mathbf{Sym}(D, L) = \{(i, L(i)/x) \mid i \in \mathbf{I}, x \in \overline{D(i)}\}$$

$(i, L(i)/x) \leq_{DL} (j, L(j)/y)$ *if for some* $m : i \to j \in \mathbf{I}, D(m)(S) = T, x \in S, y \in T$

$\sim_{DL}$ *is the equivalence relation on* $\mathbf{Sym}(D, L)$ *generated by* $\leq_{DL}$.

For every $X \in \mathbf{Sym}(D, L)/ \sim_{DL}$, *let* $\boldsymbol{Nam}(X) = \{q \mid (i, q) \in X\}$. *We say that* (D, L) *is* name-clash-free *if the sets* $\boldsymbol{Nam}(X)$ *are pairwise disjoint.*

Every plain diagram can be seen as a labeled diagram by using $L(i) = \mathbf{nil}$ for all i. In that case, the definition of name-clash-free of Definition 20 coincides with the one from Definition 5.

We can now see the power of structured symbols by giving a selection of colimits in **Vocab**:

Theorem 7 (Colimits of Vocabularies). *Let* (D, L) *be a name-clash-free labeled diagram. Then*

– *the set* $\boldsymbol{sel}(D, L)$ *defined by* $\{\boldsymbol{Nam}(X) \mid X \in \mathbf{Sym}(D, L)/ \sim_{DL}\}$ *is a vocabulary,*
– *the maps* $\mu_i : D(i) \to \boldsymbol{sel}(D, L)$ *defined by* $\mu_i([x]) = [L(i)/x]$ *are well-defined.*

Then $\boldsymbol{sel}(D, L)$ *and the* μ_i *form a colimit of* D.

sel does not exactly have the desirable properties described in Sect. 3.2. But it has variants of them, which is why we recommend **sel** as a good trade-off:

– **sel** is complete in the sense that labels can be added to any diagram to obtain name-clash-freeness.
– **sel** reduces to union for name-clash-free unlabeled diagrams of inclusions (and therefore satisfies interchange).
– **sel** has pushout-stable inclusions for name-clash-free unlabeled diagrams $A \overset{\alpha}{\leftarrow} P \hookrightarrow B$ in the sense that all qualified names of A are mapped to themselves in the selected pushout.
– **sel** has natural names in the sense that $L(i)/x$ can be used to identify the corresponding symbol in the colimit, and every symbol in the colimit is of that form.
– **sel** is coherent for all labeled diagrams in which all **sel**-colimit nodes are unlabeled.

6 Conclusion

We have provided some useful principles for colimit selection, and studied how far these principles can be actually realised. Some principles contradict each other, so they need to be prioritised. The overall goal is to give the user as much control and predictability over names as possible. This is particularly important for languages such as CASL and DOL, providing powerful constructs for both parameterisation and combination of networks, realised through colimits. We have shown that our results are stable under products and Grothendieck constructions; hence they carry over to more complex signature categories like those of many-sorted logics, HasCASL [20] (without subsorts) or even categories of heterogeneous specification (which usually are also obtained via a Grothendieck construction).

While we have worked with $\mathbb{S}et$ and $\mathbb{S}et$-like categories, future work should extend the results to more complex categories. E.g. the signature category of the subsorted CASL logic cannot be obtained from $\mathbb{S}et$ by products and indexing; instead some quotient construction is needed [11]. Another open question is whether coherence for pushouts can usefully be generalised to other types of colimit. Moreover, it also will be useful to investigate further the pros anc cons of the different selection techniques (exploitation of name-clash-freeness versus labeled diagrams and structured symbols) that we have discussed.

One important motivation for this work has been the need to obtain a better theory for the implementation of colimits in Hets. Currently, the implementation follows the majority principle only, which led to complaints from the user community, especially from the Coinvent project using colimits for conceptual blending. In the future, this will be revised according to the results of this paper.

References

1. Baumeister, H., Cerioli, M., Haxthausen, A., Mossakowski, T., Mosses, P.D., Sannella, D., Tarlecki, A.: CASL Semantics. In: Mosses, P.D. (ed.) CASL Reference Manual, Part III. LNCS, vol. 2960. Springer, London (2004). Edited by Sannella, D., Tarlecki, A
2. Borceux, F.: Handbook of Categorical Algebra I - III. Cambridge University Press, Cambridge (1994)
3. Căzănescu, V.E., Rosu, G.: Weak inclusion systems. Math. Struct. Comput. Sci. **7**(2), 195–206 (1997)
4. Codescu, M., Mossakowski, T.: Heterogeneous colimits. In: Boulanger, F., Gaston, C., Schobbens, P.-Y. (ed.) MoVaH 2008 Workshop. IEEE Press (2008)
5. Diaconescu, R.: Institution-Independent Model Theory. Birkhäuser, Basel (2008)
6. Diaconescu, R., Goguen, J.A., Stefaneas, P.: Logical support for modularisation. In: 2nd Workshop on Logical Environments, pp. 83–130. CUP, New York (1993)
7. Goguen, J.A.: A categorical manifesto. Math. Struct. Comput. Sci. **1**, 49–67 (1991)
8. Goguen, J.A., Burstall, R.M.: Institutions: abstract model theory for specification and programming. J. ACM **39**, 95–146 (1992)
9. W3C SEmantic Web Deployment Working Group. Best practice recipes for publishing RDF vocabularies. W3C Working Group Note, 28 August 2008. http://www.w3.org/TR/2008/NOTE-swbp-vocab-pub-20080828/
10. Kutz, O., Bateman, J., Mossakowski, T., Neuhaus, F., Bhatt, M.: E pluribus unum - formalisation, use-cases, and computational support for conceptual blending. In: Besold, T.R., Schorlemmer, M., Smaill, A. (eds.) Computational Creativity Research: Towards Creative Machines. Atlantis Thinking Machines, vol. 7, pp. 167–196. Atlantis Press (2015)

11. Mossakowski, T.: Colimits of order-sorted specifications. In: Presicce, F.P. (ed.) WADT 1997. LNCS, vol. 1376, pp. 316–332. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-64299-4_42
12. Mossakowski, T.: Specifications in an arbitrary institution with symbols. In: Bert, D., Choppy, C., Mosses, P.D. (eds.) WADT 1999. LNCS, vol. 1827, pp. 252–270. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-540-44616-3_15
13. Mossakowski, T., Maeder, C., Lüttich, K.: The heterogeneous tool set, Hets. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_40
14. Mossakowski, T., Kutz, O., Codescu, M., Lange, C.: The distributed ontology, modeling and specification language. In: Del Vescovo, C., Hahmann, T., Pearce, D., Walther, D. (eds) WoMo 2013, CEUR-WS Online Proceedings, vol. 1081 (2013)
15. Mossakowski, T., Tarlecki, A.: Heterogeneous logical environments for distributed specifications. In: Corradini, A., Montanari, U. (eds.) WADT 2008. LNCS, vol. 5486, pp. 266–289. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03429-9_18
16. Mosses, P.D. (ed.): Casl Reference Manual. LNCS, vol. 2960. Springer, Heidelberg (2004). https://doi.org/10.1007/b96103
17. Object Management Group: The distributed ontology, modeling, and specification language (DOL) 2015. OMG draft standard: http://www.omg.org/spec/DOL/
18. Rabe, F.: How to Identify, Translate, and Combine Logics? J. Logic Comput. (2014). https://doi.org/10.1093/logcom/exu079
19. Sannella, D., Tarlecki, A.: Specifications in an arbitrary institution. Inf. Comput. **76**, 165–210 (1988)
20. Schröder, L., Mossakowski, T.: Hascasl: integrated higher-order specification and program development. Theor. Comput. Sci. **410**(12–13), 1217–1260 (2009)
21. Schultz, P., Spivak, D.I., Vasilakopoulou, C., Wisnesky, R.: Algebraic databases. CoRR, abs/1602.03501 (2016)
22. Smith, D.R.: Composition by colimit and formal software development. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 317–332. Springer, Heidelberg (2006). https://doi.org/10.1007/11780274_17
23. Tarlecki, A., Burstall, R.M., Goguen, J.A.: Some fundamental algebraic tools for the semantics of computation: Part 3: indexed categories. Theor. Comput. Sci. **91**(2), 239–264 (1991)
24. Williamson, K.E., Healy, M., Barker, R.A.: Industrial applications of software synthesis via category theory-case studies using specware. Autom. Softw. Eng. **8**(1), 7–30 (2001)
25. Zimmermann, A., Krötzsch, M., Euzenat, J., Hitzler, P.: Formalizing ontology alignment and its operations with category theory. In: Proceedings of FOIS-2006, pp. 277–288 (2006)

Formalizing and Validating the P-Store Replicated Data Store in Maude*

Peter Csaba Ölveczky[1,2](✉)

[1] University of Oslo, Oslo, Norway
peterol@ifi.uio.no
[2] University of Illinois, Urbana-Champaign, USA

Abstract. P-Store is a well-known partially replicated transactional data store that combines wide-area replication, data partition, some fault tolerance, serializability, and limited use of atomic multicast. In addition, a number of recent data store designs can be seen as extensions of P-Store. This paper describes the formalization and formal analysis of P-Store using the rewriting logic framework Maude. As part of this work, this paper specifies group communication commitment and defines an abstract Maude model of atomic multicast, both of which are key building blocks in many data store designs. Maude model checking analysis uncovered a non-trivial error in P-Store; this paper also formalizes a correction of P-Store whose analysis did not uncover any flaw.

1 Introduction

Large cloud 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 huge amounts of (potentially important user) 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.

Designing and validating cloud storage systems are hard, as the design must take into account wide-area asynchronous communication, concurrency, and fault tolerance. The use of formal methods during the design and validation of cloud storage systems has therefore been advocated recently [9,11]. In [9], engineers at the world's largest cloud computing provider, Amazon Web Services, describe the use of TLA+ during the development of key parts of Amazon's cloud infrastructure, and conclude that the use of formal methods at Amazon has been a success. They report, for example, that: (i) "formal methods find bugs in system designs that cannot be found though any other technique we know of"; (ii) "formal

*This work was partially supported by AFOSR/AFRL Grant FA8750-11-2-0084 and NSF Grant CNS 14-09416.

P. James and M. Roggenbach (Eds.): WADT 2016, LNCS 10644, pp. 189–207, 2017.
https://doi.org/10.1007/978-3-319-72044-9_13

methods [...] give good return on investment"; (iii) "formal methods are routinely applied to the design of complex real-world software, including public cloud services"; (iv) formal methods can analyze "extremely rare" combinations of events, which the engineer cannot do, as "there are too many scenarios to imagine"; and (v) formal methods allowed Amazon to "devise aggressive optimizations to complex algorithms without sacrificing quality."

This paper describes the application of the rewriting-logic-based Maude language and tool [3] to formally specify and analyze the P-Store data store [14]. P-Store is a well-known partially replicated transactional 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).

Members of the University of Illinois Center for Assured Cloud Computing have used Maude to formally specify and analyze complex industrial cloud storage systems such as Google's Megastore and Apache Cassandra [4,8]. Why is formalizing and analyzing P-Store interesting? First, P-Store is a well-known data store design in its own right with many good properties that combines wide-area replication, data partition, some fault tolerance, serializability, and limited use of atomic multicast. Second, a number of recent data store designs can be seen as extensions and variations of P-Store [1,2,15]. Third, it uses atomic multicast to order concurrent transactions. Fourth, it uses "group communication" for atomic commit. The point is that both atomic multicast and group communication commit are key building blocks in cloud storage systems (see, e.g., [2]) that have not been formalized in previous work. Indeed, one of the main contributions of this paper is an abstract Maude model of atomic multicast that allows any possible ordering of message reception consistent with atomic multicast.

I have modeled (both versions of) P-Store, and performed model checking analysis on small system configurations. Maude analysis uncovered some significant errors in the supposedly-verified P-Store algorithm, like read-only transactions never getting validated in certain cases. An author of the original P-Store paper [14] 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 find any error. I also found that a key assumption was missing from the paper, and that an important definition was very easy to misunderstand because of how it was phrased in English. All this emphasizes the need for a formal specification and formal analysis in addition to the standard prose-and-pseudo-code descriptions and informal correctness proofs.

The rest of the paper is organized as follows. Section 2 gives a background on Maude. Section 3 defines an abstract Maude model of the atomic multicast "communication primitive." Section 4 gives an overview of P-Store. Sections 5 and 6 present the Maude model and the Maude analysis, respectively, of P-Store, and Section 7 describes a corrected version of P-Store. Section 8 discusses some related work, and Section 9 gives some concluding remarks.

Due to space limitations, only parts of the specifications and analyses are given. I refer to the longer report [10] for more details. Furthermore, the executable Maude specifications of P-Store, together with analysis commands, are available at http://folk.uio.no/peterol/WADT16.

2 Preliminaries: Maude

Maude [3] is a rewriting-logic-based formal language and simulation and model checking tool. A Maude module specifies a *rewrite theory* $(\Sigma, E \cup A, R)$, where:

- Σ is an algebraic *signature*; that is, a set of declarations of *sorts*, *subsorts*, and *function symbols*.
- $(\Sigma, E \cup A)$ is a *membership equational logic theory*, with E a set of possibly conditional equations and membership axioms, and A a set of equational axioms such as associativity, commutativity, and identity. The theory $(\Sigma, E \cup A)$ specifies the system's state space as an algebraic data type.
- R is a set of *labeled conditional rewrite rules*[1] $l : t \longrightarrow t'$ **if** $\bigwedge_{j=1}^{m} u_j = v_j$ specifying the system's local transitions. The rules are universally quantified by the variables in the terms, and are applied *modulo* the equations $E \cup A$.[2]

I briefly summarize the syntax of Maude and refer to [3] for more details. Operators are introduced with the `op` keyword: `op` f `:` $s_1 \ldots s_n$ `->` s. They can have user-definable syntax, with underbars '_' marking the argument positions, and equational *attributes*, such as `assoc`, `comm`, and `id`, stating, for example, that the operator is associative and commutative and has a certain identity element. Equations and rewrite rules are introduced with, respectively, keywords `eq`, or `ceq` for conditional equations, and `rl` and `crl`. The mathematical variables in such statements are declared with the keywords `var` and `vars`, or can be introduced on the fly having the form *var* : *sort*. An equation $f(t_1, \ldots, t_n) = t$ with the `owise` ("otherwise") attribute can be applied to a term $f(\ldots)$ only if no other equation with left-hand side $f(u_1, \ldots, u_n)$ can be applied. A *class* declaration

`class` C `|` att_1 `:` s_1`,` `...` `,` att_n `:` s_n `.`

declares a class C with attributes att_1 to att_n of sorts s_1 to s_n. An *object* of class C is represented as a term `<` $O : C \mid att_1 : val_1, ..., att_n : val_n$ `>` of sort `Object`, where O, of sort `Oid`, is the object's *identifier*, and where val_1 to val_n are the current values of the attributes att_1 to att_n. A *message* is a term of sort `Msg`.

The state is a term of the sort `Configuration`, and is a *multiset* made up of objects and messages. Multiset union for configurations is denoted by a juxtaposition operator (empty syntax) that is declared associative and commutative, so that rewriting is *multiset rewriting* supported directly in Maude.

The dynamic behavior of concurrent object systems is axiomatized by specifying each of its transition patterns by a rewrite rule. For example, the rule

[1] An equational condition $u_i = w_i$ can also be a *matching equation*, written $u_i := w_i$, which instantiates the variables in u_i to the values that make $u_i = w_i$ hold, if any.

[2] Operationally, a term is reduced to its E-normal form modulo A before a rewrite rule is applied.

```
rl [l] :  m(O,w)
          < O : C | a1 : x, a2 : O', a3 : z >
         =>
          < O : C | a1 : x + w, a2 : O', a3 : z >
          m'(O',x) .
```

defines a family of transitions in which a message `m`, with parameters `O` and `w`, is read and consumed by an object `O` of class `C`, the attribute `a1` of the object `O` is changed to `x + w`, and a new message `m'(O',x)` is generated. Attributes whose values do not change and do not affect the next state of other attributes or messages, such as `a3`, need not be mentioned in a rule. Likewise, attributes that are unchanged, such as `a2`, can be omitted from right-hand sides of rules.

A *subclass* inherits all the attributes and rules of its superclasses.

Formal Analysis in Maude. A Maude module is executable under some conditions, such as the equations being confluent and terminating, modulo the structural axioms, and the theory being coherent [3]. Maude provides a range of analysis methods, including simulation for prototyping, search for reachability analysis, and LTL model checking. This paper uses Maude's *search* command

(search [[n]] t_0 =>* *pattern* [such that *cond*] .)

which uses a breadth-first strategy to search for at most n states that are reachable from the initial state t_0, match the pattern *pattern* (a term with variables), and satisfy the (optional) condition *cond*. If '[n]' is omitted, then Maude searches for all solutions. If the arrow '`=>!`' is used instead of '`=>*`', then Maude searches for *final* states; i.e., states that cannot be further rewritten.

3 Atomic Multicast in Maude

Messages that are *atomically multicast* from (possibly) different nodes in a distributed system must be read in (pairwise) the same order: if nodes n_3 and n_4 both receive the atomically multicast messages m_1 and m_2, they must receive (more precisely: "be served") m_1 and m_2 in the same order. Note that m_2 may be read before m_1 even if m_2 is atomically multicast *after* m_1.

Atomic multicast is typically used to order events in a distributed system. In distributed data stores like P-Store, atomic multicast is used to order (possibly conflicting) concurrent transactions: When a node has finished its local execution of a transaction, it atomically multicasts a validation request to other nodes (to check whether the transaction can commit). The validation requests therefore impose an order on concurrent transactions.

Atomic multicast does not necessarily provide a global order of all events. If each of the messages m_1, m_2, and m_3 is atomically multicast to two of the receivers A, B, and C, then A can read m_1 before m_2, B can read m_2 before m_3, and C can read m_3 before m_1. These reads satisfy the *pairwise total order* requirement of atomic multicast, since there is no conflict between any *pair* of receivers. Nevertheless, atomic multicast has failed to globally order the messages m_1, m_2, and m_3. If atomic multicast is used to impose something resembling a global order (e.g., on transactions), it should also satisfy the following *uniform acyclic order* property: the relation $<$ on (atomic-multicast) messages is acyclic, where $m < m'$ holds if there exists a node that reads m before m'.

Atomic multicast is an important concept in distributed systems, and there are a number of well-known algorithms for achieving atomic multicast [5]. To model P-Store, which uses atomic multicast, I could of course formalize a specific algorithm for atomic multicast and include it in a model of P-Store. Such a solution would, however, have a number of disadvantages, including:

1. *Messy non-modular specifications.* Atomic multicast algorithms involve some complexity, including maintaining Lamport clocks during system execution, keeping buffers of received messages that cannot be served, and so on. This solution could also easily yield a messy non-modular specification that fails to separate the specification of P-Store from that of atomic multicast.
2. *Increased state space.* Running a distributed algorithm concurrently with P-Store would also lead to much larger state spaces during model checking analyses, since also the states generated by the rewrites involving the atomic multicast algorithm would contribute to new states.
3. *Lack of generality.* Implementing a particular atomic multicast algorithm might exclude behaviors possible with other algorithms. That would mean that model checking analysis might not cover all possible behaviors of P-Store, but only those possible with the selected atomic multicast algorithm.

I therefore instead define, for each of the two "versions" of atomic multicast, a general atomic multicast primitive, which allows *all possible* ways of reading messages that are consistent with the selected version of atomic multicast. In particular, such a solution will not add states during model checking analysis.

3.1 Atomic Multicast in Maude: "User Interface"

To define an atomic multicast primitive, the system maintains a "table" of read and sent-but-unread atomic-multicast messages for each node. This table must be consulted before reading an atomic-multicast message, to check whether it can be read/served already, and must be updated when the message is read.

The "user interface" of my atomic multicast "primitive" is as follows:

- *Atomically multicasting a message.* A node n that wants to atomically multicast a message m to a set of nodes $\{n_1, \ldots, n_k\}$ just "sends" the "message"

  `atomic-multicast` m `from` n `to` n_1 `...` n_k

  where the message m should be a term of sort `MsgContent`.
- *Reading an atomically multicast message.* A node must check the multicast table whether a given atomic-multicast message can be read. If so, this table must be updated to reflect that the message has been read. A rewrite rule where an object o_1 reads an atomically multicast message m should therefore have the following form, where `AM-TABLE` is a variable of sort `AM-Table`:

```
crl [read-atomically-multicast-m] :
    (msg m from o1 to o2)
    < o2 : ... |  ... >    AM-TABLE
  =>
    < o2 : ... | ... >
    updateAM(m, o2, AM-TABLE)  if okToRead(m, o2, AM-TABLE) .
```

– The user must add the term `[emptyAME]` (denoting the "empty" atomic multicast table) to the initial state.

3.2 Maude Specification of Atomic Multicast

To keep track of atomic-multicast messages sent and received, the table

```
[am-entry(o1, read1, unread1)    ...    am-entry(on, readn, unreadn)]
```

is added to the state. This table contains a record `am-entry(`o_k, $read_k$, $unread_k$`)` for each object o_k that has been sent an atomically multicast message. $read_k$ is a *list* of atomic-multicast messages read by o_k, in the order in which the messages were read, and $unread_k$ is a *set* of atomic-multicast messages not yet read by o_k.

The "wrapper" used for atomic multicast takes as arguments the message (content), the sender's identifier, and the (identifiers of the) set of receivers:

```
op atomic-multicast_from_to_ : MsgCont Oid OidSet -> Configuration .
```

The equation

```
eq (atomic-multicast MC from O to OS)  [AM-ENTRIES]
 = (distribute MC from O to OS)        [insert(MC,OS,AM-ENTRIES)] .
```

"distributes" such an `atomic-multicast` *msg* `from` o `to` $o_1 \ldots o_n$ message by: (1) "dissolving" the above multicast message into a set of messages

```
(msg msg from o to o1)  ...  (msg msg from o to on),
```

one for each receiver o_k in the set $\{o_1, \ldots, o_n\}$; and (2) by adding, for each receiver o_k, the message (content) *msg* to the set $unread_k$ of unread atomic-multicast messages in the atomic-multicast table.

The `update` function, which updates the atomic-multicast table when `O` reads a message `MC`, just moves `MC` from the set of unread messages to the end of the list of read messages in `O`'s record in the table.

The expression `okToRead(`*mc*`,` o`,` *amTable*`)` is used to check whether the object o can read the atomic-multicast message *mc* with the given global atomic-multicast table *amTable*. The function `okToRead` is defined differently depending on whether atomic multicast must satisfy the *uniform acyclic order* requirement.

***okToRead** for Pairwise Total Order Atomic Multicast.* The following equations define `okToRead` by first characterizing the cases when the message *cannot* be read; the last equation uses Maude's `owise` construct to specify that the message can be read in all other cases:

```
vars MC MC2 : MsgContent .   vars MCS MCS2 : MsgContSet .
vars MCL MCL2 MCL3 MCL4 : MsgContList .

eq okToRead(MC, O, [am-entry(O, MCL, MCS MC MC2)
                    am-entry(O2, MCL2 :: MC2 :: MCL3 :: MC :: MCL4, MCS2)
                    AM-ENTRIES]) = false .

eq okToRead(MC, O, [am-entry(O, MCL, MCS MC MC2)
                    am-entry(O2, MCL2 :: MC2 :: MCL4, MCS2 MC)
                    AM-ENTRIES]) = false .

eq okToRead(MC, O, [AM-ENTRIES]) = true [owise] .
```

In the first equation, `O` wants to read `MC`, and its AM-entry shows that `O` has not read message `MC2`. However, another object `O2` has already read `MC2` *before* `MC`, which implies that `O` cannot read `MC`. In the second equation some object `O2` has read `MC2` and has `MC` in its sets of unread atomic-multicast messages, which implies that `O` cannot read `MC` yet (it must read `MC2` first).

okToRead *for Uniform Acyclic Order Atomic Multicast.* To define atomic multicast which satisfies the uniform acyclic order requirement, the above definition must be generalized to consider the induced relation $<$ instead of pairwise reads.

The above definition checks whether a node o can read a message m_1 by checking whether it has some other *unread* message m_2 pending such reading m_1 before m_2 would conflict with the m_1/m_2-reading order of another node. This happens if another node has read m_2 before reading m_1, or if it has read m_2 and has m_1 pending (which implies that eventually, that object would read m_2 before m_1). In the more complex uniform acyclic order setting, that solution must be generalized to check whether reading m_1 before any other pending message m_2 would violate the *current* or the (necessary) *future* "global order." That is, is there some m_1 elsewhere that has been read or must eventually be read *after* m_2 somewhere? If so, node o obviously cannot read m_1 at the moment.

The function `receivedAfter` takes a set of messages and the global AM-table as arguments, and computes the $<^*$-closure of the original set of messages; i.e., the messages that cannot be read before the original set of messages:

```
op receivedAfter : MsgContSet AM-Table -> MsgContSet .

ceq receivedAfter(MC MCS, [am-entry(O2, MCL :: MC :: MC2 :: MCL2, MCS2)
                           AM-ENTRIES])
  = receivedAfter(MC MCS MC2, [am-entry(O2, MCL :: MC :: MC2 :: MCL2, MCS2)
                               AM-ENTRIES])
  if not (MC2 in MCS) .
```

In the above equation, there is a message `MC` in the current set of messages in the closure. Furthermore, the global atomic-multicast table shows that some node `O2` has read MC2 right after reading MC, and MC2 is not yet in the closure. Therefore, MC2 is added to the closure.

In the following equation, there is a message MC in the closure; furthermore, some object O2 has already read MC. This implies that all *unread* messages MCS2 of O2 must eventually be read after MC, and hence they are added to the closure:

```
ceq receivedAfter(MC MCS, [am-entry(O2, MCL2 :: MC :: MCL4, MCS2)
                           AM-ENTRIES])
  = receivedAfter(MC MCS MCS2,
                  [am-entry(O2, MCL2 :: MC :: MCL4, emptyMsgContSet)
                   AM-ENTRIES])     if MCS2 =/= emptyMsgContSet .
```

Finally, the current set is returned when it cannot be extended:

```
  eq receivedAfter(MCS, AM-TABLE) = MCS [owise] .
```

The function okToRead can then be defined as expected: O can read the pending message MC if MC is not (forced to be) read after any other pending message (in the set MCS):

```
eq okToRead(MC, O, [am-entry(O, MCL, MCS MC)  AM-ENTRIES])
 = not (MC in receivedAfter(MCS, [am-entry(O, MCL, MCS) AM-ENTRIES])) .
```

I have model-checked both specifications of atomic multicast on a number of scenarios and found no deadlocks or inconsistent multicast read orders.

4 P-Store

P-Store [14] is a partially replicated data store for wide-area networks developed by Schiper, Sutra, and Pedone that provides transactions with serializability. P-Store executes transactions *optimistically*: the execution of a transaction T at site s (which may involve remote reads of data items not replicated at s) proceeds without worrying about conflicting concurrent transactions at other sites. After the transaction T has finished executing, a *certification process* is executed to check whether or not the transaction T was in conflict with a concurrent transaction elsewhere, in which case T might have to be aborted. More precisely, in the certification phase the site s atomically multicasts a request to certify T to all sites storing data accessed by T. These sites then perform a voting procedure to decide whether T can commit or has to be aborted.

P-Store has a number of attractive features: (i) it is a *genuine* protocol: only the sites replicating data items accessed by a transaction T are involved in the certification of T; and (ii) P-Store uses atomic multicast at most once per transaction. Another issue in the certification phase: in principle, the sites certify the transactions in the order in which the certification requests are read. However, if for some reason the certification of the first transaction in a site's certification queue takes a long time (maybe because other sites involved in the voting are still certifying other transactions), then the certification of the next transaction in line will be delayed accordingly, leading to the dreaded *convoy effect.* P-Store has an "advanced" version that tries to mitigate this problem by allowing a site to start the certification also of other transactions in its certification queue, as long as they are not in a possible conflict with "older" transactions in that queue.

The authors of [14] claim that they have proved the P-Store algorithm correct.

4.1 P-Store in Detail

This section summarizes the description of P-Store in [14].

System Model and Assumptions. A database is a set of triples (k, v, ts), where k is a key, v its value, and ts its time stamp. Each site holds a partial copy of the database, with $Items(s)$ denoting the keys replicated at site s. I do not consider failures in this paper (as failure treatment is not described in the algorithms in [14]). A transaction T is a sequence of read and write operations, and is executed locally at site $proxy(T)$. $Items(T)$ is the set of keys read or written by T; $WReplicas(T)$ and $Replicas(T)$ denote the sites replicating a key written, respectively read or written, by T. A transaction T "is *local* iff for any site s in $Replicas(T)$, $Items(T) \subseteq Items(s)$; otherwise, T is *global.*"

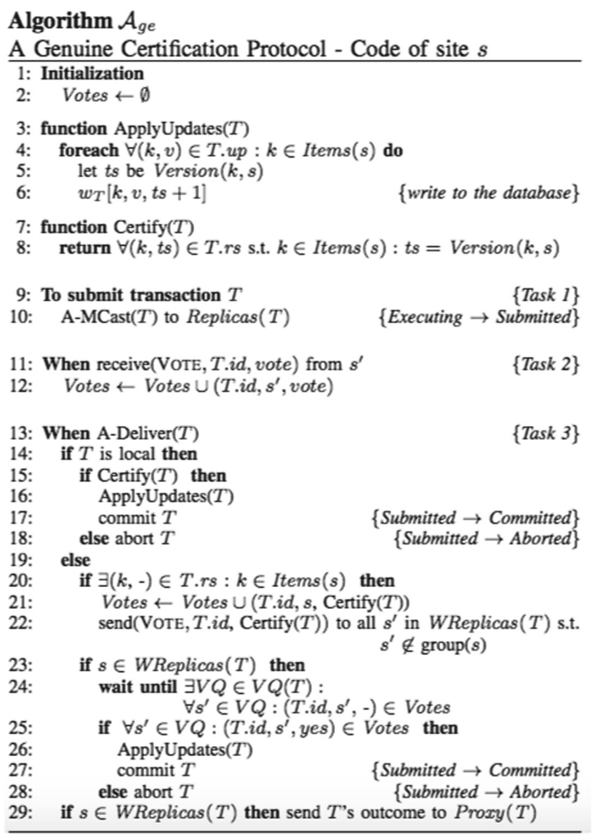
Each site ensures *order-preserving serializability* of its local executions of transactions. As already mentioned, P-Store assumes access to an atomic multicast service that guarantees uniform acyclic order.

Executing a Transaction. While a transaction T is executing (at site $proxy(T)$), a read on key k is executed at some site that stores k; k and the item time stamp ts read are stored as a pair (k, ts) in T's *read set* $T.rs$. Every write of value v to key k is stored as a pair (k, v) in T's set of updates $T.up$. If T reads a key that was previously updated by T, the corresponding value in $T.up$ is returned.

When T has finished executing, it can be committed immediately if T is read-only and local. Otherwise, we need to run the certification protocol, which also propagates T's updates to the other (write-) replicating sites.

If the certification process, described next, decides that T can commit, all sites in $WReplicas(T)$ apply T's updates. In any case, $proxy(T)$ is notified about the outcome (commit or abort) of the certification.

Certification Phase. When T is submitted for certification, T is atomically multicast to all sites storing keys read (to check for stale reads) or written (to propagate the updates) by T. When a site s reads such a request, it checks whether the values read by T are up-to-date by comparing their versions against those currently stored in the database. If they are the same, T passes the certification test; otherwise T fails at s.



Algorithm $\mathcal{A}_{ge}$

A Genuine Certification Protocol - Code of site s

```
Initialization
  $Votes \leftarrow \emptyset$

function ApplyUpdates($T$)
  foreach $\forall(k, v) \in T.up : k \in Items(s)$ do
    let $ts$ be $Version(k, s)$
    $w_T[k, v, ts + 1]$                                  {write to the database}

function Certify($T$)
  return $\forall(k, ts) \in T.rs$ s.t. $k \in Items(s) : ts = Version(k, s)$

To submit transaction $T$                                 {Task 1}
  A-MCast($T$) to $Replicas(T)$                 {Executing → Submitted}

When receive(VOTE, $T.id$, $vote$) from $s'$              {Task 2}
  $Votes \leftarrow Votes \cup (T.id, s', vote)$

When A-Deliver($T$)                                       {Task 3}
  if $T$ is local then
    if Certify($T$) then
      ApplyUpdates($T$)
      commit $T$                                {Submitted → Committed}
    else abort $T$                                {Submitted → Aborted}
  else
    if $\exists(k, -) \in T.rs : k \in Items(s)$ then
      $Votes \leftarrow Votes \cup (T.id, s, \text{Certify}(T))$
      send(VOTE, $T.id$, Certify($T$)) to all $s'$ in $WReplicas(T)$ s.t.
                                              $s' \notin$ group($s$)
    if $s \in WReplicas(T)$ then
      wait until $\exists VQ \in VQ(T)$ :
                      $\forall s' \in VQ : (T.id, s', -) \in Votes$
      if $\forall s' \in VQ : (T.id, s', yes) \in Votes$ then
        ApplyUpdates($T$)
        commit $T$                              {Submitted → Committed}
      else abort $T$                              {Submitted → Aborted}
  if $s \in WReplicas(T)$ then send $T$'s outcome to $Proxy(T)$
```

Fig. 1. The P-Store certification algorithm in [14].

The site s may not replicate all keys read by T and therefore may not be able to certify T. In this case there is a voting phase where each site s replicating keys read by T sends the result of its local certification test to all sites s_w replicating a key written by T. A site s_w can decide on T's outcome when it has received (positive) votes from a *voting quorum* for T, i.e., a set of sites that together replicate all keys read by T. If some site votes "no," the transaction must be aborted. The pseudo-code description of this certification algorithm in [14] is shown in Fig. 1.

As already mentioned, a site does not start the certification of another transaction until it is done certifying the first transaction in its certification queue. To avoid the convoy effect that this can lead to, the paper [14] also describes a version of P-Store where different transactions in a site's certification queue can be certified concurrently as long as they do not read-write conflict.

5 Formalizing P-Store in Maude

I have formalized both versions of P-Store (i.e., with and without sites initiating multiple concurrent certifications) in Maude, and present parts of the formalization of the simpler version. The executable specifications of both versions, with analysis commands, are available at http://folk.uio.no/peterol/WADT16, and the longer report [10] provides more detail.

5.1 Class Declarations

Transactions. Although the actual values of keys in the databases are sometimes ignored during analysis of distributed data stores, I choose for purposes of illustration to represent the concrete values of *keys* (or *data items*). This should not add new states that would slow down the model checking analysis.

A transaction (sometimes also called a transaction request) is modeled as an object of the following class `Transaction`:

```
class Transaction | operations : OperationList,   destination : Oid,
                    readSet : ReadSet,            writeSet : WriteSet,
                    status : TransStatus,         localVars : LocalVars .
```

The `operations` attribute denotes the list of read and write operations that remain to be executed. Such an operation is either a *read* operation x `:= read` k, where x is a "local variable" that stores the value of the (data item with) key k read by the operation, or a *write* operation `write(`k`,` *expr*`)`, where *expr* in our case is a simple arithmetic expression involving the transaction's local variables. `waitRemote(`k`,` x`)` is an "internal operation" denoting that the transaction execution is awaiting the value of a key k (to be assigned to the local variable x) which is not replicated by the transaction's proxy. An operation list is a list of such operations, with list concatenation denoted by juxtaposition. `destination` denotes the (identity of the) proxy of the transaction; that is, the site that should execute the transaction. The `readSet` attribute denotes the ','-separated set of pairs `versionRead(`k`,` *version*`)`, each such pair denoting that the transaction has read version *version* of the key k. The `writeSet` attribute denotes the write set of the transaction as a map `(`k_1 `|->` val_1`), ..., (`k_n `|->` val_n`)`. The `status` attribute denotes the commit state of the transaction, which is either `commit`, `abort`, or `undecided`. Finally, `localVars` is a map from the transaction's local variables to their current values.

Replicas. A *replicating site* (or *site* or *replica*) stores parts of the database, executes the transactions for which it is the proxy, and takes part in the certification of other transactions. A replica is formalized as an object instance of the following subclass `Replica`:

```
class Replica | datastore : DataStore,         executing : Configuration,
                submitted : Configuration,     committed : Configuration,
                aborted   : Configuration,         queue : ObjectList .
                transToCertify : CertificationData,
                decidedTranses : TransStatusSet .
```

The `datastore` attribute represents the replica's local database as a set `<` key_1 `,` val_1 `,` ver_1 `> , ... , <` key_l `,` val_l `,` ver_l `>` of triples `<` key_i `,` val_i `,` ver_i `>` denoting a *version* of the data item with key key_i, value val_i, and version number ver_i.[3] The attributes `executing`, `submitted`, `committed`, and `aborted` denote the transactions executed by the replica and which are/have been, respectively, currently executing, submitted for certification, committed, and aborted. The `queue` holds the certification queue of transactions to be certified by the replica (in collaboration with other replicas). `transToCertify` contains data used for

[3] The paper [14] does not specify whether a replica stores multiple versions of a key.

the certification of the first element in the certification queue (in the simpler algorithm), and `decidedTranses` show the status (aborted/committed) of the transactions that have previously been (partly) certified by the replica.

Clients. Finally, I add an "interface/application layer" to the P-Store specification in the form of *clients* that send transactions to be executed by P-Store:

```
class Client | txns : ObjectList,   pendingTrans : TransIdSet .
```

`txns` denotes the list of transaction (objects) that the client wants P-Store to execute, and `pendingTrans` is either the empty set or (the identity of) the transaction the client has submitted to P-Store but whose execution is not yet finished.

Initial State. The following shows an initial state `init4` (with some parts replaced by '`...`') used in the analysis of P-Store. This system has: two clients, `c1` and `c2`, that want P-Store to execute the two transactions `t1` and `t2`; three replicating sites, `r1`, `r2`, and `r3`; and three data items/keys `x`, `y`, and `z`. Transaction `t1` wants to execute the operations `(xl := read x) (yl := read y)` at replica `r1`, while transaction `t2` wants to execute `write(y, 5) write(x, 8)` at replica `r2`. The initial state also contains the empty atomic multicast table and the table which assigns to each key the sites replicating this key. Initially, the value of each key is `[2]` and its version is `1`. Site `r2` replicates both `x` and `y`.

```
eq init4
 = [emptyAME]
   [replicatingSites(x,r2) ;; replicatingSites(y, (r2 ,r3))
    ;; replicatingSites(z,r1)]
   < c1 : Client |
         txns : < t1 : Transaction | operations : ((xl :=read x) (yl :=read y)),
                                     destination : r1,    readSet : emptyReadSet,
                                     status : undecided,  writeSet : emptyWriteSet,
                                     localVars : (xl |-> [0] , yl |-> [0]) >,
         pendingTrans : empty >
   < c2 : Client |
         txns : < t2 : Transaction | operations : (write(y, 5) write(x, 8)),
                                     destination : r2,  ... >
         pendingTrans : empty >
   < r1 : Replica | datastore : (< z, [2], 1 >),
                    committed : none,   aborted : none,   executing : none,
                    submitted : none,   queue : emptyTransList,
                    transToCertify : noTrans,   decidedTranses : noTS >
   < r2 : Replica | datastore : ((< x, [2], 1 >) , (< y, [2], 1 >)),  ... >
   < r3 : Replica | datastore : (< y, [2], 1 >),  ... > .
```

5.2 Local Execution of a Transaction

The execution of a transaction has two phases. In the first phase, the transaction is executed locally by its proxy: the transaction performs its reads and writes, but the database is not updated; instead, the reads are recorded in the transaction's read set, and its updates are stored in the `writeSet` attribute.

The second phase is the certification (or validation) phase, when all appropriate nodes together decide whether or not the transaction can be committed or must be aborted. If it can be committed, the replicas update their databases.

This section specifies the first phase, which starts when a client without pending transactions sends its next transaction to its proxy. I do not show the variable declarations (see [10]), but follow the convention that variables are written with (all) capital letters.

```
rl [sendTxn] :
   < C : Client | pendingTrans : empty,
                  txns : < TID : Transaction | destination : RID > ; TXNS >
 =>
   < C : Client | pendingTrans : TID, txns : TXNS >
   (msg executeTrans(< TID : Transaction | >) from C to RID) .
```

P-Store assumes that the local executions of multiple transactions on a site are equivalent to some serialized executions. I model this assumption by executing the transactions one-by-one. Therefore, a replica can only receive a transaction request if its set of currently `executing` transactions is empty (`none`):

```
rl [receiveTxn] :
   (msg executeTrans(< TID : Transaction | >) from C to RID)
   < RID : Replica | executing : none >
 =>
   < RID : Replica | executing :  < TID : Transaction | > > .
```

There are three cases to consider when executing a read operation `X := read K`: (i) the transaction has already written to key `K`; (ii) the transaction has not written `K` and the proxy replicates `K`; or (iii) the key `K` has not been read and the proxy does not replicate `K`. I only show the specification for case (i). I do not know what version number should be associated to the read, and I choose not to add the item to the read set. (The paper [14] does not describe what to do in this case; the problem disappears if we make the common assumption that a transaction always reads a key before updating it.) As an effect, the local variable `X` gets the value `V`:

```
rl [executeRead1] :
   < RID : Replica | executing :
           < TID : Transaction | operations : (X :=read K) OPLIST,
                                 writeSet : (K |-> V), WS,  localVars : VARS > >
=>
   < RID : Replica | executing :
           < TID : Transaction | operations : OPLIST,
                                 localVars : insert(X,V,VARS) > > .
```

Write operations are easy: evaluate the expression `EXPR` to write and add the update to the transaction's `writeSet`:

```
rl [executeWrite] :
   < RID : Replica | executing :
           < TID : Transaction | operations : write(K, EXPR) OPLIST,
                                 localVars : VARS,  writeSet : WS > >
 =>
   < RID : Replica | executing :
           < TID : Transaction | operations : OPLIST,
                                 writeSet : insert(K, eval(EXPR, VARS), WS) > > .
```

5.3 Certification Phase

When all the transaction's operations have been executed by the proxy, the proxy's next step is to try to commit the transaction. If the transaction is read-only and *local*, it can be committed directly; otherwise, it must be submitted to the certification protocol.

Some colleagues and I all found the definition of *local* in [14] (and quoted in Section 4) to be quite ambiguous. We thought that "for any site s in $\mathit{Replicas}(T)$, $\mathit{Items}(T) \subseteq \mathit{Items}(s)$" means either "for each site s ..." or that $\mathit{proxy}(T)$ replicates all items in T. The first author of [14], Nicolas Schiper, told me that it actually means "for *some* s ...." In hindsight, we see that this is also a valid interpretation of the definition of *local*. To avoid misunderstanding, it is probably good to avoid the phrase "for any" and use either "for each" or "for some."

If the transaction T cannot be committed immediately, it is submitted for certification by atomically multicasting a certification request—with the transaction's identity `TID`, read set `RS`, and write set `WS`—to all replicas storing keys read or updated by T (lines 9–10 in Fig. 1):

```
crl [commit/submit2] :
    < RID : Replica | executing :
            < TID : Transaction | operations : nil, readSet : RS, writeSet : WS >,
                        submitted : TRANSES >
    REPLICA-TABLE
  =>
    < RID : Replica | executing : none,  submitted : TRANSES  < TID : Transaction | > >
    REPLICA-TABLE
    (atomic-multicast certify(TID,RS,WS) from RID
                  to replicas((keys(RS) , keys(WS)), REPLICA-TABLE))
  if WS =/= emptyWriteSet or not localTrans(keys(RS),REPLICA-TABLE) .
```

According to lines 7–8 in Fig. 1, a replica's local certification succeeds if, for each key in the transaction's read set that is replicated by the replica in question, the transaction read the same version stored by the replica:

```
op certificationOk : ReadSet DataStore -> Bool .
eq certificationOk((versionRead(K,VERSION) , READSET), (< K,V,VERSION2 > , DS))
 =  (VERSION == VERSION2) and certificationOk(READSET, (< K,V,VERSION2 > , DS)) .
eq certificationOk(RS, DS) = true [owise] .
```

If the transaction to certify is not local, the certifying sites must together decide whether or not the transaction can be committed. Each certifying site therefore checks whether the transaction passes the local certification test, and sends the outcome of this test to the other certifying sites (lines 13 and 19–22):

```
crl [certify-nonLocal] :
    (msg certify(TID, RS, WS) from RID2 to RID)
    < RID : Replica | datastore : DS, transToCertify : noTrans, decidedTranses : TSS >
    AM-TABLE      REPLICA-TABLE
  =>
    < RID : Replica | transToCertify : (if LOCAL-CERTIFICATION-OK
                                         then certify(TID, RID2, RS, WS, RID)
                                         else noTrans fi),
                      decidedTranses : (if LOCAL-CERTIFICATION-OK then TSS
                                         else (transStatus(TID, abort) ; TSS) fi) >
    (if intersection(keys(DS), keys(RS)) =/= noKey
     then (distribute
             vote(RID, TID, if LOCAL-CERTIFICATION-OK then commit else abort fi)
           from RID to  (replicas(keys(WS), REPLICA-TABLE)  RID))
     else none fi)
    (if (not LOCAL-CERTIFICATION-OK) and          --- if certification fails ...
         intersection(keys(DS), keys(WS)) =/= noKey        --- write replica ...
     then (msg abort(TID) from RID to RID2) else none fi)  --- notifies proxy
    REPLICA-TABLE
    updateAM(certify(TID, RS, WS), RID, AM-TABLE)
  if okToRead(certify(TID, RS, WS), RID, AM-TABLE)
     /\ not localTrans((keys(RS) , keys(WS)), REPLICA-TABLE)
     /\ LOCAL-CERTIFICATION-OK := certificationOk(RS, DS) .
```

If the local certification fails, the site sends an `abort` vote to the other *write* replicas and also notifies the proxy of the outcome. Otherwise, the site sends a `commit` vote to all other site replicating an item written by the transaction.

The voting phase ends when there is a voting quorum; that is, when the voting sites together replicate all keys read by the transaction. This means that a certifying site must keep track of the votes received during the certification of a transaction. The set of sites from which the site has received a (positive) vote is the fourth parameter of the `certify` record it maintains for each transaction. If a site receives a positive vote, it stores the sender of the vote (lines 11–12). If a site receives a negative vote, it decides the fate of the transaction and notifies the proxy if it replicates an item written by the transaction (lines 28–29).

If a write replica has received positive votes from a voting quorum (lines 23–27 and 29), the transaction can be committed, and the write replica applies the updates and notifies the proxy. The following rule models the behavior when a site has received votes from a voting quorum `RIDS` for transaction `TID`:

```
crl [quorum] :
    < RID : Replica | transToCertify : certify(TID, RID3, RS, WS, RIDS),
                      decidedTranses : TSS,   datastore : DS >
    REPLICA-TABLE
  =>
    < RID : Replica | transToCertify : noTrans,   datastore : applyUpdates(DS, WS),
                      decidedTranses : TSS ; transStatus(TID, commit) >
    REPLICA-TABLE
    (if intersection(keys(DS), keys(WS)) =/= noKey        --- if write replica ...
     then (msg commit(TID) from RID to RID3) else none fi)     --- notify proxy
  if (keys(RS) subset replicatedKeys(RIDS, REPLICA-TABLE)) .  --- voting quorum!
```

Finally, the proxy of transaction `TID` receives the outcome from one or more sites in `TID`'s certification set (the `abort` case is similar):

```
rl [readCommit] :
   (msg commit(TID) from RID2 to RID)
   < RID : Replica | submitted : < TID : Transaction | >,  committed : TRANSES >
 =>
   < RID : Replica | submitted : none,
                     committed : (TRANSES < TID : Transaction | >) >
   done(TID) .      --- notify client
```

6 Formal Analysis of P-Store

In the absence of failures, P-Store is supposed to guarantee *serializability* of the committed transactions, and that a decision (commit/abort) is made on all transactions.

To analyze P-Store, I search for all *final* states—i.e., states that cannot be further rewritten—reachable from a given initial state, and inspect the result. This analysis therefore also discovers undesired deadlocks. In the future, I should instead automatically check serializability, possibly using the techniques in [4], which adds to the state a "serialization graph" that is updated whenever a transaction commits, and then checks whether the graph has cycles.

The search for final states reachable from state `init4` in Section 5.1 yields a state which shows that `t1`'s proxy is not notified about the outcome of the certification (see [10] for details). The problem seems to be line 29 in the algorithm in Fig. 1: only sites replicating items *written* by transaction T ($WReplicas(T)$) send the outcome of the certification to T's proxy. It is therefore not surprising that the outcome of the *read-only* transaction `t1` does not reach `t1`'s proxy.

The transactions in `init4` are local. What about non-local transactions? The initial state `init5` is the same as `init4` in Section 5.1, except that item `y` is only replicated at site `r3`, which means that `t1` and `t2` become non-local transactions.

Searching for final states reachable from `init5` shows a result where the certification process cannot reach a decision on the outcome of transaction `t1`:

```
Maude> (search init5 =>! C:Configuration .)
...
Solution 4
...
< r1 : Replica | submitted :
        < t1 : Transaction | localVars :(xl |->[8], yl |->[5]), operations : nil,
                                readSet : versionRead(x,2), versionRead(y,2), ... >,
                 transToCertify : noTrans >
< r2 : Replica | committed : < t2 : Transaction | writeSet : (x |-> [8], y |-> [5]), ... >,
                 datastore : < x,[8],2 >, decidedTranses : transStatus(t2,commit),
                 transToCertify : certify(t1,r1,(versionRead(x,2),versionRead(y,2)),
                                          emptyWriteSet,r2) ,  ... >
< r3 : Replica | aborted : none, committed : none, datastore : < y,[5],2 >,
                decidedTranses : transStatus(t2,commit),
                transToCertify : certify(t1, r1, ..., emptyWriteSet, r3) ,  ... >
```

The fate of `t1` is not decided: both `r2` and `r3` are stuck in their certification process. The problem seems to be lines 22 and 23 in the P-Store certification algorithm: why are only *write* replicas involved in sending and receiving votes during the certification? Shouldn't both read and write replicas vote? Otherwise, it is impossible to certify non-local read-only transactions, such as `t1` in `init5`.

7 Fixing P-Store

Nicolas Schiper confirmed that the errors pointed out in Section 6 are indeed errors in P-Store. He also suggested the fix alluded to in Section 6: replace $\mathit{WReplicas}(T)$ with $\mathit{Replicas}(T)$ in lines 22, 23, and 29. The Maude specification of the proposed correction is given in http://folk.uio.no/peterol/WADT16/.

Missing Assumptions. One issue seems to remain: why can read-only local transactions be committed without certification? Couldn't such transactions have read stale values? Nicolas Schiper kindly explained that local read-only transactions are handled in a special way (all values are read from the same site and some additional concurrency control is used to ensure serializability), but admitted that this is indeed not mentioned anywhere in their paper. My specifications consider the algorithm as given in [14], without taking the unstated assumptions into account, and also subjects the local read-only transactions to certification.

Analysis of the Updated Specification. I have analyzed the corrected specification on five small initial configurations (3 sites, 3 data items, 2 transactions, 4 operations). All the final states were correct: the committed transactions were indeed serializable.

The Advanced Algorithm. I have also specified and successfully analyzed the (corrected) version of P-Store where multiple transactions can be certified concurrently. It is beyond the scope of this paper to describe that specification.

8 Related Work

Different communication forms/primitives have been defined in Maude, including wireless broadcast that takes into account the geographic location of nodes and the transmission strength/radius [12], as well as wireless broadcast in mobile systems [6]. However, I am not aware of any model of atomic multicast in Maude.

Maude has been applied to a number of industrial and academic cloud storage systems, including Google's Megastore [4], Apache Cassandra [8], and UC Berkeley's RAMP [7]. However, that work did not address issues like atomic multicast and group communication commit.

Lamport's TLA+ has also been used to specify and model check large industrial cloud storage systems like S3 at Amazon [9] and the academic TAPIR transaction protocol targeting large-scale distributed storage systems.

On the validation of P-Store and similar designs, P-Store itself has been proved to be correct using informal "hand proofs" [13]. However, such hand proofs do not generate precise specifications of the systems and tend to be error-prone and rely on missing assumptions, as I show in this paper. I have not found any model checking validation of related designs, such as Jessy [1] and Walter [15].

9 Concluding Remarks

Cloud computing relies on partially replicated wide-area data stores to provide the availability and elasticity required by cloud systems. P-Store is a well-known such data store that uses atomic multicast, group communication commitment, concurrent certification of independent transactions, etc. Furthermore, many other partially replicated data stores are extensions and variations of P-Store.

I have formally specified and analyzed P-Store in Maude. Maude reachability analysis uncovered a numbers of errors in P-Store that were confirmed by one of the P-Store developers: both read and write replicas need to participate in the certification of transactions; write replicas are not enough. I have specified the proposed fix of P-Store, whose Maude analysis did not uncover any error.

Another main contribution of this paper is a general and abstract Maude "primitive" for both variations of atomic multicast.

One important advantage claimed by proponents of formal methods is that even precise-looking informal descriptions tend to be ambiguous and contain missing assumptions. In this paper I have pointed at a concrete case of ambiguity in a precise-looking definition, and at a crucial missing assumption in P-Store.

This work took place in the context of the University of Illinois Center for Assured Cloud Computing, within which we want to identify key building blocks of cloud storage systems, so that they can be built and verified in a modular way by combining such building blocks in different ways. Some of those building blocks are group communication commitment certification and atomic multicast. In more near term, this work should simplify the analysis of other state-of-the-art data stores, such as Walter and Jessy, that can be seen as extensions of P-Store.

The analysis performed was performed using reachability analysis; in the future one should also be able to specify the desired consistency property "directly."

Acknowledgments. I would like to thank Nicolas Schiper for quick and friendly replies to my questions about P-Store, the anonymous reviewers for helpful comments, and Si Liu and José Meseguer for valuable discussions about P-Store and atomic multicast.

References

1. Ardekani, M.S., Sutra, P., Shapiro, M.: Non-monotonic snapshot isolation: scalable and strong consistency for geo-replicated transactional systems. In: SRDS 2013. IEEE Computer Society (2013)
2. Ardekani, M.S., Sutra, P., Shapiro, M.: G-DUR: a middleware for assembling, analyzing, and improving transactional protocols. In: Middleware 2014. ACM (2014)

3. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
4. Grov, J., Ölveczky, P.C.: Formal modeling and analysis of Google's Megastore in Real-Time Maude. In: Iida, S., Meseguer, J., Ogata, K. (eds.) Specification, Algebra, and Software. LNCS, vol. 8373, pp. 494–519. Springer, Heidelberg (2014)
5. Guerraoui, R., Schiper, A.: Genuine atomic multicast in asynchronous distributed systems. Theor. Comput. Sci. **254**(1–2), 297–316 (2001)
6. Liu, S., Ölveczky, P.C., Meseguer, J.: Modeling and analyzing mobile ad hoc networks in Real-Time Maude. J. Log. Algebr. Methods Program. **85**(1), 34–66 (2016)
7. Liu, S., Ölveczky, P.C., Rahman, M.R., Ganhotra, J., Gupta, I., Meseguer, J.: Formal modeling and analysis of RAMP transaction systems. In: SAC 2016. ACM (2016)
8. Liu, S., Rahman, M.R., Skeirik, S., Gupta, I., Meseguer, J.: Formal modeling and analysis of Cassandra in Maude. In: Merz, S., Pang, J. (eds.) ICFEM 2014. LNCS, vol. 8829, pp. 332–347. Springer, Cham (2014)
9. Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., Deardeuff, M.: How Amazon web services uses formal methods. Commun. ACM **58**(4), 66–73 (2015)
10. Ölveczky, P.C.: Formalizing and validating the P-Store replicated data store in Maude. http://folk.uio.no/peterol/WADT16/
11. Ölveczky, P.C.: Design and validation of cloud computing data stores using formal methods. In: Proceedings of the International Symposium on Intelligent Systems and Applications (ISA 2016) (2016). http://assured-cloud-computing.illinois.edu/publications/
12. Ölveczky, P.C., Thorvaldsen, S.: Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude. Theor. Comput. Sci. **410**(2–3), 254–280 (2009)
13. Schiper, N., Sutra, P., Pedone, F.: P-Store: genuine partial replication in wide area networks. Technical report, University of Lugano (2010)
14. Schiper, N., Sutra, P., Pedone, F.: P-Store: genuine partial replication in wide area networks. In: SRDS 2010. IEEE Computer Society (2010)
15. Sovran, Y., Power, R., Aguilera, M.K., Li, J.: Transactional storage for geo-replicated systems. In: SOSP 2011. ACM (2011)

Generic Hoare Logic for Order-Enriched Effects with Exceptions

Christoph Rauch(✉), Sergey Goncharov, and Lutz Schröder

Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
{christoph.rauch,sergey.goncharov,lutz.schroeder}@fau.de

Abstract. In programming semantics, monads are used to provide a generic encapsulation of side-effects. We introduce a monad-based metalanguage that extends Moggi's *computational metalanguage* with native exceptions and iteration, interpreted over monads supporting a dcpo structure. We present a Hoare calculus with abnormal postconditions for this metalanguage and prove relative completeness using weakest liberal preconditions, extending earlier work on the exception-free case.

1 Introduction

The use of monads as a means of modelling side-effecting computations [10] is well-established in the design and semantics of programming languages. A broad range of computational effects is subsumed under this paradigm, e.g. store, non-determinism, exceptions, probabilities, and alternation.

In addition to just *modelling* side-effects, a matter of interest is how to *reason* generically about properties of side-effecting programs, one of the most important properties being functional correctness. Goncharov and Schröder [7] have developed a monad-based generic *Hoare calculus*, and proved relative completeness using weakest liberal preconditions. More precisely the calculus is parametrized by the choice of a *predicated monad*, i.e. a monad that supports iteration via a suitable dcpo enrichment of its Kleisli category (the associated category of side-effecting functions), and is moreover equipped with a suitable submonad of *innocent* computations for use in assertions. In fact, the object of truth values is generated from a predicated monad as the type of innocent computations of unit type (pre- and postconditions thus become *affirmative assertions* in the sense of Vickers [20], e.g. semi-decidable).

Exception monads $T_E X = T(X + E)$ (with E an object of exceptions and T a predicated base monad) are, in fact, predicated but include an operation that does not fit into the required scheme for basic operations, namely exception catching $\mathsf{catch} : TA \to T(A+E)$. The framework covers only operations that are *algebraic* [15], which catch is not. We therefore extend the generic Hoare calculus with explicit support for exception handling via *abnormal postconditions* [9], which govern the behaviour of a computation for the case where it terminates with an exception. The calculus thus allows, e.g., for a meaningful specification of exception raising, for which the normal postcondition is just $\bot$. Our main result is that the calculus is, again, relatively complete; we prove this by means of a weakest liberal precondition calculus featuring abnormal postconditions.

Related Work. Schröder and Mossakowski [18] present a monad-based Hoare calculus with abnormal postconditions, interpreted over a program semantics where the *base category* of the monad, rather than only the Kleisli category, is enriched over a suitable category of domains; our setting thus allows for more instances, e.g. involving monads on the category of sets. Moreover the semantics of the Hoare logic of [18] differs from ours in that the object of truth values is assumed in the base category rather than extracted from the monad as in our setting. The calculus of [18] is proved sound but a relative completeness proof is currently missing. Relative completeness proofs in *most general formula* style have been given for specific effect combinations found in concrete programming languages with exceptions, namely fragments of Java [13,16] and Eiffel [12]. Our approach to generic weakest preconditions essentially follows our previous work on the exception-free case [7]. In work subsequent to [7], Hasuo [8] conducts an alternative categorical analysis of monad-based weakest preconditions, notably relying on similar assumptions to ensure well-behavedness of weakest preconditions (see Remark 11 for a detailed comparison). In his setup, no syntactic Hoare calculus, and a fortiori no relative completeness result, is currently provided.

2 Preliminaries: Monads and Exceptions

A *monad* $\mathbb{T}$ over a category $\mathcal{C}$ with class $|\mathcal{C}|$ of objects can be described as a *Kleisli triple* $(T, \eta, \_^*)$ in which T is an endomap over $|\mathcal{C}|$ (we adopt the general convention to use blackboard characters for monads and the corresponding Roman letters for their functorial parts), η – the *unit* of the monad – is a family of morphisms $(\eta_X : X \to TX)_{X\in|\mathcal{C}|}$ and $\_^*$ – the *Kleisli lifting* – is a map sending each $f : X \to TY$ to $f^* : TX \to TY$ so that the *monad laws* are satisfied:

$$\eta^* = \mathsf{id}, \qquad f^* \circ \eta = f, \qquad (f^* \circ g)^* = f^* \circ g^* \tag{1}$$

This is equivalent to the standard definition of a monad in terms of a functor T with unit and multiplication; in particular, $Tf = (\eta f)^*$ for morphisms f. One may think of TX as being a type of side-effecting computations delivering results in X; examples include non-termination ($TX = X+1$), non-determinism ($TX = \mathcal{P}X$), statefulness (e.g. the *partial state monad* is given by $TX = S \rightharpoonup (S \times X)$ for an object S of states, where $\rightharpoonup$ takes partial function spaces), and exceptions ($TX = X + E$ for an object E of exceptions).

Furthermore, a *strong* monad is a monad $\mathbb{T}$ together with a family of morphisms $\tau : X \times TY \to T(X \times Y)$ natural in both X and Y satisfying a set of coherence axioms (cf. e.g. [11]). As originally noted by Moggi, strong monads support a *computational metalanguage* [11], incorporated into Haskell as the *do-notation* [21]: it features an operator ret, denoting the unit η, and the do-binder, which from $f : X \to TY$ and $g : X \times Y \to TZ$ builds $\mathsf{do}\ y \leftarrow f(x); g(x, y)$, in which y is syntactically bound and whose meaning is $g^\star \circ \tau \circ \langle\,\rangle \mathsf{id}, f : X \to TZ$. For a term p in the computational metalanguage, $FV(p)$ denotes the set of free variables of p. The monad laws (1) can be rewritten in this style, and then form part of a complete axiomatization:

$$\begin{aligned} \mathsf{do}\ x \leftarrow (\mathsf{do}\ y \leftarrow p; q); r &= \mathsf{do}\ y \leftarrow p; x \leftarrow q; r \qquad\qquad y \notin FV(r) \\ \mathsf{do}\ x \leftarrow \mathsf{ret} a; p &= p[a/x] \\ \mathsf{do}\ x \leftarrow p; \mathsf{ret} x &= p. \end{aligned}$$

A monad $\mathbb{T}$ defines the *Kleisli category* $\mathcal{C}_T$ of $\mathbb{T}$, whose objects are those of $\mathcal{C}$ and $\mathsf{Hom}_{\mathcal{C}_T}(A, B) = \mathsf{Hom}_{\mathcal{C}}(A, TB)$ with composition defined using the Kleisli lifting of $\mathbb{T}$; that is, morphisms $X \to Y$ in $\mathcal{C}_T$ may be thought of side-effecting functions $X \to Y$, with side-effects specified by T.

We fix from now on an object E of exceptions and a base monad $\mathbb{T}$ on a category $\mathcal{C}$ which is *distributive*, i.e. the canonical distributivity morphism $X \times Y + X \times Z \to X \times (Y + Z)$ is an isomorphism, with inverse called dist, and which has *Kleisli exponentials* [15,19], i.e. exponentials [2] of the form TZ^Y. The latter condition means that there is a natural isomorphism $\lambda : \mathsf{Hom}_{\mathcal{C}}(X \times Y, TZ) \cong \mathsf{Hom}_{\mathcal{C}}(X, TZ^Y)$. We furthermore denote pairing and copairing morphisms by angle and square brackets, respectively.

The well-known *exception monad transformer* [4] adds exceptions as an effect to a given monad: the functor

$$T_E = T(\text{-} + E)$$

is turned into a strong monad $\mathbb{T}_E$, called an *exception monad*, by putting

- $\eta^{T_E} = \eta^T \circ \mathsf{inl}$;
- $f^* = [f, \eta^T \circ \mathsf{inr}]^*$ for $f : A \to T(B + E)$;
- $\tau^{T_E}_{A,B} = T((\mathsf{id} + \pi_2) \circ \mathsf{dist}) \circ \tau_{A,B+E}$.

That is, a computation in $\mathbb{T}_E$ performs a side effect specified by $\mathbb{T}$ and then either terminates normally or with an exception. The definition of binding in $\mathbb{T}_E$ means that in the latter case, subsequent statements have no further effects (e.g. in case $\mathbb{T}$ features statefulness, the state is frozen) other than propagating the exception. On $\mathbb{T}_E$, we have operations for raising and catching exceptions (i.e. exposing exceptions raised by a program in the output type),

$$\begin{aligned} \mathsf{raise}_X &= \eta\, \mathsf{inr} : E \to T(X + E) = T_E X \\ \mathsf{catch}_X &= T\, \mathsf{inl} : T_E X = T(X + E) \to T((X + E) + E) = T_E(X + E). \end{aligned}$$

3 Order-Enrichment and Innocence

As indicated in the introduction, we require a certain amount of infrastructure in the base monad $\mathbb{T}$ to support, on the one hand, loops, and on the other hand, an internalized assertion language. To this end, we require a complete partial order on programs, i.e. Kleisli morphisms, of a given type $A \to TB$. We will then use certain well-behaved programs as logical assertions. Formal definitions are as follows [7].

Definition 1 (Order-enriched monad). The monad $\mathbb{T}$ is *order-enriched* if

- every hom-set $\mathsf{Hom}_{\mathcal{C}}(A, TB)$ carries a bounded-complete and directed-complete partial ordering $\sqsubseteq$ (i.e. joins exist for finite bounded subsets and for directed subsets, and consequently for all bounded subsets),
- for any $h \in \mathsf{Hom}_{\mathcal{C}}(A', A)$ and any $u \in \mathsf{Hom}_{\mathcal{C}}(B, TB')$, the maps

$$f \mapsto f \circ h, \qquad f \mapsto u^* \circ f, \qquad f \mapsto \tau \circ \langle \mathsf{id}, f \rangle$$

  preserve all existing joins (including the empty join $\bot$),
- Kleisli star is Scott-continuous.

The ordering $\sqsubseteq$ is to be thought of as the usual information ordering; in particular, non-termination is below termination. Note that we do not include continuity of copairing in the above definition; in fact this follows from preservation of joins (Lemma 6).

We identify a class of *innocent* computations suitable for use within assertions of the Hoare logic. Intuitively, such a computation may read but should not modify the state, as required already in earlier work on monad-based program logics [17]. Instead of assuming a type of truth values in the base category, we will identify truth with termination, so that truth values are just innocent computations of unit type. Formally, the conditions for innocence are as follows [7].

Definition 2 (Innocence). A monad is *commutative* if it satisfies

$$\mathsf{do}\ x \leftarrow p;\ y \leftarrow q;\ \mathsf{ret}\ \langle x, y\rangle = \mathsf{do}\ y \leftarrow p; x \leftarrow q;\ \mathsf{ret}\ \langle x, y\rangle$$

for all p, q, and fresh x, y, in the computational metalanguage (Sect. 2). An order-enriched monad is *innocent* if it is commutative and satisfies *copyability*

$$\mathsf{do}\ x \leftarrow p; y \leftarrow p;\ \mathsf{ret}\ \langle x, y\rangle = \mathsf{do}\ x \leftarrow p;\ \mathsf{ret}\ \langle x, x\rangle$$

(for all p and fresh x, y) and *weak discardability*

$$\mathsf{do}\ x \leftarrow p; \mathsf{ret}\star \sqsubseteq \mathsf{ret}\star\,. \tag{2}$$

Notice that the notion of weak discardability uses the ordering of the monad. It allows for non-termination; e.g. if $p = \bot$ is the everywhere diverging program, then the left hand side of (2) is also $\bot$ so that the inequality is satisfied. (Contrastingly, previous approaches [17] used *discardability* $\mathsf{do}\ x \leftarrow p;\ \mathsf{ret}\star = \mathsf{ret}\star$, which implies termination.) On the other hand, weakly discardable computations cannot raise exceptions, as these fail to be below $\mathsf{ret}\star$. Intuitively, the only side-effects an innocent computation may exhibit are possible non-termination and read operations on implicit states encapsulated in the base monad.

A key property of innocent monads $\mathbb{P}$ is that they support a logic on the base category, which we will later use as the underlying logic of the assertions of our Hoare calculus. Specifically, $P1$ *is a distributive lattice object in* $\mathcal{C}$, i.e. $\mathsf{Hom}(\text{–}, P1)$ factors through distributive lattices, in fact even through *frames* [7], which are complete lattices in which finite meets distribute over all joins (with morphisms preserving finite meets and all joins). Consequently, $P1$ supports a form of *geometric logic* [20]. Notably, finite meets in $P1$ are given by sequential composition. In case $\mathcal{C} = \mathbf{Set}$, $P1$ is in fact a Heyting algebra, and thus supports full intuitionistic logic.

Computationally relevant monads, e.g. stateful ones, often fail to be innocent but will typically contain a useful innocent submonad. Formally:

Definition 3 (Predicated monad). A *predicated monad* $(\mathbb{T}, \mathbb{P})$ consists of an order-enriched monad $\mathbb{T}$ and an innocent order-enriched submonad $\mathbb{P}$ of $\mathbb{T}$ whose inclusion into $\mathbb{T}$ is Scott-continuous.

Example 4. In the *nondeterministic state monad* $TX = S \to \mathcal{P}(S \times X)$, a program is weakly discardable iff it (possibly reads but) does not change the state. Such a program is copyable iff it is deterministic; all copyable and weakly discardable programs in $\mathbb{T}$ commute. We thus obtain an innocent submonad $\mathbb{P}$ of $\mathbb{T}$ given by $PX = S \rightharpoonup X$, the *partial reader monad*. Taking $\mathbb{T}$ to be either the partial state monad (Sect. 2) or the nondeterministic state monad and $\mathbb{P}$ the partial reader monad, we thus obtain a predicated monad $(\mathbb{T}, \mathbb{P})$.

Lemma 5. *Let* $(\mathbb{T}, \mathbb{P})$ *be a predicated monad. Then the exception monad* $\mathbb{T}_E$ *is also order-enriched, and* $(\mathbb{T}_E, \mathbb{P})$ *is a predicated monad.*

From now on, we fix a predicated monad $(\mathbb{T}, \mathbb{P})$.

4 A Computational Metalanguage

We proceed to define the syntax of a monad-based metalanguage that extends Moggi's computational metalanguage (Sect. 2) with native support for iteration and innocence. We introduce a system of *value types* A and *computation types* C by the grammar

$$A ::= W \mid 1 \mid A \times A \mid A + A \qquad C ::= A \mid \Omega \mid T_E A \mid PA \mid A \to \Omega$$

with W ranging over a set $\mathcal{W}$ of basic types (denoting objects of $\mathcal{C}$). While the type constructor T_E represents side-effecting computations possibly raising exceptions in E, the type constructor P represents innocent computations. The type Ω of truth values is just a synonym for $P1$. Additionally, we fix a signature Σ of typed function symbols $f : A \to C$.

The term language is defined by means of the set of typing rules for terms in context shown in Fig. 1; contexts Γ consist of assignments $x : A$ of value types A to variables x. We have the usual term formation rules for products, coproducts, sequential composition of computations, application of functions from the signature Σ, and most notably for exception raising and handling. Additionally, we incorporate rules for innocent computations saying essentially that P is a submonad of T_E, i.e. closed under unit and binding, and each PA has a least element $\bot$ ($\bot$ could of course be defined as a nonterminating loop but for our purposes it is technically more convenient to make it a first-class citizen). Our loop construct is tuned to sum types:

$$\mathsf{init}\, x \leftarrow p\, \mathsf{itercase}\, c\, \mathsf{of}\ \mathsf{inl}\, y \mapsto q;\, \mathsf{inr}\, z \mapsto r$$

initializes the loop variable x with the result of p and then proceeds to execute q repeatedly, binding the result to x, as long as $c : B + C$ remains in the left hand summand. Once c ends up in the right hand summand, the loop terminates and r is executed once, with the result bound to x. Note that the construct at hand allows passing *values* through the loop; boolean loop conditions as in standard while loops arise as special cases of our sum-type-based loop conditions, as the type 2 of Booleans is the sum type $1 + 1$.

$$\frac{x : A \text{ in } \Gamma}{\Gamma \vdash x : A} \qquad \frac{f : A \to C \in \Sigma \quad \Gamma \vdash t : A}{\Gamma \vdash f(t) : C} \qquad \frac{}{\Gamma \vdash \star : 1} \qquad \frac{}{\Gamma \vdash \bot : PA}$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_1 t : A} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_2 t : B} \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{inl}\, t : A + B} \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \mathsf{inr}\, t : A + B}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : A \times B} \qquad \frac{\Gamma \vdash c : A + B \quad \Gamma, a : A \vdash t : C \quad \Gamma, b : B \vdash u : C}{\Gamma \vdash \mathsf{case}\, c\, \mathsf{of}\ \mathsf{inl}\, a \mapsto t;\ \mathsf{inr}\, b \mapsto u : C}$$

$$\frac{\Gamma \vdash p : MA \quad \Gamma, x : A \vdash q : MB}{\Gamma \vdash \mathsf{do}\ x \leftarrow p; q : MB}\ M \in \{T_E, P\} \qquad \frac{\Gamma \vdash p : A}{\Gamma \vdash \mathsf{ret}\, p : PA}\ (A \text{ value type})$$

$$\frac{\Gamma, x : A \vdash c : B + C \quad \Gamma \vdash p : T_E A \quad \Gamma, y : B \vdash q : T_E A \quad \Gamma, z : C \vdash r : T_E A}{\Gamma \vdash \mathsf{init}\, x \leftarrow p\, \mathsf{itercase}\, c\, \mathsf{of}\ \mathsf{inl}\, y \mapsto q;\ \mathsf{inr}\, z \mapsto r : T_E A}$$

$$\frac{\Gamma \vdash p : PA}{\Gamma \vdash p : T_E A} \qquad \frac{\Gamma \vdash e : E}{\Gamma \vdash \mathsf{raise}\, e : T_E A} \qquad \frac{\Gamma \vdash p : T_E A}{\Gamma \vdash \mathsf{catch}\, p : T_E(A + E)}$$

Fig. 1. Term formation rules for the simple imperative metalanguage with exceptions.

In addition to the metalanguage of programs, we can now use the fact that $P1 = \Omega$ serves as an object of truth values in geometric logic to develop rules for an assertion language. These rules, given in Fig. 2 and understood as extending those of Fig. 1, are identical to those for the exception-free case [7], essentially because innocent computations do not raise exceptions. The logic combines a very restricted set of propositional connectives with least and greatest fixpoint constructors over dedicated predicate variables. It is designed to accommodate weakest preconditions, i.e. is strong enough to guarantee relative completeness of the Hoare calculus.

$$\frac{}{\Gamma \vdash \top : \Omega} \qquad \frac{}{\Gamma \vdash \bot : \Omega} \qquad \frac{\Gamma \vdash \phi : \Omega \quad \Gamma \vdash \psi : \Omega}{\Gamma \vdash \phi \wedge \psi : \Omega} \qquad \frac{\Gamma \vdash \phi : \Omega \quad \Gamma \vdash \psi : \Omega}{\Gamma \vdash \phi \vee \psi : \Omega}$$

$$\frac{\Gamma \vdash p : PA \quad \Gamma, x : A \vdash q : \Omega}{\Gamma \vdash \mathsf{do}\ x \leftarrow p; q : \Omega} \qquad \frac{\Gamma, X : A \to \Omega \vdash \phi : A \to \Omega}{\Gamma \vdash \eta X.\, \phi : A \to \Omega}\ (\eta \in \{\mu, \nu\})$$

$$\frac{X : A \to \Omega \text{ in } \Gamma}{\Gamma \vdash X : A \to \Omega} \qquad \frac{\Gamma, x : A \vdash t : \Omega}{\Gamma \vdash \lambda x.\, t : A \to \Omega} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash s : A \to \Omega}{\Gamma \vdash s(t) : \Omega}$$

Fig. 2. Assertion language.

5 Semantics

The types C of our metalanguage are interpreted over the predicated (exception) monad $(\mathbb{T}_E, \mathbb{P})$ as $\mathcal{C}$-objects $[\![C]\!]$ in the expected way, given an assignment of $\mathcal{C}$-objects $[\![W]\!]$ to basic types W, with $+$ and $\times$ interpreted as categorical sum and product, P and T_E as the corresponding object maps of $\mathbb{T}_E$ and $\mathbb{P}, 1$ as the terminal object, and $A \to \Omega$ as the Kleisli exponential $([\![P1]\!])^{[\![A]\!]}$.

As usual, contexts are then interpreted as the product of the interpretations of the types of their variables, and a term in context $\Gamma \vdash p : C$ as a morphism $[\![\Gamma]\!] \to [\![C]\!]$. The term constructors for product and sum types are interpreted using the structure of $\mathcal{C}$ as a distributive category in the usual way, and the interpretation of the term constructors for the monadic structure (ret and do) is as in the computational metalanguage [11], instantiated for the exception monad $\mathbb{T}_E$, respectively its submonad $\mathbb{P}$ for innocent computations; $\Gamma \vdash \bot : PA$ is interpreted as the bottom element of $\mathsf{Hom}([\![\Gamma]\!], [\![PA]\!])$. The operations catch and raise are interpreted as the corresponding morphisms for $\mathbb{T}_E$ (Sect. 2); e.g. $[\![\Gamma \vdash \mathsf{catch}\, p : T_E(A + E)]\!] = \mathsf{catch} \circ [\![p]\!]$.

The interpretation of our coproduct-based loop construct itcase follows the treatment of a previous Boolean loop construct [7]. Let $? : A + B \to PA$ be the operator defined by

$$c? = \mathsf{case}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto \mathsf{ret}\, y; \mathsf{inr}\, z \mapsto \bot,$$

and let $c \mapsto \bar{c}$ denote the symmetry isomorphism $A + B \to B + A$, i.e. $\bar{c} = \mathsf{case}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto \mathsf{inr}\, y; \mathsf{inr}\, z \mapsto \mathsf{inl}\, z$ for $c : A + B$. We adopt the convention to write $c?$ instead of $(\mathsf{do}\ c?; \mathsf{ret}\, \star)$ whenever the return type $P1$ is expected.

Lemma 6. *Assume terms $\Gamma, y : A \vdash p : T_E C$; $\Gamma, z : B \vdash q : T_E C$; and $\Gamma \vdash c : A + B$. Then the join $(\mathsf{do}\ y \leftarrow c?; p) \sqcup (\mathsf{do}\ z \leftarrow \bar{c}?; q)$ exists and equals $\mathsf{case}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto p; \mathsf{inr}\, z \mapsto q$.*

Consequently, case is Scott-continuous in both program arguments. By Kleene's fixpoint theorem, we can thus define an interpretation for the special case $\mathsf{init}\, x \leftarrow \mathsf{ret}\, x\, \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto r$ as the least fixed point of the map

$$f \mapsto \big(\mathsf{case}\, c\, \mathsf{of}\ \mathsf{inl}\, y \mapsto (\mathsf{do}\ x \leftarrow q;\ f); \mathsf{inr}\, z \mapsto r\big).$$

The full itcase construct is then interpreted as

$$\begin{aligned}&[\![\Gamma \vdash \mathsf{init}\, x \leftarrow p\, \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto r : T_E A]\!] = \\ &\quad [\![\Gamma \vdash \mathsf{do}\, x \leftarrow p; (\mathsf{init}\, x \leftarrow \mathsf{ret}\, x\, \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto r) : T_E A]\!]. \qquad (3)\end{aligned}$$

Semantics of assertions. Contexts Γ for assertions may contain propositional variables $X : A \to \Omega$, giving rise to factors $([\![P1]\!])^{[\![A]\!]}$ in the interpretation of Γ. Assertions $\Gamma \vdash \phi : \Omega$ are then interpreted as morphisms $[\![\Gamma]\!] \to [\![P1]\!]$. Logical conjunction and disjunction are interpreted as meets and joins in $\Omega = P1$, respectively (which exist because $P1$ is a distributive lattice object). For lambda abstraction and evaluation of predicates, we inductively define

$$\frac{[\![\Gamma, x : A \vdash t : \Omega]\!] = f}{[\![\Gamma \vdash \lambda x.\, t : A \to \Omega]\!] = \lambda f} \qquad \frac{[\![\Gamma \vdash t : A]\!] = f \quad [\![\Gamma \vdash s : A \to \Omega]\!] = g}{[\![\Gamma \vdash s(t) : \Omega]\!] = \epsilon \circ \langle g, f\rangle},$$

where ϵ is the usual evaluation morphism. Fixpoints $\Gamma \vdash \eta X.\, \phi$ with $\eta \in \{\mu, \nu\}$ are interpreted using the fact that the hom-sets $\mathsf{Hom}([\![\Gamma]\!] \times [\![A]\!], [\![\Omega]\!])$ are complete lattices. In detail, the interpretation of $\Gamma, X : A \to \Omega \vdash \phi : \Omega$ is a morphism $g : [\![\Gamma]\!] \times ([\![P1]\!])^{[\![A]\!]} \to ([\![P1]\!])^{[\![A]\!]}$. We thus define an endomorphism F on $\mathsf{Hom}([\![\Gamma]\!] \times [\![A]\!], [\![\Omega]\!])$ by taking $F(h)$ to be

$$[\![\Gamma]\!] \times [\![A]\!] \xrightarrow{\langle \mathsf{id}, \lambda h \circ \pi_1\rangle} [\![\Gamma]\!] \times [\![A]\!] \times (P1)^{[\![A]\!]} \xrightarrow{\langle g \circ \langle \pi_1, \pi_3\rangle, \pi_2\rangle} (P1)^{[\![A]\!]} \times [\![A]\!] \xrightarrow{\epsilon} [\![\Omega]\!].$$

We then take $[\![\Gamma \vdash \eta X.\, \phi]\!]$ to be $\lambda \eta F$; the required fixpoints exist by the Knaster-Tarski theorem as all assertions are monotone in their variables.

Definition 7. A judgment $\Gamma \vdash \phi \sqsubseteq \psi$ is *valid over* $\mathbb{P}$ if $[\![\phi]\!] \sqsubseteq [\![\psi]\!]$.

6 Hoare Calculus

In the exception-free case, a semantics of Hoare triples over order-enriched monads [7] is defined by taking $\{\phi\}\, x \leftarrow p\, \{\psi\}$ (with $p : T_E A$ and assertions ϕ, ψ, where ψ may depend on $x : A$) to abbreviate the equation

$$\mathsf{do}\ \phi;\ x \leftarrow p; \psi;\ \mathsf{ret}\, x = \mathsf{do}\ \phi;\ p.$$

As indicated previously, this setup is insufficient for the verification of exception-raising programs; e.g. nothing useful can be said about raise beyond $\{\top\}\ \mathsf{raise}\ e\ \{\bot\}$. We thus need an additional postcondition for abnormal termination [9,18]: Intuitively, a *Hoare quadruple*

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

with *abnormal postcondition* $\varepsilon : E \to \Omega$ says that if p is executed in a prestate satisfying ϕ and terminates normally with result x, then the poststate and x satisfy the *normal postcondition* ψ, and if p terminates with an exception e then the poststate satisfies $\varepsilon(e)$. Using the previous definition of Hoare triple, we formally interpret $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ by letting it abbreviate

$$\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\mathsf{case}\, y\, \mathsf{of}\ \mathsf{inl}\, x \mapsto \psi; \mathsf{inr}\, e \mapsto \varepsilon(e)\}.$$

We write $\mathbb{T}, \mathbb{P} \vDash \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ if the above equality holds in $(\mathbb{T}, \mathbb{P})$ (under the given interpretation of the basic programs, elided in the notation).

The inference rules for our calculus are shown in Fig. 3. The inequalities in **(wk)** refer to the notion of validity introduced above (Definition 7); inequality $\varepsilon \sqsubseteq \varepsilon'$ of abnormal postconditions is understood pointwise. We assume a set Δ of valid axioms for basic programs, and then write $\Delta \vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ if a Hoare quadruple $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ is derivable from axioms in Δ and inequalities of assertions valid over $\mathbb{P}$.

The rules **(basic)**, **(ret)**, **(do)** and **(wk)** are derived straightforwardly from the rules for the exception-free case [7]; rule **($\perp$)** is clear. We spend some time on discussing the remaining new rules:

Rules for exceptions are taken from [18]. The **(raise)** rule is self-explanatory. The **(catch)** rule is based on the fact that $\mathsf{catch}\, p : T_E(A + E)$ renormalizes the computation p and returns an exception possibly raised by p in the right-hand summand of the normal result.

Case rule. The precondition of a case statement is a disjunction of the preconditions for the two alternatives, with access to the value of the branching condition c within the two summands provided by the $c?$ construct introduced in Sect. 5. Soundness of the rule is proved by Lemma 6.

Iterated case rule. The full itcase construction, as defined in (3), is actually a sequential composition of two programs. For simplicity, we treat only the special case $p = \mathsf{ret}\, x$ in the Hoare calculus, which we abbreviate as

$$\mathsf{itercase}\, c\, \mathsf{of}\ \mathsf{inl}\, a \mapsto q; \mathsf{inr}\, b \mapsto r := \mathsf{init}\, x \leftarrow \mathsf{ret}\, x\, \mathsf{itercase}\, c\, \mathsf{of}\ \mathsf{inl}\, a \mapsto q; \mathsf{inr}\, b \mapsto r.$$

In contrast to the case of **while** loops [7], we cannot expect that after the loop, the value of the test term c contains a right injection, as c could be affected by the program r, which is executed after the loop terminates. The intuition behind the formulation of the **(itcase)** rule is the following: The assertion ψ plays the role of the loop invariant. In every iteration, the loop body q, which depends on a, is executed with the value of a extracted from c, and possibly changes the value of c. As a postcondition of q, we thus require that either the loop continues and the invariant ψ still holds for the next iteration, i.e. $\mathsf{do}\ a \leftarrow c?; \psi$ holds, or the loop terminates and the precondition of r holds, with b replaced by the value contained in c. As the loop terminates with an execution of r, the postcondition of the loop obtained in the conclusion of the rule is then just that of r.

Theorem 8 (Soundness). *The Hoare calculus for order-enriched effects with exceptions is sound, i.e.*

$$\Delta \vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow p\ \{\psi\}\ \textit{implies}\ \mathbb{T}_E, \mathbb{P} \vDash \{\phi\}\ x \leftarrow p\ \{\psi\}.$$

$$\textbf{(basic)}\ \frac{\{\phi\}\ x \leftarrow f(z)\ \{\psi \mid \varepsilon\}}{\{\phi[t/z]\}\ x \leftarrow f(t)\ \{\psi \mid \varepsilon\}} \qquad (\bot)\ \frac{}{\{\top\}\ \bot\ \{\bot \mid \lambda e.\,\bot\}}$$

$$\textbf{(ret)}\ \frac{}{\{\phi[t/x]\}\ x \leftarrow \mathsf{ret}(t)\ \{\phi \mid \lambda e.\,\bot\}} \qquad \textbf{(do)}\ \frac{\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} \quad \{\psi\}\ y \leftarrow q\ \{\chi \mid \varepsilon\}}{\{\phi\}\ y \leftarrow \mathsf{do}\ x \leftarrow p; q\ \{\chi \mid \varepsilon\}}$$

$$\textbf{(case)}\ \frac{\{\phi\}\ x \leftarrow q\ \{\psi \mid \varepsilon\} \quad \{\xi\}\ x \leftarrow r\ \{\psi \mid \varepsilon\}}{\{(\mathsf{do}\ a \leftarrow c?; \phi) \vee (\mathsf{do}\ b \leftarrow \bar{c}?; \xi)\}\ x \leftarrow \mathsf{case}\, c\, \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r\ \{\psi \mid \varepsilon\}}$$

$$\textbf{(wk)}\ \frac{\phi' \sqsubseteq \phi \quad \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} \quad \psi \sqsubseteq \psi' \quad \varepsilon \sqsubseteq \varepsilon'}{\{\phi'\}\ x \leftarrow p\ \{\psi' \mid \varepsilon'\}} \qquad \textbf{(raise)}\ \frac{}{\{\varepsilon(e)\}\ x \leftarrow \mathsf{raise}\, e\ \{\bot \mid \varepsilon\}}$$

$$\textbf{(catch)}\ \frac{\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}}{\{\phi\}\ y \leftarrow (\mathsf{catch}\, p)\ \{\mathsf{case}\, y\, \mathsf{of}\ \mathsf{inl}\, x \mapsto \psi;\ \mathsf{inr}\, e \mapsto \varepsilon(e) \mid \lambda e.\,\bot\}}$$

$$\textbf{(itcase)}\ \frac{\{\psi\}\ x \leftarrow q\ \{\mathsf{do}\ a \leftarrow c?; \psi \vee \mathsf{do}\ b \leftarrow \bar{c}?; \xi \mid \varepsilon\} \qquad \{\xi\}\ x \leftarrow r\ \{\chi \mid \varepsilon\}}{\{\mathsf{do}\ a \leftarrow c?; \psi \vee \mathsf{do}\ b \leftarrow \bar{c}?; \xi\}\ x \leftarrow (\mathsf{itercase}\, c\, \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r)\ \{\chi \mid \varepsilon\}}$$

Fig. 3. Hoare calculus for order-enriched effects with exceptions.

The main idea in Cook's original proof of relative completeness of ordinary Hoare logic is a calculus of *weakest liberal preconditions* or shorter *weakest preconditions* [5], that is (adapted to our setting), for every program p, postcondition ψ and abnormal postcondition ε, a precondition $\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ such that

$$\{\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} \tag{4}$$

is a valid Hoare quadruple and $\phi \sqsubseteq \mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ whenever $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$. If we can prove that all quadruples (4) are derivable in the calculus then relative completeness follows by soundness and **(wk)**.

Semantically, we construct weakest preconditions in the obvious way by

$$\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon) = \bigsqcup\{\phi \mid \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}\}$$

The join exists because every $\mathsf{Hom}(X, \Omega)$ is a complete lattice. The definition of a weakest precondition $\mathsf{wp}(x \leftarrow p, \psi)$ for $\mathbb{T}$ is recovered by taking $E = 0$. By the continuity requirements for order-enriched monads, we have

Lemma 9. *For all programs p and postconditions $\psi \mid \varepsilon, \mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ is a weakest precondition.*

It remains to show that weakest preconditions are definable in our assertion language under the assumption that the weakest preconditions of all signature symbols exist. To this end we give an inductive definition of *syntactic* weakest preconditions $\mathsf{wp}_s(x \leftarrow p, \psi \mid \varepsilon)$, extending the corresponding calculus for the exception-free case [7], and prove that the definition of wp_s is sound w.r.t. wp. This requires a suitable restriction imposed on the underlying predicated monad, which we first recall for the exception-free case:

Definition 10 (Sequential Compatibility [7]). We call a predicated monad $(\mathbb{T},\mathbb{P})$ *sequentially compatible* if for all programs p,q and assertions ψ, we have

$$\mathsf{wp}(x \leftarrow (\mathsf{do}\ y \leftarrow p; q), \psi) \sqsubseteq \mathsf{wp}(y \leftarrow p, \mathsf{wp}(x \leftarrow q, \psi)). \tag{5}$$

Remark 11. The inequality converse to (5) holds automatically, and therefore for sequentially compatible predicated monads

$$\mathsf{wp}(x \leftarrow (\mathsf{do}\ y \leftarrow p; q), \psi) = \mathsf{wp}(y \leftarrow p, \mathsf{wp}(x \leftarrow q, \psi)).$$

An equivalent formulation, called the *interpolation property*, is used by Bloom and Ésik [3]; it can be reformulated in our terms as follows: for every valid Hoare triple $\{\phi\}\ x \leftarrow (\mathsf{do}\ y \leftarrow p; q)\ \{\psi\}$ there exists ξ such that $\{\phi\}\ y \leftarrow p\ \{\xi\}$ and $\{\xi\}\ x \leftarrow q\ \{\psi\}$ (w.l.o.g. we can take ξ to be $\mathsf{wp}(x \leftarrow q, \psi)$).

Given $\phi : X \to \Omega$, we can form $\phi^\circ(p) = \mathsf{wp}(x \leftarrow p, \phi(x))$, which yields an operator $(-)^\circ : \mathsf{Hom}(X,\Omega) \to \mathsf{Hom}(TX,\Omega)$. A predicated monad $(\mathbb{T},\mathbb{P})$ is sequentially compatible iff the right of the following two conditions is satisfied (the left one is satisfied automatically) for all $\phi : Y \to \Omega, f : X \to TY$:

$$\phi^\circ \circ \eta = \phi, \qquad\qquad \phi^\circ \circ f^\star = (\phi^\circ \circ f)^\circ. \tag{6}$$

It follows that $(-)^\circ$ is natural in the domain of its argument, for $\phi^\circ \circ (Tf) = \phi^\circ \circ (\eta \circ f)^\star = (\phi^\circ \circ \eta \circ f)^\circ = (\phi \circ f)^\circ$, and therefore, by the Yoneda lemma, $(-)^\circ$ is uniquely induced by a map $\Box : T\Omega \to \Omega$. The axioms (6) then amount to saying that $(\Box, \Omega)$ is a $\mathbb{T}$-algebra. This relates sequential compatibility, introduced in [7], to later work by Hasuo [8] who called such an algebra a *modality*, under the running assumption that $\mathbb{T} = \mathbb{P}$, which we do not assume in general.

The inductive definition of syntactic weakest preconditions is given in Fig. 4. Sequential compatibility of $(\mathbb{T},\mathbb{P})$ generalizes to $(\mathbb{T}_E,\mathbb{P})$:

$$\mathsf{wp}_s(x \leftarrow f(t), \psi \mid \varepsilon) = \mathsf{wp}(x \leftarrow f(z), \psi \mid \varepsilon)[t/z] \tag{7}$$
$$\mathsf{wp}_s(x \leftarrow \mathsf{ret}\, t, \psi \mid \varepsilon) = \psi[t/x] \tag{8}$$
$$\mathsf{wp}_s(x \leftarrow \bot, \psi \mid \varepsilon) = \top \tag{9}$$
$$\mathsf{wp}_s(y \leftarrow (\mathsf{do}\ x \leftarrow p; q), \psi \mid \varepsilon) = \mathsf{wp}_s(x \leftarrow p, \mathsf{wp}_s(y \leftarrow q, \psi \mid \varepsilon) \mid \varepsilon) \tag{10}$$
$$\begin{aligned}&\mathsf{wp}_s(x \leftarrow (\mathsf{case}\, c\, \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r), \psi \mid \varepsilon) = \\ &\qquad \mathsf{do}\ a \leftarrow c?; \mathsf{wp}_s(x \leftarrow q, \psi \mid \varepsilon) \vee \mathsf{do}\ a \leftarrow \bar{c}?; \mathsf{wp}_s(x \leftarrow r, \psi \mid \varepsilon)\end{aligned} \tag{11}$$
$$\begin{aligned}&\mathsf{wp}_s(x \leftarrow (\mathsf{init}\, x \leftarrow \mathsf{ret}\, x\ \ \mathsf{itercase}\, c\, \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r), \psi \mid \varepsilon) = \\ &\qquad \big(\nu X.\, \lambda x.\ \mathsf{do}\ a\ \leftarrow c?; \mathsf{wp}_s(x \leftarrow q, X(x) \mid \varepsilon) \vee \\ &\qquad\qquad\qquad \mathsf{do}\ b\ \leftarrow \bar{c}?; \mathsf{wp}_s(x \leftarrow r, \psi \mid \varepsilon)\big)(x)\end{aligned} \tag{12}$$
$$\mathsf{wp}_s(x \leftarrow \mathsf{raise}\, e, \psi \mid \varepsilon) = \epsilon(e) \tag{13}$$
$$\mathsf{wp}_s(y \leftarrow \mathsf{catch}\, p, \psi \mid \varepsilon) = \mathsf{wp}_s(x \leftarrow p, \psi[\mathsf{inl}\, x/y] \mid \lambda e.\psi[\mathsf{inr}\, e/y]) \tag{14}$$

Fig. 4. Syntactic definition of weakest preconditions.

Lemma 12. *Given a sequentially compatible monad* $(\mathbb{T},\mathbb{P})$*, we have*

$$\mathsf{wp}(x \leftarrow (\mathsf{do}\ y \leftarrow p; q), \psi \mid \varepsilon) \sqsubseteq \mathsf{wp}(y \leftarrow p, \mathsf{wp}(x \leftarrow q, \psi \mid \varepsilon) \mid \varepsilon).$$

To understand the definition of wp_s for catch, first note that catch itself never terminates abnormally; by unfolding the definitions of Hoare quadruples and catch, we thus have that $\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\psi \mid \varepsilon\}$ is equivalent to the Hoare triple $\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\psi\}$. We can now eliminate catch by decomposing ψ into a case construction and applying the definition of Hoare quadruple:

$$\begin{aligned}
&\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\psi\}\\
&\Leftrightarrow \{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\mathsf{case}\, y\, \mathsf{of}\, \mathsf{inl}\, x \mapsto \psi[\mathsf{inl}\, x/y];\ \mathsf{inr}\, e \mapsto \psi[\mathsf{inr}\, e/y]\} && (15)\\
&\Leftrightarrow \{\phi\}\ x \leftarrow p\ \{\psi[\mathsf{inl}\, x/y] \mid \lambda e.\, \psi[\mathsf{inr}\, e/y]\}, && (16)
\end{aligned}$$

immediately leading to the definition of $\mathsf{wp}_s(y \leftarrow \mathsf{catch}\, p, \psi \mid \epsilon)$ shown in Fig. 4.

For basic programs $f \in \Sigma$, we need to assume that weakest preconditions $\mathsf{wp}(x \leftarrow f(z), \psi)$ for every ψ are expressible in the assertion language, and then include $\{\mathsf{wp}(x \leftarrow f(z), \psi)\}\ x \leftarrow f(z)\ \{\psi \mid \varepsilon\}$ as an axiom in Δ.

Lemma 13. *Given a sequentially compatible predicated monad, we have for all programs* p *and postconditions* ψ

$$\Delta \vdash_{\mathbb{P}} \{\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}.$$

Proof (Sketch). The proof is via induction over p. We carry out the interesting cases of the inductive step in detail:

Case $p = \mathsf{catch}\, q$*.* Just note that the rule **(catch)** essentially connects (15) and (16) in the above chain of equivalences explaining wp_s for catch, decorated with an (irrelevant) abnormal postcondition.

Case $p = \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, a \mapsto q; \mathsf{inr}\, b \mapsto r$*.* Let ξ denote the right-hand side of (12). By the induction hypothesis, $\{\mathsf{wp}_s(x \leftarrow q, \xi \mid \varepsilon)\}\ x \leftarrow q\ \{\xi \mid \varepsilon\}$ is derivable. This quadruple is equivalent to

$$\begin{aligned}\{\mathsf{wp}_s(x \leftarrow q, \xi \mid \varepsilon)\}\ x \leftarrow q\ \{&\mathsf{do}\ a \leftarrow c?; \mathsf{wp}_s(x \leftarrow q, \xi \mid \varepsilon) \vee\\ &\mathsf{do}\ b \leftarrow \bar{c}?; \mathsf{wp}_s(x \leftarrow r, \psi \mid \varepsilon)\}.\end{aligned}$$

We also have $\{\mathsf{wp}_s(x \leftarrow r, \psi \mid \varepsilon)\}\ x \leftarrow r\ \{\psi \mid \varepsilon\}$ by induction. The required quadruple is obtained directly by applying **(itcase)** to the previous two quadruples. ☐

By Lemma 13 and soundness, $\mathsf{wp}_s(x \leftarrow p, \psi \mid \varepsilon) \sqsubseteq \mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ for all p, ψ, ϵ. Conversely, we have

Lemma 14. *For all* p, ψ, ϵ*,* $\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon) \sqsubseteq \mathsf{wp}_s(x \leftarrow p, \psi \mid \varepsilon)$*.*

Proof (Sketch). Induction over p. Again, we only detail the interesting cases:

Case $p = \mathsf{itercase}\, c\, \mathsf{of}\ \mathsf{inl}\, a \mapsto q; \mathsf{inr}\, b \mapsto r$. Let ξ denote the right-hand side of (12). Following [7], we need to prove that $\phi_0 := \mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ is a postfixed point of the functional defining ξ.

By the definition of weakest preconditions, this amounts to showing that every ϕ satisfying $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ is smaller than the functional evaluated at ϕ_0, i.e.

$$\phi_0 \sqsubseteq \mathsf{do}\ a \leftarrow c?; \mathsf{wp}_s(x \leftarrow q, \phi_0 \mid \varepsilon) \vee \mathsf{do}\ a \leftarrow \bar{c}?; \mathsf{wp}_s(x \leftarrow r, \psi \mid \varepsilon).$$

Therefore, by case distinction, we need to show that

$$\begin{aligned} c? \wedge \phi_0 &\sqsubseteq \mathsf{do}\ a \leftarrow c?; \mathsf{wp}_s(x \leftarrow q, \phi_0 \mid \varepsilon) \\ \bar{c}? \wedge \phi_0 &\sqsubseteq \mathsf{do}\ b \leftarrow \bar{c}?; \mathsf{wp}_s(x \leftarrow r, \psi \mid \varepsilon). \end{aligned}$$

– *Continuation branch.* Unrolling the first iteration of the loop, we see that $\{c? \wedge \phi\}\ x \leftarrow (\mathsf{do}\ a \leftarrow c?; x \leftarrow q; p)\ \{\psi \mid \varepsilon\}$ holds and thus

$$\begin{aligned} c? \wedge \phi &\sqsubseteq \mathsf{wp}(x \leftarrow (\mathsf{do}\ a \leftarrow c?; x \leftarrow q; p), \psi \mid \varepsilon) \\ &\sqsubseteq \mathsf{do}\ a \leftarrow c?; \mathsf{wp}(\mathsf{do}\ x \leftarrow q; p, \psi \mid \varepsilon). \end{aligned}$$

By sequential compatibility, $\mathsf{wp}(x \leftarrow (\mathsf{do}\, x \leftarrow q; p), \psi \mid \varepsilon) \sqsubseteq \mathsf{wp}(x \leftarrow q, \mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon))$ and by induction $c? \wedge \phi \sqsubseteq \mathsf{do}\ a \leftarrow c?; \mathsf{wp}_s(x \leftarrow q, \psi_0 \mid \varepsilon)$.

– *Termination branch.* By definition of **(itcase)**, $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ decomposes sequentially into

$$\{\phi\}\ x \leftarrow p[\mathsf{ret} x/r]\ \{\bar{c}? \wedge \phi\} \text{ and } \{\bar{c}? \wedge \phi\}\ x \leftarrow (\mathsf{do}\ b \leftarrow \bar{c}?; r)\ \{\psi \mid \varepsilon\}.$$

From the latter quadruple, we have $\bar{c}? \wedge \phi \sqsubseteq \mathsf{wp}(x \leftarrow (\mathsf{do}\ b \leftarrow \bar{c}?; r), \psi \mid \varepsilon)$. Again, $\bar{c}? \wedge \phi \sqsubseteq \mathsf{do}\ b \leftarrow \bar{c}?; \mathsf{wp}_s(x \leftarrow r, \psi \mid \varepsilon)$ by sequential compatibility and by induction.

Case $p = \mathsf{catch}\, q$. As seen in the derivation of the syntactic weakest precondition for catch, the Hoare quadruple in question is equivalent to

$$\{\phi\}\ x \leftarrow q\ \{\psi[\mathsf{inl}\, x/y] \mid \lambda e.\, \psi[\mathsf{inr}\, e/y]\}$$

Thus, $\phi \sqsubseteq \mathsf{wp}(x \leftarrow q, \psi[\mathsf{inl}\, x/y] \mid \lambda e.\, \psi[\mathsf{inr}\, e/y])$ which is smaller than $\mathsf{wp}_s(x \leftarrow p, \psi \mid \varepsilon)$ by induction.

Case $p = \mathsf{raise}\, e$. Since $\mathsf{catch}\,(\mathsf{raise}\, e) = \mathsf{ret inr}\, e$,

$$\begin{aligned} &\mathsf{do}\,\phi;\ y \leftarrow \mathsf{catch}(\mathsf{raise}\, e); (\mathsf{case}\, y\, \mathsf{of}\ \mathsf{inl}\, x \mapsto \psi; \mathsf{inr}\, e \mapsto \varepsilon(e)); \mathsf{ret}\, y = \mathsf{do}\,\phi;\ \mathsf{ret\ inr}\, e \\ &\qquad\qquad\qquad\qquad \Leftrightarrow \quad \mathsf{do}\,\phi; \varepsilon(e); \mathsf{ret\ inr}\, e = \mathsf{do}\,\phi;\ \mathsf{ret\ inr}\, e. \end{aligned}$$

So, $\phi \sqcap \varepsilon(e) = \phi$, and therefore $\phi \sqsubseteq \varepsilon(e) = \mathsf{wp}_s(x \leftarrow \mathsf{raise}\, e, \psi \mid \varepsilon)$. □

We are now ready to prove our main result:

Theorem 15 (Relative completeness). *Let* $(\mathbb{T}_E, \mathbb{P})$ *be a sequentially compatible predicated monad and let the weakest preconditions for basic programs be expressible in the assertion language. Then*

$$\mathbb{T}_E, \mathbb{P} \models \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} \textit{ implies } \Delta_\Sigma \vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\},$$

where Δ_Σ *are axioms for the basic programs in the signature* Σ*.*

Proof. By Lemmas 13 and 14, we can express the weakest precondition $\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ in the assertion language, and

$$\Delta_\Sigma \vdash_{\mathbb{P}} \{\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}.$$

By the definition of $\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$, we thus have $\phi \sqsubseteq \mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$, so we can use **(wk)** to derive the required quadruple. $\square$

7 Conclusion and Further Work

We have extended a previous monad-based generic Hoare calculus [7] to cover computations raising exceptions, modelled in terms of the exception monad transformer [4]. To this end, we have added *abnormal postconditions* [9] to the system, which govern poststates reached by the program in case it terminates with an exception. Our framework is based on *order-enriched monads*, which provide the requisite semantic infrastructure to support loops while avoiding the assumption that the underlying *category of data types* is enriched over complete partial orders; consequently, our generic logic applies also to monads that live on categories with less structure, in particular the category of sets. Moreover, we equip the underlying monad with the choice of a submonad of *innocent* computations causing only restricted side-effects; this choice induces an object of truth values, in the shape of the type of innocent computations of unit type, that supports a form of geometric logic in which we phrase our assertion language.

We have proved soundness and relative completeness, the latter by giving a calculus of weakest (liberal) preconditions, which turn out to be expressible in spite of the fact that the assertion language is quite weak. Here, a key role is played by fixpoint constructs in the assertion language.

Exceptions can be regarded as a very simple case of *uninterpreted operations*; in future research, we will explore the possibility of extending the framework to cover more general uninterpreted operations, such as process-algebraic action prefixing. Moreover, we will investigate a more general setup where loops are interpreted by axiomatic iteration in the spirit of *complete Elgot monads* [1,6], thus in particular covering also coinductive resumption monads [14].

References

1. Adámek, J., Milius, S., Velebil, J.: Equational properties of iterative monads. Inf. Comput. **208**(12), 1306–1348 (2010)

2. Barr, M., Wells, C. (eds.): Category Theory for Computing Science, 2nd edn. Prentice Hall International (UK) Ltd., London (1995)
3. Bloom, S.L., Ésik, Z.: Iteration Theories: The Equational Logic of Iterative Processes. Springer-Verlag New York, Inc., New York (1993). https://doi.org/10.1007/978-3-642-78034-9
4. Cenciarelli, P., Moggi, E.: A syntactic approach to modularity in denotational semantics. In: Category Theory and Computer Science, CTCS 1993 (1993)
5. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM **18**(8), 453–457 (1975)
6. Goncharov, S., Rauch, C., Schröder, L.: Unguarded recursion on coinductive resumptions. In: Mathematical Foundations of Programming Semantics, MFPS 2015. ENTCS (2015)
7. Goncharov, S., Schröder, L.: A relatively complete Hoare logic for order-enriched effects. In: Logic in Computer Science, LICS 2013, pp. 273–282. IEEE (2013)
8. Hasuo, I.: Generic weakest precondition semantics from monads enriched with order. Theoret. Comput. Sci. **604**, 2–29 (2015)
9. Huisman, M., Jacobs, B.: Java program verification via a Hoare logic with abrupt termination. In: Maibaum, T. (ed.) FASE 2000. LNCS, vol. 1783, pp. 284–303. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46428-X_20
10. Moggi, E.: A modular approach to denotational semantics. In: Pitt, D.H., Curien, P.-L., Abramsky, S., Pitts, A.M., Poigné, A., Rydeheard, D.E. (eds.) CTCS 1991. LNCS, vol. 530, pp. 138–139. Springer, Heidelberg (1991). https://doi.org/10.1007/BFb0013462
11. Moggi, E.: Notions of computation and monads. Inf. Comput. **93**, 55–92 (1991)
12. Nordio, M., Calcagno, C., Müller, P., Meyer, B.: A sound and complete program logic for Eiffel. In: Oriol, M., Meyer, B. (eds.) TOOLS EUROPE 2009. LNBIP, vol. 33, pp. 195–214. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02571-6_12
13. von Oheimb, D.: Hoare logic for Java in Isabelle/HOL. Concurr. Comput.: Pract. Exp. **13**, 1173–1214 (2001)
14. Piróg, M., Gibbons, J.: The coinductive resumption monad. In: Mathematical Foundations of Programming Semantics, MFPS 2014. ENTCS, vol. 308, pp. 273–288 (2014)
15. Plotkin, G., Power, J.: Algebraic operations and generic effects. Appl. Categ. Struct. **11**, 69–94 (2003)
16. Poetzsch-Heffter, A., Rauch, N.: Soundness and relative completeness of a programming logic for a sequential Java subset. Technical report, TU Kaiserlautern (2004)
17. Schröder, L., Mossakowski, T.: Monad-independent dynamic logic in HasCasl. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) WADT 2002. LNCS, vol. 2755, pp. 425–441. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40020-2_25
18. Schröder, L., Mossakowski, T.: Generic exception handling and the Java monad. In: Rattray, C., Maharaj, S., Shankland, C. (eds.) AMAST 2004. LNCS, vol. 3116, pp. 443–459. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27815-3_34
19. Simpson, A.K.: Recursive types in Kleisli categories. Technical report, MFPS Tutorial, April 2007 (1992)
20. Vickers, S.: Topology via Logic. Cambridge University Press, Cambridge (1989)
21. Wadler, P.: How to declare an imperative. ACM Comput. Surv. **29**, 240–263 (1997)

Author Index