José Luiz Fiadeiro
Pierre-Yves Schobbens (Eds.)

LNCS 4409

Recent Trends in Algebraic Development Techniques

18th International Workshop, WADT 2006
La Roche en Ardenne, Belgium, June 2006
Revised Selected Papers

Springer

José Luiz Fiadeiro
Pierre-Yves Schobbens (Eds.)

Recent Trends in Algebraic Development Techniques

18th International Workshop, WADT 2006
La Roche en Ardenne, Belgium, June 1-3, 2006
Revised Selected Papers

Springer

Volume Editors

José Luiz Fiadeiro
University of Leicester
Department of Computer Science
University Road, Leicester LE1 7RH, UK
E-mail: jose@mcs.le.ac.uk

Pierre-Yves Schobbens
Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage 21, 5000 Namur, Belgium
E-mail: pys@info.fundp.ac.be

Library of Congress Control Number: 2007924494

CR Subject Classification (1998): F.3.1, F.4, D.2.1, I.1

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-71997-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-71997-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12049424 06/3180 5 4 3 2 1 0

Lecture Notes in Computer Science 4409

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

This volume contains selected papers from WADT 2006, the 18th International Workshop on Algebraic Development Techniques. Like its predecessors, WADT 2006 focussed on the algebraic approach to the specification and development of systems, an area that was born around the algebraic specification of abstract data types and encompasses today the formal design of software systems, new specification frameworks and a wide range of application areas.

WADT 2006 took place at Chateau Floréal, La-Roche-en-Ardenne, Belgium, June 1–3, 2006, and was organized by Pierre-Yves Schobbens.

The program consisted of invited talks by David Rosenblum (University College London, UK) and Hubert Comon-Lundh (ENS-Cachan, France), and 32 presentations describing ongoing research on main topics of the workshop: formal methods for system development, specification languages and methods, systems and techniques for reasoning about specifications, specification development systems, methods and techniques for concurrent, distributed and mobile systems, and algebraic and co-algebraic foundations.

The Steering Committee of WADT, consisting of Michel Bidoit, José Fiadeiro, Hans-Jörg Kreowski, Till Mossakowski, Peter Mosses, Fernando Orejas, Francesco Parisi-Presicce, and Andrzej Tarlecki, with the additional help of Pierre-Yves Schobbens and Martin Wirsing, selected several presentations and invited their authors to submit a full paper for possible inclusion in this volume. All submissions underwent a careful refereeing process. We are extremely grateful to the following additional referees for their help in reviewing the submissions: A. Borzyszkowski, F. Gadducci, G. Godoy, K. Hölscher, A. Kurz, S. Kuske, A. Lopes, W. Pawlowski, H. Reichel, U. Schmid, L. Schröder, M. Sebag, and H. Wiklicky.

This volume contains the final versions of the ten contributions that were accepted.

The workshop was jointly organized with IFIP WG 1.3 (Foundations of System Specification), and received generous sponsorship from the University of Namur (Facultés Universitaires Notre-Dame de la Paix).

January 2007

José Fiadeiro
Pierre-Yves Schobbens

Lecture Notes in Computer Science

For information about Vols. 1–4352

please contact your bookseller or Springer

Vol. 4453: T. Speed, H. Huang (Eds.), Research in Computational Molecular Biology. XVI, 550 pages. 2007. (Sublibrary LNBI).

Vol. 4450: T. Okamoto, X. Wang (Eds.), Public Key Cryptography – PKC 2007. XIII, 491 pages. 2007.

Vol. 4448: M. Giacobini (Ed.), Applications of Evolutionary Computing. XXIII, 755 pages. 2007.

Vol. 4447: E. Marchiori, J.H. Moore, J.C. Rajapakse (Eds.), Evolutionary Computation,Machine Learning and Data Mining in Bioinformatics. XI, 302 pages. 2007.

Vol. 4446: C. Cotta, J. van Hemert (Eds.), Evolutionary Computation in Combinatorial Optimization. XII, 241 pages. 2007.

Vol. 4445: M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, A.I. Esparcia-Alcázar (Eds.), Genetic Programming. XI, 382 pages. 2007.

Vol. 4444: T. Reps, M. Sagiv, J. Bauer (Eds.), Program Analysis and Compilation, Theory and Practice. X, 361 pages. 2007.

Vol. 4443: R. Kotagiri, P.R. Krishna, M. Mohania, E. Nantajeewarawat (Eds.), Advances in Databases: Concepts, Systems and Applications. XXI, 1126 pages. 2007.

Vol. 4440: B. Liblit, Cooperative Bug Isolation. XV, 101 pages. 2007.

Vol. 4433: E. Şahin, W.M. Spears, A.F.T. Winfield (Eds.), Swarm Robotics. XII, 221 pages. 2007.

Vol. 4432: B. Beliczynski, A. Dzielinski, M. Iwanowski, B. Ribeiro (Eds.), Adaptive and Natural Computing Algorithms, Part II. XXVI, 761 pages. 2007.

Vol. 4431: B. Beliczynski, A. Dzielinski, M. Iwanowski, B. Ribeiro (Eds.), Adaptive and Natural Computing Algorithms, Part I. XXV, 851 pages. 2007.

Vol. 4430: C.C. Yang, D. Zeng, M. Chau, K. Chang, Q. Yang, X. Cheng, J. Wang, F.-Y. Wang, H. Chen (Eds.), Intelligence and Security Informatics. XII, 330 pages. 2007.

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

Vol. 4427: S. Uhlig, K. Papagiannaki, O. Bonaventure (Eds.), Passive and Active Network Measurement. XI, 274 pages. 2007.

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

Vol. 4425: G. Amati, C. Carpineto, G. Romano (Eds.), Advances in Information Retrieval. XIX, 759 pages. 2007.

Vol. 4424: O. Grumberg, M. Huth (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. XX, 738 pages. 2007.

Vol. 4423: H. Seidl (Ed.), Foundations of Software Science and Computational Structures. XVI, 379 pages. 2007.

Vol. 4422: M.B. Dwyer, A. Lopes (Eds.), Fundamental Approaches to Software Engineering. XV, 440 pages. 2007.

Vol. 4421: R. De Nicola (Ed.), Programming Languages and Systems. XVII, 538 pages. 2007.

Vol. 4420: S. Krishnamurthi, M. Odersky (Eds.), Compiler Construction. XIV, 233 pages. 2007.

Vol. 4419: P.C. Diniz, E. Marques, K. Bertels, M.M. Fernandes, J.M.P. Cardoso (Eds.), Reconfigurable Computing: Architectures, Tools and Applications. XIV, 391 pages. 2007.

Vol. 4418: A. Gagalowicz, W. Philips (Eds.), Computer Vision/Computer Graphics Collaboration Techniques. XV, 620 pages. 2007.

Vol. 4416: A. Bemporad, A. Bicchi, G. Buttazzo (Eds.), Hybrid Systems: Computation and Control. XVII, 797 pages. 2007.

Vol. 4415: P. Lukowicz, L. Thiele, G. Tröster (Eds.), Architecture of Computing Systems - ARCS 2007. X, 297 pages. 2007.

Vol. 4414: S. Hochreiter, R. Wagner (Eds.), Bioinformatics Research and Development. XVI, 482 pages. 2007. (Sublibrary LNBI).

Vol. 4412: F. Stajano, H.J. Kim, J.-S. Chae, S.-D. Kim (Eds.), Ubiquitous Convergence Technology. XI, 302 pages. 2007.

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

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

Vol. 4409: J.L. Fiadeiro, P.-Y. Schobbens (Eds.), Recent Trends in Algebraic Development Techniques. VII, 171 pages. 2007.

Vol. 4407: G. Puebla (Ed.), Logic-Based Program Synthesis and Transformation. VIII, 237 pages. 2007.

Vol. 4406: W. De Meuter (Ed.), Advance in Smaltalk. VII, 157 pages. 2007.

Vol. 4405: L. Padgham, F. Zambonelli (Eds.), Agent-Oriented Software Engineering VII. XII, 225 pages. 2007.

Vol. 4403: S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, T. Murata (Eds.), Evolutionary Multi-Criterion Optimization. XIX, 954 pages. 2007.

Vol. 4401: N. Guelfi, D. Buchs (Eds.), Rapid Integration of Software Engineering Techniques. IX, 177 pages. 2007.

Vol. 4400: J.F. Peters, A. Skowron, V.W. Marek, E. Orłowska, R. Słowiński, W. Ziarko (Eds.), Transactions on Rough Sets VII, Part II. X, 381 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. (Sublibrary LNAI).

Vol. 4398: S. Marchand-Maillet, E. Bruno, A. Nürnberger, M. Detyniecki (Eds.), Adaptive Multimedia Retrieval: User, Context, and Feedback. XI, 269 pages. 2007.

Vol. 4397: C. Stephanidis, M. Pieper (Eds.), Universal Access in Ambient Intelligence Environments. XV, 467 pages. 2007.

Vol. 4396: J. García-Vidal, L. Cerdà-Alabern (Eds.), Wireless Systems and Mobility in Next Generation Internet. IX, 271 pages. 2007.

Vol. 4395: M. Daydé, J.M.L.M. Palma, Á.L.G.A. Coutinho, E. Pacitti, J.C. Lopes (Eds.), High Performance Computing for Computational Science - VECPAR 2006. XXIV, 721 pages. 2007.

Vol. 4394: A. Gelbukh (Ed.), Computational Linguistics and Intelligent Text Processing. XVI, 648 pages. 2007.

Vol. 4393: W. Thomas, P. Weil (Eds.), STACS 2007. XVIII, 708 pages. 2007.

Vol. 4392: S.P. Vadhan (Ed.), Theory of Cryptography. XI, 595 pages. 2007.

Vol. 4391: Y. Stylianou, M. Faundez-Zanuy, A. Esposito (Eds.), Progress in Nonlinear Speech Processing. XII, 269 pages. 2007.

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

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

Vol. 4385: K. Coninx, K. Luyten, K.A. Schneider (Eds.), Task Models and Diagrams for Users Interface Design. XI, 355 pages. 2007.

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

Vol. 4383: E. Bin, A. Ziv, S. Ur (Eds.), Hardware and Software, Verification and Testing. XII, 235 pages. 2007.

Vol. 4381: J. Akiyama, W.Y.C. Chen, M. Kano, X. Li, Q. Yu (Eds.), Discrete Geometry, Combinatorics and Graph Theory. XI, 289 pages. 2007.

Vol. 4380: S. Spaccapietra, P. Atzeni, F. Fages, M.-S. Hacid, M. Kifer, J. Mylopoulos, B. Pernici, P. Shvaiko, J. Trujillo, I. Zaihrayeu (Eds.), Journal on Data Semantics VIII. XV, 219 pages. 2007.

Vol. 4379: M. Südholt, C. Consel (Eds.), Object-Oriented Technology. VIII, 157 pages. 2007.

Vol. 4378: I. Virbitskaite, A. Voronkov (Eds.), Perspectives of Systems Informatics. XIV, 496 pages. 2007.

Vol. 4377: M. Abe (Ed.), Topics in Cryptology – CT-RSA 2007. XI, 403 pages. 2006.

Vol. 4376: E. Frachtenberg, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing. VII, 257 pages. 2007.

Vol. 4374: J.F. Peters, A. Skowron, I. Düntsch, J. Grzymała-Busse, E. Orłowska, L. Polkowski (Eds.), Transactions on Rough Sets VI, Part I. XII, 499 pages. 2007.

Vol. 4373: K. Langendoen, T. Voigt (Eds.), Wireless Sensor Networks. XIII, 358 pages. 2007.

Vol. 4372: M. Kaufmann, D. Wagner (Eds.), Graph Drawing. XIV, 454 pages. 2007.

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

Vol. 4370: P.P Lévy, B. Le Grand, F. Poulet, M. Soto, L. Darago, L. Toubiana, J.-F. Vibert (Eds.), Pixelization Paradigm. XV, 279 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. (Sublibrary LNAI).

Vol. 4368: T. Erlebach, C. Kaklamanis (Eds.), Approximation and Online Algorithms. X, 345 pages. 2007.

Vol. 4367: K. De Bosschere, D. Kaeli, P. Stenström, D. Whalley, T. Ungerer (Eds.), High Performance Embedded Architectures and Compilers. XI, 307 pages. 2007.

Vol. 4366: K. Tuyls, R. Westra, Y. Saeys, A. Nowé (Eds.), Knowledge Discovery and Emergent Complexity in Bioinformatics. IX, 183 pages. 2007. (Sublibrary LNBI).

Vol. 4364: T. Kühne (Ed.), Models in Software Engineering. XI, 332 pages. 2007.

Vol. 4362: J. van Leeuwen, G.F. Italiano, W. van der Hoek, C. Meinel, H. Sack, F. Plášil (Eds.), SOFSEM 2007: Theory and Practice of Computer Science. XXI, 937 pages. 2007.

Vol. 4361: H.J. Hoogeboom, G. Păun, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing. IX, 555 pages. 2006.

Vol. 4360: W. Dubitzky, A. Schuster, P.M.A. Sloot, M. Schroeder, M. Romberg (Eds.), Distributed, High-Performance and Grid Computing in Computational Biology. X, 192 pages. 2007. (Sublibrary LNBI).

Vol. 4358: R. Vidal, A. Heyden, Y. Ma (Eds.), Dynamical Vision. IX, 329 pages. 2007.

Vol. 4357: L. Buttyán, V. Gligor, D. Westhoff (Eds.), Security and Privacy in Ad-Hoc and Sensor Networks. X, 193 pages. 2006.

Vol. 4355: J. Julliand, O. Kouchnarenko (Eds.), B 2007: Formal Specification and Development in B. XIII, 293 pages. 2006.

Vol. 4354: M. Hanus (Ed.), Practical Aspects of Declarative Languages. X, 335 pages. 2006.

Vol. 4353: T. Schwentick, D. Suciu (Eds.), Database Theory – ICDT 2007. XI, 419 pages. 2006.

Table of Contents

Contributed Papers

A Temporal Graph Logic for Verification of Graph Transformation Systems*

Paolo Baldan[1], Andrea Corradini[2], Barbara König[3], and Alberto Lluch Lafuente[2]

[1] Dipartimento di Matematica Pura e Applicata, Università di Padova
baldan@math.unipd.it
[2] Dipartimento di Informatica, Università di Pisa
{andrea,lafuente}@di.unipi.it
[3] Abt. für Informatik und Ang. Kognitionswissenschaft, Universität Duisburg-Essen
barbara_koenig@uni-due.de

Abstract. We extend our approach for verifying properties of graph transformation systems using suitable abstractions. In the original approach properties are specified as formulae of a propositional temporal logic whose atomic predicates are monadic second-order graph formulae. We generalize this aspect by considering more expressive logics, where edge quantifiers and temporal modalities can be interleaved, a feature which allows, e.g., to trace the history of objects in time. After characterizing fragments of the logic which can be safely checked on the approximations, we show how the verification of the logic over graph transformation systems can be reduced to the verification of a logic over suitably defined Petri nets.

1 Introduction

Graph Transformation Systems (GTSs) are suitable modeling formalisms for systems involving aspects such as object-orientation, concurrency, mobility and distribution. The use of GTSs for the verification and analysis of such systems is still at an early stage, but there have been several proposals recently, either using existing model-checking tools [10,25] or developing new techniques [20,21]. A recent line of research [1,2,3,4,5] takes the latter approach and proposes a method inspired by abstract interpretation techniques. Roughly speaking, a GTS $\mathcal{R}$, whose state space might be infinite, is approximated by a finite structure $\mathcal{C}(\mathcal{R})$, called *covering* of $\mathcal{R}$. The covering is a Petri net-like structure, called Petri graph, and it approximates $\mathcal{R}$ in the sense that any graph G reachable in $\mathcal{R}$ has an homomorphic image reachable in $\mathcal{C}(\mathcal{R})$. In a sense, this reduces the verification of GTSs to the verification of Petri nets. One central feature of this approach is

* Research partially supported by the EU IST-2004-16004 SENSORIA, the MIUR PRIN 2005015824 ART, the DFG project SANDS and CRUI/DAAD VIGONI "Models based on Graph Transformation Systems: Analysis and Verification".

J.L. Fiadeiro and P.-Y. Schobbens (Eds.): WADT 2006, LNCS 4409, pp. 1–20, 2007.

the fact that it is a partial order reduction technique using unfoldings. That is, the interleaving of concurrent events—leading to state explosion—is avoided if possible.

In [5] a logic for the approximation approach is introduced, which is basically a propositional μ-calculus whose atomic predicates are closed formulae in a monadic second-order logic for graphs. Also, fragments of the logic are identified which are reflected by the approximations, i.e., classes of formulae which, when satisfied by the approximation, are satisfied by the original system as well. For the verification of such formulae, the logic is encoded into a μ-calculus whose atomic predicates are formulae over markings of a Petri net, allowing the reuse of existing model checking techniques for Petri nets [12].

There are other related papers working with graph logics, for instance [14]. However, most of them are based, like [5], on *propositional* temporal logics, that is, logics that do not allow to interleave temporal modalities with graph-related ones. Thus, properties like *a certain edge is never removed* are neither expressible nor verifiable. The only exceptions we are aware of are [20,22].

In this paper we extend the approach of [5] by considering a more expressive logic that allows to interleave temporal and graphical aspects. As we shall see, our temporal graph logic combines a monadic-second order logic of graphs with the μ-calculus. Formulae of our logic are interpreted over *graph transition systems* (GTrS), inspired by algebra transition systems [15] and the formalism of [20], which are traditional transition systems together with a function mapping states into graphs and transitions into partial graph morphisms. Graph transition systems are suitable formalisms for modeling the state space of graph transformation systems and Petri graphs. We introduce a notion of approximation for GTrSs, identifying fragments of the logic whose formulae are preserved or reflected by approximations. Then we show that the GTrS of the covering, as defined in [1], is an approximation of the GTrS of the original graph transformation system, thus providing a concrete way of constructing approximations. Finally, we propose an encoding for a fragment of our logic into a Petri net logic. Our encoding is correct and complete, i.e., a Petri graph satisfies a formula exactly when the encoding of the formula is satisfied by the underlying Petri net.

Putting all this together, given a graph transformation system $\mathcal{G}$ and a formula F in a suitable fragment of our logic, we can construct a Petri graph P which approximates $\mathcal{G}$, using the algorithm in [1]. Then F can be translated into a Petri net formula $[F]$, such that if N_P is the Petri net underlying P, then $N_P \models [F]$ implies $\mathcal{G} \models F$, i.e., we reduce verification over graph transformation systems to verification over Petri nets.

Section 2 introduces graphs, graph transformation systems and graph transition systems. Section 3 defines syntax and semantics of our temporal graph logic. Section 4 defines Petri graphs, the structures used for approximating graph transformation systems. Section 5 identifies fragments of the logic that are preserved or reflected by approximations. Section 6 proposes an encoding of a fragment of the logic into a Petri net logic. A last section concludes the paper and proposes further work.

2 Graph Transition Systems

An (edge-labeled) *graph* G is a tuple $G = \langle V_G, E_G, s_G, t_G, lab_G \rangle$ where V_G is a set of *nodes*, E_G is a set of *edges*, $s_G, t_G : E_G \to V_G$ are the *source* and *target* functions, and $lab_G : E_G \to \Lambda$ is a *labeling function*, where Λ is a fixed set of labels. Nodes and edges are sometimes called *items* and we shall write $X_G = E_G \cup V_G$ for the set of items of G.

The transformation of a graph into another by adding, removing or merging of items is suitably modeled by (partial) graph morphisms.

Definition 1 ((partial) graph morphism). *A* graph morphism $\psi : G_1 \to G_2$ *is a pair of mappings* $\psi_V : V_{G_1} \to V_{G_2}$, $\psi_E : E_{G_1} \to E_{G_2}$ *such that* $\psi_V \circ s_{G_1} = s_{G_2} \circ \psi_E$, $\psi_V \circ t_{G_1} = t_{G_2} \circ \psi_E$ *and* $lab_{G_1} = lab_{G_2} \circ \psi_E$. *A graph morphism* $\psi : G_1 \to G_2$ *is* injective *if so are* ψ_V *and* ψ_E; *it is* edge-injective *if* ψ_E *is injective. Edge-bijective morphisms are defined analogously. A graph* G' *is a* subgraph *of graph* G, *written* $G' \hookrightarrow G$, *if* $V_{G'} \subseteq V_G$ *and* $E_{G'} \subseteq E_G$, *and the inclusion is a graph morphism.*

A partial graph morphism $\psi : G_1 \rightharpoonup G_2$ *is a pair* $\langle G_1', \psi' \rangle$ *where* $G_1' \hookrightarrow G_1$ *is a subgraph of* G_1, *called the* domain *of* ψ, *and* $\psi' : G_1' \to G_2$ *is a graph morphism.*

Graph transformation is presented in set-theoretical terms, but could be equivalently presented by using the double-pushout [7] or single-pushout [11] approaches. With respect to more general definitions, our rules can neither delete nor merge nodes, and they have a discrete interface, i.e., the interface graph contains only nodes and thus edges are never preserved. Similar restrictions are assumed in [1]. While the condition of having a discrete interface can be relaxed, the deletion and merging of nodes is quite problematic in an unfolding-based approach.

Also observe that, as it commonly happens in the algebraic approaches to graph rewriting, we consider basic graph grammars, without any distinction between terminal and non-terminal symbols and without any high-level control mechanism. We remark that, even in this basic formulation, graph grammars are Turing complete (since they can simulate string rewriting).

Definition 2 (graph transformation system). *A* graph transformation system *(GTS)* $\mathcal{R}$ *is a pair* $\langle G_0, R \rangle$, *where* G_0 *is a* start graph *and* R *is a set of* rewriting rules *of the form* $r = \langle G_L, G_R, \alpha \rangle$, *where* G_L *and* G_R *are* left- *and* right-hand side *graphs, respectively, and* $\alpha : V_L \to V_R$ *is an injective function.*

A match *of a rule* r *in a graph* G *is a morphism* $\psi : G_L \to G$ *that is edge-injective. The application of a rule* r *to a match* ψ *in* G, *resulting in a new graph* H, *is called a* direct derivation *and is written* $G \xrightarrow{r,\psi} H$, *where* H *is defined as follows. The set* V_H *is* $V_G \uplus (V_R \setminus \alpha(V_L))$ *and* E_H *is* $(E_G \setminus \psi(E_L)) \uplus E_R$, *where* $\uplus$ *denotes disjoint union. The source, target and labeling functions are defined by*

$$\begin{array}{llll} s_H(e) = s_G(e) & t_H(e) = t_G(e) & lab_H(e) = lab_G(e) & \text{if } e \in (E_G \setminus \psi(E_L)), \\ s_H(e) = \overline{\psi}(s_R(e)) & t_H(e) = \overline{\psi}(t_R(e)) & lab_H(e) = lab_R(e) & \text{if } e \in E_R, \end{array}$$

where $\overline{\psi} : V_R \to V_H$ *satisfies* $\overline{\psi}(\alpha(v)) = \psi(v)$ *if* $v \in V_L$ *and* $\overline{\psi}(v) = v$, *otherwise.*

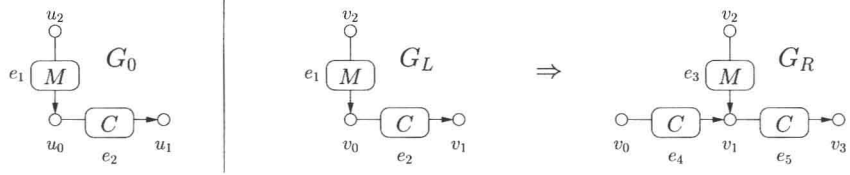


Fig. 1. A graph transformation system

Intuitively, the application of r to G at the match ψ first removes from G the image of the edges of L. Then the graph G is extended by adding the new nodes in G_R and the edges of G_R. All nodes in L are preserved.

A direct derivation $G \xrightarrow{r,\psi} H$ induces an obvious partial graph morphism $\tau_{G \xrightarrow{r,\psi} H} : G \rightharpoonup H$, injective and total on nodes, which maps items which are not deleted in G to corresponding items in H.

A *derivation* is a sequence of direct derivations starting from the start graph G_0. We write $G_0 \xrightarrow{*} H$ if there is a derivation ending in graph H, and we denote by $\mathbf{G}_{\mathcal{R}}$ the set of all graphs reachable in $\mathcal{R}$, i.e., $\mathbf{G}_{\mathcal{R}} = \{G \mid G_0 \xrightarrow{*} G\}$.

Example 1. Figure 1 depicts a GTS $\mathcal{G} = \langle G_0, \{r = \langle G_L, G_R, \alpha \rangle\} \rangle$ describing a simple message passing system. The start graph G_0 consists of three nodes u_0, u_1, u_2, one M-labeled edge e_1, representing a *message*, and one C-labeled edge e_2, representing a *connection*. The only rule consists of graphs G_L and G_R, and function α, which is the identity on V_L. The rule consumes the message and the connection, shifts the message to the successor node and recreates the connection. Furthermore a new C-labeled edge e_5 is created, along which the message can be passed in the next step. Note also that the source node of the message, representing its "identity", is preserved by the rule. In the rest of the paper we shall use this as a running example.

The state space of a GTS can be represented in a natural way as a transition system where the states are the reachable graphs and a transition between two states G and H exists whenever there is a direct derivation $G \xrightarrow{r,\psi} H$, as in [3]. However, such a structure would not be sufficient to interpret the logic introduced in the next section. Informally, since temporal modalities can be interleaved with quantification (over edges), the logic allows to trace the evolution of graph items over time and thus we need to represent explicitly which items of a graph are preserved by a rewriting step. To this aim, after recalling the standard definition of transition systems, we introduce below an enriched variant called *graph transition systems.*

Definition 3 (transition system). *A* transition system *is a tuple* $M = \langle S_M, T_M, in_M, out_M, s_0^M \rangle$ *where* S_M *is a set of* states, T_M *is a set of* transitions, $in_M, out_M : T_M \to S_M$ *are functions mapping each transition to its* start *and* end state, *and* $s_0^M \in S_M$ *is the* initial state. *We shall sometimes write* $s \xrightarrow{t} s'$

if $in_M(t) = s$ *and* $out_M(t) = s'$, *and* $s \xrightarrow{*} s'$ *if there exists a (possibly empty) sequence of transitions from* s *to* s'.

Correspondingly, a transition system morphism $h : M \to M'$ *is a pair of functions* $\langle h^S : S_M \to S_{M'}, h^T : T_M \to T_{M'} \rangle$ *such that the initial state as well as the start and end states of all transitions are preserved, i.e.,* $h^S(s_0^M) = s_0^{M'}$, $h^S \circ in_M = in_{M'} \circ h^T$, *and* $h^S \circ out_M = out_{M'} \circ h^T$.

A *graph transition system* is defined as a transition system together with a mapping which associates a graph with each state, and an injective partial graph morphism with each transition. We use the same name that is used in [20] for different, but closely related structures. The main difference is that in our case there is a clear distinction between the states and the graphs associated to the states: This leads below to a natural notion of morphism between graph transition systems, which will play a basic role in our definition of abstraction.

Definition 4 (graph transition system). *A* graph transition system *(GTrS)* $\mathcal{M}$ *is a pair* $\langle M, g \rangle$, *where* M *is a transition system and* g *is a pair* $g = \langle g^S, g^T \rangle$, *where* $g^S(s)$ *is a graph for each state* $s \in S_M$, *and* $g^T(t) : g^S(in_M(t)) \rightarrowtail g^S(out_M(t))$ *is an injective partial graph morphism for each transition* $t \in T_M$.

Note that the result of the application of a rule to a given match in a graph is determined only up to isomorphism, because of the use of disjoint union in the definition. Therefore, formally speaking, the state space of a GTS contains for each reachable graph G *all* graphs isomorphic to G as well. The next definition shows how to represent the state space of a GTS with a graph transition system (GTrS), where we get rid of such useless redundancy. Note that since the resulting GTrS is usually infinite-state, this construction is non-constructive and useless for practical purposes. We need the GTrS in order to define the semantics of the logic, but verification itself is done using a different method.

Definition 5 (graph transition system of a graph transformation system). *Given a GTS* $\mathcal{R} = \langle G_0, R \rangle$, *a GTrS representing its state space, denoted by* $\mathrm{GTrS}(\mathcal{R})$, *can be obtained as follows.*

1. *Consider first the graph transition system* $\langle M, g \rangle$, *where:* $S_M = \mathbf{G}_{\mathcal{R}}$ *(set of all graphs generated by* $\mathcal{R}$*);* $s_0^M = G_0$; $T_M = \{G \xrightarrow{r,\psi} H \mid G \xrightarrow{r,\psi} H$ *is a direct derivation of* $\mathcal{R}\}$; *and the mapping* $g = \langle g^S, g^T \rangle$ *is defined as follows:* $g^S(G) = G$ *and* $g^T(G \xrightarrow{r,\psi} H) = \tau_{G \xrightarrow{r,\psi} H} : G \rightarrowtail H$.
2. *Next, for each state* G *in* S_M *and for each pair* $\langle r, \psi \rangle$ *where* r *is a rule applicable to match* ψ *in* G, *choose one among the transitions leaving from* G *and using* r *and* ψ, *and delete from* T_M *all the remaining ones.*
3. *Finally,* $\mathrm{GTrS}(\mathcal{R})$ *is defined as the graph transition sub-system reachable from the start graph.*

Example 2. Figure 2 depicts a GTrS of the GTS depicted in Figure 1. Since state identities coincide with their corresponding graphs, the figure is simplified and we directly depict the graphs and partial graph morphisms. The leftmost state is

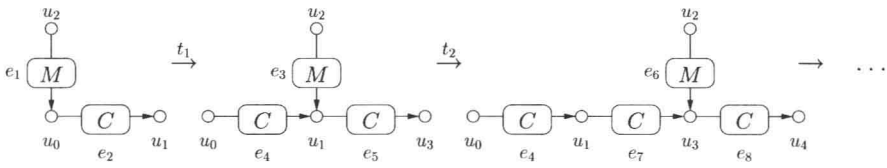


Fig. 2. A graph transition system

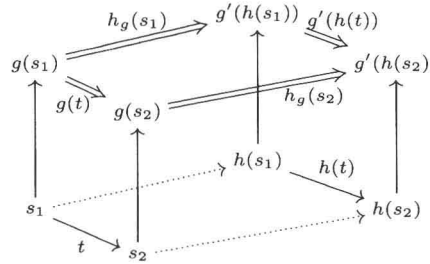
G_0, the initial state of both the GTS and its GTrS. Observe that for the second transition t_2, $g^T(t_2)_V$ (the component on nodes of $g^T(t_2)$) is an inclusion, while $g^T(t_2)_E$ is partial and is only defined on the edge e_4. All transitions correspond to different instances of the same rule.

The construction described in Definition 5 is clearly non-deterministic, because of step 2. Among the possible GTrSs associated with the GTS of Figure 1, the one drawn above enjoys some desirable properties: the partial injective morphisms associated with transitions are *partial inclusions* (i.e., every item preserves its name along rewriting), and edge and node names are not reused again in the computation after they have been deleted. The interpretation of the logic of Section 3 will be defined only over GTrSs satisfying such properties, and called *unraveled GTrSs*. For any GTrS $\mathcal{M}$ not satisfying these properties we shall consider an unraveled one which is behaviorally equivalent to $\mathcal{M}$, called its *unraveling*.

In order to characterize the unraveling of a GTrS we first need to introduce GTrS morphisms, which will also be used later for relating a system and its approximation. A morphism between two GTrSs consists of a morphism between the underlying transition systems, and, in addition, for each pair of related states, of a graph morphism between the graphs associated with such states. Furthermore, these graph morphisms must be consistent with the partial graph morphisms associated to the transitions.

Definition 6 (graph transition system morphism). *A* graph transition system morphism $h : \mathcal{M} \to \mathcal{M}'$ *from* $\mathcal{M} = \langle M, g \rangle$ *to* $\mathcal{M}' = \langle M', g' \rangle$ *is a pair* $\langle h_M, h_g \rangle$, *where* $h_M : M \to M'$ *is a transition system morphism, and for each state* $s \in S_M$, $h_g(s)$ *is a graph morphism from* $g^S(s)$ *to* $g'^S(h^S_M(s))$, *such that the following condition is satisfied: for each transition* $s_1 \xrightarrow{t} s_2 \in T_M$, $g'^T(h^T_M(t)) \circ h_g(s_1) = h_g(s_2) \circ g^T(t)$.

The diagram below illustrates the situation. The bottom square represents transition $s_1 \xrightarrow{t} s_2$ in M and its image through h_M in M' (sub- and super-scripts are avoided for the sake of readability). The vertical arrows of the left front square show how transition t is associated to a graph morphism via the g component of $\mathcal{M}$, and similarly for the back right square. Finally, the back left and front right sides of the top square are the components of the GTrS morphism associated to states s_1 and s_2, and the top square is required to commute.



Definition 7 (unraveled graph transition system). *A GTrS $\mathcal{M} = \langle M, g\rangle$ is* unraveled *whenever M is a tree, for each $t \in T_M$ the morphism $g^T(t) : g^S(in_M(t)) \rightarrow g^S(out_M(t))$ is a partial inclusion, and item names are not reused, i.e., for all $s', s'' \in S_M$, if $x \in X_{g^S(s')} \cap X_{g^S(s'')}$ there exists $s \in S_M$ such that*

$$x \in X_{g^S(s)} \wedge s \xrightarrow{*} s' \wedge s \xrightarrow{*} s'' \wedge g^T(s \xrightarrow{*} s')(x) = x \wedge g^T(s \xrightarrow{*} s'')(x) = x,$$

where $g^T(s \xrightarrow{*} s')$ is the composition of the partial morphisms associated with the transitions in $s \xrightarrow{*} s'$, which is uniquely determined since M is a tree.

An unraveling *of a GTrS $\mathcal{M} = \langle M, g\rangle$ is a pair $\langle \mathcal{M}', h\rangle$ where $\mathcal{M}'$ is an unraveled GTrS and $h = \langle h_M, h_g\rangle : \mathcal{M}' \rightarrow \mathcal{M}$ is a GTrS morphism, satisfying:*

1. *for each $s \in S_{M'}$, $h_g(s) : {g'}^S(s) \rightarrow g^S(h^S_M(s))$ is an isomorphism;*
2. *for each $s \in S_{M'}$ and transition $h^S_M(s) \xrightarrow{t'} s''$ in M, there is a transition $s \xrightarrow{t} s'$ in M' such that $h^T_M(t) = t'$ (and thus $h^S_M(s') = s''$).*

Proposition 1 (unraveling of a GTrS). *Any GTrS admits an unraveling.*

The conditions defining an unraveled GTrS $\mathcal{M}$ ensure that taking the union of all the graphs associated to the states in S_M, we obtain a well-defined graph. In fact, given any two states s and s' and an edge $e \in E_{g^S(s)} \cap E_{g^S(s')}$, the source, target and label of e coincide in $g^S(s)$ and $g^S(s')$. We shall denote the components of this "universe" graph as $\langle V_\mathcal{M}, E_\mathcal{M}, s_\mathcal{M}, t_\mathcal{M}, lab_\mathcal{M}\rangle$, where $V_\mathcal{M} = \bigcup_{s\in S_M} V_{g^S(s)}$, $E_\mathcal{M} = \bigcup_{s\in S_M} E_{g^S(s)}$, $s_\mathcal{M}(e) = s_{g^S(s)}(e)$ if $e \in g^S(s)$, and similarly for $t_\mathcal{M}$ and $lab_\mathcal{M}$.

3 A Temporal Graph Logic for Graph Transformation Systems

We now define syntax and semantics of our temporal graph logic, that extends the logic $\mu\mathcal{L}2$ of [3]. The logic is based both on the μ-calculus [6] and on second-order graph logic [8]. Let V_x, V_X, V_Z be sets of first- and second-order edge variables and propositional variables, respectively.

Definition 8 (syntax). *The logic $\mu\mathcal{G}2$ is given by the set of all formulae generated by:*

$$F ::= \eta(x) = \eta'(y) \mid x = y \mid l(x) = a \mid \neg F \mid F \vee F \mid \exists x.F \mid \exists X.F \mid x \in X \mid Z \mid \Diamond F \mid \mu Z.F$$

where $\eta, \eta' \in \{s, t\}$ (standing for source and target), $x, y \in V_x$, $X \in V_X$, $a \in \Lambda$ and $Z \in V_Z$. Furthermore $\Diamond F$ is the (existential) next-step operator. The letter μ denotes the least fixed-point operator. As usual the formula $\mu Z.F$ can be formed only if all occurrences of Z in F are positive, *i.e., they fall under an even number of negations. In the following we will use some (redundant) connectives like $\wedge$, $\forall$, $\Box$ and ν (greatest fixed-point), defined as usual. We denote by $\mu\mathcal{G}1$ its first-order fragment, where second-order edge variables and quantification are not allowed.*

Definition 9 (semantics of $\mu\mathcal{G}2$). *Let $\mathcal{M} = \langle M, g\rangle$ be an* unraveled *GTrS. The semantics of temporal graph formulae is given by an evaluation function mapping closed formulae into subsets of S_M, i.e., the states that satisfy the formula. We shall define a mapping $[\![\cdot]\!]^{\mathcal{M}}_{\sigma} : \mu\mathcal{G}2 \to 2^{S_M}$, where σ is an* environment, *i.e., a tuple $\sigma = \langle \sigma_x, \sigma_X, \sigma_Z\rangle$ of mappings from first- and second-order edge variables into edges and edge sets, respectively, and from propositional variables into subsets of S_M. More precisely, $\sigma_x : V_x \to E_{\mathcal{M}}$, $\sigma_X : V_X \to 2^{E_{\mathcal{M}}}$ and $\sigma_Z : V_Z \to 2^{S_M}$, where $E_{\mathcal{M}}$ is the set of all edge names used in $\mathcal{M}$. When $\mathcal{M}$ is implicit, we simply write $[\![\cdot]\!]_{\sigma}$.*

$$[\![\eta(x) = \eta'(y)]\!]_{\sigma} = \|\eta_{\mathcal{M}}(\sigma_x(x)) = \eta'_{\mathcal{M}}(\sigma_x(y))\| \qquad [\![x = y]\!]_{\sigma} = \|\sigma_x(x) = \sigma_x(y)\|$$
$$[\![l(x) = a]\!]_{\sigma} = \|lab_{\mathcal{M}}(\sigma_x(x)) = a\| \qquad [\![y \in Y]\!]_{\sigma} = \|\sigma_x(y) \in \sigma_X(Y)\|$$
$$[\![\neg F]\!]_{\sigma} = S_M \setminus [\![F]\!]_{\sigma} \qquad [\![F_1 \vee F_2]\!]_{\sigma} = [\![F_1]\!]_{\sigma} \cup [\![F_2]\!]_{\sigma}$$
$$[\![Z]\!]_{\sigma} = \sigma_Z(Z) \qquad [\![\mu Z.F]\!]_{\sigma} = \mathbf{lfp}(\lambda v.[\![F]\!]_{\sigma[v/Z]})$$

$$[\![\Diamond F]\!]_{\sigma} = \{s \in S_M \mid \exists s', t.\, s \xrightarrow{t} s' \wedge s' \in [\![F]\!]_{\sigma}\}$$
$$[\![\exists x.F]\!]_{\sigma} = \{s \in S_M \mid \exists e \in E_{g(s)}\,.\, s \in [\![F]\!]_{\sigma[e/x]}\}$$
$$[\![\exists X.F]\!]_{\sigma} = \{s \in S_M \mid \exists E \subseteq E_{g(s)}\,.\, s \in [\![F]\!]_{\sigma[E/X]}\}$$

where $\|\cdot\|$ maps true and false to S_M and $\emptyset$, respectively, $v \in 2^{S_M}$, and $\mathbf{lfp}(f)$ denotes the least fixed-point of the function f.

In particular, if F is a closed formula, we say that $\mathcal{M}$ satisfies *F and write $\mathcal{M} \models F$, if $s_0 \in [\![F]\!]_{\sigma}$, where σ is the empty environment. Finally, we say that a GTS $\mathcal{R} = \langle G_0, R\rangle$* satisfies *a closed formula F, written $\mathcal{R} \models F$, if the unraveling of $GTrS(\mathcal{R})$ satisfies F.*

The restriction to formulae where all occurrences of propositional variables are positive guarantees every possible function $\lambda v.[\![F]\!]_{\sigma[v/Z]}$ to be monotonic. Thus, by Knaster-Tarski theorem, fixed-points are well-defined.

Note that using unravelled GTrS is crucial for the definition of the semantics of the logic: items can be easily tracked since their identity is preserved and names are never reused. This allows to remember also the identities of deleted items, differently from what happens in the semantics given in [20,22].

Example 3. The following formula states that no M-labeled message edge is preserved by any transition: **M-consumed** $\equiv \neg\exists x.(l(x) = M \wedge \Diamond\exists y.\, x = y)$. The fact that this property holds in any reachable state is expressed by the formula: **always-M-consumed** $\equiv \nu Z.(\textbf{M-consumed} \wedge \Box Z)$. It is easy to see that **M-consumed** is satisfied by any state of the unraveled GTrS in Fig. 2, and thus $\mathcal{G} \models$ **always-M-consumed**, where $\mathcal{G}$ is the GTS of Fig. 1.

The formula **M-moves** $\equiv \neg\exists x.(l(x) = M \wedge \Diamond(\exists y.(l(y) = M \wedge t(y) = t(x) \wedge s(y) = s(x))))$ states that messages always move, i.e., there is no message edge such that in the next state there is another message edge with the same identity (i.e., source nodes coincide) attached to the same target node. And we can require this property to hold in every reachable state: **always-M-moves** $\equiv \nu Z.(\textbf{M-moves} \wedge \Box Z)$. Again, the GTS $\mathcal{G}$ satisfies this property. A GTS in which the message would at some point cross a "looping connection" or with more than one message would violate the formula.

4 Approximating GTSs with Petri Graphs

In the verification approach proposed in [1,3,4,5] *Petri graphs*, structures consisting of a Petri net and a graph component, have been introduced. They are used to represent finite approximations of the (usually infinite) unfolding of a GTS, on which to verify certain properties of the original system. Furthermore they provide a bridge to Petri net theory, allowing to reuse verification techniques developed for nets: a property expressed as a formula in the graph logic can be translated into an equivalent multiset formula to be verified on the net underlying the Petri graph. Here we shall concentrate on this latter aspect. We will not treat instead the construction of finite Petri graphs over-approximating GTSs, presented in [1,4] also for varying degrees of precision, recently enriched with a technique for counterexample-guided abstraction refinement [18], and for which the verification tool AUGUR (`http://www.ti.inf.uni-due.de/research/augur_1/`) has been developed.

Before introducing Petri graphs we need some definitions. Given a set A we will denote by $A^{\oplus}$ the free commutative monoid over A, whose elements will be called *multisets* over A. In the sequel we will sometimes identify $A^{\oplus}$ with the set of functions $m : A \to \mathbb{N}$ such that the set $\{a \in A \mid m(a) \neq 0\}$ is finite. E.g., in particular, $m(a)$ denotes the multiplicity of an element a in the multiset m. Sometimes a multiset will be identified with the underlying set, writing, e.g., $a \in m$ for $m(a) \neq 0$. Given a function $f : A \to B$, by $f^{\oplus} : A^{\oplus} \to B^{\oplus}$ we denote its monoidal extension, i.e., $f^{\oplus}(m)(b) = \sum_{f(a)=b} m(a)$ for every $b \in B$.

Definition 10 (Petri nets and Petri graphs). *A* (Place/Transition) Petri net *is a tuple* $N = \langle S_N, T_N, {}^{\bullet}(\,), (\,)^{\bullet}, m_0\rangle$, *where* S_N *is a set of* places, T_N *is a set of* transitions, ${}^{\bullet}(\,), (\,)^{\bullet} : T_N \to S_N^{\oplus}$ *determine for each transition its* pre-set *and* post-set, *and* $m_0 \in S_N^{\oplus}$ *is the* initial marking. *A transition* t *is* enabled *at a*