Bisimulation Equivalences for Graph Grammars*

Paolo Baldan, Andrea Corradini, Ugo Montanari

Dipartimento di Informatica
Università di Pisa

Abstract. Along the years the concurrent behaviour of graph grammars has been widely investigated, and, in particular, several classical approaches to the semantics of Petri nets have been extended to graph grammars. Most of the existing semantics for graph grammars provide a (possibly concurrent) operational model of computation, while little interest has been devoted to the definition of abstract observational semantics. The aim of this paper is to introduce and study a behavioural equivalence over graph grammars, inspired by the classical *history preserving bisimulation*. Several choices are conceivable according to the kind of concurrent observation one is interested in. We concentrate on the basic case where the concurrent nature of a graph grammar computation is described by means of a prime event structure. As it happens for Petri nets, history preserving bisimulation can be studied in the general framework of *causal automata* — a variation of ordinary automata introduced to deal with history dependent formalisms. In particular, we prove that history preserving bisimulation is decidable for finite-state graph grammars, by showing how the problem can be reduced to deciding the equivalence of finite causal automata.

1 Introduction

Graph grammars have been shown to be a powerful formalism for the specification of concurrent and distributed systems, which properly generalizes Petri nets. Along the years their truly concurrent behaviour has been deeply studied and a consolidated theory of concurrency is now available [Roz97,EKMR99]. 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., [CMR96,Rib96,BCM98a,BCM99]).

Most of the existing semantics for graph grammars define a (possibly concurrent) operational model of computation, which gives a concrete description of the behaviour of the system in terms of non-effective (e.g., infinite, non-decidable)

* Research partially supported by the EC TMR Network GETGRATS (General Theory of Graph Transformation Systems), by the ESPRIT Working Group APPLIGRAPH (Applications of Graph Transformation), and by the MURST project TOSCA (Teoria della Concorrenza, Linguaggi di Ordine Superiore e Strutture di Tipi).

structures. Thus they cannot be used directly to reason about the modelled system. Indeed, these operational models are intended to represent the basis for the definition of more abstract semantics, which take into account only some aspects of interest for the system at hand, disregarding inessential details. At this level one can define effective techniques for checking the equivalence of systems with respect to the selected observations, for verifying if a system satisfies a given property or for synthesizing a system satisfying a given property. Roughly, we can distinguish two main approaches to system verification based on abstract semantics. First, one can verify a system by checking its equivalence with a special system which is known to be "correct". For instance, the fact that a system is secure with respect to external attacks can be checked by verifying that the system in isolation is semantically equivalent to the system under a generic attack. Alternatively one can develop a logic, adequate with respect to the abstract semantics, which is interpreted over the class of systems at hand. Then to verify that a system satisfies a certain property, expressed as a formula in the logic, one checks if it is a model (in logical sense) for the formula, hence the name *model checking* for this approach.

Some effort has been devoted to the development of logics suited to specify the dynamics of graph transformation systems, with special interest in the integration of graphical specifications and temporal logic constraints (see, e.g., [HEWC97,Hec98,Koc99,GHK00]), but the study of abstract behavioural semantics and of the corresponding logics has received little attention. Here we move some steps in this direction, introducing an abstract semantics for graph grammars based on the classical *history preserving bisimulation* (*HP-bisimulation*, for short) [RT88,DD90], a behavioural equivalence which, differently from ordinary bisimulation, takes into account the concurrency properties of a system. Informally, two systems are HP-bisimilar if every event in the first one can be simulated by an event in the second one with an equivalent causal history and vice versa. History preserving bisimulation on ordinary P/T nets [RT88,BDKP91] relies on the notions of process and deterministic *prime event structure* (PES) associated to a process. Roughly speaking, two nets N_0 and N_1 are HP-bisimilar if for any process π_0 of N_0 we can find a process π_1 of N_1 such that the associated deterministic PES's are isomorphic. Whenever π_0 can perform an action becoming a process π_0', also π_1 can perform the same action becoming π_1' and vice versa. Moreover the isomorphism between the PES's associated to π_0 and π_1 is required to be extensible to an isomorphism between the PES's associated to π_0' and π_1'. Intuitively, history preserving bisimulation is more appropriate than ordinary bisimulation whenever in a system we are interested not only in the events which might happen, but also in the dependency relations between such events. For instance, imagine to have a system where a subset of actions is considered critical and suppose that for security reasons critical actions must not be influenced by non-critical actions. This property, which can be seen as a form of non-interference [GM82], can be formalized by asking that critical actions do not causally depend on non-critical actions and it is invariant for transformations of the system which preserve HP-bisimilarity.

A basic source of inspiration for our work is the close relation existing between graph grammars and Petri nets. The simple but crucial observation is that Petri nets are essentially rewriting systems on multisets, i.e., the markings of the net, which can be seen, in turn, as discrete graphs labelled over the places of the net. Hence graph grammars can be viewed as a generalization of Petri nets: they allow to give a more structured description of the state in term of a proper graph and to specify "contextual" rewriting steps where part of the state is preserved. In this respect graph grammars are closer to some generalizations of nets in the literature, called nets with read (test) arcs or contextual nets (see, e.g., [JK95,MR95,Vog97]), where transitions can be enriched with a context, i.e., with the possibility of checking the presence of tokens in the places of the net, without consuming such tokens.

Indeed, our study of HP-bisimulation for graph grammars is guided by the work on ordinary Petri nets [MP97,Vog91], which has been generalized to contextual nets in [BCM00b]. Graph grammars come equipped with a notion of deterministic (graph) process [CMR96,BCM98a] and with an event structure model [BCM99,Bal00], and thus the notion of HP-bisimulation can be generalized to graph grammars. We show that HP-bisimulation is decidable for *finite-state* graph grammars, called here, by analogy with Petri nets, *n-safe* graph grammars. To this aim, as in [MP97,BCM00b], we resort to *causal automata* [MP97], a variation of ordinary automata where states are sets of names (or events) and transitions allow for the creation of new names and the deallocation of old ones. A generalization of causal automata, called *history-dependent automata* (*HD-automata*), has been proposed as a general framework to study history-dependent formalisms, like CCS with causal and location semantics or with value-passing, and the π-calculus with the ordinary, early or late, or non-interleaving semantics [MP98,Pis99].

The (possibly infinite) transition system of processes of a graph grammar, which is used to define HP-bisimulation, is translated to a causal automaton via a construction which respects (preserves and reflects) bisimilarity. The automaton is proved to be finite exactly for finite-state graph grammars. Thus HP-bisimilarity of any two finite-state graph grammars can be checked by verifying the bisimilarity of the corresponding automata. This can be done concretely by using the algorithm proposed in [MP97], which after removing from the states of the automaton the events which are useless, i.e., never referenced in the future, translates the causal automaton into an ordinary automaton. Then the standard techniques for ordinary transition systems can be used to check bisimilarity or to obtain a minimal realization. More recent works [Pis99,MP00] show that a minimal realization exists and can be constructed in the class of causal automata themselves (actually, in the mentioned papers, the general case of HD-automata is worked out and a suitable extension of HD-automata, the so-called *automata with symmetries*, must be introduced to get this result). As it happens for ordinary automata, also a causal automaton can be seen as a coalgebra for a suitable functor and the minimal realization arises as the image in the final coalgebra of the given automaton.

It is worth mentioning that when considering formalisms more expressive than ordinary nets, like nets with read or inhibitor arcs, or graph grammars themselves, the dependencies between events in a computation become more complex than causality and conflict. For instance, the possibility of specifying "read-only" operations over the state leads to an asymmetric form of conflict: if an event e reads a resource which is consumed by another event e', then the execution of e' disables e, while the converse does not hold, i.e., e can precede e' in a computation. Hence the causal structure of a process can be described at various degrees of abstraction. At a basic level it can be represented as a deterministic PES, a labelled partial order which describes only the precedences between events, disregarding their origin. But we can also consider finer descriptions in terms of event structures which "observe", for instance, new kind of dependencies arising from the possibility of preserving part of the state in a rewriting step or from the need of maintaining the integrity of its graphical structure. In this paper we will concentrate on the basic case only, just hinting at the other possibilities.

The rest of the paper is structured as follows. First in Section 2 we present the basics of graph grammars and the notion of (deterministic) graph process. In Section 3 we introduce HP-bisimulation for graph grammars. In Section 4 we review causal automata and the corresponding notion of causal bisimulation. Then in Section 5 we show how a (finite-state) graph grammar can be mapped to a (finite) causal automaton via a transformation which respects HP-bisimilarity, thus offering the possibility of deciding HP-bisimulation and of building a minimal automaton for a given grammar up to HP-bisimilarity. Finally, in Section 6 we draw some conclusions and directions for future work. In particular we hint at the possibility of defining different notions of HP-bisimulation which arise by considering finer observations of the causal history of events. Furthermore we give some ideas about the logical counterpart of history preserving bisimulation, presenting a logic in the style of Hennessy-Milner which can be shown to be adequate.

2 Typed graph grammars and processes

This section briefly introduces typed graph grammars [CMR96], a variation of classical DPO graph grammars [Ehr87,CMR+97] where the rewriting takes place on so-called *typed graphs*, namely graphs labelled over a structure (the *graph of types*) that is itself a graph. After some basic definitions and a discussion about the relationship between graph grammars and (contextual) Petri nets, we will recall the notion of *process* for a typed graph grammar [CMR96,BCM98a], which plays a basic role in the definition of history preserving bisimulation.

2.1 Typed graph grammars

Let **Graph** be the category of (directed, unlabelled) graphs and total graph morphisms. For a graph G we will denote by N_G and E_G the (disjoint) sets of

nodes and *edges* of G, and by $s_G, t_G : E_G \to N_G$ its *source* and *target* functions. Given a graph TG, a *typed graph* G over TG is a graph $\langle G \rangle$, together with a morphism $t_G : \langle G \rangle \to TG$. A morphism between TG-typed graphs $f : G_1 \to G_2$ is a graph morphisms $f : \langle G_1 \rangle \to \langle G_2 \rangle$ consistent with the typing, i.e., such that $t_{G_1} = t_{G_2} \circ f$. A typed graph G is called *injective* if the typing morphism t_G is injective. More generally, for a fixed $n \in \mathbb{N}$, the graph is called *n-injective* if for any item x in TG, $|t_G^{-1}(x)| \leq n$, namely if the number of instances of "resources" of any type x is bounded by n. The category of TG-typed graphs and typed graph morphisms is denoted by TG-**Graph** and can be synthetically defined as the comma category (**Graph** $\downarrow TG$).

Fixed a graph TG of types, a *(TG-typed graph) production* $(L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ is a pair of *injective* typed graph morphisms $l : K \to L$ and $r : K \to R$, where $\langle L \rangle$, $\langle K \rangle$ and $\langle R \rangle$ are finite graphs. It is called *consuming* if morphism $l : K \to L$ is not surjective. The typed graphs L, K, and R are called the *left-hand side*, the *interface*, and the *right-hand side* of the production, respectively.

Definition 1 (typed graph grammar). *A* (TG-typed) graph grammar $\mathcal{G}$ *is a tuple* $\langle TG, G_s, P, \pi \rangle$*, where* G_s *is the* start (typed, finite) graph*,* P *is a finite set of* production names*, and* π *is a function which associates a graph production to each production name in* P*. A* labelled *graph grammar is a pair* $\langle \mathcal{G}, \lambda_{\mathcal{G}} \rangle$*, where* $\mathcal{G}$ *is a graph grammar and* $\lambda_{\mathcal{G}} : P \to \mathsf{Act}$ *is a function from* P *to a fixed set of action names* Act*.*

We will denote by $Elem(\mathcal{G})$ the set $N_{TG} \cup E_{TG} \cup P$. Furthermore, we will assume that for each production name $q \in P$ the corresponding production $\pi(q)$ is $L_q \stackrel{l_q}{\leftarrow} K_q \stackrel{r_q}{\rightarrow} R_q$. The components of a graph grammar $\mathcal{G}$ will be denoted by TG, G_s, P and π, possibly with subscripts.

Since in this paper we work only with typed notions, we will usually omit the qualification "typed", and, sometimes, we will not indicate explicitly the typing morphisms. Moreover, we will consider only *consuming* grammars, namely grammars where all productions are consuming: this corresponds, in the theory of Petri nets, to the usual requirement that transitions must have non-empty pre-set.

Definition 2 (direct derivation). *Let* $\mathcal{G}$ *be a graph grammar. Given a typed graph* G*, a production* $q \in P$*, and a* match *(i.e., a graph morphism)* $g : L_q \to G$*, a* direct derivation δ from G to H using q (based on g) *exists, written* $\delta : G \Rightarrow_q H$ *(or* $\delta : G \Rightarrow_{\mathcal{G}} H$*), if and only if the diagram*

$$\begin{array}{ccccc} q: L_q & \xleftarrow{l_q} & K_q & \xrightarrow{r_q} & R_q \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle k} & & \downarrow{\scriptstyle h} \\ G & \xleftarrow[b]{} & D & \xrightarrow[d]{} & H \end{array}$$

can be constructed, where both squares have to be pushouts in TG-**Graph**. *For a labelled grammar, if* $\lambda_{\mathcal{G}}(q) = a$*, in this situation we write* $\delta : G \Rightarrow_q^a H$ *(or* $\delta : G \Rightarrow_{\mathcal{G}}^a H$*).*

A derivation *in $\mathcal{G}$ is a sequence of direct derivations (in $\mathcal{G}$) beginning from the start graph G_s.*

Roughly speaking, the rewriting step removes from the graph G the items of the left-hand side which are not in the image of the interface, namely $L_q - l_q(K_q)$, producing in this way the graph D. Then the items in the right-hand side which are not in the image of the interface, namely $R_q - r_q(K_q)$, are added to D, obtaining the final graph H. Notice that the interface graph K_q (common part of L_q and R_q) specifies both what is preserved and how the added subgraph has to be connected to the remaining part. Given a match $g : L_q \to G$ as in the above diagram, the pushout complement of l_q and g (i.e., a graph D with morphisms k and b such that the left square is a pushout) exists if and only if the *gluing condition* is satisfied. This consists of two parts:

- *identification condition*, requiring that if two distinct nodes or edges of L_q are mapped by g to the same image, then both must be in the image of l_q;
- *dangling condition*, stating that no edge in $G - g(L_q)$ should be incident to a node in $g(L_q - l_q(K_q))$ (because otherwise the application of the production would leave such an edge "dangling").

2.2 Relation with Petri nets.

Many definitions and constructions in this paper are better understood keeping in mind the relation between Petri nets and DPO graph grammars. The basic observation (which belongs to the folklore, see, e.g., [Cor96]) is that a P/T Petri net is essentially a rewriting system on multisets, and that, given a set A, a multiset of A can be represented as a discrete graph typed over A. In this view a P/T net can be seen as a graph grammar acting on discrete graphs typed over the set of places, the productions being (some encoding of) the net transitions: a marking is represented by a set of nodes (tokens) labelled by the place where they are, and, for example, the Petri net transition t in the top part of Fig. 1 is represented by the graph production depicted aside. Notice that the interface is empty since nothing is explicitly preserved by a net transition. It is not difficult to show that this encoding satisfies the properties one would expect, namely that there is a precise correspondence between transition firings in the original net and derivations in the corresponding grammar.

The considered encoding of nets into grammars enlightens the dimensions in which graph grammars properly extend nets. First of all grammars allow for a more structured description of state, that is a general graph rather than a multiset (discrete graph). Furthermore, graph grammars allow for productions where the interface graph may not be empty, thus specifying a "context" consisting of items that have to be present for the productions to be applied, but which are not affected by the application. The context can be interpreted as a part of the state which is accessed in a "read-only" way by the rewriting step, and, consistently with this view, several rewriting steps can be applied in parallel sharing (part of) the context. In this respect, graph grammars are closer to some

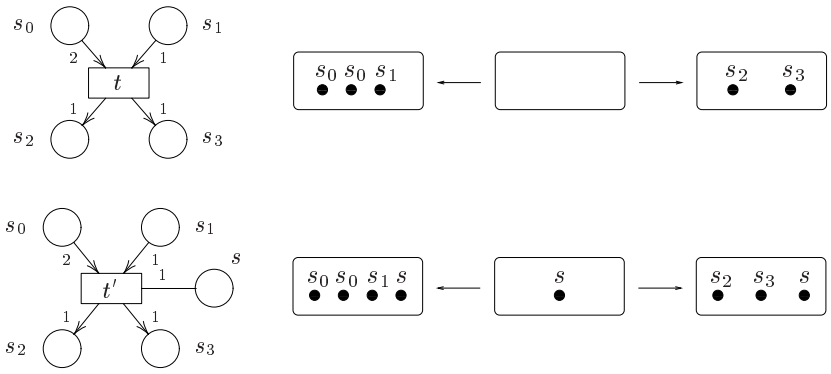

Fig. 1. Petri net transitions and corresponding DPO productions.

generalizations of Petri nets in the literature, called nets with read (test) arcs or contextual nets (see, e.g., [JK95,MR95,Vog97]), which generalize classical nets by adding the possibility of checking for the presence of tokens which are not consumed. Concretely, a transition of a contextual net, besides the pre-set and post-set, has also a context specifying tokens which must be present to enable the transitions, but which are not affected by the firing. For instance, in the bottom left part of Fig. 1, place s is a context for transition t', and hence t', to be enabled, requires a token in s which is not consumed. It is clear that the context of a contextual net transition closely corresponds to the interface graph of a DPO production, so that contextual nets can be seen as special graph grammars acting on discrete graphs, but with productions which can have a non-empty interface (see the encoding of transition t' as a DPO graph production in the bottom right part of Fig. 1).

2.3 Processes of typed graph grammars

Graph processes [CMR96,BCM98a] arise from the idea of equipping graph grammars with a semantics which on the one hand explicitly represents events and relationships among them, and on the other hand uses graph grammars themselves as semantic domain. Analogously to what happens for Petri nets, a *graph process* of a graph grammar $\mathcal{G}$ is defined as an "occurrence grammar" $\mathcal{O}$, i.e., a grammar satisfying suitable acyclicity and conflict freeness constraints, equipped with a mapping from $\mathcal{O}$ to $\mathcal{G}$. This mapping is used to associate to the derivations in $\mathcal{O}$ corresponding derivations in $\mathcal{G}$. The basic property of a graph process is that the derivations in $\mathcal{G}$ which are in the range of such mapping constitute a full class of *shift*-equivalent derivations, i.e., of derivations which differ only for the order of "independent" rewriting steps. Therefore the process can be regarded as an abstract representation of such a class and plays a role similar to a *canonical derivation* [Kre77].

It is worth remarking that in the definitions of occurrence grammar and of graph process, later in this section, we will slightly depart from the original proposal in [CMR96], as we will use explicitly the relation of asymmetric conflict (as we already did, e.g., in [BCM99]). A first step towards the definition of (deterministic) occurrence grammar is a suitable notion of safety for grammars [CMR96], generalizing that for P/T nets. More generally, we extend to graph grammars the notion of n-safety, which amounts to the property of being finite-state.

Definition 3 (safe grammar). *For a fixed $n \in \mathbb{N}$, we say that a graph grammar $\mathcal{G}$ is* n-safe *if, for all H such that $G_s \Rightarrow^* H$, H is n-injective. A* 1*-safe grammar will be simply called* safe.

The definition can be understood by thinking of nodes and edges of the type graph as a generalization of places in Petri nets. In this view the number of different items of a graph which are typed on a given item of the type graph corresponds to the number of tokens contained in a place, and thus the condition of (n-) safety for a Petri net marking, which requires each place to contain at most 1 (n) tokens, is generalized to typed graphs by the (n-) injectivity of the typing morphism. In the following, to mean that a graph grammar $\mathcal{G}$ is n-safe for some $n \in \mathbb{N}$ we will simply say that $\mathcal{G}$ is n-safe.

In particular, safe graph grammars can be given a visual net-like representation, where the items of the type graph and the productions play, respectively, the role of places and transitions. In fact, if $\mathcal{G}$ is a safe graph grammar, then each graph $\langle\langle G\rangle, t_G\rangle$ reachable in $\mathcal{G}$ can be identified with the subgraph $t_G(\langle G\rangle)$ of the type graph TG and thus it can be represented by suitably decorating the nodes and edges of the type graph. Concretely, a node is drawn as a filled circle, if it belongs to $t_G(\langle G\rangle)$ and as an empty circle, otherwise, while an edge is drawn as a plain (bold) line if it belongs to $t_G(\langle G\rangle)$ and as a dotted line otherwise. For instance, in the right part of Fig. 2, forgetting about the productions q_i and the corresponding connections, one can see a representation of the start graph G_s of the graph grammar presented in the left part: nodes B, C, D are filled since they belong to G_s, while node A is empty and edge L is dotted since they are not in G_s.

With this identification, in each derivation of a safe grammar beginning from the start graph a production q can be applied only to the subgraph of the type graph which is the image via the typing morphism of its left-hand side, i.e., to $t_{L_q}(\langle L_q\rangle)$. Therefore according to its typing, we can think that a production *produces*, *preserves* and *consumes* items of the type graph. Using a net-like language, we speak of *pre-set* ${}^\bullet q$, *context* $\underline{q}$ and *post-set* $q^\bullet$ of a production q. This is expressed by representing productions as arrow-shaped boxes, connected to the consumed and produced resources by incoming and outgoing arrows, respectively, and to the preserved resources by undirected lines. Fig. 2 presents a safe graph grammar and its net-like pictorial representation. To have a lighter presentation in the examples, we assume that the action label of each production q in grammar $\mathcal{G}$ is the name q of the production itself.

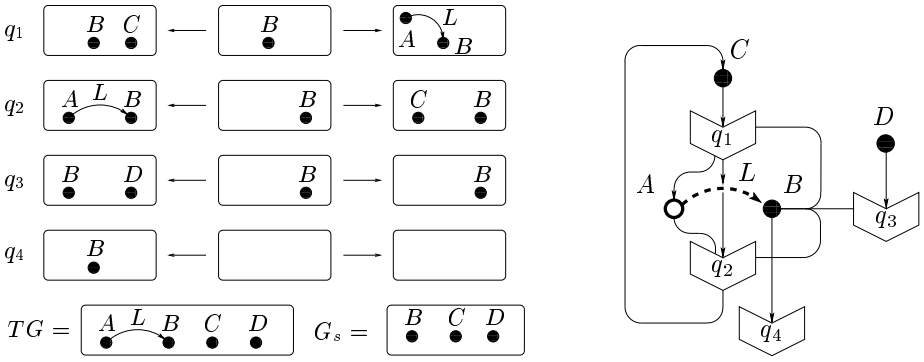

Fig. 2. A grammar $\mathcal{G}$ and its net-like representation.

The notions of pre-set, post-set and context of a production have a clear interpretation only for safe grammars. However for technical reasons it is preferable to define them for general graph grammars.

Definition 4 (pre-set, post-set, context). *Let $\mathcal{G}$ be a graph grammar. For any $q \in P$ we define*

$$^\bullet q = t_{L_q}(\langle L_q \rangle - l_q(\langle K_q \rangle)) \qquad q^\bullet = t_{R_q}(\langle R_q \rangle - r_q(\langle K_q \rangle))$$
$$\underline{q} = t_{K_q}(\langle K_q \rangle)$$

seen as sets of nodes and edges, and we say that q consumes, produces *and* preserves *items in $^\bullet q$, $q^\bullet$ and $\underline{q}$, respectively. Similarly for a node or an edge x in TG we write $^\bullet x$, $\underline{x}$ and $x^\bullet$ to denote the sets of productions which produce, preserve and consume x, respectively.*

For instance, for grammar $\mathcal{G}$ in Fig. 2, the pre-set, context and post-set of production q_1 are $^\bullet q_1 = \{C\}$, $\underline{q_1} = \{B\}$ and $q_1{}^\bullet = \{A, L\}$, while for the node B, $^\bullet B = \emptyset$, $\underline{B} = \{q_1, q_2, q_3\}$ and $B^\bullet = \{q_4\}$.

We next introduce the relations of causality and asymmetric conflict, representing the dependencies between events in a graph grammar.

Definition 5 (causal relation). *The* causal relation *of a grammar $\mathcal{G}$ is the binary relation $<$ over $Elem(\mathcal{G})$ defined as the least transitive relation satisfying: for any node or edge x in the type graph TG and for productions $q, q' \in P$*

1. *if $x \in {}^\bullet q$ then $x < q$;*
2. *if $x \in q^\bullet$ then $q < x$;*
3. *if $q^\bullet \cap \underline{q'} \neq \emptyset$ then $q < q'$.*

As usual $\leq$ denotes the reflexive closure of $<$. Moreover, for $x \in Elem(\mathcal{G})$ we write $\lfloor x \rfloor$ for the set of causes of x in P, namely $\{q \in P \mid q \leq x\}$. We will denote by $Min(\mathcal{G})$ and $Max(\mathcal{G})$ the sets of items of TG which are minimal and maximal, resp., with respect to $\leq$.

The first two clauses of the definition of relation $<$ are obvious. The third one formalizes the fact that if an item is generated by q and preserved by q', then q', to be applied, requires that q had already been applied.

Notice that the fact that an item is preserved by q and consumed by q', i.e., $\underline{q} \cap {}^\bullet q' \neq \emptyset$, does not imply $q < q'$. Actually, since q must precede q' in any computation where both appear, in such computations q acts as a cause of q'. However, differently from a true cause, q is not necessary for q' to be applied. Therefore we can think of the relation between the two productions as a *weak* form of *causal dependency*. Equivalently, we can observe that the application of q' prevents q to be applied, so that q can never follow q' in a derivation. But the converse is not true, since q *can* be applied before q'. Thus this situation can also be interpreted naturally as an *asymmetric conflict* between the two productions (see, e.g., [BCM99]). For instance, in the grammar $\mathcal{G}$ of Fig. 2 there is an asymmetric conflict between productions q_3 and q_4, since $B \in \underline{q_3} \cap {}^\bullet q_4$.

Definition 6 (asymmetric conflict). *The* asymmetric conflict relation *of a grammar $\mathcal{G}$ is the binary relation $\nearrow$ over the set P of productions, defined by:*

1. *if $\underline{q} \cap {}^\bullet q' \neq \emptyset$ then $q \nearrow q'$;*
2. *if ${}^\bullet q \cap {}^\bullet q' \neq \emptyset$ and $q \neq q'$ then $q \nearrow q'$;*
3. *if $q < q'$ then $q \nearrow q'$.*

Point (1) has been discussed above. By point (2), the symmetric conflict arising when two productions q and q' consume a common resource is represented as an asymmetric conflict in both directions $q \nearrow q'$ and $q' \nearrow q$. Finally, point (3) formalizes the intuition that asymmetric conflict can be seen as a weak form of causality and thus it is implied by causality.

A *(deterministic) occurrence grammar* is now defined as a special grammar satisfying suitable requirements of acyclicity and absence of conflicts, which will allow to view its productions as single event occurrences.

Definition 7 ((deterministic) occurrence grammar). *A* (deterministic) occurrence grammar *is a graph grammar $\mathcal{O} = \langle TG, G_s, P, \pi \rangle$ such that*

1. *each edge or node x in TG is created by at most one production in P, namely $|{}^\bullet x| \leq 1$;*
2. *$\nearrow_{\mathcal{O}}$ is acyclic and finitary; thus $(\nearrow_{\mathcal{O}})^*$ and $\leq_{\mathcal{O}}$ are finitary partial orders;*[1]
3. *$Min(\mathcal{O})$ and $Max(\mathcal{O})$, with the graphical structure inherited from TG, are well-defined subgraphs of TG; furthermore the start graph G_s coincides with $Min(\mathcal{O})$ (typed by the inclusion);*
4. *for each production $q : L_q \overset{l_q}{\leftarrow} K_q \overset{r_q}{\rightarrow} R_q$, the typing t_{L_q} is injective on the "consumed part" $\langle L_q \rangle - l_q(\langle K_q \rangle)$, and similarly t_{R_q} is injective on the "produced part" $\langle R_q \rangle - r_q(\langle K_q \rangle)$.*

[1] A relation $r \subseteq X \times X$ is called *finitary* if for any $x \in X$ the set $\{y \in X : y\, r\, x\}$ is finite. Furthermore r^* denotes the reflexive and transitive closure of a relation r.

Intuitively, conditions (1)–(4) recast in the framework of graph grammars the analogous conditions of occurrence contextual nets [BCM98b,VSY98]. In particular the acyclicity of $\nearrow$ corresponds to the requirement of absence of conflicts in occurrence Petri nets. Condition (4) is closely related to safety and requires that each production consumes and produces items with "multiplicity" one. Observe that, together with acyclicity of $\nearrow$, it disallows the presence of some productions which surely could never be applied, because they fail to satisfy the identification condition with respect to the typing morphism.

Since the start graph of an occurrence grammar $\mathcal{O}$ is determined by $Min(\mathcal{O})$, we often do not mention it explicitly. Observe that, by the defining conditions, each occurrence grammar is safe.

A (deterministic) process for a graph grammar, analogously to what happens for ordinary and contextual nets, is an occurrence grammar endowed with a mapping to the original grammar and it can be seen as a representative of a set of shift equivalent derivations of $\mathcal{G}$.

Definition 8 (graph process). *Let $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ be a typed graph grammar. A* (finite marked) process *for $\mathcal{G}$ is a mapping $\varphi : \mathcal{O}_\varphi \to \mathcal{G}$, such that $\mathcal{O}_\varphi = \langle TG_\varphi, P_\varphi, \pi_\varphi \rangle$ is an occurrence grammar and $\varphi = \langle \varphi_T, \varphi_P, \iota_\varphi \rangle$, where*

1. *$\varphi_T : TG_\varphi \to TG$ is a graph morphism;*
2. *$\varphi_P : P_\varphi \to P$ is a function mapping each production $q' : (L' \leftarrow K' \to R')$ in P_φ to an isomorphic production $q = \varphi_P(q') : (L \leftarrow K \to R)$ in P and*
3. *the ι_φ component associates to each production $q' \in P_\varphi$ a triple of isomorphisms $\iota_\varphi(q') = \langle \iota^L_\varphi(q') : L \to L', \iota^K_\varphi(q') : K \to K', \iota^R_\varphi(q') : R \to R' \rangle$, making the diagram in the left part of Fig. 3 commute. Furthermore it includes an isomorphism $\iota^s_\varphi : \langle G_s \rangle \to \langle G_{s\,\varphi} \rangle$, which makes the diagram in the right part of Fig. 3 commute.*

We denote by $Min(\varphi)$ and $Max(\varphi)$ the graphs $Min(\mathcal{O})$ and $Max(\mathcal{O})$. The same graphs typed over TG by the restrictions of φ_T are denoted by ${}^\bullet\varphi$ and $\varphi^\bullet$ and called, respectively, the source *and* target *graphs of the process (observe that ${}^\bullet\varphi \simeq G_s$).*

We call initial process *of $\mathcal{G}$ any process φ with an empty set of productions (and thus with $TG_\varphi \simeq \langle G_s \rangle$).*

For instance, Fig. 4 presents several processes of grammar $\mathcal{G}$ in Fig. 2 (for the moment ignore the fact that processes are partly shaded). For each process we only give the net-like representation of the underlying occurrence grammar. The mapping over the original grammar is implicitly represented by the labelling.

It is worth observing, that because of the dangling condition, a production q which consumes a node n can be applied only if there are no edges with source or target in n which remain dangling after the application of q. In other words, the presence of an edge e with source or target in n such that $e \notin {}^\bullet q$ *inhibits* the application of q (in [Bal00] this observation represents the basis to establish a close correspondence between graph grammars and nets with inhibitor arcs). For example, in the grammar $\mathcal{G}$ of Fig. 2, edge L inhibits production q_4 since

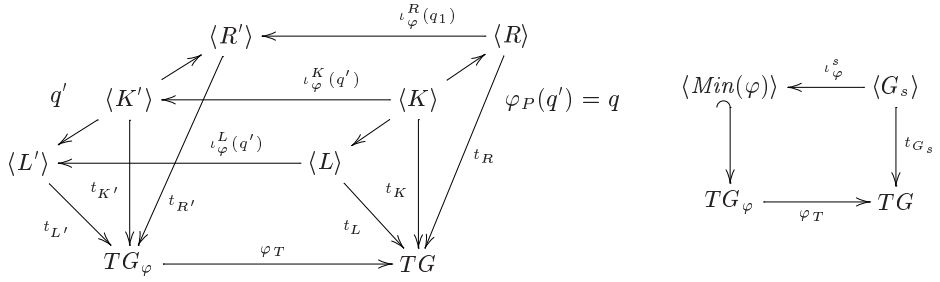

Fig. 3. Graph processes.

q_4 consumes node B which is the target of L. Observe that productions q_1 and q_2, respectively, produce and consume such an edge, and therefore, once q_1 has been applied, q_4 can occur only after the application of q_2. That is, in a process where all q_1, q_2 and q_4 are applied, they must occur exactly in this order. Indeed, q_1 and q_2, to act on edge L must produce or preserve its target node B (in this case they both preserve B) and thus, by definition of asymmetric conflict, we have $q_1 \nearrow q_4$ and $q_2 \nearrow q_4$. Hence in a deterministic computation where q_1, q_2 and q_4 occur, the relation $\nearrow$ already imposes the correct order of application for them. This holds in general: there is no need to consider explicitly the inhibiting effects due to the dangling condition in a graph process, as they are subsumed by the asymmetric conflict relation. Note that this does not hold in the case of inhibitor nets, for which the definition of process becomes more involved [Bal00].

3 History preserving bisimulation on graph grammars

As mentioned in the introduction, the theory of concurrency for graph grammars has been deeply studied and a number of concurrent operational models for graph grammars has been proposed in the literature. However, until now the problem of defining suitable abstract behavioural semantics for graph grammars has been given little attention.

Observe that the notions of (labelled) graph grammar and of direct derivation are enough to define ordinary bisimulation over graph grammars. Intuitively, two systems are bisimilar if every action of the first one can be simulated by an action of the second one, and vice versa. Formally, given two graph grammars $\mathcal{G}_1$ and $\mathcal{G}_2$, a *simulation* of $\mathcal{G}_1$ into $\mathcal{G}_2$ is a relation $\mathcal{R}$ between (abstract) graphs typed over TG_1 and TG_2, respectively, such that if $G_1 \,\mathcal{R}\, G_2$ and $G_1 \Rightarrow^a_{\mathcal{G}_1} H_1$ then there exists H_2 such that $G_2 \Rightarrow^a_{\mathcal{G}_2} H_2$ and $H_1 \,\mathcal{R}\, H_2$. The relation $\mathcal{R}$ is a *bisimulation* if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are simulations, and $\mathcal{G}_1$ and $\mathcal{G}_2$ are *bisimilar* if their initial graphs are related by a bisimulation.

Ordinary bisimulation is an "interleaving" equivalence, in the sense that it is not able to capture the concurrency properties of a system. For instance, it

equates the parallel composition of two systems and the nondeterministic choice of their possible sequentializations. Here we are interested in the so-called *history preserving bisimulation*, a behavioural equivalence which, instead, takes into account the dependencies among events. Roughly speaking, it equates two systems if each action of the first one can be simulated by an action of the second one with an equivalent history, and vice versa. In this section, relying on the work already developed on contextual nets [BCM00a], the notion of graph process is taken as a basis to extend this idea to the case of graph grammars. As a description of the "concurrent structure" of a computation we consider the *(labelled) prime event structure* (PES) underlying a process, i.e., a partially ordered structure where the elements represent events (occurrences of productions) and the partial order represents the dependencies between events. This amounts to observing the precedences between events, without taking care of their origin. We mentioned that such precedences can arise both as ordinary causal dependencies, induced by the flow of information, and as dependencies induced by read-only operations and inhibiting effects related to the dangling condition. Other finer observations, taking into account the diverse nature of these precedences, are conceivable and will be discussed in the conclusions.

The basic ingredient for the definition of history preserving bisimulation is a transition system, associated to each graph grammar, where states are processes. The initial state is the empty process, corresponding to the start graph, and any process can be extended by the "application" of any production which is enabled in its final (maximal) graph.

Definition 9 (process moves). *Given two processes φ and φ' of a labeled graph grammar $\mathcal{G}$, we write $\varphi \xrightarrow[e]{a} \varphi'$, saying that* φ moves to φ' performing action a, *if*

- $P_{\varphi'} = P_\varphi \cup \{e\}$, *with* $e \notin P_\varphi$ *and* $\lambda_{\mathcal{G}}(\varphi'_P(e)) = a$;
- TG_φ *is a subgraph of* $TG_{\varphi'}$;
- ${}^\bullet e$ *and* $\underline{e}$ *are included in* $Max(\varphi)$;
- φ_T, φ_P, π_φ *and* ι_φ *and are the restrictions to* $\mathcal{O}_\varphi$ *of the components of* φ'.

Fig. 4 presents a sequence of processes φ_i for the grammar $\mathcal{G}$ of Fig. 2, such that each φ_i moves to φ_{i+1} (the process φ_3 is not represented explicitly). For instance $\varphi_0 \xrightarrow[e_1]{q_1} \varphi_1$.

To each process φ of a graph grammar $\mathcal{G}$ we can naturally associate a (deterministic labelled) PES where events are the productions of the underlying occurrence graph grammar, causality is the transitive closure of the asymmetric conflict relation and each event is labelled by the action label of the corresponding production in $\mathcal{G}$.

Definition 10 (prime event structure for processes). *Let φ be a process of a labelled graph grammar $\mathcal{G}$. The* PES associated to φ *is defined as:*

$$ev(\varphi) = \langle P_\varphi, (\nearrow_\varphi)^*, \lambda_{\mathcal{G}} \circ \varphi_P \rangle.$$

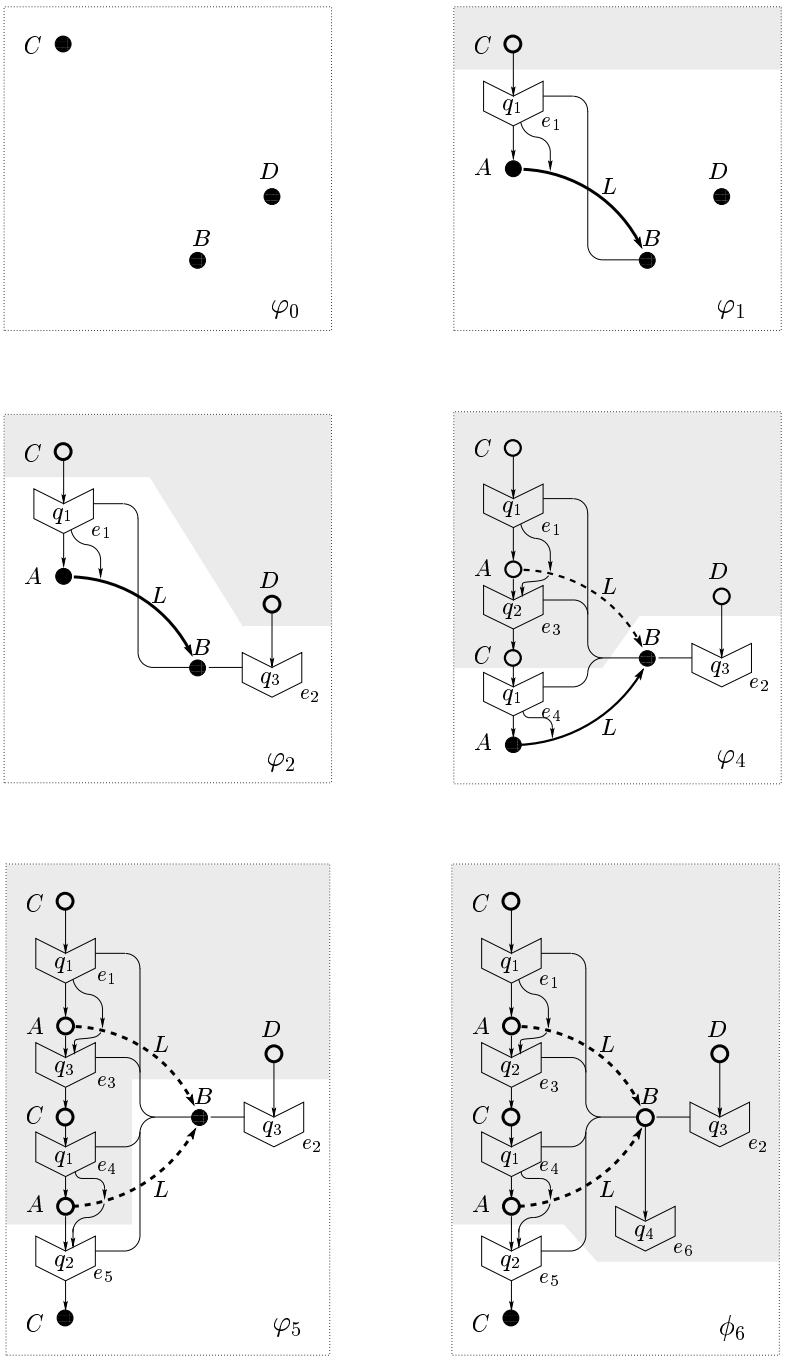

Fig. 4. A sequence of process moves for grammar $\mathcal{G}$ in Fig. 2, starting from an initial process. For any process the non-shaded part represents the corresponding partial process.

Based on the notions of process and of event structure associated to a process, *history preserving (HP-) bisimulation* is readily defined.

Definition 11 (HP-bisimulation). *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be labelled graph grammars. An* HP-simulation *$\mathcal{R}$ of $\mathcal{G}_1$ in $\mathcal{G}_2$ is a set of triples $\langle \varphi_1, f, \varphi_2 \rangle$ where φ_i is a process of $\mathcal{G}_i$ for $i \in \{1,2\}$, and $f : ev(\varphi_1) \to ev(\varphi_2)$ is an isomorphism of* PES*'s, such that*

1. $\langle \varphi_0(\mathcal{G}_1), \emptyset, \varphi_0(\mathcal{G}_2) \rangle \in \mathcal{R}$, *with $\varphi_0(\mathcal{G}_i)$ initial process of $\mathcal{G}_i$ for $i \in \{1,2\}$;*
2. $\langle \varphi_1, f, \varphi_2 \rangle \in \mathcal{R} \wedge \varphi_1 \xrightarrow[e_1]{a} \varphi_1' \Rightarrow \varphi_2 \xrightarrow[e_2]{a} \varphi_2' \wedge \langle \varphi_1', f', \varphi_2' \rangle \in \mathcal{R} \wedge f'_{|ev(\varphi_1)} = f$.

An HP-bisimulation *between $\mathcal{G}_1$ and $\mathcal{G}_2$ is a set of triples $\mathcal{R}$ such that $\mathcal{R}$ and $\mathcal{R}^{-1} = \{\langle \varphi_2, f^{-1}, \varphi_1 \rangle : \langle \varphi_1, f, \varphi_2 \rangle \in \mathcal{R}\}$ are HP-simulations. The labelled graph grammars $\mathcal{G}_1$ and $\mathcal{G}_2$ are* HP-bisimilar*, written $\mathcal{G}_1 \sim_{hp} \mathcal{G}_2$, if there is an HP-bisimulation $\mathcal{R}$ between $\mathcal{G}_1$ and $\mathcal{G}_2$.*

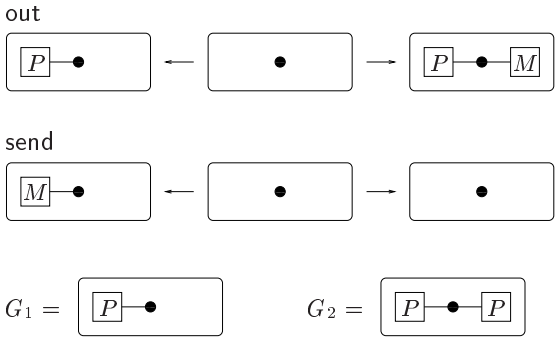

Fig. 5. Modelling the transmission of messages.

Both ordinary and history preserving bisimulation represent an abstraction of the concrete operational semantics based on the shift equivalence, in the sense that any two graph grammars with the same concurrent model of computation [Roz97] are bisimilar and HP-bisimilar.

Concerning the relationship between ordinary bisimulation and history preserving bisimulation over graph grammars, quite obviously, being based on a more detailed observation, the latter is finer than the former. To have a better understanding of the difference between the two semantics consider the productions in Fig. 5, which are intended to model the generation and delivery of messages in a single node of a network. Edges labelled by P and M represent processes and messages, respectively. Rule out represents the generation of a message by a process, while rule send represents the delivery of the message: since we consider a single node of the newtwork the message which is sent simply disappears. This minimal subsystem is only aimed at illustrating some concepts in a setting

as simple as possible: to make the model more realistic one could make explicit the reception of messages, the entire network could be represented as a graph and new rules could be added to represent message delivery over the network. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be the graph grammars with rules out and send, and with initial graph G_1 and G_2, respectively. It is easy to see that $\mathcal{G}_1$ and $\mathcal{G}_2$ are bisimilar, but not HP-bisimilar. In fact, each out operation performed by a process causally depends on the previous one and each send operation causally depends on the out operation which generated the corresponding message. Therefore in $\mathcal{G}_1$ there is a single chain of causally dependent out operations, while in $\mathcal{G}_2$ there can be two concurrent out operations, as shown by the corresponding event structures in Fig. 6.

It is worth observing that extending the grammars with an explicit rule modelling the receive operation, the (deterministic components of the) event structures would closely correspond to Message Sequence Charts [RGG96], a graphical and textual language for the description and specification of the interactions between system components.

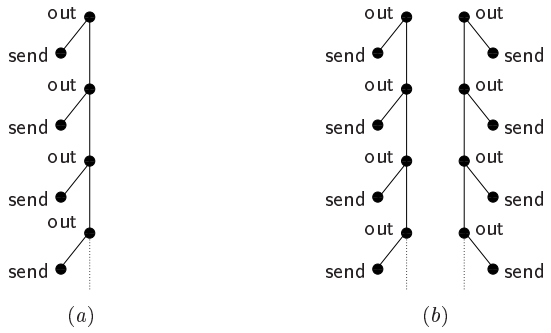

Fig. 6. The event structures corresponding to the graph grammars (a) $\mathcal{G}_1$ and (b) $\mathcal{G}_2$.

4 Causal automata

In this section we review *causal automata*, a generalization of ordinary automata introduced in [MP97] as an appropriate model for history dependent formalisms (see also [MP98,Pis99], where more general models, called HD-automata, are presented). Here causal automata will be used as an abstract framework where HP-bisimulation over graph grammars can be studied. In particular the decidability of HP-bisimulation for finite-state graph grammars will be proved by showing that the problem can be reduced to the bisimilarity of finite causal automata.

Causal automata extend ordinary automata by allowing sets of names to appear explicitly in the states and labels of the automata. The names are local,

namely they do not have a global identity, and the correspondence between the names of the source and those of the target states of each transition is specified explicitly. This allows for a compact representation of systems since states differing only for the concrete identity of the names can be identified. Moreover causal automata provide a mechanism for the generation of new names: the problem of choosing a fresh name disappears in this formalism where a new name is simply a name which does not correspond to any name in the source state. In the specific case of Petri nets and graph grammars, names are identities of transitions in a process (events) and the correspondence between names allows to represent causal dependencies.

Definition 12 (causal automaton). *Let $\mathcal{N}$ be a fixed infinite countable set of names (event names) and let* Act *be a fixed set of labels. A* causal automaton *is a tuple $\mathcal{A} = \langle Q, n, \longmapsto, q_0 \rangle$, where*

- *Q is the set of* states*;*
- *$n : Q \to \mathcal{P}_{fin}(\mathcal{N})$ is a function associating to each state a finite set of names;*
- *$\longmapsto$ is a set of transitions, each of the form $q \overset{a}{\underset{M}{\longmapsto}}_{\sigma} q'$, with*
  - *q, q' the source and target states;*
  - *$a \in$ Act the label;*
  - *$M \subseteq n(q)$ the set of dependencies of the transition;*
  - *$\sigma : n(q') \hookrightarrow n(q) \cup \{\star\}$ the injective inverse renaming function;*
- *$q_0 \in Q$ is the initial state; it is required that $n(q_0) = \emptyset$.*

For each state $q \in Q$ the set of names $n(q)$ is used to represent the past events which can (but not necessarily will) be referenced by future transitions. Conceptually, each transition $q \overset{a}{\underset{M}{\longmapsto}}_{\sigma} q'$ depends on the past events mentioned in M. Due to the local scope of names, the function $\sigma : n(q') \hookrightarrow n(q) \cup \{\star\}$ is needed to relate the names of the target state to those of the source. The event mapped to $\star$ (if any) represents the new event generated by the considered transition. In the following the components of a causal automaton will be often denoted by using the name of the automaton as subscript.

The notion of bisimulation on causal automata (CA-bisimulation) takes into account the fact that a state has attached a set of local names. Hence a bisimulation not only relates states, but also the corresponding sets of local names.

Definition 13 (CA-bisimulation). *Let $\mathcal{A}$ and $\mathcal{B}$ be two causal automata. A* CA-simulation *$\mathcal{R}$ of $\mathcal{A}$ in $\mathcal{B}$ is a set of triples $\langle q, \delta, p \rangle$, where $q \in Q_{\mathcal{A}}$, $p \in Q_{\mathcal{B}}$ and δ is a partial injective function from $n_{\mathcal{A}}(q)$ to $n_{\mathcal{B}}(p)$, such that*

1. *$\langle q_{0\mathcal{A}}, \emptyset, q_{0\mathcal{B}} \rangle \in \mathcal{R}$;*
2. *if $\langle q, \delta, p \rangle \in \mathcal{R}$ and $q \overset{a}{\underset{M}{\longmapsto}}_{\sigma} q'$ in $\mathcal{A}$ then*
   - *$p \overset{a}{\underset{\delta(M)}{\longmapsto}}_{\rho} p'$ in $\mathcal{B}$ for some p' and*

– $\langle q', \delta', p' \rangle \in \mathcal{R}$ *for some* δ' *such that* $\delta^\star \circ \sigma = \rho \circ \delta'$, *where* $\delta^\star$ *is defined as* $\delta \cup \{(\star, \star)\}$ *(see the diagram below).*

$$\begin{array}{ccc} n_{\mathcal{A}}(q) \cup \{\star\} & \xrightarrow{\delta^\star} & n_{\mathcal{B}}(p) \cup \{\star\} \\ {\scriptstyle\sigma}\uparrow & & \uparrow{\scriptstyle\rho} \\ n_{\mathcal{A}}(q') & \xrightarrow[\delta']{} & n_{\mathcal{B}}(p') \end{array}$$

A CA-bisimulation *between* $\mathcal{A}$ *and* $\mathcal{B}$ *is a set of triples* $\mathcal{R}$ *such that* $\mathcal{R}$ *and* $\mathcal{R}^{-1} = \{\langle p, \delta^{-1}, q \rangle : \langle q, \delta, p \rangle \in \mathcal{R}\}$ *are CA-simulations. The automata* $\mathcal{A}$ *and* $\mathcal{B}$ *are* CA-bisimilar, *written* $\mathcal{A} \sim_{ca} \mathcal{B}$, *if there exists a bisimulation* $\mathcal{R}$ *between* $\mathcal{A}$ *and* $\mathcal{B}$.

In [MP97] an algorithm has been proposed for checking the CA-bisimilarity of (finite) causal automata. Given a causal automaton $\mathcal{A}$, first the "useless" names, i.e., names never referenced by future transitions, are removed from the states of the automaton. For instance, in the case of Petri nets, the useless names are the events that belong to a state because they have generated a token which still exists, but which is never used later by any other event. Then the basic step of the algorithm constructs an ordinary labelled transition system $Unf(\mathcal{A})$, called the unfolding of $\mathcal{A}$, such that $\mathcal{A} \sim_{ca} \mathcal{B}$ iff the associated transition systems $Unf(\mathcal{A})$ and $Unf(\mathcal{B})$ are bisimilar. Finally, standard algorithms (e.g., a partition/refinement algorithm) can be used to verify bisimilarity on the ordinary transition systems or to obtain a minimal equivalent transition system.

As mentioned in the introduction, some more recent works [Pis99,MP00] show that, considering a generalization of the model, the so-called automata with symmetries, a minimal realization exists and can be constructed in the class of causal (or, more generally, HD) automata themselves. A causal automaton can be seen as a coalgebra of a suitable functor and the minimal realization arises as the image in the final coalgebra of the given automaton.

Abstraction homomorphisms [CFM83,Cas87], which are also called zig-zag morphisms [vB84] or transition preserving homomorphisms [FM90], are defined in the setting of ordinary automata as morphisms which "preserve" and "reflect" transitions. The existence of an abstraction homomorphism ensures that the source and target automata are bisimilar. The next definition generalizes this idea to causal automata.

Definition 14 (abstraction homomorphism). *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be causal automata. An* abstraction homomorphism $\mathsf{h} : \mathcal{A} \to \mathcal{B}$ *is a pair* $\mathsf{h} = \langle h, \{h_q\}_{q \in Q_{\mathcal{A}}} \rangle$ *where* $h : Q_{\mathcal{A}} \to Q_{\mathcal{B}}$ *is a function and for all* $q \in Q_{\mathcal{A}}$, $h_q : n_{\mathcal{B}}(h(q)) \to n_{\mathcal{A}}(q)$ *is an injective function, such that* $h(q_{0\mathcal{A}}) = q_{0\mathcal{B}}$ *and*

– *if* $q \xrightarrow[M]{a}_\sigma q'$ *in* $\mathcal{A}$ *then* $h(q) \xrightarrow[h_q^{-1}(M)]{a}_\rho h(q')$ *in* $\mathcal{B}$, *with* $\sigma \circ h_{q'} = h_q^\star \circ \rho$ *(see Fig. 7.(a));*
– *if* $h(q) \xrightarrow[M]{a}_\rho p'$ *in* $\mathcal{B}$ *then* $q \xrightarrow[h_q(M)]{a}_\sigma q'$ *in* $\mathcal{A}$ *for some* q', *with* $h(q') = p'$ *and* $\sigma \circ h_{q'} = h_q^\star \circ \rho$ *(see Fig. 7.(b)).*

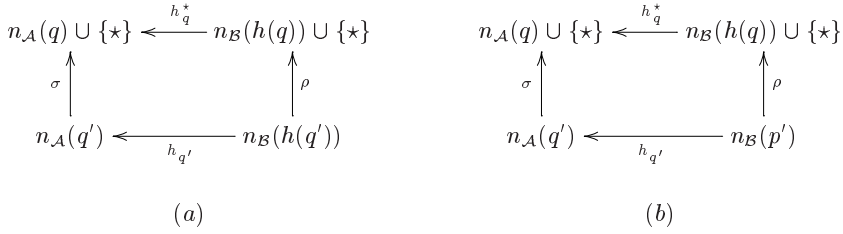

Fig. 7. Diagrams for abstraction homomorphisms.

Intuitively, via an abstraction homomorphism $\mathsf{h} : \mathcal{A} \to \mathcal{B}$ several states of $\mathcal{A}$ can collapse into a single state of $\mathcal{B}$, in a way that respects the behaviour and the naming. In particular, observe that for any state $q \in Q_{\mathcal{A}}$, the function h_q maps the names of $h(q)$ (in $\mathcal{B}$) into the names of q (in $\mathcal{A}$). The idea is that the names of q which are not in the image of h_q can be safely removed, obtaining an equivalent system, namely, in a sense, they are "useless". Indeed, also in this setting, the existence of an abstraction homomorphism $\mathsf{h} : \mathcal{A} \to \mathcal{B}$ is sufficient to conclude the bisimilarity of $\mathcal{A}$ and $\mathcal{B}$.

Lemma 15. *Let $\mathcal{A}$ and $\mathcal{B}$ be causal automata. If there exists an abstraction homomorphism $\mathsf{h} : \mathcal{A} \to \mathcal{B}$ then $\mathcal{A} \sim_{ca} \mathcal{B}$.*

It is worth observing that, as for ordinary automata, the above lemma does not provide a necessary condition. In [MP98], following the approach of [JNW96], abstraction homomorphisms have been described as open maps in a category of causal automata and it has been shown that two causal automata are CA-bisimilar if and only if they are related by a span of open maps.

5 Deciding HP-bisimulation on graph grammars

In this section we show that it is possible to associate to any graph grammar $\mathcal{G}$ a causal automaton $\mathcal{A}_{hp}(\mathcal{G})$, via a construction which respects HP-bisimulation, i.e., such that two graph grammars $\mathcal{G}_1$ and $\mathcal{G}_2$ are HP-bisimilar if and only if $\mathcal{A}_{hp}(\mathcal{G}_1)$ and $\mathcal{A}_{hp}(\mathcal{G}_2)$ are CA-bisimilar. Furthermore, for finite-state graph grammars, the corresponding automaton is proved to be finite and thus the general algorithms for causal automata mentioned in Section 4 can be used to check the bisimilarity of graph grammars and to construct a minimal realization.

First, note that, as in the case of Petri nets, the definition of HP-bisimulation on graph grammars relies on the transition system of processes and process moves, which is infinite for any non-trivial system exhibiting a cyclic behaviour. To reduce it to a finite causal automaton, or, in general, to a finite transition system, at least in the case of finite-state systems, the leading idea, already present in [DD90], is that not all the information carried by a process is relevant

for deciding HP-bisimulation. Hence processes may be replaced by more compact structures where part of the past history is discarded.

For ordinary nets, as observed in [Vog91,MP97], one can restrict the attention only to the set of events which produced at least one token in the current state and to the causal ordering among them. In the case of contextual nets one must keep information not only about the events which produced a token in the current state ("producers"), but also about the events which read a token in the current state ("readers"). Fortunately, among the readers, which can be unbounded even for a safe net, only the maximal ones play a significant role, while the others can be safely discarded. This allows to obtain a finite description of the transition system of processes for finite-state contextual nets [BCM00b].

We will show that the construction proposed for contextual nets can be generalized to graph grammars. This can be better understood by recalling that, as already observed, for a deterministic computation of a graph grammar the asymmetric conflicts induced by the possibility of expressing "contextual" rewritings (read operations) play a significant role in the ordering of events, while the inhibiting effects between production occurrences related to the dangling condition can be disregarded since they are subsumed by such asymmetric conflicts.

The next definition formalizes the notions of producer and of (maximal) reader for a process of a graph grammar.

Definition 16 (producers and (maximal) readers). *Given a process φ of a graph grammar $\mathcal{G}$, we define*

- *the set of* producers
  $p(\varphi) = \{q \in P_\varphi : q^\bullet \cap Max(\varphi) \neq \emptyset\}$;
- *the set of* readers
  $r(\varphi) = \{q \in P_\varphi : \underline{q} \cap Max(\varphi) \neq \emptyset\}$;
- *the set of* maximal readers
  $mr(\varphi) = \{q \in r(\varphi) : \exists x \in \underline{q} \cap Max(\varphi).\ q \text{ is } \nearrow_\varphi \text{-maximal in } \underline{x}\}$.

For instance, for process φ_5 of Fig. 4, the set of producers is $p(\varphi_5) = \{e_5\}$, the set of readers is $r(\varphi_5) = \{e_1, e_2, e_3, e_4, e_5\}$, while the maximal readers are $mr(\varphi_5) = \{e_2, e_5\}$.

A crucial observation is that for any *n-safe* graph grammar $\mathcal{G}$ the sets $p(\varphi)$ and $mr(\varphi)$, with φ ranging over the processes of $\mathcal{G}$ are *bounded*. In the sequel, given a graph G we will denote by $|G|$ the cardinality of the (disjoint) union of the node and edge sets of G. More generally, with abuse of notation, a graph will be sometimes identified with the set consisting of the (disjoint) union of its node and edge sets, and we will use on graphs the ordinary set-theoretical relations and operations.

Lemma 17. *Let $\mathcal{G}$ be a n-safe graph grammar. Then, for any process φ of $\mathcal{G}$ we have $|p(\varphi)| \leq n \cdot |TG_\mathcal{G}|$ and $|mr(\varphi)| \leq (n \cdot |TG_\mathcal{G}|)^2$.*

Proof (sketch). By the basic properties of graph processes, for any process φ, the graph $\varphi^\bullet$, namely $Max(\varphi)$ typed over $TG_\mathcal{G}$ by the restriction of φ_T, is reachable in $\mathcal{G}$. Since any graph reachable in $\mathcal{G}$ is n-injective, we can establish the following bound for the number of items (nodes and edges) of $Max(\varphi)$,

$$|Max(\varphi)| \leq n \cdot |TG_{\mathcal{G}}|.$$

Hence it is immediate to conclude that $n \cdot |TG_{\mathcal{G}}|$ is a bound also for the cardinality of $p(\varphi)$ since for each $q, q' \in p(\varphi)$ we have $q^\bullet \cap Max(\varphi) \neq \emptyset$ and $q^\bullet \cap q'^\bullet = \emptyset$.

Furthermore, for any item x in $Max(\varphi)$ the set A_x of maximal events in $\underline{x}$ consists of concurrent events. Hence also the corresponding pre-set ${}^\bullet A_x$ is concurrent and therefore $|{}^\bullet A_x|$ is bounded by $n \cdot |TG_{\mathcal{G}}|$. Since productions are consuming, i.e., they have a non-empty pre-set, and for any $q, q' \in A_x$ it must be ${}^\bullet q \cap {}^\bullet q' = \emptyset$, we conclude that $n \cdot |TG_{\mathcal{G}}|$ is a bound also for the cardinality of A_x. Therefore the number of productions in $mr(\varphi)$ is bounded by $(n \cdot |TG_{\mathcal{G}}|)^2$. □

We next define partial processes, which represent abstractions of graph processes where only a relevant part for discriminating non HP-bisimilar states is kept, namely the target graph of the process (i.e., the subgraph consisting of the maximal items), the producers, the maximal readers and their dependencies. For technical reasons we first introduce pre-partial processes which are required to satisfy weaker requirements.

Definition 18 (pre-partial process). *A* pre-partial process *of a graph grammar $\mathcal{G}$ is a tuple $\gamma = \langle G_\gamma, E_\gamma, \ll_\gamma, \lambda_\gamma, post_\gamma, cont_\gamma \rangle$, where*

- *G_γ is a $TG_{\mathcal{G}}$-typed graph;*
- *E_γ is a set of events;*
- *$\ll_\gamma \subseteq E_\gamma \times E_\gamma$ is a partial order;*
- *$\lambda_\gamma : E_\gamma \to \mathsf{Act}$ is a labelling function over a fixed set of actions Act;*
- *$cont_\gamma, post_\gamma : E_\gamma \to \mathcal{P}(N_{\langle G_\gamma \rangle} \cup E_{\langle G_\gamma \rangle})$ are functions which map each $e \in E_\gamma$ to the sets of items in $\langle G_\gamma \rangle$ which are read and produced, respectively, by e.*

For any $x \in \langle G_\gamma \rangle$ we denote by $cont_\gamma(x)$ the set of readers of x, i.e., the set $\{e \in E_\gamma : x \in cont_\gamma(e)\}$.

An isomorphism of pre-partial processes $i : \gamma_1 \to \gamma_2$ is a pair of functions $i = \langle i_T, i_E \rangle$ where $i_T : G_{\gamma_1} \to G_{\gamma_2}$ is an isomorphism of $TG_{\mathcal{G}}$-typed graphs and $i_E : E_{\gamma_1} \to E_{\gamma_2}$ is a bijection such that i_E establishes an isomorphism of labelled partial orders between $\langle E_{\gamma_1}, \ll_{\gamma_1}, \lambda_{\gamma_1} \rangle$ and $\langle E_{\gamma_2}, \ll_{\gamma_2}, \lambda_{\gamma_2} \rangle$ and, for any $e \in E_{\gamma_1}$, $post(i_E(e)) = i_T(post(e))$ and $cont(i_E(e)) = i_T(cont(e))$.

As for ordinary graph process, for any pre-partial process γ we define the sets of producers and of maximal readers.

Definition 19. *Let γ be a pre-partial process. The set of* producers *of γ is defined as $p(\gamma) = \{e \in E_\gamma : post_\gamma(e) \neq \emptyset\}$. The set of* maximal readers *of γ is defined as $mr(\gamma) = \{e \in E_\gamma : \exists x \in cont_\gamma(e).\ e \text{ is } \ll_\gamma\text{-maximal in } cont_\gamma(x)\}$.*

Partial processes are defined as pre-partial processes where each event is a producer or a maximal reader.

Definition 20 (partial process). *A* partial process *of a graph grammar $\mathcal{G}$ is a pre-partial process γ such that $E_\gamma = p(\gamma) \cup mr(\gamma)$. The* initial partial process *for $\mathcal{G}$ is the partial process over the initial graph, with an empty set of events, i.e., $\gamma_0 = \langle G_{s\mathcal{G}}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$.*

An obvious construction associates to each pre-partial process the corresponding partial process.

Definition 21. *Given any pre-partial process γ, the corresponding partial process, denoted by $Cut(\gamma)$, is defined as follows:*

- $G_{Cut(\gamma)} = G_\gamma$;
- $E_{Cut(\gamma)} = p(\gamma) \cup mr(\gamma)$;

and $\ll_{Cut(\gamma)}$, $\lambda_{Cut(\gamma)}$, $post_{Cut(\gamma)}$ and $cont_{Cut(\gamma)}$ are the restrictions to $E_{Cut(\gamma)}$ of the corresponding relations and functions of γ.

Given any process φ of a graph grammar, we can construct a corresponding partial process by keeping only the producers and the maximal readers of φ. Technically this is done by first constructing a pre-partial process and then using the operation $Cut(\cdot)$.

Definition 22 (partial process associated to a process). *Let φ be a process of a graph grammar $\mathcal{G}$. The corresponding partial process, denoted by $\gamma(\varphi)$, is defined as $Cut(\gamma)$ where γ is the pre-partial process satisfying*

- $G_\gamma = \varphi^\bullet = \langle Max(\varphi), \varphi_{T\,|Max(\varphi)} \rangle$;
- $E_\gamma = P_\varphi$;
- $\ll_\gamma = (\nearrow_\varphi)^*$;
- $\lambda_\gamma = \lambda_\mathcal{G} \circ \varphi_P$;
- *for any $q \in E_\gamma$, $cont_\gamma(q) = \underline{q} \cap Max(\varphi)$ and $post_\gamma(q) = q^\bullet \cap Max(\varphi)$.*

In Fig. 4, for every process φ_i, the corresponding partial process $\gamma(\varphi_i)$ is obtained by considering only the non-shaded part. The next lemma makes explicit the easy fact that the events in the partial process associated to a process φ are exactly the producers and the maximal readers of the original process φ.

Lemma 23. *Let $\mathcal{G}$ be a labelled graph grammar and let φ be a process of $\mathcal{G}$. Then $p(\gamma(\varphi)) = p(\varphi)$ and $mr(\gamma(\varphi)) = mr(\varphi)$.*

Next we introduce a *move relation* on partial processes: given a partial process γ, whenever a production of the original grammar is applicable to the graph G_γ, the partial process can evolve accordingly. This leads to a transition system of partial processes which represents the first step in the construction of the causal automaton associated to a graph grammar.

Definition 24 (partial processes move). *Given two partial processes γ and γ' of a labelled graph grammar $\mathcal{G}$ we write $\gamma \xrightarrow[e_0]{a} \gamma'$, and we say that γ* moves *to γ'* performing the action *a if $\gamma' = Cut(\gamma'')$ where γ'' is a pre-partial process satisfying the following conditions: there is a production $q \in P_\mathcal{G}$*

$$q : L_q \xleftarrow{l_q} K_q \xrightarrow{r_q} R_q$$

and a match $m : L_q \to G_\gamma$ *such that, if* $X = m(\langle L_q \rangle - l_q(\langle K_q \rangle))$ *and* $C = m(l_q(\langle K_q \rangle))$*, then*

- $G_\gamma \Rightarrow_q^a G_{\gamma''}$ *using match* m*; more specifically we assume that* $\langle G_\gamma \rangle - X \subseteq \langle G_{\gamma''} \rangle$*, i.e., the items which are preserved remains concretely the same;*
- $E_{\gamma''} = E_\gamma \cup \{e_0\}$ *and* $e_0 \notin E_\gamma$*;*
- $\ll_{\gamma''} = (\ll_\gamma \cup \{(e, e_0) : e \in E_\gamma \wedge (post(e) \cap (X \cup C)) \cup (cont(e) \cap X) \neq \emptyset\})^*$*;*
- $\lambda_{\gamma''}(e) = \lambda_\gamma(e)$ *for any* $e \in E_\gamma$ *and* $\lambda_{\gamma''}(e_0) = a = \lambda_\mathcal{G}(q)$*;*
- *for any* $e \in E_\gamma$*,* $post_{\gamma''}(e) = post_\gamma(e) - X$*,* $cont_{\gamma''}(e) = cont_\gamma(e) - X$*, and* $cont_{\gamma''}(e_0) = C$*,* $post_{\gamma''}(e_0) = \langle G_{\gamma''} \rangle - \langle G_\gamma \rangle$*.*

As mentioned above, a partial process γ of a grammar $\mathcal{G}$ can perform a move when there exists a production q in $\mathcal{G}$ which is applicable to its graphical component G_γ. The graph $G_{\gamma'}$ underlying the new partial process is obtained by rewriting G_γ using q. Observe that the new event e_0, representing the occurrence of q, depends on the events which have generated a graph item which is consumed or read by q (causality), and also on the events which have read an item consumed by q (asymmetric conflict). The functions *cont* and *post* are extended to the new event e_0, but they must be updated also to take into account the fact that some items of G_γ might have been deleted. Consequently an event might cease to be a producer or a maximal reader and thus, by effect of the application of $Cut(\cdot)$, some events can disappear. A sequence of partial process move is exemplified in Fig. 4, if we consider only the non-shaded parts.

To each process and partial process move we associate the set of maximal (weak or strong) causes of the executed production, which will play a basic role in the definition of the automaton. In fact, to observe the partial order associated to an evolving computation it is sufficient to look, step by step, only at the immediate maximal causes of each single production (the other dependencies being implicitly given by the transitivity of the partial order).

Definition 25 (immediate and maximal causes). *The set of* immediate (weak or strong) causes *of a process move* $\varphi \xrightarrow[e]{a} \varphi'$ *is defined as* $\mathsf{IC}(\varphi \xrightarrow[e]{a} \varphi') = \{q \in P_\varphi : q^\bullet \cap (\underline{e} \cup {}^\bullet e) \neq \emptyset \vee \underline{q} \cap {}^\bullet e \neq \emptyset\}$*. We denote by* $\mathsf{MC}(\varphi \xrightarrow[e]{a} \varphi')$ *the set of* maximal causes*, namely the subset of* $\nearrow_\varphi$*-maximal elements of* $\mathsf{IC}(\varphi \xrightarrow[e]{a} \varphi')$*.*

The set of immediate causes *of a partial process move* $\gamma \xrightarrow[e]{a} \gamma'$*, adopting the notation of Definition 24, is defined by* $\mathsf{IC}(\gamma \xrightarrow[e_0]{a} \gamma') = \{e \in E_\gamma : (post_\gamma(e) \cap (X \cup C)) \cup (cont_\gamma(e) \cap X) \neq \emptyset\}$*. The set of maximal causes* $\mathsf{MC}(\gamma \xrightarrow[e_0]{a} \gamma')$ *is the subset of* $\ll_\gamma$*-maximal immediate causes.*

For example, considering transition $\varphi_5 \xrightarrow[e_6]{q_4} \varphi_6$ in Fig. 4, the immediate causes are $\{e_1, e_2, e_3, e_4, e_5\}$, while the immediate maximal causes are $\{e_2, e_5\}$.

The next lemma relates the transition system of processes and the transition system of partial processes.

Lemma 26. *Let $\mathcal{G}$ be any labelled graph grammar.*

1. *If φ and φ' are processes of $\mathcal{G}$ and $\varphi \xrightarrow[e]{a} \varphi'$ then we have $\gamma(\varphi) \xrightarrow[e]{a} \gamma(\varphi')$, with $\mathsf{MC}(\varphi \xrightarrow[e]{a} \varphi') = \mathsf{MC}(\gamma(\varphi) \xrightarrow[e]{a} \gamma(\varphi'))$;*
2. *If φ is a process of $\mathcal{G}$ and $\gamma(\varphi) \xrightarrow[e]{a} \gamma'$ then there exists a process φ' of $\mathcal{G}$, such that $\varphi \xrightarrow[e']{a} \varphi'$, with γ' and $\gamma(\varphi')$ isomorphic and $\mathsf{MC}(\gamma(\varphi) \xrightarrow[e]{a} \gamma') = \mathsf{MC}(\varphi \xrightarrow[e']{a} \varphi')$.*

It is worth noting that we cannot replace point (2) above with the stronger "if $\gamma(\varphi) \xrightarrow[e]{a} \gamma'$ then there exists a process φ' of $\mathcal{G}$, such that $\varphi \xrightarrow[e]{a} \varphi'$, with $\gamma' = \gamma(\varphi')$", since in general the event e and the new graph items in $G_{\gamma'}$ can appear in φ.

By Lemma 26 we conclude that if a partial process γ of a graph grammar $\mathcal{G}$ is reachable from an initial partial process via a finite sequence of moves, then $\gamma = \gamma(\varphi)$ for some process φ of $\mathcal{G}$. Hence, when the graph grammar $\mathcal{G}$ is n-safe, the definition of $\gamma(\varphi)$ and Lemmata 17 and 23 allow us to conclude the validity of the following result.

Lemma 27. *For any n-safe labelled graph grammar the set of partial processes reachable from the initial process (and taken up to isomorphism) is finite.*

We are now ready to present the construction of the causal automaton associated to a graph grammar for checking HP-bisimilarity. To obtain a "compact" automaton (with a finite number of states for n-safe graph grammar) we must consider partial processes up to isomorphism. To this aim we fix a standard representative in each class of isomorphic partial processes. Furthermore we consider a normalization function $\mathtt{norm}$ such that for any partial process γ, $\mathtt{norm}(\gamma) = \langle \gamma', i \rangle$, where γ' is the standard representative in the isomorphism class of γ and $i : \gamma' \to \gamma$ is a chosen partial process isomorphism. We assume that the names of the productions in any (partial) process γ are taken from $\mathcal{N}$, namely that $E_\gamma \subseteq \mathcal{N}$.

Definition 28 (causal automaton for HP-bisimulation). *Let $\mathcal{G}$ be a labelled graph grammar. The* HP-causal automaton *associated to $\mathcal{G}$ is the automaton $\mathcal{A}_{hp}(\mathcal{G}) = \langle Q, n, \longmapsto, q_0 \rangle$, having (standard representatives of) partial processes as states. The initial state q_0 is the standard representative γ_0 of the initial partial processes of $\mathcal{G}$ and whenever $\gamma \in Q$ then*

- $n(\gamma) = E_\gamma$;
- *if $\gamma \xrightarrow[e]{a} \gamma'$ and $\mathtt{norm}(\gamma') = \langle \gamma'', i \rangle$ then $\gamma'' \in Q$ and $\gamma \xmapsto[M]{a}_{\sigma} \gamma''$ where*
  - *$\sigma : E_{\gamma''} \hookrightarrow E_\gamma \cup \{\star\}$ is defined as $\sigma = (id_{E_\gamma} \cup \{(e, \star)\}) \circ i_E$;*
  - *$M = \mathsf{MC}(\gamma \xrightarrow[e]{a} \gamma')$.*

Observe that the renaming function in a transition of the causal automaton is obtained from the isomorphism given by the normalization function `norm`, simply by redirecting the new name e to $\star$ (if e belongs to $E_{\gamma'}$). As anticipated, the maximal causes of a process move are used as dependencies in the automaton transition.

The states of the automaton are standard representatives of partial processes reachable from the initial partial process. Hence by Lemma 27 we deduce that for any n-safe graph grammar the above defined automaton has a finite number of states (and also a finite number of transitions leaving from each state, since the number of productions is finite). Vice versa, if the graph grammar is not n-safe for some n, then the automaton will have an infinite number of states.

Theorem 29. *Let $\mathcal{G}$ be a labelled graph grammar. Then $\mathcal{G}$ is n-safe for some n iff the automaton $\mathcal{A}_{hp}(\mathcal{G})$ is finite.*

To effectively build the automaton we can perform an inductive construction based on Definition 28. The only thing to observe is that, given a partial process γ, there might be infinitely many moves $\gamma \xrightarrow[e]{a} \gamma'$ since the event e can be chosen arbitrarily among the unused events in $\mathcal{N}$ and a similar consideration holds for the new graph items in $G_{\gamma'}$. However, without loss of generality, we can limit our attention only to some partial process moves, called the *representative* moves, where the newly generated name and items are chosen in a canonical way. For instance we can suppose that the set of names $\mathcal{N}$ is well-ordered and assume that a transition $\gamma \xrightarrow[e]{a} \gamma'$ to be representative must satisfy $e = \min(\mathcal{N} - P_\gamma)$.

The main result now states that there is a precise correspondence between HP-bisimulation on graph grammars and CA-bisimulation on causal automata. Hence HP-bisimilarity of graph grammars can be checked on the corresponding automata.

Theorem 30. *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two labelled graph grammars. Then $\mathcal{G}_1 \sim_{hp} \mathcal{G}_2$ if and only if $\mathcal{A}_{hp}(\mathcal{G}_1) \sim_{ca} \mathcal{A}_{hp}(\mathcal{G}_2)$.*

Proof (sketch). The proof is organized in two steps. First observe that the transition system of processes of a graph grammar $\mathcal{G}$ can be seen itself as a causal automaton $\mathcal{A}_{pr}(\mathcal{G}) = \langle Q, n, \longmapsto, q_0 \rangle$, where

- Q is the set of processes φ of $\mathcal{G}$ and $n(\varphi) = P_\varphi$ for any process φ;
- $\varphi \xmapsto[M]{a}_\sigma \varphi'$ if, according to Definition 9, $\varphi \xrightarrow[e]{a} \varphi'$, $M = \mathsf{MC}(\varphi \xrightarrow[e]{a} \varphi')$, and the naming $\sigma : P_{\varphi'} \to P_\varphi \cup \{\star\}$ is defined as the identity for $x \in P_{\varphi'} - \{e\}$, while $\sigma(e) = \star$;
- the initial state q_0 is any initial process of $\mathcal{G}$.

Then, it is possible to prove that HP-bisimulation on graph grammars coincides with CA-bisimulation on the causal automata of processes, namely $\mathcal{G}_1 \sim_{hp} \mathcal{G}_2$ iff $\mathcal{A}_{pr}(\mathcal{G}_1) \sim_{ca} \mathcal{A}_{pr}(\mathcal{G}_2)$.

The second step of the proof shows that, for any graph grammar $\mathcal{G}$ there exists an abstraction homomorphism $\mathsf{h} : \mathcal{A}_{pr}(\mathcal{G}) \to \mathcal{A}_{hp}(\mathcal{G})$, and thus, by Lemma 15,

$\mathcal{A}_{pr}(\mathcal{G}) \sim_{ca} \mathcal{A}_{hp}(\mathcal{G})$. The abstraction homomorphism $\mathsf{h} = \langle h, \{h_\varphi\}_\varphi \rangle$ can be defined as follows: for any process φ (state of $\mathcal{A}_{pr}(\mathcal{G})$), if $\mathtt{norm}(\gamma(\varphi)) = \langle \gamma', i \rangle$ then $h(\varphi) = \gamma'$ and $h_\varphi : E_{\gamma'} \to P_\varphi$ is simply i_E. To prove that h satisfies the conditions in Definition 14 one essentially resorts to Lemma 26.

Summing up, by the above considerations we have that $\mathcal{A}_{pr}(\mathcal{G}_i) \sim_{ca} \mathcal{A}_{hp}(\mathcal{G}_i)$ for $i \in \{1, 2\}$, and moreover $\mathcal{A}_{pr}(\mathcal{G}_1) \sim_{ca} \mathcal{A}_{pr}(\mathcal{G}_2)$ iff $\mathcal{G}_1 \sim_{hp} \mathcal{G}_2$. Hence the thesis easily follows. □

By Theorems 29 and 30 we immediately conclude the desired decidability result.

Corollary 31. *HP-bisimulation on n-safe graph grammars is decidable.*

It is worth observing that, in this setting, due to the Turing completeness of graph grammars, differently from what happens for ordinary and contextual nets, the property of being n-safe for some n, i.e., the property of being finite-state, is not decidable.

6 Conclusions

In this paper we have introduced an abstract semantics for graph grammars inspired by the classical history preserving bisimulation. Extending the work already developed on ordinary and contextual P/T nets, we have shown how history preserving bisimulation on graph grammars can be studied in the general framework of causal automata. A translation of graph grammars into causal automata has been proposed, which respects (preserves and reflects) history preserving bisimulation. The translation produces finite automata for finite-state graph grammars, thus allowing to reuse the algorithms existing for this general formalism in order to decide bisimulation and to obtain a minimal realization.

We conclude by discussing two possible directions of further investigation which we find interesting: on the one hand the possibility of defining different notions of history preserving bisimulation by considering observations of the causal history of a computation finer than the associated PES; on the other hand the development of a logic in the style of Hennessy-Milner for HP-bisimulation.

6.1 Refining the observation

The notion of HP-bisimulation considered in this paper is obtained by taking as observation of a concurrent computation of a graph grammar the PES underlying the corresponding graph process. We have already mentioned that this corresponds to observe only the precedences between events, confusing the weak causality deriving from the possibility of preserving part of the state in a rewriting step, the inhibiting effects related to the dangling condition and the "strong" causality deriving from the flow of information. It could be reasonable to consider, instead, equivalences which arise by assuming different, finer descriptions of concurrent computations.

For instance, a natural refinement consists of distinguishing the flow of information from the other dependencies. This is easily achieved by extracting from a process a different event structure, which is called *asymmetric event structure* [BCM98b,BCM00a] where causality and asymmetric conflict are kept separate. The asymmetric event structure associated to a graph process φ is defined as

$$aev(\varphi) = \langle P_\varphi, \leq_\varphi, \nearrow_\varphi, \lambda_\mathcal{G} \circ \varphi_P \rangle.$$

Then the corresponding bisimulation, which can be called *read history preserving (RHP-) bisimulation*, is defined as HP-bisimulation, by simply refining the observation, namely by changing $ev(\varphi_i)$ with $aev(\varphi_i)$ in Definition 11.

Any RHP-bisimulation relating two graph grammar $\mathcal{G}_1$ and $\mathcal{G}_2$ is also an HP-bisimulation. In fact if φ_1 and φ_2 are processes of $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively, and $f : aev(\varphi_1) \to aev(\varphi_2)$ is an isomorphism of asymmetric event structures then it is easy to see that f is also an isomorphism of PES's between $ev(\varphi_1)$ and $ev(\varphi_2)$. Therefore $\mathcal{G}_1 \sim_{rhp} \mathcal{G}_2$ implies $\mathcal{G}_1 \sim_{hp} \mathcal{G}_2$. As for contextual nets, the converse implication, instead, does not hold. Regarding the decidability of RHP-bisimulation, the natural extension of the construction which has been introduced for HP-bisimulation consists of considering partial processes where all the readers (not only the maximal ones) are kept. Unfortunately in this way the construction produces a causal automaton which may be infinite also for safe graph grammars. Indeed, the decidability of RHP-bisimulation is an open question already for contextual nets [BCM00b].

6.2 Hennessy-Milner logic for HP-bisimulation

The ordinary bisimulation over transition systems has a logical counterpart, the so-called Hennessy-Milner logic [HM85], a kind of modal logic with two basic modalities which can be interpreted as *possibility* and *necessity*. The syntax of formulae is the following

$$\phi ::= \mathsf{true} \mid \phi \wedge \phi \mid \neg\phi \mid \langle a \rangle \phi.$$

The formula constructed with the "*diamond*" modality $\langle a \rangle \phi$, where a is an action and ϕ a formula, intuitively is satisfied by any state from which an a-action can be executed leading to a state which satisfies ϕ. The dual modality, i.e., the "*box*" modality $[a]\phi$, can be defined as $\neg\langle a \rangle \neg\phi$. It is satisfied by all the states where any a-action leads to a state that satisfies ϕ. Hennessy-Milner logic can be shown to be *adequate* for bisimulation in the sense that, two states of a transition system are bisimilar if and only if they satisfy the same set of formulae [HM85].

An interesting direction of further research is the study of an analogue of Hennessy-Milner logic for HP-bisimulation, which has been initiated in [Bar99]. The basic syntax of formulae is the following

$$\phi ::= \mathsf{true} \mid \phi \wedge \phi \mid \neg\phi \mid \mathsf{EX}\{n, a, M\}\phi.$$

The *existential* modality allows to construct a formula $\mathsf{EX}\{e, a, M\}\phi$ which, intuitively, is satisfied by a state where an action a can be executed, which generates a new name (or event) e directly caused by the set of events in M, leading to a state which satisfies ϕ. Also in this case there is a dual *universal* modality: the formula $\mathsf{AX}\{n, a, M\}\phi$, defined as $\neg\mathsf{EX}\{e, a, M\}\neg\phi$, is satisfied by a state where any action a which can be executed, generates a new name (or event) e directly caused by the set of events in M, leading to a state which satisfies ϕ.

Like ordinary Hennessy-Milner logic is naturally interpreted over transition systems (labelled graphs), this variation of the logic has a natural interpretation over causal automata, but also over the transition system of processes of a net or of a graph grammar. The possibility of declaring new names/events in a formula is reflected, at semantical level, by the the presence in the model of a kind of environment which links the events in the current state and the names "declared" in the formula. An *adequateness* result for such a logic over causal automata has been proved in [Bar99] showing that two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are CA-bisimilar iff they satisfy the same set of formulae. Resorting to our results, adequateness for the logic over graph grammars would be easily proved by showing that for any labelled graph grammar $\mathcal{G}$

$$\mathcal{G} \Vdash \phi \quad \Leftrightarrow \quad \mathcal{A}_{pr}(\mathcal{G}) \Vdash \phi.$$

where "$\Vdash$" means "is a model of".

As in the case of ordinary Hennessy-Milner logic, the expressiveness would greatly benefit form the introduction of some "recursion" operator, e.g., minimal/maximal fix-point operators in the style of the μ-calculus (ν-calculus). This should be done by retaining some interesting properties of the logic, like decidability, at least for a significant fragment.

Acknowledgements. We are grateful to the anonymous referees for their insightful comments and suggestions.

References

[Bal00] P. Baldan. *Modelling concurrent computations: from contextual Petri nets to graph grammars*. PhD thesis, Department of Computer Science, University of Pisa, 2000. Available as technical report n. TD-1/00.

[Bar99] R. Bartolini. Model checking di proprietà causali di reti di Petri. MSc Thesis, University of Pisa, 1999. (In Italian).

[BCM98a] 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.

[BCM98b] P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. In M. Nivat, editor, *Proceedings of FoSSaCS '98*, volume 1378 of *LNCS*, pages 63–80. Springer Verlag, 1998.

[BCM99] 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.

[BCM00a] P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures and processes. To appear in *Information and Computation.*, 2000.

[BCM00b] P. Baldan, A. Corradini, and U. Montanari. History preserving bisimulations for contextual nets. In D. Bert and C. Choppy, editors, *WADT'99 Conference Proceedings*, number 1827 in LNCS, pages 291–310. Springer Verlag, 2000.

[BDKP91] E. Best, R. Devillers, A. Kiehn, and L. Pomello. Concurrent bisimulations in Petri nets. *Acta Informatica*, 28(3):231–264, 1991.

[Cas87] I. Castellani. Bisimulations and abstraction homomorphisms. *Journal of Computer and System Sciences*, 34(2/3):210–235, 1987.

[CFM83] I. Castellani, P. Franceschi, and U. Montanari. Labeled event structures: a model for observable concurrency. In D. Bjørner, editor, *Proceedings of IFIP TC2 Working Conference on Formal Description of Programming Concepts – II*, pages 383–389. North-Holland, 1983.

[CMR96] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.

[CMR+97] 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 G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations.* World Scientific, 1997.

[Cor96] A. Corradini. Concurrent graph and term graph rewriting. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*, pages 438–464. Springer Verlag, 1996.

[DD90] P. Darondeau and P Degano. Causal trees: Interleaving + causality. In *Proc. 18th École de Printemps sur la Semantique de Parallelism*, number 469 in LNCS, pages 239–255. Springer Verlag, 1990.

[Ehr87] H. Ehrig. Tutorial introduction to the algebraic approach of graph-grammars. In H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors, *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, volume 291 of *LNCS*, pages 3–14. Springer Verlag, 1987.

[EKMR99] H. Ehrig, J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Concurrency, Parallelism and Distribution.* World Scientific, 1999.

[FM90] G. Ferrari and U. Montanari. Towards the unification of models of concurrency. In A. Arnold, editor, *Proceedings of CAAP '90*, volume 431 of *LNCS*, pages 162–176. Springer-Verlag, 1990.

[GHK00] F. Gadducci, R. Heckel, and M. Koch. A fully abstract model for graph-interpreted temporal logic. In H. Ehrig, G. Engels, H.J. Kreowski, and G. Rozenberg, editors, *Proceedings of TAGT'98*, volume 1764 of *LNCS*, pages 310–322. Springer Verlag, 2000.

[GM82] 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.

[Hec98] R. Heckel. *Open Graph Transformation Systems: A New Approach to the Compositional Modelling of Concurrent and Reactive Systems.* PhD thesis, TU Berlin, 1998.

[HEWC97] R. Heckel, H. Ehrig, U. Wolter, and A. Corradini. Integrating the Specification Techniques of Graph Transformation and Temporal Logic. In *Proceedings of MFCS'97*, number 1295 in LNCS. Springer Verlag, 1997.

[HM85] M. Hennessy and R. Milner. Algebraic laws for indeterminism and concurrency. *Journal of the ACM*, 32:137–162, 1985.

[JK95] R. Janicki and M. Koutny. Semantics of inhibitor nets. *Information and Computation*, 123:1–16, 1995.

[JNW96] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

[Koc99] M. Koch. *Integration of graph transformation and temporal logic for the specification of distributed systems.* PhD thesis, TU Berlin, 1999.

[Kre77] H.-J. Kreowski. *Manipulation von Graphmanipulationen.* PhD thesis, Technische Universität Berlin, 1977.

[MP97] U. Montanari and M. Pistore. Minimal transition systems for history-preserving bisimulation. In *14th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *LNCS*, pages 413–425. Springer Verlag, 1997.

[MP98] U. Montanari and M. Pistore. History-dependent automata. Technical Report TR-98-11, Dipartimento di Informatica, 1998. Available as ftp://ftp.di.unipi.it/pub/techreports/TR-98-11.ps.Z.

[MP00] U. Montanari and M. Pistore. Structured coalgebras and minimal HD-automata. In M. Nielsen and B. Roman, editors, *Proc. of MFCS 2000*, volume 1983 of *LNCS*, pages 569–578. Springer Verlag, 2000.

[MR95] U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32(6), 1995.

[Pis99] M. Pistore. *History Dependent Automata.* PhD thesis, Department of Computer Science, University of Pisa, 1999.

[RGG96] E. Rudolph, J. Grabowski, and P. Graubmann. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996.

[Rib96] L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars.* PhD thesis, Technische Universität Berlin, 1996.

[Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations.* World Scientific, 1997.

[RT88] A. Rabinovich and B. A. Trakhtenbrot. Behavior Structures and Nets. *Fundamenta Informaticæ*, 11(4):357–404, 1988.

[vB84] J. van Bentham. Correspondence theory. In *Handbook of Philosophical Logic*, volume II. Reidel, 1984.

[Vog91] W. Vogler. Deciding history preserving bisimilarity. In J. Leach Albert, B. Monien, and M. Rodríguez-Artalejo, editors, *Proceedings of ICALP'91*, volume 510 of *LNCS*, pages 495–505. Springer-Verlag, 1991.

[Vog97] W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In *Proceedings of ICALP'97*, volume 1256 of *LNCS*, pages 538–548. Springer Verlag, 1997.

[VSY98] W. Vogler, A. Semenov, and A. Yakovlev. Unfolding and finite prefix for nets with read arcs. In *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 501–516. Springer-Verlag, 1998.