ALGEBRAIC APPROACHES TO GRAPH TRANSFORMATION PART II: SINGLE PUSHOUT APPROACH AND COMPARISON WITH DOUBLE PUSHOUT APPROACH

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

A. CORRADINI
Dipartimento di Informatica, Corso Italia 40,
I-56125 Pisa, Italy

Abstract

The algebraic approaches to graph transformation are based on the concept of gluing of graphs corresponding to pushouts in suitable categories of graphs and graph morphisms. This allows one to give not only 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 the previous chapter we have presented an overview of the basic notions and problems common to the two algebraic approaches, the double-pushout (DPO) approach and the single-pushout (SPO) approach, and their solutions in the DPO-approach. In this chapter we introduce the SPO-approach to graph transformation and some of its main results. We study application conditions for graph productions and the transformation of more general structures than graphs in the SPO-approach, where similar generalizations have been or could be studied also in the DPO-approach. Finally, we present a detailed comparison of the DPO- and the SPO-approach, especially concerning the solutions to the problems discussed for both approaches in the previous chapter.

1 Introduction

The algebraic approach of graph grammars has been invented at TU Berlin in the early 70'ies by H. Ehrig, M. Pfender and H.J. Schneider in order to generalize Chomsky grammars from strings to graphs [1]. Today this algebraic approach is called double-pushout approach, because it is based on two pushout constructions, in contrast to the single-pushout (SPO) approach where direct derivations are defined by a single pushout in the category of graphs and partial graph morphisms. A general introduction with main application areas and an overview of both algebraic approaches is presented in [2], in this handbook.

The main aim of this chapter is to provide an introduction to the basic notions of the SPO-approach, to present some of its main results and relevant extensions and generalizations, and to compare the notions and results of the SPO-approach with the corresponding ones presented in chapter [2] (in this handbook) for the DPO-approach. After this introduction, we suggest to read first Section I.2 [a], where many relevant concepts and results common to both algebraic approaches are introduced informally, in terms of problems.

The following section gives an introduction to the basic notions of the SPO-approach, like production, match, and (direct) derivation, and describes the historical roots of the SPO-approach. In Section 3 we give answers to the problems of Section I.2, concerning independence and parallelism, embedding of derivations, and amalgamation and distribution of derivations. In several graph grammar approaches it is possible to formulate application conditions for individual productions. But in very few cases the theoretical results can be extended to include these conditions. In Section 4 the SPO-approach and the results concerning independence and parallelism are extended in order to allow for user-defined application conditions, as recently done in [3]. In Section 5 we show how the SPO-approach can be applied to more general kinds of graphs and structures like attributed graphs [4], graph structures [5] (including for example hypergraphs), generalized graph structures [6], and even more general structures in high-level replacement systems [7]. In Section 6 we compare the main concepts and results of the two approaches, presented in [2] and this chapter, w.r.t. the problems stated in the overview of both approaches in Section I.2. Finally in the conclusion we point out that both approaches are suitable for different application areas and discuss some main ideas for further research.

2 Graph Transformation Based on the SPO Construction

In this section we introduce the basic notions of the single-pushout approach to graph transformation. In order to be able to follow the formal definitions of this section the reader should be familiar with the notions introduced in Section I.3 for the double-pushout approach, although the informal motivation in the following can be understood without any preliminary knowledge.

2.1 Graph Grammars and Derivations in the SPO Approach

As introductory example we use the model of a (simple version of a) Pacman game.

[a] All along this chapter, references to [2] are preceded by "I." (for Part I).

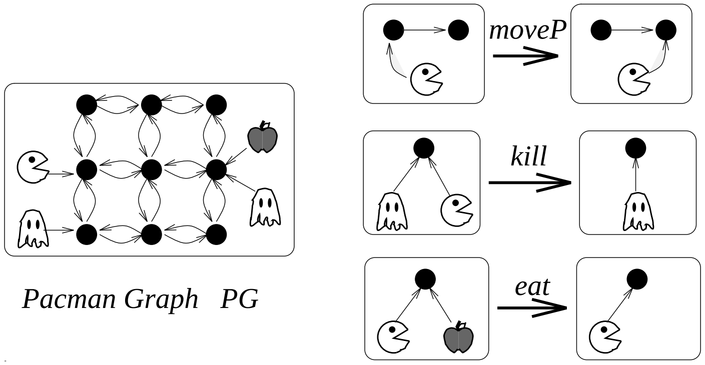


Figure 1: Pacman Game

Example 1 (Pacman game). The game starts with a number of figures (Pacman, ghosts, and apples) placed on a board. Pacman and each of the ghosts may autonomously move from field to field (up, down, left, right). Apples do not move. Pacman wins if he manages to eat all the apples before he is killed by one of the ghosts.

Each of the states of the Pacman game can easily be modeled as a graph (see Figure 1). Vertices are used to represent Pacman, ghosts and apples as well as the fields of the board. These different kinds of vertices are distinguished using different graphical layouts (the obvious ones for Pacman, ghosts and apples and the black dots for the fields). Edges pointing from a vertex representing a figure to a field vertex model the current position of the figure on the board. Neighborhood relations between fields, are explicitly represented using edges too.

Each possible activity in the Pacman game causes a change of the current state and hence is modeled by a transformation of the graph representing this state. To describe which activities, i.e., which graph transformations, are possible we use productions. The production *moveP* in Figure 1 describes that Pacman is moving one field. All objects are preserved but the edge pointing from the Pacman vertex to the field vertex is deleted and a new one is inserted. The production can be applied to a graph if Pacman is on a field which has a neighbor. As the effect of the application of this production, Pacman moves to a new position while everything else is left unchanged. Such an application of a production to a current graph is called derivation or transformation. To move the ghosts we have a production (*moveG*) which is not drawn but which follows the same scheme. The production *eat* can be applied when Pacman and an apple are on the same field. The result is that the apple and its outgoing edge are deleted. Analogously if Pacman and a ghost are on the same field the production *kill* can be applied and Pacman is deleted. □

Graphs and (total) graph morphisms have been formally introduced in Definition I.6. In contrast to spans of total morphisms in the DPO-approach, we use partial morphisms in order to describe, for example, the relationship between the given and derived graph in a derivation. The aim is to model a direct derivation by a single pushout instead of two.

Definition 1 (partial graph morphism). Let $G = (G_V, G_E, s^G, t^G, lv^G, le^G)$ be a graph. Recall that G_V, G_E denote the sets of vertices and edges of G, s^G, t^G its source and target mappings, and lv^G, le^G the label assignments for vertices and edges, respectively. A **subgraph** S of G, written $S \subseteq G$ or $S \hookrightarrow G$, is a graph with $S_V \subseteq G_V$, $S_E \subseteq G_E$, $s^S = s^G|_{S_E}$, $t^S = t^G|_{S_E}$, $lv^S = le^G|_{S_E}$, and $le^S = le^G|_{S_E}$. A **(partial) graph morphism** g from G to H is a total graph morphism from some subgraph $dom(g)$ of G to H, and $dom(g)$ is called the **domain** of g. □

By defining composition of these morphisms by composition of the components, and identities as pairs of component identities[b], the graphs over a fixed labeling alphabet and the partial morphisms among them form a category denoted by $\boldsymbol{Graph^P}$.

Usually, a production $L \xrightarrow{p} R$ consists of two graphs L and R, called the left- and the right-hand side, respectively, and a partial morphism p between them. The idea is to describe in the left-hand side which objects a graph must contain such that the production can be applied, i.e., all objects (vertices and edges) which shall be deleted and the application context which shall be preserved. The right-hand side describes how this part of the graph shall look like after the transformation, i.e., it consists of all the objects which are added together with the application context. The morphism p makes clear which objects in the left-hand side correspond to which ones in the right-hand side. Objects where the morphism is undefined are deleted. All objects which are preserved by the morphism form the application context. If an object in the right-hand side of the production has no preimage under the morphism, it is added. Note that the role of the application context coincides with that of the interface graph K in the DPO productions (see Definition I.7). In the formal definition below the concept of production is enriched with a name, which is used to describe the internal structure of composed productions as, for example, the parallel production (see also the corresponding Definition I.7 for the DPO-approach).

Definition 2 (production, graph grammar). A **production** $p : (L \stackrel{r}{\to} R)$ consists of a **production name** p and of an injective partial graph morphism

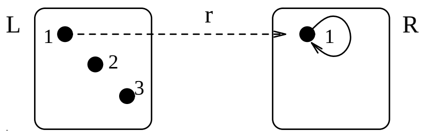
[b] Note that a partial morphism can also be considered as a pair of partial functions.

r, called the **production morphism**. The graphs L and R are called the left- and the right-hand side of p, respectively. If no confusion is possible, we will sometimes make reference to a production $p : (L \xrightarrow{r} R)$ simply as p, or also as $L \xrightarrow{r} R$.

A **graph grammar** $\mathcal{G}$ is a pair $\mathcal{G} = \langle (p : r)_{p \in P}, G_0 \rangle$ where $(p : r)_{p \in P}$ is a family of production morphisms indexed by production names, and G_0 is the **start graph** of the grammar. □

Since production names are used to identify productions, for example, in direct derivations, they should be unique, i.e., there must not be two different productions with the same name. This is ensured be the above definition of graph grammar. Later on, production names will be used to store the structure of productions which are composed from elementary productions. In such cases the production name is usually a diagram, consisting of the elementary productions and their embeddings in the composed production. In elementary productions, production names are sometimes disregarded. Then we just write $p : L \to R$ in order to denote the production $p : (L \xrightarrow{p} R)$ where the production morphism p is also used as the name of the production. In this section, all the productions are elementary and we use the notation just introduced.

Example 2 (production). The figure below shows a production r which inserts a loop and deletes two vertices ($\bullet_2$ and $\bullet_3$ resp.). We consider the labeling alphabet to contain only one element $*$, i.e., the graph can be seen as unlabeled. The application context consists of the vertex $\bullet_1$. As indicated by the dashed line, r is defined for vertex $\bullet_1$ mapping it to the only vertex on the right hand side. r is undefined for the other two vertices. The numbers are used to illustrate the morphism such that we can omit the dashed lines later on if graphs are more complex. Hence graph L can for example be formally denoted by $L = (\{\bullet_1, \bullet_2, \bullet_3\}, \emptyset, s^L, t^L, lv^L, le^L)$, where s^L, t^L and le^L are empty functions. lv^L is defined by $lv^L(\bullet_i) = *$ for $i = 1..3$.



□

Using partial instead of total morphisms, the categorical notion of pushout captures not only the intuitive idea of gluing but that of gluing and deletion. Therefore the construction is much more complicated. In order to make it easier to understand we divided it into three steps. The first two steps are gluings as known from Section I.3. In the third step the deletion is performed. Intuitively, deletion is not seen as inverse to addition here, but as equalizing morphisms having different domains of definition. Two morphisms with the same source and target objects are reduced to their equal kernel by removing all those items from their range which have different preimages under the morphisms. Formally this is captured by the categorical notion of a co-equalizer.

Definition 3 (co-equalizer). Given a category C and two arrows $a : A \to B$ and $b : A \to B$ of C, a tuple $\langle C, c : B \to C \rangle$ is called **co-equalizer** of $\langle a, b \rangle$ if $c \circ a = c \circ b$ and for all objects D and arrows $d : B \to D$, with $d \circ a = d \circ b$, there exists a unique arrow $u : C \to D$ such that $u \circ c = d$.

$$\begin{tikzcd}
A \arrow[r, shift left, "a"] \arrow[r, shift right, "b"'] & B \arrow[r, "c"] & C
\end{tikzcd}$$

□

For our pushout construction of partial morphism a very specific co-equalizer is sufficient, which is constructed in the following.

Construction 4 (specific co-equalizer in *Graph*$^{\boldsymbol{P}}$). *Let $a, b : A \to B$ be two (partial) morphisms such that for each $x \in A$, if both a and b are defined on x then $a(x) = b(x)$. Then, the co-equalizer of a and b in **Graph**$^{\boldsymbol{P}}$ is given by $\langle C, c \rangle$ where*

- *$C \subseteq B$ is the largest subgraph of $[b(A) \cap a(A)] \cup [(\overline{a(A)} \cap \overline{b(A)}])$, and*
- *$dom(c) = C$ and c is the identity morphism on its domain.*

Proof. It is obvious that $c \circ a = c \circ b$. The universal co-equalizer property follows from the fact that for each other morphism $d : B \to D$ with $d \circ a = d \circ b$ we have $dom(d) \subseteq dom(c)$. □

Example 3 (deletion as a co-equalizer). Figure 2 shows the co-equalizer of the empty morphism a and a total morphism b in ***Graph***$^{\boldsymbol{P}}$. According to Construction 4, C is the maximal subgraph of B such that for all $x \in C$, either $x \in a(A) \cap b(A)$, or $x \notin a(A)$ and $x \notin b(A)$. Thus, vertex 2 is deleted because it has a preimage under b but not under a. In order to obtain a graph, the dangling edge is deleted, too. □

With these preliminaries we are able to construct the pushout of two partial morphisms in ***Graph***$^{\boldsymbol{P}}$.

Proposition 5 (pushout in *Graph*$^{\boldsymbol{P}}$). *The pushout of two morphisms $b : A \to B$ and $c : A \to C$ in **Graph**$^{\boldsymbol{P}}$ always exists and can be computed in three steps, as shown in Figure 3:*

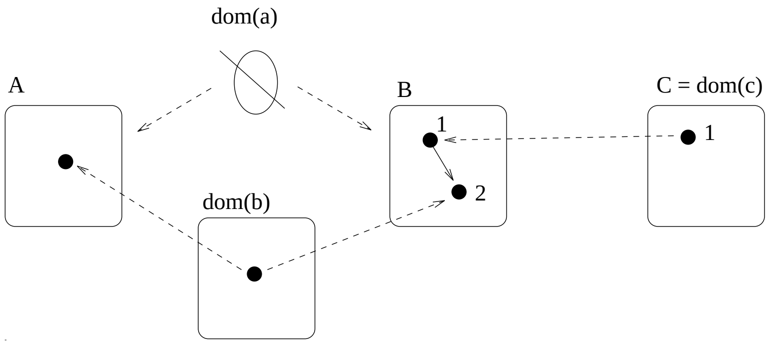


Figure 2: Deletion as a co-equalizer

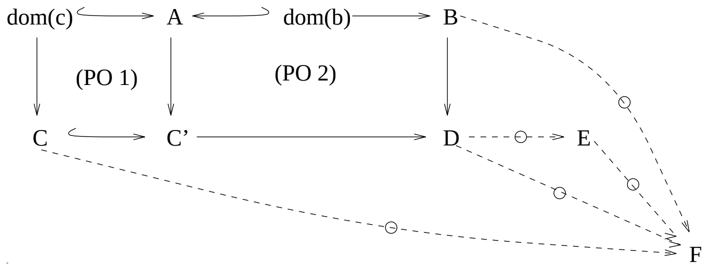


Figure 3: Construction of pushout in $\boldsymbol{Graph^P}$

[gluing 1] *Construct the pushout $\langle C', A \to C', C \to C'\rangle$ of the total morphisms $dom(c) \to C$ and $dom(c) \to A$ in* ***Graph****(cf. Example 1.3).*

[gluing 2] *Construct the pushout $\langle D, B \to D, C' \to D\rangle$ of the total morphisms $dom(b) \to A \to C'$ and $dom(b) \to B$ in* ***Graph****.*

[deletion] *Construct the co-equalizer $\langle E, D \to E\rangle$ of the partial morphisms $A \to B \to D$ and $A \to C \to C' \to D$ in* $\boldsymbol{Graph^P}$*.*

$\langle E, C \to C' \to D \to E, B \to D \to E\rangle$ is the pushout of b and c in $\boldsymbol{Graph^P}$*.*

Proof. Commutativity holds by construction. Assume that there is $\langle F, C \to F, B \to F\rangle$ with $A \to B \to F = A \to C \to F$. Prove that $dom(b) \to B \to F = dom(b) \to A \to C' \to F$. Use the fact that pushouts in **Graph**are pushouts in $\boldsymbol{Graph^P}$ [5] to construct a universal morphism $D \to F$. With the universal co-equalizer property we obtain a unique morphism $E \to F$ with $B \to D \to E \to F = B \to F$ and $C \to C' \to D \to E \to F = C \to F$. □

To apply a production to a graph G, one has to make sure that G contains at least all objects necessary for performing the desired operations and the application context. These are exactly the objects of the production's left-hand side L. Hence we introduce the notion of a match as a total morphism matching the left-hand side of the production with (a part of) G. The match must be total because otherwise not all needed objects have a corresponding image in the graph G. We allow that different items from L are mapped onto the same item in G. This avoids additional productions in some cases. The actual transformation of G is performed by removing the part matched by the production's left-hand side and adding the right-hand side. The result is the derived graph H.

$$\begin{tikzcd}
L \arrow[r, "r"] \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & R \arrow[d, "m^*"] \\
G \arrow[r, "r^*"'] & H
\end{tikzcd}$$

Figure 4: Direct derivation as pushout in $\boldsymbol{Graph^P}$.

Definition 6 (match, derivation). A **match** for $r : L \to R$ in some graph G is a total morphism $m : L \to G$. Given a production r and a match m for r in a graph G, the **direct derivation** from G with r at m, written $G \overset{r,m}{\Longrightarrow} H$, is the pushout of r and m in $\boldsymbol{Graph^P}$, as shown in Figure 6. A sequence of direct derivations of the form $\rho = (G_0 \overset{r_1,m_1}{\Longrightarrow} \ldots \overset{r_k,m_k}{\Longrightarrow} G_k)$ constitutes a **derivation** from G_0 to G_k by $r_1, \ldots, r_k$, briefly denoted by $G_0 \Longrightarrow^* G_k$. The **graph language** generated by a graph grammar $\mathcal{G}$ is the set of all graphs G_k such that there is a derivation $G_0 \Longrightarrow^* G_k$ using productions of $\mathcal{G}$. □

To construct the direct derivation from graph G with r at m we use Proposition 5. Since match m is a total morphism, step one of the construction is trivial and can be omitted.

Example 4 (direct derivation as pushout construction). The production $moveP$ from Example 1 can be applied to the graph PG (Figure 1) at match m, leading to the direct derivation shown in Figure 5. Remember that due to the totality of the match the first step of the construction given in Proposition 5 is trivial. Hence we start with the second step: An edge between Pacman and the vertex modeling the field he moves to is added by performing the pushout of the total morphisms $dom(moveP) \to L_{moveP} \to PG$ and $dom(moveP) \to R_{moveP}$. Afterwards the co-equalizer of $L_{moveP} \to R_{moveP} \to PG'$ and $L_{moveP} \to PG \to PG'$ leads to the derived graph PH which is the subgraph of PG' where the edge between Pacman and the vertex representing his old position is deleted. □

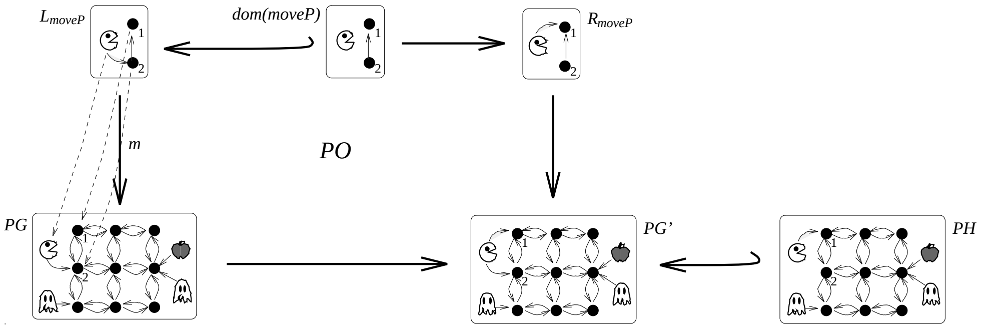


Figure 5: Direct derivation (example).

In the example derivation above deletion is rather intuitive. In contrast Figure 6 depicts a situation where a vertex is meant to be preserved as well as deleted on the left. The result of this derivation is the empty graph, i.e., deletion has priority. This property of the approach is caused by the order of the construction steps: first gluing of conflicting items and then their deletion. Similarly the problem of the dangling edge on the right of Figure 6 is solved: it is deleted, too. This is illustrated in Example 3.

In the following we will characterize simple deletion by properties of the match. If a match is *d(elete)-injective* then conflict situations are avoided. If it is *d-complete*, deletion does not cause the implicit deletion of dangling edges.

Definition 7 (special matches). Given a match $m : L \to G$ for a production $r : L \to R$. Then m is called **conflict-free** if $m(x) = m(y)$ implies $x, y \in dom(r)$ or $x, y \notin dom(r)$. It is called **d-injective** if $m(x) = m(y)$ implies $x = y$ or $x, y \in dom(r)$. Finally, m is **d-complete** if for each edge $e \in G_E$ with $s^G(e) \in m_V(L_V - dom(r)_V)$ or $t^G(e) \in m_V(L_V - dom(r)_V)$ we have $e \in m_E(L_E - dom(r)_E)$. □

Lemma 8 (pushout properties). *If $(H, r^* : G \to H, m^* : R \to H)$ is the pushout of $r : L \to R$ and $m : G \to H$ in $\boldsymbol{Graph^P}$, then the following properties are fulfilled:*

1. *Pushouts preserve surjectivity, i.e. r surjective implies r^* surjective.*
2. *Pushouts preserve injectivity, i.e. r injective implies r^* injective.*
3. *r^* and m^* are jointly surjective.*
4. *m conflict-free implies m^* total.*

Proof. See [8]. □

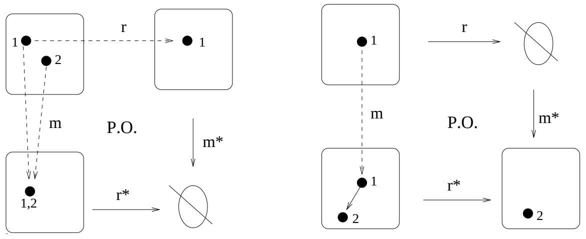


Figure 6: Deletion with conflicts.

2.2 Historical Roots of the SPO-Approach

Single-pushout transformations in a setting of some sort of partial morphisms have been investigated already by Raoult [9] and Kennaway [10]. The following historical roots are taken from [5].

Raoult [9] introduces two conceptually very different approaches. The first one is described in the category of sets and partial mappings. A rule is a partial morphism $r : L \to R$, i.e. a partial map which respects the graph structure on all objects of L it is defined for. A match $m : L \to G$ in some graph G is a total morphism of this type. The result of applying r at m is constructed in two steps. First, the pushout $(H, r^* : G \to H, m^* : R \to H)$ of r and m in the category of sets and partial maps is built. In the second step, a graph structure is established on H such that the pushout mappings r^* and m^* become morphisms. He characterizes the situations in which this graph structure uniquely exists; double-pushout transformations with gluing conditions are special cases of these situations. The second model of graph transformation in [9] uses another kind of partiality for the morphisms: a rule is a total map $r : L \to R$, which is only partially compatible with the graph structure. Let $rewrite(r)$ denote the set of objects which are not homomorphically mapped by r. A match $m : L \to G$ is total which means now $rewrite(m) = \emptyset$. Application of r at m is again defined by two steps. First construct the pushout $(H, r^* : G \to H, m^* : R \to H)$ of r and m in the category of sets and total mappings and then impose a graph structure on H such that the pushout mappings become as compatible as possible, i.e. such that $rewrite(r^*) = m(rewrite(r))$ and $rewrite(m^*) = r(rewrite(m))$. Raoult [9] gives sufficient conditions for the unique existence of this structure. This approach has the major disadvantage that objects cannot be deleted at all.

Kennaway [10] provides a categorical description for the second approach of [9]. Graphs are represented the same way. Morphisms $f : A \to B$ are pairs (f, hom). The first component is a total mapping from A to B. The second component provides a subset of A on which f respects the graph structure. A rule $r : L \to R$ is any morphism in this sense and a match $m : L \to G$ is a total morphism which now means $hom = L$. He shows that under certain conditions the two-step construction of [9] coincides with the pushout construction in the category of graphs and the so-defined morphisms.

Unfortunately, only sufficient conditions for the existence of pushouts are given. Besides that, object deletion remains impossible. The concept in [10] has been further developed in [11]. They introduce "generalized graph rewriting" which uses the same kind of graph morphism. The corresponding transformation concept not only involves a pushout construction but also a coequalizer. Since both constructions are carried out in different categories (of total resp. partial morphisms) theoretical results are difficult to obtain. The SPO-approach in [5] as discussed above is closely related to the first approach in [9]. His concept of partial mappings which are compatible with the graph structure on their domain can be generalized to a concept of partial homomorphisms on special categories of algebras such that pushout construction in the categories is always possible. Hence, we get rid of any application conditions. If, however, the necessary and sufficient conditions of [9] are satisfied, the construction of pushout objects coincides with his two-step construction.

Recently, Kennaway [12] independently started to study graph transformation in some categories of partial morphisms of this type. His work is based on the categorical formulation of a partial morphism provided by [13]. While [5] considers concrete algebraic categories, [12] stays in a purely categorical framework. Future research has to show how both approaches can benefit from each other. Van den Broek [14] introduces another kind of single-pushout transformations based on "partial" morphisms. Partiality in this framework is described by total morphisms which map objects "outside their domain" to marked objects in their codomain. In this chapter we follow the SPO-approach as introduced in [8,5] and extended in several subsequent papers mentioned below.

3 Main Results in the SPO-Approach

In this section we present some of the main results of the SPO-approach. They are concerned with the conceptual notions of parallelism, context embedding, synchronization and distribution, interpreted into the world of SPO derivations. The main reason for this choice is the fact that a most natural and very basic view on a graph transformation system is to see a graph as a system state and a production as a syntactical description of corresponding state transformations obtained by direct derivations. By further (informal) arguments certain derivations can be classified as to be concurrent, embedded into each other, or (somehow) synchronized or distributed.

Complementary, on the syntactical level, different ways of composing productions can be considered, generating an integrated description, i.e., the composed production. As for derivations, we consider parallel, derived, and synchronized/amalgamated productions. An elementary production essentially provides a description of the direct derivations it may perform. A composed production, additionally, contains all the information about its construction from elementary productions, which is stored in the production name. If, e.g., $p : r$ is a parallel production, the production name p contains the elementary productions and their embeddings into the resulting production morphism r. The (complete or partial) correspondence between composed derivations and derivations using (correspondingly) composed productions provides the essential results of this section, which may be seen as answers to the problems stated in Section I.2. Related results are mentioned briefly.

3.1 Parallelism

There are essentially two different ways to model concurrent computations, the interleaving and the truly concurrent model. This subsection provides the basic concepts of these models in the world of SPO graph rewriting, including the solutions to the Local Church-Rosser Problem I.1 and the Parallelism Problem I.2.

Interleaving

In an interleaving model two actions are considered to be concurrent (i.e., potentially in parallel) if they may occur in any order with the same result. Modeling single actions by direct derivations, we introduce two notions of independence, formalizing this concept from two different points of view. The condition of *parallel independence* shall ensure that two *alternative* direct derivations are not mutually exclusive. Safely, we may expect this if these direct derivations act on totally disjoint parts of the given graph. More precisely, a direct derivation $d_1 = (G \overset{p_1,m_1}{\Longrightarrow} H_1)$ does not affect a second $d_2 = (G \overset{p_2,m_2}{\Longrightarrow} H_2)$ if d_1 does not delete elements in G which are accessed by d_2. In other words, the overlapping of the left hand sides of p_1 and p_2 in G must not contain elements which are deleted by p_1. The vertices deleted from G by d_1 are those in $m_1(L_1 - dom(p_1))$. An edge will be deleted from G if it is in $m_1(L_1 - dom(r_1))$ or - which is an additional feature in the SPO approach - one of its incident vertices is deleted. Formally, we obtain the following definition of weakly parallel independent derivations.

Definition 9 (parallel independence). Let $d_1 = (G \overset{p_1,m_1}{\Longrightarrow} H_1)$ and $d_2 = (G \overset{p_2,m_2}{\Longrightarrow} H_2)$ be two alternative direct derivations. Then we say that d_2 is **weakly parallel independent** of d_1 iff $m_2(L_2) \cap m_1(L_1 - dom(p_1)) = \emptyset$. We call the derivations d_1 and d_2 **parallel independent** if they are mutually weakly parallel independent. □

$$\begin{tikzcd}
R_2 \arrow[d, "{m_2}^*"'] & L_2 \arrow[l, "p_2"'] \arrow[dr, "m_2"] & & L_1 \arrow[r, "p_1"] \arrow[dl, "m_1"'] & R_1 \arrow[d, "{m_1}^*"] \\
H_2 & & G \arrow[ll, "{p_2}^*"] \arrow[rr, "{p_1}^*"'] & & H_1
\end{tikzcd}$$

Weak parallel independence can be characterized in more abstract terms on a categorical level.

Characterization 10 (parallel independence). *Let $d_1 = (G \overset{p_1,m_1}{\Longrightarrow} H_1)$ and $d_2 = (G \overset{p_2,m_2}{\Longrightarrow} H_2)$ be two direct derivations. Then d_2 is weakly parallel independent of d_1 if and only if ${p_1}^* \circ m_2 : L_2 \to H_1$ is a match for p_2.*

Proof. Let there be a match $m'_2 = {p_1}^* \circ m_2$ as above. Then $m_2(L_2) \subseteq dom({p_1}^*)$ by definition of composition. The commutativity of pushouts provides $m_1(L_1 - dom(p_1)) \cap dom({p_1}^*) = \emptyset$ which is the desired weak parallel independence.

Vice versa, we must show that $m'_2 = {p_1}^* \circ m_2$ is total. According to the construction of a derivation, each vertex which is not explicitly deleted by p_1 is preserved, i.e., $v \in G - m_1(L_1 - dom(p_1))$ implies $v \in dom({p_1}^*)$. This implies that each preimage of v under m_2 has also an image in H_1. Each edge which is not explicitly deleted by p_1 is either (i) preserved or (ii) implicitly deleted by the deletion of an incident vertex i.e., for each edge $e \in G - m_1(L_1 - dom(p_1))$ we either have (i) $e \in dom({p_1}^*)$ which inherits the arguments for vertices; Otherwise, in case (ii), $s^G(e) \in m_1(L_1 - dom(p_1))$ or $t^G(e) \in m_1(L_1 - dom(p_1))$. By definition of parallel independence this implies $s^G(e) \notin m_2(L_2)$ or $t^G(e) \notin m_2(L_2)$ which, by definition of graph morphisms excludes $e \in m_2(L_2)$. In other words $e \in m_2(L_2)$ implies $e \in dom({p_1}^*)$. □

The condition of *sequential independence* shall ensure that two *consecutive* direct derivations are not causally dependent. A direct derivation is weakly sequential independent of a preceding one if it could already have been performed before that. Analogous to the parallel case above, weak sequential independence requires that the overlapping of the right hand side of the first production and the left hand side of the next must not contain elements which were generated by the first. The idea of the stronger notion of sequential independence is that additionally the second will not delete anything which was needed by the first.

Definition 11 (sequential independence). Let $d_1 = (G \overset{p_1,m_1}{\Longrightarrow} H_1)$ and $d'_2 = (H_1 \overset{p_2,m'_2}{\Longrightarrow} X)$, be two consecutive direct derivations. Then we say that d'_2 is **weakly sequentially independent** of d_1 if $m'_2(L_2) \cap {m_1}^*(R_1 - p_1(L_1)) = \emptyset$. If additionally $m'_2(L_2 - dom(p_2)) \cap {m_1}^*(R_1) = \emptyset$, we say that d'_2 is **sequentially independent** of d_1, and the derivation $(G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m'_2}{\Longrightarrow} X)$ is called **sequentially independent**. □

$$\begin{tikzcd}
L_1 \arrow[r, "p_1"] \arrow[d, "m_1"'] & R_1 \arrow[dr, "{m_1}^*"] & & L_2 \arrow[r, "p_2"] \arrow[dl, "m'_2"'] & R_2 \arrow[d, "{m'_2}^*"] \\
G \arrow[rr, "{p_1}^*"'] & & H_1 \arrow[rr, "{p_2}^*"'] & & X
\end{tikzcd}$$

Again we will provide a categorical characterization. The formulation of this statement is analogous to the case of weak parallel independence. The proof has therefore been omitted

Characterization 12 (sequential independence). *Assume two direct derivations $d_1 = (G \overset{p_1,m_1}{\Longrightarrow} H_1)$ and $d'_2 = (H_1 \overset{p_2,m'_2}{\Longrightarrow} X)$ to be given. Then d'_2 is weakly sequentially independent of d_1 iff there is a match $m_2 : L_2 \to G$ for p_2 such that $m'_2 = {p_1}^* \circ m_2$. The derivation d'_2 is sequentially independent of d_1 iff d'_2 is weakly sequentially independent of d_1 and d_1 is weakly parallel independent of the correspondingly existing derivation $d_2 = (G \overset{p_2,m_2}{\Longrightarrow} H_2)$.* □

By definition, the weakly parallel independence of two direct derivations implies the existence of consecutive direct derivations. The definition of weakly sequential independent derivations contains a symmetric implication. The argumentation could be summarized in the following way: weak parallel independence allows to delay a derivation — complementary, weak sequential independence allows to anticipate a derivation. Formally this is captured by the following lemma.

Lemma 13 (weak independence). *Given a direct derivation $d_1 = (G \overset{p_1,m_1}{\Longrightarrow} H_1)$, the following statements are equivalent:*

1. *There is a direct derivation $d_2 = (G \overset{p_2,m_2}{\Longrightarrow} H_2)$ which is weakly parallel independent of d_1.*
2. *There is a direct derivation $d'_2 = (H_1 \overset{p_2,m'_2}{\Longrightarrow} X)$ which is weakly sequentially independent of d_1.*

Up to isomorphism, a bijective correspondence between 1. and 2. above is given by $m'_2 = {p_1}^* \circ m_2$ and $m_2 = ({p_2}^*)^{-1} \circ m'_2$.

$$\begin{tikzcd}
 & L_1 \arrow[r, "p_1"] \arrow[d, "m_1"'] \arrow[dr, phantom, "(1)"] & R_1 \arrow[d, "m_1^*"] \\
L_2 \arrow[r, "m_2"] \arrow[d, "p_2"'] \arrow[dr, phantom, "(2)"] & G \arrow[r, "p_1^*"] \arrow[d, "p_2^*"] \arrow[dr, phantom, "(3)"] & H_1 \arrow[d, "m_1^{**}"] \\
R_2 \arrow[r, "m_2^*"] & H_2 \arrow[r, "m_1^{**}"] & X
\end{tikzcd}$$

Figure 7: Local Church Rosser

Proof. This is a direct consequence of Characterizations 10 and 12 together with the fact that, by Lemma 8, injectivity of productions implies injectivity of ${p_1}^*$ and thus $m_2 = ({p_1}^*)^{-1} \circ {p_1}^* \circ m_2$ as well as $m_2' = {p_1}^* \circ ({p_1}^*)^{-1} \circ m_2'$. □

According to our interpretation, the conditions of parallel and sequential independence formalize the concepts of concurrent computation steps. In a more abstract sense, two such steps are concurrent if they may be performed in any order with the same result. The following Local Church-Rosser Theorem shows the correctness of the characterization of concurrent direct derivations by their parallel and sequential independence. It provides the solution to the Local Church-Rosser Problem I.1.

Theorem 14 (local Church-Rosser). *Let* $d_1 = (G \overset{p_1,m_1}{\Longrightarrow} H_1)$ *and* $d_2 = (G \overset{p_2,m_2}{\Longrightarrow} H_2)$ *be two direct derivations. Then the following statements are equivalent:*

1. *The direct derivations* d_1 *and* d_2 *are parallel independent.*

2. *There is a graph* X *and direct derivations* $H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ *and* $H_2 \overset{p_1,m_1'}{\Longrightarrow} X$ *such that* $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ *and* $G \overset{p_2,m_2}{\Longrightarrow} H_2 \overset{p_1,m_1'}{\Longrightarrow} X$ *are sequentially independent derivations.*

Up to isomorphism, a bijective correspondence between 1. and 2. above is given by $m_2' = {p_1}^* \circ m_2$ *and* $m_1' = {p_2}^* \circ m_1$.

Proof. Consider the diagram in Figure 7. Subdiagrams (1) and (2) depict the derivations d_1 and d_2, respectively. Subdiagram (3) represents the pushout of ${p_1}^*$ and ${p_2}^*$. The composition of pushout diagrams (2) and (3) yields a pushout diagram (2)+(3) which is a derivation $H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ provided that $m_2' = {p_1}^* \circ m_2$ is a match, i.e., total. But this is ensured since d_1 and d_2 have been required to be parallel independent. Analogously we obtain a derivation $H_2 \overset{p_1,m_1'}{\Longrightarrow} X$ by composing pushout diagrams (1) and (3). The stated sequential independence and the bijective correspondence follow from Lemma 12. □

Explicit Parallelism

In contrast to the interleaving model above, a *truly* parallel computation step has to abstract from any possible interleaving order, i.e., it must not generate any intermediate state. Therefore, a parallel direct derivation is a *simultaneous* application of productions, which are combined into a single *parallel production*. Constructing this production as the disjoint union of the given elementary productions reflects the view that the overall effect of a parallel production application can be described by a 'most independent' combination of both component descriptions. The formal definition below uses the fact that the disjoint union of graphs (obtained from the disjoint union of carrier sets) can categorically be characterized by a coproduct. The construction of the parallel production as a coproduct of elementary productions is recorded in the production name $p_1 + p_2$, which is given by the coproduct diagram below. This notation is well-defined, i.e., the diagram below is uniquely determined by the term $p_1 + p_2$, if we fix a certain coproduct construction (like, for example, the disjoint union).

$$\begin{tikzcd}
L_1 \arrow[rr, "p_1"] \arrow[dr, "in_L^1"'] & & R_1 \arrow[drr, "in_R^1"] & L_2 \arrow[rr, "p_2"] \arrow[dll, "in_L^2"'] & & R_2 \arrow[dl, "in_R^2"] \\
 & L_1 + L_2 \arrow[rrr, "p"] & & & R_1 + R_2 &
\end{tikzcd}$$

Definition 15 (parallel productions and derivations). Given two productions $p_1 : L_1 \to R_1$ and $p_2 : L_2 \to R_2$, the **parallel production** $p_1 + p_2 : (L_1 + L_2 \xrightarrow{p} R_1 + R_2)$ is composed of the production name $p_1 + p_2$, i.e., the diagram above, and the associated partial morphism p. The graphs $L_1 + L_2$ and $R_1 + R_2$ (together with the corresponding injections in_L^1, in_L^2, in_R^1, and in_R^2) are the coproducts of L_1, L_2 and R_1, R_2 respectively. The partial morphism $L_1 + L_2 \xrightarrow{p} R_1 + R_2$ is induced uniquely by the universal property of the coproduct $L_1 + L_2$ such that $p \circ in_L^1 = in_R^1 \circ p_1$ and $p \circ in_L^2 = in_R^2 \circ p_2$. The application of a parallel production $p_1 + p_2$ at a match m constitutes a **direct parallel derivation**, denoted by $G \overset{p_1+p_2,m}{\Longrightarrow} X$. By referring to morphisms $m_1 = m \circ in_L^1$ and $m_2 = m \circ in_L^2$ we also write $G \overset{p_1+p_2,m_1+m_2}{\Longrightarrow} X$. □

Direct parallel derivations provide us with an explicit notion of parallelism, which shall now be related to the interleaving model that has been formulated before. The Parallelism Problem I.2 asked for conditions allowing the sequentialization of a parallel derivation, and the parallelization of a sequential one. The following Weak Parallelism Theorem answers these questions in the SPO-approach.

$$\begin{tikzcd}
 & & L_2 \arrow[rrr, "p_2"] \arrow[ddl, "in_L^2"'] \arrow[dddl, "m_2"] & & & R_2 \arrow[ddl, dashed, "in_R^2"'] \arrow[dddl, dashed, "m_2^*"] \\
L_1 \arrow[rrr, "p_1"] \arrow[dr, "in_L^1"] \arrow[ddr, "m_1"'] & & & R_1 \arrow[dr, dashed, "in_R^1"] \arrow[ddr, dashed, "m_1^*"'] & & \\
 & L_1+L_2 \arrow[rrr, "p"] \arrow[d, "m_1+m_2"] & & & R_1+R_2 \arrow[d, dashed, "(m_1+m_2)^*"] & \\
 & G \arrow[rrr, dashed, "p^*"'] & & & X &
\end{tikzcd}$$

Figure 8: Weak Parallelism

Theorem 16 (weak parallelism). *Given productions $p_1 : L_1 \to R_1$ and $p_2 : L_2 \to R_2$, the following statements are equivalent:*

1. *There is a direct parallel derivation $G \overset{p_1+p_2,m_1+m_2}{\Longrightarrow} X$ such that $G \overset{p_2,m_2}{\Longrightarrow} H_2$ is weakly parallel independent of $G \overset{p_1,m_1}{\Longrightarrow} H_1$.*
2. *There is a weakly sequential independent derivation $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$.*

Up to isomorphism, a bijective correspondence between 1. and 2. above is given by $m_2' = {p_1}^* \circ m_2$.

Proof. Constructing the parallel derivation means first to construct the colimits (coproducts) for in_L^1 and in_L^2 as well as for in_R^1 and in_R^2 then the colimit (pushout) of p and $m_1 + m_2$ in a second step. In other words it means to construct the colimit of the diagram given by p_1, m_1 and p_2, m_2 as depicted in the Figure 8 above. Consider now Figure 7 of the Local Church Rosser Theorem. Also this shows a colimit constructed from the diagram given by p_1, m_1 and p_2, m_2. Due to the commutativity of colimit construction (see [15]) ensuring that all colimits can iteratively be obtained from composing partial colimits, we conclude that both of these constructions coincide. [c] So, the result of the parallel derivation can equivalently be obtained by first constructing the colimits (pushout) of p_1, m_1 and p_2, m_2 and second the colimit (pushout) for the resulting morphisms ${p_1}^*$ and ${p_2}^*$. But this means first to construct the direct derivations $d_1 = (G \overset{p_1,m_1}{\Longrightarrow} H_1)$ and $d_2 = (G \overset{p_2,m_2}{\Longrightarrow} H_2)$ represented by sub-diagrams (1) and (2) in Figure 7 respectively. Their preassumed weak parallel independence leads then to a derivation $d_2' = (H \overset{p_2,m_2'}{\Longrightarrow} X)$ represented by subdiagram (2)+(3). By definition 10 we finally observe that d_2' is weakly sequentially independent of d_1 as required.

[c] i.e. both colimits may at most differ up to isomorphisms, but each obtained in one way can also be obtained in the other way.

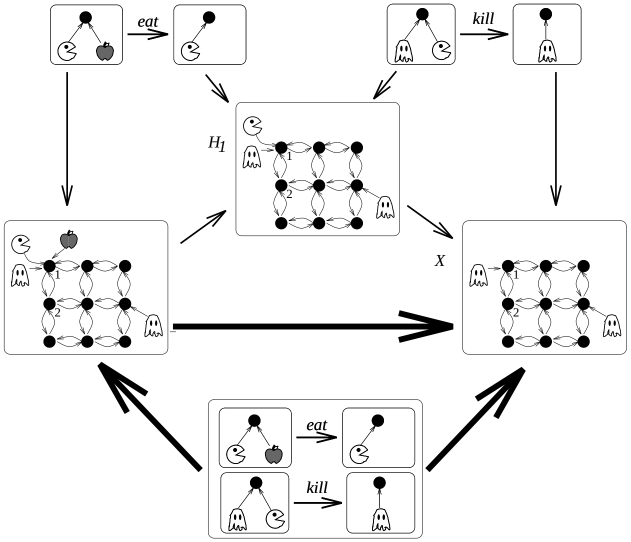


Figure 9: Killing and eating can be parallelized

Vice versa, given an arbitrary sequentially independent derivation, we observe that all the arguments above can uniquely be reversed. The bijective correspondence between 1. and 2. is due to Lemma 13. □

Before looking at the counter-example below, the typical situation of Theorem 16 shall be illustrated by the Pacman-game.

Example 5 (killing is weakly parallel independent of eating.). In Figure 9 a situation is shown where Pacman, a ghost and an apple are on the same field. We observe that two production applications are possible: Pacman may eat an apple and the ghost may kill Pacman. Weak parallel independence allows the ghost to be merciful. Pacman may eat his last apple; nevertheless, the ghost will in both situations G and H_1 be sure about his victim. In other words, killing is weakly independent of eating. Contrastingly, eating is not at all weakly independent of killing because killing implies that eating becomes impossible. The Weak Parallelism Theorem 16 now ensures that the procedure of eating the last apple and being killed can be shortcut by correspondingly applying the parallel production. □

The following example proves that there are parallel derivations which cannot be sequentialized.

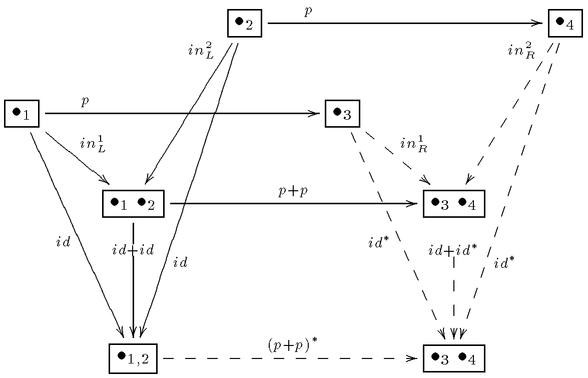


Figure 10: A Parallel Derivation which cannot be sequentialized

Example 6 (non-sequentializable parallel derivations). Consider a production $\emptyset : L \to R$ where both L and R contain exactly one vertex $\bullet$.

Let $L \overset{\emptyset+\emptyset, id+id}{\Longrightarrow} X$ be a parallel derivation with $\emptyset + \emptyset$. Note that the two component matches $m_1 = m_2 = id : L \to L$ are parallel dependent since the vertex in L is meant to be deleted. However, the derived graph X contains two vertices. Clearly this effect cannot be obtained by applying $\emptyset$ in two sequential steps. Let us compare this formally obtained result with our intuition: We observe that $\emptyset$ deletes a vertex and re-generates it afterwards. Correspondingly, the parallel production deletes two vertices and re-generates two. Hence we may expect that a match by which the two vertices in $L + L$ are identified leads to a co-match $(m_1 + m_2)^* : R + R \to X$ which identifies the two vertices. However, this does not happen in the parallel derivation which is due to the fact that, formally, the vertices in L and R are completely unrelated (i.e., there is no formal notion of 're'-generation). Hence, the two applications of $\emptyset$ generate two different vertices. □

Additional Remarks: The Weak Parallelism Theorem allows to define an operation on derivations, which shifts the application of a certain production one step towards the beginning of the derivation. Iterated applications of this so-called *shift operation* lead to a derivation in which each production is applied as early as possible. This maximally parallel derivation is introduced in [16] as the *canonical derivation*. Another fundamental theoretical problem addressed in [16] are abstract derivations. It was shown that for each derivation there is a unique abstract canonical derivation. See also [2] for a detailed discussion of this topic.

In the SPO approach, the results of this section have essentially been formulated in [8,5]. Originally, however, these problems have been investigated in the DPO-approach, see Section I.4 and I.5. The differences and similarities of the corresponding results are discussed to some depth in Section 6.

In [17] a new notion of a concurrent derivation captures the idea of a concurrent history. A complementary notion of a morphism describes a concurrent subhistory relation; it is based on causal rather than sequential dependencies between activities. This leads to a category of abstract concurrent derivations taken as the concurrency semantics of a SPO-graph grammar. In addition an interleaving semantics is proposed, given by a subcategory of abstract concurrent derivations, partially ordered by a sequential subcomputation relation. The concurrency semantics is characterized as a configuration domain of a prime event structure. The explicit consideration of infinite derivations leads to a notion of a fair derivation and a corresponding fair concurrency semantics.

3.2 Embedding of Derivations and Derived Productions

In this section we answer the question, under which conditions a derivation can be embedded into a larger context (cf. Problem I.3). Therefore, we first have to formalize the notion of "embedding". Below, an embedding is defined as a family of compatible injections between the graphs of the original derivation and the graphs of the embedded one.

Definition 17 (embedding). Given derivations $\rho = (G_0 \overset{p_1,m_0}{\Longrightarrow} \cdots \overset{p_k,m_{k-1}}{\Longrightarrow} G_k)$ and $\delta = (X_0 \overset{p_1,n_0}{\Longrightarrow} \cdots \overset{p_k,n_{k-1}}{\Longrightarrow} X_k)$, an **embedding** of ρ into δ is a family of total injective morphisms $e = (G_i \xrightarrow{e_i} X_i)_{i \in \{1,\ldots,k\}}$ such that $e_i \circ m_i = n_i$ for all $i \in \{1, \ldots, k\}$, see also the diagram below. The embedding of ρ into δ is denoted by $\rho \xrightarrow{e} \delta$, and the first injection $e_0 : G_0 \to X_0$ is called the **embedding morphism** of e. □

$$\begin{tikzcd}
& L_1 \arrow[rr, "p_1"] \arrow[dl, "m_0"'] \arrow[dd] & & R_1 \arrow[dl, "m_0^*"'] \arrow[dd] & \cdots & L_k \arrow[rr, "p_k"] \arrow[dl, "m_{k-1}"'] \arrow[dd] & & R_k \arrow[dl, "m_{k-1}^*"'] \arrow[dd] \\
G_0 \arrow[rr] \arrow[dr, dotted, "e_0"'] & & G_1 \arrow[dr, dotted, "e_1"'] & & G_{k-1} \arrow[rr] \arrow[dr, dotted, "e_{k-1}"'] & & G_k \arrow[dr, dotted, "e_k"'] & \\
& X_0 \arrow[rr] & & X_1 & \cdots & X_{k-1} \arrow[rr] & & X_k
\end{tikzcd}$$

Given a derivation ρ, a possible embedding e of ρ into a derivation δ is completely determined by the the embedding morphism e_0. The existence of the embedding $e : \rho \to \delta$ can therefore be characterized in terms of the derivation ρ and the embedding morphism e_0. In the sequel, a *derived production* will be constructed simulating the effect of the derivation ρ, such that the applicability of this derived production at the match e_0 is equivalent to the embedding e of ρ into δ induced by e_0. First we consider the basic case of a direct derivation

$$\begin{tikzcd}
L \arrow[r, "p"] \arrow[d, "m"] \arrow[dr, phantom, "(1)"] & R \arrow[d, "m^*"] \\
G \arrow[r, "p^*"] \arrow[d, "e"] \arrow[dr, phantom, "(2)"] & H \arrow[d, "e^*"] \\
X \arrow[r, "p^{**}"] & Z
\end{tikzcd}$$

$$\begin{tikzcd}
L_1 \arrow[r, "p_1"] \arrow[rr, bend left, "p"] \arrow[d, "m_1"] \arrow[dr, phantom, "(3)"] & R_1 = L_2 \arrow[r, "p_2"] \arrow[d, "m_2"] \arrow[dr, phantom, "(4)"] & R_2 \arrow[d, "m_2^*"] \\
G \arrow[r, "p_1^*"] \arrow[rr, bend right, "p^*"'] & H_1 \arrow[r, "P_2^*"] & H_2
\end{tikzcd}$$

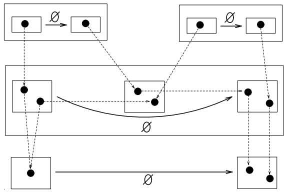


Figure 11: Horizontal and sequential composition of direct derivations, and non-sequentializable derived derivation.

leading to the notion of directly derived production:

Definition 18 (directly derived production). The **directly derived production** $\langle d\rangle : G \xrightarrow{p^*} H$ of a direct derivation $d = (G \overset{p,m}{\Longrightarrow} H)$ is composed of the production name $\langle d\rangle$ and the partial morphism p^*, i.e., the co-production of the direct derivation. A direct derivation $X \overset{\langle d\rangle,e}{\Longrightarrow} Z$ using $\langle d\rangle : p^*$ is called **directly derived derivation**. □

The following equivalence, relating directly derived derivations with embeddings of the original derivation, is also known as the vertical composition property of derivation diagrams.

Proposition 19 (directly derived production). *Given the directly derived production $\langle d\rangle : p^*$ of a derivation $d = (G \overset{p,m}{\Longrightarrow} H)$ as in Figure 11 on the left, the following statements are equivalent:*

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

2. *There is a direct derivation $d'' = (X \overset{p,n}{\Longrightarrow} Z)$ and an embedding $\langle e, e^*\rangle : d' \to d''$.*

Up to isomorphism, a bijective correspondence between 1. and 2. is given by $n = e \circ m$.

Proof. The directly derived derivation in 1. is represented by the pushout diagrams (1) and (2) on the left of Figure 11. Due to well-known pushout composition properties, diagram (1+2) is a pushout, too, which represents the direct derivation in 2. Vice versa, we can reconstruct the pushouts (1) and (2) from (1+2) by the pushout (1) of p and m and the pushout (2) of p^* and e. Uniqueness of pushouts up to isomorphism ensures the bijective correspondence between 1. and 2. □

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 its 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.

Definition 20 (sequential composition). Given two productions $p_1 : L_1 \to R_1$ and $p_2 : L_2 \to R_2$ with $R_1 = L_2$, the **sequentially composed production** $p_1; p_2 : L_1 \xrightarrow{p} R_2$ consists of the production name $p_1; p_2$ and the associated partial morphism $p = p_2 \circ p_1$. □

Derivations $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2}{\Longrightarrow} H_2$, 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 into a derivation $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_1^*}{\Longrightarrow} H_2$, provided that the co-match m_1^* of the first direct derivation is total. In the Proposition below, this is ensured by the assumption that m_1 is conflict-free w.r.t. p_1.

Proposition 21 (sequential composition). *Given two productions p_1 and p_2 and their sequential composition $p_1; p_2 : p$ as above, the following statements are equivalent:*

1. *There is a direct derivation $G \overset{p_1;p_2,m_1}{\Longrightarrow} H_2$ where m_1 is conflict-free w.r.t. p_1.*

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.*

Up to isomorphism, a bijective correspondence between 1. and 2. above is given by $p = p_2 \circ p_1$.

Proof. By properties of pushouts in $\boldsymbol{Graph^P}$ (see Lemma 8), the co-match m^* is total if and only if m is conflict-free w.r.t. p_1. Then Proposition 21 follows from pushout composition and decomposition properties, similar to Proposition 19. □

Combining directly derived productions by sequential composition we now define derived productions for derivations of length greater than one:

Definition 22 (derived production). Let $\rho = (G_0 \overset{p_1,m_0}{\Longrightarrow} \cdots \overset{p_k,m_{k-1}}{\Longrightarrow} G_k)$ be a derivation consisting of direct derivations $d_i = (G_{i-1} \overset{p_i,m_{i-1}}{\Longrightarrow} G_i)$ and $\langle d_i\rangle : p_i^*$ the corresponding directly derived productions. Then, the **derived production** $\langle\rho\rangle : G_0 \xrightarrow{p^*} G_k$ of ρ is given by the production name $\langle\rho\rangle =$

$\langle d_1\rangle; \ldots; \langle d_k\rangle$ and the associated partial morphism $p^* = p_k^* \circ \ldots \circ p_1^*$. A direct derivation $K \overset{\langle\rho\rangle, e}{\Longrightarrow} Y$ using $\langle\rho\rangle : p^*$ is called a **derived derivation**. □

Each derivation sequence may be shortcut by a corresponding derived derivation. Vice versa, each derived derivation based on a d-injective match corresponds to an embedding of the original derivation. The following theorem provides the solution to the Derived Production Problem I.4.

Theorem 23 (derived production). *Let $\rho = (G_0 \overset{p_1, m_0}{\Longrightarrow} \cdots \overset{p_k, m_{k-1}}{\Longrightarrow} G_k)$ be a derivation, $\langle\rho\rangle : p^*$ the corresponding derived production, and $G_0 \xrightarrow{e_0} X_0$ a d-injective match for $\langle\rho\rangle$. Then the following statements are equivalent:*

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

Up to isomorphism, a bijective correspondence between 1. and 2. above is given by $p^* = p_k^* \circ \ldots \circ p_1^*$ and $e_i \circ m_i = n_i$ for $i \in \{0, \ldots, k-1\}$.

Proof. By induction over the length k of ρ: Let $k = 1$. Then $\langle\rho\rangle$ is a directly derived production, and Theorem 24 follows from Proposition 19. Assume that 1. and 2. are equivalent for derivations ρ of length $k = l$. Let $\rho' = (G_0 \overset{p_0, m_0}{\Longrightarrow} \ldots \overset{p_l, m_{l-1}}{\Longrightarrow} G_l \overset{p_{l+1}, m_l}{\Longrightarrow} G_{l+1})$ be a derivation of length $k = l + 1$, ρ the prefix of ρ' consisting of the first l direct derivations, and $d_{l+1} = (G_l \overset{p_{l+1}, m_l}{\Longrightarrow} G_{l+1})$ the last direct derivation of ρ'. Then, there is a derivation δ' and an embedding $\rho' \xrightarrow{e'} \delta'$ iff there are embeddings $\rho \xrightarrow{e} \delta$ and $\langle e_l, e_{l+1}\rangle : d_{l+1} \to d'_{l+1}$ where e_l is also the last injection of e (cf. Definition 17). We show that the embeddings e and $\langle e_l, e_{l+1}\rangle$ exist iff there are corresponding derived derivations using the derived productions $\langle\rho\rangle : G_0 \xrightarrow{p^*} G_l$ of the derivation ρ and $\langle d_{l+1}\rangle : p$ of the direct derivation d_{l+1}. Then, Theorem 23 follows from Proposition 21 using the fact that, since e_0 is injective, it is in particular conflict-free w.r.t. $\langle\rho\rangle$.

By applying the assumption, there is a derived derivation $X_0 \overset{\langle\rho\rangle, e_0}{\Longrightarrow} X_l$ as in 1. iff there is an embedding $\rho \xrightarrow{e} \delta$ into a derivation $\delta = (X_0 \overset{p_1, n_0}{\Longrightarrow} \cdots \overset{p_l, n_{l-1}}{\Longrightarrow} X_l)$ as in 2. Using Proposition 19, the same equivalence holds between directly derived derivations $X_l \overset{\langle d_{l+1}\rangle, e_l}{\Longrightarrow} X_{l+1}$ and direct derivations $X_l \overset{p_{l+1}, n_l}{\Longrightarrow} X_{l+1}$ which are embeddings of d_{l+1}. This concludes the proof of Theorem 23. □

The following example is taken from [8]. It illustrates that a derived derivation based on a non-d-injective match may not be sequentializable.

Example 7 (non-sequentializable derived derivation). Consider the right diagram of Figure 11 on page 21, where the very same production is used twice for generating the derived production. The original production deletes one vertex and generates a new one.

The derived production specifies the deletion of two vertices and the generation of two others. But in fact by mapping the vertices to be deleted onto a single one leads to the generation of two vertices out of one. This is due to the fact that there is no direct connection between deleted and generated items, which in this situation means that, indeed, two instead of one vertex is newly added. □

Let us come back now to Problem I.3 which asked 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 $\langle\rho\rangle : p^*$ is applicable at e_0. Since in the SPO-approach there is no further condition for the application of a production but the existence of a match, this implies that there is an embedding $\rho \xrightarrow{e} \delta$ of the derivation ρ via each embedding morphism e_0.

Theorem 24 (embedding). *Let $\rho = (G_0 \overset{p_1, m_0}{\Longrightarrow} \cdots \overset{p_k, m_{k-1}}{\Longrightarrow} G_k)$ be a derivation and $G_0 \xrightarrow{e_0} X_0$ an embedding morphism. Then there is a derivation $\delta = (X_0 \overset{p_1, n_0}{\Longrightarrow} \cdots \overset{p_k, n_{k-1}}{\Longrightarrow} X_k)$ and an embedding $\rho \xrightarrow{e} \delta$.*

Proof. Let $\langle\rho\rangle : p^*$ be the derived production of ρ and $X_0 \overset{\langle\rho\rangle, e_0}{\Longrightarrow} X_k$ the derived derivation at the embedding morphism $G_0 \xrightarrow{e_0} X_0$. Since e_0 is injective, it is in particular d-injective. By Theorem 23 this implies the existence of the derivation δ and the embedding $\rho \xrightarrow{e} \delta$. □

Additional Remarks: A derived production represents the overall effect of the derivation it is constructed from, and fixes the interaction of the productions applied in the derivation. This representation, however, can still be reduced by cutting of the unnecessary context, i.e., all those elements of the starting and ending graphs of the derivation which are never touched by any of the productions. Such *minimal derived productions* (minimal w.r.t. the embedding relation) have been introduced in the SPO-approach in [5] for two-step derivations, generalized in [18] to derivations of length $k \geq 1$.

3.3 Amalgamation and Distribution

In this section we will investigate the simultaneous application of graph productions, which shall not work concurrently (as in a parallel derivation) but cooperatively, synchronizing their applications w.r.t. commonly accessed elements in the graph (cf. Section I.2). The synchronization of productions w.r.t.

certain deleted and/or generated elements is expressed by a common subproduction in the definition below.

Definition 25 (subproduction, synchronized productions). Let $p_i : L_i \to R_i$ be three productions for $i = 0, 1, 2$. We say that p_0 is a **subproduction** of p_1 if there is an embedding $in^1 = \langle in^1_L, in^1_R \rangle : p_0 \to p_1$, i.e. , a pair of total graph morphisms $in^1_L : L_0 \to L_1$ and $in^1_R : R_0 \to R_1$ such that $in^1_R \circ p_0 = p_1 \circ in^1_L$. The 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. □

$$\begin{tikzcd}
L_0 \arrow[r, "p_0"] \arrow[d, "in^1_L"'] & R_0 \arrow[d, "in^1_R"] \\
L_1 \arrow[r, "p_1"] & R_1
\end{tikzcd}$$

If synchronized productions $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ shall be applied simultaneously to one global graph, this may be modeled by the application of an amalgamated production which is constructed as the gluing of p_1 and p_2 along p_0. Formally such a gluing is described by a pushout construction. Since the amalgamated production is a composed production, its production name has to record this construction. So similar as for the parallel production, the name $p_1 \oplus_{p_0} p_2$ of an amalgamated production is a whole diagram comprising the given synchronized productions $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ and their embeddings into the production morphism of the amalgamated production.

Definition 26 (amalgamated productions and derivations). Let $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ be synchronized productions with $p_i : L_i \to R_i$ for $i \in \{0, 1, 2\}$. Then the **amalgamated production** $p_1 \oplus_{p_0} p_2 : L \xrightarrow{p} R$, consists of the production name $p_1 \oplus_{p_0} p_2$ and the associated partial morphism p. The production name $p_1 \oplus_{p_0} p_2$ is constructed on the left of Figure 12, where $\langle {in^2_L}^*, {in^1_L}^* \rangle$ and $\langle {in^2_R}^*, {in^1_R}^* \rangle$ are pushouts of $\langle in^1_L, in^2_L \rangle$ and $\langle in^1_R, in^2_R \rangle$, respectively, and $L \xrightarrow{p} R$ is obtained as the universal morphism satisfying $p \circ {in^1_L}^* = {in^1_R}^* \circ p_1$ and $p \circ {in^2_L}^* = {in^2_R}^* \circ p_2$. A direct derivation $G \overset{p_1 \oplus_{p_0} p_2, m}{\Longrightarrow} X$ using the amalgamated production $p_1 \oplus_{p_0} p_2$ is called **amalgamated derivation**. By referring to morphisms $m_1 = m \circ {in^2_L}^*$ and $m_2 = m \circ {in^1_L}^*$ we also write $G \overset{p_1 \oplus_{p_0} p_2, m_1 \oplus m_2}{\Longrightarrow} X$. □

A distributed graph models a state which is splitted into substates related by common interface states. Gluing the local states along their interfaces yields the global state again. Below these ideas are formalized for two local states related by one interface.

$$\begin{tikzcd}
 & L_0 \arrow[rrr, "p_0"] \arrow[dr, "in^2_L"'] \arrow[ddl, "in^1_L"'] & & & R_0 \arrow[dr, "in^2_R"] \arrow[ddl, "in^1_R"'] & \\
 & & L_2 \arrow[rrr, "p_2"] \arrow[ddl, "{in^1_L}^*"] & & & R_2 \arrow[ddl, "{in^1_R}^*"] \\
L_1 \arrow[rrr, "p_1"] \arrow[dr, "{in^2_L}^*"'] & & & R_1 \arrow[dr] & & \\
 & L \arrow[rrr, "p", "{in^2_R}^*"'] & & & R &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & L_0 \arrow[rrr, "p_0"] \arrow[dr, "in^2_L"'] \arrow[ddl, "in^1_L"'] & & & R_0 \arrow[dr, "in^2_R"] \arrow[ddl, "in^1_R"'] & \\
 & & L_2 \arrow[rrr, "p_2"] \arrow[ddl, "{in^1_L}^*"] \arrow[dddl, "m_2"] & & & R_2 \arrow[ddl, "{in^1_R}^*"] \\
L_1 \arrow[rrr, "p_1"] \arrow[dr, "{in^2_L}^*"'] \arrow[ddr, "m_1"'] & & & R_1 \arrow[dr, "{in^2_R}^*"] & & \\
 & L \arrow[rrr, "p"] \arrow[d, "m_1 \oplus m_2"] & & & R \arrow[d, dashed, "m_1 \oplus {m_2}^*"] & \\
 & G \arrow[rrr, dashed, "p^*"] & & & X &
\end{tikzcd}$$

Figure 12: Amalgamated Production Amalgamated Derivation

Definition 27 (distributed graph). 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 two local graphs. The **global graph** $G = \oplus\ DG = G_1 \oplus_{G_0} G_2$ of DG is defined as the pushout object of g_1 and g_2 in the diagram below. The distributed graph DG is a **total splitting** of G if the graph morphisms g_1 and g_2 are total. In general DG is called a **partial splitting**.

$$\begin{tikzcd}
 & G_0 \arrow[dl, "g_1"'] \arrow[dr, "g_2"] & \\
G_1 \arrow[dr, "g_2^*"'] & (PO) & G_2 \arrow[dl, "g_1^*"] \\
 & G_1 \oplus_{G_0} G_2 &
\end{tikzcd}$$

□

A truly partial splitting models an inconsistent distributed state where, for example, there are dangling references between the local substates. In such situations, there is not general agreement if a certain item belongs to the corresponding global state or not. Constructing the global graph as the pushout object of the splitting, this question is decided in favor of deletion, that is, conflicting items are removed from the global state.

Distributed graphs can be transformed by synchronized productions, which specify a simultaneous update of all local graphs.

Definition 28 (distributed derivation). Let $DG = (G_1 \xleftarrow{g_1} G_0 \xrightarrow{g_2} G_2)$ be a distributed graph and $L_i \xrightarrow{m_i} G_i$ for $i \in \{0, 1, 2\}$ be matches for synchronized productions $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ as in Figure 13 on the left. The matches $(m_i)_{i \in \{0,1,2\}}$ form a **distributed match** for $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ into DG if $g_k \circ m_0 = m_k \circ in^k_L$ for $k \in \{1, 2\}$. In this case, the **distributed derivation** $d_1||_{d_0}d_2 : DG \Longrightarrow DH$ with $DH = (H_1 \xleftarrow{h_1} H_0 \xrightarrow{h_2} H_2)$ is constructed by the **local direct derivations** $d_1 = (G_1 \overset{p_1, m_1}{\Longrightarrow} H_1)$ and $d_2 = (G_2 \overset{p_2, m_2}{\Longrightarrow} H_2)$

and the **interface derivation** $d_0 = (G_0 \overset{p_0,m_0}{\Longrightarrow} H_0)$. The partial morphisms h_1 and h_2 are induced by the universal property of the pushout $\langle p_0^*, m_0^* \rangle$ of $\langle p_0, m_0 \rangle$ such that the left and bottom squares of the right diagram of Figure 13 commute. If g_1 and g_2 as well as h_1 and h_2 are total, we speak of a **synchronous distributed derivation** $d_1||_{d_0}d_2$. □

A distributed graph becomes inconsistent (i.e., a partial splitting) if we delete an item in a local graph which has a preimage in the interface that is not deleted. This situation can be characterized in set-theoretical terms:

Characterization 29 (synchronous distributed derivation). *Let $d_i = (G_i \overset{p_i,m_i}{\Longrightarrow} H_1)$ for $i \in \{0,1,2\}$ be direct derivations at conflict-free matches. A distributed derivation $d_1||_{d_0}d_2 : DG \Longrightarrow DH$ as in Figure 13 is synchronous if and only if DG is a total splitting, and for $k \in \{1,2\}$ we have that $y \in G_0$ with $g_k(y) \in m_k(L_k - dom(p_k))$ implies $y \in m_0(L_0 - dom(p_0))$.*

Proof sketch. The structures $m_k(L_k - dom(p_k))$ for $k = 1,2$ and $m_0(L_0 - dom(p_0))$ consist of those elements of G_k and G_0 which are explicitly deleted by the direct derivations d_k and d_0, respectively. The distributed derivation $d_1||_{d_0}d_2$ is synchronous if the morphisms h_k in the right diagram of Figure 13 are total. We sketch the "if" part of the proof of Characterization 29. For a complete proof the reader is referred to [19].

Let $x \in H_0$. Since p_0^* and m_0^* are pushout morphisms, they are jointly surjective by Lemma 8, i.e., x has either a preimage $y \in G_0$ or $z \in R_0$. In the first case there is $g_k(y) \in G_k$ since g_k is total. Since y is preserved by p_0^*, it is not in $m_0(L_0 - dom(p_0))$. Hence $g_k(y) \notin m_k(L_k - dom(p_k))$, which implies by some further arguments that $g_k(y) \in dom(p_k^*)$, i.e., $p_k^* \circ g_k(y)$ is defined. By commutativity of the bottom square this implies that also $h_k(p_0^*(y)) = h_k(x)$ is defined. If x has a preimage $z \in R_0$, $m_k^*(in_R^k(z))$ is defined since m_k^* is total by conflict-freeness of m_k (see Lemma 8). By commutativity of the left square this implies that also $h_k(m_0^*(z)) = h_k(x)$ is defined. □

Finally, we investigate the relationship between distributed and amalgamated derivations. Amalgamating a distributed derivation means to construct a global observation of the local actions. This is possible only if there is a consistent global view at least of the given distributed state DG. Hence DG is assumed to be a total splitting in the distribution theorem below. On the other hand, distributing an amalgamated derivation means to split a global action into local ones. Therefore, the matches of the elementary productions have to be compatible with the splitting of the global graph. The Distribution Theorem below provides the solution to Problem I.5.

Theorem 30 (distribution). *Let $DG = (G_1 \xleftarrow{g_1} G_0 \xrightarrow{g_2} G_2)$ be a total splitting of $G = \oplus DG$, $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ synchronized productions and $p_1 \oplus_{p_0} p_2 :$*

$$\begin{tikzcd}
 & L_0 \arrow[rrr, "p_0"] \arrow[ddl] \arrow[dr] \arrow[ddd, "m_0"] & & & R_0 \arrow[ddl] \arrow[dr] \arrow[ddd, dashed, "m_0^*"] & \\
 & & L_2 \arrow[rrr, "p_2"] \arrow[ddd, "m_2"] & & & R_2 \arrow[ddd, dashed, "m_2^*"] \\
L_1 \arrow[rrr, "p_1"] \arrow[ddd, "m_1"] & & & R_1 \arrow[ddd, dashed, "m_1^*"] & & \\
 & G_0 \arrow[rrr, dashed, "p_0^*"] \arrow[ddl] \arrow[dr] & & & H_0 \arrow[ddl, dashed] \arrow[dr, dashed] & \\
 & & G_2 \arrow[rrr, dashed, "p_2^*"] & & & H_2 \\
G_1 \arrow[rrr, dashed, "p_1^*"] & & & H_1 & &
\end{tikzcd}
\qquad
\begin{tikzcd}
L_0 \arrow[rr, "p_0"] \arrow[dr, "in_L^k"] \arrow[dd, "m_0"] & & R_0 \arrow[dr, "in_R^k"] \arrow[dd, "m_0^*"] & \\
 & L_k \arrow[rr, "p_k"] \arrow[dd, "m_k"] & & R_k \arrow[dd, "m_k^*"] \\
G_0 \arrow[rr, "p_0^*"] \arrow[dr, "g_k"] & & H_0 \arrow[dr, "h_k"] & \\
 & G_k \arrow[rr, "p_k^*"] & & H_k
\end{tikzcd}$$

Figure 13: A distributed derivation $d_1||_{d_0}d_2 : DG \to DH$ with $d_0 = (G_0 \overset{p_0,m_0}{\Longrightarrow} H_0)$ and $d_k = (G_k \overset{p_k,m_k}{\Longrightarrow} H_k)$ for $k \in \{1,2\}$.

$L \xrightarrow{p} R$ their amalgamated production. Then the following statements are equivalent:

1. *There are an amalgamated derivation $G \overset{p_1 \oplus_{p_0} p_2, m}{\Longrightarrow} H$, and a distributed match $(m_i)_{i \in \{0,1,2\}}$ for $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ into DG, which is compatible with m, i.e., $g_2^* \circ m_1 = m \circ in_L^{2\,*}$ and $g_1^* \circ m_2 = m \circ in_L^{1\,*}$.*

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

Proof sketch. The proof of this theorem in [20] uses the 4-cube lemma presented in [21], which is valid in every category and can be derived as a special case of the commutativity of colimits. □

Additional Remarks: Amalgamated derivations are introduced in the SPO-approach in [5], while corresponding concepts in the DPO-approach have been developed in [22]. The Amalgamation Theorem in [5] is concerned with the sequentialization of an amalgamated derivation. It is based on the notion of a remainder (production) which can be roughly be considered as that part of a given production which is not covered by its subproduction. A suitably given amalgamated derivation can then be simulated by a derivation consisting of an application of the common subproduction, followed by a parallel application of the remainders [5]. The concept of amalgamated productions and derivations was generalized to cases including more than two elementary productions. Corresponding developments in [19] were motivated by the idea to specify derivations in which a variable number of mutually interacting productions must be synchronized.

Distributed graphs and derivations in the SPO-approach are introduced in [20], where also the Distribution Theorem is formulated. The comparison of several kinds of global and (synchronous and asynchronous) distributed derivations led to a hierarchy theorem for distributed derivations. Splitting a state into two substates with one interface is, of course, a very basic kind of a distribution. In [19] this is generalized to arbitrary many local states, pairwise related by interfaces. Even more general topologies are considered in [23,24], in the DPO-approach.

4 Application Conditions in the SPO-Approach

Using the rule-based formalism introduced so far we may easily and intuitively describe, *how* given graphs shall be transformed into derived graphs. For specifying *when* these transformations should occur, however, we are restricted to positive application conditions concerning the existence of certain nodes and edges, which can be specified within the left-hand side of the productions. In this section we introduce the possibility to specify also negative application conditions for each particular production, and extend the results of Section 3.1 concerning independence and parallelism to productions and derivations with application conditions.

4.1 Negative Application Conditions

We use the model of the Pacman game to motivate the use of negative application conditions.

Example 8. Recall the production *moveP* from Figure 1. As effect of this production Pacman moves to a new field not taking into account whether this field is dangerous for him or not. If Pacman moves to a field where a ghost is waiting, he may be killed in the next derivation step. An intelligent player of the Pacman game would like to apply the *moveP* production only if there *is no ghost* on the field Pacman is moving to. Hence we have a *negative application condition* for our production. The production *ImoveP* which models an intelligent moving of Pacman is depicted as the left upper production of Figure 14. The forbidden context, i.e., the ghost with the edge pointing to the field Pacman wants to move to, is enclosed by a dotted line and crossed out, in order to denote that these elements must not exist if the production shall be applied. Analogously one could think of an intelligent behavior of the ghosts. If they want to kill Pacman they have to occupy as many fields as possible. Hence a ghost should only be moved to a field on which there is not already another ghost (see production *ImoveG* in the right upper part of Figure 14).

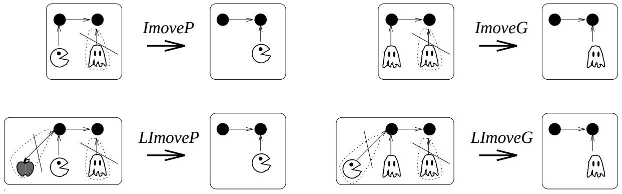


Figure 14: Intelligent moving of Pacman and ghosts.

By now we have shown that negative application conditions can be used to include rather simple strategies of the game in our model. Graph transformations are by default non-deterministic. But for our model it is desirable to restrict this non-determinism, i.e. to have priorities on the productions. Moving has a lower priority than eating an apple for Pacman or than killing Pacman for a ghost. These priorities can also be coded into negative application conditions for the move productions, such that they are only applicable if the *eat* resp. *kill* production is not applicable to Pacman resp. the same ghost (see lower part of Figure 14). Note that intelligent moving with lower priority has a negative application condition consisting of two forbidden contexts, called constraints in the following, one taking care of the field to move to and one making sure that moving is of lower priority. These application conditions with more than one constraint are satisfied if each of the constraints is satisfied. □

The general idea is to have a left-hand side not only consisting of one graph but of several ones, connected by morphisms $L \xrightarrow{l} \hat{L}$, called constraints, with the original left-hand L. For each constraint, $\hat{L} - l(L)$ represents the forbidden structure, i.e., the dotted bordered part of the example productions. A match satisfies a constraint if it cannot be extended to the forbidden graph $\hat{L}$, i.e., if the additional elements are not present in the context of the match. If a constraint is non-injective and surjective, i.e., the forbidden structure $\hat{L} - l(L)$ is empty but items of L are identified by l, it can only be satisfied by a match not identifying these items. Thus, negative constraints can also express injectivity requirements for the match. A match satisfies a negative application condition if it satisfies all constraints the application condition consists of.

Definition 31 (application conditions).

1. A **negative application condition**, or application condition for short, over a graph L is a finite set A of total morphisms $L \xrightarrow{l} \hat{L}$, called **constraints**.

2. A total graph morphism $L \xrightarrow{m} G$ **satisfies** a constraint $L \xrightarrow{l} \hat{L}$, written $m \models l$, if there is no total morphism $\hat{L} \xrightarrow{n} G$ such that $n \circ l = m$. m

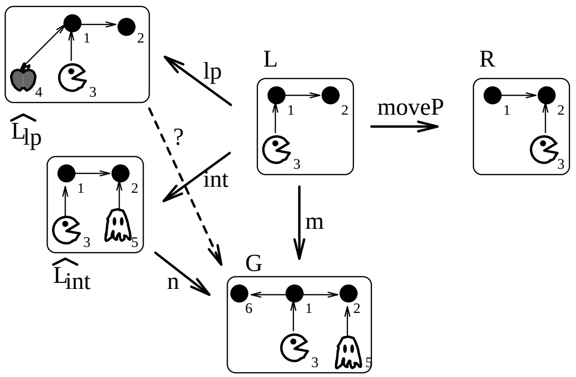


Figure 15: Formal representation of the conditional production $LImoveP$

satisfies an application condition A over L, written $m \models A$, if it satisfies all constraints $l \in A$.

3. An application condition A is said to be **consistent** if there is a graph G and a total morphism $L \xrightarrow{m} G$ s.t. m satisfies A.

4. A **production with application condition** $\hat{p} : (L \xrightarrow{p} R, A(p))$, or **conditional production** for short, is composed of a production name $\hat{p}$ and a pair consisting of a partial graph morphism p and an application condition $A(p)$ over L. It is **applicable** to a graph G at $L \xrightarrow{m} G$ if m satisfies $A(p)$. In this case, the direct derivation $G \overset{p,m}{\Longrightarrow} H$ is called **direct conditional derivation** $G \overset{\hat{p},m}{\Longrightarrow} H$.

□

Example 9. In Figure 15 we show the formal representation of the production $LImoveP : (moveP, \{lp, int\})$ of Figure 14 consisting of the unconditional production $moveP$ of Figure 1 and two constraints. Morphisms are denoted by numbers at corresponding nodes. The constraint int states that there must not be a ghost at the field Pacman wants to move to, and lp ensures that there is no apple at Pacmans current position. Accordingly the match m for $moveP$ satisfies lp since we cannot extend m to $\hat{L}_{lp}$, while m does not satisfy int. Hence m does not satisfy $\{lp, int\}$. If, however, Pacman moves to the left, i.e. vertex $\bullet_2$ of L is mapped to vertex $\bullet_6$ in G, int is satisfied as well and the production can be applied. □

It is possible to define application conditions which can not be satisfied by any match, i.e., the corresponding conditional production is never applicable. This is due to the fact that contradictions may appear between the positive requirements of the productions left-hand side and the negative constraints. A trivial example for such a contradiction is a constraint which is an isomorphism. More generally, a contradiction occurs if the forbidden items of $\hat{L} - l(L)$ can be mapped onto the required ones in L, i.e., a total morphism $\hat{L} \xrightarrow{n} L$ exists, extending the identity on L. Hence, a negative application condition A is consistent in the sense of Definition 31.3 (i.e., free of these contradictions) iff the match $L \xrightarrow{id} L$ satisfies A.

Lemma 32 (consistency of application conditions). *An application condition A over L is consistent if and only if it is satisfied by id_L.*

Proof. If A is satisfied by id_L, A is consistent by Definition 31. Vice versa, let id_L not satisfy A. Then there are a constraint $L \xrightarrow{l} \hat{L} \in A$ and a total morphism $\hat{L} \xrightarrow{n} L$ s.t. $n \circ l = id_L$. Since for any given match $L \xrightarrow{m} G$ we have that $m = m \circ id_L$, this implies that $(m \circ n) \circ l = m$, i.e., m does not satisfy the constraint $l \in A$, and hence not A. □

Even more powerful application condition, which can express, for example, cardinality restrictions on the number of incoming edges of a given node, are obtained if the the forbidden morphisms $\hat{L} \xrightarrow{n} G$ of Definition 31.2 are required to be injective. These *application conditions with injective satisfaction*, as they are called in [3], may, for example, express the gluing condition of the DPO approach (see Proposition I.9) consisting of the dangling condition and the identification condition: For a given production $L \xrightarrow{p} R$ we let

- the **identification condition** of p be the set $IC(p)$ of all total surjective morphisms l, except isomorphisms, such that for all l we have $l(x) = l(y)$ for some $x, y \in L$ with $x \neq y$ and $x \notin dom(p)$, and
- the **dangling condition** of p be the set $DC(p)$ of all total morphisms $L \xrightarrow{l} \hat{L}$ such that l is surjective up to an edge e (and possibly a node) with $s(e)$ or $t(e)$ in $l(L - dom(p))$.

Now a match m satisfies the gluing condition if and only if it injectively satisfies the application conditions $IC(p)$ and $DC(p)$, i.e., iff there is no total *injective* morphism $\hat{L} \xrightarrow{n} G$ for any constraint $L \xrightarrow{l} \hat{L} \in IC(p) \cup DC(p)$ satisfying $n \circ l = m$. Using this equivalence, DPO derivations can be characterized by conditional SPO derivations.

4.2 Independence and Parallelism of Conditional Derivations

Following the line of [3] we extend the results of Section 3.1 on independence and parallelism to conditional derivations. Thereby, we provide a solutions to

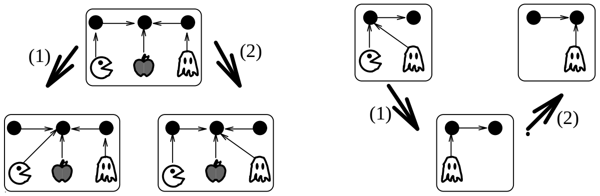


Figure 16: Parallel and sequential independence (example).

the Local Church-Rosser Problem I.1 and the Parallelism Problem I.2 for the SPO-approach with application conditions.

Interleaving

According to Section 3, two derivations d_1 and d_2 are considered to be parallel independent if they may occur in any order with the same result. For unconditional derivations this is mainly a problem of deletion, i.e., none of the two derivations should delete something that is needed for the match of its alternative. Taking into account negative application conditions specifying forbidden application contexts, we have to ensure that, in addition, no such forbidden context is established.

Definition 33 (parallel independence). Let $d_1 = (G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1)$ and $d_2 = (G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2)$ be two direct conditional derivations using $\hat{p}_1 : (p_1, A(p_1))$ and $\hat{p}_2 : (p_2, A(p_2))$, respectively. Then we say that d_2 is **weakly parallel independent** of d_1 if $m_2' = {r_1}^* \circ m_2 : L_2 \to H_1$ is a match for p_2 that satisfies the application condition $A(p_2)$ of $\hat{p}_2$ (see left diagram of Figure 17). Direct conditional derivations d_1 and d_2 are **parallel independent** if they are mutually weakly parallel independent. □

Example 10. Let (1) and (2) in the left side of Figure 16 be applications of the productions *moveP* and *moveG* of Figure 1, respectively. Then these unconditional direct derivations are parallel independent. If we apply the conditional production *ImoveP* of Figure 14 instead of *moveP*, preventing Pacman from approaching the ghost, (1) is not weakly parallel independent of (2). □

A derivation d_2' is sequentially independent of its predecessor d_1 if d_2' may occur also alternatively to, or even before d_1. In Section 3 this is shown to be the case if the application context of d_2' has already been present before the occurrence of d_1. In the conditional case we additionally require that no forbidden context of d_2' has been destroyed by d_1.

Definition 34 (sequential independence). Let $d_1 = (G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1)$ and $d_2' = (H_1 \overset{\hat{p}_2,m_2'}{\Longrightarrow} X)$ be two direct conditional derivations using $\hat{p}_1 : (p_1, A(p_1))$ and $\hat{p}_2 : (p_2, A(p_2))$, respectively. Then we say that d_2' is **weakly sequentially independent** of d_1 if $m_2 = ({r_1}^*)^{-1} \circ m_2' : L_2 \to G$ is a match for p_2 that satisfies the application condition $A(p_2)$ of $\hat{p}_2$ (see right diagram in Figure 17). Let $d_2 = (G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2)$ be the corresponding direct derivation. In case that, additionally, d_1 is weakly parallel independent of d_2 the derivation sequence $(G1 \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,m_2'}{\Longrightarrow} X)$ is called **sequentially independent**. □

Example 11. In the right side of Figure 16 a sequential independent derivation sequence using *kill* and *moveG* of Figure 1 is shown. If, however, (2) results from an application of *LImoveG* of Figure 14, (2) is not independent of (1) because the ghost has to kill Pacman before leaving his field. □

The following theorem is a straightforward generalization of the Local Church-Rosser Theorem of Section 3 to conditional derivations. It provides the solution to the Local Church-Rosser Problem I.1 for SPO-derivations with application conditions.

Theorem 35 (conditional local Church-Rosser). *Let $d_1 = (G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1)$ and $d_2 = (G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2)$ be two direct conditional derivations. Then the following statements are equivalent:*

1. *d_1 and d_2 are parallel independent.*
2. *There are direct conditional derivations $H_1 \overset{\hat{p}_2,m_2'}{\Longrightarrow} X$ and $H_2 \overset{\hat{p}_1,m_1'}{\Longrightarrow} X$ such that $G \overset{\hat{p}_1,m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2,m_2'}{\Longrightarrow} X$ and $G \overset{\hat{p}_2,m_2}{\Longrightarrow} H_2 \overset{\hat{p}_1,m_1'}{\Longrightarrow} X$ are sequentially independent conditional derivations.*

Up to isomorphism, a bijective correspondence between 1. and 2. above is given by $m_2' = {r_1}^* \circ m_2$ and $m_1' = {r_2}^* \circ m_1$.

Proof. Directly from Definitions 33 and 34 and the Local Church-Rosser Theorem 14. □

Explicit Parallelism

Now we consider the construction of a parallel conditional production from two conditional elementary productions. Starting with the parallel production $p_1 + p_2$ defined in Definition 15 we have to find a suitable application condition for $p_1 + p_2$. For unconditional single-pushout derivations, the applicability of

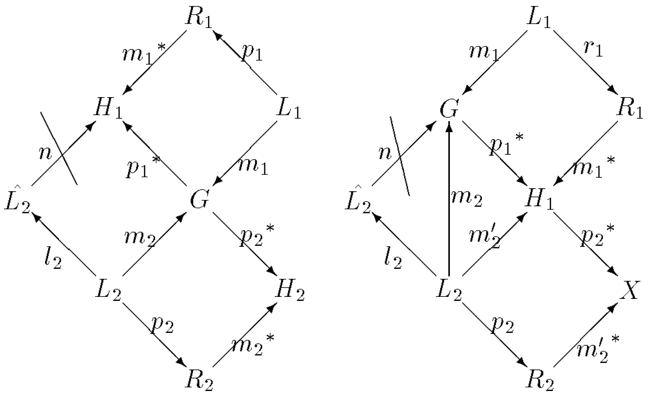


Figure 17: Independence of conditional derivations.

the parallel production is (trivially) equivalent to the applicability of the elementary productions at their corresponding matches. Generalizing this we have to construct $A(p_1 + p_2)$ as conjunction of $A(p_1)$ and $A(p_2)$.

If two application conditions are defined over the same left-hand side, their conjunction is simply given by their union. In our case however, $A(p_1)$ and $A(p_2)$ are application conditions over L_1 and L_2, respectively. Thus the problem remains, how to extend the application conditions over L_1 and L_2 to the larger graph $L_1 + L_2$. Below the extension of a constraint l along a total morphism m is defined by the pushout of l and m.

Definition 36 (extension). If $L \xrightarrow{m} G$ is a total morphism and $L \xrightarrow{l} \hat{L}$ a constraint, the *extension* $m^{\#}(l)$ *of* l *along* m is given by the pushout diagram (1) in the left-hand side of Figure 18. The **extension of an application condition** A over L is defined by $m^{\#}(A) = \{m^{\#}(l) | l \in A\}$. □

Proposition 37 (extension). *Let* $L \xrightarrow{m} G$ *be a total morphism and* A *an application condition over* L. *Then, for all matches* $G \xrightarrow{e} K$, $e \models m^{\#}(A)$ *iff* $e \circ m \models A$.

Proof. We show that for each $l \in A$ we have $e \models m^{\#}(l)$ iff $e \circ m \models l$. Assume n s.t. (2) in Diagram 18 commutes. Then $n' = n \circ \hat{m}$ and $n' \circ l = m \circ e$ by commutativity of (1) and (2). Vice versa, let $\hat{L} \xrightarrow{n'} K$ be given with $n' \circ l = e \circ m$. Then n with $n \circ g = e$ exists by the universal property of (1). □

Now we define the parallel conditional production $\hat{p}_1 + \hat{p}_2$ as the parallel production of the underlying productions p_1 and p_2 of $\hat{p}_1$ and $\hat{p}_2$, equipped with the extensions of the application conditions $A(p_1)$ and $A(p_2)$ of the component productions along in_L^1 and in_L^2, respectively.

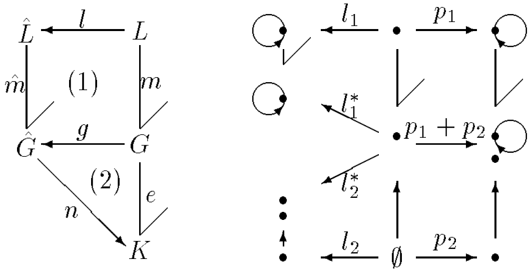


Figure 18: Extension of constraints and construction of a parallel conditional production

Definition 38 (parallel conditional production). Let $\hat{p}_1 : (L_1 \xrightarrow{r_1} R_1, A(p_1))$ and $\hat{p}_2 : (L_2 \xrightarrow{r_2} R_2, A(p_2))$ be conditional productions and $p_1 + p_2 : L \xrightarrow{p} R$ the parallel production of p_1 and p_2 according to Definition 15. Then the **parallel conditional production** $\hat{p}_1 + \hat{p}_2 : (p, A(p))$ is composed of the production name $\hat{p}_1 + \hat{p}_2$ and the pair $(p, A(p))$, where $A(p) = in_L^{1\,\#}(A(p_1)) \cup in_L^{2\,\#}(A(p_2))$. A conditional derivation $G \overset{\hat{p}_1+\hat{p}_2,m}{\Longrightarrow} X$ using $\hat{p}_1 + \hat{p}_2$ is called **parallel conditional derivation**. □

Example 12. The construction above does not guarantee, that the application condition $A(p_1 + p_2)$ of $\hat{p}_1 + \hat{p}_2$ is consistent, even if both $A(p_1)$ and $A(p_2)$ are. In the right side of Figure 18 the production $\hat{p}_1 : (p_1, \{l_1\})$ adds a loop to a node if there isn't already one. The production $\hat{p}_2 : (p_2, \{l_2\})$ inserts a node in an empty graph. The application condition of the parallel production $\hat{p}_1 + \hat{p}_2 : (p_1 + p_2, \{l_1^*, l_2^*\})$ is not consistent (compare Lemma 32) because we can reverse l_2^* by identifying the two nodes. On the other hand, there is no graph to which we can apply both $\hat{p}_1$ and $\hat{p}_2$, i.e., their application domains are disjoint. That this is no mere coincidence is shown by the following proposition. □

Proposition 39 (applicability of parallel production). *Let* $\hat{p}_1, \hat{p}_2$ *and* $\hat{p}_1 + \hat{p}_2$ *be given as above together with matches* $L_1 \xrightarrow{m_1} G$ *and* $L_2 \xrightarrow{m_2} G$ *for* p_1 *and* p_2 *into* G *and let* $L_1 + L_2 \xrightarrow{m_1+m_2} G$ *be the parallel match for* $p_1 + p_2$. *Then* $m_1 + m_2 \models A(p_1 + p_2)$ *if and only if* $m_1 \models A(p_1)$ *and* $m_2 \models A(p_2)$.

Proof. $m_k = m_1 + m_2 \circ in_L^k$ for $k = 1, 2$ by universal property of $L_1 + L_2$. Then Proposition 39 is a direct consequence of Proposition 37. □

Since we can check consistency of application conditions by Lemma 32 this provides us with a method to decide whether a parallel production makes sense or not. Furthermore, we may now extend the Weak Parallelism Theorem 16 to conditional derivations, which solves the Parallelism Problem I.2 for conditional derivations.

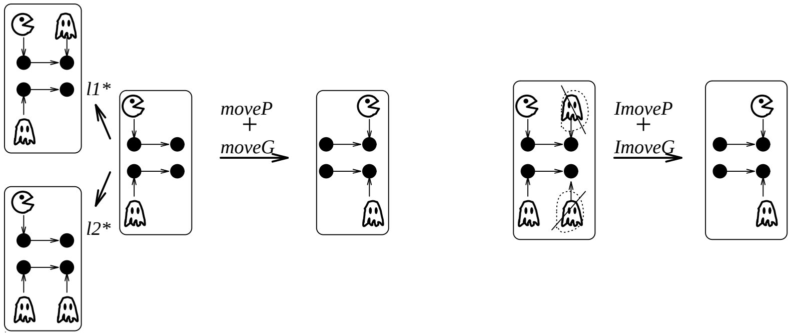


Figure 19: Parallel production of *ImoveP* and *ImoveG*.

Theorem 40 (conditional weak parallelism). *Given conditional productions $\hat{p}_1$ and $\hat{p}_2$, the following two statements are equivalent:*

1. *There is a direct conditional parallel derivation $G \overset{\hat{p}_1+\hat{p}_2, m_1+m_2}{\Longrightarrow} X$, s.t. $G \overset{\hat{p}_2, m_2}{\Longrightarrow} H_2$ is weakly parallel independent of $G \overset{\hat{p}_1, m_1}{\Longrightarrow} H_1$.*

2. *There is a conditional derivation $G \overset{\hat{p}_1, m_1}{\Longrightarrow} H_1 \overset{\hat{p}_2, m_2'}{\Longrightarrow} X$, where $H_1 \overset{\hat{p}_2, m_2'}{\Longrightarrow} X$ is weakly sequentially independent of $G \overset{\hat{p}_1, m_1}{\Longrightarrow} H_1$.*

Up to isomorphism, a unique correspondence between 1. and 2. above is given by $m_2' = {r_1}^* \circ m_2$.

Proof. Directly from Proposition 39, Definition 33 and 34 and the Parallelism Theorem 16. □

Example 13. In the left side of Figure 19 the parallel production of *ImoveP* and *ImoveG* is shown, modeling a simultaneous move of Pacman and one of the ghosts. Its graphical representation is given on the right. Applying this production to the graph in the upper left of Figure 16 we have an example of a parallel conditional derivation that cannot be sequentialized, because the alternative derivations using *ImoveG* and *ImoveP* (denoted by (1) and (2) in the left side of Figure 16) are not parallel independent (cf. Example 10). □

Additional Remarks: The results of this section have been obtained in [3]. In the case of application conditions with injective satisfaction (cf. Section 4), similar results are possible. Moreover, most of the remaining results of Section 3 have already been extended to such application conditions in [25,26]. In addition to the left-sided negative application conditions of Definition 31, also right-sided application conditions and so-called propositional application conditions, i.e., propositional expressions over constraints, are considered in [26]. This is even further generalized in [27] and [28] by conditional application conditions.

Another interesting line of research is the generative power of graph grammars with (different kinds of) conditional productions. In [3] it has been shown that context-free graph grammars with positive and/or negative application conditions are strictly more powerful than unconditional ones. Similar investigations can be found in [27] for graph grammars without nonterminal labels.

5 Transformation of More General Structures in the SPO-Approach

Until now we presented SPO transformations and corresponding results based on labeled graphs. In this section we will show that the SPO approach is not restricted to this kind of graphs, but it is also applicable to more sophisticated structures. We will basically consider two kinds of extensions: a more powerful labeling concept – leading to the concept of attributed graphs in Section 5.1; and more complex graphical structures – leading to the concept of graph structures in Section 5.2. By a generalization of graph structures we will then be able to cope with these two different extensions within the same framework. This framework, introduced in [6,17], opens new possibilities for defining graphs. (For example we can use instead of sets and labeling *functions*, graphs and labeling graph *morphisms* – this idea was worked out in [29] for the definition of class-based graph grammars.) Finally in Section 5.3 we review *high-level replacement systems*: a general axiomatic framework for transformation systems and their properties.

5.1 Attributed Graphs

The existence of labels allows us to distinguish vertices or edges within a graph according to their associated label. Thus, labels provide a very basic type concept for graphs. Usually in a software system the types of elements do not change during the execution of the system, and therefore the treatment of labels presented in the last sections (where label morphisms were identities) is very reasonable. Nevertheless, often a situation occurs in which the labels (types) of the elements in a graph are not relevant for the application of a production. In this case labels should take the role of parameters (generic types). But actually, the strict typing through labels presented so far requires a number of productions (one for each possible combination of labels) to describe this situation. For practical reasons this is not adequate, a higher-level kind of typing concept is needed. From now on we will refer to this kind of high-level types as attributes. The basic idea of attributes in graph transformations is to allow the handling of labels abstractly. This is realized by the presence of corresponding variables and a concept of assignment of these variables in an actual situation. In particular, the concept of attributes includes the presence of operations on these sets (of attributes).

Attributes are used in all graph grammar proposals for software engineering since they integrate structural (i.e. graphical) aspects of a system with data-type aspects (i.e. calculation of values). This leads to compact descriptions in which e.g. well-known arithmetic operations need not artificially be coded into graphical structures. Similar concepts of combining structural and algebraic aspects can be found in the theory of attributed string grammars [30] in algebraic high-level Petri-nets [31,32,33], which are a combination of Petri-nets and algebraic specifications, and in the specification language LOTOS [34], which integrates CCS-like specifications for the structural part with algebraic specifications for the data type component.

In the SPO-approach, attributes have been integrated in [4] (see [35] for a corresponding extension of the DPO-approach). This integration preserves the fundamental derivation concept of the algebraic approach i.e., both the manipulation of the graphical structure and the calculation of the new attributes are combined within a (single) pushout construction. In [4] attributes were specified using algebraic specifications in the sense of [36]. Algebraic specifications provide a well-established formalism for treating data-types, variables, evaluations and substitutions. Moreover not only a set of types is available as attributes, but we can make use of the operations of the specification in order to indicate abstractly relationships between types. The proposal for the integration of attributes into graph transformation reviewed here is a simplification of the approach in [4]. It has already been used in [37].

Before introducing formally the concept of an attributed graph, we need to introduce some basic notions of universal algebra. A *signature* $Sig = (S, OP)$ consists of a set of sorts and a family of sets $OP = (OP_{w,s})_{w \in S^*, s \in S}$ of operation symbols. For $op \in OP_{w,s}$, we also write $op : w \to s$. A *Sig-algebra* A is an S-indexed family $(A_s)_{s \in S}$ of carrier sets together with an OP-indexed family of mappings $(op^A)_{op \in OP}$ such that $op^A : A_{s_1} \times \ldots \times A_{s_n} \to A_s$ if $op \in OP_{s_1 \ldots s_n, s}$. If $w = s_1 \ldots s_n \in S^*$, we sometimes write A_w for $A_{s_1} \times \ldots \times A_{s_n}$. A *Sig-homomorphism* $f : A \to B$ between two Sig-algebras A and B is a sort-indexed family of total mappings $f = (f_s : A_s \to B_s)_{s \in S}$ such that $op^B(f(x)) = f(op^A(x))$ for all $x \in A_s$. The category $\boldsymbol{Alg(Sig)}$ has as objects all Sig-algebras and as morphisms all total homomorphisms between them. It is well-know that $\boldsymbol{Alg(Sig)}$ has all colimits. $\mathcal{U} : \boldsymbol{Alg(Sig)} \to \boldsymbol{Set^P}$ is a functor assigning to each Sig-algebra A the disjoint union of its carrier sets A_s, and to each homomorphism f the disjoint union the total functions f_s, for all $s \in S$.

Attributes are labels of graphical objects taken from an *attribute algebra*. Hence, an attributed graph consists of a (labeled) graph and an attribute algebra, together with some *attribute functions* connecting the graphical and the algebraic part.

Definition 41 (attributed graph). Given a label alphabet L and a signature $Sig = (S, OP)$. Then $AG = (AG_V, AG_E,\ s^{AG}, t^{AG},\ lv^{AG}, le^{AG},\ AG_A,\ av^{AG}, ae^{AG})$ is a *Sig-attributed graph*, where

i) $AG_G = (AG_V, AG_E, s^{AG}, t^{AG}, lv^{AG}, le^{AG})$ is an L-labeled graph with labeling functions lv^{AG}, le^{AG} for vertices and edges (cf. Definition I.6),

ii) AG_A is a Sig-algebra,

iii) $av^{AG} : AG_V \to \mathcal{U}(AG_A)$ and $ae^{AG} : AG_E \to \mathcal{U}(AG_A)$ are the vertex and edge attributing functions

A *(Sig-attributed graph) morphism* between two Sig-attributed graphs $AGi = (AGi_V, AGi_E, s^{AGi}, t^{AGi}, lv^{AGi}, le^{AGi}, AGi_A, av^{AGi}, ae^{AGi})$ for $i = 1, 2$ is a tuple $f = (f_G, f_A)$ where $f_G = (f_V, f_E)$ is a partial graph morphism, and f_A is a total algebra homomorphism such that $\forall v \in dom(f_V) \cdot \mathcal{U}(f_A)(av^{AG1}(v)) = av^{AG2}(f_V(v))$ and $\forall e \in dom(f_E) \cdot \mathcal{U}(f_A)\ (ae^{AG1}(e)) = ae^{AG2}(f_E(e))$; f is total (injective) if f_G and f_A are total (injective). □

Proposition 42 (category $\boldsymbol{AGraph^P}$). *Sig-attributed graphs and Sig-attributed graph morphisms form a category, called* $\boldsymbol{AGraph^P}$.

Proof. The proof is based on the fact that the composition of morphisms is well-defined. □

For the definition of direct derivations of attributed graphs as single-pushout constructions, it is essential that $\boldsymbol{AGraph^P}$has pushouts. The construction of pushouts in $\boldsymbol{AGraph^P}$ can be seen as a special case of [4]: pushouts are constructed componentwise in $\boldsymbol{Graph^P}$ and $\boldsymbol{Alg(Sig)}$. Actually, the category $\boldsymbol{Agraph^P}$ has not only pushouts but all colimits (due to the fact that coproducts in $\boldsymbol{Agraph^P}$ can be constructed componentwise in $\boldsymbol{Set}$). As most of the results presented in Section 3 are based on colimit constructions, they should carry over to the $\boldsymbol{Agraph^P}$ setting. For a proof of the following theorem we refer to [4].

Theorem 43. *The category* $\boldsymbol{AGraph^P}$ *has pushouts.* □

Productions, grammars, matches, and (direct) derivations using attributed graphs are defined analogously to the Definitions 2 and 6 in Section 2, by

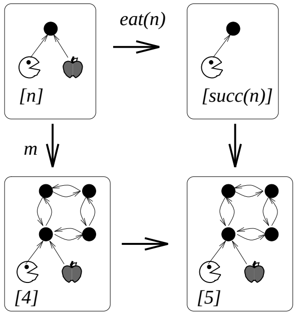


Figure 20: Counting Pacmans apples.

replacing the category $\boldsymbol{Graph^P}$ of graphs and partial graph morphisms by some category $\boldsymbol{AGraph^P}$ of attributed graphs.

Below we extend the Pacman example (Example 1 of Section 2) using attributes for counting the apples Pacman has eaten.

Example 14 (attributed graph transformation). For the graphical part AG_G we use the same labeled graphs as in Example 1. The signature of natural numbers

Signature Nat: **sorts** nat
opns $0 \to nat$
$succ : nat \to nat$
$+ : nat \times nat \to nat$

is used as the signature for the attribute algebras AG_A.

Attributes in productions are usually taken from “syntactical” (term) algebras. Figure 20 shows the attributed graph production $eat(n)$, where the left– and right-hand side graphs are attributed over the term algebra $T_{Nat}(X)$ with variables in $X = \{n\}$. The graphical part is similar to production eat in Figure 1 on page 40.[d] On the left-hand side, Pacman is attributed with the variable n representing the number of apples Pacman has eaten before the application of the production. On the right-hand side, n is replaced by $succ(n)$, i.e., the number of apples increases by one.

For the graphs to be rewritten, we fix some “semantical algebra”, the algebra $I\!N$ of natural numbers. As a transformation of labeled graphs does not change the label set, this algebra is preserved by an attributed graph transformation as well. Figure 20 shows an application of the production $eat(n)$ to a graph representing a state where Pacman has already eaten four apples, and is about to eat the fifth one. The graphical part is matched as usual. Then we have to find an assignment to the variable n which is compatible with the matching of the graphical part, i.e., n is mapped to 4. The derived graph is constructed componentwisely by taking the pushout in $\boldsymbol{Graph^P}$ for the graphical part, and by extending the assignment $n \mapsto 4$ to $succ(n)$, leading to $succ(n) \mapsto 5$. In this way, attributes in the derived graphs are calculated from attributes in the given graphs. □

5.2 Graph Structures and Generalized Graph Structures

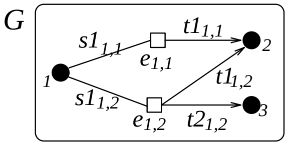
In order to be more flexible in system modeling, often hypergraphs are used instead of graphs [38,39,40], see also [41] in this handbook.

Definition 44 (hypergraph). A *hypergraph* $G = (V, E, s, t)$ consists of a set of vertices V, a set of edges E, and two total mappings $s, t : E \to V^*$ from the set of edges into the free monoid over the set of vertices which provide each hyperedge $e \in E$ with a sequences of n source and m target vertices $s(e) = v_1 \dots v_n$ and $t(e) = v_1 \dots v_m$, respectively. □

Since in the algebraic approaches graphs are considered as algebras w.r.t. a certain signature, also hypergraphs are defined in this way. The signature for hypergraphs is given below, where hyperedges are sorted according to their number of source vertices n and target vertices m.

Signature $HSig$: **sorts** $V, (E_{n,m})_{n,m \in I\!N}$
opns $(s_1, \dots, s_n, t_1, \dots, t_m : E_{n,m} \to V)_{n,m \in I\!N}$

Example 15 (hypergraph). In the following picture a concrete hypergraph of this signature is shown, having 3 non-empty carrier sets, namely $G_V = \{\bullet_1, \bullet_2, \bullet_3\}$, $G_{E1,1} = \{e_{1,1}\}$, and $G_{E1,2} = \{e_{1,2}\}$.



□

Generalizing partial graph morphisms, a partial homomorphism $f : G \to H$ between two algebras can be defined as total homomorphism $f! : dom(f) \to H$ from some subalgebra $dom(f) \subseteq G$. For each signature Sig this leads to a category of algebras and partial morphisms $\boldsymbol{Alg(Sig)^P}$. In [5] it was shown that this category is cocomplete if and only if all operations in Sig are unary, which motivated the following definition.

[d] Since the compatibility conditions for attributed graph morphisms (cf. Definition 41) do not allow to change attributes, carrier loops have to be introduced at attributed vertices, which are deleted and re-generated each time an attribute is modified. In order to simplify the presentation, these carrier loops are often omitted in the drawings.

Definition 45 (graph structure signature, graph structure). A **graph structure signature** GS is a signature which contains unary operator symbols only. A **graph structure** is a GS-algebra. □

Most of the results presented in Section 3 have originally been elaborated for graph structures [8,5], which do not only include graphs and hypergraphs, but also hierarchical graphs and higher order graphs (having edges between edges).

An even larger framework is obtained by the concept of generalized graph structures [17]. In order to explain this let us reconsider Definition 44, where a hypergraph is given by two carrier sets G_V, G_E and two operations $s^G, t^G : G_E \to G_V^*$. Such an algebra can not be defined directly as a graph structure since the mappings s^G, t^G are not between the carriers but from the set of edges G_E to the free monoid G_V^* over the set of vertices G_V. This has led to the infinite graph structure signature for hypergraphs above. Using generalized graph structures instead we may realize directly the hypergraphs of Definition 44: A hypergraph is an algebra $G = (G_V, G_E, s^G, t^G : F_E(G_E) \to F_V(G_V))$, where G_V and G_E are sets of vertices and hyperedges, and s^G, t^G are operations between sets derived from the carrier sets by application of the functors F_E and F_V, associated with the sort symbols E and V, respectively. The functor F_E is the identity functor, i.e., $F_E(G_E) = G_E$, and the functor F_V maps every set of vertices to its free monoid, i.e., $F_V(G_V) = G_V^*$.

On morphisms (partial functions) these functors are defined as follows: F_E is the identity functor, i.e., $F_E(f_E) = f_E$. $F_V(f_V) = f_V^*$ shall be defined pointwisely: for each sequence of vertices $l = v_1 v_2 \ldots v_n \in G_V^*$ $f_V^*(l) = f_V(v_1) f_V(v_2) \ldots f_V(v_n) \in H_V^*$ if $f_V(v_i)$ is defined for all $i = 1 \ldots n$, and otherwise it is undefined.

This allows to define GGS-morphisms. Figure 21 depicts a partial morphism $f = (f_E, f_V) : G \to H$ between generalized graph structures $G = (G_V, G_E, s^G, t^G)$ and $H = (H_V, H_E, s^H, t^H)$. It consists of a pair of mappings $f_E : G_E \to H_E$ and $f_V : G_V \to H_V$ between edges and vertices of G and H, respectively, such that the top diagram commutes for each operation symbol, i.e. $f_V^* \circ s^G \circ f_E? = s^H \circ f_E!$ and $f_V^* \circ t^G \circ f_E? = t^H \circ f_E!$, where $f_E? : dom(f_E) \to G_E$ and $f_E! : dom(f_E) \to H_E$ form the span-representation of f_E.

Thus generalized graph structures may be described as comma categories w.r.t. certain signatures where the compatibility requirement of morphisms has been relaxed in order to allow partial morphisms ('weak homomorphisms'). Analogously to comma-categories, colimits in GGS-categories can be constructed componentwisely followed — in the GGS-case — by a free construction ('totalization').

$$\begin{tikzcd}
 & G_V^* \arrow[rrr, dashed, "f_V^*"] \arrow[dr, phantom, "="] & & & H_V^* \\
G_E \arrow[ur, shift left, tail, "s^G"] \arrow[ur, shift right, tail, "t^G"'] & & dom(f_E) \arrow[ll, dashed, hook', "f_E?"'] \arrow[r, dashed, "f_E!"] & H_E \arrow[ur, shift left, tail, "s^H"] \arrow[ur, shift right, tail, "t^H"'] & \\
 & G_V \arrow[uu, "F_V"] \arrow[rrr, dashed, "f_V"'] & & & H_V \arrow[uu, "F_V"'] \\
G_E \arrow[uu, "F_E"] \arrow[rrr, dashed, "f_E"'] & & & H_E \arrow[uu, "F_E"'] &
\end{tikzcd}$$

Figure 21: Definition of a partial GGS morphism $f : G \to H$

The concept of GGS provides a constructive approach for generating internally structured categories of graph-like structures and partial morphisms from simpler ones. Among the large number of examples there are labeled and unlabeled graphs, hypergraphs, attributed graphs[e], attributed and labeled hypergraphs, hierarchical graphs, graph-interpreted (or typed) graphs, graph grammars, Petri Nets, and Algebraic High-Level (AHL) Nets. Properties of these categories, like cocompleteness, can be inferred from properties of the components and the way in which they are composed.

5.3 High-Level Replacement Systems

The formal investigation of graph grammars based on different kinds of graphs has often led to similar results based on similar proofs. This fact gave raise to the question whether it is possible to 'reuse' the proofs from one kind of graph grammar to obtain analogous results in another kind. The definition of *High-Level Replacement Systems* [7], shortly HLR systems, was the answer to this question. They provide an abstract framework in which not graphs, but objects of an arbitrary (instance) category are transformed (generalizing the ideas of graph grammars to arbitrary categorical grammars). Obviously, not every instance category gives raise to the same results; they depend very much on the properties of these categories. Hence, due to its categorical nature, the focus of HLR systems is not on the structure of objects (as in the GGS

[e] Recall that attributed graphs cannot be seen as graph structures, due to the fact that arbitrary signatures, like of booleans and natural numbers, may have non-unary operations.

approach) but on the properties of its instance categories. Minimal conditions are extracted which allow for definitions and theorems concerning, for example, parallelism, embedding, or amalgamation, and in this way such theorems can be proven for the biggest class of instance categories. Originally HLR systems were defined following the DPO approach [42]. Here we present the basic definitions of HLR systems of type SPO, developed in [7]. In the following, HLR systems are assumed to be of type SPO.

The basic assumption of HLR systems is that the structures we want to transform belong to some category ***Cat***. The theory of HLR systems defines the basic concepts of production, derivation and grammar in a generic (categorical) way: a production is a morphism in ***Cat***, a match belongs to a distinguished class of morphism, and direct derivations are given by pushouts. For example, if $\boldsymbol{Cat}$=$\boldsymbol{Graph^P}$, we obtain the SPO framework as introduced in Section 2.

Definition 46 (HLR system). Let ***Cat*** be a category and ***Match*** be a subcategory of ***Cat*** such that ***Match*** and ***Cat*** coincide on objects.

1. A **production** $r : L \to R$ is a ***Cat*** morphism.

2. An object G can be **directly derived** to an object H using a production $r : L \to R$ if there is a pushout (1) in ***Cat*** such that m is a **match**, i.e., it belongs to ***Match***.

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

3. A **derivation** is a sequence of direct derivations.

4. An **HLR system** $HLRS = (I, P, T)$ over ***Cat*** is given by a start object I, a set of productions P, and a class of terminal objects $T \subseteq \boldsymbol{Cat}$.

□

The HLR framework covers several concrete rewrite formalisms, which are obtained by choosing a concrete category ***Cat*** of structures, satisfying some basic conditions. The following conditions ensure that the well-known parallelism results are valid in the instance category ***Cat***.

Definition 47 (SPO conditions and categories). The following conditions (1)–(4) are called *SPO-conditions for parallelism of HLR systems.* A category ***Cat*** together with a subcategory ***Match*** as given in Definition 46 is called *SPO-category* if these conditions are satisfied.

1. Existence of pushouts with ***Match*** morphisms, i.e. ***Cat*** has pushouts if at least one morphism is in ***Match***.

2. ***Match*** has finite coproducts, which are preserved by the inclusion functor $\subseteq$: $\boldsymbol{Match} \to \boldsymbol{Cat}$.

3. Prefix closure of coproducts.[f]

4. Prefix closure of ***Match***, i.e. if $f \circ g \in \boldsymbol{Match}$ then $g \in \boldsymbol{Match}$.

□

Examples for SPO-categories are $\boldsymbol{Graph^P}$, $\boldsymbol{Alg(GS)^P}$ (with GS being a graph structure signature), $\boldsymbol{SPEC^P}$ (category of algebraic specifications and strict partial morphisms) [7]. The interpretation of productions and derivations depends on the instance category. For example, transformations of algebraic specifications may be interpreted as interconnection of modules in the context of modular system design [43]. In [7] it is shown that slightly different versions of the Local Church Rosser Theorem 14 and the Parallelism Theorem 16 hold in HLR-systems over SPO-categories, and it is an interesting topic for further research to generalize also the other results of Section 3 to the HLR-framework.

6 Comparison of DPO- and SPO-Approach

Section I.2[g] provided an informal introduction in the algebraic theory of graph rewriting, where many relevant concepts and results have been introduced in terms of problems. In this section the solutions to these problems – given throughout [2] and this chapter for the DPO and SPO approaches, respectively – are summarized and compared with each other. Working out the similarities and the differences of the corresponding results will allow to understand the relationships between the two algebraic approaches.

Since in Section I.2 the concepts are introduced independently of a particular approach, this provides us with an abstract terminology that can be considered as a common "signature" for (algebraic) graph rewriting approaches. This signature has sort symbols for graphs, productions, and derivations, etc., and operation or predicate symbols for (direct) derivation, parallel and sequential composition, etc., and the various relations that may hold between

[f] A morphism $p : L \to P$ is called a *prefix* of $r : L \to R$ if there is $x : P \to R$ such that $x \circ p = r$. This type of properties which make reference to prefixes of productions and requires some closure properties for all prefixes is typical for the theory of HLRS. It is the device to control the interaction of "partiality" and "totality" of morphisms in ***Cat*** resp. ***Match*** (see [7] for more details).

[g] Recall, that numbers preceded by "I." (for Part I) refer to Chapter [2] in this handbook, on the DPO-approach.

productions or derivations like the subproduction or the embedding relation. Most of the problems of Section I.2 ask for conditions, i.e., relations on derivations, ensuring that some construction is defined for these derivations. The parallelization condition of Problem I.2, for example, which is a unary relation on two-step sequential derivations, ensures that the synthesis construction is defined, leading to an equivalent direct parallel derivation.

In [2] and this chapter, the signature of Section I.2 is interpreted in the DPO- and SPO-approach, respectively, by providing a definition for its sorts, operation and predicate symbols in terms of the productions, derivations and constructions, etc., of the approach. Hence, the DPO- and the SPO-approach can be considered as models of the same signature. The interpretation of the parallelization condition of Problem I.2.2 in the DPO-approach, for example, is given by Definition I.13 of sequential independence, and the Synthesis Lemma I.20 shows that this interpretation is indeed a solution to Problem I.2.2. In a similar way, many of the definitions of [2] and this chapter can be seen as interpretations of the signature of Section I.2, and the corresponding results show that they satisfy the statement of the problems.

This view allows for two different ways of comparing the two algebraic approaches. The first one is w.r.t. to their properties: A property of an approach is a statement using the abstract terminology of Section I.2 that is valid in this particular approach. The basic property of the SPO-approach, for example, is the completeness of its direct derivations, that is, given a match for a production there is always a corresponding direct derivation. DPO direct derivations are not complete because of the gluing condition. On the other hand, in the DPO-approach we have that direct derivations are invertible, which is not true in the SPO-approach. Summarizing in this way the valid statements leads to a nice abstract characterization of the two algebraic approaches in terms of their properties.

It is well-known that models of the same signature can be compared by homomorphisms. In our case the signature of Section I.2 defines a notion of "homomorphism" between graph rewriting approaches being interpretations of this signature. Such a homomorphism is given by mappings between the carriers, i.e., productions, matches, derivations, etc., which are compatible with the interpretation of the operation and predicate symbols, i.e., with the parallel or sequential composition of productions, for example. Thereby, it embeds one approach in the other approach, which is necessary in order to compare directly the concepts of the two approaches. Below we show how the DPO-approach may be embedded in the SPO-approach. This applies not only to the basic concepts, like productions and direct derivations, but also to the constructions and results of the theory, like the parallel production and the parallelism theorem. Beside the direct comparison of corresponding concepts, such an embedding can be used in several other ways. On the one hand, we may transfer theoretical results, concerning for example analysis techniques for graph grammars, between the two approaches. On the other hand, it shows that we may simulate the constructions of the DPO approach within the SPO approach, which may be quite useful if we aim at a common implementation for the algebraic approaches. Finally, if we restrict the SPO approach by suitable application conditions (as introduced, for example, in Section 4), we may use the idea of a homomorphism in order to show the equivalence of the DPO-approach and the correspondingly restricted SPO-approach.

The section is organized like Section I.2. In each of the following sections we summarize the solutions to the problems of Section I.2, state the corresponding properties of the approaches, and discuss the embedding of the DPO- in the SPO-approach w.r.t. to the concepts of this section.

6.1 Graphs, Productions, and Derivations

According to Section I.2.1 each approach to graph rewriting is distinguished by three characteristics: its notion of a graph, the conditions under which a production may be applied, and the way the result of such an application is constructed. These characteristics define what is called a direct derivation. The DPO- and SPO-approach use the same kind of graphs (see Definition I.6). The productions of the two approaches are essentially the same except of differences in the representation: A DPO-production $L \xleftarrow{l} K \xrightarrow{r} R$ is a span of total injective graph morphism (compare Definition I.7). A SPO-production is instead a partial graph morphism $L \xrightarrow{p} R$, i.e., a total graph morphism $dom(p) \xrightarrow{p!} R$ from some subgraph $dom(p)$ of L to R (cf. Definitions 2 and 1). Both concepts of production have been extended by a name, which is used for identifying the production and for storing its internal structure (in case of a composed production).

Definition 48 (translation of SPO- and DPO-productions). Let $p : (L \xrightarrow{s} R)$ be a SPO-production. Then $\mathcal{D}(p) : (L \xleftarrow{l} dom(r) \xrightarrow{r} R)$ denotes its **translation to a DPO-production**, where l is the inclusion of $dom(s)$ in L and r is the domain restriction of s to $dom(s)$. Conversely, let $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ be a DPO-production. Then, $\mathcal{S}(p) : (L \xrightarrow{s} R)$ denotes its **translation to a SPO-production**, where $dom(s) = l(K)$ and $s = r \circ l^{-1}$. The partial morphism s is well-defined since l is supposed to be injective in Definition I.7. □

Hence, SPO-productions can be considered as DPO-productions where the interface graph K is represented in a unique way as a subgraph of the left-hand side graph L. The mapping $\mathcal{S}$ of DPO-productions p to SPO-productions $\mathcal{S}(p)$ will be used throughout this section to translate the concepts and results of the DPO-approach to the corresponding ones of the SPO-approach. The most basic concept of each graph rewriting approach is that of a direct derivation. In fact, the mapping $\mathcal{S}$ translates each direct DPO-derivation to a direct SPO-derivation where the match is d-injective and d-complete (compare Definition 7).

Proposition 49 (translation of DPO-derivations). *Let $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ be a DPO-production, $\mathcal{S}(p) : L \to R$ its translation to a SPO-production, and $L \xrightarrow{m} G$ a match for p into a graph G. Then m is also a match for $\mathcal{S}(p)$ and there is a direct DPO-derivation $d = (G \overset{p,m}{\Longrightarrow} H)$ if and only if there is a direct SPO-derivation $\mathcal{S}(d) = (G \overset{\mathcal{S}(p),m}{\Longrightarrow} H)$ such that m is a d-injective and d-complete match for $\mathcal{S}(p)$. In this way, the mapping $\mathcal{S}$ is extended from productions to derivations.*

Proof. See [5]. □

Not every direct derivation in the SPO-approach can be obtained from a DPO-derivation using the translation $\mathcal{S}$. In contrast to DPO-derivations a direct SPO-derivation $G \overset{p,m}{\Longrightarrow} H$ always exists if there is a match m for a production p. This first and most important property distinguishing the DPO- and the SPO-approach is called *completeness of direct derivations*. Indeed, most of the differences between the DPO- and the SPO-approach are caused by the fact that SPO derivations are complete while DPO derivations are not.

Property 50 (completeness of SPO-derivations). *Direct derivations in the SPO-approach are* **complete**, *i.e., for each production $p : L \to R$ and each match $L \xrightarrow{m} G$ for p into a graph G there is a direct derivation $G \overset{p,m}{\Longrightarrow} H$.* □

Because of the gluing condition (see Proposition I.9), DPO derivations are not complete. In fact, being complete essentially means to be free of any application conditions, that is, SPO derivations with application conditions as introduced in Section 4 are incomplete as well. The gluing condition, however, causes another interesting property of the DPO-approach, the *invertibility of direct derivations*.

Property 51 (invertibility of DPO-derivations). *Direct derivations in the DPO-approach are* **invertible**, *i.e., for each direct derivation $G \overset{p,m}{\Longrightarrow} H$ using production $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ there is an inverse derivation $H \overset{p^{-1},m^*}{\Longrightarrow} G$ using the inverse production $p^{-1} : (R \xleftarrow{r} K \xrightarrow{l} L)$, where $R \xrightarrow{m^*} H$ is the co-match of $G \overset{p,m}{\Longrightarrow} H$.*

Proof. See [44]. □

Constructing the inverse of a direct derivation can be seen as a kind of "undo". SPO direct derivations are not invertible in general since they may model implicit effects, like the deletion of dangling edges, which are not invertible. Because of these "side effects", SPO-derivations are more difficult to understand and to control than DPO-derivations and may be considered, in certain situations, as "unsafe". If direct derivations are invertible, there are no implicit effects. Hence, DPO derivations show a "safe" behavior that is easier to determine beforehand. In view of the overall complexity of the formalism, this is important if people from other areas shall use graph rewriting techniques for modeling their problems.

Most graph grammar approaches are not dedicated to a particular application area but may be considered as "general purpose" formalisms. For many applications, however, a tailored approach is more adequate which is as powerful as necessary and as simple as possible, in order to allow for a natural modeling of the problems. Then, standard application conditions like in the DPO-approach are needed to restrict the expressiveness of the approach if necessary. Exceptions of these conditions, however, should be possible as well, if they are explicitly specified by the user. The graph grammar based specification language PROGRES [45] (see also [46] in this handbook) provides an example of this concept, where e.g. injective matches are standard, but identification of vertices may be explicitly allowed.

It depends on the choice of the standard application conditions and of the possible exceptions, which approach is the most adequate one for a particular application. A very general solution is provided by user-defined application conditions, as introduced in Section 4. In particular in the SPO setting they complement in a nice way the generality of the pure approach.

6.2 Independence and Parallelism

Interleaving

The Local Church-Rosser Problem I.1 asked for two conditions formalizing the concept of concurrent direct derivations from two different points of view:

1. Two alternative direct derivations $H_1 \overset{p_1,m_1}{\Longleftarrow} G \overset{p_2,m_2}{\Longrightarrow} H_2$ are concurrent if they are not in conflict (parallel independence).

2. Two consecutive direct derivations $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ are concurrent if they are not causally dependent, i.e., if there are also direct derivations $G \overset{p_2,m_2}{\Longrightarrow} H_2 \overset{p_1,m_1'}{\Longrightarrow} X$ (sequential independence).

For the DPO-approach these conditions are given the Definitions I.12 and I.13, and the Local Church-Rosser Theorem I.14 ensures that they indeed solve Problem I.1. The corresponding solution for the SPO-approach is provided by the Definitions 9 and 11 together with Theorem 14.

Both solutions formalize the same intuitive idea. Due to the different representation of productions in the DPO and SPO approach, however, the corresponding conditions are stated in a different way. The following proposition shows that they are equivalent via the correspondence of productions and derivations established in Definition 48 and Proposition 49.

Proposition 52 (equivalence of DPO and SPO independence). *Let $p_1 : (L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1)$ and $p_2 : (L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2)$ be two DPO-productions, and $\mathcal{S}(p_1) : L_1 \to R_1$ and $\mathcal{S}(p_2) : L_2 \to R_2$ their translation to SPO-productions. Two alternative direct DPO-derivations $H_1 \overset{p_1,m_1}{\Longleftarrow} G \overset{p_2,m_2}{\Longrightarrow} H_2$ are parallel independent in the sense of Definition I.12 if and only if the corresponding SPO-derivations $H_1 \overset{\mathcal{S}(p_1),m_1}{\Longleftarrow} G \overset{\mathcal{S}(p_2),m_2}{\Longrightarrow} H_2$ are parallel independent in the sense of Definition 9.*

Two consecutive direct DPO-derivations $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ are sequential independent in the sense of Definition I.13 if and only if the corresponding SPO-derivations $G \overset{\mathcal{S}(p_1),m_1}{\Longrightarrow} H_1 \overset{\mathcal{S}(p_2),m_2'}{\Longrightarrow} X$ are sequential independent in the sense of Definition 11.

Proof. According to Definition I.12 the direct DPO derivations $H_1 \overset{p_1,m_1}{\Longleftarrow} G \overset{p_2,m_2}{\Longrightarrow} H_2$ are parallel independent if $m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$. Using the translation $\mathcal{S}$ to SPO productions (see Definition 48) this is the case iff $m_1(L_1) \cap m_2(L_2) \subseteq m_1(dom(\mathcal{S}(p_1)) \cap m_2(dom(\mathcal{S}(p_2)))$ holds for the corresponding SPO derivations. The two direct SPO derivations $H_1 \overset{\mathcal{S}(p_1),m_1}{\Longleftarrow} G \overset{\mathcal{S}(p_2),m_2}{\Longrightarrow} H_2$ are parallel independent if $m_1(L_1) \cap m_2(L_2 - dom(\mathcal{S}(p_2))) = \emptyset$ and $m_2(L_2) \cap m_1(L_1 - dom(\mathcal{S}(p_1))) = \emptyset$, which is equivalent to $m_1(L_1) \cap m_2(L_2) \subseteq dom(\mathcal{S}(p_2)))$ and $m_2(L_2) \cap m_1(L_1(\subseteq dom(\mathcal{S}(p_1)))$, and hence to $m_1(L_1) \cap m_2(L_2) \subseteq m_1(dom(\mathcal{S}(p_1)) \cap m_2(dom(\mathcal{S}(p_2)))$.

In a similar way one shows the equivalence of the notions of sequential independence in the DPO and SPO approach. □

In Section I.2.2 independent derivations have been interpreted as to be concurrent in the interleaving model of concurrency. In this view, Proposition 52 states that both approaches allow for the same amount of "interleaving parallelism".

Recently, both algebraic approaches came up with a weaker, asymmetric notion of independence, ensuring that two alternative direct derivation can at least be sequentialized into one order. In the DPO-approach this concept has been introduced in [47] under the name "serializability", while in the SPO-approach we speak of "weak parallel independence" in Definition 9. In turn, the weak notion of sequential independence, introduced in Definition 11, ensures that the second direct derivation does not depend on the first one, that is, they may also occur in parallel.

Explicit Parallelism

In order to represent parallel computations in a more explicit way, Section I.2.2 assumed a parallel composition "+" of productions leading to the notions of parallel production and derivation. In the DPO- and SPO-approaches this operator is interpreted as the coproduct (disjoint union) of productions, see Definition I.15[h] and Definition 15, respectively. In both approaches, a direct parallel derivation is a direct derivation using the parallel production. With respect to these definitions, the Parallelism Problem I.2 asked for the relationship between parallel and sequential derivations. The basic requirement is, of course, that each two direct derivations which are concurrent in the interleaving model can be put in parallel using the explicit notion of parallelism. This property is called completeness of parallel derivations below. On the other hand, if explicit parallelism allows for exactly the same amount of concurrency as the interleaving model, we say that it is safe (w.r.t. interleaving parallelism). In this case, the effect of a parallel derivation can always be obtained by a sequential derivation, too. With the following DPO parallelism properties we summarize the DPO solution to the Parallelism Problem I.2 provided by the DPO Parallelism Theorem I.17.

Property 53 (parallelism properties of DPO). *Parallel derivations in the DPO-approach satisfy the following completeness and safety properties:*

Completeness: *Each sequentially independent DPO-derivation $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ satisfies the* parallelization condition, *i.e., there is an equivalent direct parallel DPO-derivation $G \overset{p_1+p_2,m}{\Longrightarrow} X$.*

[h] In fact, Definition I.15 introduces the parallel composition of an arbitrary, finite number of productions.

Safety: *Each direct parallel DPO-derivation* $G \overset{p_1+p_2,m}{\Longrightarrow} X$ *satisfies the* sequentialization condition, *i.e., there are equivalent, sequentially independent DPO-derivations* $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m'_2}{\Longrightarrow} X$ *and* $G \overset{p_2,m_2}{\Longrightarrow} H_2 \overset{p_1,m'_1}{\Longrightarrow} X$.

□

Due to the completeness of SPO direct derivations, also its parallel direct derivations are complete, i.e., a parallel production $L_1 + L_2 \overset{p_1+p_2}{\longrightarrow} R_1 + R_2$ is applicable to any match $L_1 + L_2 \overset{m}{\longrightarrow} G$. Therefore, each two alternative direct derivations $H_1 \overset{p_1,m_1}{\Longleftarrow} G \overset{p_2,m_2}{\Longrightarrow} H_2$ can be composed to a parallel direct derivation $G \overset{p_1+p_2,m_1+m_2}{\Longrightarrow} X$, regardless of their independence. In particular, this allows for the parallel composition of weakly (parallel or sequential) independent direct derivations using the same notion of parallel production. Hence, SPO parallel derivations are also complete w.r.t. weakly independent sequential derivations, which is shown by the implication from 1. to 2. in the Weak Parallelism Theorem 16. Contrastingly, in the DPO-approach, a more complex, so called synchronized (parallel) composition had to be developed in [47] in order to put two serializable (i.e., weakly parallel independent) direct derivations in parallel.

On the other hand, because of its generality, SPO parallelism is no longer safe w.r.t. to sequential derivations, neither in the symmetric nor asymmetric version. We can, however, distinguish those direct parallel derivations which may be sequentialized at least in one order by considering the corresponding pair of alternative derivations, which is shown by the implication from 2. to 1. in the Weak Parallelism Theorem 16.

Finally let us investigate the relationship between DPO and SPO parallelism using the translation $\mathcal{S}$ introduced in Section 6.1. The first important observation is, that this translation is compatible with parallel composition, that is, given DPO-productions p_1 and p_2, we have that $\mathcal{S}(p_1)+\mathcal{S}(p_2) = \mathcal{S}(p_1+p_2)$.[i] This gives us the possibility to compare in a more direct way the relationships between parallel and sequential derivations in the two approaches. The following proposition states that the mapping $\mathcal{S}$ preserves the sequentialization and the parallelization conditions and is compatible with the analysis and synthesis construction.

Proposition 54 (compatibility of $\mathcal{S}$ with analysis and synthesis). *Let* $d = (G \overset{p_1+p_2,m}{\Longrightarrow} X)$ *be a direct parallel DPO-derivation, and* $\rho_1 = (G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m'_2}{\Longrightarrow} X)$ *and* $\rho_2 = (G \overset{p_2,m_2}{\Longrightarrow} H_2 \overset{p_1,m'_1}{\Longrightarrow} X)$ *its sequentializations. Then* $\mathcal{S}(d) = (G \overset{\mathcal{S}(p_1)+\mathcal{S}(p_2),m}{\Longrightarrow} X)$ *may be sequentialized to* $\mathcal{S}(\rho_1) = (G \overset{\mathcal{S}(p_1),m_1}{\Longrightarrow} H_1 \overset{\mathcal{S}(p_2),m'_2}{\Longrightarrow} X)$ *and* $\mathcal{S}(\rho_2) = (G \overset{\mathcal{S}(p_2),m_2}{\Longrightarrow} H_2 \overset{\mathcal{S}(p_1),m'_1}{\Longrightarrow} X)$.

Moreover, let $\rho = (G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m'_2}{\Longrightarrow} X)$ *be a sequentially independent DPO-derivation and* $d = (G \overset{p_1+p_2,m}{\Longrightarrow} X)$ *the direct parallel DPO-derivation obtained by the synthesis construction. Then* $\mathcal{S}(\rho) = (G \overset{\mathcal{S}(p_1),m_1}{\Longrightarrow} H_1 \overset{\mathcal{S}(p_2),m'_2}{\Longrightarrow} X)$ *is sequentially independent, and the corresponding direct parallel SPO-derivation is* $\mathcal{S}(d) = (G \overset{\mathcal{S}(p_1+p_2),m}{\Longrightarrow} X)$.

Proof. Let d be the direct parallel DPO derivation above. Then there are parallel independent direct derivations $H_1 \overset{p_1,m_1}{\Longleftarrow} G \overset{p_2,m_2}{\Longrightarrow} H_2$. According to the analysis construction in the DPO approach (cf. Lemma I.19), the match m'_2 of the second direct derivation in ρ_1 is defined by $m'_2 = r^*_1 \circ k_1$ in the diagram below, where $p^*_1 : (G \overset{l^*_1}{\longleftarrow} D_1 \overset{r^*_1}{\longrightarrow} H_1)$ is the co-production of $G \overset{p_1,m_1}{\Longrightarrow} H_1$ and k_1 exists by the above parallel independence s.t. subdiagram (1) commutes (cf. Definition I.12).

$$\begin{tikzcd}
 & L_2 \arrow[d, "m_2"'] \arrow[dr, "k_1"] \arrow[drr, "m'_2"] \arrow[dr, phantom, "(1)"', near start] & & & \\
p^*_1 : (& G \arrow[rr, bend right, "\mathcal{S}(p^*_1)"'] & D_1 \arrow[l, "l^*_1"'] \arrow[r, "r^*_1"] & H_1 &)
\end{tikzcd}$$

Now let $\mathcal{S}(p^*_1)$ be the translation of p^*_1 a SPO production (cf. Definition 48). Then, $m'_2 = r^*_1 \circ k_1 = r^*_1 \circ (l^*_1)^{-1} \circ m_2 = \mathcal{S}(p^*_1) \circ m_2$ since $k_1 = (l^*_1)^{-1} \circ m_2$ by commutativity of (1) and $\mathcal{S}(p^*_1) = r^*_1 \circ (l^*_1)^{-1}$ by Definition 48. But this is exactly the definition of the match of the second direct derivation in the SPO Weak Parallelism Theorem 16, i.e., $G \overset{\mathcal{S}(p_1),m_1}{\Longrightarrow} H_1 \overset{\mathcal{S}(p_2),m'_2}{\Longrightarrow} X$ is indeed a sequentialization of $G \overset{\mathcal{S}(p_1)+\mathcal{S}(p_2),m}{\Longrightarrow} X$.

The second part of Proposition 54 can be shown in a similar way by exchanging m_2 and m'_2. □

6.3 Embedding of Derivations and Derived Productions

Section I.2.3 introduced two problems, the Embedding Problem I.3 and the Derived Production Problem I.4, and it has been anticipated that the solution to

[i] In fact, this is true only up to isomorphism since there are infinitely many isomorphic parallel productions for the same two elementary productions. This problem can be solved by assuming a fixed coproduct construction (like, for example, the disjoint union), which induces a coproduct functor as shown in Appendix A.1 of [2].

the first problem is based on the solution to the second. The Embedding Problem asked for a condition under which an embedding of a graph G_0 via a morphism $e_0 : G_0 \to X_0$ may be extended to a derivation $\rho = (G_0 \overset{p_1,m_1}{\Longrightarrow} \cdots \overset{p_k,m_k}{\Longrightarrow} G_k)$, leading to an embedded derivation $\delta = (X_0 \overset{p_1,n_1}{\Longrightarrow} \cdots \overset{p_k,n_k}{\Longrightarrow} X_k)$. The idea was, to represent the informations relevant for the embedding of ρ by a *derived production* $\langle\rho\rangle : G_0 \rightsquigarrow G_n$ such that there is an embedding ρ via e_0 if and only if the derived production $\langle\rho\rangle$ is applicable at this match. The corresponding constructions are given in Section I.6.1 for the DPO-approach and in Section 3.2 for the SPO-approach. In both cases, the derived production $\langle\rho\rangle$ is obtained as the sequential composition $\langle d_1\rangle; \ldots; \langle d_k\rangle$ of the directly derived productions $\langle d_i\rangle : G_{i-1} \rightsquigarrow G_i$ of the given derivation ρ. Then, horizontal and vertical composition properties of direct derivations ensure that the derived production is applicable at a given injective match if and only if the original derivation may be embedded along this morphism. This is formulated as the *derived production property* below, see Theorem I.44 and Theorem 23 for corresponding statements in the DPO- and SPO-approach, respectively.

Property 55 (derived production). *DPO- and SPO-approach satisfy the derived production property, that is, for each derivation* $\rho = (G_0 \overset{p_1}{\Longrightarrow} \cdots \overset{p_n}{\Longrightarrow} G_n)$ *and each injection* $G_0 \xrightarrow{e_0} X_0$ *there is a derived derivation* $X_0 \overset{\langle\rho\rangle}{\Longrightarrow} X_n$ *using the derived production* $\langle\rho\rangle : G_0 \rightsquigarrow G_n$ *if and only if there is an an embedding* $e : \rho \to \delta$ *of* ρ *into a derivation* $\delta = (X_0 \overset{p_1}{\Longrightarrow} \cdots \overset{p_n}{\Longrightarrow} X_n)$ *using the original sequence of productions* $p_1, \ldots, p_n$. □

The embedding problem is now reduced to the question if the derived production is applicable at the embedding morphism e_0. In the DPO-approach approach this means that a derivation ρ may be embedded via e_0 if this match satisfies the gluing condition of the derived production (which specializes to the dangling condition since e_0 is supposed to be injective). Since in the SPO-approach there are no further conditions for the applicability of a production but the existence of a match, the embedding condition is satisfied for each derivation $G_0 \overset{p_1,m_1}{\Longrightarrow} \cdots \overset{p_k,m_k}{\Longrightarrow} G_k$ and each embedding morphism $G_0 \xrightarrow{e_0} X_0$. Therefore, we say that the embedding of SPO derivations is *complete.*

Property 56 (completeness of SPO embedding). *The embedding of derivations in the SPO-approach is complete, i.e., for each derivation* $\rho = (G_0 \overset{p_1,m_1}{\Longrightarrow} \cdots \overset{p_k,m_k}{\Longrightarrow} G_k)$ *and each embedding morphism* $e_0 : G_0 \to X_0$ *there is a derivation* $\delta = (X_0 \overset{p_1,n_1}{\Longrightarrow} \cdots \overset{p_k,n_k}{\Longrightarrow} X_k)$ *and an embedding* $e : \rho \to \delta$. □

As anticipated above, the DPO-approach is not complete w.r.t. the embedding of derivations.

Finally it is worth stressing that the translation $\mathcal{S}$ of productions and derivations from the DPO-approach to the SPO-approach is also compatible with

the concepts of this section. For DPO-productions p_1 and p_2, for example, we have that $\mathcal{S}(p_1; p_2) = \mathcal{S}(p_1); \mathcal{S}(p_2)$ provided that the left-hand side is defined. As a consequence, we may show that $\mathcal{S}$ is compatible with the construction of derived productions, i.e., given a DPO-derivation ρ we have $\mathcal{S}(\langle\rho\rangle) = \langle\mathcal{S}(\rho)\rangle$.

6.4 Amalgamation and Distribution

The concepts of amalgamated and distributed derivations are informally introduced in Section I.2.4. Their formalization in the DPO approach is given in Section I.6.2 in the same chapter, while the SPO variants are introduced in Section 3.3. The main idea is more or less the same for both approaches: Synchronization of productions p_1 and p_2 is modeled by a common subproduction p_0, i.e., a production that is related to the elementary productions by compatible embeddings. The application of two synchronized productions (to a global state) is modeled by applying their amalgamated production, that is, the gluing of the elementary productions along their common subproduction.

A distributed graph $DG = (G_1 \xleftarrow{g_1} G_0 \xrightarrow{g_2} G_2)$ models a state that is splitted into two local substates, related by a common interface state. Gluing the local graphs G_1 and G_2 along their interface G_0 yields the global graph $\oplus DG = G_1 \oplus_{G_0} G_2$ of the system. The local states are consistent (with each other) if they form a total splitting of the global graph, i.e., if the interface embeddings g_1 and g_2 are total. The transformation of a distributed graph $DG = (G_1 \xleftarrow{g_1} G_0 \xrightarrow{g_2} G_2)$ is done by local direct derivations $d_i = (G_i \overset{p_i,m_i}{\Longrightarrow} H_i$ for $i \in \{0, 1, 2\}$ where $p_1 \overset{in^1}{\leftarrow} p_0 \overset{in^2}{\rightarrow} p_2$ are synchronized productions and $m_i : L_i \to G_i$ are compatible matches for p_i into the two local graphs and the interface graph. A distributed derivation is synchronous if the given and the derived distributed graphs are total splittings.

Distributed derivations in the DPO approach are synchronous by definition. The distributed gluing condition, which is used as a standard application condition for distributed derivations, ensures that no element in a local graph is deleted as long as it is referenced from the interface graph. This ensures that the derived distributed graph is again a total splitting. Hence, in the DPO approach, distributed derivations show the same safe behavior like “normal” direct derivations. The prize we have to pay is a global application condition, which may not be easy to check in a real distributed system.

A distributed derivation in the SPO approach always exists if there are compatible matches for synchronized productions in a distributed graph. Hence, also in the SPO approach, the basic property of direct derivations (i.e., their completeness) is resembled by distributed derivations. In contrast to the DPO approach, a distributed SPO derivation needs no global application conditions.

However, it may cause global effects: Deleting a vertex in a local graph which is referenced from other local components leads to partial interface embeddings. Constructing the corresponding global graph all dangling references are deleted. It depends very much on the problem to be solved, whether global conditions or global effects (i.e., DPO or SPO distributed derivations) are more appropriate. Relying on the distributed gluing condition certainly leads to a more abstract specification, which assumes that dangling references are avoided by some standard mechanism on a lower level. However, if the references itself are subject of the specification, as e.g. for garbage collection in distributed systems, we need to model explicitly the inconsistencies caused by the deletion of referenced items.

These observations are also reflected in the solutions to the Distribution Problem I.5. In the DPO approach, each distributed derivation satisfies the amalgamation condition, which means that each distributed computation can be observed from a global point of view. Vice versa, an amalgamated derivation can be distributed if its match can be splitted according to the splitting of the given graph and if this splitting satisfies the distributed gluing condition, see Theorem I.46. In the SPO approach, a distributed derivation may be amalgamated if at least the given distributed graph represents a consistent state. The distribution of an amalgamated derivation, on the other hand, requires only the existence of compatible matches. We do not state the corresponding properties here but refer to Theorem I.46 for the DPO approach and Theorem 30 for the SPO approach.

7 Conclusion

In [2] and this chapter we have presented the two algebraic approaches and compared them with each other. The double-pushout (DPO) approach was historically the first and has a built-in application condition, called the gluing condition, which is important in several application areas in order to prevent undesirable possibilities for the application of productions. The single-pushout (SPO) approach allows to apply productions without any application conditions, because the problem of dangling edges, for example, is solved by deletion of these edges. This is adequate in some application areas and problematic in other ones. In general it seems to be important to allow user-defined application conditions for productions in order to prevent application of productions in undesirable cases. In this chapter it is shown how to extend the SPO-approach to handle such user-defined application conditions, and in a similar way the DPO-approach could be extended.

In fact, the DPO-approach could be considered as a special case of the SPO-approach, because the SPO-approach with gluing condition is equivalent to the DPO-approach. However, as shown in the comparison of both approaches, they provide different solutions to the general problems stated in Section 2 of [2]. Moreover, it does not seem adequate to specialize all the results in the SPO-approach to the case with gluing condition, because the explicit proofs in the DPO-approach are much simpler in several cases. For this reason DPO and SPO should be considered as two different graph transformation approaches within the context of this handbook.

Finally, let us point out that most of the concepts and results presented in [2] and this chapter are concerned with a single graph transformation system. This might be called the "theory of graph transformation systems in the small" in contrast to structuring and refinement concepts combining and relating different graph transformation systems which may form a "theory of graph transformation systems in the large". The "theory in the large" is especially important if graph transformation concepts are used for the specification of concurrent, object oriented and/or distributed systems. In fact, both algebraic approaches seem to be most suitable to handle such "problems in the large", and first attempts and results have been presented within the last couple of years (see [48,49,50,51,52]). Moreover, we believe that graph transformation in the large will be a main topic of research and development in the future.

Acknowledgment The research results presented in this paper have been obtained within the ESPRIT Working Groups COMPUGRAPH (1989-1992) and COMPUGRAPH 2 (1992-1995). We are most grateful to Raoult and Kennaway for initial motivation of the single pushout approach and to the members of the working group for their contributions and stimulating discussions.

References

1. H. Ehrig, M. Pfender, and H. J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180, 1973.
2. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. *Algebraic Approaches to Graph Transformation I: Basic Concepts and Double Pushout Approach.* World Scientific, 1996. In this book.
3. A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. Accepted for special issue of Fundamenta Informaticae, 1996.
4. M. Löwe, M. Korff, and A. Wagner. An algebraic framework for the transformation of attributed graphs. In M.R. Sleep, M.J. Plasmeijer, and

M.C. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 14, pages 185–199. John Wiley & Sons Ltd, 1993.

5. M. Löwe. Algebraic approach to single-pushout graph transformation. *TCS*, 109:181–224, 1993.
6. M. Korff. Single pushout transformations of generalized graph structures. Technical Report RP 220, Federal University of Rio Grande do Sul, Porto Alegre, Brazil, 1993.
7. 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.
8. M. Löwe. *Extended Algebraic Graph Transformations*. PhD thesis, Technical University of Berlin, 1990. short version in TCS (109):181 – 224.
9. J. C. Raoult. On graph rewriting. *Theoretical Computer Science*, 32:1–24, 1984.
10. R. Kennaway. On "On graph rewriting". *Theoretical Computer Science*, 52:37–58, 1987.
11. J. Glauert, R. Kennaway, and R. Sleep. A categorical construction for generalised graph rewriting. Technical report, School of Information Systems, University of East Anglia, Norwich NR4 7TJ, U.K., 1989.
12. R. Kennaway. Graph rewriting in some categories of partial maps. In Ehrig et al. [53], pages 475–489. Lecture Notes in Computer Science 532.
13. E. Robinson and G. Rosolino. Categories of partial maps. *Information and Computation*, 79:95 – 130, 1988.
14. P.M. van den Broek. Algebraic graph rewriting using a single pushout. In *Int. Joint Conf. on Theory and Practice of Software Development (TAPSOFT'91), LNCS 493*, pages 90–102. Springer Verlag, 1991.
15. H. Herrlich and G. Strecker. *Category Theory*. Allyn and Bacon, Rockleigh, New Jersey, 1973.
16. M. Löwe and J. Dingel. Parallelism in single-pushout graph rewriting. *Lecture Notes in Computer Science 776*, pages 234–247, 1994.
17. M. Korff. *Generalized graph structure grammars with applications to concurrent object-oriented systems*. PhD thesis, Technical University of Berlin, 1995.
18. M. Korff. Minimality of derived rules in single pushout graph rewriting. Technical Report 94/10, Technical University of Berlin, 1994.
19. G. Taentzer. Towards synchronous and asynchronous graph transformations. Accepted for special issue of Fundamenta Informaticae, 1996.
20. H. Ehrig and M. Löwe. Parallel and distributed derivations in the single pushout approach. *Theoretical Computer Science*, 109:123 – 143, 1993. Also in Tech. Rep. 91/01, Technical University of Berlin.

21. H. Ehrig and B. K. Rosen. Parallelism and concurrency of graph manipulations. *Theoretical Computer Science*, 11:247–275, 1980.
22. 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.
23. G. Taentzer. Hierarchically distributed graph transformation. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94, LNCS* , 1996. Accepted.
24. G. Taentzer. *Parallel and Distributed Graph Transformation: Formal Description and Application to Communication-Based Systems*. PhD thesis, Technical University of Berlin, Dep. of Comp. Sci., 1996.
25. R. Heckel. Embedding of conditional graph transformations. 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.
26. R. Heckel. Algebraic graph transformations with application conditions. Master's thesis, TU-Berlin, 1995.
27. A. Wagner. On the expressive power of algebraic graph grammars with application conditions. In *Int. Joint Conf. on Theory and Practice of Software Development (TAPSOFT'95), LNCS 915*. Springer Verlag, 1995.
28. R. Heckel and A. Wagner. Ensuring consistency of conditional graph grammars – a constructive approach. *Proc. of SEGRAGRA'95 "Graph Rewriting and Computation", Electronic Notes of TCS*, 2, 1995. http://www.elsevier.nl/locate/entcs .
29. M. Korff. Graph-interpreted graph transformations for concurrent object-oriented systems. Extended abstract for the 5th International Workshop on Graph Grammars and their Application to Computer Science, 1994.
30. K. Räihä. Bibliography of attribute grammars. *SIGPLAN Notices*, 15(3):35–44, 1980.
31. C. Dimitrovici, U. Hummert, and L. Petrucci. Composition and net properties of algebraic high-level nets. In *Advances of Petri Nets*, volume 483 of *Lecture Notes in Computer Science*. Springer Verlag Berlin, 1991.
32. W. Reisig. Petri nets and algebraic specifications. *Theoretical Computer Science*, 80:1–34, 1991.
33. H. Ehrig, J. Padberg, and L. Ribeiro. Algebraic high-level nets: Petri nets revisited. In *Recent Trends in Data Type Specification*, pages 188–206, Caldes de Malavella, Spain, 1994. Springer Verlag. Lecture Notes in Computer Science 785.
34. ISO. Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering

of observational behaviour. International Standard ISO 8807, ISO, 1989.

35. G. Schied. *Über Graphgrammatiken, eine Spezifikationsmethode für Programmiersprachen und verteilte Regelsysteme.* Arbeitsberichte des Institus für mathematische Maschinen und Datenverarbeitung (Informatik), University of Erlangen, 1992.
36. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science.* Springer, Berlin, 1985.
37. M. Korff. True concurrency semantics for single pushout graph transformations with applications to actor systems. In *Working papers of the International Workshop on Information Systems – Corretness and Reusability IS-CORE'94*, pages 244–258, 1994. Tech. Report IR-357, Free University, Amsterdam.
38. A. Habel and H.-J. Kreowski. May we introduce to you: Hyperedge replacement. In *3rd Int. Workshop on Graph Grammars and their Application to Computer Science, LNCS 291*, Berlin, 1987. Springer Verlag.
39. A. Habel. *Hyperedge Replacement: Grammars and Languages.* PhD thesis, University of Bremen, 1989.
40. A. Habel. *Hyperedge replacement: Grammars and Languages*, volume 643 of *LNCS.* Springer Verlag, Berlin, 1992.
41. F. Drewes, H.-J. Kreowski, and Habel. *Hyperedge Replacement Graph Grammars.* World Scientific, 1996. In this book.
42. H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. From graph grammars to High Level Replacement Systems. In Ehrig et al. [53], pages 269–291. Lecture Notes in Computer Science 532.
43. F. Parisi-Presicce. Modular system design applying graph grammar techniques. In *ICALP'89.* Springer Lecture Notes in Computer Science, 1989.
44. H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *1st Graph Grammar Workshop, Lecture Notes in Computer Science 73*, pages 1–69. Springer Verlag, 1979.
45. A. Schürr. Progress: A vhl-language based on graph grammars. In *LNCS532.* Springer, 1991.
46. A. Schürr. *Programmed Graph Replacement Systems.* World Scientific, 1996. In this book.
47. A. Corradini and F. Rossi. Synchronized composition of graph grammar productions. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94, LNCS* , 1996. Accepted.
48. H.-J. Kreowski and S. Kuske. On the interleaving semantics of transformation units - a step into GRACE. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94, LNCS* , 196. Accepted.
49. H. Ehrig and G. Engels. Pragmatic and semantic aspects of a module concept for graph transformation systems. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94, LNCS* , 1996. Accepted.
50. G. Taentzer and A. Schürr. DIEGO, another step towards a module concept for graph transformation systems. *Proc. of SEGRAGRA'95 "Graph Rewriting and Computation", Electronic Notes of TCS*, 2, 1995. http://www.elsevier.nl/locate/entcs .
51. F. Parisi-Presicce. Transformation of graph grammars. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94, LNCS* , 1996. Accepted.
52. R. Heckel, A. Corradini, H. Ehrig, and M. Löwe. Horizontal and vertical structuring of typed graph transformation systems. Accepted for special issue of MSCS, 1996.
53. H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors. *4th International Workshop on Graph Grammars and Their Application to Computer Science.* Springer Verlag, 1991. Lecture Notes in Computer Science 532.

Index