Electronic Communications of the EASST
Volume 2 (2006)

Proceedings of the Workshop on Petri Nets and Graph Transformation (PNGT 2006)

Reversing graph transformations[1]

Paweł Sobociński

7 pages

Guest Editors: Paolo Baldan, Hartmut Ehrig, Julia Padberg, Grzegorz Rozenberg
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Reversing graph transformations[2]

Paweł Sobociński

ECS, University of Southampton, United Kingdom

Abstract: In recent work with Vincent Danos and Jean Krivine the author introduced a general framework for backtracking in concurrent formalisms, thus allowing modelling of situations where deadlock can arise without the necessity of explicitly encoding the often involved backtracking mechanisms. Here we shall discuss how the framework can be applied to the well-known formalism of double-pushout graph transformation.

Keywords: graph transformation, reversibility, transaction

We present an application of the framework developed in [5] to the theory of graph transformation systems. Here we say "graph" to mean an object of an arbitrary category with pushouts along monomorphisms where the local Church-Rosser theorem holds. An example is an adhesive category [10]. Of course, the category **Graph** of ordinary graphs is adhesive and in this setting it suffices to require that the interface embeds in both the left and the right hand sides of each production.

In order to establish the basic concepts we shall consider a concrete example, working in the category $\mathbf{C}$ of directed graphs whose vertices are tagged with the elements of a set; the presheaf topos $\mathbf{C} = \mathbf{Set}^{\cdot \rightarrow \cdot \leftarrow \cdot}$. The edges of such graphs will represent physical proximity of entities represented by the vertices. The elements with which vertices may be tagged represent the internal state of the entities. Let T be the graph illustrated in Fig 1. The graph has two vertices ("Philosopher" and "Table") and one edge. The tags are "fork", "eating", "thinking" and "fork". Then $\mathbf{C}/T$ is the adhesive category [10] of graphs typed over T, in the usual way. One can thus think of the objects in $\mathbf{C}/T$ as graphs consisting of two types of vertices, the philosopher vertices and the table vertices. The philosopher vertices, which may be tagged with a set of "fork", "eating" or "thinking" tags, may be the source of an arbitrary set of edges with targets the table vertices, which themselves may be tagged with a set of "fork" tags.

Our main example is a graph transformation system over the category $\mathbf{C}/T$ and models Hoare's dining philosophers problem [9]. Let $\mathscr{P}$ be the DPO grammar over $\mathbf{C}/T$ with the start graph S as illustrated in Fig 2 and the three productions q_1, q_2 and q_3 illustrated in Fig 3. Note that we only illustrate the left and the right hand sides of the productions, their interface is the obvious one in each case. Each thinking philosopher may claim a fork next to her using the production q_1. Once a thinking philosopher has two forks in her possession, she may start eating via the production q_2. Finally, an eating philosopher can release her forks at any time and return to thinking using production q_3. The dining philosophers problem is famous not least for the fact that it succinctly illustrates the fundamental issue of *deadlock* in parallel programming. If each of the philosophers picks up the fork to her left then no further productions are possible and the philosophers starve to death.

To solve this problem, one could partition the set of productions into *reversible* and *irreversible* productions; the idea being that the reversible productions may lead to deadlock and thus should

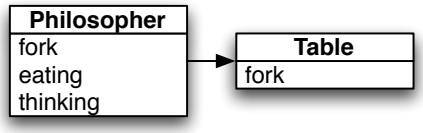

Figure 1: Type graph T.

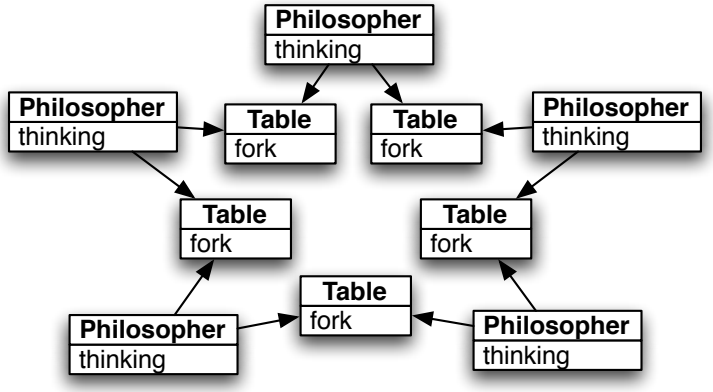

Figure 2: Start graph S.

be allowed to be (correctly) backtracked. On the other hand, the irreversible productions occur only in the presence of a desired global state possibly reached via the application of a number of the reversible productions. In our particular example one could specify the set of reversible productions $R = \{q_1\}$ and the set of irreversible productions $I = \{q_2, q_3\}$.

One may now view of the behaviour of each philosopher as a series of *transactions – causal* sequences of reversible actions followed by a single irreversible action. Each philosopher can perform two possible transactions, the first being two instances of q_1 followed by q_2 (claiming two forks and starting to eat) and the second being a single instance of q_3 (relinquishing the forks and starting to think). In order to avoid deadlock, one specifies that each of the initial actions can be reversed. The correctness of the backtracking means roughly that the behaviour of the resulting system should be precisely the behaviour of the transactions of the original system "up to" reversible moves.

The naive solution of simply adding a reversed production $q_{1\star}$, illustrated in Fig 4 is unsatisfactory. Indeed, a philosopher can now begin by picking up her left fork with q_1 and placing it via $q_{1\star}$ together with her right fork. This sequence of actions results in states not reachable from the start state by performing the transactions of the original system - consider for instance the state illustrated in Fig 5. Thus although the addition of $q_{1\star}$ solves the problem of deadlock, it is

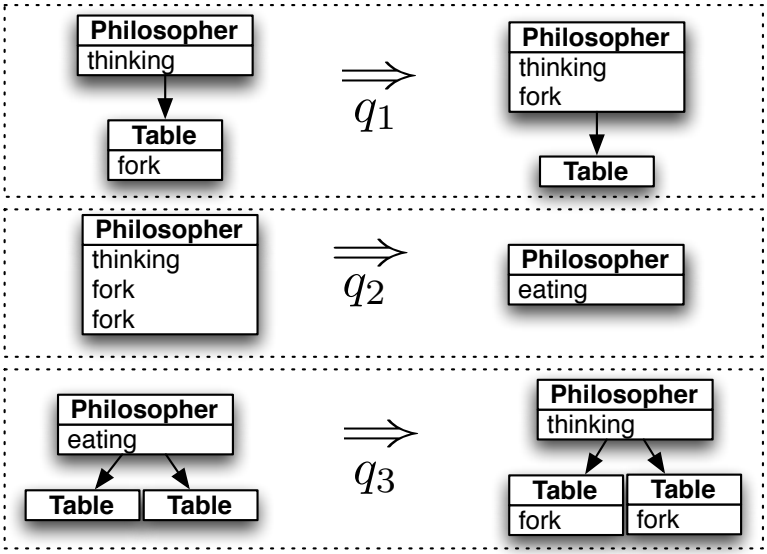

Figure 3: Productions q_1, q_2 and q_3 of $\mathscr{P}$.

clearly incorrect.

There are several ways to fix the naive solution: for example, one can label the edges out of each philosopher with l and r, replace the rule q_1 with two rules q_{1l}, q_{1r} and add their inverses $q_{1l\star}$, $q_{1r\star}$, thus disallowing the aforementioned errant behaviour. There are two apparent problems with such an ad-hoc solution: firstly, one has to prove that the transactions are indeed modelled correctly (trivial in this case, but not always so); secondly, there is a danger of making the model too complex to be of use. There is a general solution, described and shown to be correct in [5]. It arose by generalising previous work on reversing the process calculus CCS by Danos and Krivine [3,4]. It is general in the sense that it applies also to other models for concurrency such as Petri nets and process calculi. We briefly outline this solution below, instantiated with the example of graph transformation systems.

Given a grammar $\mathscr{G}$, let the *category of computations* $\mathbf{Cp}\mathscr{G}$ be the category with objects those of $\mathbf{C}$ (the ambient category of $\mathscr{G}$) and arrows finite (possibly empty) paths of direct derivations modulo switch-equivalence. The arrows of $\mathbf{Cp}\mathscr{G}$ are thus the concurrent computations of $\mathscr{G}$.[4] Such categories are natural and have been studied for other models, notably for Petri nets [11]. They have also been considered in the setting of graph transformation, see [8, Ch 4] for an in-depth presentation and a proof that this category also arises as a free construction.

In the problem specification, the set of productions P of $\mathscr{G}$ is partitioned into sets of reversible

[4] Note that the description given here is quite concrete since the states are not quotiented by isomorphism – in particular, thinking of the category as a transition system results in infinite branching for trivial reasons since if a state q can do a transition to state q' then it can do the same transition to any state isomorphic to q'. Ways of cutting down the state space by considering only isomorphism classes of states have been considered in the graph transformation literature, see [2] and references therein.

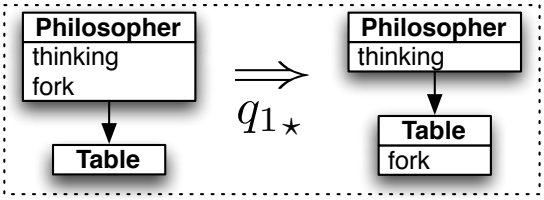

Figure 4: Reversed rule $q_{1\star}$.

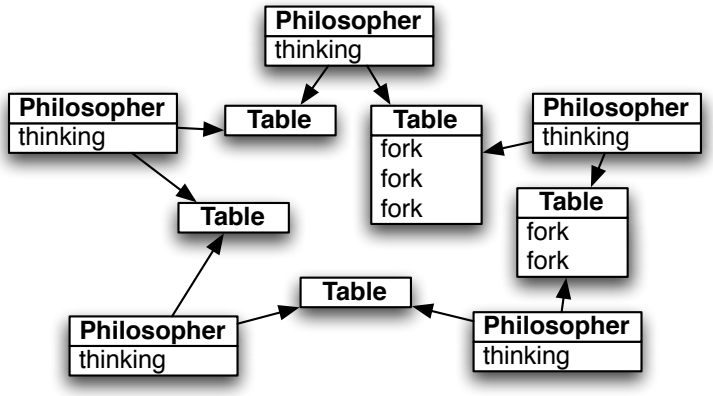

Figure 5: A state reachable from the start state after adding $\{q_1\}_\star$ to the set of productions.

productions R and irreversible productions I. Recall that in our running example, $R = \{q_1\}$ and $I = \{q_2, q_3\}$. Let $\mathscr{R}$ be the subcategory of $\mathbf{Cp}(\mathscr{G})$ with arrows the derivations consisting of only the reversible productions. Let $\mathscr{I}$ be the subcategory of $\mathbf{Cp}(\mathscr{G})$ consisting of the irreversible computations – roughly, those where the last action in each thread is irreversible. The arrows of $\mathscr{I}$ can be considered as paths (of possibly zero length) of transactions modulo concurrency.

The category $\mathscr{I}$ can actually be defined in a more abstract way using a particular closure construction from the theory of prefactorisation systems of Freyd and Kelly [6], see [5] for details. It is not difficult to verify that $\langle \mathscr{I}, \mathscr{R} \rangle$ is actually a factorisation system on $\mathbf{Cp}(\mathscr{G})$ – each computation can be factorised into a (possibly empty) irreversible component followed by a (possibly empty) reversible component, moreover, such factorisation is essentially unique. Notice that to able to factorise computations in such a way is to be able to ascertain exactly which part of a computation history is reversible – it is exactly the reversible component in the irreversible-reversible factorisation. The presence of such a factorisation system is thus closely related to the idea of backtracking and forms an integral part of the general construction.

The first step of the construction is the construction of the so-called category of histories. The idea is to obtain a new category of computations $h(\mathbf{Cp}\mathscr{G}, \mathscr{R})$ with the objects being the reversible

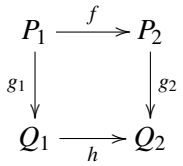

Figure 6: An arrow in $h(\mathbf{Cp}\mathscr{G},\mathscr{R})$.

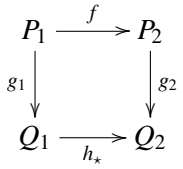

Figure 7: An arrow in $h_\star(\mathbf{Cp}\mathscr{G},\mathscr{R})$.

computations (the arrows in $\mathscr{R}$) of the original category $\mathbf{Cp}\mathscr{G}$. In a graph transformation system this would mean that a state is no longer a particular graph but rather a concurrent computation made up of applications of reversible productions. The computations of the new system also deserve an explanation. The natural choice is that a computation starting from a state (reversible computation) $g_1 : P_1 \to Q_1$ is *any* computation $h : Q_1 \to Q_2$, ie h is an arbitrary arrow of $\mathbf{Cp}\mathscr{G}$. The only question remaining is what is the final state of this computation. The natural choice is to take this to be $g_2 : P_2 \to Q_2$ where $g_2 \circ f$ is the $\langle \mathscr{I},\mathscr{R}\rangle$ factorisation of $h \circ g_1$. Roughly, the intuition is that g_2 consists of the parts of g_1 which are causally visible after performing h (ie they were not in the causal history of an application of an irreversible production in h) together with any causally visible reversible components of h. More formally, the objects of $h(\mathbf{Cp}\mathscr{G},\mathscr{R})$ are arrows in $\mathscr{R}$, while the arrows are commutative diagrams, as illustrated in Fig 6, where h is in $\mathbf{Cp}\mathscr{G}$ and f is in $\mathscr{I}$. Given a reversible computation $g : P_1 \to Q_1$ (an object of the history category), clearly *any* computation $h : Q_1 \to Q_2$ leads to a (unique up to isomorphism) object $g_2 : P_2 \to Q_2$ of $h(\mathbf{Cp}\mathscr{G},\mathscr{R})$ and irreversible computation f resulting in a map $g_1 \to g_2$ – here $g_2 \circ f$ is simply the $\langle \mathscr{I},\mathscr{R}\rangle$-factorisation. The category $h(\mathbf{Cp}\mathscr{G},\mathscr{R})$ is related to $\mathscr{I}$ via an adjunction, see [5] for details.

It turns out that all that is missing is the ability to actually "undo" the reversible computations represented by the objects of $h(\mathbf{Cp}\mathscr{G},\mathscr{R})$. In order to do this, we shall need the notion of a category of fractions [7]. Given a set of morphisms $\mathscr{R}$ of a category $\mathbf{C}$, the category of fractions $\mathbf{C}[\mathscr{R}^{-1}]$ is the category resulting from $\mathbf{C}$ by "freely" adding inverses to the arrows of $\mathscr{R}$. We obtain a canonical functor $\Phi : \mathbf{C} \to \mathbf{C}[\mathscr{R}^{-1}]$ which sends each arrow in $\mathscr{R}$ to an isomorphism.

Let $h_\star(\mathbf{Cp}\mathscr{G},\mathscr{R})$ be the category of reversible histories. It has the same objects as $h(\mathbf{Cp}\mathscr{G},\mathscr{R})$ but arrows are formal diagrams, as illustrated in Fig 7, where $h_\star$ is in $\mathbf{Cp}\mathscr{G}[\mathscr{R}^{-1}]$ and $h_\star\Phi(g_1) = \Phi(g_2 f)$. Roughly speaking, this category is as $h(\mathbf{Cp}\mathscr{G},\mathscr{R})$ but histories can be backtracked [5]. Note that while we constructed the history categories for our particular category $\mathbf{Cp}\mathscr{G}$, in fact the constructions rely only on the presence of a factorisation system. Interestingly, while the

description of $h_\star(\mathbf{Cp}\,\mathbf{C},\mathscr{R})$ given here may appear ad-hoc, the result is actually equivalent to the category of fractions obtained by reversing those computations of $h(\mathbf{Cp}\,\mathbf{C},\mathscr{R})$ which have a reversible lower component.

The main result of [5] states that there is an equivalence of categories $h_\star(\mathbf{Cp}\mathscr{G},\mathbf{C}) \simeq \mathscr{I}$. This implies, as we shall explain in more detail, that to correctly capture the transactions for a graph grammar $\mathscr{G}$, one can replace the category of computations $\mathbf{Cp}\mathscr{G}$ with $h_\star(\mathbf{Cp}\mathscr{G},\mathscr{R})$. The equivalence of categories result can be seen as a proof of correctness of the "implementation" $h_\star(\mathbf{Cp}\mathscr{G},\mathbf{C})$ with respect to the "specification" $\mathscr{I}$. The proof in [5] proceeds to show that the evident functor $M_\star : h_\star(\mathbf{Cp}\mathscr{G},\mathbf{C}) \to \mathscr{I}$ which takes a diagram as in Fig 7 to its upper component is an equivalence of categories, thus full, faithful and essentially surjective on objects. The fact that $M_\star$ is a functor can be understood roughly in process-algebra terminology as a simulation of $h_\star(\mathbf{Cp}\mathscr{G},\mathbf{C})$ in $\mathscr{I}$. The fact that the functor is full means that, roughly, the simulation is also a functional bisimulation.

In order to concretely implement the solution described in the previous paragraph it is helpful to have a convenient way of representing an arbitrary concurrent computation in $\mathscr{G}$. One possibility is to take some notion of process, for instance as presented in [1]. Assuming such a "syntax", one:

(i) replaces the states of a computation; instead of a graph Q, a state is a reversible computation $g\colon P \to Q$ represented by an appropriate "syntactic" expression;

(ii) the original productions of $\mathscr{G}$ are allowed to be applied in the usual way, in a way which corresponds to the arrows of $h(\mathbf{Cp}\,\mathbf{C},\mathscr{R})$. Thus an application of a reversible production simply results in a "larger" state since there is now one more production which may be reversed. On the other hand an application of an irreversible production possibly results in a "smaller" state since it may causally depend on a certain part of the start state – that part of the state then needs to be removed in order to obtain the end state;

(iii) add an inverse production $q_\star$ for each reversible production $q \in R$, such reversed productions are then allowed to be applied precisely when their inverse appears as a final production (in terms of causality) in the state – in other words $q_\star$ can be applied at state g precisely when there exists a reversible computation g' such that $g = q \circ g'$. Applying the inverse then removes that application of q from the state.

The resulting computations are "weakly" (modulo reversible moves) equivalent to the transactions of the original grammar.

Bibliography

[1] P. Baldan, A. Corradini, T. Heindel, B. König, and P. Sobociński. Processes for adhesive rewriting systems. In *Proceedings of FoSSaCS '06*, volume 3921 of *LNCS*, pages 202–216. Springer, 2006.

[2] A. Corradini, H. Ehrig, R. Heckel, M. Lowe, U. Montanari, and F. Rossi. Algebraic approaches to graph transformation part I: Basic concepts and double pushout approach. In

Handbook of Graph Grammars and Computing by Graph Transformation, volume 1, pages 162–245, World Scientific, 1997.

[3] V. Danos and J. Krivine. Reversible communicating systems. In *Proceedings of Concur'04*, volume 3170 of *LNCS*, pages 292–307. Springer, 2004.

[4] V. Danos and J. Krivine. Transactions in RCCS. In *Proceedings of Concur'05*, volume 3653 of *LNCS*, pages 398–412. Springer, 2005.

[5] V. Danos, J. Krivine, and P. Sobociński. General reversibility. In EXPRESS '06, Electronic Notes in Theoretical Computer Science. Elsevier, 2006. To appear.

[6] P. J. Freyd and G. M. Kelly. Categories of continuous functors, I. *Journal of Pure and Applied Algebra*, 2:169–191, 1972.

[7] P. Gabriel and M. Zisman. *Calculus of fractions and homotopy theory*. Springer-Verlag, 1967.

[8] R. Heckel. *Open Graph Transformation Systems: A New Approach to Compositional Modelling of Concurrent and Reactive Systems*. PhD thesis, TU Berlin, 1998.

[9] C. A. R. Hoare. Towards a theory of parallel programming. In *Seminar at Queen's University, Belfast, Northern Irelend*. Academic Press, 1972.

[10] S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications*, 39(3):511–546, 2005.

[11] J. Meseguer and U. Montanari. Petri nets are monoids. *Information and computation*, 88:105–155, 1990.