Università di Pisa

Dipartimento di Informatica

Technical Report : TR-96-17

Algebraic Approaches to Graph Transformation, Part I: Basic Concepts and Double Pushout Approach

A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, M. Loewe

March 21, 1996

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

ALGEBRAIC APPROACHES TO GRAPH TRANSFORMATION PART I: BASIC CONCEPTS AND DOUBLE PUSHOUT APPROACH

A. CORRADINI, U. MONTANARI, F. ROSSI
Dipartimento di Informatica, Corso Italia 40, I-56125 Pisa, Italy

H. EHRIG, R. HECKEL, M. LÖWE
Technische Universität Berlin, Fachbereich 13 Informatik, Franklinstraße 28/29, D-10587 Berlin, Germany

The algebraic approaches to graph transformation are based on the concept of gluing of graphs, modelled by pushouts in suitable categories of graphs and graph morphisms. This allows one not only to give an explicit algebraic or set theoretical description of the constructions, but also to use concepts and results from category theory in order to build up a rich theory and to give elegant proofs even in complex situations. In this chapter we start with an overwiev of the basic notions common to the two algebraic approaches, the *double-pushout (DPO) approach* and the *single-pushout (SPO) approach*; next we present the classical theory and some recent development of the double-pushout approach. The next chapter is devoted instead to the single-pushout approach, and it is closed by a comparison between the two approaches.

1 Introduction

The algebraic approach to graph grammars has been invented at the Technical University of Berlin in the early seventies by H. Ehrig, M. Pfender and H.J. Schneider in order to generalize Chomsky grammars from strings to graphs [1]. The main idea was to generalize the concatenation of strings to a gluing construction for graphs. This allowed to formulate a graph rewriting step by two gluing constructions. The approach was called "algebraic" because graphs are considered as special kinds of algebras and the gluing for graphs is defined by an "algebraic construction", called *pushout*, in the category of graphs and total graph morphisms. This basic idea allows one to apply general results from algebra and category theory in the algebraic theory of graph grammars. In addition to the idea to generalize Chomsky grammars from strings to graphs, it was B. K. Rosen's idea to use graph grammars as a generalization of term rewriting systems where terms are represented by specific kinds of trees. In fact, graphs are suitable to model terms with shared subterms, which are no longer trees, and therefore term rewriting with shared subterms can be modelled by graph rewriting.

Graph grammars in general provide an intuitive description for the manipulation of graphs and graphical structures as they occur in programming language semantics, data bases, operating systems, rule based systems and various kinds of software and distributed systems. In most of these cases graph grammars allow one to give a formal graphical specification which can be executed as far as corresponding graph grammar tools are available. Moreover, theoretical results for graph grammars are useful for analysis, correctness and consistency proofs of such systems. The algebraic approach [2,3,4,5] has been worked out for several years and provides interesting results for parallelism analysis [6,7], efficient evaluation of functional expressions [8,9] and logic programs [10], synchronization mechanisms [11], distributed systems [11,12,13,5], object-oriented systems [14], applied software systems [15], implementation of abstract data types [16] and context-free hyperedge replacement [17] (see also [18] in this book).

Historically, the first of the algebraic approaches to graph transformation is the so-called *double-pushout (DPO) approach* introduced in [1], which owes its name to the basic algebraic construction used to define a direct derivation step: This is modelled indeed by two gluing diagrams (i.e., pushouts) in the category of graphs and total graph morphisms. More recently a second algebraic approach has been proposed in [5], which defines a basic derivation step as a *single* pushout in the category of graphs and *partial* graph morphisms: Hence the name of *single-pushout (SPO) approach*.[a] This chapter includes an informal description of the basic concepts and results common to the DPO and SPO approaches, followed by a more detailed, formal presentation of the DPO approach. On the other hand, the SPO approach is the main topic of the next chapter, [19], which is closed by a formal comparison of the two approaches.

More precisely, in Section 2 many relevant concepts and results of the algebraic approaches to graph transformation are introduced in the form of problems stated in quite an abstract way: This provides an outline that will be followed (to a great extent) along the presentation of the two approaches, each of which will provide its own answers to such problems. The stated problems concern independence and parallelism of direct derivations, embedding of derivations, amalgamation, and distribution. The last part of the section shortly recalls some other issues addressd in the literature of the algebraic approaches, including computation based semantics, application conditions, structuring of grammars, consistency conditions, and generalization of the algebraic approaches to other structures. Some of these topics will be presented

[a] Other definitions of graph rewriting using a single pushout in a category of partial graph morphisms exist: Their relationship with the SPO approach is discussed in Section 2.2 of the next chapter

in detail for just one of the two approaches, namely computation based semantics (in particular the so-called *models of computation*) for the DPO, and application conditions for the SPO approach; for the other topics relevant references are indicated only.

Section 3 starts the presentation of the DPO approach by introducing the very basic notions, including the category of graphs we shall use and the definitions of production, direct derivation, and derivation; also the fundamental notion of pushout and the gluing conditions (i.e., the conditions that ensures the applicability of a production at a given match) are discussed in some depth. Section 4 presents the main results of the theory of parallelism for the DPO approach, including the notions of parallel and sequential independence, the Local Church Rosser and the Parallelism theorems, and the synthesis and analysis constructions. This section on the one hand answers the problems about independence and parallelism raised in Section 2.2 and, on the other hand, it provides some basic notions exploited in the following section.

Section 5 presents various models of computations for graph grammars on different levels of abstraction, i.e., various categories having graphs as objects and graph derivations as arrows. All such models are obtained from the most concrete one by imposing suitable equivalences on graphs and derivations. In particular, in the most abstract model, called the *abstract truly-concurrent model*, all isomorphic graphs are identified, as well as all derivations that are related by the classical shift-equivalence (which considers as equal all derivations which differ for the order of independent direct derivations only) or by equivalence $\equiv_3$, which relates derivations that are isomorphic and satisfy a further technical (but fundamental) condition. Finally Section 6 summarizes the main results concerning embedding of derivations, amalgamation and distribution for the DPO approach, providing answers for the remaining problems raised in Section 2.

Two appendices close the chapter. Appendix A introduces the definition of binary coproducts in a category, and shows that coproducts cannot be assumed to be commutative (in a strict way) unless the category is a preorder. This fact justifies the way parallel productions are defined and manipulated in Section 4, which may appear more complex than the corresponding definitions in previous papers in the literature. Appendix B presents the proof of the main technical result of Section 5. That proof is based on the analysis and synthesis constructions which are reported as well, not only for completeness, but also because they demonstrate a typical example of reasoning that is used when results in the algebraic approach to graph transformations are proven on an abstract level.

A formal, detailed comparison of the DPO and SPO approaches is presented in Section 6 of the next chapter, where it is also shown that the DPO approach can be embedded (via a suitable translation) into the SPO approach.

All the categorical notions used in this chapter are explicitly introduced; nevertheless, some knowledge of the basic concepts of category theory would be helpful. Standard references are [20,21].

How to read the chapters on the algebraic approaches.[b]
Readers interested in just one of the algebraic approaches are suggested to read either this chapter (for the DPO approach), or Section I.2 and then all Part II excluded Section II.6 (for the SPO approach; a few references to Section I.3 will have to be followed).

Alternatively, some parts may be skipped, depending on the specific interests of the reader. All readers (and in particular newcomers) are encouraged to read Section I.2 first, where they can get a precise, although informal idea of the main topics addressed in the theory of the algebraic approaches. Such topics are then presented with all technical details for the DPO approach in Section I.3, in the first part of Section I.4 (till Theorem 17), and in Section I.6. For the SPO approach, they are presented instead in Section II.2 and II.3. After these sections, the reader interested in a careful comparison of the two approaches can read Section II.6.

2 Overview of the Algebraic Approaches

The purpose of this section is to provide an overview of the two algebraic approaches to graph transformation, the double-pushout (DPO) and the single-pushout (SPO) approach. Taking a problem-oriented point of view allows us to discuss the main questions raised (and solutions proposed) by both approaches quite informally using the same conceptual terminology. Explicit technical constructions and results are then presented for the DPO approach in the remaining sections of this chapter, and for the SPO approach in the next chapter. A detailed comparison of the two approaches can be found in Section 6 of the next chapter.

Among the various approaches to graph transformation, the algebraic approaches are characterized by the use of categorical notions for the very basic definitions of graph transformation rules (or *productions*, as they are usually called), of matches (i.e., of occurrences of the left-hand side of a production in a graph), and of rule applications (called *direct derivations*). The main advantage is clearly that techniques borrowed from category theory can be used for

[b] In this paragraph "I" refers to "Part I", i.e., this chapter, and "II" to "Part II", i.e., the next chapter.

proving properties and results about graph transformation, and such proofs are often be simpler than corresponding proofs formulated without categorical notions. In addition, such proofs are to a great extent independent from the structure of the objects that are rewritten. Therefore the large body of results and constructions of the algebraic approaches can be extended quite easily to cover the rewriting of arbitrary structures (satisfying certain properties), as shown for example by the theory of High-Level Replacement Systems [22,23].

In this section, we first introduce informally the basic notions of graph, production and derivation for the DPO and SPO approaches in Section 2.1. We address questions concerning the independence of direct derivations and their parallel application in Section 2.2; the embedding of derivations in larger contexts and the related notion of derived production in Section 2.3; and the relationship between the amalgamation of productions along a common subpart (a sort of synchronization) and the distributed application of several productions to a graph in Section 2.4. Finally Section 2.5 shortly recalls some other issues addressed in the literature of the algebraic approaches, including computation based semantics, application conditions, structuring of grammars, consistency conditions, and generalization of the algebraic approaches to other structures.

2.1 Graphs, Productions and Derivations

The basic idea of all graph transformation approaches is to consider a *production* $p : L \rightsquigarrow R$, where graphs L and R are called the left- and the right-hand side, respectively, as a finite, schematic description of a potentially infinite set of *direct derivations*. If the *match* m fixes an occurrence of L in a *given graph* G, then $G \overset{p,m}{\Longrightarrow} H$ denotes the direct derivation where p is applied to G leading to a *derived graph* H. Intuitively, H is obtained by replacing the occurrence of L in G by R. The essential questions distinguishing the particular graph transformation approaches are:

1. What is a "graph"?

2. How can we match L with a subgraph of G?

3. How can we define the replacement of L by R in G?

In the algebraic approaches, a graph is considered as a two sorted *algebra* where the sets of vertices V and edges E are the carriers, while source $s : E \to V$ and target $t : E \to V$ are two unary operations. Moreover we have label functions $lv : V \to L_V$ and $le : E \to L_E$, where L_V and L_E are fixed label

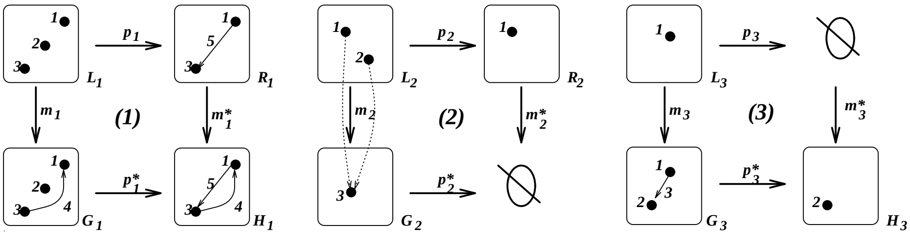


Figure 1: Direct derivations.

alphabets for vertices and edges, respectively. Since edges are objects on their own right, this allows for multiple (parallel) edges with the same label.

Each graph production $p : L \rightsquigarrow R$ defines a partial correspondence between elements of its left- and of its right-hand side, determining which nodes and edges have to be preserved by an application of p, which have to be deleted, and which must be created. To see how this works, consider the production $p_1 : L_1 \rightsquigarrow R_1$ which is applied on the left of Figure 1 to a graph G_1, leading to the direct derivation (1). The production assumes three vertices on the left-hand side L_1, and leaves behind two vertices, connected by an edge, on the right-hand side R_1. The numbers written near the vertices and edges define a partial correspondence between the elements of L_1 and R_1: Two elements with the same number are meant to represent the same object before and after the application of the production. In order to apply p_1 to the given graph G_1, we have to find an occurrence of the left-hand side L_1 in G_1, called match in the following. A match $m : L \to G$ for a production p is a graph homomorphism, mapping nodes and edges of L to G, in such a way that the graphical structure and the labels are preserved. The match $m_1 : L_1 \to G_1$ of the direct derivation (1) maps each element of L_1 to the element of G_1 carrying the same number. Applying production p_1 to graph G_1 at match m_1 we have to delete every object from G_1 which matches an element of L_1 that has no corresponding element in R_1, i.e., vertex 2 of G_1 is deleted. Symmetrically, we add to G_1 each element of R_1 that has no corresponding element in L_1, i.e., edge 5 is inserted. All remaining elements of G_1 are preserved, thus, roughly spoken, the derived graph H_1 is constructed as $G_1 - (L_1 - R_1) \cup (R_1 - L_1)$.

The application of a production can be seen as an embedding into a context, which is the part of the given graph G that is not part of the match, i.e., the edge 4 in our example. Hence, at the bottom of the direct derivation, the *co-production* $p_1^* : G_1 \rightsquigarrow H_1$ relates the given and the derived graphs, keeping track of all elements that are preserved by the direct derivation. Sym-

$$
\begin{tikzcd}
L \arrow[r, "p"] \arrow[d, "m"] & R \arrow[d, "m^*"] \\
G \arrow[r, "p^*"] & H
\end{tikzcd}
$$

Figure 2: Example and schematic representation of direct derivation $G \stackrel{p,m}{\Longrightarrow} H$.

metrically, the *co-match* ${m^*}_1$, which is a graph homomorphism, too, maps the right-hand side R_1 of the production to its occurrence in the derived graph H_1. A direct derivation from G to H resulting from an application of a production p at a match m is schematically represented as in Figure 2, and denoted by $d = (G \stackrel{p,m}{\Longrightarrow} H)$.

In general L does not have to be isomorphic to its image $m(L)$ in G, i.e., elements of L may be identified by m. This is not always unproblematic, as we may see in the direct derivation (2) of Figure 1. A production p_2, assuming two vertices and deleting one of them, is applied to a graph G_2 containing only one vertex, i.e., vertices 1 and 2 of L_2 are both mapped to the vertex 3 of G_2. Thus, the production p_2 specifies both the deletion and the preservation of 3: We say that the match m_2 contains a conflict. There are three possible ways to solve this conflict: Vertex 3 of G_2 may be preserved, deleted, or the application of the production p_2 at this match may be forbidden. In the derivation (2) of Figure 1 the second alternative has been choosen, i.e., the vertex is deleted.

Another difficulty may occur if a vertex shall be deleted which is connected to an edge that is not part of the match, as shown in direct derivation (3) of Figure 1: Deleting vertex 1 of G_3, as specified by production p_3, would leave behind the edge 3 without a source vertex, i.e., the result would no longer be a graph. Again there are two ways to avoid such dangling edges. We may either delete the edge 3 together with its source node, as in derivation (3) of Figure 1, or forbid the application of p_3.

The basic difference between the DPO and the SPO approach lies in the way they handle the above problematic situations. In the DPO approach, rewriting is not allowed in these cases (i.e., productions p_2 and p_3 are not applicable to matches m_2 and m_3, respectively). On the contrary, in the SPO approach such direct derivations are allowed, and deletion has priority over preservation; thus (2) and (3) of Figure 1 are legal SPO direct derivations.

In both algebraic approaches direct derivations are modeled by gluing constructions of graphs, that are formally characterized as pushouts in suitable categories having graphs as objects, and (total or partial) graph homomorphisms as arrows. A production in the DPO approach is given by a pair $L \xleftarrow{l} K \xrightarrow{r} R$ of graph homomorphisms from a common *interface graph* K, and a direct derivation consists of two gluing diagrams of graphs and total graph morphisms, as (1) and (2) in the left diagram below. The *context graph* D is obtained from the given graph G by deleting all elements of G which have a pre-image in L, but none in K. Via diagram (1) this deletion is described as an inverse gluing operation, while the second gluing diagram (2) models the actual insertion into H of all elements of R that do not have a pre-image in K. In order to avoid problematic situations like (2) and (3) in Figure 1, in the DPO approach the match m must satisfy an application condition, called the *gluing condition.* This condition consists of two parts. To ensure that D will have no dangling edges, the *dangling condition* requires that if p specifies the deletion of a vertex of G, then it must specify also the deletion of all edges of G incident to that node. On the other hand the *identification condition* requires that every element of G that should be deleted by the application of p has only one pre-image in L. The gluing condition ensures that the application of p to G deletes exactly what is specified by the production.

$$\begin{tikzcd} L \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] \arrow[dr, phantom, "(2)"] & R \arrow[d, "m^*"] \\ G & D \arrow[l, "l^*"] \arrow[r, "r^*"'] & H \end{tikzcd} \qquad \begin{tikzcd} L \arrow[r, "p"] \arrow[d, "m"'] \arrow[dr, phantom, "(3)"] & R \arrow[d, "m^*"] \\ G \arrow[r, "p^*"'] & H \end{tikzcd}$$

In the SPO approach a production p is a partial graph homomorphism. A direct derivation is given by a single gluing diagram, as (3) in the right diagram above, where the match m is required to be total. No gluing condition has to be satisfied for applying a production, and this may result in side effects as shown in Figure 1. In fact, on the one hand, the deletion of a node of G automatically causes the deletion of all incident edges, even if this is not specified by the production. On the other hand conflicts between deletion and preservation of nodes and edges are solved in favor of deletion, and in this case the co-match m^* becomes a partial homomorphism, because elements of the right-hand side that should have been preserved but are deleted because of some conflict do not have an image in H.

Direct derivations in the SPO approach are more expressive than DPO derivations, in the sense that they may model effects that can not be obtained in the more restricted DPO approach. A formal correspondence between direct derivations in the two approaches is presented in Section 6 of the next chapter.

A *graph grammar* $\mathcal{G}$ consists of a set of productions P and a *start graph* G_0. A sequence of direct derivations $\rho = (G_0 \stackrel{p_1}{\Longrightarrow} G_1 \stackrel{p_2}{\Longrightarrow} \dots \stackrel{p_n}{\Longrightarrow} G_n)$ constitutes a *derivation* of the grammar, also denoted by $G_0 \Longrightarrow^* G_n$. The *language* $\mathcal{L}(\mathcal{G})$

generated by the grammar $\mathcal{G}$ is the set of all graphs G_n such that $G_0 \Longrightarrow^* G_n$ is a derivation of the grammar.

2.2 Independence and Parallelism

Parallel computations can be described in two different ways. If we stick to a sequential model, two parallel processes have to be modeled by interleaving arbitrarily their atomic actions, very similarly to the way multitasking is implemented on a single processor system. On the contrary, explicit parallelism means to have one processor per process, which allows the actions to take place simultaneously.

Interleaving

Considering the interleaving approach first, two actions are concurrent, i.e., potentially in parallel, if they may be performed in any order with the same result. In terms of graph transformations, the question whether two actions (direct derivations) are concurrent or not can be asked from two different points of view.

1. Assume that a given graph represents a certain system state. The next evolution step of this state is obtained by the application of a production at a certain match, where the production and the match are chosen non-deterministically from a set of possible alternatives. Clearly, each choice we make leads to a distinct derivation sequence, but the question remains, whether two of these sequences indeed model different computations or if we have only chosen one of two equivalent interleavings. (Here, two derivation sequences from some graph G_0 to G_n are equivalent if both insert and delete exactly the same nodes and edges.) In other words, given two alternative direct derivations $H_1 \stackrel{p_1}{\Longleftarrow} G \stackrel{p_2}{\Longrightarrow} H_2$ we ask ourselves if there are direct derivations $H_1 \stackrel{p_2}{\Longrightarrow} X \stackrel{p_1}{\Longleftarrow} H_2$, showing that the two given direct derivations are not mutually exclusive, but each of them can instead be postponed after the application of the other, yielding the same result.

2. Given a derivation, intuitively two consecutive direct derivations $G \stackrel{p_1}{\Longrightarrow} H_1 \stackrel{p_2}{\Longrightarrow} X$ are concurrent if they can be performed in a different order, as in $G \stackrel{p_2}{\Longrightarrow} H_2 \stackrel{p_1}{\Longrightarrow} X$, without changing the result. The existence of two different orderings ensures that there is no causal dependency between these applications of p_1 and p_2.

Summarizing, we can say that two alternative derivations are concurrent if they are not mutually exclusive while two consecutive derivations are concurrent if they are not causally dependent. To stress the symmetry between these two conditions they are called *parallel* and *sequential independence*, respectively, in the more formal statement of this problem below. Originally investigated in view of the local confluence property of (term) rewriting systems, it is called the Local Church-Rosser Problem.

Problem 1 (Local Church Rosser).

1. *Find a condition, called* parallel independence*, such that two alternative direct derivations $H_1 \overset{p_1}{\Longleftarrow} G \overset{p_2}{\Longrightarrow} H_2$ are parallel independent iff there are direct derivations $H_1 \overset{p_2}{\Longrightarrow} X$ and $H_2 \overset{p_1}{\Longrightarrow} X$ such that $G \overset{p_1}{\Longrightarrow} H_1 \overset{p_2}{\Longrightarrow} X$ and $G \overset{p_2}{\Longrightarrow} H_2 \overset{p_1}{\Longrightarrow} X$ are equivalent.*

$$\begin{tikzcd}
 & H_1 \arrow[dr, "p_2"] & \\
G \arrow[ur, "p_1"] \arrow[dr, "p_2"'] & & X \\
 & H_2 \arrow[ur, "p_1"'] &
\end{tikzcd}$$

2. *Find a condition, called* sequential independence*, such that a derivation $G \overset{p_1}{\Longrightarrow} H_1 \overset{p_2}{\Longrightarrow} X$ is sequentially independent iff there is an equivalent derivation $G \overset{p_2}{\Longrightarrow} H_2 \overset{p_1}{\Longrightarrow} X$.* □

More concretely, two alternative direct derivations are parallel independent if their matches do only overlap in items that are preserved by both derivations, that is, if none of the two deletes any item that is also accessed by the other one. If an item is preserved by one of the direct derivations, say $G \overset{p_1}{\Longrightarrow} H_1$, but deleted by the other one, the second one can be delayed after the first; we say that $G \overset{p_2}{\Longrightarrow} H_2$ is *weakly parallel independent* of $G \overset{p_1}{\Longrightarrow} H_1$. *Parallel independence* can then be defined as mutual weakly parallel independence.

Two consecutive direct derivations are sequentially independent if (a) the match of the second one does not depend on elements generated by the first one, and (b) the second derivation does not delete an item that has been accessed by the first. The first part of this condition ensures that p_2 may already be applied before p_1, leading to an alternative direct derivation $G \overset{p_2}{\Longrightarrow} H_2$. We say that $H_1 \overset{p_2}{\Longrightarrow} X$ is *weakly sequentially independent* of $G \overset{p_1}{\Longrightarrow} H_1$. From (b) we can show that the latter is weakly parallel independent of $G \overset{p_2}{\Longrightarrow} H_2$ in the above sense. So two consecutive direct derivations $G \overset{p_1}{\Longrightarrow} H_1 \overset{p_2}{\Longrightarrow} X$ are *sequentially independent* if $H_1 \overset{p_2}{\Longrightarrow} X$ is weakly sequentially independent of $G \overset{p_1}{\Longrightarrow} H_1$ and the latter is weakly *parallel* independent of $G \overset{p_2}{\Longrightarrow} H_2$.

Explicit Parallelism

Parallel computations can be represented more directly by parallel application of productions. *Truly* parallel application of productions essentially requires to abstract from any possible application order, which implies that no intermediate graph is generated. In other words, a (true) parallel application may be modeled by the application of a single production, the *parallel production*.

Given productions $p_1 : L_1 \leadsto R_1$ and $p_2 : L_2 \leadsto R_2$, their parallel composition is denoted by $p_1 + p_2 : L_1 + L_2 \leadsto R_1 + R_2$. Intuitively, $p_1 + p_2$ is just the disjoint union of p_1 and p_2. Direct derivations like $G \stackrel{p_1+p_2}{\Longrightarrow} X$, using a parallel production $p_1 + p_2$, are referred to as *parallel direct derivations*.

Now we have seen two different ways to model parallel computations by graph transformation, either using interleaving sequences or truly parallel derivations. How are they related? This question can be asked from two different points of view. Given a parallel direct derivation $G \stackrel{p_1+p_2}{\Longrightarrow} X$ we may look for a sequentialization, say $G \stackrel{p_1}{\Longrightarrow} H_1 \stackrel{p_2}{\Longrightarrow} X$. On the other hand, starting from the latter sequence we can try to put the two consecutive direct derivations in parallel. The problem of finding necessary and sufficient conditions for these sequentialization and parallelization constructions, also called *analysis* and *synthesis* in the literature, is stated in the following way.

Problem 2 (Parallelism).

1. *Find a* sequentialization condition *such that a parallel direct derivation $G \stackrel{p_1+p_2}{\Longrightarrow} X$ satisfies this condition iff there is an equivalent sequential derivation $G \stackrel{p_1}{\Longrightarrow} H_1 \stackrel{p_2}{\Longrightarrow} X$ (or $G \stackrel{p_2}{\Longrightarrow} H_2 \stackrel{p_1}{\Longrightarrow} X$, respectively).*

$$\begin{tikzcd}
 & H_1 \arrow[dr, "p_2"] & \\
G \arrow[ur, "p_1"] \arrow[rr, "p_1+p_2"] \arrow[dr, "p_2"'] & & X \\
 & H_2 \arrow[ur, "p_1"'] &
\end{tikzcd}$$

2. *Find a* parallelization condition *such that a sequential derivation $G \stackrel{p_1}{\Longrightarrow} H_1 \stackrel{p_2}{\Longrightarrow} X$ (or $G \stackrel{p_2}{\Longrightarrow} H_2 \stackrel{p_1}{\Longrightarrow} X$) satisfies this condition iff there is an equivalent parallel direct derivation $G \stackrel{p_1+p_2}{\Longrightarrow} X$.* □

The solution to this problem is closely related to the notions of sequential and parallel independence. In the double pushout approach it turns out that each parallel direct derivation $G \stackrel{p_1+p_2}{\Longrightarrow} X$ may be sequentialized to derivations

$G \stackrel{p_1}{\Longrightarrow} H_1 \stackrel{p_2}{\Longrightarrow} X$ and $G \stackrel{p_2}{\Longrightarrow} H_2 \stackrel{p_1}{\Longrightarrow} X$, which are in turn sequentially independent; indeed, the sequentialization condition is already part of the gluing condition for the application of the parallel production. Vice versa, consecutive direct derivations can be put in parallel if they are sequentially independent, thus the parallelization condition requires sequential independence of the derivation. Hence in the double pushout approach the two ways of modeling concurrency are in fact equivalent, in the sense that they allow one to represent exactly the same amount of parallelism.

In the single pushout setting, however, a parallel direct derivation $G \stackrel{p_1+p_2}{\Longrightarrow} X$ can be sequentialized to $G \stackrel{p_1}{\Longrightarrow} H_1 \stackrel{p_2}{\Longrightarrow} X$ if $G \stackrel{p_2}{\Longrightarrow} H_2$ is weakly parallel independent of $G \stackrel{p_1}{\Longrightarrow} H_1$, where the two alternative direct derivations from G are obtained by restricting the match of p_1+p_2 into G to the left-hand side of p_1 and p_2, respectively. Since for a given parallel direct derivation this condition may not hold, it turns out that in this setting parallel direct derivations can express a higher degree of concurrency with respect to derivation sequences, because there are parallel direct derivations that do not have an equivalent interleaving sequence. On the other hand, $G \stackrel{p_1}{\Longrightarrow} H_1 \stackrel{p_2}{\Longrightarrow} X$ can be put in parallel if the second direct derivation is weakly sequentially independent of the first one.

2.3 Embedding of Derivations and Derived Productions

Systems are usually not monolithic, but consist of a number of subsystems which may be specified and implemented separately. After defining each subsystem over its own local state space, they have to be joined together in order to build the whole system. Then the local computations of each subsystem have to be embedded into the computations of the enclosing system.

Speaking about derivations, $\rho = (G_0 \stackrel{p_1}{\Longrightarrow} \cdots \stackrel{p_n}{\Longrightarrow} G_n)$ is embedded into a derivation $\delta = (X_0 \stackrel{p_1}{\Longrightarrow} \cdots \stackrel{p_n}{\Longrightarrow} X_n)$ if both have the same length n and each graph G_i of ρ is embedded into the corresponding graph X_i of δ in a compatible way. The embedding is represented by a family of injections $(G_i \xrightarrow{e_i} X_i)_{i \in \{0,\ldots n\}}$, denoted by $\rho \xrightarrow{e} \delta$. If it exists, it is uniquely determined by the first injection $e_0 : G_0 \to X_0$. Given ρ and e_0, the following problem asks under which conditions there is a derivation δ such that e_0 induces an embedding e of ρ into δ.

Problem 3 (Embedding). *Find an* embedding condition *such that, for a given derivation $\rho = (G_0 \stackrel{p_1}{\Longrightarrow} \cdots \stackrel{p_n}{\Longrightarrow} G_n)$, an injection $e_0 : G_0 \to X_0$ satisfies this condition iff there is a derivation $\delta = (X_0 \stackrel{p_1}{\Longrightarrow} \cdots \stackrel{p_n}{\Longrightarrow} X_n)$ and an embedding $e : \rho \to \delta$.*

$$
\begin{tikzcd}
G_0 \arrow[r, "p_1"] \arrow[d, "e_0"] & G_1 \arrow[r, "p_1"] \arrow[d, "e_1"] & \cdots \arrow[r, "p_n"] & G_n \arrow[d, "e_n"] \\
X_0 \arrow[r, "p_1"] & X_1 \arrow[r, "p_1"] & \cdots \arrow[r, "p_n"] & X_n
\end{tikzcd}
$$

□

Let us consider the simpler problem of embedding a direct derivation $G \stackrel{p}{\Longrightarrow} H$ along an injection $e_0 : G \to X$ first. According to our general introduction, the direct derivation of Figure 2 defines a co-production $p^* : G \rightsquigarrow H$, relating the given and the derived graphs. The injection $e_0 : G \to X$ provides us with a match for this *directly derived production* p^*. The embedding condition for $G \stackrel{p}{\Longrightarrow} H$ is now equivalent to the applicability of the directly derived production p^* at the match e_0.

This idea can be generalized to derivations of arbitrary length: Given $\rho = (G_0 \stackrel{p_1}{\Longrightarrow} \cdots \stackrel{p_n}{\Longrightarrow} G_n)$, we have to define a *derived production* $p^* : G_0 \rightsquigarrow G_n$ that is applicable at an injective match $G_0 \stackrel{e_0}{\longrightarrow} X_0$ if and only if e_0 induces an embedding e of ρ into a derivation δ.

Problem 4 (Derived Production). *Let $\rho = (G_0 \stackrel{p_1}{\Longrightarrow} \cdots \stackrel{p_n}{\Longrightarrow} G_n)$ be a derivation. Find a* derived production $p^* : G_0 \rightsquigarrow G_n$ *such that for each injection $G_0 \stackrel{e_0}{\longrightarrow} X_0$ there is a direct derivation $X_0 \stackrel{p^*}{\Longrightarrow} X_n$ at e_0 iff e_0 induces an embedding $e : \rho \to \delta$ of ρ into a derivation $\delta = (X_0 \stackrel{p_1}{\Longrightarrow} \cdots \stackrel{p_n}{\Longrightarrow} X_n)$.* □

In order to obtain this derived production we will introduce a *sequential composition* $p_1; p_2$ of two productions $p_1 : L_1 \rightsquigarrow R_1$ and $p_2 : L_2 \rightsquigarrow R_2$ with $R_1 = L_2$. Then we define the *derived production* p^* for a given derivation sequence $G_0 \stackrel{p_1}{\Longrightarrow} \cdots \stackrel{p_n}{\Longrightarrow} G_n$ by $p^* = p_1^*; \ldots; p_n^*$.

The embedding condition of $G_0 \stackrel{p_1}{\Longrightarrow} \cdots \stackrel{p_n}{\Longrightarrow} G_n$ for e_0 is equivalent to the applicability of the derived production p^* at this match. In the DPO approach this means that e_0 has to satisfy the gluing condition for p^*, which in this case reduces to the dangling condition, since the identification condition is ensured by the injectivity of e_0. In the SPO approach, instead, the embedding is always possible since SPO derivations do not need any application conditions.

There are other interesting applications for derived productions and derivations. First they can be used to shortcut derivation sequences. Thereby they reduce the number of intermediate graphs to be constructed and fix the interaction of the productions applied in the sequence. Similar to a database transaction we obtain an "all-or-nothing" semantics: The single actions described by the direct derivations are either performed together (the derived production is applied) or they leave the current state unchanged (the derived production is not applicable). Due to the absence of intermediate states the transaction may not be interrupted by other actions.

In parallel and concurrent systems intermediate states correspond to synchronization points. A parallel derivation sequence, as introduced in Section 2.2, may be seen as a synchronous computation: Several independent processors perform a direct derivation each and wait for the next clock tick to establish a global intermediate state. Then they proceed with the next application. The derived production forgets about these synchronization points. Therefore it may perform even sequentially dependent derivations in a single step.

The representation of derivations by derived productions, however, is not at all minimal. Given a derivation δ, each derivation ρ that can be embedded into δ defines a derived production which can simulate the effect of ρ. If we look for a more compact representation, we can take all those derived productions corresponding to derivations ρ which are minimal w.r.t. the embedding relation. We speak of *minimal derived productions.* Intuitively, the minimal derived production of a derivation ρ does not contain any context but is constructed from the elements of the productions of ρ, only.

2.4 Amalgamation and Distribution

According to Subsection 2.2, truly parallel computation steps can be modeled by the application of a parallel production, constructed as the disjoint union of two elementary productions. This provides us with a notion of parallel composition of productions and derivations. In many specification and programming languages for concurrent and distributed systems, however, such a parallel composition is equipped with some synchronization mechanism in order to allow for cooperation of the actions that are put in parallel. If two graph productions shall not work concurrently but cooperatively, they have to synchronize their applications w.r.t. commonly accessed elements in the graph: A common subproduction specifies the shared effect of the two productions. This general idea is used in this subsection in two different ways. First we consider systems with global states, represented as graphs in the usual way. In this setting the synchronized application of two subproduction-related productions may be described by the application of a so-called amalgamated production, i.e., the gluing of the elementary productions w.r.t. the common subproduction. In the second part we consider distributed states, where each production is applied to one local state, such that synchronized local derivations define a distributed derivation.

Amalgamation

The synchronization of two productions p_1 and p_2 is expressed by a *subproduction* p_0 which is somehow embedded into p_1 and p_2. The *synchronized productions* are then denoted by $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$. They may be glued along p_0, leading to the *amalgamated production* $p_1 \oplus_{p_0} p_2$. A direct derivation $G \overset{p_1 \oplus_{p_0} p_2}{\Longrightarrow} H$ using $p_1 \oplus_{p_0} p_2$ is called *amalgamated derivation*. It may be seen as the simultaneous application of p_1 and p_2 where the common action described by the subproduction p_0 is performed only once. If the subproduction p_0 is empty, the amalgamated production $p_1 \oplus_{p_0} p_2$ specializes to the parallel production $p_1 + p_2$.

Distribution

Like an amalgamated production is somehow distributed over two synchronized productions, a graph representing a certain system state can be splitted into local graphs, too. A *distributed graph* $DG = (G_1 \overset{g_1}{\leftarrow} G_0 \overset{g_2}{\rightarrow} G_2)$ consists of two *local graphs* G_1 and G_2 that share a common *interface graph* G_0, embedded into G_1 and G_2 by graph morphisms g_1 and g_2, respectively. Gluing G_1 and G_2 along G_0 yields the global graph $\oplus DG = G_1 \oplus_{G_0} G_2$ of DG. Then DG is also called a *splitting* of $\oplus DG$. The embeddings in^1, in^2 of the interface into the local graphs may be either total or partial; accordingly we call DG a *total* or a *partial splitting*. A partial splitting models a distributed state where the local states have been updated in an inconsistent way, which may result from a loss of synchronization of the local updates. Imagine, for example, a vertex v that is shared by the two local graphs, i.e., which is in G_1, G_2, and in the interface G_0. If this vertex is deleted from the local graph G_1 but not from the interface G_0, the embedding $g_1 : G_0 \rightarrow G_1$ becomes undefined for the vertex v, i.e., the two local states have inconsistent information about their shared substate. Then, the global graph $\oplus DG$ of this partial splitting does not contain the vertex v, but only those elements where the local informations are consistent with each other.

Distributed graphs can be transformed by synchronized productions: Given *local direct derivations* $d_i = (G_i \overset{p_i}{\Longrightarrow} H_i)$ for $i \in \{0, 1, 2\}$ which are compatible with the splitting $G_1 \overset{g_1}{\leftarrow} G_0 \overset{g_2}{\rightarrow} G_2$, d_0 becomes a subderivation of d_1 and d_2. In this case, the *(direct) distributed derivation* $d_1||_{d_0}d_2 : DG \Longrightarrow DH$ transforms the given distributed graph DG into the new one $DH = (H_1 \overset{h_1}{\leftarrow} H_0 \overset{h_2}{\rightarrow} H_2)$. The distributed derivation is *synchronous* if both DG and DH are total splittings.

Now, since synchronized rules can be amalgamated and distributed graphs may be glued together, it seems natural to simulate a distributed derivation by an amalgamated one. Vice versa, assuming a splitting of the global graph, we may want to distribute a given amalgamated derivation. The corresponding conditions, enabling these constructions, are called *amalgamation* and *distribution condition*, respectively.

Problem 5 (Distribution). *Let* $DG = (G_1 \overset{g_1}{\leftarrow} G_0 \overset{g_2}{\rightarrow} G_2)$ *be a distributed graph and* $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ *be synchronized productions.*

1. *Find an* amalgamation condition *such that a distributed derivation* $d_1||_{d_0}d_2 : DG \Longrightarrow DH$ *with* $d_i = (G_i \overset{p_i}{\Longrightarrow} H_i)$ *satisfies this condition iff there is an equivalent amalgamated derivation* $\oplus DG \overset{p_1 \oplus_{p_0} p_2}{\Longrightarrow} \oplus DH$.

$$
\begin{tikzcd}
 & G_0 \arrow[ddl] \arrow[dr] \arrow[rrr, "p_0"] & & & H_0 \arrow[ddl] \arrow[dr] & \\
 & & G_2 \arrow[ddl] \arrow[rrr, "p_2"] & & & H_2 \arrow[ddl] \\
G_1 \arrow[dr] \arrow[rrr, "p_1"] & & & H_1 \arrow[dr] & & \\
 & G \arrow[rrr, "p_1 \oplus_{p_0} p_2"] & & & H &
\end{tikzcd}
$$

2. *Find a* distribution condition *such that an amalgamated derivation* $G \overset{p_1 \oplus_{p_0} p_2}{\Longrightarrow} H$ *satisfies this condition iff there is an equivalent distributed derivation* $d_1||_{d_0}d_2 : DG \Longrightarrow DH$ *with* $G = \oplus DG$, $H = \oplus DH$ *and* $d_i = (G_i \overset{p_i}{\Longrightarrow} H_i)$. □

Simulating a distributed derivation by an amalgamated one means to observe some local activities from a global point of view. Since for asynchronous systems such a global view does not always exist, it is not too surprising that the amalgamation construction in 1. is only defined if the given distributed graph is a total splitting, i.e., represents a consistent distributed state. In order to distribute an amalgamated derivation, the match of the amalgamated production has to be splitted as well. Such a splitting may not exist if the matches of the elementary productions are not in their corresponding local graphs. If the distributed derivation shall be synchronous, we need an additional condition, which can be seen as a distributed dangling condition. It takes care that nothing is deleted in any local component which is in the image of some interface, unless its preimage is deleted as well.

2.5 Further Problems and Results

This subsection is an overview of some further issues, problems and results addressed in the literature of the algebraic approaches.

Semantics

If a graph grammar is considered from the point of view of formal languages, its usual semantics is the class of graphs derivable from the start graph using the productions of the grammar, its *generated language*. Using graph grammars for specification of systems, however, we are not only interested in the generated graphs, corresponding to the states of the system, but also in the derivations, i.e., the systems computations. There are several proposals for semantics of graph grammars that represent graphs and derivations at different levels of abstraction.

- A *model of computation* of a graph grammar is a category having graphs as objects and derivations as arrows. Different models of computations can be defined for a grammar, by imposing different equivalence relations on graphs and derivations. The main problem in this framework is to find an adequate notion of equivalence on graphs and derivations, that provides representation independence as well as abstraction of the order of independent (concurrent) derivation steps. Models of computations for grammars in the DPO are introduced in Section 5.

- Processes are well-accepted as a truly concurrent semantics for Petri nets (see for example [24]), and such a semantics has been defined for both algebraic approaches to graph grammars as well, using slightly different terminologies. In a *graph process* (DPO) or *concurrent derivation* (SPO), the graphs of a derivation are no longer represented explicitly. Instead a so-called *type graph* (DPO) or *core graph* (SPO) is constructed by collecting all nodes and edges of the derivation. Moreover, the linear ordering of the direct derivations is replaced by a partial order representing the causal dependencies among the production applications, or *actions* (SPO). DPO processes and SPO concurrent derivations are introduced in [25] and [26,27], respectively.

- An *event structure* [28] consists of a collection of *events* together with two relations "$\leq$" and "#" modeling *causal dependency* and *mutual exclusion*, respectively. Event structures are widely accepted as abstract semantic domains for systems that exhibit concurrency and non-determinism. Such a truly concurrent semantics is more abstract than the one based on processes, where it is still possible to reconstruct the graphs belonging to a derivation (from the type or core graph), while this is not possible from the event structure. Event structure semantics have been proposed for the double-pushout approach in [29,30,31].

Control

In order to specify systems with a meaningful behavior it is important that the application of productions can be controlled somehow.

- *Application conditions* restrict the applicability of individual productions by describing the admissible matches. In the SPO approach they are considered in some detail in Section 4 of the next chapter.

- Imperative control structures like sequential composition, iteration, nondeterministic choice, etc, lead to the notions of programmed graph transformation and transactions. Such concepts are discussed, for example, in [32], while in the algebraic approaches they have been introduced in [33].

Structuring

Systems usually consists of a number of subsystems, which are specified separately and have some meaning in their own right. Structuring mechanisms for graph transformation systems based on colimits in the *category of graph grammars* are investigated in [34,35], and are shown to be compatible with the functorial semantics introduced in [36].

Analysis

A main advantage of formal specification methods is the possibility of formal reasoning about properties of the specified systems.

- *Consistency conditions* describe properties of all derived graphs. The problem of specifying these conditions and of analyzing them w.r.t. a given grammar has been considered in [37,38] in the SPO approach.

- Since graph grammars generalize term rewriting systems, *rewriting properties* as, for example, confluence, termination, etc., are interesting here as well. Corresponding results for (DPO) hypergraph rewriting can be found in [39,40], while confluence of critical pairs in the SPO-approach has been studied in [41].

More general structures

Different application areas of graph transformation require different notions of graphs. Fortunately, most of the results of the algebraic approaches are obtained on a categorical level, that is, independently of the specific definition of graph. Main parts of the theory have already been generalized to arbitrary categories of structures in high-level replacement (HLR) systems (mainly in the DPO approach [23], see [42] for corresponding concepts in the SPO approach). The SPO approach has been developed for graph structures from the very beginning [43,5], which have recently been extended to generalized graph structures in [44].

3 Graph Transformation Based on the DPO Construction

Following the outline of Section 2.1, in this section we introduce the formal definition of the very basic concepts of the double-pushout approach to graph transformation. This theory has been started in [1]; comprehensive tutorials can be found in [2], [3], [4].

First of all, let us introduce the class of graphs that we shall consider in the rest of this chapter (and also in the next one). These are labeled, directed multigraphs, i.e., graphs where edges and nodes are labelled over two different sets of labels, and between two nodes many parallel edges, even with the same label, are allowed for. A fundamental fact is that graphs and graph morphisms form a category since this allows one to formulate most of the definitions, constructions and results in pure categorical terms. Only seldom some specific properties of the category of graphs are exploited, usually to present the concrete, set-theoretical counterpart of some abstract categorical construction.

Definition 6 (labeled graphs). Given two fixed alphabets Ω_V and Ω_E for node and edge labels, respectively, a *(labeled) graph (over* (Ω_V, Ω_E)*)* is a tuple $G = \langle G_V, G_E, s^G, t^G, lv^G, le^G \rangle$, where G_V is a set of *vertices* (or *nodes*), G_E is a set of *edges* (or *arcs*), $s^G, t^G : G_E \to G_V$ are the *source* and *target* functions, and $lv^G : G_V \to \Omega_V$ and $le^G : G_E \to \Omega_E$ are the *node* and the *edge labeling* functions, respectively.

A *graph morphism* $f : G \to G'$ is a pair $f = \langle f_V : G_V \to G'_V, f_E : G_E \to G'_E \rangle$ of functions which preserve sources, targets, and labels, i.e., which satisfies $f_V \circ t^G = t^{G'} \circ f_E$, $f_V \circ s^G = s^{G'} \circ f_E$, $lv^{G'} \circ f_V = lv^G$, and $le^{G'} \circ f_E = le^G$. A graph morphism f is an *isomorphism* if both f_V and f_E are bijections. If there exists an isomorphism from graph G to graph H, then we write $G \cong H$; moreover, $[G]$ denotes the isomorphism class of G, i.e., $[G] = \{H \mid H \cong G\}$. An *automorphism* of graph G is an isomorphism $\phi : G \to G$; it is *non-trivial* if $\phi \neq id_G$.

The category having labeled graphs as objects and graph morphisms as arrow is called **Graph**. □

Within this chapter we shall often call an isomorphism class of graphs *abstract graph*, like $[G]$; correspondingly, the objects of **Graph** will be called sometimes *concrete* graphs.

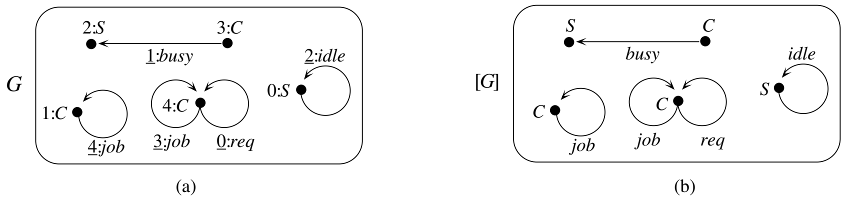


Figure 3: (a) A sample labeled graph and (b) its isomorphism class.

Example 1 (client-server systems). As a running example, we will use a simple graph grammar which models the evolution of client-server systems. The (labeled) graphs are intended to represent possible configurations containing servers and clients (represented by nodes labeled by S and C, respectively), which can be in various states, indicated by edges. A loop on a server labeled by *idle* indicates that the server is ready to accept requests from clients; a loop on a client labeled by *job* means that the client is performing some internal activity, while a loop labeled by *req* means that the client issued a request. An edge from a client to a server labeled by *busy* models the situation where the server is processing a request issued by the client.

In our running example, the alphabets of labels contain only the labels just mentioned, thus $\Omega_V = \{C, S\}$ and $\Omega_E = \{\mathit{idle}, \mathit{job}, \mathit{req}, \mathit{busy}\}$. Figure 3(a) shows the graphical representation of graph $G = \langle G_V, G_E, s^G, t^G, lv^G, le^G \rangle$ with $G_V = \{0, 1, 2, 3, 4\}$, $G_E = \{\underline{0}, \underline{1}, \underline{2}, \underline{3}, \underline{4}\}$. Edges are drawn in the usual way as arrows from the source to the target, and the label of each node or edge is written after its identity, separated by a colon. We shall use natural numbers to denote nodes and underlined numbers to denote edges. Graph G represents a system with three clients and two servers, whose states are specified by the depicted edges.

Figure 3(b) shows the graphical representation of the isomorphism class of graphs $[G]$: it is obtained from G by deleting all natural numbers, i.e., by forgetting the identity of nodes and edges. Their labels are kept because isomorphisms must preserve labels. □

A production in the double-pushout approach is usually defined as a *span*, i.e., a pair of graph morphisms with common source. In the formal definition that follows, a production is defined instead as a structure $p : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$, where l and r are the usual two morphisms, and the additional component p is the production name. The name of a production plays no role when a production is applied to a graph (see Definition 10 of direct derivation below), but it is relevant in certain transformations of derivations (as in the *analysis* construction, Proposition 19) and when relating different derivations (as in Definition 30); in fact, since the name will often be used to encode the construction that yields the production, it allows one to distinguish, for example, between two productions obtained in completely unrelated ways, but that happen to have the same associated span. Thus on the one hand we insist that the name is a relevant part of a productions, but, on the other hand, when applying a production to a graph the name is sometimes disregarded.

Definition 7 (graph productions, graph grammars). A *graph production* $p : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ is composed of a *production name* p and a pair of injective graph morphisms $l : K \rightarrow L$ and $r : K \rightarrow R$. The graphs L, K, and R are called the *left-hand side* (lhs), the *interface*, and the *right-hand side* (rhs) of p, respectively. Two productions $p : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ and $p' : (L' \stackrel{l'}{\leftarrow} K' \stackrel{r'}{\rightarrow} R')$ are *span-isomorphic* if there are three isomorphisms $L \xrightarrow{\phi_L} L'$, $K \xrightarrow{\phi_K} K'$ and $R \xrightarrow{\phi_R} R'$ that make the two resulting squares commutative, i.e., $\phi_L \circ l = l' \circ \phi_K$ and $\phi_R \circ r = r' \circ \phi_K$. If no confusion is possible, we will sometimes make reference to a production $p : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ simply as p, or also as $L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$.

A *graph grammar* $\mathcal{G}$ is a pair $\mathcal{G} = \langle (p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)_{p \in P}, G_0 \rangle$ where the first component is a family of productions indexed by production names in P, and G_0 is the *start graph.* [c] □

Suppose (without loss of generality) that in a production $p : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ morphisms l and r are inclusions. Then, intuitively, the production specifies that all the items in K have to be preserved (in fact, they are both in L and in R); that the items of L which are not in K have to be deleted, and that the items in R which are not in K have to be created.

Example 2 (graph grammar for client-server systems). The graph grammar which models the evolution of client-server systems depicted as labeled graphs (as in Example 1) includes three productions, named *REQ*, *SER*, and *REL*, respectively, which are presented in Figure 4. Production *REQ* models the issuing of a request by a client. After producing the request, the client continues its internal activity (*job*), while the request is served asynchronously; thus a request can be issued at any time, even if other requests are pending or if the client is being served by a server. Production *SER* connects a client that issued a request with an idle server through a *busy* edge, modeling the beginning of the service. Production *REL* (for *release*) disconnect the client from the server (modeling the end of the service), restoring the *idle* state of the server.

[c] Usually the definition of a grammar also includes a set of terminal labels, used to identify the graphs belonging to the generated language. Since we are not directly interested in graph languages, we omit them.

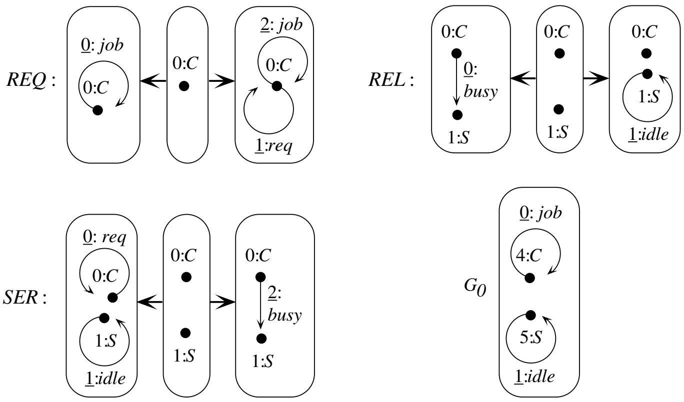


Figure 4: Productions and start graph of the grammar modeling a client-server systems.

$$\begin{tikzcd}
p : (L \arrow[d, "m"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] & R) \arrow[d, "m^*"] \\
p^* : (G & D \arrow[l, "l^*"] \arrow[r, "r^*"'] & H)
\end{tikzcd}$$

Figure 5: Direct derivation as double-pushout construction.

It is worth stressing that the three productions neither delete nor create nodes, which means that the number of clients and servers is invariant. The interface of each production is a subgraph of both the left- and the right-hand sides, thus the morphisms are just inclusions. Note moreover that the natural numbers used as identities of nodes and edges are chosen arbitrarily.

Formally, the grammar is called $\mathcal{C}\text{-}\mathcal{S}$ (for *client-server*), and it is defined as $\mathcal{C}\text{-}\mathcal{S} = \langle\{REQ,\ SER,\ REL\}, G_0\rangle$, where G_0 is the start graph depicted in Figure 4. $\square$

Given a production $p : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ and a graph G, one can try to apply p to G if there is an occurrence of L in G, i.e., a graph morphism, called match, $m : L \rightarrow G$. If p is applicable to match m yielding a graph H, then we obtain a direct derivation $G \stackrel{p,m}{\Longrightarrow} H$. Such a direct derivation is a "double-pushout construction", the construction that characterizes the algebraic approach we are presenting, and it is shown in Figure 5.

Intuitively, regarding graphs as distributed states of a system, a *pushout* is a sort of "generalized union" that specifies how to merge together two states having a common substate [45]. For example, if the right square of Figure 5 is a pushout in **Graph**, then graph H is obtained by gluing together graphs R and D along the common subgraph K, and r^* and m^* are the resulting injections.

Therefore the double-pushout construction can be interpreted as follows. In order to apply the production p to G, we first need to find an occurrence of its left-hand side L in G, i.e., a match $m : L \to G$. Next, we have to delete from G all the (images of) items of L which are not in the interface graph K; in other words, we have to find a graph D and morphisms d and l^* such that the resulting square is a pushout: The context graph D is therefore required to be a *pushout complement object* of $\langle l, m\rangle$. Finally, we have to embed the right-hand side R into D, to model the addition to D of the items to be created, i.e., those that are in R but not in K: This embedding is expressed by the right pushout.

In order to introduce this construction formally we first have to define pushouts and pushout complements. It is worth stressing that the advantage of describing such operation via a categorical construction instead of using standard set-theoretical notions is two-fold. On the one hand, in a set-theoretical framework one needs to handle explicitly the possible clashing of names used in the two states (in the non-shared parts): this is given for free in the categorical construction, at the price of determining the result only up to isomorphism. On the other hand, category theory provides handy techniques, based for example on diagram chasing, for describing and relating complex constructions, that it would be quite cumbersome to handle using standard set theory.

Definition 8 (pushout [20] and pushout complement [2]). Given a category $\mathbf{C}$ and two arrows $b : A \to B$, $c : A \to C$ of $\mathbf{C}$, a triple $\langle D, g : B \to D, f : C \to D\rangle$ as in the diagram below is called a *pushout* of $\langle b, c\rangle$ if

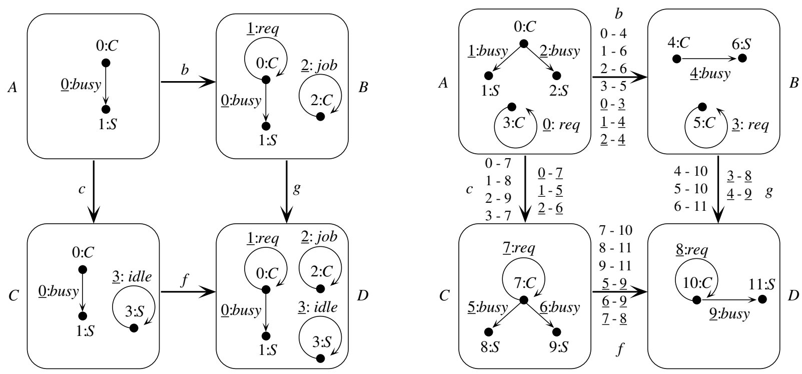
$$\begin{tikzcd}
A \arrow[r, "b"] \arrow[d, "c"] & B \arrow[d, "g"] \arrow[ddr, "g'"] & \\
C \arrow[r, "f"] \arrow[drr, "f'"'] & D \arrow[dr, dotted, "h"] & \\
& & D'
\end{tikzcd}$$

[**Commutativity**] $g \circ b = f \circ c$, and

[**Universal Property**] for all objects D' and arrows $g' : B \to D'$ and $f' : C \to D'$, with $g' \circ b = f' \circ c$, there exists a unique arrow $h : D \to D'$ such that $h \circ g = g'$ and $h \circ f = f'$.

In this situation, D is called a *pushout object* of $\langle b, c\rangle$. Moreover, given arrows $b : A \to B$ and $g : B \to D$, a *pushout complement* of $\langle b, g\rangle$ is a triple $\langle C, c : A \to C, f : C \to D\rangle$ such that $\langle D, g, f\rangle$ is a pushout of $\langle b, c\rangle$. In this case C is called a *pushout complement object* of $\langle b, g\rangle$. □

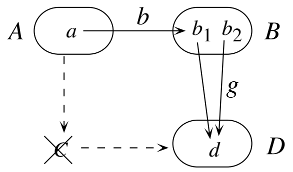
Example 3 (Pushout in **Set** *and in* **Graph***).* The paradigmatic example of category is **Set**, i.e., the category having sets as objects and total functions as arrows. It is easy to show that the pushout of two arrows in **Set** always exists, and that it is characterized (up to isomorphism) as follows. If $b : A \to B$ and $c : A \to C$ are two functions, then the pushout object of $\langle b, c\rangle$ is the set $D = (B + C)_{\approx}$, i.e., the quotient set of the disjoint union of B and C, modulo the equivalence relation "$\approx$", which is the least equivalence relation such that for all $a \in A$, $b(a) \approx c(a)$, together with the two functions $g : B \to D$ and $f : C \to D$ that map each element to its equivalence class.

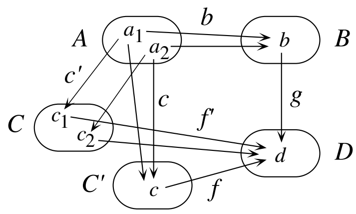


(a) (b)

In category **Graph** the pushout of two arrows always exists as well: It can be computed componentwise (as a pushout in **Set**) for the nodes and for the edges, and the source, target, and labeling mappings are uniquely determined. The above diagrams show two examples. In (a) morphisms b and c are inclusions, thus the pushout object D is obtained by gluing B and C on the common subgraph A. Diagram (b), where all morphisms are specified by listing the pairs *argument-result* (e.g., $b_V(0) = 4$ and $c_E(\underline{1}) = \underline{5}$), shows instead a pushout in **Graph** of two non-injective morphisms b and c. □

In any category, if the pushout object of two arrows exists then it is unique up to a unique isomorphism because of the universal property (see Fact 48 for the corresponding statement for coproducts). On the contrary, since the pushout complement object is not characterized directly by a universal property, for a pair of arrows there may not be any coproduct, or there can exist many pushout complement objects which are not isomorphic. For example, consider the diagrams below in the category of sets and functions. For the left diagram, there exist no set and functions which can close the square making it a pushout, while in the right diagram there exist two pushout complement objects C and C' which are not isomorphic.





The characterization of sufficient conditions for the existence and uniqueness of the pushout complement of two arrows is a central topic in the algebraic theory of graph grammars, because, as we shall see below, in the DPO approach it allows one to check for the applicability of a production to a given match. As shown for example in [2], in category **Graph**, the pushout complement object of two morphisms $\langle b, g\rangle$ exists iff the *gluing condition* is satisfied; moreover, it is unique if b is injective.

Proposition 9 (existence and uniqueness of pushout complements). *Let $b : A \to B$ and $g : B \to D$ be two morphisms in* **Graph**. *Then there exists a pushout complement $\langle C, c : A \to C, f : C \to D\rangle$ of $\langle b, g\rangle$ if and only if the following conditions are satisfied:*

[***Dangling condition***] *No edge $e \in D_E - g_E(B_E)$ is incident to any node in $g_V(B_V - b_V(A_V))$;*

[***Identification condition***] *There is no $x, y \in B_V \cup B_E$ such that $g(x) = g(y)$ and $y \notin b(A_V \cup A_E)$.*

In this case we say that $\langle b, g\rangle$ satisfy the gluing condition *(or g satisfy the gluing condition with respect to b). If moreover morphism b is injective,then the pushout complement is unique up to isomorphism, i.e., if $\langle C, c, f\rangle$ and $\langle C', c', f'\rangle$ are two pushout complements of $\langle b, g\rangle$, then there is an isomorphism $\phi : C \to C'$ such that $\phi \circ c = c'$ and $f' \circ \phi = f$.* □

Example 4. Morphisms $\langle b, g\rangle$ of Figure 6 (a) satisfy the dangling condition, because the only edge incident to node $6 : S$ (which is in $g_V(B_V - b_V(A_V))$) is $\underline{4} : busy$, which is not in $D_E - g_E(B_E)$. They also satisfy the identification condition because the only items identified by g, nodes $2 : C$ and $3 : C$, are in the image of b. Since b is injective, a pushout complement object of $\langle b, g\rangle$ is given by $C = D - g(B - b(A))$, i.e., it is the subgraph of D obtained by

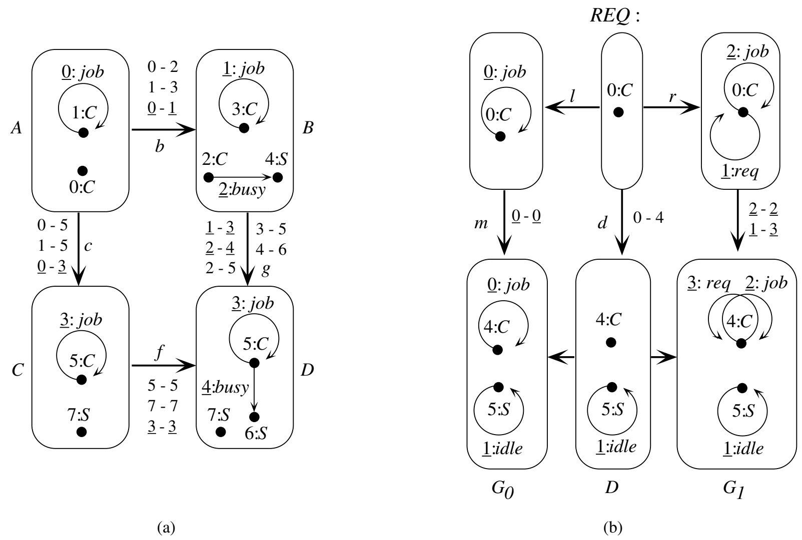


Figure 6: (a) Pushout complement in **Graph**. (b) A direct derivation.

removing all items that are in the image of g but not in the image of $g \circ b$. Morphism $C \xrightarrow{f} D$ is the inclusion, and $A \xrightarrow{c} C$ is the (codomain) restriction of g to C. $\square$

Definition 10 (direct derivation). Given a graph G, a graph production $p : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, and a *match* $m : L \to G$, a *direct derivation from G to H using p (based on m)* exists if and only if the diagram in Figure 5 can be constructed, where both squares are required to be pushouts in **Graph**. In this case, D is called the *context* graph, and we write $G \overset{p,m}{\Longrightarrow} H$, or also $G \overset{p}{\Longrightarrow} H$; only seldom we shall write $G \overset{\langle p,m,d,m^*,l^*,r^* \rangle}{\Longrightarrow} H$, indicating explicitly all the morphisms of the double-pushout. $\square$

Example 5. Figure 6 (b) shows the direct derivation $G_0 \overset{REQ}{\Longrightarrow} G_1$. All horizontal morphisms are inclusions. The vertical morphisms are instead depicted explicitly by listing enough pairs *argument-result* (for example, the morphism from the right-hand side of the production to graph G_1, denoted by $\{\underline{2}\text{-}\underline{2}, \underline{1}\text{-}\underline{3}\}$, maps edges $\underline{2}$ and $\underline{1}$ to edges $\underline{2}$ and $\underline{3}$, respectively, and therefore node 0 to node 4). Morphisms $\langle l, m \rangle$ obviously satisfy the dangling condition because no node is deleted by REQ, and also the identification condition because m is injective. Graph D is the pushout complement object of $\langle l, m \rangle$, and G_1 is the pushout

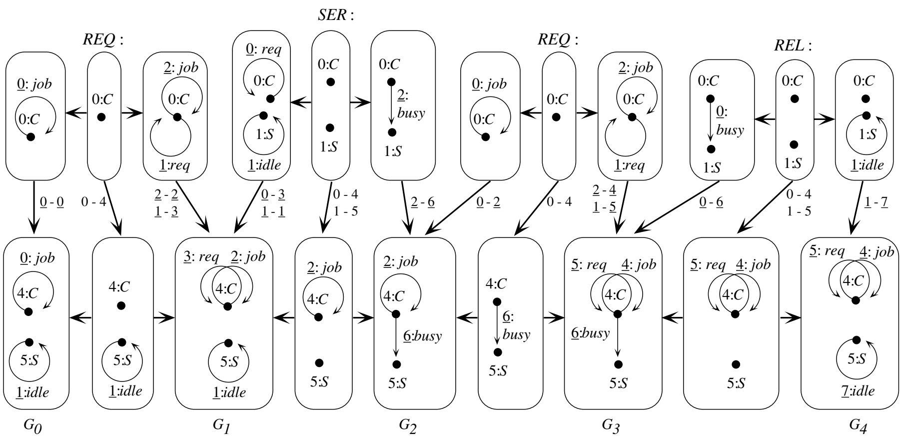


Figure 7: A sequential derivation in grammar $\mathcal{C}$-$\mathcal{S}$ starting from graph G_0.

object of $\langle r, d\rangle$. □

Definition 11 (sequential derivations). A *sequential derivation (over $\mathcal{G}$)* is either a graph G (called an *identity derivation*, and denoted by $G : G \Rightarrow^* G$), or a sequence of direct derivations $\rho = \{G_{i-1} \stackrel{p_i}{\Longrightarrow} G_i\}_{i\in\{1,\ldots,n\}}$ such that p_i is a production of $\mathcal{G}$ for all $i \in \{1, \ldots, n\}$. In the last case, the derivation is written $\rho : G_0 \Rightarrow^*_{\mathcal{G}} G_n$ or simply $\rho : G_0 \Rightarrow^* G_n$. If $\rho : G \Rightarrow^* H$ is a (possibly identity) derivation, then graphs G and H are called the *starting* and the *ending graph* of ρ, and will be denoted by $\sigma(\rho)$ and $\tau(\rho)$, respectively. The *length* of a sequential derivation ρ is the number of direct derivations in ρ, if it is not identity, and 0 otherwise. The *sequential composition* of two derivations ρ and ρ' is defined if and only if $\tau(\rho) = \sigma(\rho')$; in this case it is denoted $\rho\,;\,\rho' : \sigma(\rho) \Rightarrow^* \tau(\rho')$, and it is obtained by identifying $\tau(\rho)$ with $\sigma(\rho')$. □

The identity derivation $G : G \Rightarrow^* G$ is introduced just for technical reasons. By the definition of sequential composition, it follows that for each derivation $\rho : G \Rightarrow^* H$ we have $G\,;\,\rho = \rho = \rho\,;\,H$.

Example 6 (derivation). Figure 7 shows a sequential derivation using grammar $\mathcal{C}$-$\mathcal{S}$ and starting from the start graph G_0. The derivation models the situation where a request is issued by the client, and while it is handled by the server, a new request is issued. □

$$
\begin{tikzcd}
 & & R_1 \arrow[dr, "m_1^*"] & \\
 & K_1 \arrow[ur, "r_1"] \arrow[dl, "l_1"'] \arrow[dr, "d_1"] & & H_1 \\
L_1 \arrow[dr, "m_1"'] \arrow[ddrr, bend left, "k_2"] & & D_1 \arrow[ur, "r_1^*"'] \arrow[dl, "l_1^*"] & \\
 & G & & \\
L_2 \arrow[ur, "m_2"] \arrow[uurr, bend right, "k_1"'] & & D_2 \arrow[ul, "l_2^*"'] \arrow[dr, "r_2^*"] & \\
 & K_2 \arrow[ul, "l_2"] \arrow[ur, "d_2"'] \arrow[dr, "r_2"'] & & H_2 \\
 & & R_2 \arrow[ur, "m_2^*"'] &
\end{tikzcd}
$$

Figure 8: Parallel independence of direct derivations

4 Independence and Parallelism in the DPO approach

According to the overview of Section 2.1 we have to define two notions of independence on direct derivations formalizing the following concepts:

- two alternative direct derivations $H_1 \overset{p_1,m_1}{\Longleftarrow} G \overset{p_2,m_2}{\Longrightarrow} H_2$ are not in conflict (parallel independence), and

- two consecutive direct derivations $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ are not causally dependent (sequential independence).

Intuitively, two alternative direct derivations are *parallel independent* (of each other), if each of them can still be applied after the other one has been performed. This means that neither $G \overset{p_1,m_1}{\Longrightarrow} H_1$ nor $G \overset{p_2,m_2}{\Longrightarrow} H_2$ can delete elements of G which are also needed by the other direct derivation. In other words, the overlapping of the left-hand sides of p_1 and p_2 in G must be included in the intersection of the corresponding interface graphs K_1 and K_2.

Definition 12 (parallel independence). Let $G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ be two direct derivations from the same graph G, as in Figure 8. They are *parallel independent* if $m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$.

This property can be formulated in categorical terms in the following way: There exist two graph morphisms $L_1 \xrightarrow{k_2} D_2$ and $L_2 \xrightarrow{k_1} D_1$ such that $l_2^* \circ k_2 = m_1$ and $l_1^* \circ k_1 = m_2$. □

$$\begin{tikzcd}
L_1 \arrow[d, "m_1"'] & K_1 \arrow[l, "l_1"'] \arrow[r, "r_1"] \arrow[d, "d_1"] & R_1 \arrow[dr, "m_1^*"] \arrow[rrrd, "k_2"] & & L_2 \arrow[dl, "m_2"'] \arrow[llld, "k_1"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "d_2"] & R_2 \arrow[d, "m_2^*"] \\
G & D_1 \arrow[l, "l_1^*"] \arrow[rr, "r_1^*"] & & H_1 & & D_2 \arrow[ll, "l_2^*"'] \arrow[r, "r_2^*"] & H_2
\end{tikzcd}$$

Figure 9: Sequential independent derivation.

Two consecutive direct derivations $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2}{\Longrightarrow} X$ are *sequentially independent* if they may be swapped, i.e., if p_2 can be applied to G, and p_1 to the resulting graph. Therefore p_2 at match m_2 cannot delete anything that has been explicitly preserved by the application of p_1 at match m_1, and, moreover, it cannot use (neither consuming nor preserving it) any element generated by p_1; this implies that the overlapping of R_1 and L_2 in H_1 must be included in the intersection of the interface graphs K_1 and K_2.

Definition 13 (sequential independence). Given a two-step derivation $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2}{\Longrightarrow} H_2$ (as in Figure 9), it is *sequential independent* iff $m_1^*(R_1) \cap m_2(L_2) \subseteq m_1^*(r_1(K_1)) \cap m_2(l_2(K_2))$.

This property can be formulated also in categorical terms in the following way: There exist two graph morphisms $R_1 \xrightarrow{k_2} D_2$ and $L_2 \xrightarrow{k_1} D_1$ such that $l_2^* \circ k_2 = m_1^*$ and $r_1^* \circ k_1 = m_2$. □

Example 7 (sequential independence). Consider the derivation of Figure 7. The first two direct derivations are *sequential dependent*; in fact, the edge $\underline{3} : req$ of graph G_1 is in the image of both the right-hand side of the first production and the left-hand side of the second one, but it is in the context of neither the first nor the second direct derivation: Thus the required morphisms for sequential independence do not exist. On the contrary, in the same figure both the derivation from G_1 to G_3 and that from G_2 to G_4 are sequential independent, and the required morphisms are obvious. □

The following theorem says that these two definitions of parallel and sequential independence indeed solve the Local Church-Rosser Problem of Section 2.2 in the DPO approach.

Theorem 14 (Local Church Rosser).

1. *Let $G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ be two parallel independent direct derivations, as in Figure 8, and let $m_2' = r_1^* \circ k_1$, where arrow $L_2 \xrightarrow{k_1} D_1$ exists by parallel independence. Then morphisms $\langle l_2, m_2' \rangle$ satisfy the gluing condition, and thus production p_2 can be applied to match m_2'. Moreover, derivation $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ is sequential independent.*

2. *Let* $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2}{\Longrightarrow} H_2$ *be a sequential independent derivation as in Figure 9, and let* $m_2' = l_1^* \circ k_1$*, where arrow* $L_2 \xrightarrow{k_1} D_1$ *exists by sequential independence. Then morphisms* $\langle l_2, m_2' \rangle$ *satisfy the gluing condition, and thus production* p_2 *can be applied to match* m_2'*. Moreover, direct derivations* $G \overset{p_1,m_1}{\Longrightarrow} H_1$ *and* $G \overset{p_2,m_2'}{\Longrightarrow} Y$ *are parallel independent.* □

The proof of the Local Church Rosser theorem is postponed, because it follows easily from the proof of the Parallelism Theorem presented later.

The notion of direct derivation introduced in Definition 10 is intrinsically sequential: it models the application of a single production of a grammar $\mathcal{G}$ to a given graph. The categorical framework provides an easy the definition of the parallel application of more than one production to a graph. The following definition of parallel production is slightly more complex than analogous definitions previously presented in the literature, which only used a binary, commutative and associative operator "+" on productions. Such a complexity is justified by the result reported in Appendix A, where it is shown that the coproduct cannot be assumed to be commutative (in a strict way), unless the category is a preorder.

Definition 15 (parallel productions). Given a graph grammar $\mathcal{G}$, a *parallel production (over* $\mathcal{G}$*)* has the form $\langle (p_1, in^1), \ldots, (p_k, in^k) \rangle : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ (see Figure 10), where $k \geq 0$, $p_i : (L_i \overset{l_i}{\leftarrow} K_i \overset{r_i}{\rightarrow} R_i)$ is a production of $\mathcal{G}$ for each $i \in \{1, \ldots, k\}$, L is a coproduct object[d] of the graphs in $\langle L_1, \ldots, L_k \rangle$, and similarly R and K are coproduct objects of $\langle R_1, \ldots, R_k \rangle$ and $\langle K_1, \ldots, K_k \rangle$, respectively. Moreover, l and r are uniquely determined by the families of arrows $\{l_i\}_{i \leq k}$ and $\{r_i\}_{i \leq k}$, respectively (they are defined in a way similar to morphism $f + g$ of Definition 49). Finally, for each $i \in \{1, \ldots, k\}$, in^i denotes the triple of injections $\langle in_L^i : L_i \rightarrow L, in_K^i : K_i \rightarrow K, in_R^i : R_i \rightarrow R \rangle$. Note that all the component productions are recorded in the name of a parallel production.

A parallel production like the one above is *proper* if $k > 1$; the *empty* production is the (only) parallel production with $k = 0$, having the empty graph $\emptyset$ as left- and right-hand sides and as interface. Each production $p : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ of $\mathcal{G}$ will be identified with the parallel production $\langle (p, \langle id_L, id_K, id_R \rangle) \rangle : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$.

[d] Binary coproducts are introduced in Definition 47. The generalization to arbitrary coproducts is straightforward. The coproduct of a list of graphs is given by their disjoint union (the coproduct object) and by all the injections.

$$\begin{tikzcd}
p_1 : L_1 \arrow[d, "in^1_L"'] & K_1 \arrow[l, "l_1"'] \arrow[r, "r_1"] \arrow[drr, "in^1_K"] & R_1 \arrow[drrrr, "in^1_R"] & \cdots & p_k : L_k \arrow[dllll, "in^k_K"'] & K_k \arrow[l, "l_k"'] \arrow[r, "r_k"] \arrow[dll, "in^k_K"'] & R_k \arrow[d, "in^k_R"] \\
L = L_1 + \ldots + L_k & & & K = K_1 + \ldots + K_k \arrow[lll, "l"'] \arrow[rrr, "r"] & & & R = R_1 + \ldots + R_k
\end{tikzcd}$$

Figure 10: The parallel production $\langle (p_1, in^1), \ldots, (p_k, in^k) \rangle : (L \xleftarrow{l} K \xrightarrow{r} R)$.

Two parallel productions over $\mathcal{G}$, $p = \langle (p_1, in^1), \ldots, (p_k, in^k) \rangle : (L \xleftarrow{l} K \xrightarrow{r} R)$ and $q = \langle (q_1, \underline{in}^1), \ldots, (q_{k'}, \underline{in}^{k'}) \rangle : (L' \xleftarrow{l'} K' \xrightarrow{r'} R')$, are *isomorphic via* Π if $k = k'$ and Π is a permutation of $\{1, \ldots, k\}$ such that and $p_i = q_{\Pi(i)}$ for each $i \in \{1, \ldots, k\}$; that is, the component productions of $\mathcal{G}$ are the same, up to a permutation. We say that p and q are *isomorphic* if there is a Π such that they are isomorphic via Π.

The *graph grammar with parallel productions* $\mathcal{G}^+$ generated by a grammar $\mathcal{G}$ has the same start graph of $\mathcal{G}$, and as productions all the parallel productions over $\mathcal{G}$. A *parallel direct derivation over* $\mathcal{G}$ is a direct derivation over $\mathcal{G}^+$; it is *proper* or *empty* if the applied parallel production is proper or empty, respectively. If $G \overset{\emptyset}{\Longrightarrow} H$ is an empty direct derivation, then morphisms l^* and r^* are isomorphisms (see Figure 5); morphism $r^* \circ l^{*-1} : G \to H$ is called the *isomorphism induced* by the empty direct derivation.

A *parallel derivation over* $\mathcal{G}$ is a sequential derivation over $\mathcal{G}^+$. $\square$

It is worth noting that two isomorphic parallel productions are, *a fortiori*, span-isomorphic (see Definition 7).

Fact 16 (isomorphic parallel productions are span-isomorphic). *Let* $\langle (p_1, in^1), \ldots, (p_k, in^k) \rangle : (L \xleftarrow{l} K \xrightarrow{r} R)$ *and* $\langle (q_1, \underline{in}^1), \ldots, (q_k, \underline{in}^k) \rangle : (L' \xleftarrow{l'} K' \xrightarrow{r'} R')$ *be two parallel productions isomorphic via* Π. *Then they are span-isomorphic.*

Proof. By definition, we have that for each $X \in \{L, K, R\}$, $\langle X, in^1_X, \ldots, in^k_X \rangle$ and $\langle X', \underline{in}^1_X, \ldots, \underline{in}^k_X \rangle$ are two coproducts of the same objects, and thus there is a unique isomorphism $X \xrightarrow{\phi_X} X'$ which commutes with the injections (see Fact 48). It can be shown easily that these isomorphisms satisfy $\phi_L \circ l = l' \circ \phi_K$ and $\phi_R \circ r = r' \circ \phi_K$. $\square$

In the rest of the chapter we will assume, without loss of generality, that a "canonical" *choice of coproducts* is given once and for all, i.e., a mapping associating with every pair of graphs $\langle A, B \rangle$ a fixed coproduct $\langle A + B, in_1^{A+B} : A \to A + B, in_2^{A+B} : B \to A + B \rangle$ (see Definitions 47 and 49). Then for each pair of parallel productions $p : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ and $p' : (L' \stackrel{l'}{\leftarrow} K' \stackrel{r'}{\rightarrow} R')$, where $p = \langle (p_1, in^1), \ldots, (p_k, in^k) \rangle$ and $p' = \langle (p'_1, in'^1), \ldots, (p'_{k'}, in'^{k'}) \rangle$ we denote by $p + p'$ the parallel production $\langle (p_1, in \circ in^1), \ldots, (p_k, in \circ in^k), (p'_1, in' \circ in'^1), \ldots, (p'_{k'}, in' \circ in'^{k'}) \rangle : (L + L' \stackrel{l+l'}{\leftarrow} K + K' \stackrel{r+r'}{\rightarrow} R + R')$, where '+' is the coproduct functor induced by the choice of coproducts (Definition 49), and in and in' are triples of canonical injections. Moreover, given a list $\langle p_1, \ldots, p_k \rangle$ of parallel productions of $\mathcal{G}$, by $p_1 + p_2 + \ldots + p_k$ we denote the parallel production $((\ldots(p_1 + p_2) + \ldots) + p_k)$. Note that the $+$ operator on productions defined in this way is in general neither associative nor commutative: such properties would depend on analogous properties of the choice of coproducts. Nevertheless, it is obvious that $+$ is associative and commutative "up to isomorphism", in the sense that, for example, productions $p + p'$ and $p' + p$ are isomorphic. The canonical choice of coproducts is assumed here only to simplify the syntax of parallel productions: it does not imply that coproducts are unique (see also Proposition 52 and Corollary 53).

Example 8 (parallel direct derivation). Figure 11 shows a parallel direct derivation based on the productions of Example 2. The upper part of the diagram shows the parallel production *SER + REQ*. The component productions *SER* and *REQ* and the corresponding inclusions are not depicted explicitly in order to simplify the drawing; anyway, it is easy to recognize in the parallel production the isomorphic copies of *SER* and *REQ*.

The parallel production *SER + REQ* is applied to graph G_1 via a non-injective match. This direct derivation models the situation where a new request is issued by client 4 exactly when the service of an older request is started by server 5. □

Let us consider now Problem 2 of Section 2.2, asking under which conditions a parallel direct derivation can be sequentialized into the application of the component productions, and, viceversa, under which conditions the sequential application of two (or more) productions can be put in parallel. As the following Parallelism Theorem states, such conditions are nothing else than the parallel and sequential independence conditions already introduced. More precisely, it states that the *sequentialization condition* is already part of the gluing condition of the parallel production, that is, each parallel direct derivation may be sequentialized in an arbitrary order leading to sequentially independent derivations. Vice versa, the *parallelization condition* is the sequential independence of the derivation, since consecutive direct derivations can be put in parallel if they are sequentially independent.

Theorem 17 (Parallelism). *Given (possibly parallel) productions* $p_1 : (L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1)$ *and* $p_2 : (L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2)$ *the following statements are*

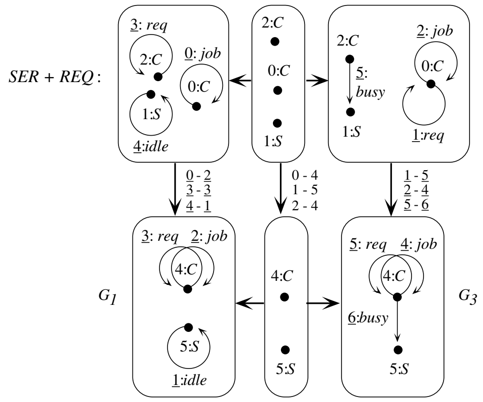


Figure 11: A parallel direct derivation via a non-injective match.

equivalent:

1. *There is a parallel direct derivation* $G \overset{p_1+p_2,m}{\Longrightarrow} X$

2. *There is a sequentially independent derivation* $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$. □

The rest of this section is devoted to the proofs of the Parallelism and of the Local Church Rosser Theorems, and to the presentation of some technical results that will be used in the next section. First of all, let us clarify the relationship between a match of a parallel production and the induced matches of the component productions.

Proposition 18 (applicability of parallel productions). *Let* $q = \langle (p_1, in^1), \ldots, (p_k, in^k) \rangle : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ *be a parallel production (see Definition 15), and let* $L \overset{m}{\longrightarrow} G$ *be a match for it. For each* $i \in \{1, \ldots, k\}$, *let* $L_i \overset{m_i}{\longrightarrow} G$ *be the match of the* i*-th production in* G *induced by* m, *defined as* $m_i = m \circ in^i_L$.

Then $\langle l, m \rangle$ *satisfies the gluing condition, i.e., there is a parallel direct derivation* $G \overset{q,m}{\Longrightarrow} H$, *if and only if for all* $i \in \{1, \ldots, k\}$ $\langle l_i, m_i \rangle$ *satisfies the gluing condition, i.e.,* p_i *can be applied at match* m_i, *say with result* H_i, *and for each* $1 \leq i < j \leq k$, *direct derivations* $G \overset{p_i,m_i}{\Longrightarrow} H_i$ *and* $G \overset{p_j,m_j}{\Longrightarrow} H_j$ *are parallel independent.*

Proof sketch. The parallel independence of the matches of the composing productions follows from the gluing conditions of the parallel direct derivations, and viceversa. □

Therefore there is a very tight relationship between parallel independence and parallel direct derivations: different matches of productions in a graph induce a match of the corresponding parallel production if and only if they are pairwise parallel independent. It is worth stressing here that this property does not hold in the SPO approach, where a parallel production can be applied even if the induced matches are not parallel independent (as discussed in Sections 3.1 and 6 of the next chapter).

The next important results state that every parallel direct derivation can be decomposed in an arbitrary way as the sequential application of the component productions, and, conversely, that every sequential independent derivation can be transformed into a parallel direct derivation. These constructions are in general non-deterministic, and will be used in Section 5 to define suitable relations among derivations. The constructive proofs of Lemmas 19 and 20 are reported in Appendix B (see also [46,23]). It is worth stressing that since we do not assume commutativity of coproducts, the statements of the following lemmas are slightly different from the corresponding statements in the related literature.

Lemma 19 (analysis of parallel direct derivations). *Let $\rho = (G \overset{q}{\Longrightarrow} H)$ be a parallel direct derivation using the parallel production $q = p_1 + \ldots + p_k : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$. Then for each ordered partition $\langle I = \langle i_1, \ldots, i_n\rangle, J = \langle j_1, \ldots, j_m\rangle\rangle$ of $\{1, \ldots, k\}$ (i.e., $I \cup J = \{1, \ldots, k\}$ and $I \cap J = \emptyset$) there is a constructive way to obtain a sequential independent derivation $\rho' = (G \overset{q'}{\Longrightarrow} X \overset{q''}{\Longrightarrow} H)$, called an* analysis *of ρ, where $q' = p_{i_1} + \ldots + p_{i_n}$, and $q'' = p_{j_1} + \ldots + p_{j_m}$. Such a construction is in general not deterministic (for example, graph X is determined only up to isomorphisms). If ρ and ρ' are as above, we shall write $\rho' \in ANAL_{I,J}(\rho)$ (or $\langle \rho', \rho\rangle \in ANAL_{I,J}$).* □

Therefore any parallel direct derivation can be transformed into a sequence of sequential independent direct derivations by repeated applications of the above construction.

Lemma 20 (synthesis of sequential independent derivations). *Let $\rho = (G \overset{q'}{\Longrightarrow} X \overset{q''}{\Longrightarrow} H)$ be a sequential independent derivation. Then there is a constructive way to obtain a parallel direct derivation $\rho' = (G \overset{q'+q''}{\Longrightarrow} H)$, called a* synthesis *of ρ. Also this construction is in general not deterministic. If ρ and ρ' are as above, we shall write $\rho' \in SYNT(\rho)$.* □

Example 9 (analysis and synthesis). Let ρ be the two-steps derivation from graph G_1 to graph G_3 depicted in Figure 7, and let ρ' be the parallel direct derivation of Figure 11. Since ρ is sequential independent, the synthesis construction can be applied to it, and one possible result is indeed ρ'; thus we have $\rho' \in SYNT(\rho)$. Conversely, the application of the analysis construction to ρ' can produce ρ, and we have $\rho \in ANAL_{\langle 1\rangle,\langle 2\rangle}(\rho')$. Also, through a different analysis of ρ' we may obtain the derivation $\rho_1 ; \rho_2$ of Figure 15 (a): In fact, we have $\rho_1 ; \rho_2 \in ANAL_{\langle 2\rangle,\langle 1\rangle}(\rho')$. □

It is worth stressing that the analysis and synthesis constructions are not inverse to each other in a strict sense, but they are so up to isomorphism. Isomorphisms of (direct) derivations and other equivalences among derivations will be discussed in depth in the next section.

We are now ready to present the outline of the proofs of the Parallelism and of the Local Church Rosser Theorems

Proof sketch of Theorem 17 (Parallelism). Lemma 19 provides a constructive proof that $1 \Rightarrow 2$, and, similarly, Lemma 20 provides a constructive proof that $2 \Rightarrow 1$. □

Proof sketch of Theorem 14 (Local Church Rosser). $G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ are two parallel independent direct derivations, iff (by Proposition 18) there is a graph H such that $G \overset{p_1+p_2,[m_1,m_2]}{\Longrightarrow} H$ is a parallel direct derivation, iff (by Theorem 17) there is a sequentially independent derivation $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$. □

5 Models of Computation in the DPO Approach

In the previous sections we considered mainly definitions and properties (for the DPO approach) of *direct* derivations, i.e., of the basic graph rewriting steps. Since the derivations of a grammar are intended to model the computations of the system modeled by the grammar, it is certainly worth studying the operational behaviour of a grammar by developing a formal framework where its derivations can be described and analyzed. A *model of computation* for a grammar is a mathematical structure which contains relevant information concerning the potential behaviour of the grammar, that is, about all the possible computations (derivations) of the grammar. Since the basic operation on derivations is concatenation, it is quite obvious that *categories* are very suited as mathematical structures for this use. In fact, a model of computation for a given grammar is just a category having graphs as objects, and where every arrow is a derivation starting from the source graph and ending at the target graph. Then the categorical operation of sequential composition of arrows corresponds to the concatenation of derivations. Models of computation of this kind (i.e., categories of computations, possibly equipped with some additional algebraic structure) have been proposed (with various names) for many formalisms, like Phrase Structure Grammars [47], Petri nets [48,49], CCS [50], Logic Programming [51,52], and others. Although in this section we focus on the DPO approach, it is obvious that models of computation can be defined easily for the SPO approach as well, and also for all other approaches to graph transformation.

A *concrete* model of computation for a grammar is defined easily according to the above intuition: it has all concrete graphs as objects and all derivations as arrows. However, such a model often contains a lot of redundant information; in fact one is usually interested in observing the behaviour of a grammar from a more abstract perspective, ignoring some details which are considered as unessential. Consequently, more abstract models of computation are often of interest, where derivations (and graphs) differing only for insignificant aspects are identified. In general such abstract models can be defined by imposing an equivalence relation on derivations (and graphs) of the most concrete model: Such relation equates exactly those derivations (graphs) which cannot be distinguished by the chosen observation criterion. A very natural requirement for such an equivalence is that it is a congruence with respect to sequential composition. If such a condition is satisfied, one gets automatically the *abstract* model of computation corresponding to the chosen observation by taking the quotient category of the concrete model with respect to the equivalence relation.

In the following subsections, besides the concrete model for a graph grammar in the DPO approach, we consider two different observation mechanisms on graph derivations, and we define the corresponding abstract models. Such observation mechanisms are intended to capture *true concurrency* and *representation independence*, respectively. The equivalence we use for capturing true concurrency is a slight variation of the well-known *shift equivalence* [46,3], and it is presented in Section 5.1. On the other hand, the most natural equivalence on graph derivations intended to capture representation independence (i.e., the equivalence equating pairs of isomorphic derivations) fails to be a congruence with respect to sequential composition. Therefore it has to be refined carefully through the use of the so-called *standard isomorphisms* (first introduced in [53]). This will be the topic of Sections 5.2 and 5.3. The full proofs of the main results of Section 5.3 are given in Appendix B.

Besides the models corresponding to these two equivalences, we shall introduce in Section 5.4 a third abstract model, called the *abstract, truly-concurrent model*, which satisfactorily composes the two equivalences, and we will relate it to the other models through suitable functors. In our view, such a model represents the behaviour of a graph grammar at a very reasonable level of abstraction, at which many classical results of the algebraic theory of grammars can be lifted. This model is used for example in [30] as a starting point for the definition of a truly-concurrent semantics for graph grammars based on Event

Structures [28]. The present section is based mainly on the paper [54].

5.1 The Concrete and Truly Concurrent Models of Computation

As suggested above, the simplest, more concrete model of computation is just a category having graphs as objects and derivations as arrows. Thus the next definition just emphasizes the obvious fact that since derivations can be composed, they can be regarded as arrows of a category. All along this section, by derivation we mean a possibly parallel derivation.

Definition 21 (the concrete model of computation). Given a graph grammar $\mathcal{G}$, the *concrete model of computation for* $\mathcal{G}$, denoted $\mathbf{Der}(\mathcal{G})$, is the category having all graphs as objects (thus it has the same objects as category **Graph**), and where $\rho : G \to H$ is an arrow iff ρ is a (parallel) derivation, $\sigma(\rho) = G$ and $\tau(\rho) = H$ (see Definitions 11 and 15). Arrow composition is defined as the sequential composition of derivations, and the identity of object G is the identity derivation $G : G \Rightarrow^* G$ (see Definition 11). □

The following example shows why, in our view, the concrete model of computation for a grammar contains too much information.

Example 10 (concrete model of computation for grammar $\mathcal{C}$-$\mathcal{S}$). Let us have a look at the structure of category $\mathbf{Der}(\mathcal{C}\text{-}\mathcal{S})$. As objects, it contains all labeled graphs introduced in Example 1. Thus, for example, for each non-empty graph G it contains infinitely many distinct objects isomorphic to G.

As arrows are concerned, consider for example the graphs G_1 and G_3 of Figure 7. Between these two graphs we have one arrow for the two-step derivation of Figure 7 which applies first SER and then REQ, and infinitely many additional distinct arrows, one for each isomorphic copy of that derivation, which can be obtained by changing arbitrarily the identity of nodes and edges in all the graphs of the derivation, except the starting and the ending ones. Moreover, between G_1 and G_3 there are infinitely many distinct arrows representing the parallel direct derivation of Figure 11 and all its isomorphic copies. There are also infinitely many arrows having G_1 both as source and as target: among them there is the identity derivation $G_1 : G_1 \Rightarrow^* G_1$, and some of the empty direct derivations mentioned at the end of Definition 15. □

Therefore many arrows of the concrete model should be considered as not distinguishable, when looking at derivations from a more abstract perspective. Postponing to the next section the discussion about how isomorphic derivations can be identified, we introduce here an equivalence intended to capture true concurrency, yielding the *truly-concurrent model of computation*.

According to the Parallelism Theorem (Theorem 17), two productions can be applied at parallel independent matches in a graph either at the same time, or one after the other in any order, producing the same resulting graph. An observation mechanism which does not distinguish two derivations where independent rewriting steps are performed in different order is considered to observe the *concurrent* behaviour of a grammar, instead of the *sequential* one. The corresponding equivalence is said to capture "true concurrency". Such an equivalence has been deeply studied in the literature, where it is called *shift equivalence* [46,2,55,56], and it is based on the analysis and synthesis constructions (Lemmas 19 and 20): It equates two derivations if they are related by a finite number of applications of analysis and synthesis. The name "shift equivalence" refers to the basic operation of "shifting" a production one step towards the left, if it is sequential independent from the previous direct derivation: In fact, Proposition 25 below will show that the analysis and synthesis constructions can be simulated by repeated shifts.

Definition 22 (shift equivalence). If ρ and ρ' are two derivations such that $\rho' \in ANAL_{I,J}(\rho)$ (see Lemma 19), $\rho_1 = \rho_2 \,;\, \rho \,;\, \rho_3$, $\rho_1' = \rho_2 \,;\, \rho' \,;\, \rho_3$, and ρ_2 has length $n-1$, then we will write $\rho_1' \in ANAL^n_{I,J}(\rho_1)$, indicating that ρ_1' is an analysis of ρ_1 at step n. Similarly, if $\rho' \in SYNT(\rho)$ (see Lemma 20), $\rho_1 = \rho_2 \,;\, \rho \,;\, \rho_3$, $\rho_1' = \rho_2 \,;\, \rho' \,;\, \rho_3$ and ρ_2 has length $n-1$, then we will write $\rho_1' \in SYNT^n(\rho_1)$, indicating that ρ_1' is obtained from ρ_1 via a synthesis construction at step n.

Now, let $ANAL$ be the union of all analysis relations (i.e., $ANAL = \bigcup\{ANAL^n_{I,J} \mid n \in \mathbb{N} \wedge I, J \in \mathbb{N}^*\}$), and similarly let $SYNT$ be the union of all synthesis relations ($SYNT = \bigcup\{SYNT^n \mid n \in \mathbb{N}\}$). Furthermore, let $SHIFT^0_0$ be the relation defined as $\langle \rho_1, \rho_2 \rangle \in SHIFT^0_0$ iff ρ_1 is an identity derivation $G : G \Rightarrow^* G$ and $\rho_2 : G \overset{\emptyset}{\Longrightarrow} G$ is an empty direct derivation such that the induced isomorphism is the identity $id_G : G \to G$ (see Definition 15).

The smallest equivalence relation on derivations containing $ANAL \cup SYNT \cup SHIFT^0_0$ is called *shift equivalence* and it is denoted by $\equiv_{sh}$. If ρ is a derivation, by $[\rho]_{sh}$ we denote the equivalence class containing all derivations shift-equivalent to ρ. □

From the last definition it follows immediately that $\rho \equiv_{sh} \rho'$ implies $\sigma(\rho) = \sigma(\rho')$ and $\tau(\rho) = \tau(\rho')$, because synthesis and analysis do not affect the extremes of a derivation. The truly-concurrent model of computation for a graph grammar is, like the concrete model of Definition 21, a category having concrete graphs as objects; however, its arrows are equivalence classes of derivations modulo the shift equivalence. There is an obvious functor relating the concrete model and the truly-concurrent one.

Definition 23 (the truly-concurrent model of computation). Given a graph grammar $\mathcal{G}$, the *truly-concurrent model of computation for* $\mathcal{G}$, denoted

$\mathbf{Der}(\mathcal{G})/_{sh}$, is the category having graphs as objects, and as arrows equivalence classes of derivations with respect to the shift equivalence. More precisely, $[\rho]_{sh} : G \rightarrow H$ is an arrow of $\mathbf{Der}(\mathcal{G})/_{sh}$ iff $\sigma(\rho) = G$ and $\tau(\rho) = H$. The composition of arrows is defined as $[\rho]_{sh} ; [\rho']_{sh} = [\rho ; \rho']_{sh}$, and the identity of G is the equivalence class $[G]_{sh}$, containing the identity derivation G.

We denote by $[\_]_{sh}$ the obvious functor from $\mathbf{Der}(\mathcal{G})$ to $\mathbf{Der}(\mathcal{G})/_{sh}$ which is the identity on objects, and maps each concrete derivation to its shift equivalence class. □

Example 11 (truly-concurrent model of computation for grammar $\mathcal{C}$-$\mathcal{S}$).
Consider the concrete model for $\mathcal{C}$-$\mathcal{S}$ described in Example 10. Category $\mathbf{Der}(\mathcal{C}\text{-}\mathcal{S})/_{sh}$ has the same objects but much fewer arrows. For example, since the two-steps derivation $G_1 \Rightarrow_{SER} G_2 \Rightarrow_{REQ} G_3$ of Figure 11 and the parallel direct derivation $G_1 \Rightarrow_{SER+REQ} G_3$ of Figure 11 are shift-equivalent, they are represented by the same arrow between graphs G_1 and G_3; moreover, since the analysis and synthesis constructions are non-deterministic, it turns out that also all the isomorphic copies of the mentioned derivations are shift-equivalent to them. Therefore there is only one arrow from G_1 to G_3 representing the application of productions *SER* and *REQ* in any order, as one would expect in an abstract model.

However, since the objects of the category are concrete graphs, looking at the arrows starting from G_1 one may still find infinitely many arrows corresponding to the application of the same productions *SER* and *REQ*, but ending at different isomorphic copies of G_3. This fact shows that the truly-concurrent model still contains too many arrows, as it may manifests *unbounded non-determinism*, like in the case just described. □

Since arrows of the truly-concurrent model are equivalence classes of derivations, it is natural to ask if they contain a "standard representative", i.e., a derivation which can be characterized as the only one in the equivalence class which enjoys a certain property. Canonical derivations provide a partial answer to this question. In the thesis of Hans-Jörg Kreowski ([46]) it is shown that any graph derivation can be transformed into a shift-equivalent *canonical* derivation, where each production is applied as early as possible. In general, since the analysis and synthesis constructions are not deterministic, from a given derivation many distinct canonical derivations can be obtained: nevertheless, all of them are isomorphic. This is formalized by a result in [46] stating that every derivation has, up to isomorphism, a unique shift-equivalent canonical derivation.

Given a derivation, a shift-equivalent canonical derivation can be obtained by repeatedly "shifting" the application of the individual productions as far as possible towards the beginning of the derivation. It is worth stressing that, unlike the original definition, we explicitly consider here the deletion of empty direct derivations.

Definition 24 (shift relation, canonical derivations). Let us write $\rho' \in ANAL_i^n(\rho)$ if $\rho' \in ANAL_{I,J}^n(\rho)$, where $I = \langle i \rangle$ and $J = \langle 1, \ldots, i-1, i+1, \ldots, k \rangle$; i.e., when ρ' is obtained from derivation ρ by anticipating the application of the i-th production of the n-th direct derivation.

Now, for each pair of natural numbers $\langle n, j \rangle$, both greater than 0, the relation $SHIFT_j^n$ is defined as $SHIFT_j^n = SYNT^{n-1} \circ ANAL_j^n$ ($\circ$ is the standard composition of relations). Moreover, for each $n > 0$, we write $\langle \rho_1, \rho_2 \rangle \in SHIFT_0^n$ iff ρ_1 has length n, $\rho_1 = \rho\,; G \stackrel{q}{\Longrightarrow} H$, $\rho_2 = \rho\,; G \stackrel{q}{\Longrightarrow} X \stackrel{\emptyset}{\Longrightarrow} H$, and the double pushout $G \stackrel{q}{\Longrightarrow} H$ is obtained from $G \stackrel{q}{\Longrightarrow} X$ by composing the right-hand side pushout with the isomorphism from X to H induced by the empty direct derivation $X \stackrel{\emptyset}{\Longrightarrow} H$. Finally, let $SHIFT_0^0$ be as in Definition 22.

Relation $<_{sh}$ is defined as the union of relations $SHIFT_j^n$ for each n, $j \in \mathbb{N}$. The transitive and reflexive closure of $<_{sh}$, denoted $\leq_{sh}$, is called the *shift relation*. A derivation is *canonical* iff it is minimal with respect to $\leq_{sh}$. □

Thus we have that $\rho_1 <_{sh} \rho_2$ iff ρ_1 is obtained from ρ_2 either by removing the last direct derivation if it is empty (if ρ_1 is the identity, then an additional condition must be satisfied), or by moving the application of a single production one step towards left. It is worth stressing here that for each $n \in \mathbb{N}$, relation $SHIFT_0^n$ is deterministic, in the sense that $\langle \rho_1, \rho_2 \rangle, \langle \rho_1', \rho_2 \rangle \in SHIFT_0^n$ implies $\rho_1 = \rho_1'$.

Example 12 (canonical derivation). Let δ_1 be the derivation depicted in Figure 7. Figures 12 and 13 show two derivations, δ_2 and δ_4, both shift-equivalent to δ_1, and where δ_4 is canonical.

Let us explain how δ_4 can be obtained from δ_1 through repeated applications of the shift construction. First, note that we have $\langle \delta_2, \delta_1 \rangle \in SHIFT_1^3$, i.e., δ_2 is obtained from δ_1 by moving the (second) application of REQ one step towards left. More formally, by Definition 24 we have $SHIFT_1^3 = SYNT^2 \circ ANAL_1^3$; thus δ_2 is obtained applying first an analysis construction to direct derivation $G_2 \Rightarrow_{REQ} G_3$, obtaining the two-step derivation $G_2 \Rightarrow_{REQ} G_3 \Rightarrow_{\emptyset} G_3$, and then by applying the synthesis construction to the sequential independent derivation $G_1 \Rightarrow_{SER} G_2 \Rightarrow_{REQ} G_3$, obtaining the direct derivation $G_1 \Rightarrow_{SER+REL} G_3$. In particular, note that the application of the empty derivation in δ_2 is generated by the analysis construction (see the corresponding proof in Appendix B).

Next, we can apply the shift construction to the last part of δ_2, because the derivation $G_3 \Rightarrow_{\emptyset} G_3 \Rightarrow_{REL} G_4$ is clearly sequential independent. In this way we get a new derivation $\delta_3 \in SHIFT_1^4(\delta_2)$, which is obtained by moving

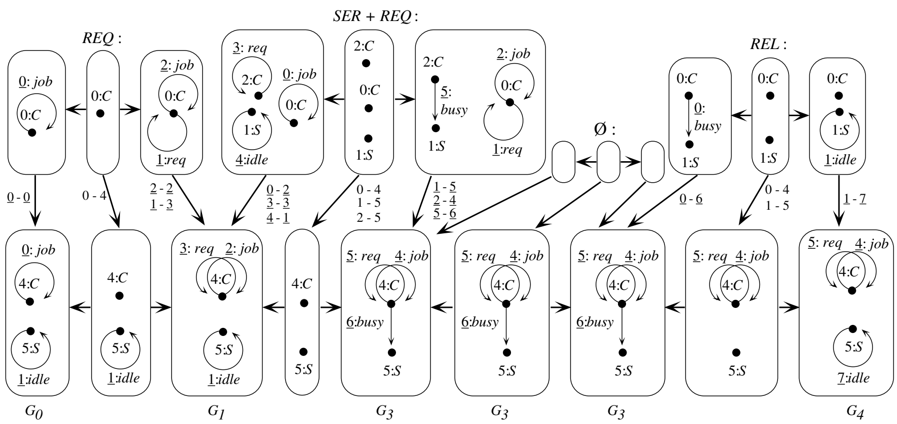


Figure 12: Derivation δ_2, shift-equivalent to derivation δ_1 of Figure 7. More precisely, $\langle \delta_2, \delta_1 \rangle \in SHIFT_1^3$.

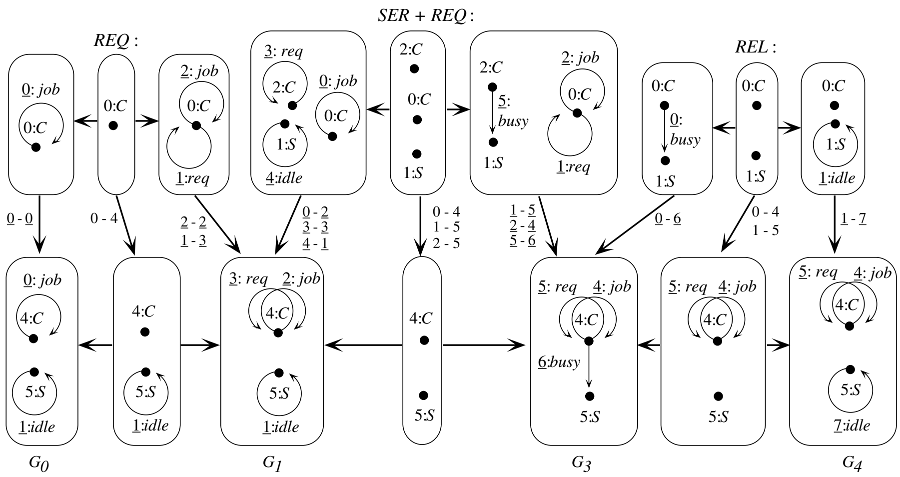


Figure 13: The canonical derivation δ_4, shift-equivalent to derivation δ_1 of Figure 7.

one step earlier the application of REL; derivation δ_3 is not depicted, but it is obtained easily by adding an empty direct derivation $G_4 \Rightarrow_\emptyset G_4$ to δ_4. Now, since the last direct derivation of δ_3 is empty, by removing it we get the last derivation $\delta_4 \in SHIFT_0^3(\delta_3)$. Finally, the fact that δ_4 is canonical can be shown easily by checking that every production application in it deletes at least one item (edge) generated by the previous direct derivation. Thus the shift construction cannot be applied anymore. $\square$

The next proposition on the one hand justifies the name of the equivalence introduced in Definition 22, on the other hand it guarantees the existence of a shift-equivalent canonical derivation for any given derivation. This fact will be used in the next section.

Proposition 25 (properties of the shift relation).

1. *The smallest equivalence relation containing the $\leq_{sh}$ relation is exactly relation $\equiv_{sh}$ introduced in Definition 22.*

2. *Relation $\leq_{sh}$ is well-founded, i.e., there is no infinite chain $\rho_1, \rho_2, \ldots$ of derivations such that $\rho_{i+1} <_{sh} \rho_i$ for all $i \in \mathbb{N}$. Thus for each derivation ρ there exists a canonical (i.e., minimal with respect to $\leq_{sh}$) derivation ρ' such that $\rho' \leq_{sh} \rho$.*

Proof sketch. For the first point, denote by $\approx_{sh}$ the smallest equivalence relation containing $\leq_{sh}$. Now $\approx_{sh} \subseteq \equiv_{sh}$ is obvious by the definition. Conversely, one can prove that $\equiv_{sh} \subseteq \approx_{sh}$ by showing that for each $n \in \mathbb{N}$ and $I, J \in \mathbb{N}^*$, $ANAL_{I,J}^n \subseteq \approx_{sh}$ and $SYNT^n \subseteq \approx_{sh}$; in other words, if two derivations are related by a single analysis (or synthesis) construction, then they can be related by a sequence of basic shift constructions. For synthesis, if $\rho' \in SYNT(\rho)$, and $\rho = G \stackrel{q}{\Longrightarrow} X \stackrel{p_1 + \ldots + p_k}{\Longrightarrow} H$, then ρ' can be obtained from ρ by anticipating the production applications in $X \stackrel{p_1 + \ldots + p_k}{\Longrightarrow} H$ one at a time, and deleting the remaining empty direct derivation. For analysis the solution is a bit more tricky, because relation $ANAL_{I,J}$ allows one to reorder all the productions in the parallel direct derivation. This effect can be simulated via shifts by moving individual productions back and forth until they reach the desired position.

For the second point, see Lemma 4.6 and Theorem 5.3 of [46]. The corresponding proofs can be adjusted easily in order to match the slightly different definitions. $\square$

5.2 Requirements for capturing representation independence.

Graph grammars are often introduced (in most of the approaches) as a formalism for specifying the evolution of certain systems. In this perspective, some sort of labeled graphs are used to represent system states, while graph transformation rules encode the basic system transformations as in the running example of this section.

Almost invariably, two isomorphic graphs are considered as representing the *same* system state. In fact, such a state is determined by the topological structure of the graph and by the labels only, while the true identity of nodes and edges is considered just as a representation detail. Therefore, in order to define a model of computation which represents faithfully the behaviour of a system, it is very natural to look for an equivalence which equates all isomorphic graphs. This is even more demanding in the algebraic approaches to graph grammars. In fact, since the pushout object of two arrows is unique only up to isomorphism, from the definition of direct derivation (Definition 10) it follows that given a graph G and a production p, if $G \stackrel{p}{\Longrightarrow} H$, then $G \stackrel{p}{\Longrightarrow} H'$ for each graph H' isomorphic to H. Thus the application of a production to a graph can produce infinitely many distinct results, which implies that both in the concrete and in the truly-concurrent models of computation there are infinitely many arrows starting from a graph and corresponding to distinct applications of the same production. This fact is highly counter-intuitive, because in the above situation one would expect a deterministic result, or, at most, a finite set of possible outcomes.

Indeed, in the classical theory of the algebraic approach to graph grammars, one often reasons (more or less explicitly) in terms of *abstract graphs*, i.e., of isomorphism classes of concrete graphs: With this choice, given as above a graph G and a production p with match m (satisfying the gluing condition), there is only one abstract graph $[H]$ such that $[G] \stackrel{p,m}{\Longrightarrow} [H]$, thus a direct derivation becomes in a sense deterministic.

Such a solution is satisfactory if one is interested just in the set of (abstract) graphs generated by a grammar. However, if one considers instead a kind of semantics which associates with each grammar all the possible derivations (like the models of computation we are interested in), even with abstract graphs one still has too much representation-dependent information. In fact, in the above situation there are still infinitely many distinct direct derivations (i.e., double-pushout diagrams) from $[G]$ to $[H]$ via p and m (as it follows from Definition 10). Like for graphs, one can find many distinct derivations which should not be considered distinct as system evolutions, because they differ just for the identity of nodes and edges in the involved graphs. Therefore it is natural to reason not only in terms of abstract graphs, but also in terms of *abstract derivations*, i.e., suitable equivalence classes of derivations.

We shall say that an equivalence on graphs and graph derivations captures "representation independence" if it equates all isomorphic graphs (and not more), and if it equates derivations which differ for representation details. It is worth stressing here that the shift equivalence (Definition 22), which has been recognized to capture the essentials of the semantics of parallel graph derivations, is clearly not representation independent, because it is able to relate only derivations starting from and ending with the same concrete graphs.

In the following we first formalize the notion of "isomorphism of derivations" (which is the most natural candidate for the equivalence we are looking for), showing that the obvious definition based on the categorical notion of isomorphism of diagrams is not acceptable from a semantical point of view, because it would identify too many derivations. Next we will introduce some reasonable requirements for equivalences on derivations intended to capture representation independence, formalizing them as suitable algebraic properties. More precisely, we will look for an equivalence that is a congruence with respect to sequential composition, and that is compatible with the construction of canonical derivations, i.e., such that any pair of canonical derivations obtained from two equivalent derivations are equivalent as well. It is worth stressing that these requirements are motivated by the goal of defining an abstract, truly-concurrent model of computation for a grammar: If one would like to carry on to the abstract setting other results of the theory (for example those concerning concurrency and amalgamation (see Section 6.1 and [2]), it might be the case that additional requirements should be considered.

Let us start with the definition of isomorphism of derivations.

Definition 26 (isomorphism of derivations). Let $\rho : G_0 \Rightarrow^* G_n$ and $\rho' : G'_0 \Rightarrow^* G'_n$ be two derivations having the same length, as depicted in Figure 14. They are *isomorphic* (written $\rho \equiv_0 \rho'$) if there exists a family of isomorphisms $\{\phi_{G_0} : G_0 \to G'_0, \phi_{X_i} : X_i \to X'_i\}$ with $X \in \{L, K, R, G, D\}$ and $i \in \{1, \ldots, n\}$ between corresponding graphs of the two derivations, such that all those squares commute, which are obtained by composing two corresponding morphisms $X \to Y$ and $X' \to Y'$ of the two derivations and the isomorphisms ϕ_X and ϕ_Y relating the source and the target graphs. □

It is not difficult to see that the equivalence induced by isomorphisms of derivations identifies too many derivations. In fact, as it is clear from the picture, the productions in corresponding direct derivations only need to be span-isomorphic. Thus derivations which can hardly be considered as differing just for representation details may be identified. For example, if a grammar $\mathcal{G}$ contains two span-isomorphic productions p and p', two direct derivations using p and p' would be considered as equivalent. Even worst, two parallel productions $q = p_1 + \ldots + p_k$ and $q' = p'_1 + \ldots + p'_{k'}$ may happen to be span-isomorphic even if some of the involved sequential productions are distinct or if $k \neq k'$, i.e., if the number of applied productions is different.

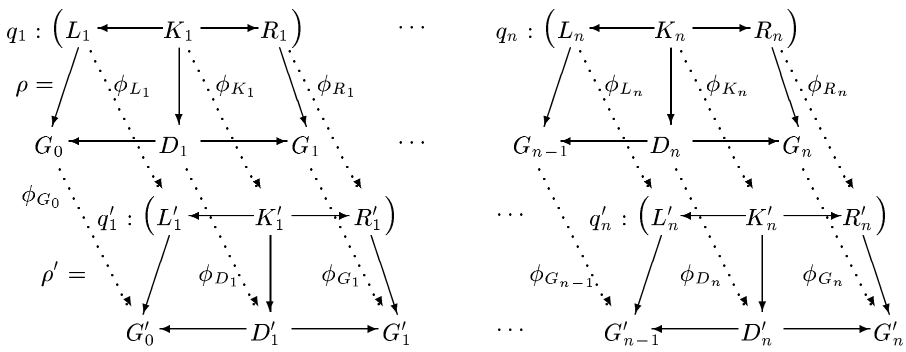
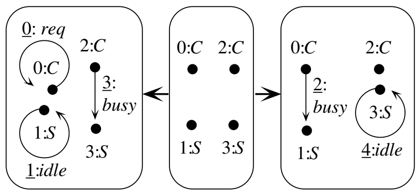


Figure 14: Isomorphism of derivations.

SYNCHR :



Example 13 (extending the graph grammar). Suppose that grammar $\mathcal{C}$-$\mathcal{S}$ is extended with the above production *SYNCHR*, which is intended to model the situation where the service of a request starts exactly when another service terminates.

Such production happens to be span-isomorphic to the parallel production *SER* + *REL*, thus two direct derivations $G \overset{SYNCHR}{\Longrightarrow} H$ and $G \overset{SER+REL}{\Longrightarrow} H$ would be identified by equivalence $\equiv_0$. However, such direct derivations are "semantically" very different: the second one is a parallel direct derivations, and, using the analysis construction, it can be sequentialized in any order, while the other is a sequential direct derivation, which cannot be decomposed. □

In the next section we will propose other equivalences on derivations which require that the productions applied at each direct derivation are not only span-isomorphic, but also isomorphic: this is the main reason why in Definition 15 we defined a production as a pair including both a name and a span.

In the rest of this section, we establish some further requirements for a notion of equivalence on derivations which naturally captures representation independence. Throughout this section $\approx$ denotes an equivalence relation on derivations, and an *abstract* derivation, denoted $[\rho]$, is an equivalence class of derivations modulo $\approx$. The first, obvious requirement is that the notion of abstract derivation is consistent with the notion of abstract graph, i.e., any abstract derivation should start from an abstract graph and should end in an abstract graph.

Definition 27 (well-defined equivalences). For each abstract derivation $[\rho]$, define $\sigma([\rho]) = \{\sigma(\rho') \mid \rho' \in [\rho]\}$, and similarly $\tau([\rho]) = \{\tau(\rho') \mid \rho' \in [\rho]\}$. Then equivalence $\approx$ is *well-defined* if for each derivation $\rho : G \Rightarrow^* H$ we have that $\sigma([\rho]) = [\sigma(\rho)]$, and $\tau([\rho]) = [\tau(\rho)]$. □

If $\approx$ is well-defined, then for each derivation $\rho : G \Rightarrow^* H$ we can write $[\rho] : [G] \Rightarrow^* [H]$. Since our goal is to define abstract models of computation as quotient categories of concrete ones, the second requirement is that the equivalence is a congruence with respect to sequential composition.

Definition 28 (equivalence allowing for sequential composition). Given two abstract derivations $[\rho]$ and $[\rho']$ such that $\tau([\rho]) = \sigma([\rho'])$, their *sequential composition* $[\rho] ; [\rho']$ is defined iff for all $\rho_1, \rho_2 \in [\rho]$ and $\rho'_1, \rho'_2 \in [\rho']$ such that $\tau(\rho_1) = \sigma(\rho'_1)$ and $\tau(\rho_2) = \sigma(\rho'_2)$, one has that $\rho_1 ; \rho'_1 \approx \rho_2 ; \rho'_2$. In this case $[\rho] ; [\rho']$ is, by definition, the abstract derivation $[\rho_1 ; \rho'_1]$.

An equivalence $\approx$ *allows for sequential composition* iff $[\rho] ; [\rho']$ is defined for all $[\rho]$ and $[\rho']$ such that $\tau([\rho]) = \sigma([\rho'])$. □

The third requirement guarantees that we reobtain the result of uniqueness of canonical derivations in the abstract framework, i.e., that each abstract derivation has a unique shift-equivalent abstract canonical derivation.

Definition 29 (uniqueness of canonical derivations). Equivalence $\approx$ *enjoys uniqueness of canonical derivations* iff for each pair of equivalent derivations $\rho \approx \rho'$ and for each pair $\langle \rho_c, \rho'_c \rangle$ of canonical derivations, $\rho \equiv_{sh} \rho_c$ and $\rho' \equiv_{sh} \rho'_c$ implies that $\rho_c \approx \rho'_c$. □

5.3 Towards an equivalence for representation independence.

In this section we introduce two different equivalences on derivations, and analyze their properties with respect to the requirements established in Section 5.2. Actually, since by the definitions all the equivalences are clearly well-defined in the sense of Definition 27, we will focus only on the other requirements. The first equivalence we consider is obtained from equivalence $\equiv_0$ of Definition 26 by requiring that the productions applied at each (parallel) direct derivation are isomorphic and not just span-isomorphic.

Definition 30 (equivalence $\equiv_1$). Let $\rho : G_0 \Rightarrow^* G_n$ and $\rho' : G'_0 \Rightarrow^* G'_n$ be two derivations such that $\rho \equiv_0 \rho'$ as in Figure 14. Then they are 1-*equivalent* (written $\rho \equiv_1 \rho'$) if

1. there exists a family of permutations $\langle \Pi_1, \ldots, \Pi_n \rangle$ such that for each $i \in \{1, \ldots, n\}$, productions q_i and q'_i are isomorphic via Π_i;
2. For each $i \in \{1, \ldots, n\}$, the family of isomorphisms $\{\phi_{L_i} : L_i \to L'_i, \phi_{K_i} : K_i \to K'_i, \phi_{R_i} : R_i \to R'_i\}$ that exists by Definition 26 is exactly the family of isomorphisms between corresponding graphs of q_i and q'_i which is induced by permutation Π_i, according to Fact 16.

We say that *ρ and ρ' are 1-equivalent via $\langle \Pi_i \rangle_{i \leq n}$* if $\langle \Pi_i \rangle_{i \leq n}$ is the family of permutations of point 1 above. Given a derivation ρ, its equivalence class with respect to $\equiv_1$ is denoted by $[\rho]_1$, and is called a 1-*abstract derivation*. □

Let us consider now the algebraic properties of equivalence $\equiv_1$. We prove below that $\equiv_1$ enjoys uniqueness of canonical derivations. This result is based on an important lemma which states that $\equiv_1$ behaves well with respect to analysis, synthesis, and shift of derivations. This is the main technical result of this section, and the whole Appendix B is devoted to its proof.

Lemma 31 (analysis, synthesis and shift preserve equivalence $\equiv_1$). *Let $\rho \equiv_1 \rho'$ via $\langle \Pi_i \rangle_{i \leq n}$, and let ϕ_{G_0} and ϕ_{G_n} be the isomorphisms between their starting and ending graphs (see Figure 14).*

1. *If $\rho_1 \in ANAL^i_j(\rho)$ then there is at least one derivation $\rho_2 \in ANAL^i_{\Pi_i(j)}(\rho')$. Moreover, for each $\rho_2 \in ANAL^i_{\Pi_i(j)}(\rho')$, it holds that $\rho_2 \equiv_1 \rho_1$.*
2. *If $\rho_1 \in SYNT^i(\rho)$ then there is at least one derivation $\rho_2 \in SYNT^i(\rho')$. Moreover, for each $\rho_2 \in SYNT^i(\rho')$, it holds that $\rho_2 \equiv_1 \rho_1$.*
3. *If $\rho_1 \in SHIFT^i_j(\rho)$ then there is at least one derivation $\rho_2 \in SHIFT^i_{\Pi_i(j)}(\rho')$. Moreover, for each $\rho_2 \in SHIFT^i_{\Pi_i(j)}(\rho')$, it holds that $\rho_2 \equiv_1 \rho_1$.*

Furthermore, in all these three cases, the isomorphisms relating the starting and ending graphs of the 1-equivalent derivations ρ_2 and ρ_1 are exactly isomorphisms ϕ_{G_0} and ϕ_{G_n}. □

The observation that analysis, synthesis and shift preserve not only equivalence $\equiv_1$, but also the isomorphisms between the starting and ending graphs of equivalent derivations will be used later in Proposition 36.

Theorem 32 ($\equiv_1$ enjoys uniqueness of canonical derivations). *Let ρ, ρ' be two derivations such that $\rho \equiv_1 \rho'$, and let ρ_c, ρ'_c be two canonical derivations such that $\rho \equiv_{sh} \rho_c$ and $\rho' \equiv_{sh} \rho'_c$. Then $\rho_c \equiv_1 \rho'_c$.*

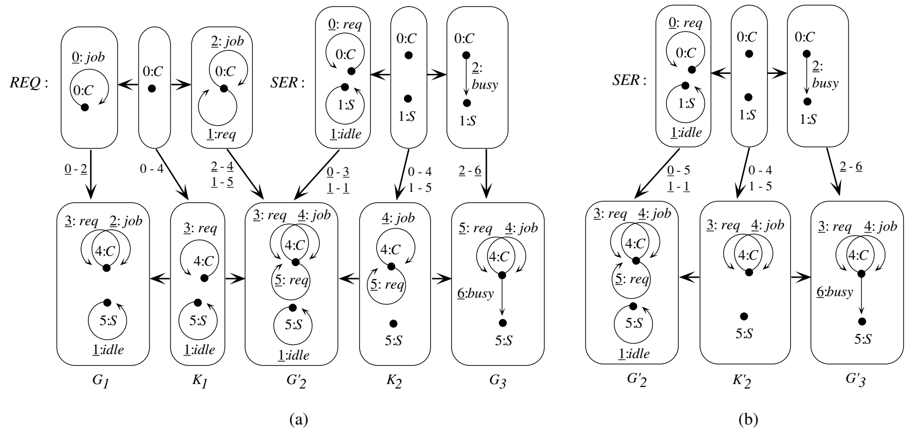


Figure 15: (a) Derivation $\rho_1 \,;\, \rho_2$, and (b) direct derivation ρ'_2. Since $\tau(\rho_1) = G'_2 = \sigma(\rho'_2)$, $\rho_1 \,;\, \rho'_2$ is well defined.

Proof sketch. Let $\sqsubset_{sh}$ be the relation on 1-abstract derivations defined as $[\rho_1]_1 \sqsubset_{sh} [\rho_2]_1$ if $\rho_1 <_{sh} \rho_2$, and let $\sqsubseteq_{sh}$ be its reflexive and transitive closure. Relation $\sqsubset_{sh}$ is clearly well-defined by point 3 of Lemma 31; moreover it is well-founded because so is $<_{sh}$, by Proposition 25.

Furthermore, $\sqsubset_{sh}$ enjoys the following local confluence property: if $[\rho_2]_1 \sqsubset_{sh} [\rho_1]_1$ and $[\rho_3]_1 \sqsubset_{sh} [\rho_1]_1$, then there is a $[\rho_4]_1$ such that $[\rho_4]_1 \sqsubseteq_{sh} [\rho_2]_1$ and $[\rho_4]_1 \sqsubseteq_{sh} [\rho_3]_1$. This can be proved by following the outline of Lemma 4.7 of [46] (where the case of concrete derivations is considered, and all critical pairs of relation $<_{sh}$ are analyzed), and by exploiting Lemma 31 for handling the equivalence classes.

Local confluence and well-foundedness (i.e., termination) of relation $\sqsubset_{sh}$ imply confluence, which implies the statement because canonical derivations are minimal with respect to $<_{sh}$. □

Despite the last result, equivalence $\equiv_1$ is not completely satisfactory, because it does not allow for sequential composition, as the following counterexample shows (a similar counterexample is discussed in [53]).

Example 14 ($\equiv_1$ does not allow for sequential composition). Let $\rho_1 \,;\, \rho_2$ and ρ'_2 be the derivations of Figure 15 (a) and (b), respectively. It is easy to check that $\rho_2 \equiv_1 \rho'_2$: In fact the productions applied are the same, and the family of isomorphisms $\Phi_2 = \{\phi_{G'_2} : G'_2 \to G'_2, \phi_{K_2} : K_2 \to K'_2, \phi_{G_3} : G_3 \to G'_3\}$ with $\phi_{G'_2}(\underline{3}) = \underline{5}$, $\phi_{G'_2}(\underline{5}) = \underline{3}$, $\phi_{K_2}(\underline{5}) = \underline{3}$, $\phi_{G_3}(\underline{5}) = \underline{3}$, and identities elsewhere, makes everything commute. Clearly, we also have $\rho_1 \equiv_1 \rho_1$. using the family

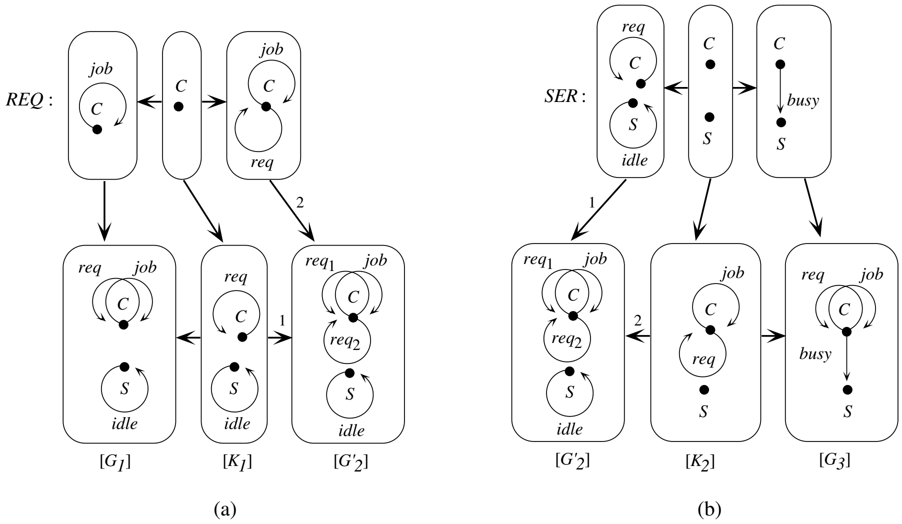


Figure 16: The abstract direct derivations corresponding to ρ_1 and ρ_2 of Figure 11 (a).

of isomorphisms $\Phi_1 = \{id_{G_1}, id_{K_1}, id_{G_2'}\}$. Moreover, $\rho_1 ; \rho_2'$ is defined because $\tau(\rho_1) = G_2' = \sigma(\rho_2')$, but derivations $\rho_1 ; \rho_2$ and $\rho_1 ; \rho_2'$ are not 1-equivalent, because families Φ_1 and Φ_2 do not agree on graph G_2', and actually there is no family of isomorphisms between the corresponding graphs of the two derivations making the resulting diagram commutative.

Note that the two derivations $\rho_1 ; \rho_2$ and $\rho_1 ; \rho_2'$ of Figure 15 are "semantically" different (and thus they should be kept distinct by a reasonable abstract model): in fact, in the first derivation the first request issued by the client is served, while in the second derivation the second request is served.

Figure 16 shows an alternative, equivalent way of presenting the same counterexample. We used a graphical representation of 1-abstract direct derivations, where all the involved graphs are abstract (thus identities of nodes and edges have been dropped). The figure shows abstract derivations $[\rho_1]_1$ (left) and $[\rho_2]_1 = [\rho_2']_1$ (right). In both abstract double-pushouts all morphisms are determined by the labels of nodes and edges, except those having $[G_2']$ as target: Thus we added subscripts to the two edges of $[G_2']$ labeled by *req*, indicating for each morphism which edge is in the codomain. Such subscripts only have a "local scope", in the following sense: if in $[\rho_2]_1$ (or $[\rho_1]_1$) we exchange 1 and 2 in the morphisms having abstract graph $[G_2']$ as target, then the resulting abstract derivation is still $[\rho_2]_1$ (or $[\rho_1]_1$).

Now the fact that the sequential composition of $[\rho_1]_1$ and $[\rho_2]_1$ does not yield a well-defined 1-abstract derivation follows from the observation that there are two possible ways of matching $\tau([\rho_1]_1)$ with $\sigma([\rho_2]_1)$, yielding different results. $\square$

In [53] it is shown that the problem is due to the fact that in the categorical framework many relevant notions cannot be defined "up to isomorphism", not even arrow composition. The solution proposed there will be used in the next notion of equivalence. We first need to introduce "standard isomorphisms".

Definition 33 (standard isomorphisms). A family s of *standard isomorphisms* in category **Graph** is a family of isomorphisms indexed by pairs of isomorphic graphs (i.e., $s = \{s(G, G') \mid G \cong G'\}$), satisfying the following conditions for each G, G' and $G'' \in |\mathbf{Graph}|$:

- $s(G, G') : G \to G'$;
- $s(G, G) = id_G$;
- $s(G'', G') \circ s(G, G'') = s(G, G')$. $\square$

In the equivalence we are going to introduce, denoted $\equiv_3$, we make a limited use of standard isomorphisms: We require that the starting and ending graphs of equivalent derivations are related by such isomorphisms (see also [54]). It is worth recalling that in [54] also another equivalence is introduced, denoted $\equiv_2$, that requires that all the isomorphisms of the form ϕ_{X_i} for $X \in \{G, D\}$ are standard: However, it is shown, through a counter-example, that such equivalence does not enjoy uniqueness of canonical derivations.

Definition 34 (Equivalence $\equiv_3$). Let s be an arbitrary but fixed family of standard isomorphisms of category **Graph**, and let ρ, ρ' be two derivations. We say that ρ and ρ' are 3-*equivalent* (written $\rho \equiv_3 \rho'$) iff they are 1-equivalent, and moreover the isomorphisms ϕ_{G_0} and ϕ_{G_n} relating their starting and ending graphs (see Figure 14) are standard. An equivalence class of derivations with respect to $\equiv_3$ is denoted by $[\rho]_3$, and is called a *3-abstract derivation.* $\square$

Now the fact that equivalence $\equiv_3$ allows for sequential composition follows easily by the above considerations.

Proposition 35 ($\equiv_3$ allows for sequential composition). *Let* ρ_1, ρ_1', ρ_2 *and* ρ_2' *be four derivations such that* $\rho_1 \equiv_3 \rho_1'$, $\rho_2 \equiv_3 \rho_2'$, $\tau(\rho_1) = \sigma(\rho_2)$ *and* $\tau(\rho_1') = \sigma(\rho_2')$. *Then* $\rho_1 ; \rho_2 \equiv_3 \rho_1' ; \rho_2'$.

Proof. By hypothesis there are two families of isomorphisms making ρ_1 and ρ_2 3-equivalent to ρ_1' and ρ_2', respectively. The two families must agree on $\tau(\rho_1) = \sigma(\rho_2)$ (because there is only one standard isomorphism between a given pair of isomorphic graphs), and thus their union provides a family of isomorphisms which makes 3-equivalent $\rho_1 \,;\, \rho_2$ and $\rho'_1 \,;\, \rho'_2$. □

Example 15 (sequential composition of 3-abstract derivations). The counterexample of Example 14 does not apply to equivalence $\equiv_3$: In fact, looking at the direct derivations of Figure 15, we have that $\rho_2 \not\equiv_3 \rho'_2$, because one of the isomorphisms in Φ_2 (namely, $\phi_{G'_2}$) is certainly not standard (it is not the identity, see Definition 33).

Consider now the abstract direct derivations of Figure 16. It can be shown that they are a faithful representation of 3-abstract derivations, provided that we give the subscripts 1 and 2 in graph $[G'_2]$ a "global scope" in the following sense: if we assume that the right abstract derivation is $[\rho_2]_3$, then by exchanging subscripts 1 and 2 in the morphisms having $[G'_2]$ as target, we obtain abstract derivation $[\rho'_2]_3$, which is a different one.[e] Moreover, subscripts can now be used to concatenate abstract derivations: $[\rho_1]_3 \,;\, [\rho_2]_3$ is obtained by matching the two copies of $[G'_2]$ in such a way that subscripts are preserved (because of the uniqueness of standard representatives). □

The fact that the shift relation preserves the isomorphisms relating the starting and ending graphs of two equivalent derivations, as stated explicitly in Lemma 31, guarantees that equivalence $\equiv_3$ enjoys uniqueness of canonical derivations. Therefore equivalence $\equiv_3$ satisfies all the requirements of Section 5.2, as summarized in the next proposition.

Proposition 36 (Properties of equivalence $\equiv_3$). *Equivalence $\equiv_3$ is well-defined, it allows for sequential composition, and it enjoys uniqueness of canonical derivations.*

Proof sketch. For the uniqueness of canonical derivations, the same proof of Theorem 32 can be used. In fact, since the analysis, synthesis and shift relations preserve not only equivalence $\equiv_1$, but also the isomorphisms relating the starting and the ending graphs of 1-equivalent derivations (Lemma 31), it is immediate to show that they preserve equivalence $\equiv_3$ as well. Sequential composition is well-defined by Proposition 35. □

5.4 The abstract models of computation for a grammar.

Since equivalence $\equiv_3$ on derivations is a congruence with respect to sequential composition, and since all canonical derivations which are shift-equivalent to 3-equivalent derivations belong to the same 3-equivalence class, then this equivalence allows one to abstract out from the representation dependent aspects

[e] Subscripts 1 and 2 can be replaced by any other marking able to distinguish the two *req* edges: the relevant fact is that once such a marking is fixed, by permuting the marks one gets a different abstract derivation.

of derivations in a satisfactory way, defining the *abstract model of computation* for a grammar.

Definition 37 (abstract model of computation). Given a grammar $\mathcal{G}$, its *abstract model of computation* $\mathbf{ADer}(\mathcal{G})$ is defined as follows. The objects of $\mathbf{ADer}(\mathcal{G})$ are abstract graphs, i.e., isomorphism classes of objects of **Graph**. Arrows of $\mathbf{ADer}(\mathcal{G})$ are 3-abstract , i.e., equivalence classes of (parallel) derivations of $\mathcal{G}$ with respect to equivalence $\equiv_3$. The source and target mappings are given by σ and τ, respectively; thus $[\rho]_3 : \sigma([\rho]_3) \to \tau([\rho]_3)$. Composition of arrows is given by the sequential composition of the corresponding 3-abstract derivations, and for each object $[G]$ the identity arrow is the 3-abstract derivation $[G : G \Rightarrow^* G]_3$ containing the identity derivation G.

Category $\mathbf{ADer}(\mathcal{G})$ is equipped with an equivalence relation $\equiv_{\underline{sh}}$ on arrows, defined as $[\rho]_3 \equiv_{\underline{sh}} [\rho']_3$ if $\rho \equiv_{sh} \rho'$. □

Proposition 38 (category $\mathbf{ADer}(\mathcal{G})$ is well-defined). *Category* $\mathbf{ADer}(\mathcal{G})$ *is well-defined. Moreover, equivalence* $\equiv_{\underline{sh}}$ *is a congruence with respect to arrow compositions, i.e., if* $[\rho_1]_3 \equiv_{\underline{sh}} [\rho'_1]_3$ *and* $[\rho_2]_3 \equiv_{\underline{sh}} [\rho'_2]_3$*, then* $[\rho_1]_3 ; [\rho_2]_3 \equiv_{\underline{sh}} [\rho'_1]_3 ; [\rho'_2]_3$ *(if the compositions are defined).*

Proof sketch. The two facts follow easily from the properties of equivalence $\equiv_3$ summarized in Proposition 36 and from Theorem 32. □

Since $\equiv_{\underline{sh}}$ is a congruence with respect to arrow composition, it is safe to define the quotient category of $\mathbf{ADer}(\mathcal{G})$ with respect to $\equiv_{\underline{sh}}$: This yields the abstract, truly concurrent model of computation.

Definition 39 (abstract, truly-concurrent model of computation). The *abstract, truly concurrent model of computation* for a grammar $\mathcal{G}$ is defined as category $\mathbf{ADer}(\mathcal{G})/_{\underline{sh}}$, having the same objects as $\mathbf{ADer}(\mathcal{G})$, and as arrows equivalence classes of arrows of $\mathbf{ADer}(\mathcal{G})$ with respect to equivalence $\equiv_{\underline{sh}}$. □

The next definition and result show the relationship among the four models of computations for a grammar, introduced in Definitions 21, 23, 37, and 39, respectively, in terms of functors relating them.

$$
\begin{tikzcd}
\mathbf{Der}(\mathcal{G}) \arrow[r, "[\_]_{sh}"] \arrow[d, "\mathcal{A}"] & \mathbf{Der}(\mathcal{G})/_{sh} \arrow[d, "\underline{\mathcal{A}}"] \\
\mathbf{ADer}(\mathcal{G}) \arrow[r, "[\_]_{\underline{sh}}"] & \mathbf{ADer}(\mathcal{G})/_{\underline{sh}}
\end{tikzcd}
$$

Definition 40 (functors among models of computation). Let $\mathcal{G}$ be a graph grammar. The various models of computation for $\mathcal{G}$ introduced in Definitions 21, 23, 37, and 39, are related by the four functors (as in the above diagram) defined as follows:

- Functor $[\_]_{sh} : \mathbf{Der}(\mathcal{G}) \to \mathbf{Der}(\mathcal{G})/_{sh}$ is the functor introduced in Definition 23 which is the identity on objects, and maps each concrete derivation to its shift equivalence class.
- Functor $[\_]_{\underline{sh}} : \mathbf{ADer}(\mathcal{G}) \to \mathbf{ADer}(\mathcal{G})/_{\underline{sh}}$ is the identity on objects, and maps each 3-abstract derivation to its equivalence class with respect to $\equiv_{\underline{sh}}$.
- Functor $\mathcal{A} : \mathbf{Der}(\mathcal{G}) \to \mathbf{ADer}(\mathcal{G})$ maps each object G to its isomorphism class $[G]$, and each concrete derivation ρ to its 3-abstract equivalence class $[\rho]_3$.
- Functor $\underline{\mathcal{A}} : \mathbf{Der}(\mathcal{G})/_{sh} \to \mathbf{ADer}(\mathcal{G})/_{\underline{sh}}$ maps each object G to its isomorphism class $[G]$, and each arrow $[\rho]_{sh}$ to the $\underline{sh}$-equivalence class of the corresponding 3-abstract derivation, i.e., to $[[\rho]_3]_{\underline{sh}}$. □

Theorem 41 (functorial relationship among models of computation). *All the functors of Definition 40 are well-defined, and the diagram commutes. Moreover, functor $\underline{\mathcal{A}}$ is an equivalence of categories.* □

Proof sketch. Well-definedness is obvious for functors $[\_]_{sh}$, $[\_]_{\underline{sh}}$ and $\mathcal{A}$. For $\underline{\mathcal{A}}$, we have to show that if $\rho \equiv_{sh} \rho'$, then $[\rho]_3 \equiv_{\underline{sh}} [\rho']_3$. This follows directly from the definition of $\equiv_{\underline{sh}}$. Also the commutativity of the diagram is immediate by the definitions.

To prove that $\underline{\mathcal{A}}$ is an equivalence of categories it suffices to show that it is full and faithful, and that each object $[G]$ of $\mathbf{ADer}(\mathcal{G})/_{\underline{sh}}$ is isomorphic to $\underline{\mathcal{A}}(H)$ for some object H of $\mathbf{Der}(\mathcal{G})_{sh}$. This last fact and fullness are ensured by surjectivity of $\underline{\mathcal{A}}$. For faithfulness, let $[\rho]_{sh}$ and $[\rho']_{sh}$ be two parallel arrows of $\mathbf{Der}(\mathcal{G})_{sh}$. We have to show that $\underline{\mathcal{A}}([\rho]_{sh}) = \underline{\mathcal{A}}([\rho']_{sh})$ implies $\rho \equiv_{sh} \rho'$.

By definition of $\underline{\mathcal{A}}$, $\underline{\mathcal{A}}([\rho]_{sh}) = \underline{\mathcal{A}}([\rho']_{sh})$ implies $[\rho]_3 \equiv_{\underline{sh}} [\rho']_3$. Then $\rho \equiv_{sh} \rho'$ follows by the general property that if $\rho \equiv_3 \rho'$, $\sigma(\rho) = \sigma(\rho')$, and $\tau(\rho) = \tau(\rho')$, then $\rho \equiv_{sh} \rho'$ (that is, equivalence $\equiv_3$ is contained in the shift equivalence, if restricted to derivations having the same starting and ending graphs). The last property can be proved by exploiting the fact that the shift equivalence allows one to add empty direct derivations at the end of a derivation, to move them inside the derivation, and to change every intermediate graph of a derivation with an isomorphic copy. □

Let us shortly comment on the relationship among the various models of computation. The concrete model $\mathbf{Der}(\mathcal{G})$ records all the possible derivations of grammar $\mathcal{G}$ without abstracting any detail; its definition is straightforward because the sequential composition of concrete derivations is well-understood, and it is obtained simply by the juxtaposition of the corresponding diagrams. Since the shift equivalence proposed in the literature relates only derivations between the same graphs, it can be used to define a congruence on the concrete derivations yielding the truly concurrent model of computation $\mathbf{Der}(\mathcal{G})/_{sh}$. However, this category is still too concrete, because it has all graphs as objects. Therefore from each graph there are still infinitely many "identical" derivations, differing only for the ending graph. In the other direction, we showed that if one wants to reason on derivations disregarding "representation dependent" details, then equivalence $\equiv_3$ can be used, because it allows for sequential composition; moreover, it is compatible with the shift equivalence.

Finally, thanks to the properties of $\equiv_3$, the abstract, truly concurrent model of computation $\mathbf{ADer}(\mathcal{G})/_{\underline{sh}}$ can be defined. This model seems to represent all the derivations of grammar $\mathcal{G}$ at a very reasonable level of abstraction, both independent of representation-dependent details and consistent with the shift equivalence. The fact that functor $\underline{\mathcal{A}}$ is an equivalence of categories essentially means that category $\mathbf{Der}(\mathcal{G})/_{sh}$ already has between each pair of objects the "right" structure, although this structure has, in general, infinitely many copies.

6 Embedding, Amalgamation and Distribution in the DPO approach

Problems 1 and 2 of Section 2.2 have already been addressed, for the DPO approach, in Section 4. In this section we shortly recall some further concepts and results of the DPO approach concerning the topics addressed in Sections 2.3 and 2.4.

6.1 Embedding of Derivations and Derived Productions

In Section 2.3 the Embedding Problem (Problem 3) asked for conditions ensuring that a derivation can be embedded into a larger context. Given derivations $\rho = (G_0 \stackrel{p_1,m_0}{\Longrightarrow} \cdots \stackrel{p_k,m_{k-1}}{\Longrightarrow} G_k)$ and $\delta = (X_0 \stackrel{p_1,n_0}{\Longrightarrow} \cdots \stackrel{p_k,n_{k-1}}{\Longrightarrow} X_k)$, an *embedding* of ρ into δ is a family of injections $e = (G_i \xrightarrow{e_i} X_i)_{i \in \{1,\ldots,k\}}$ such that for all $i \in \{1,\ldots,k\}$

- $e_i \circ m_i = n_i$, and
- there is an injection $D_i \xrightarrow{c_i} Y_i$ making the two resulting squares commutative (see the diagram below).

The embedding of ρ into δ is denoted by $\rho \xrightarrow{e} \delta$. Given ρ and δ, the embedding e is completely determined by the first injection e_0, called the *embedding morphism* of e.

$$\begin{tikzcd}
& L_1 \arrow[dl, "m_0"'] \arrow[dd] & & K_1 \arrow[ll, "l_1"'] \arrow[rr, "r_1"] \arrow[dl] \arrow[dd] & & R_1 \arrow[dl, "m_0^*"'] \arrow[dd] \\
G_0 \arrow[dr, dotted, "e_0"'] & & D_1 \arrow[ll] \arrow[rr] \arrow[dr, dotted, "c_1"'] & & G_1 \arrow[dr, dotted, "e_1"'] & \\
& X_0 & & Y_1 \arrow[ll] \arrow[rr] & & X_1
\end{tikzcd}
\cdots
\begin{tikzcd}
& L_k \arrow[dl, "m_{k-1}"'] \arrow[dd] & & K_k \arrow[ll, "l_k"'] \arrow[rr, "r_k"] \arrow[dl] \arrow[dd] & & R_k \arrow[dl, "m_{k-1}^*"'] \arrow[dd] \\
G_{k-1} \arrow[dr, dotted, "e_{k-1}"'] & & D_k \arrow[ll] \arrow[rr] \arrow[dr, dotted, "c_k"'] & & G_k \arrow[dr, dotted, "e_k"'] & \\
& X_{k-1} & & Y_k \arrow[ll] \arrow[rr] & & X_k
\end{tikzcd}$$

The applicability of a *derived production* simulating the effect of the derivation ρ shall ensure the existence of the embedding $\rho \xrightarrow{e} \delta$. First we consider the basic case of a direct derivation leading to the notion of directly derived production: Let $d = (G \overset{p,m}{\Longrightarrow} H)$ be a direct derivation. Then $\langle d \rangle : (G \xleftarrow{l^*} D \xrightarrow{r^*} H)$ denotes the *directly derived production* of d, where $G \xleftarrow{l^*} D \xrightarrow{r^*} H$ is the co-production of d, i.e., the bottom row of the double-pushout diagram (1)+(2) on the left of Figure 17. The production name $\langle d \rangle$ records the construction of the directly derived production by the direct derivation d. A direct derivation $X \overset{\langle d \rangle, e}{\Longrightarrow} Z$ using $\langle d \rangle$ is called a *directly derived derivation*. The following equivalence, relating directly derived derivations with derivations using the original production, is also known as vertical composition property of derivation diagrams.

Proposition 42 (Directly Derived Production). *Given the directly derived production $\langle d \rangle$ of a derivation $d = (G \overset{p,m}{\Longrightarrow} H)$ as in Figure 17, the following statements are equivalent:*

1. *There is a directly derived derivation $X \overset{\langle d \rangle, e}{\Longrightarrow} Z$.*

2. *There is a direct derivation $X \overset{p, e \circ m}{\Longrightarrow} Z$.*

Proof sketch. By composition and decomposition of the pushout diagrams (1) to (4) on the left of Figure 17. □

Now we want to extend this equivalence to derivations $\rho = (G_0 \overset{p_1, m_0}{\Longrightarrow} \cdots \overset{p_k, m_{k-1}}{\Longrightarrow} G_k)$ with $k > 1$. To this aim we introduce the notion of "sequential composition" of two productions, in order to obtain a single derived production $\langle \rho \rangle$ from the composition of the sequence of directly derived productions $\langle d_1 \rangle; \langle d_2 \rangle; \ldots; \langle d_k \rangle$. We speak of *sequential composition* because the application of such a composed production has the same effect of a sequential derivation based on the original productions.

$$
\begin{tikzcd}
L & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] & R \arrow[d, "m^*"] \\
G \arrow[from=u, "m"'] \arrow[d, "e"'] & D \arrow[l, "l^*"'] \arrow[r, "r^*"] \arrow[d] & H \arrow[d, "e^*"] \\
X & Y \arrow[l, "l^{**}"'] \arrow[r, "r^{**}"] & Z
\arrow[from=1-1, to=2-2, phantom, "(1)"]
\arrow[from=1-2, to=2-3, phantom, "(2)"]
\arrow[from=2-1, to=3-2, phantom, "(3)"]
\arrow[from=2-2, to=3-3, phantom, "(4)"]
\end{tikzcd}
\qquad
\begin{tikzcd}
 & & & & K \arrow[dll, "l_2'"'] \arrow[drr, "r_1'"] \arrow[dddr] & & & & \\
L_1 \arrow[ddd, "m_1"'] & & K_1 \arrow[ll, "l_1"'] \arrow[drr, "r_1"] \arrow[ddd] & & (5) & & K_2 \arrow[dll, "l_2"'] \arrow[rr, "r_2"] \arrow[ddd] & & R_2 \arrow[ddd, "m_2^*"] \\
 & & & & R_1 = L_2 \arrow[ddd, "m_2"'] & & & & \\
 & & & & & D \arrow[dlll] \arrow[dr] & & & \\
G & & D_1 \arrow[ll, "l_1^*"'] \arrow[drr, "r_1^*"'] & & (6) & & D_2 \arrow[rr, "r_2^*"] \arrow[dll, "l_2^*"] & & H_2 \\
 & & & & H_1 & & & &
\end{tikzcd}
$$

Figure 17: Horizontal and sequential composition of direct derivations.

Given two productions $p_1 : (L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1)$ and $p_2 : (L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2)$ with $R_1 = L_2$, the *sequentially composed production* $p_1;p_2 : (L_1 \xleftarrow{l_1 \circ l_2'} K \xrightarrow{r_2 \circ r_1'} R_2)$ is obtained as shown at the top of the right diagram of Figure 17, where (5) is a pullback diagram. Derivation $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2}{\Longrightarrow} H_2$, where the match m_2 of the second direct derivation is the co-match of the first, may now be realized in one step using the sequentially composed production $p_1;p_2$. Vice versa, each direct derivation $G \overset{p_1;p_2,m_1}{\Longrightarrow} H_2$ via $p_1;p_2$ can be decomposed in a derivation $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_1^*}{\Longrightarrow} H_2$.

Proposition 43 (Sequential Composition). *Given two productions p_1 and p_2 and their sequential composition $p_1;p_2$ as above, the following statements are equivalent:*

1. *There is a direct derivation $G \overset{p_1;p_2,m_1}{\Longrightarrow} H_2$.*

2. *There is a derivation $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_1^*}{\Longrightarrow} H_2$ such that m_1^* is the co-match of m_1 w.r.t. p_1.*

Proof sketch. By the 3-cube pushout-pullback-lemma [36]. $\square$

Combining directly derived productions by sequential composition we now define derived productions for derivations of length greater than one: The *derived production* $\langle\rho\rangle$ of a derivation $\rho = (G_0 \overset{p_1,m_0}{\Longrightarrow} \cdots \overset{p_k,m_{k-1}}{\Longrightarrow} G_k)$ is defined as the sequential composition $\langle d_1\rangle; \ldots ; \langle d_k\rangle$ of the directly derived productions $d_i = (G_{i-1} \overset{p_i,m_{i-1}}{\Longrightarrow} G_i)$ of ρ. A direct derivation $K \overset{\langle\rho\rangle,e}{\Longrightarrow} Y$ using $\langle\rho\rangle$ is called *derived derivation.*

Each derivation sequence may be shortcut by a corresponding derived derivation. Vice versa, each derived derivation corresponds to a derivation using the original productions. Hence, these notions solve the Derived Production Problem 4 of Section 2.3 in the DPO-approach, which is formally stated by the theorem below.

Theorem 44 (Derived Productions). *Let ρ be a derivation and $\langle\rho\rangle$ its derived production as defined above. Then the following statements are equivalent:*

1. *There is a derived derivation $X_0 \stackrel{\langle\rho\rangle, e_0}{\Longrightarrow} X_k$.*
2. *There is a derivation $\delta = (X_0 \stackrel{p_1, n_0}{\Longrightarrow} \cdots \stackrel{p_k, n_{k-1}}{\Longrightarrow} X_k)$ and an embedding $\rho \stackrel{e}{\longrightarrow} \delta$, where e_0 is the embedding morphism of e.*

Proof sketch. Follows from Proposition 42, Proposition 43 by induction over the length k of the derivation ρ. $\square$

Let us come back to Problem 3 of Section 2.3 asking for an embedding of a derivation into a larger context. This question is now answered using the equivalence above, that is, a derivation ρ may be embedded via an embedding morphism e_0 if and only if the corresponding derived production $\langle\rho\rangle$ is applicable at e_0.

Theorem 45 (Embedding). *Let $\rho = (G_0 \stackrel{p_1, m_0}{\Longrightarrow} \cdots \stackrel{p_k, m_{k-1}}{\Longrightarrow} G_k)$ be a derivation, $\langle\rho\rangle$ its derived production, and $G_0 \stackrel{e_0}{\longrightarrow} X_0$ an embedding morphism. Then there is a derivation $\delta = (X_0 \stackrel{p_1, e_0 \circ m_0}{\Longrightarrow} \cdots \stackrel{p_k, e_{k-1} \circ m_{k-1}}{\Longrightarrow} X_k)$ and an embedding $\rho \stackrel{e}{\longrightarrow} \delta$ if and only if $\langle\rho\rangle$ is applicable at e_0, that is, if there is a derived derivation $X_0 \stackrel{\langle\rho\rangle, e_0}{\Longrightarrow} X_k$.*

Proof sketch. Theorem 45 follows from Theorem 44. $\square$

The embedding theorem above differs from the classical one in [2], which considers not only the embedding of a derivation into a larger context (called "JOIN" in [2]), but also the reduction of the context of a derivation (called "CLIP"). In fact, Theorem 45 corresponds to the "JOIN" part of the classical embedding theorem, and the JOIN condition is just the applicability of the derived production.

As anticipated in Section 2.3 already, the construction of a derived production for a given derivation ρ is not only useful in order to solve the embedding problem but is interesting in its own right. A related concept has been introduced in [57] under the name "concurrent production", where two productions are composed w.r.t. a given *dependency relation* specifying an overlapping of the right-hand side of the first with the left-hand side of the second production. The resulting concurrent production enjoys similar properties as the derived production above, and is, moreover, minimal w.r.t. to the context included in the interface graph

6.2 Amalgamation and Distribution

Section 2.4 introduced two concepts modeling the synchronization of direct derivations, namely amalgamated and (synchronous) distributed derivations. In this subsection we formalize these two concepts in the DPO approach and show how they are related.

Amalgamation

Amalgamated derivations describe the synchronized application of productions by a so-called amalgamated production. The synchronization of two productions p_1 and p_2 is modeled by identifying a common subproduction p_0: Let $p_i : (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i)$ be three productions for $k \in \{0,1,2\}$. Then production p_0 together with graph morphisms $in_L^1 : L_0 \to L_1, in_K^1 : K_0 \to K_1$ and $in_R^1 : R_0 \to R_1$ is a *subproduction* of p_1 if the resulting squares (1) and (2) below commute. The embedding of p_0 into p_1 is denoted by $in^1 = \langle in_L^1, in_K^1, in_R^1 \rangle : p_0 \to p_1$.

$$\begin{tikzcd}
L_0 \arrow[d, "in_L^1"'] \arrow[dr, phantom, "(1)"] & K_0 \arrow[l, "l_0"'] \arrow[r, "r_0"] \arrow[d, "in_K^1"] \arrow[dr, phantom, "(2)"] & R_0 \arrow[d, "in_R^1"] \\
L_1 & K_1 \arrow[l, "l_1"] \arrow[r, "r_1"'] & R_1
\end{tikzcd}$$

Productions p_1 and p_2 are synchronized w.r.t. p_0, shortly denoted by $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$, if p_0 is a subproduction of both p_1 and p_2 with embeddings in^1 and in^2, respectively. The amalgamated production is obtained as the gluing of p_1 and p_2 w.r.t. p_0, formally described by pushout constructions: Given productions p_i for $i \in \{0,1,2\}$ as above, the *amalgamated production* is $p_1 \oplus_{p_0} p_2 : (L \xleftarrow{l} K \xrightarrow{r} R)$ in the left diagram below, where subdiagrams (1), (2) and (3) are pushouts, and l and r are induced by the universal property of (2) such that all subdiagrams commute. A direct derivation $G \overset{p,m}{\Longrightarrow} X$ using the amalgamated production $p = p_1 \oplus_{p_0} p_2$ is called *amalgamated derivation*.

$$\begin{tikzcd}
 & L_0 \arrow[dl] \arrow[dr] & & K_0 \arrow[ll, "l_0"'] \arrow[rr, "r_0"] \arrow[dl] \arrow[dr] & & R_0 \arrow[dl] \arrow[dr] & \\
 & (1) & L_2 \arrow[dl] & (2) & K_2 \arrow[ll, "l_2"'] \arrow[rr, "r_2"] \arrow[dl] & (3) & R_2 \arrow[dl] \\
L_1 \arrow[dr] & & K_1 \arrow[ll, "l_1"'] \arrow[rr, "r_1"] \arrow[dr] & & R_1 \arrow[dr] & & \\
 & L & & K \arrow[ll, "l"'] \arrow[rr, "r"] & & R &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & G_0 \arrow[dl, "g_1"'] \arrow[dr, "g_2"] & \\
G_1 \arrow[dr, "g_2^*"'] & (PO) & G_2 \arrow[dl, "g_1^*"] \\
 & G_1 \oplus_{G_0} G_2 &
\end{tikzcd}$$

The concept of synchronizing derivations by amalgamation of productions is introduced in [11]. Based on a sequential decomposition of amalgamated productions into the common subproduction p_0 and the remainders of the elementary production $p_1 - p_0$ and $p_2 - p_0$, a sequential decomposition of amalgamated derivations is developed. Generalizing the Parallelism Theorem, the Amalgamation Theorem of [11] then establishes a connection between amalgamated derivations and derivations using the elementary productions p_1, p_2 and their remainders.

Distribution

The concept of distributed graphs allows to model the state of a system by interrelated local substates: A *distributed graph* $DG = (G_1 \overset{g_1}{\leftarrow} G_0 \overset{g_2}{\rightarrow} G_2)$ consists of two *local graphs* G_1 and G_2, an *interface graph* G_0, and graph morphisms g_1 and g_2 embedding the interface graph into the local graphs. The *global graph* $\oplus DG = G_1 \oplus_{G_0} G_2$ of DG is defined as the pushout object of g_1 and g_2 in the right diagram above. Given $G = \oplus DG$, the distributed graph is called a *splitting* of G. In the DPO approach we consider only total splittings, i.e., distributed graphs $G_1 \overset{g_1}{\leftarrow} G_0 \overset{g_2}{\rightarrow} G_2$ where g_1 and g_2 are total graph morphisms.

The local graphs can be transformed by local direct derivations. In order to form a distributed derivation, these direct derivations have to fulfill some compatibility conditions, which are formulated in terms of their matches [58,59]: Let DG be a distributed graph, $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ synchronized productions, and $L_i \xrightarrow{m_i} G_i$, $(i \in \{0,1,2\})$ be matches for p_i which are compatible with DG, i.e., $g_k \circ m_0 = m_k \circ in_L^k$ for $k \in \{1,2\}$. In this case, the family $(m_i)_{i \in \{0,1,2\}}$ is called a *distributed match* for $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ in DG. It satisfies the *distributed gluing condition* if all m_i satisfy the local gluing condition of p_i for $i \in \{0,1,2\}$, and the *connection condition*, i.e., $g_k(G_0 - m_0(L_0 - l_0(K_0)) \subseteq G_k - m_k(L_k - l_k(K_k))$ for $k \in \{1,2\}$.

The connection condition states, that a distributed derivation is not allowed to delete, for example, a vertex v_1 of the local graph G_1 if a corresponding vertex v_0 in the interface graph G_0 (i.e., such that $g_1(v_0) = v_1$) is preserved. Given a distributed match as above, there are local direct derivations $d_i = (G_i \overset{p_i,m_i}{\Longrightarrow} H_i)$ for $i \in \{0,1,2\}$ and distributed graphs $DD = (D_1 \overset{c_1}{\leftarrow} D_0 \overset{c_2}{\rightarrow} D_2)$ and $DH = (H_1 \overset{h_1}{\leftarrow} H_0 \overset{h_2}{\rightarrow} H_2)$ such that $d_1 ||_{d_0} d_2 : DG \Longrightarrow DH$ forms a *distributed derivation*, as shown in the left diagram of Figure 18. The graph morphisms c_1 and c_2 of the distributed graph DD are defined by $c_1 = l_1^{*-1} \circ g_1 \circ l_0^*$ and $c_2 = l_2^{*-1} \circ g_2 \circ l_0^*$. The graph morphisms h_1 and h_2 of DH are induced by the universal property of the right-hand side pushout $\langle r_0^*, m_0^* \rangle$ of the direct derivation d_0.

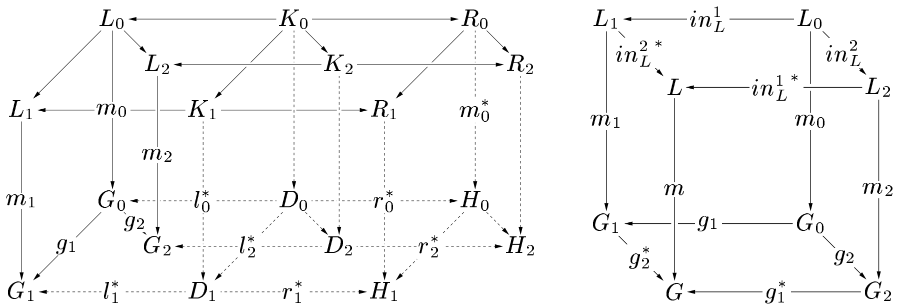


Figure 18: Distributed derivation, and applicability of subproduction p_i to local graph G_i.

Distributed derivations in the DPO approach are synchronous by definition, since only total splittings are considered. Given a total splitting $\oplus DG$, the connection condition ensures that also the distributed graphs DD and DH are total splittings [59].

The idea of modeling the distribution of states by a splitting a global graph into local subgraphs, related by an interface, has been developed in [12]. The derivation mechanism of [12], however, is more restricted than the one presented here because the local direct derivations are not allowed to change the interface graph G_0. This corresponds to distributed derivations (in the above sense) using a constant interface production p_0. In order to be able to change the interface graph, a global direct derivation had to be performed with the global graph $\oplus DG$ of the distributed graph DG, and two additional operations SPLIT and JOIN had to be introduced in order to switch between the distributed and the global representation. Distributed derivations with dynamic interfaces (like in this section) have been introduced in [60] in the SPO approach.

Of course, splitting a state into two substates with one interface is a very basic kind of a distribution. More general topologies are allowed in [59], where a distributed graph may be any diagram in the category ***Graph*** of graphs and total graph morphisms. In fact, the distribution concepts presented in this section are obtained by restricting distributed derivations in the sense of [59] to diagrams of the form $G_1 \xleftarrow{g_1} G_0 \xrightarrow{g_2} G_2$. The following distribution theorem is a special case of a corresponding theorem in [59]. It provides the solution to Problem 5 by relating amalgamated and distributed derivations.

Theorem 46 (Distribution). *Let DG be a distributed graph and $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ be synchronized productions. Then the following statements are equivalent:*

1. *There are an amalgamated derivation $\oplus DG \overset{p_1 \oplus_{p_0} p_2, m}{\Longrightarrow} H$, and a dis-*

tributed match $(m_i)_{i\in\{0,1,2\}}$ *such that*

(a) $g_2^* \circ m_1 = m \circ {in_L^2}^*$ *and* $g_1^* \circ m_2 = m \circ {in_L^1}^*$ *(i.e., the front and left square in the right diagram of Figure 18 commute),*

(b) $(m_i)_{i\in\{0,1,2\}}$ *satisfies the connection condition, and*

(c) *for* $i = 0,1,2$, $m_i|^{S_i} : L_i \to S_i$ *satisfies the local gluing condition of* p_i*, where* $S_1 = {g_2^*}^{-1}(m(L))$, $S_2 = {g_1^*}^{-1}(m(L))$, $S_0 = (g_2^* \circ g_1)^{-1}(m(L))$*, and* $m_i|^{S_i}$ *denotes the codomain restriction of* m_i *to* S_i*.*

2. *There is a distributed graph* DH *with* $\oplus DH = H$ *and a distributed derivation* $d_1||_{d_0}d_2 : DG \Longrightarrow DH$ *with* $d_i = (G_i \overset{p_i,m_k}{\Longrightarrow} H_i)$ *for* $i = 0,1,2$*.*

Proof sketch. "1. $\Longrightarrow$ 2.": According to the definition of distributed derivations we have to show that there are local direct derivations $d_i = (G_i \overset{p_i,m_i}{\Longrightarrow} H_i)$ for $i = 0,1,2$. Then, the existence of the distributed derivation $d_1||_{d_0}d_2 : DG \Longrightarrow DH$ follows from the existence of a distributed match satisfying (a) and (b) of 1. Condition (c) covers the difference between the gluing conditions of the local derivations and that of the given amalgamated derivation, i.e., the gluing condition for d_i follows from the gluing condition of $\oplus DG \overset{p_1\oplus_{p_0}p_2,m}{\Longrightarrow} H$ and the *difference gluing condition* (c).

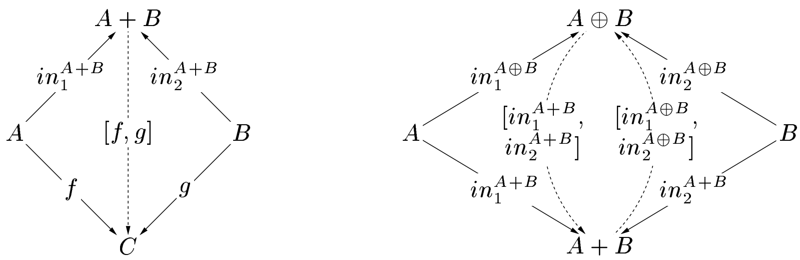
"2. $\Longrightarrow$ 1.": Given a distributed derivation $d_1||_{d_0}d_2 : DG \Longrightarrow DH$ as in Figure 18 on the left, the amalgamated derivation $\oplus DG \overset{p_1\oplus_{p_0}p_2,m}{\Longrightarrow} \oplus DH$ is obtained by constructing the global graphs $\oplus DG, \oplus DD$, and $\oplus DH$ as pushouts, and the morphisms of the co-production $\oplus DG \xleftarrow{l^*} \oplus DD \xrightarrow{r^*} \oplus DH$ as the universal morphisms induced by the pushout property of $\oplus DD$. The vertical morphisms $m : L \to \oplus DG$, $d : K \to \oplus DD$, and $m^* : R \to \oplus DH$ are induced in a similar way by the universal property of the pushout objects L, K, and R of $\langle in_L^1, in_L^2\rangle, \langle in_K^1, in_K^2\rangle$, and $\langle in_R^1, in_R^2\rangle$. □

7 Conclusions

For a detailed comparison between the DPO and the SPO approach and for concluding remarks about the contents of the two parts we address the reader to Sections 6 and 7 of the next chapter.

Acknowledgements

Most of the research results presented in this paper have been developed within the ESPRIT Working Groups COMPUGRAPH (1989-1992) and COMPU-



(a) (b)

Figure 19: (a) Coproduct. (b) Isomorphisms between distinct coproduct objects.

GRAPH II (1992-1996).

Appendix A: On commutativity of coproducts

This appendix addresses a quite technical topic: it introduces the notion of binary coproduct in a category, and shows that coproducts cannot be assumed to be commutative (in a strict way) unless the category is a preorder. This fact justifies the way we defined parallel productions (Definition 15) and their manipulation in Section 4, which may appear more complex than the corresponding definitions in previous papers in the literature.

In any category, a coproduct is an instance of the more general concept of *colimit* [20]. In the category **Set** of sets and functions (as well as in **Graph**) the coproduct of two objects is their disjoint union. In the following discussion we stick to binary coproducts, but most statements can be adapted to any kind of limit or colimit construction.

Definition 47 (coproducts). Let $\mathbf{C}$ be a category and A, B be two objects of $\mathbf{C}$. A *coproduct* of $\langle A, B\rangle$ is a triple $\langle A+B, in_1^{A+B}, in_2^{A+B}\rangle$ where $A+B$ is an object and $in_1^{A+B} : A \to A+B$, $in_2^{A+B} : B \to A+B$ are arrows of $\mathbf{C}$, such that the following universal property holds (see Figure 7(a)):

- for all pair of arrows $\langle f : A \to C, g : B \to C\rangle$ there exists a unique arrow $[f, g] : A+B \to C$ such that $[f, g] \circ in_1^{A+B} = f$ and $[f, g] \circ in_2^{A+B} = g$.

In this case $A+B$ is called a *coproduct object* of $\langle A, B\rangle$, and in_1^{A+B}, in_2^{A+B} are called the *injections*; arrow $[f, g]$ is called the *copairing* of f and g. One says that category $\mathbf{C}$ *has coproducts* if each pair of objects of $\mathbf{C}$ has a coproduct. □

In the following we assume that $\mathbf{C}$ has coproducts. From the definition it is evident that the coproduct of two objects is in general not unique. However,

$$
\begin{tikzcd}
A \arrow[rr, "f"] \arrow[d, "in_1^{A+B}"'] \arrow[drr, "in_1^{C+D} \circ f"] & & C \arrow[d, "in_1^{C+D}"] \\
A+B \arrow[rr, dotted, "f + g = \left[in_1^{C+D} \circ f,\, in_2^{C+D} \circ g\right]"] & & C+D \\
B \arrow[rr, "g"] \arrow[u, "in_2^{A+B}"] \arrow[urr, "in_2^{C+D} \circ g"'] & & D \arrow[u, "in_2^{C+D}"']
\end{tikzcd}
$$

(a)

$$
\begin{tikzcd}
& +(A,B) = A+B \arrow[dd, dotted, "\gamma_{A,B}"] & \\
A \arrow[ur, "in_1^{A+B}"] \arrow[dr, "in_2^{B+A}"'] & & B \arrow[ul, "in_2^{A+B}"'] \arrow[dl, "in_1^{B+A}"] \\
& +(X(A,B)) = B+A &
\end{tikzcd}
$$

(b)

Figure 20: (a) Coproduct of arrows. (b) Commutativity up to natural isomorphism.

given any two coproducts of two objects they are not only isomorphic, but there exists *only one* isomorphism commuting with the injections. This fact holds for limits and colimits in general [20].

Fact 48 (uniqueness of coproducts up to isomorphism). *Let* $\langle A + B, in_1^{A+B}, in_2^{A+B}\rangle$ *and* $\langle A \oplus B, in_1^{A\oplus B}, in_2^{A\oplus B}\rangle$ *be two coproducts of* $\langle A, B\rangle$ *in* $\mathbf{C}$. *Then objects* $A + B$ *and* $A \oplus B$ *are isomorphic, and there exists only one isomorphism* $\phi : A + B \to A \oplus B$ *such that* $in_1^{A\oplus B} = \phi \circ in_1^{A+B}$ *and* $in_2^{A\oplus B} = \phi \circ in_2^{A+B}$. *As shown in Figure 7(b),* ϕ *is determined by the universal property of* $A + B$ *as* $\phi = [in_1^{A\oplus B}, in_2^{A\oplus B}]$, *and similarly for its inverse* $\phi^{-1} = [in_1^{A+B}, in_2^{A+B}]$. □

Often it is useful to regard the formal operator $+$ used to denote coproduct objects as a real function, returning a specific object of $\mathbf{C}$, and not merely specifying a class of isomorphic objects. This can be obtained by fixing a *choice of coproducts*, i.e., by fixing a specific coproduct for each pair of objects. Every choice of coproducts determines a coproduct functor as follows.

Definition 49 (coproduct functor). Suppose that a *choice of coproducts* for $\mathbf{C}$ is given, i.e., a mapping associating with each pair of objects $\langle A, B\rangle$ a coproduct, say $\langle A + B, in_1^{A+B}, in_2^{A+B}\rangle$. This determines a bifunctor $+ : \mathbf{C}^2 \to \mathbf{C}$, called the *coproduct functor*, defined on objects as $+(A, B) = A + B$. On arrows, if $f : A \to C$ and $g : B \to D$ are in $\mathbf{C}$, then $+(f, g)$ (more often written $f + g$) is uniquely determined, using the universal property of coproducts, as $f+g \overset{def}{=} [in_1^{C+D} \circ f, in_2^{C+D} \circ g] : A + B \to C + D$, as shown in Figure 20(a). □

Coproducts are widely used in the algebraic theory of graph grammars, for example in the definition of parallel productions (see Definition 15). A natural question that arises in this framework is: Can one consider the coproduct functor as commutative? We show here that although every coproduct functor is commutative up to a natural isomorphism, in general it is not safe to assume that it is *strictly* commutative, i.e., commutative up to the *identity* natural transformation.

Fact 50 (commutativity up to natural isomorphism of coproducts). *Any coproduct functor* $+ : \mathbf{C}^2 \to \mathbf{C}$ *is* commutative up to a natural isomorphism, *in the following sense. Let* $X : \mathbf{C}^2 \to \mathbf{C}^2$ *be the functor that exchanges its arguments (i.e.,* $X(A,B) = \langle B,A \rangle$*). Then there exists a natural transformation*[f] $\gamma : + \Rightarrow + \circ X : \mathbf{C}^2 \to \mathbf{C}$*, such that* $\gamma_{A,B}$ *is an isomorphism for each* A,B*. The component* $\gamma_{A,B}$ *of the natural isomorphism* γ *is easily determined as in Figure 20(b), and the proof of the naturality of* γ *is routine.* □

We are now ready to say when a coproduct functor is strictly commutative. Although the following definition may look very restrictive, it seems to be the only reasonable one in a categorical framework: compare it, for example, to the definition of *strict monoidal categories* in [20].

Definition 51 (strictly commutative coproduct functors). A coproduct functor $+ : \mathbf{C}^2 \to \mathbf{C}$ is *strictly commutative* if the natural isomorphism $\gamma : + \Rightarrow + \circ X : \mathbf{C}^2 \to \mathbf{C}$ is the identity natural transformation. This means not only that $A + B = B + A$ (they are the same object), but also that $\gamma_{A,B} = id_{A+B}$ for each pair $\langle A, B \rangle$. □

The following result shows that if a category $\mathbf{C}$ has a coproduct functor that is strictly commutative, then $\mathbf{C}$ is a preorder, i.e., there is at most one arrow between each pair of objects. Since none of the categories of graphs (or of other structures) considered in the algebraic theory of graph grammars is a preorder, this fact shows that one cannot assume, in general, that "coproducts are strictly commutative".

Proposition 52 (strictly commutative coproducts only exist in preorders). *Let* $\mathbf{C}$ *be a category that has coproducts, and suppose that there exists a choice of coproducts for* $\mathbf{C}$ *such that the induced coproduct functor* $+ : \mathbf{C}^2 \to \mathbf{C}$ *is strictly commutative. Then* $\mathbf{C}$ *is a preorder, i.e., for each pair of arrows* $f, g : A \to B$*, it holds* $f = g$*.*

Proof. By the definition of strict commutativity, $\gamma_{A,B} = id_{A+B}$; thus the diagram in Figure 7(a) shows that the injections of A in $A + B$ and in $B + A$, i.e., in_1^{A+B} and in_2^{B+A}, are exactly the same arrow, because the two triangles must commute. Because of the same reason, if we consider the coproduct of A with itself, then it turns out that there is *only one injection* $in_1^{A+A} = in_2^{A+A}$

[f] Given two functors $F, G : \mathbf{C} \to \mathbf{D}$, a *natural transformation* $\gamma : F \Rightarrow G : \mathbf{C} \to \mathbf{D}$ is a family of arrows of $\mathbf{D}$ indexed by objects of $\mathbf{C}$, $\gamma = \{\gamma_A \mid A \in Obj(\mathbf{C})\}$, such that $\gamma_A : F(A) \to G(A)$, and satisfying the following *naturality* condition: for each arrow $f : A \to B$ of $\mathbf{C}$, $G(f) \circ \gamma_A = \gamma_B \circ F(f)$ (see [20]).

$$
\begin{tikzcd}
& A+B \arrow[dd, dotted, "\gamma_{A,B}"] & \\
A \arrow[ur, "in_1^{A+B}"] \arrow[dr, "in_2^{B+A}"'] & & B \arrow[ul, "in_2^{A+B}"'] \arrow[dl, "in_1^{B+A}"] \\
& B+A &
\end{tikzcd}
\qquad
\begin{tikzcd}
& A+A \arrow[dd, dotted, "id_{A+B}"] & \\
A \arrow[ur, "in_1^{A+A}"] \arrow[dr, "in_2^{A+A}"'] & & A \arrow[ul, "in_2^{A+A}"'] \arrow[dl, "in_1^{A+A}"] \\
& A+A &
\end{tikzcd}
\qquad
\begin{tikzcd}
& A+A \arrow[dd, dotted, "{[f,g]}"] & \\
A \arrow[ur, "in_1^{A+B}"] \arrow[dr, "f"'] & & A \arrow[ul, "in_2^{A+A}"'] \arrow[dl, "g"] \\
& B &
\end{tikzcd}
$$

(a) (b) (c)

Figure 21: Strict commutativity of coproducts implies that all parallel arrows (like f and g) are identical.

from A to $A + A$, as depicted in Figure 7(b). Suppose now that there are two arrows $f, g : A \to B$. By the universal property there is only one arrow $[f, g] : A + A \to B$ (as shown in Figure 7(c)) such that $[f, g] \circ in_1^{A+A} = f$ and $[f, g] \circ in_2^{A+A} = g$. Therefore, since $in_1^{A+A} = in_2^{A+A}$, we have that $f = g$, and $\mathbf{C}$ is a preorder. $\square$

Other assumptions on the properties of coproducts or on the category $\mathbf{C}$ may yield similar results, as stated by the following corollary.

Corollary 53.

1. *Say that a category* $\mathbf{C}$ *has* unique coproducts *if it has coproducts and for each pair of objects* $\langle A, B\rangle$ *there is only one triple* $\langle A + B, in_1^{A+B} : A \to A + B, in_2^{A+B} : B \to A + B\rangle$ *satisfying the universal property. If a category* $\mathbf{C}$ *has unique coproducts, then it is a preorder.*

2. *If* $\mathbf{C}$ *has coproducts and* $\mathbf{C}$ *has no non-trivial automorphisms (i.e., for each object* A *there is only one isomorphism from* A *to itself, the identity), then* $\mathbf{C}$ *is a preorder.* $\square$

Both statements are easily proved by showing first that the hypotheses imply that there is only one injection from A to $A + A$, and then using the diagram of Figure 7(c).

Finally, it is worth stressing that the *uniqueness of coproduct objects* is not sufficient to deduce that a category is a preorder. More formally, if $\mathbf{C}$ has coproducts, say that it has *unique coproduct objects* if for each pair of objects $\langle A, B\rangle$ and for each pair of coproducts of $\langle A, B\rangle$, say $\langle C, in_1^C : A \to C, in_2^C : B \to C\rangle$ and $\langle D, in_1^D : A \to D, in_2^D : B \to D\rangle$, one has $C = D$. For example, the (skeletal) full sub-category of finite sets having as objects $\{\underline{n} \mid n \in \mathbb{N}\}$, where $\underline{n} = \{1, 2, \ldots, n\}$, has unique coproduct objects but it is not a preorder.

$$
\begin{tikzcd}
A_1 + A_2 \arrow[d, "{[m_1, m_2]}"'] \arrow[dr, phantom, "(1)"] & B_1 + B_2 \arrow[l, "a_1 + a_2"'] \arrow[d, "{[n_1, n_2]}"] \\
C & D \arrow[l, "x"]
\end{tikzcd}
\qquad
\begin{tikzcd}
A_1 \arrow[d, "r"] \arrow[dd, bend right=60, dotted, "m_1"'] & B_1 \arrow[l, "a_1"'] \arrow[d, "n_1"] \arrow[dl, phantom, "(2)"] & \\
X_1 \arrow[d, "v"'] \arrow[dr, phantom, "(4)"] & D \arrow[l, "s"'] \arrow[d, "t"] \arrow[dl, dotted, "x"] \arrow[dr, phantom, "(3)"] & B_2 \arrow[l, "n_2"'] \arrow[d, "a_2"] \\
C & X_2 \arrow[l, "w"'] & A_2 \arrow[l, "u"'] \arrow[ll, bend left=30, dotted, "m_2"]
\end{tikzcd}
$$

Figure 22: Butterfly Lemma.

Appendix B: Proof of main results of Section 5

This appendix presents the proof of Lemma 31, which is the main technical result of Section 5.3. This proof is based on the constructive proofs of Lemmas 19 and 20 (i.e., the synthesis and analysis constructions) which are reported below for completeness, and also because they show in an instructive way the kind of reasoning that is used when proving results in the algebraic approach to graph transformations. To start with, we need the following well-known technical lemma [3].

Lemma 54 (Butterfly lemma). *Let a_1 and a_2 be injective graph morphisms. Then the square (1) of Figure 22 is a pushout if and only if there are graphs X_1, X_2 and graph morphisms r, s, v, w, t and u, such that in the right part of the same figure diagrams (2), (3), and (4) are pushouts, (4) is a pullback, and all triangles commute.* $\square$

Proof of Lemma 19 (analysis of parallel direct derivations). *Let $\rho = (G \stackrel{q}{\Longrightarrow} H)$ be a parallel direct derivation using the parallel production $q = p_1 + \ldots + p_k : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$. Then for each ordered partition $\langle I = \langle i_1, \ldots, i_n \rangle, J = \langle j_1, \ldots, j_m \rangle \rangle$ of $\{1, \ldots, k\}$ (i.e., $I \cup J = \{1, \ldots, k\}$ and $I \cap J = \emptyset$) there is a constructive way to obtain a sequential independent derivation $\rho' = (G \stackrel{q'}{\Longrightarrow} X \stackrel{q''}{\Longrightarrow} H)$, called an* analysis *of ρ, where $q' = p_{i_1} + \ldots + p_{i_n}$, and $q'' = p_{j_1} + \ldots + p_{j_m}$.*

$$
\begin{tikzcd}
\multicolumn{3}{c}{q = p_1 + \ldots + p_k :} \\
\big(L \arrow[d, "m"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] & R\big) \arrow[d, "n"] \\
G & D \arrow[l, "x"] \arrow[r, "y"] & H
\end{tikzcd}
\qquad
\begin{tikzcd}
\multicolumn{3}{c}{q' + q'' = (p_{i_1} + \ldots + p_{i_n}) + (p_{j_1} + \ldots + p_{j_m}) :} \\
\big(L_1 + L_2 \arrow[d, "{[m_1, m_2]}"] \arrow[dr, phantom, "(1)"] & K_1 + K_2 \arrow[l, "l_1 + l_2"'] \arrow[r, "r_1 + r_2"] \arrow[d, "{[k_1, k_2]}"] \arrow[dr, phantom, "(2)"] & R_1 + R_2\big) \arrow[d, "{[n_1, n_2]}"'] \\
G & D \arrow[l, "x"] \arrow[r, "y"] & H
\end{tikzcd}
$$

Figure 23: Splitting a parallel production.

If $I = \langle\rangle$ (or $J = \langle\rangle$) the statement follows by taking $X = G$ ($X = H$), $q'' = q$ ($q' = q$), and an empty direct derivation as q' (q''). Now let us assume that q is a proper parallel production, and that I and J are not empty. By the hypotheses on I and J, productions $q' + q''$ and q are clearly isomorphic via the permutation Π defined as $\Pi(x) = i_x$ if $1 \leq x \leq n$ and $\Pi(x) = j_{x-n}$ if $n < x \leq n+m$ (see Definition 15. Thus since the left diagram of Figure 23 is a double-pushout by hypothesis, the right diagram is a double-pushout as well, where we supposed that $q' : (L_1 \stackrel{l_1}{\leftarrow} K_1 \stackrel{r_1}{\rightarrow} R_1)$ and $q'' : (L_2 \stackrel{l_2}{\leftarrow} K_2 \stackrel{r_2}{\rightarrow} R_2)$.

Therefore we can apply Lemma 54 to (1) and (2), obtaining the pushouts (3)–(5) and (6)–(8), respectively, shown in Figure 24. Now let (9) be the pushout of arrows l and u, where the pushout object is called X (left part of Figure 25). Then we can build the required derivation via q' and q'' as in the right part of Figure 25, where we exploited the well-known fact that a square consisting of two adjacent pushouts is again a pushout. Such derivation is sequential independent because it is easy to check that arrows $f : L_2 \rightarrow X_2$ and $t : R_1 \rightarrow Y_1$ satisfy the conditions of Definition 13. □

Proof of Lemma 20 (synthesis of sequential independent derivations). *Let $\rho = (G \stackrel{q'}{\Longrightarrow} X \stackrel{q''}{\Longrightarrow} H)$ be a sequential independent derivation. Then there is a constructive way to obtain a parallel direct derivation $\rho' = (G \stackrel{q'+q''}{\Longrightarrow} H)$, called a* synthesis *of ρ.*

Let ρ be the sequential independent derivation of Figure 26. We have to show that the commutative diagrams of Figure 24 can be constructed, where squares from (3) to (8) have to be pushouts, and (4) and (8) need to be pullbacks. Then by two applications of the Butterfly Lemma we obtain the parallel production in the right part of Figure 23, as required.

$$
\begin{tikzcd}
L_1 \arrow[d, "a"] \arrow[dd, bend right=60, dotted, "m_1"'] \arrow[dd, phantom, bend right=30, "="] \arrow[dr, phantom, "(3)"] & K_1 \arrow[l, "l_1"'] \arrow[d, "k_1"] & \\
X_1 \arrow[d, "c"] \arrow[dr, phantom, "(4)"] & D \arrow[l, "b"'] \arrow[d, "l"] \arrow[dl, dotted, "x"] \arrow[dr, phantom, "(5)"] & K_2 \arrow[l, "k_2"'] \arrow[d, "l_2"] \\
G & X_2 \arrow[l, "d"] \arrow[d, phantom, "="] & L_2 \arrow[l, "f"] \arrow[ll, bend left=40, dotted, "m_2"] \\
& {} &
\end{tikzcd}
\qquad
\begin{tikzcd}
& K_1 \arrow[r, "r_1"] \arrow[d, "k_1"'] \arrow[dr, phantom, "(6)"] & R_1 \arrow[d, "t"] \arrow[dd, bend left=60, dotted, "n_1"] \arrow[dd, phantom, bend left=30, "="] \\
K_2 \arrow[r, "k_2"] \arrow[d, "r_2"'] \arrow[dr, phantom, "(7)"] & D \arrow[r, "u"] \arrow[d, "v"] \arrow[dr, dotted, "y"'] \arrow[dr, phantom, "(8)"] & Y_1 \arrow[d, "p"] \\
R_2 \arrow[r, "w"] \arrow[rr, bend right=40, dotted, "n_2"'] & Y_2 \arrow[r, "q"] \arrow[d, phantom, "="] & H \\
& {} &
\end{tikzcd}
$$

Figure 24: Application of the Butterfly Lemma.

$$
\begin{tikzcd}
D \arrow[r, "u"] \arrow[d, "l"'] \arrow[dr, phantom, "(9)"] & Y_1 \arrow[d, "i"'] \\
X_2 \arrow[r, "j"] & X
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=small]
\multicolumn{3}{c}{q' = p_{i_1} + \ldots + p_{i_n} :} & & q'' = p_{j_1} + \ldots + p_{j_m} : & & \\
\big(L_1 \arrow[d] \arrow[dr, phantom, "(3)+(4)"] & K_1 \arrow[l, "l_1"'] \arrow[r, "r_1"] \arrow[d] \arrow[dr, phantom, "(6)+(9)"] & R_1\big) \arrow[dr] & & \big(L_2 \arrow[dl] \arrow[dr, phantom, "(5)+(9)"] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d] \arrow[dr, phantom, "(7)+(8)"] & R_2\big) \arrow[d] \\
G & X_2 \arrow[l] \arrow[rr] & & X & & Y_1 \arrow[ll] \arrow[r] & H
\end{tikzcd}
$$

Figure 25: Building the sequential derivation.

$$
q' = p_{i_1} + \ldots + p_{i_n} : \qquad q'' = p_{j_1} + \ldots + p_{j_m} :
$$

$$
\begin{tikzcd}
\big(L_1 \arrow[d, "m_1"'] & K_1 \arrow[l, "l_1"'] \arrow[r, "r_1"] \arrow[d, "g"'] & R_1\big) \arrow[dr, "r"'] \arrow[drrr, dotted, "t"] & & \big(L_2 \arrow[dl, "s"] \arrow[dlll, dotted, "f"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "h"] & R_2\big) \arrow[d, "n_2"] \\
G & X_2 \arrow[l, "d"] \arrow[rr, "j"'] & & X & & Y_1 \arrow[ll, "i"] \arrow[r, "p"'] & H
\end{tikzcd}
$$

Figure 26: A sequential independent derivation.

The diagrams of Figure 24 are constructed as follows. Some of the graphs and of the morphisms are taken from the derivation ρ of Figure 26. The others are obtained constructively in the following way:

- $\langle D, k_2, l\rangle$ is taken as a pushout complement of $\langle l_2, f\rangle$. It is determined up to isomorphism because l_2 is injective. Thus square (5) is a pushout by construction, and l is injective because so is l_2 and in category **Graph** pushouts push out monos.

- Morphism $k_1 : K_1 \to D$ is determined by showing that morphism $g : K_1 \to X_2$ factorizes through l, i.e., that there exists a k_1 such that $l \circ k_1 = g$; then, since l is injective, k_1 is unique.

  We have to show that the image of K_1 in X_2 (i.e., $g(K_1)$) is contained in the image of D in X_2, i.e., in $l(D)$. Since X_2 is the pushout object of $\langle l_2, k_2\rangle$, this is false only if $g(K_1)$ contains an item $z \in f(L_2)$ such that $z \notin f(l_2(K_2))$. But since X is the pushout object of $\langle l_2, h\rangle$ and $j \circ f = s$, this would mean that $j(z)$ is in $s(L_2)$ but not in $i(Y_1)$, which is absurd by the existence of a morphism $t : R_1 \to Y_1$ such that $i \circ t = r$.

- $\langle X_1, a, b\rangle$ is taken as a pushout of $\langle l_1, k_1\rangle$. Thus (3) is a pushout by construction, and b is injective because so is l_1.

- Morphism x is defined as $x = d \circ l$.

- Morphism c is uniquely determined by condition $m_1 \circ l_1 = d \circ g = d \circ l \circ k_1 = x \circ k_1$, because (3) is a pushout.

- By a well-known decomposition property of pushouts, square (4) is a pushout because so are (3) and (3) + (4). Thus c is injective because so is l.

- Square (4) is a pullback, because any pushout in **Graph** made of injective morphisms is a pullback as well.

- Let us show that $j \circ l$ factorizes through i: this uniquely determines morphism u so that square (9) in Figure 24 commutes and u is injective, because so are l, j, and i; moreover, square (9) is a pushout because so are (5) and (5) + (9).

  To show that $j \circ l$ factorizes through i we prove that the image of D in X, $j(l(D))$, is contained in the image of Y_1 in X, $i(Y_1)$. Otherwise, since X is the pushout object of $\langle l_2, h\rangle$, there is an item $z = j(l(z''))$ such that $z \in s(L_2)$ and $z \notin s(l_2(K_2))$. Let $z' \in L_2$ be such that $s(z') = z$ (thus

$$
\begin{tikzcd}
A \arrow[rr, "f"] \arrow[dr, dotted, "\phi_A"'] \arrow[dd, "g"'] & & B \arrow[dr, dotted, "\phi_B"] \arrow[dd, "h"] & \\
& A' \arrow[rr, "f'"] \arrow[dd, "g'"'] & & B' \arrow[dd, "h'"] \\
C \arrow[rr, "k"] \arrow[dr, dotted, "\phi_C"'] & & D \arrow[dr, dotted, "\phi_D"] & \\
& C' \arrow[rr, "k'"'] & & D'
\end{tikzcd}
$$

Figure 27: Isomorphic pushouts and pushout complements.

$z' \notin l_2(K_2)$); then we have $j(f(z')) = s(z') = z = j(l(z''))$, which implies $f(z') = l(z'')$ by injectivity of j. But since X_2 is the pushout object of $\langle l_2, k_2 \rangle$, this implies that $z' \in l_2(K_2)$, which is absurd.

- $\langle Y_2, w, v \rangle$ is taken as a pushout of $\langle k_2, r_2 \rangle$. Therefore (7) is a pushout and v is injective because so is r_2.

- Morphism y is defined as $y = p \circ u$.

- Morphism q is uniquely determined by the condition $n_2 \circ r_2 = p \circ h = p \circ u \circ k_2 = y \circ k_2$, because (7) is a pushout.

- Since (7) and (7) + (8) are pushouts, also (8) is a pushout. Thus q is injective because so is u. Square (8) is also a pullback because all its morphisms are injective.

- Finally, by the so-called "pushout-pullback decomposition property" (see [22]) which holds for category **Graph** (with injective morphism as distinguished class of morphisms **M**), since (6) + (8) is a pushout, (8) is a pullback, and morphisms r_1, u, v, q and p are injective, then (6) is a pushout as well. $\square$

The next easy technical lemma will be helpful in the proof of the main result below.

Lemma 55 (existence and uniqueness of some categorical constructions).
1. Given morphisms f, g, f', and g' and isomorphisms ϕ_A, ϕ_B, and ϕ_C, as in Figure 27, such that the two resulting squares commute, let $\langle D, h, k \rangle$ be a pushout of $\langle f, g \rangle$. Then

- *There exists at least one pushout of $\langle f', g' \rangle$, given by $\langle D, h \circ \phi_B^{-1}, k \circ \phi_C^{-1} \rangle$.*

- *For each pushout $\langle D', h', k' \rangle$ of $\langle f', g' \rangle$, there exists a unique isomorphism $\phi_D : D \to D'$ that makes the diagram commutative.*

2. Given injective *morphisms f and f', morphism h and h', and isomorphisms ϕ_A, ϕ_B, and ϕ_D, as in Figure 27, such that the two resulting squares commute, let $\langle C, g, k \rangle$ be a pushout complement of $\langle f, h \rangle$. Then:*

- *There exists at least one pushout complement of $\langle f', h' \rangle$, given by $\langle C, g \circ \phi_A^{-1}, \phi_D \circ k \rangle$.*

- *For each pushout complement $\langle C', g', k' \rangle$ of $\langle f', h' \rangle$, there exists an isomorphism $\phi_C : C \to C'$ that makes the diagram commutative.*

Proof. Point 1 holds in any category, and it is a direct consequence of the uniqueness of colimits up to a unique mediating isomorphism [20]. Point 2 follows instead from specific properties of the category **Graph**, where the pushout complement happens to be unique up to isomorphism if the top arrow is injective 9. □

Finally, we can present the proof of Lemma 31, stating that analysis, synthesis and shift preserve equivalence $\equiv_1$ on derivations. Since most of the work was in the presentations of the constructive proofs of Lemmas 19 and 20, some parts of the proof are just outlined.

Proof of Lemma 31 (analysis, synthesis and shift preserve $\equiv_1$). *Let $\rho \equiv_1 \rho'$ via $\{\Pi_i\}_{i \leq n}$, and let ϕ_{G_0} and ϕ_{G_n} be the corresponding isomorphisms between their starting and ending graphs (see Figure 14).*

1. **(Analysis)** *If $\rho_1 \in ANAL^i_j(\rho)$, then there exists at least one derivation $\rho_2 \in ANAL^i_{\Pi_i(j)}(\rho')$. Moreover, for each $\rho_2 \in ANAL^i_{\Pi_i(j)}(\rho')$, it holds that $\rho_2 \equiv_1 \rho_1$ with the same isomorphisms ϕ_{G_0} and ϕ_{G_n} relating their starting and ending graphs.*

For simplicity, we assume that ρ and ρ' are parallel *direct* derivations (thus $\rho \equiv_1 \rho'$ via Π): the generalization to equivalent derivations of arbitrary length is straightforward. The existence of a derivation $\rho_2 \in ANAL_{\Pi(j)}(\rho')$ is ensured by Lemma 19.

If $\rho = (G \overset{p_1+\ldots+p_k}{\Longrightarrow} H)$ and $\rho' = (G' \overset{p'_1+\ldots+p'_k}{\Longrightarrow} H')$, exploiting the same technique used above at the beginning of the proof of Lemma 19, the two direct derivations can be transformed into corresponding derivations $\eta = (G \overset{p_j+q_2}{\Longrightarrow} H)$ and $\eta' = (G' \overset{p'_{\Pi(j)}+q'_2}{\Longrightarrow} H')$, where $q_2 = p_1 + \ldots + p_{j-1} + p_{j+1} + \ldots + p_k$, and $q'_2 = p'_1 + \ldots + p'_{\Pi(j)-1} + p'_{\Pi(j)+1} + \ldots + p'_k$.

Suppose now that $p_j : (L_1 \overset{l_1}{\leftarrow} K_1 \overset{r_1}{\rightarrow} R_1)$, $p'_{\Pi(j)} : (L'_1 \overset{l'_1}{\leftarrow} K'_1 \overset{r'_1}{\rightarrow} R'_1)$, $q_2 : (L_2 \overset{l_2}{\leftarrow} K_2 \overset{r_2}{\rightarrow} R_2)$, and $q'_2 : (L'_2 \overset{l'_2}{\leftarrow} K'_2 \overset{r'_2}{\rightarrow} R'_2)$.[g] By construction it is clear that η and η' are 1-equivalent because so are ρ and ρ'. Let Π_1 be the unique permutation on $\{1\}$ and Π_2 be the bijective mapping between the productions of q_2 and q'_2 induced by Π: Π_1 and Π_2 will be used below.

Without loss of generality, let us assume that derivations η and η' have the same shape of the right double-pushout of Figure 23, more precisely, that the direct derivation $(1)+(2)$ of Figure 23 is η, that η' is an identical diagram $(1')+(2')$ where all the names of graphs and morphisms have an apex, and that η and η' are related by a family of isomorphisms $\{\phi_X : X \to X' \mid X \text{ is a graph of } \eta\}$ making the diagram commutative. In particular, also the squares obtained by considering individually the components of each coproduct commute (for example, $\phi_G \circ m_1 = m'_1 \circ \phi_{L_1}$ and $\phi_G \circ m_2 = m'_2 \circ \phi_{L_2}$).

Following the construction of the proof of Lemma 19, diagrams as in Figure 24 plus square (9) of Figure 25 can be constructed (in a non-deterministic way) for both η and η'. We show below that each pair of collections of diagrams obtained in this way induces a unique family of isomorphisms between corresponding graphs of the diagrams, which includes the family of isomorphisms relating η and η'. These isomorphisms, together with the permutations Π_1 and Π_2 introduced above, prove that the two sequential derivations obtained from η and η' (like that in the right part of Figure 25) are indeed 1-equivalent. This construction shows also that the isomorphisms ϕ_G and ϕ_H relating the starting end ending graphs of ρ and ρ' are not changed.

Let us assume that the diagrams of Figure 24 and square (9) of Figure 25 are obtained from derivation η, and that similar diagrams (but where all the names of graphs and morphisms have an apex) are obtained from η'. All the corresponding graphs of the two families of diagrams are related by the isomorphisms relating η and η', except of X_1, X_2, Y_1, Y_2 and X. Isomorphism $\phi_{X_1} : X_1 \to X'_1$ is uniquely determined by point 1 of Lemma 55, because (3) and (3') are pushouts; using the same argument also, isomorphism ϕ_{X_2}, ϕ_{Y_1}, ϕ_{Y_2} and ϕ_X are uniquely determined, because their source and target graphs are pushout objects. It is straightforward to check that the resulting family of isomorphisms makes everything commutative.

2. **(Synthesis)** *If $\rho_1 \in SYNT^i(\rho)$, then there exists at least one derivation $\rho_2 \in SYNT^i(\rho')$. Moreover, for each $\rho_2 \in SYNT^i(\rho')$, it holds that $\rho_2 \equiv_1 \rho_1$ with the same isomorphisms ϕ_{G_0} and ϕ_{G_n} relating their starting and ending*

[g] Note that since $p_j = p'_{\Pi(j)}$, we have $X_1 = X'_1$ for $X \in \{L, K, R, l, r\}$. They have different names just for ease of reference.

graphs.

We assume without loss of generality that ρ and ρ' have length 2, thus $\rho \equiv_1 \rho'$ via $\langle \Pi_1, \Pi_2 \rangle$. Since $\rho_1 \in SYNT(\rho)$ by hypothesis, derivation ρ is sequential independent: let it be the derivation of Figure 26. To show the existence of a $\rho_2 \in SYNT(\rho')$, by Lemma 20 it is sufficient to show that that ρ' is sequential independent as well. Let ρ' be as in Figure 26, but with all names of graphs and morphisms having an apex. Since $\rho \equiv_1 \rho'$, there is a family of isomorphisms relating the corresponding graphs of ρ and ρ' so that everything commutes: thus morphism f' and t' making ρ' sequential independent exists, and are uniquely determined (because i' and j' are injective) as $f' = \phi_{X_2} \circ f \circ \phi_{L_2}^{-1}$ and $t' = \phi_{Y_1} \circ t \circ \phi_{R_1}^{-1}$.

Since both ρ and ρ' are sequential independent, the diagrams of Figure 24 can be constructed (non-deterministically) for both of them, in such a way that the required squares are pushouts or pullbacks.

We show below that such constructions induce a family of isomorphism between the corresponding graphs of the diagrams, which includes the family of isomorphisms relating ρ and ρ'. The resulting isomorphisms can then be used to show that the two parallel direct derivations (similar to that of Figure 23) obtained from ρ and ρ' respectively are 1-equivalent. For the mapping between productions, it is obtained in the obvious way by joining the permutations $\langle \Pi_1, \Pi_2 \rangle$ which relate the productions used in ρ and ρ'.

To prove that the diagrams of Figure 24 constructed for ρ and ρ' induce isomorphisms between corresponding graphs, it is sufficient to go through the proof of Lemma 20. In particular, the missing isomorphisms $\phi_D : D \to D'$, $\phi_{X_1} : X_1 \to X_1'$, and $\phi_{Y_2} : Y_2 \to Y_2'$ are determined by Lemma 55, because the corresponding graphs are either pushout objects or pushout complement objects (and suitable morphisms are injective). The remaining steps of the mentioned proof are needed to show constructively that all the squares induced by the family of isomorphisms commute.

3. **(Shift)** *If $\rho_1 \in SHIFT_j^i(\rho)$, then there exists at least one derivation $\rho_2 \in SHIFT_{\Pi_i(j)}^i(\rho')$. Moreover, for each $\rho_2 \in SHIFT_{\Pi_i(j)}^i(\rho')$, it holds that $\rho_2 \equiv_1 \rho_1$ with the same isomorphisms ϕ_{G_0} and ϕ_{G_n} relating their starting and ending graphs.*

If $j \geq 1$, the proof is immediate by exploiting points 1) and 2), because $SHIFT_j^i = SYNT^{i-1} \circ ANAL_j^i$ by Definition 24. If instead $j = 0$ (and thus $\Pi_i(j) = 0$ as well), then both ρ and ρ' end with an empty direct derivation, and ρ_1 and ρ_2 are uniquely determined by the construction described in Definition 24, because relation $SHIFT_0^i$ is deterministic. It is easy to check that if this construction is applicable to ρ, then it is applicable to ρ' as well, producing 1-equivalent derivations. $\square$

References

1. H. Ehrig, M. Pfender, and H.J. Schneider. Graph-grammars: an algebraic approach. In *Proceedings IEEE Conf. on Automata and Switching Theory*, pages 167–180, 1973.
2. H. Ehrig. Introduction to the Algebraic Theory of Graph Grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *Proceedings of the 1st International Workshop on Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *Lecture Notes in Computer Science*, pages 1–69. Springer Verlag, 1979.
3. 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 *Lecture Notes in Computer Science*, pages 3–14. Springer Verlag, 1987.
4. H. Ehrig, M. Korff, and M. Löwe. Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proceedings of the 4th International Workshop on Graph-Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 24–37. Springer Verlag, 1991.
5. M. Löwe. Algebraic approach to single-pushout graph transformation. *Theoret. Comput. Sci.*, 109:181–224, 1993.
6. H.-J. Kreowski. Is parallelism already concurrency? Part 1: Derivations in 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 *Lecture Notes in Computer Science*, pages 343–360. Springer Verlag, 1987.
7. H.-J. Kreowski and A. Wilharm. Is parallelism already concurrency? Part 2: Non-sequential processes in graph grammars. In *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, volume 291 of *Lecture Notes in Computer Science*, pages 361–377. Springer Verlag, 1987.
8. P. Padawitz. Graph grammars and operational semantics. *Theoret. Comput. Sci.*, 19:37–58, 1982.

9. B. Hoffmann and D. Plump. Implementing Term Rewriting by Jungle Evaluation. *Informatique théorique et Applications/Theoretical Informatics and Applications*, 25:445–472, 1991.
10. A. Corradini and F. Rossi. Hyperedge Replacement Jungle Rewriting for Term Rewriting Systems and Logic Programming. *Theoret. Comput. Sci.*, 109:7–48, 1993.
11. P. Böhm, H.-R. Fonio, and A. Habel. Amalgamation of graph transformations: a synchronization mechanism. *Journal of Computer and System Science*, 34:377–408, 1987.
12. H. Ehrig, P. Böhm, U. Hummert, and M. Löwe. Distributed parallelism of graph transformation. In *13th Int. Workshop on Graph Theoretic Concepts in Computer Science*, volume 314 of *Lecture Notes in Computer Science*, pages 1–19. Springer Verlag, 1988.
13. H.J. Schneider. Describing distributed systems by categorial graph grammars. In *15th International Workshop on Graph-theoretic Concepts in Computer Science*, volume 411 of *Lecture Notes in Computer Science*, pages 121–135. Springer Verlag, 1990.
14. M. Korff. Graph-interpreted graph transformations for concurrent object-oriented systems. Submitted for publication, 1995.
15. H. Ehrig and R. Bardohl. Specification techniques using dynamic abstract data types and application to shipping software. In *Proc. of the International Workshop on Advanced Software Technology*, pages 70–85, 1994.
16. M. Löwe. Implementing algebraic specifications by graph transformations. *Journal for Information Processing and Cybernetics (EIK)*, 26(11/12):615–641, 1990.
17. A. Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer Verlag, 1992.
18. F. Drewes, H.-J. Kreowski, and Habel. *Hyperedge Replacement Graph Grammars*. World Scientific, 1996. In this book.
19. H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. *Algebraic Approaches to Graph Transformation II: Single Pushout Approach and Comparison with Double Pushout Approach.* World Scientific, 1996. In this book.
20. S. Mac Lane. *Categories for the working mathematician.* Springer Verlag, 1971.
21. J. Adamek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories.* Wiley Interscience, 1990.
22. H. Ehrig, H.-J. Kreowski, and G. Taentzer. Canonical derivations for high-level replacement systems. In H.-J. Schneider and H. Ehrig, editors,

Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science, volume 776 of *Lecture Notes in Computer Science*, pages 153–169. Springer Verlag, 1994.

23. H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and Concurrency in High-Level Replacement Systems. *Mathematical Structures in Computer Science*, 1:361–404, 1991.
24. W. Reisig. *Petri Nets: An Introduction*. EACTS Monographs on Theoretical Computer Science. Springer Verlag, 1985.
25. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 1996. To appear.
26. M. Korff. True Concurrency Semantics for Single Pushout Graph Transformations with Applications to Actor Systems. In *Proceedings International Workshop on Information Systems – Corretness and Reusability, IS-CORE'94*, pages 244–258. Vrije Universiteit Press, 1994. Tech. Report IR-357.
27. M. Korff and L. Ribeiro. Concurrent Derivations as Single Pushout Graph Grammar Processes. In A. Corradini and U. Montanari, editors, *Proceedings SEGRAGRA'95 Workshop on Graph Rewriting and Computation*, volume 2 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences, 1995. `http://www.elsevier.nl/locate/entcs/volume2.html`.
28. G. Winskel. An introduction to event structures. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 325–392. Springer Verlag, 1989.
29. G. Schied. On relating rewriting systems and graph grammars to event structures. In H.-J. Schneider and H. Ehrig, editors, *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *Lecture Notes in Computer Science*, pages 326–340. Springer Verlag, 1994.
30. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. An event structure semantics for safe graph grammars. In E.-R. Olderog, editor, *Programming Concepts, Methods and Calculi*, IFIP Transactions A-56, pages 423–444. North-Holland, 1994.
31. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. An Event Structure Semantics for Graph Grammars with Parallel Productions. In *Proceedings of the Fifth International Workshop on Graph Grammars and their Application to Computer Science*, Lecture Notes in Computer Science. Springer Verlag, 1996. To appear.

32. A. Schüerr. *Programmed Graph Replacement Systems.* World Scientific, 1996. In this book.
33. R. Heckel, J. Müller, G. Taentzer, and A. Wagner. Attributed graph transformations with controlled application of rules. In G. Valiente and F. Rossello Llompart, editors, *Proc. Colloquium on Graph Transformation and its Application in Computer Science.* Technical Report B - 19, Universitat de les Illes Balears, 1995.
34. A. Corradini and R. Heckel. A Compositional Approach to Structuring and Refinement of Typed Graph Grammars. In A. Corradini and U. Montanari, editors, *Proceedings SEGRAGRA'95 Workshop on Graph Rewriting and Computation*, volume 2 of *Electronic Notes in Theoretical Computer Science.* Elsevier Sciences, 1995. `http://www.elsevier.nl/locate/entcs/volume2.html`.
35. R. Heckel, A. Corradini, H. Ehrig, and M. Löwe. Horizontal and Vertical Structuring of Graph Transformation Systems. *Mathematical Structures in Computer Science*, 1996. To appear.
36. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and J. Padberg. The Category of Typed Graph Grammars and their Adjunction with Categories of Derivations. In *Proceedings of the Fifth International Workshop on Graph Grammars and their Application to Computer Science*, Lecture Notes in Computer Science. Springer Verlag, 1996. To appear.
37. M. Korff. Single pushout transformations of equationally defined graph structures with applications to actor systems. In *Proc. Graph Grammar Workshop Dagstuhl 93*, pages 234–247. Springer, 1994. Lecture Notes in Computer Science 776.
38. R. Heckel and A. Wagner. Ensuring consistency of conditional graph grammars – a constructive approach. In *Proc. of SEGRAGRA'95 "Graph Rewriting and Computation", Electronic Notes of TCS, volume 2.* Elsevier Science, 1995.
39. Detlef Plump. Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence. In M.R Sleep, M.J. Plasmeijer, and M. C.J.D. van Eekelen, editors, *Term Graph Rewriting*, pages 201–214. Wiley, 1993.
40. D. Plump. On termination of graph rewriting. In *Proc. 21st Workshop on Graph-Theoretic Concepts in Computer Science (WG '95).* LNCS, Springer Verlag, 1995. to appear.
41. M. Löwe and J. Müller. Critical pair analysis in single-pushout graph rewriting. In G. Valiente Feruglio and F. Rosello Llompart, editors, *Proc. Colloquium on Graph Transformation and its Application in Computer Science.* Technical Report B-19, Universitat de les Illes Balears, 1995.

42. H. Ehrig and M. Löwe. Categorical principles, techniques and results for high-level replacement systems in computer science. *Applied Categorical Structures*, 1(1):21–50, 1993.
43. M. Löwe. *Extended algebraic graph transformation*. PhD thesis, Technische Universität Berlin, 1990.
44. M. Korff. *Generalized graph structure grammars with applications to object-oriented systems*. PhD thesis, Technical University of Berlin, 1995.
45. J.A. Goguen. A categorical manifesto. *Math. Struc. Comput. Sci.*, 1, 1991.
46. H.-J. Kreowski. *Manipulation von Graphmanipulationen*. PhD thesis, Technische Universität Berlin, 1977.
47. D.B. Benson. The basic algebraic structures in categories of derivations. *Info. and Co.*, 28:1–29, 1975.
48. J. Meseguer and U. Montanari. Petri Nets are Monoids. *Info. and Co.*, 88:105–155, 1990.
49. P. Degano, J. Meseguer, and U. Montanari. Axiomatizing Net Computations and Processes. In *Proceedings 4th Annual Symposium on Logic in Computer Science*, pages 175–185, 1989.
50. G. Ferrari and U. Montanari. Towards the Unification of Models for Concurrency. In *Proceedings CAAP 1990*, volume 431 of *Lecture Notes in Computer Science*, pages 162–176. Springer Verlag, 1990.
51. A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its application to logic programs. *Theoret. Comput. Sci.*, 103:51–106, 1992.
52. A. Corradini and A. Asperti. A categorical model for logic programs: Indexed monoidal categories. In *Proceedings REX Workshop, Beekbergen, The Netherlands, June 1992*, volume 666 of *Lecture Notes in Computer Science*. Springer Verlag, 1993.
53. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Note on Standard Representation of Graphs and Graph Derivations. In H.-J. Schneider and H. Ehrig, editors, *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *Lecture Notes in Computer Science*, pages 104–118. Springer Verlag, 1994.
54. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Abstract Graph Derivations in the Double-Pushout Approach. In H.-J. Schneider and H. Ehrig, editors, *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *Lecture Notes in Computer Science*, pages 86–103. Springer Verlag, 1994.
55. H. Ehrig. Aspects of Concurrency in Graph Grammars. In H. Ehrig, M. Nagl, and G. Rozenberg, editors, *Proceedings of the 2nd International*

Workshop on Graph-Grammars and Their Application to Computer Science, volume 153 of *Lecture Notes in Computer Science*, pages 58–81. Springer Verlag, 1983.
56. M. Löwe and J. Dingel. Parallelism in single-pushout graph rewriting. *Lecture Notes in Computer Science 776*, pages 234–247, 1994.
57. E. Ehrig and B. K. Rosen. Parallelism and concurrency of graph manipulations. *Theoretical Computer Science*, 11:247–275, 1980.
58. G. Taentzer. Hierarchically distributed graph transformation. In *Proc. of 5. Int. Workshop on Graph Grammars and Their Applications to Computer Science, LNCS, Springer, Berlin*, 1995. accepted for publication.
59. G. Taentzer. *Parallel and Distributed Graph Transformation: Formal Description and Application to Communication-Based Systems*. PhD thesis, Technical University Berlin, FB13, 1996.
60. H. Ehrig and M. Löwe. Parallel and distributed derivations in the single pushout approach. *TCS*, 109:123 – 143, 1993.