Equivalence and Independence in Controlled Graph-Rewriting Processes

Géza Kulcsár[1] *[0000−0002−5387−8277], Andrea Corradini[2][0000−0001−6123−4175], and Malte Lochau[1][0000−0002−8404−753X]

[1] Real-Time Systems Lab, TU Darmstadt, Germany
{geza.kulcsar,malte.lochau}@es.tu-darmstadt.de
[2] Dipartimento di Informatica, University of Pisa, Italy
andrea@di.unipi.it

Abstract. Graph transformation systems (GTS) are often defined as sets of rules that can be applied repeatedly and non-deterministically to model the evolution of a system. Several semantics proposed for GTSs are relevant in this case, providing means for analysing the system's behaviour in terms of dependencies, conflicts and potential parallelism among the relevant events. Several other approaches equip GTSs with an additional control layer useful for specifying rule application strategies, for example to describe graph manipulation algorithms. Almost invariably, the latter approaches consider only an input-output semantics, for which the above mentioned semantics are irrelevant.
We propose an original approach to controlled graph transformation, where we aim at bridging the gap between these two complementary classes of approaches. The control is represented by terms of a simple process calculus. Expressiveness is addressed by encoding in the calculus the Graph Processes defined by Habel and Plump, and some initial results are presented relating parallel independence with process algebraic notions like bisimilarity.

1 Introduction

Graph-rewriting systems are used for many different purposes and in various application domains. They provide an expressive and theoretically well founded basis for the specification and the analysis of concurrent and distributed systems [7]. Typically, a set of graph-rewriting rules describes the potential changes of graph-based, abstract representations of the states of a system under consideration. Each rule can be applied when a certain pattern occurs in the state, producing a local change to it. Thus, graph-rewriting systems are inherently non-deterministic regarding both the rule sequencing and the selection of the match, i.e. the pattern to rewrite. Several semantics have been proposed for graph transformation systems (GTS), which emphasize the parallelism that naturally arises

* This work has been funded by the German Research Foundation (DFG) as part of project A1 within CRC 1053–MAKI.

between rules that are applied to independent parts of the distributed state. They include, among others, the *trace-based*, the *event structure* and the *process semantics* summarized and compared in [2]. The common intuition is that the semantic domain should be rich enough to describe the computations of a system (i.e., not only the reachable states), but also abstract enough to avoid distinguishing computations that differ for irrelevant details only, or for the order in which independent rule applications are performed.

Sometimes however, mainly in the design of graph manipulation algorithms, a finer control on the order of application of graph rules is desirable, for example including sequential, conditional, iterative or even concurrent composition operators. To address this problem, several approaches to *programmed graph grammars* or *controlled graph rewriting* have been proposed, which generalize the *controlled string grammars* originally introduced with the goal of augmenting language generation with constraints on the order of application of productions [6]. One of the first approaches to programmed graph grammars is due to Bunke [3]. Regarding the semantics of controlled graph rewriting, Schürr proposes a semantic domain including possible input/output graph pairs [14], which has been the basis for the development of several tools (such as PROGRES [15], Fujaba (SDM) [9] and eMoflon [12]). Habel and Plump [10] propose a minimal language for controlled graph rewriting (see Section 4) with the goal of showing its computational completeness, for which an input/output semantics is sufficient. Also Plump and Steinert propose an input/output semantics, presented in an operational style, for the controlled graph rewriting language *GP* [13].

In this paper, we propose an original approach to controlled graph rewriting, where control is specified with terms of a simple process calculus able to express non-deterministic choice, parallel composition, and prefixing with non-applicability conditions, thus constraining in a strict but not necessarily sequential way the order of application of rules of a given system. In the present paper we introduce the relevant definitions and start exploring the potentialities of the approach, while the long-term goal is to equip such systems with an abstract truly-concurrent semantics, suitable as foundation of efficient analysis techniques (like for example [1] for contextual Petri nets). We start by introducing process terms which specify a labeled transition system (LTS) where transitions represent potential applications of possibly parallel rules. This corresponds conceptually to the code of an algorithm, or to an unmarked net in the theory of Petri nets, an analogy that we adopt by calling those processes *unmarked*. Next, we define the operational semantics of such process terms when applied to a graph, yielding the *marked* LTS where transitions become concrete rule applications. The LTS semantics involves explicit handling of parallel independence (i.e., arbitrary sequentialization) of rule applications, if they are fired by controlled processes running in parallel: our notion of *synchronization* allows for single shared transitions of parallel processes whenever there are parallel independent rule applications of the involved processes. Particularly, synchronized actions represent a first step towards adequately capturing true concurrency of independent rule applications in a controlled setting.

From an expressiveness perspective, in order to enrich controlled graph-rewriting processes by conditional branching constructs as necessary in any control mechanism, we introduce *non-applicability conditions* formulated over rules, corresponding to the condition that a given rule is not applicable. We also prove, as a sanity check, that our control language including non-applicability conditions is able to encode in a precise way the language proposed by Habel and Plump in [10]: an input/output semantics is sufficient to this aim.

The LTS framework is exploited next to start exploring other potentialities of the approach. We introduce an abstract version of the marked LTS showing that it is finite branching under mild assumptions, and define trace equivalence and bisimilarity among marked processes. Besides some pretty obvious results concerning such equivalence, we show that in a simple situation bisimilarity can be used to check that two derivations are not parallel independent: a link between the classical theories of GTSs and of LTSs that we intend to explore further.

2 Preliminaries

We introduce here the basic definitions related to (typed) graphs, algebraic Double-Pushout (DPO) rewriting, parallel derivations and shift equivalence [7].

Definition 1 (Graphs and Typed Graphs)**.** *A* (directed) graph *is a tuple $G = \langle N, E, s, t\rangle$, where N and E are finite sets of nodes and edges, and $s, t : E \to N$ are the source and target functions. The components of a graph G are often denoted by N_G, E_G, s_G, t_G. A* graph morphism *$f : G \to H$ is a pair of functions $f = \langle f_N : N_G \to N_H, f_E : E_G \to E_H\rangle$ such that $f_N \circ s_G = s_H \circ f_E$ and $f_N \circ t_G = t_H \circ f_E$; it is an isomorphism if both f_N and f_E are bijections.*

Graphs G and H are isomorphic, denoted $G \cong H$, if there is an isomorphism $f : G \to H$. We denote by $[G]$ the class of all graphs isomorphic to G, and we call it an abstract graph*. We denote by* **Graph** *the category of graphs and graph morphisms, by* $|\mathbf{Graph}|$ *the set of its objects, that is all graphs, and by* $[|\mathbf{Graph}|]$ *the set of all abstract graphs.*

The category of typed graphs *over a* type graph *T is the slice category $(\mathbf{Graph} \downarrow T)$, also denoted $\mathbf{Graph}_T$ [4]. That is, objects of $\mathbf{Graph}_T$ are pairs (G, t) where $t : G \to T$ is a* typing *morphism, and an arrow $f : (G, t) \to (G', t')$ is a morphism $f : G \to G'$ such that $t' \circ f = t$.*

Along the paper we will mostly work with typed graphs, thus when clear from the context we omit the word "typed" and the typing morphisms.

Definition 2 (Graph Transformation System)**.** *A* (DPO T-typed graph) rule *is a span $(L \xleftarrow{l} K \xrightarrow{r} R)$ in $\mathbf{Graph}_T$ where l is mono. The graphs L, K, and R are called the* left-hand side*, the* interface*, and the* right-hand side *of the rule, respectively. A* graph transformation system (GTS) *is a tuple $\mathcal{G} = \langle T, \mathcal{R}, \pi\rangle$, where T is a type graph, $\mathcal{R}$ is a finite set of* rule names*, and π maps each rule name in $\mathcal{R}$ into a rule.*

The categorical framework allows to define easily the parallel composition of rules, by taking the coproduct of the corresponding spans.

Definition 3 (Parallel Rules)**.** *Given a* GTS $\mathcal{G} = \langle T, \mathcal{R}, \pi \rangle$*, the set of* parallel rule names $\mathcal{R}^*$ *is the free commutative monoid generated by* $\mathcal{R}$*,* $\mathcal{R}^* = \{p_1 | \dots | p_n \mid n \geq 0, p_i \in \mathcal{R}\}$*, with monoidal operation "*$|$*" and unit* ε*. We use* ρ *to range over* $\mathcal{R}^*$*. Each element of* $\mathcal{R}^*$ *is associated with a span in* $\mathbf{Graph}_T$*, up to isomorphism, as follows:*

1. ε: $(\emptyset \leftarrow \emptyset \rightarrow \emptyset)$*, where* $\emptyset$ *is the empty graph;*
2. p: $(L \xleftarrow{l} K \xrightarrow{r} R)$ *if* $p \in \mathcal{R}$ *and* $\pi(p) = (L \xleftarrow{l} K \xrightarrow{r} R)$*;*
3. $\rho_1 | \rho_2$: $(L_1 + L_2 \xleftarrow{l_1 + l_2} K_1 + K_2 \xrightarrow{r_1 + r_2} R_1 + R_2)$ *if* ρ_1: $(L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1)$ *and* ρ_2: $(L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2)$*, where* $G + H$ *denotes the* coproduct *(disjoint union) of graphs* G *and* H*, and if* $g : G \rightarrow G'$ *and* $h : H \rightarrow H'$ *are morphisms, then* $g + h : G + H \rightarrow G' + H'$ *denotes the obvious mediating morphism.*

For $\rho \in \mathcal{R}^*$*, we denote by* $\langle \rho \rangle$ *the set of rule names appearing in* ρ*, defined inductively as* $\langle \varepsilon \rangle = \emptyset$*,* $\langle p \rangle = \{p\}$ *if* $p \in \mathcal{R}$*, and* $\langle \rho_1 | \rho_2 \rangle = \langle \rho_1 \rangle \cup \langle \rho_2 \rangle$*.*

Note that the above definition is well-given because coproducts are associative and commutative up to isomorphism. Clearly, the same rule name can appear several times in a parallel rule name. In the following, we assume that $\mathcal{G} = \langle T, \mathcal{R}, \pi \rangle$ denotes an arbitrary but fixed GTS.

Definition 4 (Rule Application, Derivations)**.** *Let* G *be a graph, let* $\rho : (L \xleftarrow{l} K \xrightarrow{r} R)$ *be a possibly parallel rule, and let* m *be a* match*, i.e., a (possibly non-injective) graph morphism* $m : L \rightarrow G$*. A* DPO rule application *from* G *to* H *via* ρ *(based on* m*) is a diagram* δ *as in (1), where both squares are pushouts in* $\mathbf{Graph}_T$*. In this case we write* $G \overset{\delta}{\Rightarrow} H$ *or simply* $G \overset{\rho @ m}{\Longrightarrow} H$*. We denote by* $\mathcal{D}$ *the set of* DPO *diagrams, ranged over by* δ*. For a rule* $p \in \mathcal{R}$ *and a graph* G*, we write* $G \overset{p}{\not\Rightarrow}$ *if there is no match* m *such that* $G \overset{p @ m}{\Longrightarrow} H$ *for some graph* H*.*

A (parallel) derivation φ *from a graph* G_0 *is a finite sequence of rule applications* $\varphi = G_0 \overset{\delta_1}{\Rightarrow} G_1 \cdots G_{n-1} \overset{\delta_n}{\Rightarrow} G_n$*, via* $\rho_1, \dots, \rho_n \in \mathcal{R}^*$*. A derivation is* linear *if* $\rho_1, \dots, \rho_n \in \mathcal{R}$*.*

$$\begin{tikzcd} L \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dr, phantom, "(PO)"] & R \arrow[d, "n"] \\ G & D \arrow[l, "f"] \arrow[r, "g"'] & H \end{tikzcd} \quad (1)$$

Intuitively, two rule applications starting from the same graph are *parallel independent* if they can be sequentialized arbitrarily with isomorphic results. This property is captured categorically by the following definition [5].

Definition 5 (Parallel Independence). *Given two (possibly parallel) rules $\rho_1 : (L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1)$ and $\rho_2 : (L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2)$ and two matches $L_1 \xrightarrow{m_1} G \xleftarrow{m_2} L_2$ in a graph G, the rule applications $\rho_1@m_1$ and $\rho_2@m_2$ are* parallel independent *if there exist arrows $a_1 : L_1L_2 \to K_1$ and $a_2 : L_1L_2 \to K_2$ such that $l_1 \circ a_1 = \pi_1$ and $l_2 \circ a_2 = \pi_2$ as in Diagram (2), where L_1L_2 is the pullback object over $L_1 \xrightarrow{m_1} G \xleftarrow{m_2} L_2$.*

$$\begin{tikzcd}
& L_1 \arrow[dl, "m_1"'] & & K_1 \arrow[ll, "l_1"'] \\
G & (PB) & L_1L_2 \arrow[ul, "\pi_1"'] \arrow[ur, "a_1"'] \arrow[dl, "\pi_2"] \arrow[dr, "a_2"] & \\
& L_2 \arrow[ul, "m_2"] & & K_2 \arrow[ll, "l_2"]
\end{tikzcd} \tag{2}$$

As discussed in [5], this definition is equivalent to others proposed in literature, but does not need to compute the pushout complements to be checked.

As recalled by the next result, two parallel independent rule applications can be applied in any order to a graph G obtaining the same resulting graph, up to isomorphism. Furthermore, the same graph can be obtained by applying to G the parallel composition of the two rules, at a match uniquely determined by the coproduct construction.

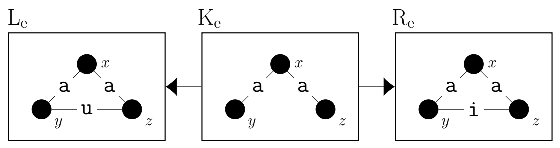
Proposition 1 (Local Church-Rosser and Parallelism Theorems [7]). *Given two rule applications $H_1 \xLeftarrow{\rho_1@m_1} G \xRightarrow{\rho_2@m_2} H_2$ with parallel independent matches $m_1 : L_1 \to G$ and $m_2 : L_2 \to G$, there exist matches $m'_1 : L_1 \to H_2$, $m'_2 : L_2 \to H_1$ and $m : L_1 + L_2 \to G$ such that there are rule applications $H_1 \xRightarrow{\rho_2,m'_2} H_{12}$, $H_2 \xRightarrow{\rho_1,m'_1} H_{21}$ and $G \xRightarrow{\rho_1|\rho_2,m} H$, and graphs H_{12}, H_{21} and H are pairwise isomorphic.*

3 Controlled Graph-Rewriting Processes

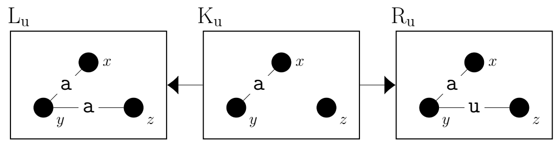
In this section, we first motivate diverse aspects of our approach by presenting a (simplified) application of controlled graph-rewriting processes (Sec. 3.1). Then, in Sec. 3.2, we start to develop the theory by first introducing *unmarked* processes, representing a process-algebraic control-flow specification, i.e., a process whose executions specify permitted derivations. Afterwards (Sec. 3.3), we introduce *marked processes* as pairs of unmarked processes and graphs, and compare marked traces to parallel derivations [2].

3.1 An Illustrative Example: WSN Topology Control

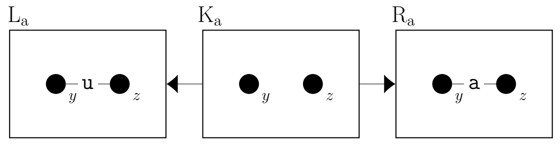
We illustrate controlled graph-rewriting processes by an example: a simplified *wireless sensor network* (WSN) model in which autonomous sensors communicate through wireless channels, represented as typed graphs where nodes denote sensors and edges denote bidirectional communication links via channels. We use different edge types to represent the link status: *active* (`a`) indicates that the link is currently used for communication, whereas link status *inactive* (`i`) denotes links currently not in use. Links with status *unclassified* (`u`) require status revision.



(a) Rule p_e: Eliminate Active Triangle



(b) Rule p_u: Unclassify Active Neighbor



(c) Rule p_a: Activate Edge

Fig. 1: Topology Control Operations as DPO Rules

The DPO rules shown in Figures 1a-1c represent *topology control* (TC) operations [11]: p_e and p_u reduce link redundancy either conservatively by eliminating u-edges from active triangles (p_e), or through unclassifying edges with active neighbors (p_u), whereas p_a is a stability counter-measure, activating unclassified edges. We use the rules in Fig. 1 to specify *controlled graph-rewriting processes* expressing different topology control strategies. Due to the decentralized nature of WSN, both *sequential rule control with non-applicability conditions* and *parallel processes* are inherent in topology control strategies. As a concrete example for a TC strategy, let us consider (using a yet informal process-algebraic notation)

$$P_{TC} := P_e \,||\, P_u \qquad P_e := p_e.P_e + (p_a, \{p_e\}).P_e \qquad P_u := p_u.P_u$$

Here, each P (with a subscript) is a process name that can appear in other processes, allowing to express recursion. The dot (".") operator represents prefixing, while "+" represents non-deterministic choice and " || " parallel composition. Actions can be either plain rule applications (like p_e in P_e) or rule applications with additional *non-applicability conditions* (as in $(p_a, \{p_e\})$). The second component of the action is a set of rule names (here, containing only p_e), denoting that p_a should be applied *only if* p_e *is not applicable.* (Here, as also later in the paper, we omit the second component of an action if it is empty, writing for example p_e for $(p_e, \emptyset)$.)

P_{TC} defines a strategy where, in parallel, unclassified edges get inactivated if being part of a triangle or activated otherwise (P_e), while P_u repeatedly unclassifies edges with active neighbors. Although this strategy specification provides an intuitive separation of classification and unclassification, and guarantees using a non-applicability condition that no a-triangles arise, still, the possibly overlapping applications of p_e and p_u might create unwanted triangles in our concurrent setting. The above example illustrates the need for a formal analysis methodology to reason about controlled parallel graph-rewriting processes.

3.2 Unmarked Processes

As suggested by the example in the previous section, unmarked processes are terms of a process calculus including prefixing of actions, non-deterministic choice, parallel composition, as well as recursion to express iteration and intended non-termination (e.g., for specifying reactive behaviors).

An action $\gamma = (\rho, N)$ consists of a (possibly parallel) rule name $\rho \in \mathcal{R}^*$ and a set N of rule names, $N = \{p_1, \ldots, p_k\}$. Intuitively, given a graph G such an action can be fired by applying ρ to G only if none of the rules in N is applicable to G. For the definition of unmarked processes, we use the following sets: $\mathcal{K}$ is a set of *process identifiers*, ranged over by A, and $\mathcal{P}$ is the set of *(unmarked) processes*, ranged over by P, Q.

Definition 6 (Unmarked Process Terms)**.** *The syntax of an unmarked process term $P \in \mathcal{P}$ is inductively defined as*

$$P, Q ::= \ \mathbf{0} \ \mid \ \gamma.P \ \mid \ A \ \mid \ P + Q \ \mid \ P \,||\, Q$$

where $A \in \mathcal{K}$ and γ ranges over $\mathcal{R}^* \times 2^{\mathcal{R}}$.

The process $\mathbf{0}$ is the *inactive* process incapable of actions. Given a process P, $\gamma.P$ represents an *action prefix*, meaning that this process can perform an action γ and then continue as P. Process identifiers are used to represent process terms through *defining equations*, and thus might be used to describe recursive process behavior. A defining equation for $A \in \mathcal{K}$ is of the form $A := P$ with $P \in \mathcal{P}$. We assume that each $A \in \mathcal{K}$ has a unique defining equation. $P + Q$ represents a process which non-deterministically behaves either as P or as Q. The *parallel composition* of P and Q, denoted as $P \,||\, Q$, is a process which might interleave the actions of P and Q or even execute them *in parallel*.

There are some syntactically different processes which we treat as equivalent in each context. This relation is called *structural congruence* and denoted $\equiv$.

Definition 7 (Structural Congruence of Unmarked Processes)**.** *The relation $\equiv$ on unmarked process terms is the least equivalence relation s.t.*

$$P \,||\, \mathbf{0} \equiv P \qquad\qquad P + Q \equiv Q + P \qquad\qquad P \,||\, Q \equiv Q \,||\, P$$

The semantics of an unmarked process term is a *labeled transition system* having processes as states and moves labeled by actions as transitions. We first recall the standard definition of labeled transition systems and of their traces.

Definition 8 (Labeled Transition System, Trace, Trace Equivalence)**.** *A* labeled transition system *(LTS) is a tuple $(S, A, \rightarrow_X)$, where S is a set of* states*, A is a set of* actions *containing the distinguished element "$\checkmark$" representing successful*

$$\textsc{struct}\,\frac{P \equiv Q \qquad P \xrightarrow{\alpha} P'}{Q \xrightarrow{\alpha} P'} \qquad \textsc{pre}\,\frac{}{\gamma.P \xrightarrow{\gamma} P} \qquad \textsc{stop}\,\frac{}{\mathbf{0} \xrightarrow{\checkmark} \mathbf{0}}$$

$$\textsc{choice}\,\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \textsc{par}\,\frac{P \xrightarrow{\gamma} P'}{P \,||\, Q \xrightarrow{\gamma} P' \,||\, Q} \qquad \textsc{rec}\,\frac{A := P \qquad P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}$$

$$\textsc{sync}\,\frac{P \xrightarrow{(\rho_1, N_1)} P' \qquad Q \xrightarrow{(\rho_2, N_2)} Q' \qquad \langle \rho_1 \rangle \cap N_2 = \emptyset \qquad \langle \rho_2 \rangle \cap N_1 = \emptyset}{P \,||\, Q \xrightarrow{(\rho_1 | \rho_2, N_1 \cup N_2)} P' \,||\, Q'}$$

Fig. 2: Transition Rules of Unmarked Processes

termination, and $\rightarrow_X \subseteq S \times A \times S$ *is a* transition relation*. As usual, we will write* $s \xrightarrow{a} s'$ *if* $(s, a, s') \in \rightarrow_X$*.*

A trace $t = a_1 a_2 \dots a_n \in A^*$ *of a state* $s \in S$ *is a sequence of actions such that there exist states and transitions with* $s \xrightarrow{a_1}_X s_1 \xrightarrow{a_2}_X \dots \xrightarrow{a_n}_X s_n$*. A trace is* successful *if its last element, and only it, is equal to* $\checkmark$*.*

States $s, s' \in S$ *are* trace equivalent *w.r.t.* $\rightarrow_X$*, denoted as* $\simeq^T_X$*, if* s *and* s' *have the same set of traces.*

The LTS for unmarked process terms is defined by inference rules as follows.

Definition 9 (Unmarked Transition System)**.** *The* unmarked transition system *(UTS) of* $\mathcal{G}$ *is an LTS* $(\mathcal{P}, (\mathcal{R}^* \times 2^{\mathcal{R}}) \cup \{\checkmark\}, \rightarrow)$ *with* $\rightarrow$ *being the least relation satisfying the rules in Fig. 2, where* α *ranges over* $(\mathcal{R}^* \times 2^{\mathcal{R}}) \cup \{\checkmark\}$*,* γ *ranges over* $\mathcal{R}^* \times 2^{\mathcal{R}}$*, and* N *over* $2^{\mathcal{R}}$*.*

Rule STRUCT expresses that structural congruent processes share every transition. Rule PRE states that any action γ appearing as a prefix induces a transition labeled by γ and then the process continues as specified. Rule REC says that process identifiers behave as their defining processes. Rule CHOICE expresses that $P+Q$ can proceed as P or Q by firing any of their transitions (commutativity of $+$ is provided by STRUCT). In the case of *parallel composition*, interleaved actions as in rule PAR mean that one side of the composition proceeds independently of the other. In contrast, SYNC represents *synchronization*, i.e., that the two sides agree on performing their respective actions *in parallel*, which in the case of rules amounts to performing the *composed rule*, where the non-applicability conditions of both sides hold, while both sides proceed. Finally, STOP introduces the special $\checkmark$-transition to denote termination, i.e. that the empty process $\mathbf{0}$ was reached. Note that termination is global, in the sense that a process of the shape $P \,||\, \mathbf{0}$ does not have a $\checkmark$-transition unless $P \equiv \mathbf{0}$.

3.3 Marked Processes

Now, we extend unmarked process specifications by letting them not only specify potential rule sequences, but also operate on a given graph. The states of a marked process are pairs containing an unmarked process and a graph, while the marked transitions correspond to rule applications. Since a concrete rule application is characterized by a DPO diagram (as in Diagram (1)), we include in the labels of the marked transition system not only the names of the applied (parallel) rule and of the rules in the non-applicability condition, but also the resulting DPO diagram.

Definition 10 (Marked Transition System)**.** *The* marked transition system *(*MTS*) is an LTS* $(\mathcal{P} \times |\mathbf{Graph}_T|, \mathcal{R} \times \mathcal{D} \times 2^{\mathcal{R}} \cup \{\checkmark\}, \rightarrow_{\mathcal{D}})$ *where* $\rightarrow_{\mathcal{D}}$ *is the least relation satisfying the following rules and* δ *is a* DPO *diagram over* ρ*:*

$$MARK\frac{P \xrightarrow{(\rho,N)} P' \quad G \overset{\rho@m}{\Longrightarrow} H \quad \forall p \in N : G \overset{p}{\not\Rightarrow}}{(P,G) \xrightarrow{(\rho,\delta,N)}_{\mathcal{D}} (P',H)} \qquad STOP\frac{P \xrightarrow{\checkmark} P'}{(P,G) \xrightarrow{\checkmark}_{\mathcal{D}} (P',G)}$$

It easily follows from the definition that, as desired, traces of controlled graph-rewriting processes correspond to the parallel derivations of Definition 4. In particular, Proposition 2.1 states that every successful trace of a marked process naturally determines a(n underlying) parallel derivation; Proposition 2.2 provides a process definition by recursive choice, which has a successful trace for each linear derivation starting from a given graph, while Proposition 2.3 does the same for parallel (i.e., not necessarily linear) derivations by providing a recursive process allowing for arbitrary parallel composition of the rules as well.

Proposition 2 (Traces and Derivations)**.** *1. Given a marked process* (P, G)*, each of its successful traces uniquely identifies an* underlying parallel derivation *of* $\mathcal{G}$ *starting from* G*. In particular, if* $(\rho_1, \delta_1, N_1) \cdots (\rho_n, \delta_n, N_n)\checkmark$ *is a successful trace of* (P, G)*, then* $\delta_1; \cdots; \delta_n$ *is its underlying derivation.*

2. Let $P_{\mathcal{R}}$ *be the unmarked process defined as follows:*

$$P_{\mathcal{R}} = \mathbf{0} + \textstyle\sum_{p\in\mathcal{R}} p.P_{\mathcal{R}}$$

Then for each graph G *and for each linear derivation* φ *starting from* G *there is a successful trace of* $(P_{\mathcal{R}}, G)$ *such that* φ *is its underlying derivation.*

3. Let $Q_{\mathcal{R}}$ *be the unmarked process defined as follows:*

$$Q_{\mathcal{R}} = \mathbf{0} + \big((\textstyle\sum_{p\in\mathcal{R}} p.\mathbf{0} + \varepsilon.\mathbf{0}) \,||\, Q_{\mathcal{R}}\big)$$

Then for each graph G *and for each parallel derivation* φ *starting from* G *there is a successful trace of* $(Q_{\mathcal{R}}, G)$ *such that* φ *is its underlying derivation.*

4 On the Expressiveness of Unmarked Processes

Unmarked processes are intended to provide a high-level, declarative language for specifying the evolution of systems modeled using graphs and rewriting rules on them. In a trace of the corresponding marked system, as it results from Proposition 2, all the relevant information about the computation are recorded, and this can be exploited for analyses concerned with the truly concurrent aspects of such computations, including causalities, conflicts and parallelism among the individual events. Some preliminary results in this direction are presented in the next section.

In this section, as a proof of concept for the choice of our unmarked processes, we consider an alternative control mechanism for graph rewriting and discuss how it can be encoded into ours. The chosen approach is declarative and abstract like ours, therefore the encoding is pretty simple. Still, it may provide some insights for encoding more concrete and expressive control structures (like those of [15]) which is left as future work.

Habel and Plump [10] interpret *computational completeness* as the ability to compute every computable partial function on labelled graphs: we refer the reader to the cited paper for motivations and details of this notion. They show in [10] that three programming constructs suffice to guarantee computational completeness: (1) non-deterministic choice of a rule from a set of DPO rules, (2) sequential composition, and (3) *maximal iteration*, in the sense that a program is applied repeatedly as long as possible. *Graph Programs* are built using such constructs, and their semantics is defined as a binary relation on abstract graphs relating the start and end graphs of derivations, as recalled by the following definitions.

Definition 11 (Graph Programs [10])**.** Graph programs *over a label alphabet* $\mathcal{C}$ *are inductively defined as follows:*
(1) A finite set of DPO *rules over* $\mathcal{C}$ *is an* elementary *graph program.*
(2) If GP_1 *and* GP_2 *are graph programs, then* $GP_1; GP_2$ *is a graph program.*
(3) If GP *is a graph program by (1) or (2), then* $GP\!\downarrow$ *is a graph program.*
The set of graph programs is denoted as $\mathcal{GP}$*.*

Notice that Graph Programs are based on graphs labeled on a label alphabet $\mathcal{C} = \langle \mathcal{C}_E, \mathcal{C}_N \rangle$, and the main result of completeness exploits constructions based on this assumption. It is an easy exercise to check that such graphs are one-to-one with graphs typed over the type graph $T_{\mathcal{C}} = \langle \mathcal{C}_N, \mathcal{C}_N \times \mathcal{C}_E \times \mathcal{C}_N, \pi_1, \pi_3 \rangle$. It follows that $\mathcal{A}_{\mathcal{C}}$, the class of abstract graphs labeled over $\mathcal{C}$ introduced in [10], is actually isomorphic to $[|\mathbf{Graph}_{T_{\mathcal{C}}}|]$. Nevertheless, we still use $\mathcal{A}_{\mathcal{C}}$ in definitions and results of the rest of this section, when they depend on the concrete representation of graphs as defined in [10].

Definition 12 (Semantics of Graph Programs [10])**.** *Given a program* GP *over a label alphabet* $\mathcal{C}$*, the semantics of* GP *is a binary relation* $\rightarrow_{GP}$ *on* $\mathcal{A}_{\mathcal{C}}$*, which is inductively defined as follows:*
(1) $\rightarrow_{GP} \; = \; \Rightarrow_{GP}$ *if* $GP = \{p_1, \ldots, p_n\}$ *is an elementary program;*
(2) $\rightarrow_{GP_1;GP_2} \; = \; \rightarrow_{GP_2} \circ \rightarrow_{GP_1}$*;*
(3) $\rightarrow_{GP\downarrow} \; = \; \{\langle G, H \rangle \mid G \rightarrow^*_{GP} H$ *and* H *is a normal form w.r.t.* $\rightarrow_{GP}\}$*.*

We show that there is an encoding of Graph Programs in unmarked processes that preserves the semantics.

Definition 13 (Encoding Graph Programs as Processes)**.** *Given unmarked processes* P *and* Q*, their* sequentialization *is the process* $P \fatsemi Q := P[A_Q/\mathbf{0}]$ *where* $A_Q \in \mathcal{K}$ *is a fresh identifier with* $A_Q := Q$ *and* $t[x/y]$ *denotes the syntactic substitution of* x *for* y *in a term* t*.*

The encoding function $[\![\_]\!] : \mathcal{GP} \rightarrow \mathcal{P}$ *is defined as follows:*

- *If* $GP = \{p_1, \ldots, p_n\}$ *is an elementary graph program, then* $[\![GP]\!] := \sum_{i=1}^{n} p_i.\mathbf{0}$.
- $[\![GP_1; GP_2]\!] := [\![GP_1]\!] \mathbin{⨟} [\![GP_2]\!]$.
- $[\![GP{\downarrow}]\!] := A_{GP\downarrow} \in \mathcal{K}$ *where* $A_{GP\downarrow} := [\![GP]\!] \mathbin{⨟} A_{GP\downarrow} + \widehat{[\![GP]\!]}$.

Process $\widehat{[\![GP]\!]}$ *is a process which acts as the identity (and terminates successfully) on all and only the graphs which are normal forms with respect to* $[\![GP]\!]$. *It is defined inductively as follows:*

- $\widehat{[\![GP]\!]} := (\varepsilon, \{p_1, \ldots, p_n\}).\mathbf{0}$ *if* $GP = \{p_1, \ldots, p_n\}$ *is an elementary program;*
- $\widehat{[\![GP_1; GP_2]\!]} := \widehat{[\![GP_1]\!]} + [\![GP_1]\!] \mathbin{⨟} \widehat{[\![GP_2]\!]}$;
- $\widehat{[\![GP{\downarrow}]\!]} := (p, \{p\}).\mathbf{0}$, *where* p *is any rule.*

Proposition 3 (Encoding Preserves Semantics). *For each graph program* GP *and graph* G *in* $Graph_{T_{\mathcal{C}}}$ *it holds* $G \to_{GP} H \quad \text{iff} \quad ([\![GP]\!], G) \to_{\mathcal{D}}^{*} (\mathbf{0}, H)$.

An easy consequence of this precise encoding is that the main result of [10], stating the computational completeness of graph programs, also holds for unmarked processes.

Corollary 1. *Given a label alphabet* $\mathcal{C}$ *and subalphabets* $\mathcal{C}_1$ *and* $\mathcal{C}_2$, *for every computable partial function* $f : \mathcal{A}_{\mathcal{C}_1} \to \mathcal{A}_{\mathcal{C}_2}$, *there exists an unmarked process that computes* f.

5 Equivalence and Independence

In this section, we elaborate on the semantics of marked processes by first introducing an *abstraction* of DPO diagrams to provide a more compact representation of marked transition systems. Afterwards, we investigate different equivalence notions (trace, bisimilarity) and re-interpret parallel independence of actions in marked processes.

Abstract Labels. When reasoning about a system in terms of graphs representing its possible states and of graph transformations modeling its evolution, a natural attitude is to abstract from irrelevant details like the identity of the involved nodes and edges. Formally, this corresponds to considering individual graphs, or also diagrams in the category of graphs, up to isomorphism. By applying this standard abstraction technique to our marked transition systems we define an abstract variant of them having the advantage of exhibiting a state space where branching is bounded under some obvious, mild assumptions. This is certainly valuable for the analysis of such systems, but this is left as future work. On the contrary, note that the marked transition systems of Definition 10 are infinitely branching even for a single rule and a single state, because the pushout object is defined only up to isomorphism.

Definition 14 (Abstract DPO Diagrams). *Given two* DPO *diagrams* δ_1 *and* δ_2 *as in Fig. 1 with each graph indexed by* 1 *and* 2, *respectively, they are equivalent, denoted as* $\delta_1 \cong \delta_2$, *if there exist isomorphisms* $L_1 \to L_2, K_1 \to K_2, R_1 \to R_2, G_1 \to G_2, D_1 \to D_2, H_1 \to H_2$, *such that each arising square commutes.*

An abstract DPO diagram *is an equivalence class* $[\delta] = \{\delta' \mid \delta \cong \delta'\}$. *We denote by* $[\mathcal{D}]$ *the set of abstract* DPO *diagram.*

Definition 15 (Abstract Marked Transition System). *The* abstract marked transition system *(*AMTS*) of $\mathcal{G}$ is an LTS $(\mathcal{P} \times [|\mathbf{Graph}_T|], \mathcal{R} \times [\mathcal{D}] \times 2^{\mathcal{R}} \cup \{\checkmark\}, \rightarrow_{[\mathcal{D}]})$ with $\rightarrow_{[\mathcal{D}]}$ being the least relation satisfying the following rule as well as rule STOP from Def. 10:*

$$\mathit{MARK}\ \frac{P \xrightarrow{(\rho,N)} P' \qquad G \overset{\rho@m}{\Longrightarrow} H \qquad \forall p \in N : G \overset{p}{\not\Rightarrow}}{(P,[G]) \xrightarrow{(\rho,[\delta],N)}_{[\mathcal{D}]} (P',[H])}$$

where $[\delta]$ is an abstract DPO diagram over ρ.

Proposition 4 (AMTS is Finite Branching). *If for each rule $p\colon (L \xleftarrow{l} K \xrightarrow{r} R)$ in $\mathcal{R}$ the left-hand side l is not surjective, then the* AMTS *of $\mathcal{G}$ is finite-branching, i.e., for each unmarked process P and T-typed graph G there is a finite number of transitions from $(P,[G])$.*

Considering graphs and DPO diagrams only up to isomorphism is safe for the kind of equivalences of systems considered below, based on traces or on bisimilarity, but it is known to be problematic for example for a truly concurrent semantics of GTSs [2]. For instance, rule p_u in Sec. 3.1 has two different matches in a graph identical to its left-hand side, but it induces a single abstract DPO diagram. If only the latter is given, it could be impossible to determine whether such rule application is parallel independent or not from another one.

Equivalences. To capture the equivalence of reactive (non-terminating) processes in a branching-sensitive manner, finer equivalence notions are required. *Bisimilarity* is a well-known branching-sensitive equivalence notion.

Definition 16 (Simulation, Bisimulation). *Given an LTS $(S, A, \rightarrow_X)$ and $s, t \in S$. A* simulation *is a relation $\mathsf{R} \subseteq S \times S$ s.t. whenever $s \mathrel{\mathsf{R}} t$, for each transition $s \xrightarrow{\alpha}_X s'$ (with $\alpha \in A$), there exists a transition $t \xrightarrow{\alpha}_X t'$ with $s' \mathrel{\mathsf{R}} t'$. State s is* simulated by *t if there is a simulation relation R such that $s \mathrel{\mathsf{R}} t$.*

A bisimulation *is a symmetric simulation. States s and t are* bisimilar*, denoted $s \simeq_X^{BS} t$, if there is a bisimulation R such that $s \mathrel{\mathsf{R}} t$.*

Note that as we compare labels containing full DPO diagrams involving also input graphs, in order for two MTS processes to be bisimilar, their graphs should be the same concrete graphs. In the case of AMTS, both graphs should be in the same isomorphism class, i.e., they are isomorphic.

First, we state that simulation is “faithful” to synchronization, i.e., a parallel process simulates a sequential one if the latter can apply the corresponding parallel rule.

Proposition 5. *Given unmarked processes P_1, P_2, P_3, Q with bisimilar associated UTSs and actions $\gamma_1 = (\rho_1, N_1), \gamma_2 = (\rho_2, N_2), \gamma_c = (\rho_1|\rho_2, N_1 \cup N_2)$. Let $P_0 := \gamma_1.\gamma_2.P_1 + \gamma_2.\gamma_1.P_2 + \gamma_c.P_3$ and $Q_0 := \gamma_1.Q \,||\, \gamma_2.\mathbf{0}$.*

Then, the process (P_0, G) is simulated by (Q_0, G). Moreover, (P_0, G) and (Q_0, G) are bisimilar if $P_1 := \mathbf{0}$, $P_2 := \mathbf{0}$, $P_3 := \mathbf{0}$ and $Q := \mathbf{0}$.

The following proposition states that, as expected, "concrete" equivalence implies abstract equivalence w.r.t. trace equivalence as well as bisimilarity.

Proposition 6. *Given* $P, Q \in \mathcal{P}$ *and* $G, H \in |\mathbf{Graph}_T|$*, (1)* $(P, G) \simeq^T_{\mathcal{D}} (Q, H)$ *implies* $(P, [G]) \simeq^T_{[\mathcal{D}]} (Q, [H])$ *and (2)* $(P, G) \simeq^{BS}_{\mathcal{D}} (Q, H)$ *implies* $(P, [G]) \simeq^{BS}_{[\mathcal{D}]} (Q, [H])$*.*

Now, to conclude the different kinds of transition systems and their equivalences, we show that unmarked process equivalence and graph isomorphism implies marked process equivalence for both trace equivalence and bisimilarity, in both a concrete and an abstract setting, as expected. (The abstract case is a direct consequence of Proposition 6.)

Proposition 7. *For any* $P, Q \in \mathcal{P}$ *and* $G \in |\mathbf{Graph}_T|$*, (1)* $P \simeq^T Q$ *implies* $(P, G) \simeq^T_{\mathcal{D}} (Q, G)$ *and (2)* $P \simeq^{BS} Q$ *implies* $(P, G) \simeq^{BS}_{\mathcal{D}} (Q, G)$*.*

For a bisimilarity relation, it is an important property if the processes retain bisimilarity if put into different contexts, i.e., if it is a *congruence.* In the following, we show that our abstract AMTS bisimilarity has the desired property of being retained if the control processes are expanded by a further context. (We have a similar result for $\simeq^{BS}_{\mathcal{D}}$ if we set the same concrete starting graph G for both sides.)

Theorem 1. *Given* $P, Q \in \mathcal{P}$ *with* $P \simeq^{BS} Q$ *as well as* $G \in |\mathbf{Graph}_T|$*. Then, the following hold (with* $R \in \mathcal{P}$*):*

1. $(P + R, [G]) \simeq^{BS}_{[\mathcal{D}]} (Q + R, [G])$,
2. $(P \,||\, R, [G]) \simeq^{BS}_{[\mathcal{D}]} (Q \,||\, R, [G])$*, and*
3. $(\gamma.P, [G]) \simeq^{BS}_{[\mathcal{D}]} (\gamma.Q, [G])$ *for any* $\gamma \in \mathcal{R}^* \times 2^{\mathcal{R}}$*.*

Proof. First, we prove that UTS bisimilarity ($\simeq^{BS}$) is a congruence w.r.t. those operators. In particular, $P \simeq^{BS} Q$ implies the following:

1. $P + R \simeq^{BS} Q + R$: The resulting transition system on both sides arises as a union (i.e., "gluing" at the root) of P and R on the left-hand side as well as Q and R on the right-hand side. Then, the statement follows from $P \simeq^{BS} Q$.
2. $P \,||\, R \simeq^{BS} Q \,||\, R$: The proof is done by coinduction. At the starting state, there are three possibilities for firing transitions: (i) P (Q) fires, (ii) R fires or (iii) synchronization (cf. rule SYNC in Fig. 2) takes place.
   Transitions of case (i) are covered w.r.t. bisimilarity through the assumption. If R fires as in case (ii), proceeding to R', the resulting marked states constitute a pair which is of the same form as our starting pair: $P \,||\, R'$ and $Q \,||\, R'$. Thus, the statement holds by coinduction. In case (iii), we have the same situation as in case (ii): if each of the partaking processes $X \in \{P, Q, R\}$ proceeds to X' and synchronization takes place, the resulting processes are $P' \,||\, R'$ and $Q' \,||\, R'$. Moreover, $P' \simeq^{BS} Q'$ due to $P \simeq^{BS} Q$. Thus, the statement holds by coinduction.

3. $\gamma.P \simeq^{BS} \gamma.Q$: Here, on both sides, the outgoing transitions of the starting state correspond to transitions over γ, after which the two sides become P and Q, respectively. Thus, the statement follows from $P \simeq^{BS} Q$.

The statements in the Theorem are easy consequences of the fact that UTS bisimilarity is a congruence, as well as Propositions 6 and 7. □

Note that a similar result would hold only for choice if we do not require unmarked bisimilarity; assuming only marked bisimilarity, parallel composition and prefixing might introduce fresh rule applications, leading to fresh graph states where the behavior of the two sides might diverge. For instance, using rules from Sec. 3.1, $(\rho_u.\mathbf{0}, G) \simeq^{BS}_{[\mathcal{D}]} (\rho_u.\mathbf{0} + \rho_e.\mathbf{0}, G)$ if G has no triangle to apply ρ_e on; however, a parallel (or prefix) context where ρ_a might create a match for ρ_e ruins bisimilarity as the right-side process has a ρ_e-transition that the other cannot simulate.

Summarizing, an abstract representation of a marked transition system might be a useful tool in order to gain a finite representation. Investigating equivalence notions, AMTS exhibits a trade-off between hiding some execution details on the one hand, but enabling a bisimilarity congruence for control processes on the other hand.

Independence. In this section, we demonstrate how abstract marked processes allow for a novel characterization of parallel independence in the context of controlled graph-rewriting processes. Intuitively, two rule applications available simultaneously are *parallel independent* if after performing any of the applications, the other rule is still applicable *on the same match image as in the original rule application* (cf. Proposition 1). In the following, we refer to a 4-tuple of DPO diagrams $\delta_2', \delta_1, \delta_2, \delta_1'$ as *strictly confluent* [7] if they correspond to some matches m_2', m_1, m_2, m_1' as in Proposition 1. Now, we are ready to re-interpret the notion of parallel independence for marked transitions.

Definition 17 (Parallel Transition Independence)**.** *Given an abstract marked process* $(P, [G])$ *with outgoing transitions* $(P, [G]) \xrightarrow{(\rho_1, [\delta_1], N_1)}_{[\mathcal{D}]} (P_1, [H_1])$ *and* $(P, [G]) \xrightarrow{(\rho_2, [\delta_2], N_2)}_{[\mathcal{D}]} (P_2, [H_2])$*, these outgoing* transitions *are* parallel independent *if there exist the following transitions:*

(i) $(P_1, [H_1]) \xrightarrow{(\rho_2, [\delta_2'], N_2)}_{[\mathcal{D}]} (P_{12}, [H_{12}])$*, and*
(ii) $(P_2, [H_2]) \xrightarrow{(\rho_1, [\delta_1'], N_1)}_{[\mathcal{D}]} (P_{21}, [H_{21}])$

such that there is a 4-tuple of (representative) elements of $[\delta_2'], [\delta_1], [\delta_2], [\delta_1']$ *which is strict confluent and* $(P_{12}, [H_{12}]) \simeq^{BS}_{[\mathcal{D}]} (P_{21}, [H_{21}])$*.*

The following proposition states that parallel *transition* independence implies parallel independence of the involved rule applications. Note that the inverse implication does not hold, as parallel independence is defined only for rules and their matches; thus, it might happen that some non-applicability conditions prevent a subsequent rule application even if the applications themselves are parallel independent.

Proposition 8. *Given two parallel independent transitions* $(P, [G]) \xrightarrow{(\rho_1,[\delta_1],N_1)}_{[\mathcal{D}]} (P_1, [H_1])$ *and* $(P, [G]) \xrightarrow{(\rho_2,[\delta_2],N_2)}_{[\mathcal{D}]} (P_2, [H_2])$*, the corresponding rule applications* $\rho_1@m_1$ *and* $\rho_2@m_2$*, respectively, are parallel independent.*

For instance, in our example in Sec. 3.1, if (P_{TC}, G) has outgoing transitions for both p_a and p_u (on some graph G), then we know that the rule applications inducing those transitions were parallel independent. In that case, those applications can be executed also as a synchronized action. Note, instead, that if we consider a different pair of rules like p_e and p_u, not each of their applications to the same graph is independent as their left-hand sides have common elements and thus their matches might overlap.

Finally, we elaborate on the consequences of parallel independence in the presence of synchronization. Particularly, (1) for actions without non-applicability conditions, the absence of a synchronized transition indicates the parallel *dependence* of transitions, and (2) parallel transition independence is equivalent to the existence of a synchronized action in parallel processes, i.e., synchronization implies strict confluence.

Theorem 2 (Bisimilarity and Parallel Independence).
Given bisimilar unmarked processes P_1, P_2, Q_1, Q_2 *and actions* $\gamma_1 = (\rho_1, N_1)$, $\gamma_2 = (\rho_2, N_2)$ *with rules* ρ_1, ρ_2.

1. *Let* $P_0' := \rho_1.(\rho_2 \,||\, P_1) + \rho_2.(\rho_1 \,||\, P_2)$ *and* $Q_0 := \rho_1.Q_1 \,||\, \rho_2.\mathbf{0}$*. There exist no parallel independent applications* $\rho_1@m_1, \rho_2@m_2$ *on* G*, if and only if* $(P_0', [G]) \simeq^{BS}_{\mathcal{D}} (Q_0, [G])$.
2. *Let* $(Q_0', [G]) := (\gamma_1.Q_1 \,||\, \gamma_2.Q_2, [G])$*. Two transitions* $(Q_0', [G]) \xrightarrow{(\rho_1,\delta_1,N_1)}_{[\mathcal{D}]} (Q_1 \,||\, \gamma_2.Q_2, [H_1]), (Q_0', [G]) \xrightarrow{(\rho_2,\delta_2,N_2)}_{[\mathcal{D}]} (\gamma_1.Q_1 \,||\, Q_2, [H_2])$ *are parallel independent if and only if there are transitions* $(Q_0', [G]) \xrightarrow{(\rho_1|\rho_2,\delta_c,N_1\cup N_2)}_{[\mathcal{D}]} (Q_1 \,||\, Q_2, [H])$*,* $(Q_1 \,||\, \gamma_2.Q_2, [G]) \xrightarrow{(\rho_2,\delta_2',N_2)}_{[\mathcal{D}]} (Q_1 \,||\, Q_2, [H])$*, and* $(\gamma_1.Q_1 \,||\, Q_2, [H_2]) \xrightarrow{(\rho_1,\delta_1',N_1)}_{[\mathcal{D}]} (Q_1 \,||\, Q_2, [H])$.

Proof.

1. **If:** We prove the statement indirectly. First, let us observe that if there is a pair of outgoing transitions over ρ_1 and ρ_2 in $(P_0', [G])$, then those transitions are also present in (Q_0, G). Thus, let us assume that there are $\rho_1@m_1, \rho_2@m_2$ parallel independent. Then, there is also an outgoing transition $(\rho_1|\rho_2, [\delta_c], \emptyset)$: We set $m_c : L_1 + L_2 \to G$ of δ_c such that $m_c = m_1 + m_2$. This transition cannot be mimicked by $(P_0', [G])$, a contradiction.
Only if: If there are no parallel independent transitions of ρ_1 and ρ_2, then $(Q_0, [G])$ is unable to synchronize: If there would be a match m_c of $\rho_1|\rho_2$, then there also would be parallel independent matches m_1, m_2 of ρ_1, ρ_2 separately, by taking $m_1 = m_c \circ e_1$ and $m_2 = m_c \circ e_2$, where e_1 and e_2 are the obvious

embeddings of the left-hand sides in the coproduct, i.e., $L_1 \xrightarrow{e_1} L_1 + L_2 \xleftarrow{e_2} L_2$. Thus, the transition sequences induced by ρ_1, ρ_2 are the same in $(P'_0, [G])$ and $(Q_0, [G])$.

2. **If:** The existence of a transition $(\rho_1|\rho_2, [\delta_c], N_1 \cup N_2)$ implies the existence of parallel independent matches m_1, m_2 for ρ_1, ρ_2 due to the construction in Clause 1 above. Thus, there are parallel independent transitions from (Q'_0, G) over $\rho_1@m_1$ and $\rho_2@m_2$, respectively, as we know from the synchronized transition that both N_1 and N_2 hold in G. By the assumption, we also know that there is at least one application of ρ_1 after which ρ_2 is applicable and N_2 holds, and the same vice versa. If those applications were using other matches than m_1, m_2, i.e., if the corresponding DPO diagrams were not isomorphic to $\delta_1, \delta_2, \delta'_1, \delta'_2$, then at least one of the graphs resulting from the sequences $\rho_1.\rho_2$ and $\rho_2.\rho_1$ were not isomorphic to H, the result of the synchronized rule application.

   Only if: This is a direct consequence of Definition 17 and the construction of m_c in Clause 1 above.

□

6 Conclusions and Future Work

In this paper we have introduced an original approach to controlled graph-rewriting, where the control layer is described using terms of a simple process calculus, instead of standard programming constructs as in most other approaches. We have presented an operational semantics for processes, enabling a novel perspective on the equivalence of those processes on the one hand, and (in)dependence of processes running in parallel on the other hand. Among other things, we have shown that congruence of the bisimilarity relation is achieved by abstracting from concrete graph details. Furthermore, we have re-interpreted the notion of parallel independence in our operational setting and shown that synchronization and bisimulation captures parallel (in)dependence as present in controlled graph-rewriting processes.

Among the several topics that we intend to address in future work, we mention (i) to study conditions of bisimilarity and/or simulation among marked processes which are more interesting than those pretty elementary addressed in this paper; (ii) to compare our notion of *process* bisimulation with the *graph-interface* bisimulation of Ehrig and König [8] by including their generalized notion of graph-rewriting steps in our framework; (iii) to investigate a Petri net interpretation, particularly the connection between the non-applicability conditions introduced here and *inhibitor arcs*; (iv) exploiting the process calculus framework, to explore composition (or synchronization) operations that are not conservative with respect to linear derivations, like for example amalgamation; (v) to consider a more elaborate notion of transition independence for capturing true concurrency of rule applications more faithfully.

References

1. Baldan, P., Bruni, A., Corradini, A., König, B., Rodríguez, C., Schwoon, S.: Efficient unfolding of contextual Petri nets. Theor. Comput. Sci. **449**, 2–22 (2012). https://doi.org/10.1016/j.tcs.2012.04.046
2. Baldan, P., Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Löwe, M.: Concurrent Semantics of Algebraic Graph Transformation. In: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3. pp. 107–187. World Scientific (1999)
3. Bunke, H.: Programmed graph grammars. In: Claus, V., Ehrig, H., Rozenberg, G. (eds.) Graph-Grammars and Their Application to Computer Science and Biology, International Workshop. LNCS, vol. 73, pp. 155–166. Springer (1978). https://doi.org/10.1007/BFb0025718
4. Corradini, A., Montanari, U., Rossi, F.: Graph processes. Fundamenta Informaticae **26**(3/4), 241–265 (1996)
5. Corradini, A., Duval, D., Löwe, M., Ribeiro, L., Machado, R., Costa, A., Azzi, G.G., Bezerra, J.S., Rodrigues, L.M.: On the essence of parallel independence for the double-pushout and sesqui-pushout approaches. In: Graph Transformation, Specifications, and Nets: In Memory of Hartmut Ehrig. LNCS, vol. 10800, pp. 1–18. Springer (2018)
6. Dassow, J., Păun, G., Salomaa, A.: Grammars with Controlled Derivations. In: Handbook of Formal Languages: Volume 2. pp. 101–154. Springer (1997)
7. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer (2006)
8. Ehrig, H., König, B.: Deriving bisimulation congruences in the dpo approach to graph rewriting. In: Walukiewicz, I. (ed.) FoSSaCS. LNCS, vol. 2987, pp. 151–166. Springer (2004)
9. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In: TAGT 98. LNCS, vol. 1764, pp. 157–167. Springer (2000)
10. Habel, A., Plump, D.: Computational completeness of programming languages based on graph transformation. In: FoSSaCS. LNCS, vol. 2987, pp. 230–245. Springer (2001)
11. Kluge, R., Stein, M., Varró, G., Schürr, A., Hollick, M., Mühlhäuser, M.: A systematic approach to constructing families of incremental topology control algorithms using graph transformation. Software & Systems Modeling **38**, 47 – 83 (2017)
12. Leblebici, E., Anjorin, A., Schürr, A.: Developing eMoflon with eMoflon. In: ICMT. LNCS, vol. 8568, pp. 138–145. Springer (2014)
13. Plump, D., Steinert, S.: The Semantics of Graph Programs. In: RULE. EPTCS, vol. 21 (2009)
14. Schürr, A.: Logic-Based Programmed Structure Rewriting Systems. Fundam. Inf. **26**(3,4), 363–385 (1996)
15. Schürr, A., Winter, A.J., Zündorf, A.: The PROGRES-Approach: Language and Environment. In: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2, pp. 487–550. World Scientific (1999). https://doi.org/10.1142/9789812815149_0013