Boris Konev
Frank Wolter (Eds.)

LNAI 4720

Frontiers of Combining Systems

6th International Symposium, FroCoS 2007
Liverpool, UK, September 2007
Proceedings

Springer

Boris Konev Frank Wolter (Eds.)

Frontiers of Combining Systems

6th International Symposium, FroCoS 2007
Liverpool, UK, September 10-12, 2007
Proceedings

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Boris Konev
Frank Wolter
University of Liverpool
Department of Computer Science
Ashton Building, Liverpool L69 3BX, UK
E-mail: {B.Konev; F.Wolter}@csc.liv.ac.uk

Library of Congress Control Number: 2007933812

CR Subject Classification (1998): I.2.3, F.4.1, F.4

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN	0302-9743
ISBN-10	3-540-74620-X Springer Berlin Heidelberg New York
ISBN-13	978-3-540-74620-1 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12115235 06/3180 5 4 3 2 1 0

Lecture Notes in Artificial Intelligence 4720

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Lecture Notes in Artificial Intelligence (LNAI)

Vol. 4720: B. Konev, F. Wolter (Eds.), Frontiers of Combining Systems. X, 283 pages. 2007.

Vol. 4682: D.-S. Huang, L. Heutte, M. Loog (Eds.), Advanced Intelligent Computing Theories and Applications. XXVII, 1373 pages. 2007.

Vol. 4660: S. Džeroski, J. Todorovski (Eds.), Computational Discovery of Scientific Knowledge. X, 327 pages. 2007.

Vol. 4651: F. Azevedo, P. Barahona, F. Fages, F. Rossi (Eds.), Recent Advances in Constraints. VIII, 185 pages. 2007.

Vol. 4635: B. Kokinov, D.C. Richardson, T.R. Roth-Berghofer, L. Vieu (Eds.), Modeling and Using Context. XIV, 574 pages. 2007.

Vol. 4632: R. Alhajj, H. Gao, X. Li, J. Li, O.R. Zaïane (Eds.), Advanced Data Mining and Applications. XV, 634 pages. 2007.

Vol. 4626: R.O. Weber, M.M. Richter (Eds.), Case-Based Reasoning Research and Development. XIII, 534 pages. 2007.

Vol. 4617: V. Torra, Y. Narukawa, Y. Yoshida (Eds.), Modeling Decisions for Artificial Intelligence. XII, 502 pages. 2007.

Vol. 4612: I. Miguel, W. Ruml (Eds.), Abstraction, Reformulation, and Approximation. XI, 418 pages. 2007.

Vol. 4604: U. Priss, S. Polovina, R. Hill (Eds.), Conceptual Structures: Knowledge Architectures for Smart Applications. XII, 514 pages. 2007.

Vol. 4603: F. Pfenning (Ed.), Automated Deduction – CADE-21. XII, 522 pages. 2007.

Vol. 4597: P. Perner (Ed.), Advances in Data Mining. XI, 353 pages. 2007.

Vol. 4594: R. Bellazzi, A. Abu-Hanna, J. Hunter (Eds.), Artificial Intelligence in Medicine. XVI, 509 pages. 2007.

Vol. 4585: M. Kryszkiewicz, J.F. Peters, H. Rybinski, A. Skowron (Eds.), Rough Sets and Intelligent Systems Paradigms. XIX, 836 pages. 2007.

Vol. 4578: F. Masulli, S. Mitra, G. Pasi (Eds.), Applications of Fuzzy Sets Theory. XVIII, 693 pages. 2007.

Vol. 4573: M. Kauers, M. Kerber, R. Miner, W. Windsteiger (Eds.), Towards Mechanized Mathematical Assistants. XIII, 407 pages. 2007.

Vol. 4571: P. Perner (Ed.), Machine Learning and Data Mining in Pattern Recognition. XIV, 913 pages. 2007.

Vol. 4570: H.G. Okuno, M. Ali (Eds.), New Trends in Applied Artificial Intelligence. XXI, 1194 pages. 2007.

Vol. 4565: D.D. Schmorrow, L.M. Reeves (Eds.), Foundations of Augmented Cognition. XIX, 450 pages. 2007.

Vol. 4562: D. Harris (Ed.), Engineering Psychology and Cognitive Ergonomics. XXIII, 879 pages. 2007.

Vol. 4548: N. Olivetti (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods. X, 245 pages. 2007.

Vol. 4539: N.H. Bshouty, C. Gentile (Eds.), Learning Theory. XII, 634 pages. 2007.

Vol. 4529: P. Melin, O. Castillo, L.T. Aguilar, J. Kacprzyk, W. Pedrycz (Eds.), Foundations of Fuzzy Logic and Soft Computing. XIX, 830 pages. 2007.

Vol. 4520: M.V. Butz, O. Sigaud, G. Pezzulo, G. Baldassarre (Eds.), Anticipatory Behavior in Adaptive Learning Systems. X, 379 pages. 2007.

Vol. 4511: C. Conati, K. McCoy, G. Paliouras (Eds.), User Modeling 2007. XVI, 487 pages. 2007.

Vol. 4509: Z. Kobti, D. Wu (Eds.), Advances in Artificial Intelligence. XII, 552 pages. 2007.

Vol. 4496: N.T. Nguyen, A. Grzech, R.J. Howlett, L.C. Jain (Eds.), Agent and Multi-Agent Systems: Technologies and Applications. XXI, 1046 pages. 2007.

Vol. 4483: C. Baral, G. Brewka, J. Schlipf (Eds.), Logic Programming and Nonmonotonic Reasoning. IX, 327 pages. 2007.

Vol. 4482: A. An, J. Stefanowski, S. Ramanna, C.J. Butz, W. Pedrycz, G. Wang (Eds.), Rough Sets, Fuzzy Sets, Data Mining and Granular Computing. XIV, 585 pages. 2007.

Vol. 4481: J. Yao, P. Lingras, W.-Z. Wu, M. Szczuka, N.J. Cercone, D. Ślęzak (Eds.), Rough Sets and Knowledge Technology. XIV, 576 pages. 2007.

Vol. 4476: V. Gorodetsky, C. Zhang, V.A. Skormin, L. Cao (Eds.), Autonomous Intelligent Systems: Multi-Agents and Data Mining. XIII, 323 pages. 2007.

Vol. 4456: Y. Wang, Y.-m. Cheung, H. Liu (Eds.), Computational Intelligence and Security. XXIII, 1118 pages. 2007.

Vol. 4455: S. Muggleton, R. Otero, A. Tamaddoni-Nezhad (Eds.), Inductive Logic Programming. XII, 456 pages. 2007.

Vol. 4452: M. Fasli, O. Shehory (Eds.), Agent-Mediated Electronic Commerce. VIII, 249 pages. 2007.

Vol. 4451: T.S. Huang, A. Nijholt, M. Pantic, A. Pentland (Eds.), Artifical Intelligence for Human Computing. XVI, 359 pages. 2007.

Vol. 4441: C. Müller (Ed.), Speaker Classification. X, 309 pages. 2007.

Vol. 4438: L. Maicher, A. Sigel, L.M. Garshol (Eds.), Leveraging the Semantics of Topic Maps. X, 257 pages. 2007.

Vol. 4434: G. Lakemeyer, E. Sklar, D.G. Sorrenti, T. Takahashi (Eds.), RoboCup 2006: Robot Soccer World Cup X. XIII, 566 pages. 2007.

Vol. 4429: R. Lu, J.H. Siekmann, C. Ullrich (Eds.), Cognitive Systems. X, 161 pages. 2007.

Vol. 4428: S. Edelkamp, A. Lomuscio (Eds.), Model Checking and Artificial Intelligence. IX, 185 pages. 2007.

Vol. 4426: Z.-H. Zhou, H. Li, Q. Yang (Eds.), Advances in Knowledge Discovery and Data Mining. XXV, 1161 pages. 2007.

Vol. 4411: R.H. Bordini, M. Dastani, J. Dix, A.E.F. Seghrouchni (Eds.), Programming Multi-Agent Systems. XIV, 249 pages. 2007.

Vol. 4410: A. Branco (Ed.), Anaphora: Analysis, Algorithms and Applications. X, 191 pages. 2007.

Vol. 4399: T. Kovacs, X. Llorà, K. Takadama, P.L. Lanzi, W. Stolzmann, S.W. Wilson (Eds.), Learning Classifier Systems. XII, 345 pages. 2007.

Vol. 4390: S.O. Kuznetsov, S. Schmidt (Eds.), Formal Concept Analysis. X, 329 pages. 2007.

Vol. 4389: D. Weyns, H.V.D. Parunak, F. Michel (Eds.), Environments for Multi-Agent Systems III. X, 273 pages. 2007.

Vol. 4386: P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, E. Matson (Eds.), Coordination, Organizations, Institutions, and Norms in Agent Systems II. XI, 373 pages. 2007.

Vol. 4384: T. Washio, K. Satoh, H. Takeda, A. Inokuchi (Eds.), New Frontiers in Artificial Intelligence. IX, 401 pages. 2007.

Vol. 4371: K. Inoue, K. Satoh, F. Toni (Eds.), Computational Logic in Multi-Agent Systems. X, 315 pages. 2007.

Vol. 4369: M. Umeda, A. Wolf, O. Bartenstein, U. Geske, D. Seipel, O. Takata (Eds.), Declarative Programming for Knowledge Management. X, 229 pages. 2006.

Vol. 4343: C. Müller (Ed.), Speaker Classification. X, 355 pages. 2007.

Vol. 4342: H. de Swart, E. Orłowska, G. Schmidt, M. Roubens (Eds.), Theory and Applications of Relational Structures as Knowledge Instruments II. X, 373 pages. 2006.

Vol. 4335: S.A. Brueckner, S. Hassas, M. Jelasity, D. Yamins (Eds.), Engineering Self-Organising Systems. XII, 212 pages. 2007.

Vol. 4334: B. Beckert, R. Hähnle, P.H. Schmitt (Eds.), Verification of Object-Oriented Software. XXIX, 658 pages. 2007.

Vol. 4333: U. Reimer, D. Karagiannis (Eds.), Practical Aspects of Knowledge Management. XII, 338 pages. 2006.

Vol. 4327: M. Baldoni, U. Endriss (Eds.), Declarative Agent Languages and Technologies IV. VIII, 257 pages. 2006.

Vol. 4314: C. Freksa, M. Kohlhase, K. Schill (Eds.), KI 2006: Advances in Artificial Intelligence. XII, 458 pages. 2007.

Vol. 4304: A. Sattar, B.-h. Kang (Eds.), AI 2006: Advances in Artificial Intelligence. XXVII, 1303 pages. 2006.

Vol. 4303: A. Hoffmann, B.-h. Kang, D. Richards, S. Tsumoto (Eds.), Advances in Knowledge Acquisition and Management. XI, 259 pages. 2006.

Vol. 4293: A. Gelbukh, C.A. Reyes-Garcia (Eds.), MICAI 2006: Advances in Artificial Intelligence. XXVIII, 1232 pages. 2006.

Vol. 4289: M. Ackermann, B. Berendt, M. Grobelnik, A. Hotho, D. Mladenič, G. Semeraro, M. Spiliopoulou, G. Stumme, V. Svátek, M. van Someren (Eds.), Semantics, Web and Mining. X, 197 pages. 2006.

Vol. 4285: Y. Matsumoto, R.W. Sproat, K.-F. Wong, M. Zhang (Eds.), Computer Processing of Oriental Languages. XVII, 544 pages. 2006.

Vol. 4274: Q. Huo, B. Ma, E.-S. Chng, H. Li (Eds.), Chinese Spoken Language Processing. XXIV, 805 pages. 2006.

Vol. 4265: L. Todorovski, N. Lavrač, K.P. Jantke (Eds.), Discovery Science. XIV, 384 pages. 2006.

Vol. 4264: J.L. Balcázar, P.M. Long, F. Stephan (Eds.), Algorithmic Learning Theory. XIII, 393 pages. 2006.

Vol. 4259: S. Greco, Y. Hata, S. Hirano, M. Inuiguchi, S. Miyamoto, H.S. Nguyen, R. Słowiński (Eds.), Rough Sets and Current Trends in Computing. XXII, 951 pages. 2006.

Vol. 4253: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part III. XXXII, 1301 pages. 2006.

Vol. 4252: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part II. XXXIII, 1335 pages. 2006.

Vol. 4251: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part I. LXVI, 1297 pages. 2006.

Vol. 4248: S. Staab, V. Svátek (Eds.), Managing Knowledge in a World of Networks. XIV, 400 pages. 2006.

Vol. 4246: M. Hermann, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XIII, 588 pages. 2006.

Vol. 4223: L. Wang, L. Jiao, G. Shi, X. Li, J. Liu (Eds.), Fuzzy Systems and Knowledge Discovery. XXVIII, 1335 pages. 2006.

Vol. 4213: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Knowledge Discovery in Databases: PKDD 2006. XXII, 660 pages. 2006.

Vol. 4212: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Machine Learning: ECML 2006. XXIII, 851 pages. 2006.

Vol. 4211: P. Vogt, Y. Sugita, E. Tuci, C.L. Nehaniv (Eds.), Symbol Grounding and Beyond. VIII, 237 pages. 2006.

Vol. 4203: F. Esposito, Z.W. Raś, D. Malerba, G. Semeraro (Eds.), Foundations of Intelligent Systems. XVIII, 767 pages. 2006.

Vol. 4201: Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, E. Tomita (Eds.), Grammatical Inference: Algorithms and Applications. XII, 359 pages. 2006.

Preface

This volume contains the proceedings of the Sixth International Symposium on Frontiers of Combining Systems (FroCoS 2007) held September 10-12, 2007 in Liverpool, UK. Previously, FroCoS was organized in Munich (1996), Amsterdam (1998), Nancy (2000), Santa Margeritha Ligure near Genoa (2002), and Vienna (2005). In 2004 and 2006, FroCoS joined IJCAR, the International Joint Conference on Automated Reasoning. Like its predecessors, FroCoS 2007 offered a forum for the presentation and discussion of research activities on the combination, integration, analysis, modularization and interaction of formally defined systems, with an emphasis on logic-based ones. These issues are important in many areas of computer science, such as logic, computation, program development and verification, artificial intelligence, automated reasoning, constraint solving, declarative programming, and symbolic computation.

There were 31 submissions to FroCoS 2007. Each submission was reviewed by at least three Programme Committee members. After extensive discussion within the Programme Committee, 14 papers were accepted for presentation and publication in this volume. In addition to technical papers, the volume also includes four invited contributions by Sava Krstic (Intel Corporation, USA), Roberto Sebastiani (University of Trento, Italy), Viorica Sofronie-Stokkermans (Max-Planck-Institut für Informatik, Germany), and Michael Zakharyaschev (Birkbeck College London, UK).

Many people and institutions contributed to making FroCoS 2007 a success. We are indebted to the members of the Programme Committee and the additional refereees for the thorough reviewing work; the members of the FroCoS Steering Committee for their support, to Andrei Voronkov for free use of the EasyChair conference management system, to sponsorship from EPSRC, and to Dave Shield and Thelma Williams for their invaluable assistance in hosting this conference.

June 2007

Boris Konev
Frank Wolter

Conference Organization

Conference Chair

Boris Konev

Programme Chair

Frank Wolter

Programme Committee

Alessandro Armando
Franz Baader
Jacques Calmet
Silvio Ghilardi
Bernhard Gramlich
Deepak Kapur
Till Mossakowski
Joachim Niehren
Albert Oliveras
Dirk Pattinson
Silvio Ranise
Mark Reynolds
Christophe Ringeissen
Ulrike Sattler
Amilcar Sernadas
Cesare Tinelli
Luca Viganò

External Reviewers

Pedro Adao
Daniel Bond
Torben Braüner
Sylvain Conchon
Giovanna D'Agostino
F. Miguel Dionisio
Tim French
Olivier Gauwin
Isabelle Gnaedig
Guillem Godoy

Florent Jacquemard
Oliver Kutz
Giacomo Lenzi
Denis Lugiez
Christopher Lynch
Jacopo Mantovani
Paulo Mateus
Aart Middeldorp
Sara Negri
Andrei Popescu
Albert Rubio
Peter Schneider-Kamp
Lutz Schröder
Aaron Stump
Guido Tack
Rene Thiemann
Kumar Neeraj Verma
Dirk Walther

Table of Contents

Section 1. Invited Contributions

Section 2. Technical Papers

Architecting Solvers for SAT Modulo Theories: Nelson-Oppen with DPLL

Sava Krstić and Amit Goel

Strategic CAD Labs, Intel Corporation

Abstract. We offer a transition system representing a high-level but detailed architecture for SMT solvers that combine a propositional SAT engine with solvers for multiple disjoint theories. The system captures succintly and accurately all the major aspects of the solver's global operation: boolean search with across-the-board backjumping, communication of theory-specific facts and equalities between shared variables, and cooperative conflict analysis. Provably correct and prudently underspecified, our system is a usable ground for high-quality implementations of comprehensive SMT solvers.

1 Introduction

SMT solvers are fully automated theorem provers based on decision procedures. The acronym is for *Satisfiability Modulo Theories*, indicating that an SMT solver works as a satisfiability checker, with its decision procedures targeting queries from one or more logical theories. These proof engines have become vital in verification practice and hold an even greater promise, but they are still a challenge to design and implement. From the seminal *Simplify* [9] to the current state-of-the-art *Yices* [10], with notable exceptions such as *UCLID* [6], the prevailing wisdom has been that an SMT solver should contain a SAT solver for managing the boolean complexity of the input formula and several specialized solvers—linear arithmetic and "theory of uninterpreted functions" obbligato—that communicate by exchanging equalities between variables ("the Nelson-Oppen style"[15]). This much granted, there is a host of remaining design issues at various levels of abstraction, the response to which distinguishes one solver from another.

Our goal is to define the top-level architecture of an SMT solver as a mathematical object that can be grasped as a whole and fruitfully reasoned about. We want an abstract model that faithfully captures the intricacies of the solver's global operation—what is going on between the architectural components and what is going on inside the components that is essential for interaction. We achieve this goal by presenting the SMT solver as a non-deterministic transition system. The ten rules of our system (Figure 5) provide a rather detailed rational reconstruction of the mainstream SMT solvers, covering the mechanisms for boolean search, communication of theory-specific facts and equalities between shared variables, and global conflict analysis. The system provides a solid theoretical basis for implementations, which can explore various execution

B. Konev and F. Wolter (Eds.): FroCos 2007, LNAI 4720, pp. 1–27, 2007.

strategies, refinements and optimizations, assured of fundamental correctness as long as they "play by the rules".

Following the precursor [12] to this paper, we adopt a logic with parametric polymorphism as the natural choice for SMT solvers, emphasizing *cardinality constraints*—not the traditional *stable-infinity condition*—as an accurate expression of what matters for completeness of the Nelson-Oppen method in practice.[1] Our main results are the termination, soundness, and completeness theorems for our transition system.

Related Work. We were inspired mainly by the work of Nieuwenhuis, Oliveras, and Tinelli [16] on *abstract DPLL* and *abstract DPLL modulo theories*—transition systems that model a DPLL-style SAT solver [8] and an SMT solver that extends it with a solver for *one* theory. In the follow-up paper [3], the same authors with Barrett extend their system with features for "splitting on demand" and derive from it the $DPLL(T_1, \ldots, T_n)$ *architecture.* This architecture is closely related to our system NODPLL (Section 5), but is significantly less detailed and transparent. It refines DPLL *modulo a single (composite) theory* with appropriate purity requirements on some, but not all rules. In contrast, NODPLL is explicitly *modulo multiple theories*, with rules specifying actions of specific theory solvers and the solvers' interaction made vivid. For example, equality propagation is spelled out in NODPLL, but which solver in $DPLL(T_1, \ldots, T_n)$ derives $x = z$ from $x = y$ and $y = z$ is not clear. Another important difference is in the modeling of conflict analysis and it shows even if our systems are compared at the propositional (SAT solver) level. While [16] and [3] view confict analysis abstractly, tucking it in a general rule for backjumping, NODPLL has rules that directly cover its key steps: conflict detection, the subsequent sequence of "explanations", generation of the "backjump clause", and the actual backjump. In an SMT solver, in particular with multiple theories, conflict analysis is even more subtle than in a SAT solver, and the authors of [16] are the first to point out its pitfalls ("too new explanations") and identify a condition for its correct behavior. NODPLL neatly captures this condition as a guard of a rule.

Our work also builds on [7], which has a transition system modeling a Nelson-Oppen solver for multiple theories, but does not address the cooperation with the SAT solver. Formal models of SMT solvers that do handle a SAT solver together with more than one theory are given only in the paper [3] discussed above and earlier works [2], [5]. Barrett's architecture of *CVC Lite* as described in [2] is complex and too low-level for convenient analysis and application. The system $SMT(T_1 \cup T_2)$ of Bozzano et al. [5] describes in pseudo-code a particular approach for equality propagation taken by the *MathSAT* solver, which can be modeled in NODPLL; see Section 5.6.

Outline. Section 2 contains (termino)logical background as developed in [12], but divorcing the solver's polymorphic language from *HOL*, to emphasize that

[1] The justification for the presence of non-stably-infinite theories in the Nelson-Oppen framework is studied in recent papers [20,17,4]; in [12], it is shown that the concept of stable-infinity can be dismissed altogether.

$$\Sigma_{\mathsf{Eq}} = \langle \mathsf{Bool} \mid =^{\alpha^2 \to \mathsf{Bool}}, \mathsf{ite}^{[\mathsf{Bool},\alpha,\alpha]\to\alpha}, \mathsf{true}^{\mathsf{Bool}}, \mathsf{false}^{\mathsf{Bool}}, \neg^{\mathsf{Bool}\to\mathsf{Bool}}, \wedge^{\mathsf{Bool}^2\to\mathsf{Bool}}, \ldots \rangle$$
$$\Sigma_{\mathsf{UF}} = \langle \Rightarrow \mid @^{[\alpha\Rightarrow\beta,\alpha]\to\beta} \rangle$$
$$\Sigma_{\mathsf{Int}} = \langle \mathsf{Int} \mid 0^{\mathsf{Int}}, 1^{\mathsf{Int}}, (-1)^{\mathsf{Int}}, \ldots, +^{\mathsf{Int}^2\to\mathsf{Int}}, -^{\mathsf{Int}^2\to\mathsf{Int}}, \times^{\mathsf{Int}^2\to\mathsf{Int}}, \leq^{\mathsf{Int}^2\to\mathsf{Bool}}, \ldots \rangle$$
$$\Sigma_{\times} = \langle \times \mid \langle \text{-}, \text{-} \rangle^{[\alpha,\beta]\to\alpha\times\beta}, \mathsf{fst}^{\alpha\times\beta\to\alpha}, \mathsf{snd}^{\alpha\times\beta\to\beta} \rangle$$
$$\Sigma_{\mathsf{Array}} = \langle \mathsf{Array} \mid \mathsf{mk\_arr}^{\beta\to\mathsf{Array}(\alpha,\beta)}, \mathsf{read}^{[\mathsf{Array}(\alpha,\beta),\alpha]\to\beta}, \mathsf{write}^{[\mathsf{Array}(\alpha,\beta),\alpha,\beta]\to\mathsf{Array}(\alpha,\beta)} \rangle$$
$$\Sigma_{\mathsf{List}} = \langle \mathsf{List} \mid \mathsf{cons}^{[\alpha,\mathsf{List}(\alpha)]\to\mathsf{List}(\alpha)}, \mathsf{nil}^{\mathsf{List}(\alpha)}, \mathsf{head}^{[\mathsf{List}(\alpha),\alpha]\to\mathsf{Bool}}, \mathsf{tail}^{[\mathsf{List}(\alpha),\mathsf{List}(\alpha)]\to\mathsf{Bool}} \rangle$$

Fig. 1. Signatures for theories of some familiar datatypes. For space efficiency, the constants' arities are shown as superscripts. Σ_{Eq} contains the type operator Bool and standard LOGICAL CONSTANTS. All other signatures by definition contain Σ_{Eq}, but to avoid clutter we leave their Σ_{Eq}-part implicit. In Σ_{UF}, the symbol UF is for *uninterpreted functions* and the intended meaning of @ is the function application. The list functions head and tail are partial, so are represented as predicates in Σ_{List}.

parametricity is not tied to higher-order logic, even though it is most conveniently expressed there. In Section 3, we overview purification—a somewhat involved procedure in the context of parametric theories—and give a suitable form of the non-deterministic Nelson-Oppen combination theorem of [12]. Section 4 is a quick rendition of the core DPLL algorithm as a transition system covering the essential features of modern SAT solvers. Section 5 contains the description of our main transition system for modeling combined SMT solvers, the basic correctness results for it, and some discussion. All proofs are given in the appendix.

2 Preliminaries

We are interested in logical theories of common datatypes and their combinations (Figure 1). A datatype has its syntax and semantics; both are needed to define the theory of the datatype. We give a brief overview of the syntax and an informal sketch of semantics, referring to [12] for technical details.

Types. A set O of symbols called TYPE OPERATORS, each with an associated non-negative arity, and an infinite set of TYPE VARIABLES define the set Tp_O of TYPES over O. It is the smallest set that contains type variables and expressions $F(\sigma_1, \ldots, \sigma_n)$, where $F \in O$ has arity n and $\sigma_i \in \mathsf{Tp}_O$.

A TYPE INSTANTIATION is a finite map from type variables to types. For any type σ and type instantiation $\theta = [\sigma_1/\alpha_1, \ldots, \sigma_n/\alpha_n]$, $\theta(\sigma)$ denotes the simultaneous substitution of every occurrence of α_i in σ with σ_i. We say that τ is an INSTANCE of σ if there is some θ such that $\tau = \theta(\sigma)$.

Signatures. A SIGNATURE is a pair $\langle O \mid K \rangle$, where O is a set of type operators and K is a set of CONSTANTS typed over O. By this we mean that every element of K has an ARITY, which is a tuple of types $(\sigma_0, \ldots, \sigma_n)$. Here, $\sigma_1, \ldots, \sigma_n$ are

the argument types of k, and σ_0 is its range type. Constants whose range type is Bool will be called PREDICATES.

We will use the more intuitive notation $k :: [\sigma_1, \ldots, \sigma_n] \to \sigma_0$ to indicate the arity of a constant. Moreover, we will write $k\colon [\tau_1, \ldots, \tau_n] \to \tau_0$ if there is a type instantiation that maps $\sigma_0, \ldots, \sigma_n$ to $\tau_0, \ldots, \tau_n$ respectively. Note the use of $::$ and $:$ for the "principal type" and "type instance" of k respectively. Also note that arities are not types—the symbol $\to$ is not a type operator.[2]

Terms. For a given signature $\Sigma = \langle O \mid K \rangle$ and every $\sigma \in \mathsf{Tp}_O$, we assume there is an infinite set of *variables of type* σ; we write them in the (name,type)-form v^σ. The sets Tm_σ of Σ-TERMS OF TYPE σ are defined inductively by these rules:

(1) every variable v^σ is in Tm_σ
(2) if $t_1 \in \mathsf{Tm}_{\tau_1}, \ldots, t_n \in \mathsf{Tm}_{\tau_n}$ and $k\colon [\tau_1, \ldots, \tau_n] \to \tau_0$, then $k\, t_1 \ldots t_n \in \mathsf{Tm}_{\tau_0}$

Type instantiations act on terms: define $\theta(t)$ to be the term obtained by replacing every variable x^σ in t with $x^{\theta(\sigma)}$. If $t \in \mathsf{Tm}_\sigma$, then $\theta(t) \in \mathsf{Tm}_{\theta(\sigma)}$. We define $t' \sqsubseteq t$ to mean that $t' = \theta(t)$ for some θ, and we then say that t' is a TYPE INSTANCE of t and t is a TYPE ABSTRACTION of t'.

For every term t, there exists the MOST GENERAL ABSTRACTION t^{abs} characterized by: (1) $t \sqsubseteq t^{\mathrm{abs}}$; and (2) $t' \sqsubseteq t^{\mathrm{abs}}$ for every t' such that $t \sqsubseteq t'$. The term t^{abs} is unique up to renaming of type variables and can be obtained by erasing all type information from t and then applying a type inference algorithm. For type inference, see, e.g., [13].

Semantics. The type operators List and Array have arities one and two respectively. The meaning of List is a function of arity one (by abuse of notation, also denoted List) that given a set E as an argument produces the set $\mathsf{List}(E)$ of all lists with elements in E. The meaning of Array is a function that given two sets I and E as arguments produces the set $\mathsf{Array}(I, E)$ of arrays indexed by I with elements in E.

The meaning of polymorphic types is defined once we know the meaning of type operators. For example, the meaning of the type $\mathsf{Array}(\alpha, \mathsf{Array}(\alpha, \beta))$ is a function that given any two sets I and E (as interpretations of type variables α, β) produces the set $\mathsf{Array}(I, \mathsf{Array}(I, E))$. If there are no occurrences of type variables in a type (e.g., $\mathsf{List}(\mathsf{Bool} \times \mathsf{Int})$), then the meaning of that type is always the same set; if the set is finite, we call the type FINITE.

The meaning of a constant is an indexed family of functions. For example, the meaning of cons is the family $\{\mathsf{cons}_E \mid E \text{ is a set}\}$, where cons_E is a function that takes an argument in E and an argument in $\mathsf{List}(E)$ and produces a result in $\mathsf{List}(E)$.

The meanings of type operators and constants of a signature together determine a STRUCTURE for that signature. The structure gives meaning to all terms.

[2] In [12], the type and term languages asssociated with a signature were defined as subsets of the higher-order logic, where the function space type operator is primitive and so arities could be seen as types.

Consider $t = \mathsf{read}(\mathsf{write}(a^{\mathsf{Array}(\alpha,\beta)}, i^{\alpha}, x^{\beta}), j^{\alpha})$. Once α and β are interpreted as concrete sets (I and E, say) and interpretations for the variables a, i, x, j (elements of $\mathsf{Array}(I, E), I, E, I$ respectively) are given, the polymorphic term t becomes a well-defined element of E. In [12], which should be consulted for more details, this element is denoted $[\![t]\!]\langle\iota, \rho\rangle$, where ι and ρ together define an *environment* for t: ι maps the type variables α, β to sets I, E respectively, and ρ maps the variables a, i, x, j to elements of $\mathsf{Array}(I, E), I, E, I$ respectively.

As a boring exercise, the reader may furnish the signatures in Figure 1 with meanings of their type operators and constants, thus obtaining definitions of structures $\mathcal{T}_{\mathsf{Eq}}, \mathcal{T}_{\mathsf{UF}}, \mathcal{T}_{\mathsf{Int}}, \mathcal{T}_{\times}, \mathcal{T}_{\mathsf{Array}}, \mathcal{T}_{\mathsf{List}}$.

Satisfiability. A Σ-FORMULA is an element of $\mathsf{Tm}_{\mathsf{Bool}}$. If ϕ is a Σ-formula and $\mathcal{T}$ is a Σ-structure, we say that ϕ is SATISFIABLE in $\mathcal{T}$ if $[\![\phi]\!]\langle\iota, \rho\rangle = \mathsf{true}$ for some environment $\langle\iota, \rho\rangle$; this environment then is called a MODEL of ϕ. We also say that ϕ is VALID if $\neg\phi$ is unsatisfiable. Validity is denoted $\models_{\mathcal{T}} \phi$, and $\phi_1, \ldots, \phi_n \models_{\mathcal{T}} \phi$ is an abbreviation for $\models_{\mathcal{T}} \phi_1 \wedge \cdots \wedge \phi_n \supset \phi$. The THEORY of a structure is the set of formulas that are valid in it.

An ATOMIC Σ-FORMULA is either a propositional variable or a term of the form $k\, t_1 \ldots t_n$, where k is a predicate. A Σ-LITERAL is an atomic formula or its negation. A CLAUSE is a disjunction of literals. A QUERY is a conjunction of formulas. Clauses containing the same literals in different order are considered equal. (We think of clauses and queries as sets of literals and formulas respectively.) A CONVEX THEORY is defined by the property that if a set of literals implies a disjunction of equalities, then one of the disjuncts must be implied.

A CARDINALITY CONSTRAINT is an "equality" of the form $\alpha \doteq n$, where α is a type variable and n is a positive integer; an enviroment $\langle\iota, \rho\rangle$ satisfies this constraint if α is in the domain of ι and the cardinality of the set $\iota(\alpha)$ is n.

Combining Structures. Two signatures are DISJOINT if the only type operators and constants they share are those of Σ_{Eq}. If $\mathcal{T}_1, \ldots, \mathcal{T}_n$ are structures with pairwise disjoint signatures, then there is a well-defined *sum structure* $\mathcal{T} = \mathcal{T}_1 + \cdots + \mathcal{T}_n$; the semantics of its type operators and constants is defined by the structures they come from. The types and terms of each $\mathcal{T}_i$ are types and terms of $\mathcal{T}$ too. We will call them PURE, or i-PURE when we need to be specific. The attribute MIXED will be used for arbitrary terms and types of a sum structure.

Solvers. A SOLVER for a fragment of a theory is a sound and complete satisfiability checker for sets of formulas (QUERIES) in the fragment. A STRONG SOLVER checks satisfiability of queries that contain formulas and cardinality constraints.

In practice, theory solvers are built for queries consisting of literals only. The well-known argument that this is sufficient in general begins with the observation that every query Φ is equisatisfiable with one of the form $Q = \Phi_0 \cup \{p_1 \Leftrightarrow \phi_1, \ldots, p_n \Leftrightarrow \phi_n\}$, where the p_i are propositional variables, the ϕ_i are literals, and Φ_0 is a propositional query, the boolean skeleton of Φ. A truth assignment M to propositional variables that satisfies Φ_0 can be extended to a model for Φ if and only if the query of literals $Q_M = \{\phi'_1, \ldots, \phi'_n\}$ is $\mathcal{T}$-satisfiable, where ϕ'_i is

either ϕ_i or $\neg\phi_i$, depending on whether $M(p_i)$ is true or false. Thus, satisfiability of Φ is decided by checking if Q_M is satisfiable for some model M of Φ_0. This, of course, calls for a SAT solver to efficiently enumerate the models M.

Parametricity. There is uniformity in the way "polymorphic" functions like cons compute their results—a consequence of the fact that the definition of cons_E takes the set E as a parameter, making no assumptions about it. Precisely pinning down this uniformity concept is somewhat tricky and we content ourselves with definitions of *parametric type operators* and *parametric constants* that are most convenient for our purposes. They are needed for proper understanding of Theorem 1 below, but not for much else in this paper. Thus, the reader may safely proceed with only a cursory reading of the rest of this section.

Recall first that a relation between two sets A and B is a PARTIAL BIJECTION if it can be seen as a bijection between a subset of A and a subset of B. Define an n-ary set function F to be PARAMETRIC if it is functorial on partial bijections. This means that given any partial bijections $f_i\colon A_i \leftrightarrow B_i$, where $i = 1, \ldots, n$, there exists a partial bijection $F(f_1, \ldots, f_n)\colon F(A_1, \ldots, A_n) \leftrightarrow F(B_1, \ldots, B_n)$; moreover, there is a requirement that the identity and composition be preserved. That is, $F(id_{A_1}, \ldots, id_{A_n}) = id_{F(A_1,\ldots,A_n)}$ and $F(g_1 \circ f_1, \ldots, g_n \circ f_n) = F(g_1, \ldots, g_n) \circ F(f_1, \ldots, f_n)$, where $A_i \overset{f_i}{\leftrightarrow} B_i \overset{g_i}{\leftrightarrow} C_i$.

Consider a structure whose type operators are all parametric in the above sense and let $k :: [\sigma_1, \ldots, \sigma_n] \to \sigma_0$ be a constant of this structure. Observe that if $\alpha_1, \ldots, \alpha_m$ are all type variables that occur in the types σ_i, then any interpretation ι of type variables (that is, an assignment, for each i, of a set A_i to α_i) interprets each type σ_j as a set, say S_j, and interprets k as a function $k_\iota\colon S_1 \times \cdots \times S_n \to S_0$. Suppose now we have two interpretations ι, ι' for type variables, the first just as above, and the second with A'_i and S'_j in place of A_i and S_j. Suppose also that $f_i : A_i \leftrightarrow A'_i$ are partial bijections. Since the type operators of our structure are assumed parametric, there are induced partial bijections $g_j\colon S_j \leftrightarrow S'_j$, for $j = 0, \ldots, n$. We say that the constant k is PARAMETRIC if in this situation we have $g_0(k_\iota(x_1, \ldots, x_n)) = k_{\iota'}(g_1(x_1), \ldots, g_n(x_n))$ for every $x_1 \in \mathtt{dom}(f_1), \ldots, x_n \in \mathtt{dom}(f_n)$.

Finally, a PARAMETRIC STRUCTURE is required to have all type operators and all constants parametric. Our example structures $\mathcal{T}_{\mathsf{Eq}}, \mathcal{T}_{\mathsf{UF}}, \mathcal{T}_{\mathsf{Int}}, \mathcal{T}_{\times}, \mathcal{T}_{\mathsf{Array}}, \mathcal{T}_{\mathsf{List}}$, with the notable exception of $\mathcal{T}_{\mathsf{UF}}$, are all parametric; so are the structures describing sets, multisets, and arbitrary algebraic datatypes [12].

We should note that the well-known concept of REYNOLDS PARAMETRICITY in programming languages [18] is neither weaker nor stronger than the concept we are using. In particular, $\mathcal{T}_{\mathsf{UF}}$ is Reynolds parametric. See [12].

3 Purification and Non-deterministic Nelson-Oppen

In the untyped setting, to *purify* a query consisting of mixed formulas is to transform it into an equisatisfiable query consisting of pure formulas. The transformation iteratively replaces a pure subterm t in a mixed formula with a fresh