Integrated Design and Process Technology, **IDPT-2002**
Printed in the United States of America, June, 2002

Static Analysis of Distributed Systems with Mobility Specified by Graph Grammars—A Case Study

Paolo Baldan and Andrea Corradini
Dipartimento di Informatica, Università di Pisa, Italy
{baldan,andrea}@di.unipi.it

Barbara König
Institut für Informatik, Technische Universität München, Germany
koenigb@in.tum.de

ABSTRACT: We consider a distributed system with mobility modelled as a graph transformation system. Then we show that non-secure level processes cannot influence secure level processes, a property formalized as the absence of causal dependencies between such processes. This is done by resorting to an analysis technique for graph transformation systems, called approximated unfolding, based on the construction of an approximation of the unfolding semantics.

I. Introduction

Graph grammars have been shown to be an expressive formalism for the specification of concurrent and distributed systems [15], [8]. The use of graphs to model system states allows one to provide an explicit representation of the physical or logical connections between system components and, in particular, it eases the explicit representation of the topology of a distributed system. The rewriting rules add a dynamic dimension to the representation, in that they permit to specify how such a topology evolves, a capability which is essential to describe systems with mobility. On the one hand, various calculi for mobility have been defined directly using graphs and graph transformation as a basic language [6], [12]. On the other hand, graph transformation systems have been used to provide a concurrent semantics for several calculi for mobility, like the π-*calculus* [13] and the *ambient calculus* [10]. A set of graph rewriting rules encodes the dynamics of such calculi, acting over the graphical representations of agents. Hence graph transformation systems can be seen as a common intermediate language where such calculi can be "compiled".

In both cases, an advantage of resorting to graph grammars is represented by the fact that their concurrent behaviour has been deeply studied in the last years and a consolidated theory of concurrency is now available [15], [8]. In particular, several classical approaches to the semantics of Petri nets, like process and unfolding semantics, have been extended to graph grammars (see, e.g., [4], [14], [2], [3]). These concrete operational models provide a description of the behaviour in terms of (usually) non-effective (e.g., infinite, non-decidable) structures, which cannot be used directly to reason about the modelled system. However they can serve as a basis for the definition of more abstract semantics and of effective techniques for the verification of systems modelled in terms of graph transformation.

In this paper, following this line of research, we introduce a simple distributed system with mobility represented as a graph grammar. Then, by resorting to an analysis technique for graph grammars presented in [1], we prove that in such a system non-secure level processes cannot influence secure level ones.

More precisely, the system consists of a set of locations and a set of processes running on such locations. Locations are linked by connections through which processes can travel. A subset of locations and a subset of connections is considered to be secure. Secure locations are positioned behind a firewall which processes can cross in one direction only. Hence one expects that in such system processes running on secure locations cannot be influenced by those running on non-secure locations, a property closely related to multi-level security and non-interference [11], [9].

The mentioned property, formalized as the absence of causal dependencies from non-secure to secure level processes, is proved by using a static analysis technique for graph grammars, proposed in [1], which is based on the so-called unfolding of the grammar, a semantics which represents all the possible computations of the grammar in a single branching structure. Given a graph transformation system and a start graph, an algorithm produces a finite structure consisting of a hypergraph decorated with transitions (Petri graph) which can be seen as an approximation of the Winskel's style unfolding of the graph transformation system. The fact that any graph reachable in the grammar has an homomorphic image in the Petri graph and the additional causal information provided by transitions allow us to prove several interesting properties of the original system. In particular, in our case, an analysis of the approximated unfolding reveals that, as desired, processess run-

Research partially supported by the ESPRIT Working Group APPLIGRAPH (Applications of Graph Transformation), by the Joint CNR/CNPq Project IQ-Mobile and by the IST Project AGILE: Architectures for Mobility.

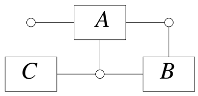


Fig. 1. A hypergraph.

ning on secure locations will never be causally dependent on processes running on non-secure locations.

The rest of the paper is structured as follows. In Section II we outline the technique for the construction of the approximated unfolding of a graph grammar and we point out some of its relevant properties. To this aim, we first review the basics of (hyper)graph rewriting and of the unfolding semantics for graph transformation systems. Then, in Section III we illustrate the approximated unfolding technique, by showing how it can be applied to prove a security property for a simple distributed system with mobility, modelled as a graph grammar. Finally, in Section IV we draw some conclusions and point out directions for future work.

We have tried to keep the paper as much self-contained as possible, giving at least an informal description of the essential notions and results. Still, the reader would very much benefit from some basic knowledge of graph transformation and of Petri net theory.

II. The approximated unfolding technique

In this section we give an overview of the technique, proposed in [1], for the construction of a finite approximation of the unfolding of a graph grammar. More precisely, first we introduce the class of (hyper)graph transformation systems we will deal with. Then we review the basic ideas underlying the ordinary unfolding construction for graph grammars [14], [3] and we describe Petri graphs, the structure combining hypergraphs and Petri nets, which is used to approximate the behaviour of graph grammars. Finally, we present the algorithm computing the approximated unfolding of a graph grammar and we discuss some of its relevant properties.

A. Hypergraph rewriting

The structures on which rewriting takes place are *hypergraphs*, i.e., a kind of graphs where each edge is attached to a (possibly empty) list of nodes. For instance, Fig. 1 presents a hypergraph with three edges, depicted as rectangles, and three nodes, depicted as small circles. Observe that edges are labelled, while nodes are not. The edge labelled A is attached to three nodes, while the edge labelled C only to one. Given a hypergraph G, we will denote by V_G and E_G its sets of nodes and edges, respectively.

A *rewriting rule* r is a triple (L,R,α), where L and R are hypergraphs, called *left-hand side* and *right-hand side*, respectively, and $\alpha : V_L \to V_R$ is an injective function mapping nodes of the left-hand side to nodes of the right-hand side.

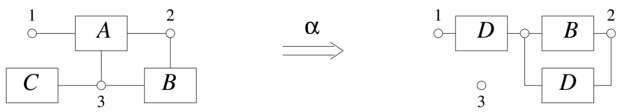


Fig. 2. A graph rewriting rule.

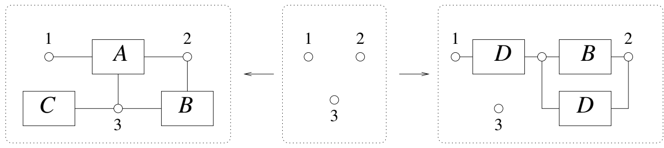


Fig. 4. A DPO rule corresponding to the rule in Fig. 2.

An example of a rewriting rule is presented in Fig. 2. The mapping α is implicitly represented by the numbering of nodes, namely, any node of the left-hand side is mapped by α to the node of the right-hand side with the same number. Unnumbered nodes of the right-hand side are new, i.e., generated by the application of the rule.

We recall that in [1] the treatment is limited to so-called *basic* rules, i.e., rules $r = (L,R,\alpha)$ where different edges in the left-hand side L have different labels, no node in L is isolated and no node in $V_R - \alpha(V_L)$ is isolated in R. These restrictions are not strictly needed, but make the presentation simpler.

Intuitively, a rule $r = (L,R,\alpha)$ specifies that an occurrence of the left-hand side L can be "replaced" by R. More concretely, to apply r to a graph G one must find a *match* of r in G, i.e., an occurrence of the left-hand side L in G (formally a graph morphism $\varphi : L \to G$). The application of r to G at the match φ first removes from G the image of the edges of L. Then the graph G is extended by adding the new nodes in R (i.e., the nodes in $V_R - \alpha(V_L)$) and the edges of R. Observe that the (images of the) nodes in L are "preserved", i.e., not affected by the rewriting step. Fig. 3 represents an application of the rule of Fig. 2. The match in the left-hand graph is represented by the grey part.

A *graph transformation system* $\mathcal{G}$ is a finite set of rewriting rules. A graph transformation system with a start graph $(\mathcal{G}, G_{\mathcal{G}})$ is called a *graph grammar*. Often, with abuse of notation, the graph $G_{\mathcal{G}}$ will be omitted and we will speak of the graph grammar $\mathcal{G}$.

The reader who is familiar with the double-pushout (DPO) approach [7], [5] to graph rewriting might have recognized that our rules (L,R,α) can be seen as special DPO rules $(L \hookleftarrow V_L \overset{\alpha}{\hookrightarrow} R)$ and that our notion of rewriting corresponds to a DPO construction. For instance Fig. 4 shows a DPO rule corresponding to the rule in Fig. 2.

B. Unfolding of graph grammars

Let $\mathcal{G}$ be a graph grammar. The *unfolding* of the grammar [14], [3] is constructed inductively beginning from the

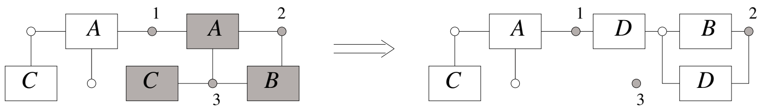
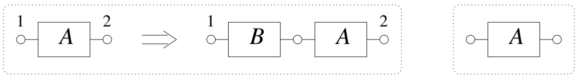


Fig. 3. An application of the rule in Fig. 2.



Rule r Start graph

Fig. 5. A simple graph grammar $\mathcal{S}$.

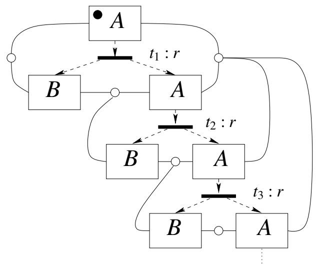


Fig. 6. A fragment of the unfolding of the graph grammar $\mathcal{S}$ of Fig. 5.

start graph and then applying at each step in all possible ways the rules, without deleting the left-hand side, and recording each occurrence of a rule and each new graph item generated in the rewriting process. As a result one obtains an acyclic branching structure describing the behaviour of $\mathcal{G}$. In particular every reachable graph embeds in (a concurrent subgraph of) the graph produced by the unfolding construction.

For instance, consider the very simple graph grammar $\mathcal{S}$ in Fig. 5, with only one rewriting rule r. A fragment of its unfolding $\mathcal{U}(\mathcal{S})$ is represented in Fig. 6, in the form of a structure, called *Petri graph*.

A *Petri graph* for a graph transformation system $\mathcal{G}$ consists of a hypergraph G together with a Petri net having the graph edges as places. Any marking m of the Petri graph $\mathcal{G}$ can be interpreted as a graph G_m. For a safe marking G_m is the subgraph of G consisting only of the marked edges. More generally, for a (possibly) non-safe marking the graph G_m can include multiple copies of the same edge of G, one for each token marking that edge. For instance, the marking of the Petri graph in Fig. 6 represents the start graph of the grammar, consisting of a single A-labelled edge. The transitions, which are represented as small black rectangles, can be interpreted as occurrences of rewriting rules. Each transition t is labelled by the corresponding rule r (written as $t : r$) and it consumes and produces tokens in the edges of the graph, as denoted by the dashed arrows, according to the shape of the left- and right-hand side of r. For instance, transition t_1 in Fig. 6 represents the application of rule r to the start graph of the grammar and indeed it consumes a token in the edge labelled by A, producing new tokens in edges labelled by A and B, respectively. Similarly, t_2 represents the application of r to the edge A produced by the first occurrence of r, and so on.

It is worth noting that the unfolding makes explicit the causal dependencies among the rule occurrences and the produced graph items. For instance, t_2 causally depends on t_1 since it consumes a token (in an A-labelled edge) produced by t_1. Formally, the causality relation is defined as the least transitive relation such that $t < t'$ whenever t' consumes a token produced by t. In other words, $t < t'$ if there exists a chain of transitions $t_0 = t, t_1, \ldots, t_{n-1}, t_n = t'$ such that each t_i consumes an item produced by its predecessor t_{i-1}. Therefore, the causality relation can be informally interpreted as a relation revealing the flow of information in the system. Altenatively, we can think that $t < t'$ means that the presence of t' in the past history of t is relevant for t, and thus that rule r labelling t influences the behaviour of rule r' labelling t'. Following this intuition, we will later formalize our security property relying on the causality relation.

C. Approximated unfolding

The unfolding is usually infinite, also in the case of finite-state systems. To ensure that the construction produces a finite structure, in [1] we propose to consider—besides the *unfolding rule*, which extends the graph by simulating the application of a rule—a *folding rule*, which, under suitable conditions, allows us to “merge” different parts of the structure which has been generated.

To perform an *unfolding step* one must find a match φ of a rule r, i.e., an occurrence of its left-hand side, in the current graph. Then the rule is applied without deleting the match, but only adding the new items as specified by the right-hand side of the rule. In a *folding step*, instead, one has to find two matches of the same rule r. Then the two occurrences of the left-hand side of r in the current graph are merged. In both kinds of steps the matches are required to be coverable, i.e., when interpreted as markings of the Petri graph they must be covered by a marking reachable from the initial marking in the Petri graph. Fig. 7 presents an unfolding step and a folding step for the graph grammar in Fig. 5. The involved matches are depicted in grey.

The algorithm for the construction of the approximated unfolding of a graph grammar $\mathcal{G}$ can be described as follows:

(Step 0) Begin from the start graph of the grammar;

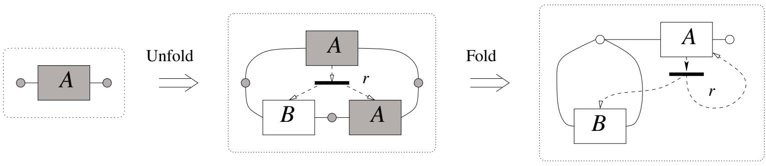


Fig. 7. An unfolding and a folding step for the graph grammar $\mathcal{S}$ in Fig. 5.

(Step $i+1$) *Let U_i be the Petri graph produced at step i. Choose non-deterministically one of the following actions*

⋆ Folding: *Find a rule r and two (coverable) matches of r in U_i such that one causally depends on the other, i.e., each item x in the second match causally depends on the transition labelled r applied to the first match or coincides with an item in the first match. Then merge these two matches.*

⋆ Unfolding: *Find a rule r and a (coverable) match of r in U_i such that the match cannot be involved in a folding step (and no other r-labelled transition has this match as precondition). Then apply the unfolding step to such a match.*

The algorithm stops when no (folding or unfolding) step can be performed. The resulting Petri graph is called approximated unfolding *of $\mathcal{G}$ and denoted by $\mathcal{A}(\mathcal{G})$.*

For instance, the approximated unfolding of the graph grammar $\mathcal{S}$ in Fig. 5, is the rightmost Petri graph in Fig. 7. It is obtained by performing first an unfolding and then a folding step. Observe that, after the first step, another unfolding step applied to the newly produced A-labelled edge would not be legal, according to the application conditions outlined above, since the same match can be involved in a folding step.

D. Properties of the approximated unfolding

The algorithm for the construction of the approximated unfolding can be shown to be *terminating* and *confluent*. Hence by a classical result, its result is uniquely determined and *finite*.

Moreover, although the Petri graph $\mathcal{A}(\mathcal{G})$ introduces a degree of approximation in the representation of the behaviour of a graph grammar $\mathcal{G}$, it enjoys some properties which allow one to use the approximated unfolding for verification purposes.

First of all, *every graph G reachable in the original graph grammar $\mathcal{G}$ can be mapped homomorphically to the graph underlying $\mathcal{A}(\mathcal{G})$, i.e., there exists a (graph) morphism $\varphi : G \to \mathcal{A}(\mathcal{G})$. Furthermore, the φ-image of G in $\mathcal{A}(\mathcal{G})$ corresponds to a marking which is reachable in the Petri graph.* Therefore, both the graphical and the Petri net structure of the approximated unfolding can be helpful for verification. Properties which are reflected by graph morphisms, like the non-adjacency of some labels, the non-existence of a given path of edges or the absence of cycles, can be checked directly on the graph underlying the approximated unfolding. The Petri net underlying the approximated unfolding can be analyzed with standard Petri net techniques, e.g., to establish a bound to the number of edges with a certain label in reachable graphs. Usually, the more fruitful approach consists of using both kinds of information at the same time.

A second relevant property is the *existence of a morphism from the full unfolding to the approximated unfolding $\Psi : \mathcal{U}(\mathcal{G}) \to \mathcal{A}(\mathcal{G})$, which "preserves" the causal relation*, i.e., such that for any pair of items x and y in $\mathcal{U}(\mathcal{G})$, if y causally depends on x then also their Ψ-images are in the same relation. This can be helpful to show that some items in the full unfolding are not causally dependent, a fact which can be proved by showing that their Ψ-images in the approximated unfolding are not causally related.

III. Verifying a security property of a distributed system with mobility

In this section we consider a distributed system with mobility, suggesting how it can be modelled as a graph grammar. Then the verification technique described in the previous section is used to show that no process running on a secure location can ever be influenced by a process running on a non-secure location.

A. Modelling a distributed system with mobility

We consider a simple distributed system with mobility, with locations and processes running on these locations, represented as a graph grammar. Locations, connections and processes are represented as hyperedges. The meaning of the edge labels is the following: P denotes a process running on a location, Q stands for a process which travels between locations. The labels L and sL stand for locations and secure locations, respectively, and, in the same way, *conn* and *sconn* stand for connections and secure connections, respectively. And finally *firewall* represents the firewall connection which can only be crossed in one direction. For instance, the start graph in Fig. 8 represents a network with three locations, one of which is secure. Observe that the secure location is connected to the others through firewalls and initially no process is running. The dynamics of the system is modelled by the rewriting rules in Fig. 8.

Most of the rules come in two versions, the secure and the non-secure version, referred to as [s-rule name] and [rule name], which are obtained by keeping or removing,

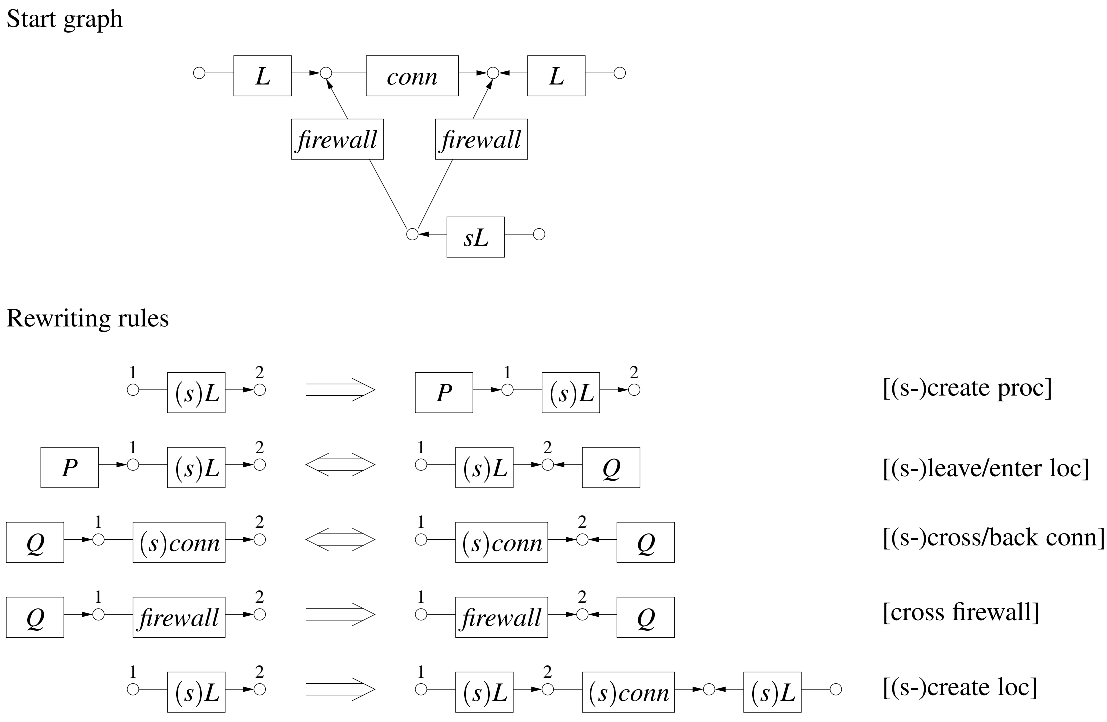


Fig. 8. A start graph and rewriting rules for mobile processes (graph grammar $\mathcal{M}$).

respectively, in all the edges of a rule the letter "*s*" in brackets. The rules enable a location to create processes or new connected locations (rules [(s-)create proc] and [(s-)create loc]), they allow a process to leave a location (rule [(s-)leave loc]), to travel along connections (rules [(s-)cross conn] and [(s-)back conn]) and firewalls (rule [cross firewall]), and to reenter a location (rule [(s-)enter loc]). Note that some rules for process movement can be applied in both directions, as indicated by the presence of a double arrow. This is clearly not possible in the case of firewalls.

Let us denote by $\mathcal{M}$ the graph grammar consisting of the start graph and rewriting rules specified in Fig. 8. A possible evolution of $\mathcal{M}$ is depicted in Fig. 9: a new secure location is created which afterwards spawns a new process. This process travels along the new secure connection and then crosses a firewall.

B. The approximated unfolding at work

We want to make sure that in the system described in the previous section no process running on a secure location is ever influenced by a process running on a non-secure location. Instead, an influence in the opposite direction is allowed.

The intuitive idea of "influence" is formalized by resorting to the notion of causality, i.e., we require that processes running on secure locations are never causally dependent on processes running on non-secure locations. The absence of such causal dependencies could be detected in the full unfolding of the system, which however, being infinite, cannot be directly constructed and analyzed. Therefore, we exploit the algorithm described in the previous section to construct the approximated unfolding. Since causality is preserved by the mapping from the full into the approximated unfolding, we are sure that whenever two processes are *not* causally related in the approximation, they are consequently not causally related in the full unfolding.

The algorithm applied to the graph grammar $\mathcal{M}$ in Fig. 8, performs several folding and unfolding steps and finally produces the Petri graph $\mathcal{A}(\mathcal{M})$ in Fig. 10, where the initial marking corresponds to the start graph of the grammar.

Some of these unfolding and folding steps are presented in Fig. 11:

1. We begin from the start graph of the system in Fig. 8.
2. We choose—non-deterministically—to apply an unfolding step corresponding to the creation of a new process at a non-secure location (rule [create proc]).
3. Since the newly created location is a match for rule [create proc] and it causally dependends on transition t which corresponds to rule [create proc], a folding step allows us to merge such an edge and the precondition of t.
4. A sequence of several similar unfolding and folding steps follows.
5. Since the process P and its location on the right represent a match for rule [leave loc] and they both causally depend on transition t', the match and the precondition of t' are consequently merged.

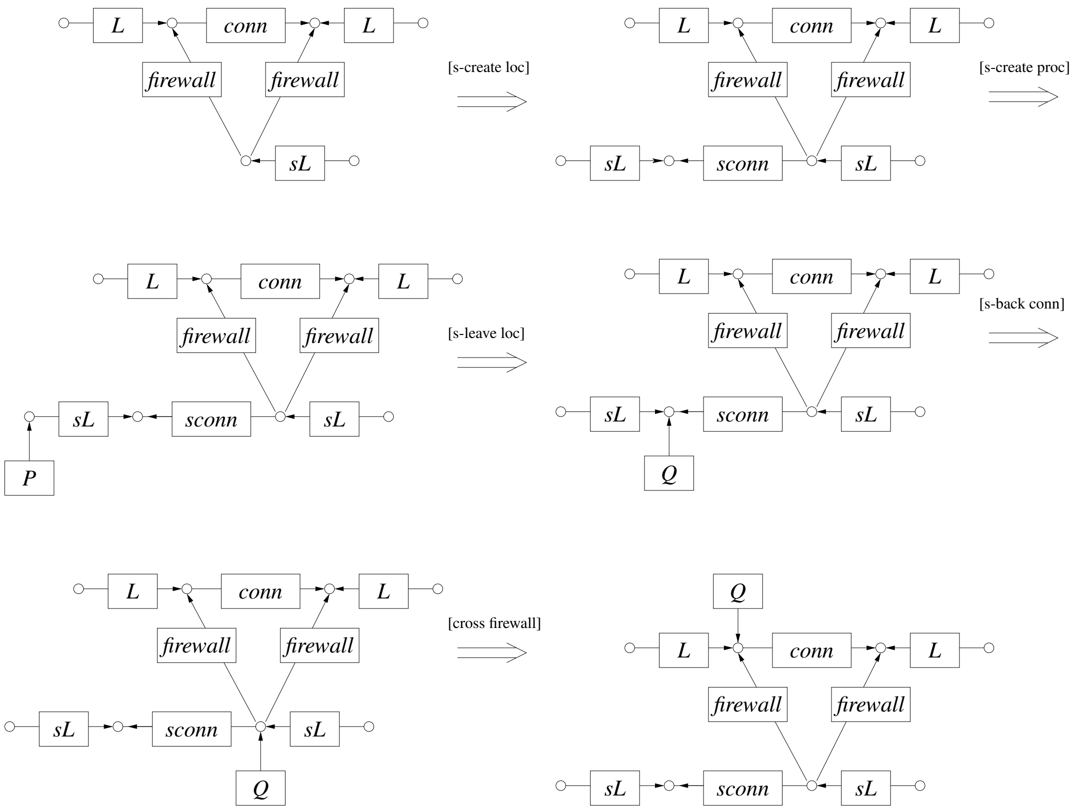


Fig. 9. A few transformation steps of the mobile system.

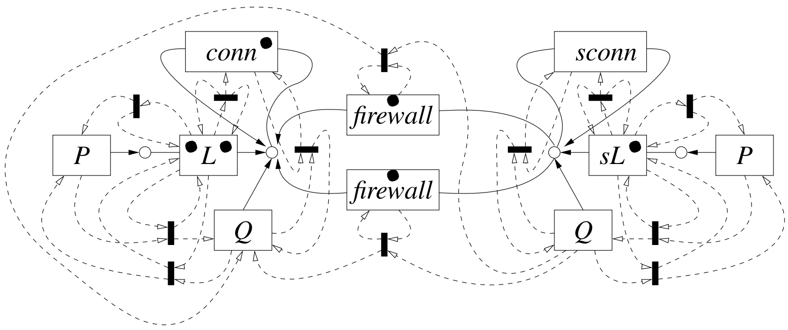


Fig. 10. Approximated unfolding of the graph grammar $\mathcal{M}$ in Fig. 8.

Several more steps finally lead to the Petri graph in Fig. 10.

An analysis of the approximated unfolding $\mathcal{A}(\mathcal{M})$ reveals that no process running on a non-secure location can be a cause for a process running on a secure location. In fact, let $\Psi : \mathcal{U}(\mathcal{M}) \to \mathcal{A}(\mathcal{M})$ be the morphism from the full unfolding to the approximated unfolding of the system. Observe that in the approximated unfolding the edge P on the right represents processes running on secure locations (i.e., processes running on secure locations in the full unfolding are mapped by Ψ to this edge), the edge P on the left represents processes running on non-secure locations. Furthermore the left-hand P is *not* a cause for the right-hand P, since there is no chain of transitions such that the first one consumes a token in the left-hand P, each of the intermediate transitions consumes a token which is produced by its predecessor and the last one produces a token in the right-hand P. Since causality is preserved in the approximation, i.e., the mapping Ψ preserves causality, we can conclude that, as desired, no process running on a non-secure location is a cause for a process running on a secure location.

Note that the system would not satisfy this security property any more if we replaced one of the firewalls by a secure or non-secure connection, even if there is no explicit communication between processes. The reason for this is that a

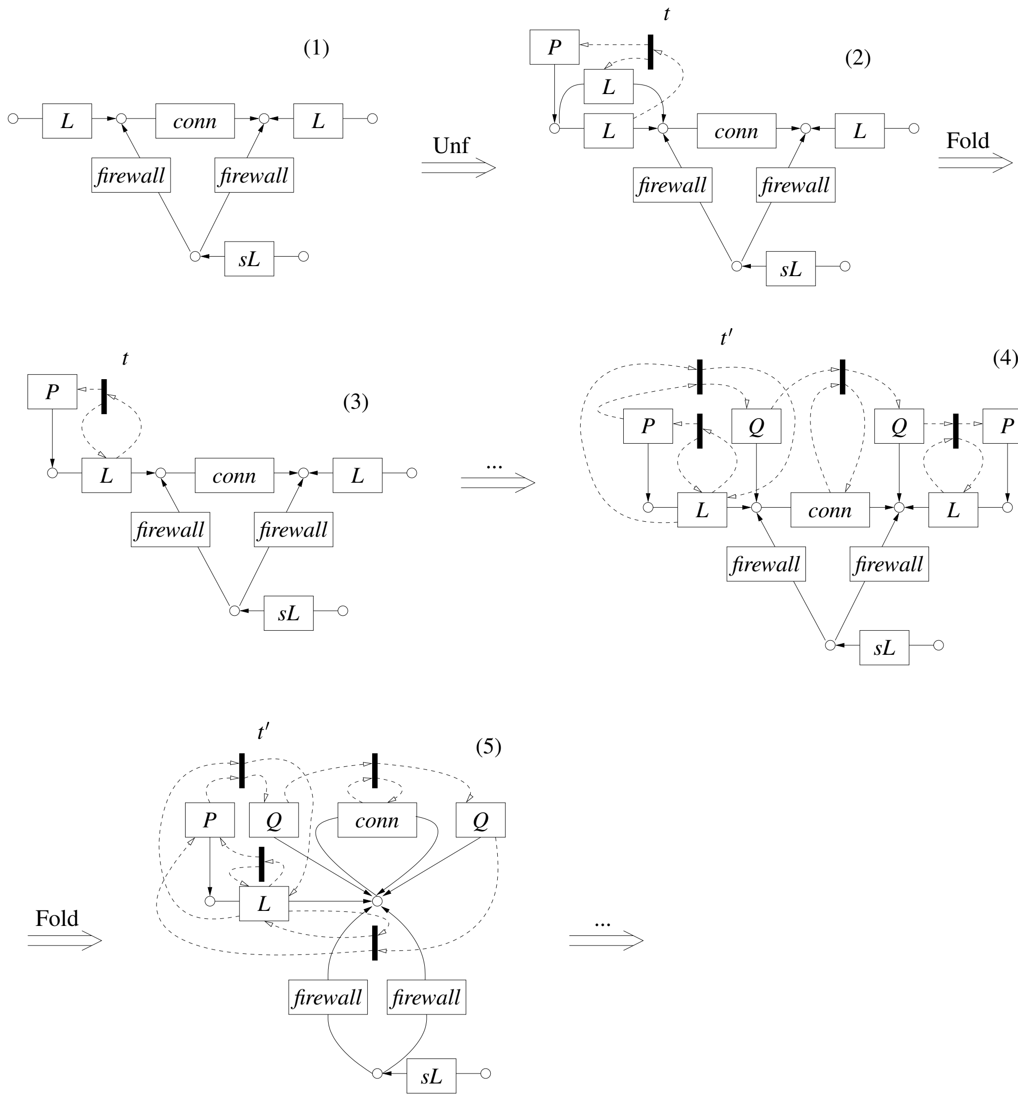


Fig. 11. Folding and unfolding steps in the approximated unfolding technique.

process running on a non-secure location could move along the connection and influence a secure location (in practice this could mean modifying some of its data) which afterwards creates a new process.

The example could of course be extended by introducing rewriting rules which model explicit process communication. The same technique would allow to prove the same property at the price of having a larger approximated unfolding.

IV. Conclusions

We have shown how a static analysis technique based on the construction of an approximation of the Winskel's style unfolding can be used to verify a security property in a simple graph-based formalism for systems with mobility.

Our current investigations concern the applicability of (variants of) the presented technique to the verification of more sophisticated graph-based distributed systems with mobility. Moreover, as already mentioned, also the graphical representations of calculi for mobility proposed in the literature offer an interesting framework where our technique could be applied.

In this context our future work will address the problem of understanding which safety properties can be analyzed by our technique and how it can be adapted to the various needs arising in the specification and analysis of systems with mobility.

Another interesting issue is the automation of the described technique, including the generation of the approximated unfolding for a given graph grammar and its analysis. As already mentioned, any property of reachable states which is reflected by graph morphisms can be checked directly on the graph underlying the approximated unfolding.

Furthermore most of the properties regarding causality are concerned with the structure of the approximated unfolding itself, seen as a graph, and, in particular, with the presence of certain (causal) paths. Hence we believe that classic algorithms and tools for the analysis of graph properties could be of great help. On the other hand, since the Petri net underlying the approximated unfolding provides an over-approximation of the behaviour of the original rewriting system (i.e., any computation in the original system is a legal computation in the approximated unfolding) any technique and tool devised for ordinary Petri nets is potentially useful for the analysis of the approximated unfolding.

REFERENCES

[1] P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In K.G. Larsen and M. Nielsen, editors, *Proceedings of CONCUR 2001*, volume 2154 of *LNCS*, pages 381–395. Springer Verlag, 2001.

[2] P. Baldan, A. Corradini, and U. Montanari. Concatenable graph processes: relating processes and derivation traces. In *Proceedings of ICALP'98*, volume 1443 of *LNCS*, pages 283–295. Springer Verlag, 1998.

[3] P. Baldan, A. Corradini, and U. Montanari. Unfolding and Event Structure Semantics for Graph Grammars. In W. Thomas, editor, *Proceedings of FoSSaCS '99*, volume 1578 of *LNCS*, pages 73–89. Springer Verlag, 1999.

[4] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.

[5] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic Approaches to Graph Transformation I: Basic Concepts and Double Pushout Approach. In [15].

[6] F. L. Dotti and L. Ribeiro. Specification of mobile code systems using graph grammars. In *Proc. of FMOODS '00*. Kluwer Academic Publishers, 2000.

[7] H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *Proceedings of the 1st International Workshop on Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *LNCS*, pages 1–69. Springer Verlag, 1979.

[8] H. Ehrig, J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism and Distribution*. World Scientific, 1999.

[9] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In U. Montanari, J. Rolim, and E. Welzl, editors, *Proceedings of the 27th ICALP*, volume 1853 of *LNCS*, pages 354–372. Springer Verlag, 2000.

[10] F. Gadducci and U. Montanari. A concurrent graph semantics for mobile ambients. In S. Brookes and M. Mislove, editors, *Proceedings of the 17th MFPS*, volume 45 of *Electronic Notes in Computer Science*. Elsevier Science, 2001.

[11] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings 1982 IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society, 1982.

[12] B. König. *Description and Verification of Mobile Processes with Graph Rewriting Techniques*. PhD thesis, Technische Universität München, 1999.

[13] U. Montanari and M. Pistore. Concurrent semantics for the π-calculus. In *Proceedings of the 11th MFPS*, volume 1 of *Electronic Notes in Computer Science*. Elsevier Science, 1995.

[14] L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars*. PhD thesis, Technische Universität Berlin, 1996.

[15] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*. World Scientific, 1997.