

Parallel Independence in Hierarchical Graph Transformation

Annegret Habel¹ and Berthold Hoffmann²

¹ Carl-v.-Ossietzky-Universität Oldenburg, Germany
habel@informatik.uni-oldenburg.de

² Universität Bremen, Germany
hof@informatik.uni-bremen.de

Abstract. Hierarchical graph transformation as defined in [1, 2] extends double-pushout graph transformation in the spirit of term rewriting: Graphs are provided with hierarchical structure, and transformation rules are equipped with graph variables. In this paper we analyze conditions under which diverging transformation steps $H \leftarrow G \Rightarrow H'$ can be joined by subsequent transformation sequences $H \xrightarrow{*} M \xleftarrow{*} H'$. Conditions for joinability have been found for graph transformation (called parallel independence) and for term rewriting (known as non-critical overlap). Both conditions carry over to hierarchical graph transformation. Moreover, the more general structure of hierarchical graphs and of transformation rules leads to a refined condition, termed fragmented parallel independence, which subsumes both parallel independence and non-critical overlap as special cases.

1 Introduction

Graph transformation combines two notions that are ubiquitous in computer science (and beyond). Graphs are frequently used as visual models of structured data that consists of entities with relationships between them. Rules allow the modification of data to be specified in an axiomatic way. The book [3] gives a general survey on graph transformation, and [4, 5] describe several application areas.

When graph transformation is used to program or specify systems, it should be possible to group large graphs in a hierarchical fashion so that they stay comprehensible. Many notions of hierarchical graphs have been proposed, and several ways of transforming hierarchical graphs have been studied in the literature. See [6] for a rather general definition. This paper is based on [1], where double-pushout graph transformation [7] has been extended to a strict kind of hierarchical graphs where the hierarchy is a tree, and edges may not connect nodes in different parts of the hierarchy. This is adequate for programming; applications like software modeling may call for a looser notion of hierarchical graphs, e.g., the one in [8]. In [2], transformation rules have been extended to rules with variables [9]. This is done in the spirit of term rewriting [10], a

rule-based model for computing with expressions (trees): Rules are equipped with variables that may be instantiated by graphs, so that a single rule application may compare, delete, or copy subgraphs of arbitrary size. Hierarchical graph transformation with variables is the computational model of DIAPLAN, a language for programming with graphs and diagrams that is currently being designed [11].

In general, graph transformation is nondeterministic like other rule-based systems. Several rules may compete for being applied, at different places in a given graph. It is thus important to study under which conditions the result of a transformation sequence is independent of the order in which competing rules are applied. For term rewriting, parallel independence holds if steps have a non-critical overlap [10], and for double pushout graph transformation, the slightly stronger property of direct joinability holds if steps are parallelly independent [12]. These results carry over to hierarchical graph transformation. More precisely, we shall prove that they are special cases of the Fragmented Parallel Independence Theorem.

The paper is organized as follows. Section 2 collects basic notions of graphs and graph morphisms. In Sect. 3, we recall the basic notions of hierarchical graphs and hierarchical graph transformation, and show the relationship to substitution-based graph transformation. In Sect. 4, we discuss how independence results from graph transformation and term rewriting carry over to hierarchical graph transformation, and establish the Fragmented Parallel Independence Theorem. In Sect. 5, we conclude with a brief summary and with some topics for future work.

Acknowledgments. We thank the anonymous referees for their constructive remarks, and for their confidence.

2 Preliminaries

In the following, we recall standard notions of graphs and graph morphisms [7]. As in [9], we distinguish a subset of the label alphabet as variables. Variable edges are placeholders that can be substituted by graphs.

Let \mathcal{C} be an alphabet with a subset $X \subseteq \mathcal{C}$ of *variables* where every symbol l comes with a *rank* $\text{rank}(l) \geq 0$.

A *graph* (with variables in X) is a system $G = \langle V_G, E_G, \text{att}_G, \text{lab}_G \rangle$ with finite sets V_G and E_G of *nodes* (or *vertices*) and *edges*, an *attachment function* $\text{att}_G: E_G \rightarrow V_G^*$ ³, and a *labeling function* $\text{lab}_G: E_G \rightarrow \mathcal{C}$ such that the attachment $\text{att}_G(e)$ of every edge e consists of $\text{rank}(\text{lab}_G(e))$ nodes (that need not be distinct).

³ For a set A , A^* denotes the set of all sequences over A . The empty sequence is denoted by ε . For a mapping $f: A \rightarrow B$, $f^*: A^* \rightarrow B^*$ denotes the extension of f with $f^*(\varepsilon) = \varepsilon$ and $f^*(a_1 \dots a_k) = f(a_1) \dots f(a_k)$ for $a_1, \dots, a_k \in A$.

A *graph morphism* $g: G \rightarrow G'$ between two graphs G and G' consists of two functions $g_V: V_G \rightarrow V_{G'}$ and $g_E: E_G \rightarrow E_{G'}$ that preserve labels and attachments, that is, $lab_{G'} \circ g_V = lab_G$ and $att_{G'} \circ g_E = g_V^* \circ att_G$. It is *injective* (*surjective*) if g_V and g_E are injective (surjective), and an *isomorphism* if it is both injective and surjective. It is an *inclusion* if g_V and g_E are inclusions.

3 Hierarchical Graph Transformation

In this section, we define hierarchical graphs, hierarchical graph morphisms, and hierarchical graph transformation. For lack of space, we just recall the concepts devised in [1, 2]; a broader discussion of these concepts, and further references to the scientific literature can be found in these papers. At the end of the section, we relate our definitions to their origins, namely double-pushout graph transformation [7], and substitutive graph transformation [9].

A graph becomes hierarchical if its edges contain graphs, the edges of which may contain graphs again, in a nested fashion. Variables may not contain graphs; they are used as placeholders for graphs in transformation rules.

Definition 1 (Hierarchical Graph). The set $\mathcal{H}(X)$ of *hierarchical graphs* (with variables in X) consists of triples $H = \langle \widehat{H}, F_H, cts_H \rangle$ where \widehat{H} is a graph (with variables in X), $F_H \subseteq E_{\widehat{H}}$ is a set of *frame edges* (or just *frames*) that are labeled in $\mathcal{C} \setminus X$, and $cts_H: F_H \rightarrow \mathcal{H}(X)$ is a *contents function* mapping frames to hierarchical graphs.

A hierarchical graph I is a *part* of H if $I = H$, or if I is a part of $cts_H(f)$ for some frame $f \in F_H$. An X -labeled edge in some part I of H is called a *variable edge* of H .

The *skeleton* of a hierarchical graph H is obtained by removing all variable edges from all parts of H ; it is denoted by \underline{H} . $\text{Var}(H)$ denotes the set of *variables* occurring in the parts of H . A hierarchical graph H is *variable-free* if $H = \underline{H}$.

Example 1 (Control flow graphs). In simple control flow diagrams of sequential imperative programs, execution *states* are represented by nodes (depicted as small circles), and execution *steps* are represented by edges: statements (drawn as boxes) are labeled by assignments, and branches (drawn as diamonds) are labeled by conditions. Each step connects one predecessor state to one successor state (for assignments), or to two (for branches, distinguished by “ \oplus ” and “ \ominus ”, respectively). Hierarchies are used for representing procedure calls (drawn like assignments, but with doubled vertical lines). They contain control flow graphs of the procedures’ bodies. Since procedures may call other procedures, control flow graphs may be nested to arbitrary depth. In Fig. 7 below we show six hierarchical control flow graphs.

Definition 2 (Hierarchical Graph Morphism). A *top hierarchical graph morphism* (*top morphism*, for short) $h: H \rightarrow H'$ between two hierarchical graphs

H and H' is a pair $h = \langle \widehat{h}, M \rangle$ where $\widehat{h}: \widehat{H} \rightarrow \widehat{H}'$ is a graph morphism such that $\widehat{h}(F_H) \subseteq F_{H'}$, and $M = (h_f: \text{cts}_H(f) \rightarrow \text{cts}_{H'}(\widehat{h}(f)))_{f \in F_H}$ is a family of top morphisms between the contents of the frames. A *hierarchical graph morphism* $h: H \rightarrow H'$ is a top morphism $h': H \rightarrow H''$ between H and some part H'' of H' . A hierarchical graph morphism h is *injective* if the graph morphism \widehat{h} and all morphisms in M are injective; it is an *inclusion* if \widehat{h} and all morphisms in M are inclusions. A top morphism h is *surjective* if \widehat{h} and all morphisms in M are surjective. A top morphism $h: H \rightarrow H'$ is an *isomorphism* if it is injective and surjective; then we call H and H' *isomorphic*, and write $H \cong H'$.

Definition 3 (Substitution). A *substitution pair* $x \mapsto \langle H, p \rangle$ consists of a variable $x \in X$ and of a hierarchical graph H with a sequence $p \in V_H^*$ of *rank*(x) mutually distinct *points*. A finite set

$$\sigma = \{x_1 \mapsto \langle H_1, p_1 \rangle, \dots, x_n \mapsto \langle H_n, p_n \rangle\}$$

of substitution pairs is a *substitution* if the variables are pairwise distinct. Then $\text{Dom}(\sigma) = \{x_1, \dots, x_n\}$ is the *domain* of σ .

Let I be a hierarchical graph where the top graph \widehat{I}' of some part I' contains an edge e labeled with x . Then the application of a substitution pair $x \mapsto \langle H, p \rangle$ to e is obtained by replacing I' with a hierarchical graph constructed as follows: Unite \widehat{I}' disjointly with \widehat{H} , remove e , identify every point in p with the corresponding attached node in $\text{att}_{\widehat{I}'}(e)$, and preserve the contents of the frames. The *instantiation* of a hierarchical graph I according to a substitution σ is obtained by applying all substitution pairs in σ to all edges with a variable label in $\text{Dom}(\sigma)$ simultaneously, and is denoted by $I\sigma$.

Definition 4 (Rule). A *rule* $p = \langle L \leftarrow K \rightarrow R \rangle$ consists of two top morphisms with a common domain K . We assume that $K \rightarrow L$ and $K \rightarrow R$ are inclusions and that $\text{Var}(L) \supseteq \text{Var}(R)$.

The *instance* of a rule p for a substitution σ is defined as $p\sigma = \langle L\sigma \leftarrow \underline{K} \rightarrow \underline{R} \rangle$, and the *skeleton* of p is given by $\underline{p} = \langle \underline{L} \leftarrow \underline{K} \rightarrow \underline{R} \rangle$. A rule p is *variable-free* if $p = \underline{p}$. (We explain in App. A why we take the skeleton \underline{K} of the interface in the instance $p\sigma$, instead of $K\sigma$.)

Definition 5 (Hierarchical Graph Transformation). Consider hierarchical graphs G and H and a rule $p = \langle L \leftarrow K \rightarrow R \rangle$. Then G *directly derives* H *through* p if there is a double-pushout

$$\begin{array}{ccccc} L\sigma & \longleftarrow & \underline{K} & \longrightarrow & R\sigma \\ g \downarrow & (1) & \downarrow & (2) & \downarrow \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

for some substitution σ so that the vertical morphisms are injective. We write $G \Rightarrow_{p,\sigma,g} H$ or $G \Rightarrow_p H$ and call this a *direct derivation* where g is the hierarchical graph morphism $g: L \rightarrow G$ defining the *occurrence* of p in G .

A direct derivation $G \Rightarrow_{p,\sigma,g} H$ exists if and only if the occurrence $g:L \rightarrow G$ above satisfies the following *hierarchical dangling condition*: (i) The graph morphism \hat{g} satisfies the dangling condition for graph morphisms (see [7]), (ii) all morphisms in M satisfy the hierarchical dangling condition, and (iii) for all deleted frames $f \in F_L \setminus F_I$, the hierarchical morphism $g_f: cts_L(f) \rightarrow cts_G(g(f))$ is bijective.

Given a hierarchical graph G , a rule p as above, a substitution σ with $\text{Dom}(\sigma) = \text{Var}(L)$, and an occurrence g satisfying the hierarchical dangling condition, a direct derivation is uniquely determined by the following steps: (1) Remove $g(L\sigma - \underline{K})$ from the part G' of G where $g:L \rightarrow G'$ is top, yielding a hierarchical graph D , a top morphism $d:\underline{K} \rightarrow D'$ which is the restriction of g , and the inclusion $D' \rightarrow G'$. (2) Add $R\sigma$ disjointly to D' and identify the corresponding nodes and edges in \underline{K} and $d(\underline{K})$, yielding a hierarchical graph H' and top morphisms $D \rightarrow H$ and $R\sigma \rightarrow H$. (3) Obtain H by replacing H' for G' in G .

Remark 1 (Relation to Adhesive High-Level Replacement). Hierarchical graphs without variables and injective hierarchical graph morphisms form a category HiGraphs . We conjecture that the category $\langle \text{HiGraphs}, \mathcal{M} \rangle$ of hierarchical graphs without variables with the class \mathcal{M} of all injective top morphisms forms an adhesive HLR category. In this case, application of the general results in [13] would yield the Local Church-Rosser Theorems, the Embedding, Extension, and Local Confluence Theorem for $\langle \text{HiGraphs}, \mathcal{M} \rangle$. The statement no longer holds for the category $\langle \text{HiGraphs}(X), \mathcal{M} \rangle$ of hierarchical graphs with variables with the class \mathcal{M} of all injective top morphisms.

Example 2 (Transformation of Control Flow Graphs). In Figs. 1 and 2 we show two rules for transforming hierarchical control flow graphs. The rule *loop* removes duplicated code before a loop. The rule *inl* performs “inlining” of a procedure’s body for its call. In the figures, the images of the interface’s nodes and edges in the left- and right-hand side graphs can be found by horizontal projection.

Figure 7 shows transformations that use the rule *loop* in vertical direction, and the rule *inl* in horizontal direction, respectively. For applying *loop*, the variable D must be instantiated with the assignment “ $x := e$ ” in the right column, and by the procedure call edge containing that assignment in the other columns. For applying *inl*, its variable D must be instantiated with the control flow graphs representing the statements “ $x := e$ ” (in the transformations to the right), and “ $y := e'; z := e''$ ” (in the transformations to the left), respectively.

A rule is applied by instantiating its variables according to some substitution, and constructing a double-pushout for this instance.

In substitutive graph transformation [9], the application of a rule is determined entirely by instantiation with a substitution.

Definition 6 (Substitutive Graph Transformation). A *substitutive rule* $p^* = \langle L^*, R^* \rangle$ is a pair of hierarchical graphs. Given two hierarchical graphs

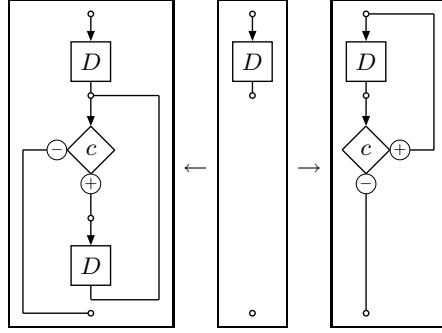


Fig. 1. The rule *loop*

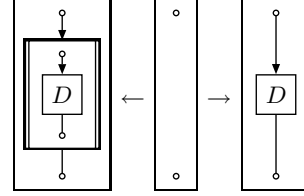


Fig. 2. The rule *inl*

G, H, G directly derives H through p^* , denoted by $G \Rightarrow_{p^*} H$, if there is a substitution σ^* such that $L^*\sigma^* \cong G'$ for some part G' of G , and H equals a copy of G wherein G' is replaced with $R^*\sigma^*$.

Every rule $p = \langle L \leftarrow K \rightarrow R \rangle$ induces a substitutive rule $p^* = \langle L^*, R^* \rangle$ as follows: Extend every part K' of K by a variable edge that is attached to all nodes in $V_{K'}$, and is labeled with a fresh variable label (of rank $|V_{K'}|$), and obtain the hierarchical graphs L^* and R^* by inserting this hierarchical graph for the occurrence of the part K' in L and R , respectively.

Theorem 1 (Substitutive Graph Transformation). *For all hierarchical graphs G, H and all rules p ,*

$$G \Rightarrow_p H \text{ if and only if } G \Rightarrow_{p^*} H .$$

Proof Sketch. Without loss of generality, p is a rule with discrete interface \underline{K} . (Otherwise, if we consider the modified rule p^- in which all non-variable edges are removed from K , we easily see that $G \Rightarrow_p H \Leftrightarrow G \Rightarrow_{p^-} H$.)

For a start, let us consider the case that the rules are applied on top level. First, let $G \Rightarrow_{p,\sigma,g} H$ be a direct derivation. Define σ^* as the extension of σ by the substitution pair $\{x \mapsto D\}$, where D is the intermediate hierarchical graph of the direct derivation. Then $L^*\sigma^* \cong G$, and $R^*\sigma^* \cong H$ and $G \Rightarrow_{p^*,\sigma^*} H$ is a direct substitutive derivation. Conversely, let $G \Rightarrow_{p^*,\sigma^*} H$ be a direct substitutive derivation. Then $L^*\sigma^* \cong G$ and $R^*\sigma^* \cong H$ for some isomorphisms $g^*: L^*\sigma^* \rightarrow G$ and $h^*: R^*\sigma^* \rightarrow H$. Let σ be the restriction of σ^* to $\text{Dom}(\sigma^*) - \{x\}$ and g be the restriction of g^* to $L\sigma$. Then there is a direct derivation $G \Rightarrow_{p,\sigma,g} H$.

Now, let the direct derivations apply to a part G' of G . Then both kinds of direct derivations construct a graph H wherein G' is replaced by a part H' with a direct top-level derivation.

Theorem 1 shows the close relationship between the double-pushout approach and the substitution-based approach. As a consequence, the main proofs can be done on a substitution-based level.

4 Parallel Independence

The term “parallel independence” has been coined for a criterion of commutativity (or the Local Church-Rosser property) in double-pushout graph transformation (see, e.g., [7]). The related area of term rewriting is about the transformation of terms, or trees, by rules with variables. Commutativity has been studied for term rewriting as well, along with a more general property, called joinability. Commutativity and joinability are important prerequisites for showing that a transformation mechanism has unique normalforms: If all competing direct derivations are commutative (joinable), transformation is strongly confluent (or locally confluent, resp.). Strongly confluent, and terminating locally confluent “abstract reduction systems” do have unique normalforms. (See, e.g., [10].)

We re-phrase commutativity and joinability for hierarchical graph transformation.

Definition 7 (Commutativity and Joinability). A pair of direct derivations $H \leftarrow G \Rightarrow H'$ of the same hierarchical graph is called *competing* if $H \not\cong H'$. Competing direct derivations are

- *commutative* if $H \Rightarrow M \leftarrow H'$, and
- *joinable* if $H \xRightarrow{*} M \xleftarrow{*} H'$,

for some hierarchical graph M , respectively. (See Figs. 3 and 4 below.)

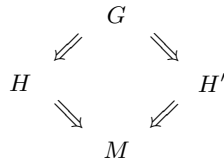


Fig. 3. Commutativity

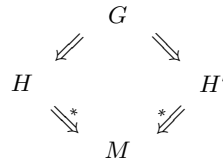


Fig. 4. Joinability

For double-pushout graph transformation it has been shown that commutativity holds if competing direct derivations are parallelly independent of each other (see, e.g., [12, 7, 14]). For term rewriting, the presence of variables in rules has made it necessary to study joinability. Term rewriting steps are joinable if they are non-critically overlapping.

We shall first demonstrate that both criteria, parallel independence as well as non-critical overlaps, carry over to hierarchical graph transformation. However, since hierarchical graphs generalize both graphs and terms, these criteria turn out to be special cases of a more general condition for joinability that will be discussed in the sequel.

General Assumption. In the following, let $H \leftarrow_{p,\sigma,g} G \Rightarrow_{p',\sigma',g'} H'$ be a pair of competing direct derivations using the rules $p = \langle L \leftarrow K \rightarrow R \rangle$ and $p' = \langle L' \leftarrow K' \rightarrow R' \rangle$.

The morphism g of the rule instance $L\sigma$ in G defines a *skeleton fragment* $g(\underline{L})$ of G that contains an *interface fragment* $g(\underline{K}) \subseteq g(\underline{K})$; also, every variable edge e in L defines a *variable fragment* $g(\sigma(e))$. In the same way, g' defines a skeleton fragment $g'(\underline{L}')$ of G with an interface fragment $g'(\underline{K}')$, and variable fragments $g'(\sigma(e'))$ for the variable edges e' in L' .

Figure 5 below illustrates how “classical” parallel independence carries over to hierarchical graph transformation. Competing direct derivations are parallelly independent if and only if the images of the rules’ left-hand side skeletons (the semicircles) overlap only in their skeleton interface fragments (the white areas of the semicircles). The deleted part of the skeleton fragments (drawn as grey semicircles) and their variable fragments (drawn as grey boxes) must be disjoint. In this situation, competing direct derivations leave the occurrence of the respective other rule intact; they commute by a direct derivation using the other rule at the unchanged occurrence.

Figure 6 shows the non-critical overlap of two direct derivations. The left-hand side of one rule must occur completely inside a single variable fragment (of x in the illustration) of the other rule. In this case, the competing direct derivations are not commutative. In general, several steps may be necessary to join them again. Let p be the rule subsuming the occurrence of p' in the variable fragment $g(\sigma(x))$. In this example x occurs twice in p ’s left hand side. A direct derivation with p leads to a hierarchical graph H wherein $g(\sigma(x))$ will occur as often as x occurs in p ’s right hand side, say i times. Then H contains $i \geq 0$ occurrences of the left hand side of p' . The occurrences of p' in G and in H are parallelly independent, and can be transformed in 2 and i steps with p' , respectively. In the resulting graphs, every variable fragment of x has been transformed in the same way, so that there is a direct derivation with p between the hierarchical graphs, which joins the derivations.

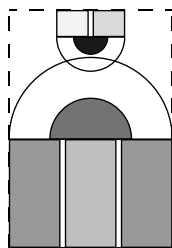


Fig. 5. “Classical” parallel independence

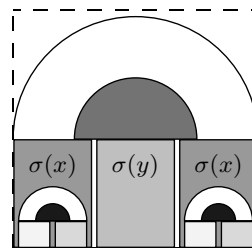


Fig. 6. Non-critical overlap

Definition 8 (“Classical” Parallel Independence). A pair of competing direct derivations is “classically” parallelly independent if the intersection of $g(L\sigma)$ and $g'(L'\sigma')$ in G is contained in the intersection $g(\underline{K}) \cap g'(\underline{K}')$ of their skeleton interface fragments.

Fact. “Classically” parallelly independent direct derivations are commutative.

Definition 9 (Non-Critical Overlap). A pair of competing direct derivations is *non-critically overlapping* if the intersection of $L\sigma$ and $L'\sigma'$ in G consists of items of a single variable fragment, that is, $g(L\sigma) \subseteq g'(\sigma'(e'))$ for some variable edge e' in L' or, vice versa, $g'(L'\sigma') \subseteq g(\sigma(e))$ for some variable edge e in L .

Theorem 2. *Noncritically overlapping derivations are joinable.*

Proof Sketch. Let $H \leftarrow_{p,\sigma,g} G \Rightarrow_{p',\sigma',g'} H'$ be non-critically overlapping. Without loss of generality, $g'(L'\sigma') \subseteq g(\sigma(e))$ for some variable edge e in L with label, say x . Assume first that g is top. By the Restriction Lemma [7], there is a restricted direct derivation $d'(e)$ of the direct derivation $d': G \Rightarrow_{p'} H'$ to the variable fragment $g(\sigma(e))$ with result, say $H'(e)$. By theorem 1, $G = L^*\sigma^*$ and $H = R^*\sigma^*$. By the Embedding Lemma [7], the direct derivation $d'(e)$ can be embedded into every variable fragment $g(\sigma(e))$ of $G = L^*\sigma^*$ with $lab_L(e') = lab_L(e)$. The embedded derivations are parallelly independent. By parallel independence [12, 7], there is a derivation $L^*\sigma^* \Rightarrow_{p'}^+ L^*\tau^*$ where τ^* is the modification of the substitution σ^* with $\tau^*(x) = H'(e)$ and $\tau^*(y) = \sigma^*(y)$ otherwise. By theorem 1, there is a direct derivation $L^*\tau^* \Rightarrow_p R^*\tau^*$. The direct derivation $d'(e)$ can be embedded into every variable fragment $g(\sigma(e))$ of $R^*\sigma^*$ with $lab_L(e') = lab_L(e)$. Again, the embedded derivations are parallelly independent. Thus, there is a derivation $R^*\sigma^* \Rightarrow_p^* R^*\tau^*$, and, the direct derivations are joinable, see below.

$$\begin{array}{ccc}
 & L^*\sigma^* & \\
 \swarrow \text{q} & & \searrow \text{q} \\
 R^*\sigma^* & & L^*\tau^* \\
 \searrow \text{q}^* & & \swarrow \text{q} \\
 & R^*\tau^* &
 \end{array}$$

Now, if g is not top, let \bar{G} be the part of G where g is top. Then there are competing derivations $\bar{H} \leftarrow \bar{G} \Rightarrow \bar{H}'$ that have joining derivation sequences $\bar{H} \xrightarrow{*} \bar{M} \xleftarrow{*} \bar{H}'$. Since derivations are closed under the part relation, graphs H , H' and M can be constructed by replacing the parts in corresponding to \bar{G}' in those graphs so that we get the diagram above.

Example 3 (Commuting and Joining Control Flow Derivations). Figure 7 shows several direct derivations of control flow graphs. The graph in the middle of the top row can be transformed in four ways: The rule *loop* applies to the loop in the else part of the top branch, and *inl* can be applied to its procedure call edges. To its left, we see the graph after applying *inl* to the procedure call on the left;

beneath it we see the result of applying *loop* to the loop in its else part; the result of applying *inl* twice, to the (isomorphic) procedure calls in that loop is shown on the right.

The *loop* step is “classically” parallelly independent of the left *inl* step, and the result of the commuting steps is shown in the lower left. Both occurrences of the *inl* steps leading to the right are contained in the fragment of the variable D of the *loop* rule; since this variable occurs twice on the left hand side, and once on the right hand side of *loop*, two steps are needed in the top row, and one in the bottom row, until they lead to the graphs on the right where the *loop* rule can be applied again.

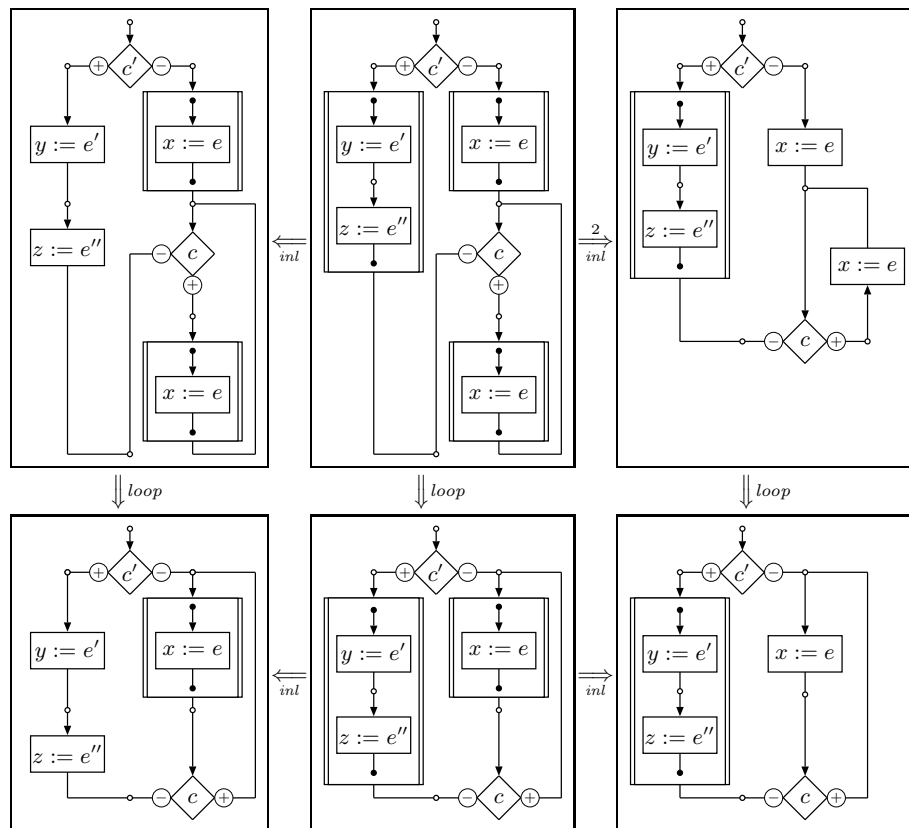


Fig. 7. Parallel independent transformations of control flow graphs

Consider the noncritical overlap illustrated in Fig. 6. For term rewriting, where trees are being transformed, the occurrence of a rule (like p') can only overlap

with a single variable fragment (of x), because the occurrence is connected, and the variable fragment is disconnected from other variable fragments. However, graphs need not be connected so that further situations arise in the case of hierarchical graph transformation, which are sketched in Fig. 8.

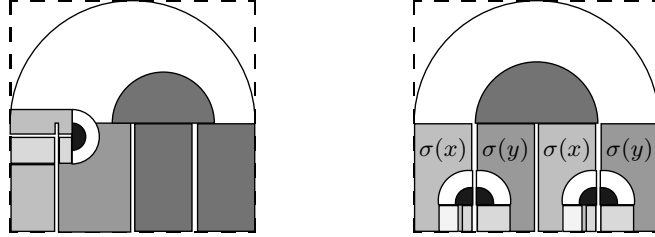


Fig. 8. Fragmented parallel independence

In the situation on the left, p' overlaps with the skeleton interface fragment, and with two variable fragments of p . The competing derivations would be joinable if the involved fragments of p are preserved in the direct derivation with p . This is the case if they are left intact, i.e., if the involved variables occur once on both sides of p because the skeleton of p is also involved. In the situation on the right, this need not be the case as the skeleton fragment is not involved in the overlap. Here, it suffices when the involved variables have the same number of occurrences on both sides of p .

Thus, whenever the intersection of $L\sigma$ and $L'\sigma'$ in G consists of several variable fragments we have to require that one occurrence induces a decomposition of the other rule into subrules such that the fragments can be transformed separately by the subrules. Furthermore, the transformation must be consistent, i.e. same fragments have to be transformed in the same way, and complete. Furthermore, the transformation must be repetitive, i.e. after the application of the rule a complete parallel transformation must be possible.

A variable edge e in L is *involved* in the direct derivation $G \Rightarrow_{p'} H'$ if the intersection of the skeleton fragment $g'(\underline{L}')$ and the variable fragment $g(\sigma(e))$ is non-trivial, i.e. if the intersection consist not only of points. The label of an involved edge is called an *involved label*.

Definition 10 (Fragmented Direct Derivations). Let $\langle d, d' \rangle$ be a pair of direct derivations. Then d' is *g-decomposable* if there is a decomposition of d' into a non-changing subderivation on the skeleton fragment and subderivations $d'(e)$ on the variable fragments for e in L^* . In this case, we speak of a *g-decomposition* of d' . A *g-decomposition* is *consistent* if $lab_L(e) = lab_L(e')$ implies $\tau(e) = \tau(e')$ for all involved edges. It is *complete* if there is no not-involved edge with involved label. It is *completable* if d' can be extended to a derivation $G \Rightarrow_{p'}^+ I'$ with complete set of involved edges. A *g-decomposable*, *consistent*, and *completable* direct derivation is called *g-compatible*. The direct derivation d is *g'-repetitive*

if there is a derivation $H \Rightarrow_{p'}^* R^* \tau^*$ of some substitution τ^* . The pair $\langle d, d' \rangle$ is *fragmented* if d' is g -compatible and d is g' -repetitive or d is g' -compatible and d' is g -repetitive.

Fact. Every g -compatible direct derivation $G \Rightarrow_{p', \sigma', g'} H'$ through a top morphism g' can be extended to a derivation $G \Rightarrow_{p'}^+ L^* \tau^*$ for some substitution τ^* .

Proof Sketch. Let $d': G \Rightarrow_{p'} H'$ be g -compatible. Then d' is g -decomposable, consistent, and completable. By g -decomposability, there is a decomposition of d' into a non-changing subderivation on the skeleton fragment and subderivations $d'(e)$ on the variable fragments for e in L^* such that H' is obtained from L^* by replacing the ordinary variables e in L^* by the result $\tau(e)$ of the subderivation $d'(e)$ and the context variable in L^* by the intermediate hierarchical graph D . By consistency, the replacements define a substitution

$$\tau^* = \{ \text{lab}_L(e) \mapsto \tau(e) \mid e \in E_L \} \cup \{ x \mapsto D \} .$$

In the case of completeness, $H' = L^* \tau^*$. In the case of completability, d' can be extended to a derivation $G \Rightarrow_{p'}^+ I'$ such that $I' = L^* \tau^*$.

Definition 11 (Fragmented Parallel Independence). A pair of direct derivations $\langle d, d' \rangle$ is *fragmentedly parallel independent* if the skeleton fragments overlap only in the skeleton interface fragments, and if $\langle d, d' \rangle$ is fragmented.

Theorem 3 (Fragmented Parallel Independence). *Every pair of fragmentedly parallel independent direct derivations is joinable.*

Proof Sketch. Let $d: G \Rightarrow_{p, \sigma, g} H$ and $d': G \Rightarrow_{p', \sigma', g'} H'$ be fragmentedly parallel independent. Without loss of generality, assume that d' is g -compatible and d is g' -repetitive. We first consider the case that g is top. By Theorem 1, $G = L^* \sigma^*$ and $H = R^* \sigma^*$. By the g -compatibility of d' , there is a derivation $L^* \sigma^* \Rightarrow_{p'}^+ L^* \tau^*$ for some substitution τ^* . By Theorem 1, there is a direct derivation $L^* \tau^* \Rightarrow_p R^* \tau^*$. By g' -repetitiveness of d , there is derivation $R^* \sigma^* \Rightarrow_{p'}^* R^* \tau^*$. Thus, the direct derivations are joinable.

$$\begin{array}{ccc}
 & L^* \sigma^* & \\
 \swarrow \scriptstyle p & & \searrow \scriptstyle p' \\
 R^* \sigma^* & & L^* \tau^* \\
 \searrow \scriptstyle p & & \swarrow \scriptstyle p \\
 & R^* \tau^* &
 \end{array}$$

The case that g is not top can be reduced to the situation above by the same argument as in the proof of Thm. 2.

Note that fragmented parallel independence subsumes both “classical” parallel independence (illustrated in Fig. 5), and non-critical overlaps (shown in Fig. 6): Only the skeleton fragment of p is involved in the first case, and a single variable fragment is involved in the second case.

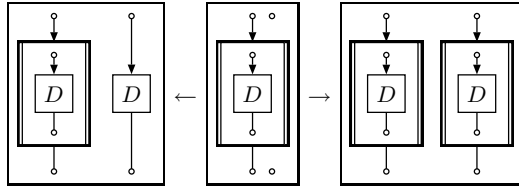


Fig. 9. The rule *fold*

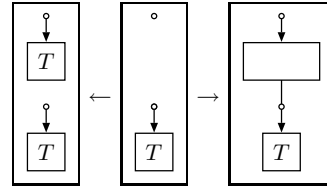


Fig. 10. The rule *join*

Example 4 (Fragmented Parallel Independence of Control Flow Graph Transformations). In Figs. 9 and 10, we define two rules that illustrate particular cases of fragmented parallel independence: If some control flow graph D matches the variable fragment of a procedure, *fold* replaces the body D by a call to that procedure, and if a control flow graph ends in two copies of the same subdiagrams T , *join* redirects one of them to the other. (The empty assignment represents a neutral computation.)

The rule *fold* is a parallel rule of the form $fold = id + inl^{-1}$ (where *id* is the identical rule) and hence decomposable into two subrules. The rule *join* is not decomposable. Figure 11 shows two fragmentedly parallelly independent direct derivations steps through the rules *fold* and *loop* that overlap in a nontrivial way: The occurrences of the left-hand sides intersect not only in the body of the loop (which is an instantiation of the variable D in *loop*), but also in the “ \ominus ”-successor state of the branch at the bottom. Nevertheless, the direct derivations are joinable, as the *fold* rule divides into two rules, and one of them is just the identity.

5 Conclusion

We have studied under which conditions direct transformations of a graph are independent so that they can be joined to a common graph by subsequent transformations. Graphs and rules have been generalized by concepts known from term rewriting: Graphs are equipped with a tree-like hierarchy, as edges may contain graphs which are again hierarchical; rules contain graph variables by which subgraphs of arbitrary size can be compared, deleted, or copied in a single transformation step. Our results combine properties known for plain graph transformation and term rewriting.

To our knowledge, parallel independence of graph transformation has only been studied for the double- and single-pushout approaches. In both cases, neither hierarchies, nor graph variables have been considered. Parallel independence has also been investigated in the more general framework of adhesive high-level replacement systems [15, 13]. It looks as if hierarchical graph transformation without variables is an instance of adhesive high-level replacement; this is not true for hierarchical graph transformation with variables, however.

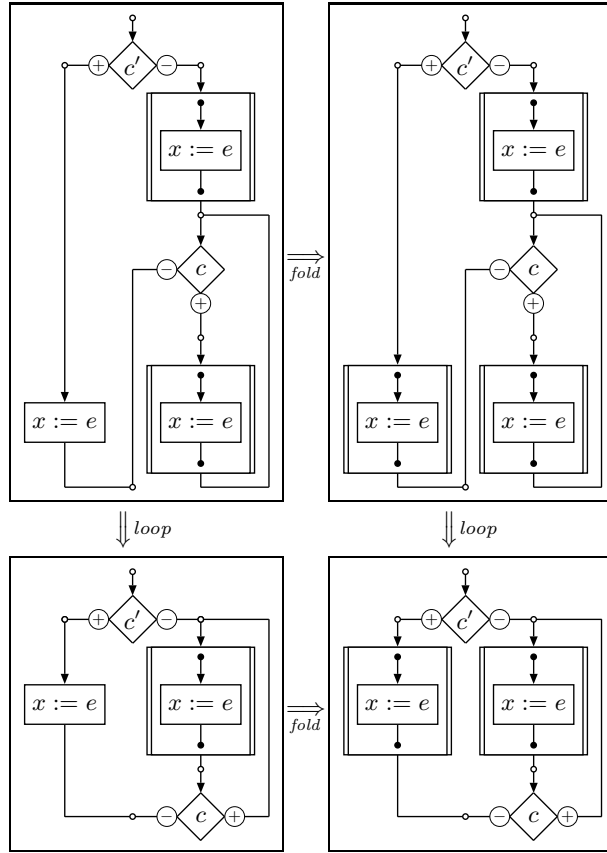


Fig. 11. Fragmented parallel independent transformations of control flow graphs

The study of parallel independence has led to critical pair lemmata, both for term rewriting and for graph transformation [16]: whenever transformation steps are not parallelly independent, these systems are locally confluent if joinability can be shown for finitely many critical pairs of graphs and terms, respectively. Since parallel independence of hierarchical graph transformation turned out to be a reasonable combination of the results for graph transformation and term rewriting, we shall try to combine these lemmata to obtain a critical pair lemma for hierarchical graph transformation as well.

Furthermore, local confluence implies general confluence if the rules are also terminating. Since termination can be characterized by the finiteness of so-called forward closures of rules, both for term rewriting and for graph transformation [17], we think it may be possible to combine these results to a similar theorem for hierarchical graph transformation. Finally, if we are able to find

decidable sufficient criteria for termination, this, together with a critical pair lemma, would allow to decide confluence in restricted cases. This would give immediate benefits for the analysis of DIAPLAN, a language for programming with graphs and diagrams that shall be based on hierarchical graph transformation [11].

References

1. Drewes, F., Hoffmann, B., Plump, D.: Hierarchical graph transformation. *Journal of Computer and System Sciences* **64** (2002) 249–283
2. Hoffmann, B.: Shapely hierarchical graph transformation. In: *Proc. IEEE Symposium on Human-Centric Computing Languages and Environments*. IEEE Computer Press (2001) 30–37
3. Rozenberg, G., ed.: *Handbook of Graph Grammars and Computing by Graph Transformation*. Volume 1: Foundations. World Scientific (1997)
4. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: *Handbook of Graph Grammars and Computing by Graph Transformation*. Volume 2: Applications, Languages and Tools. World Scientific (1999)
5. Ehrig, H., Kreowski, H.J., Montanari, U., Rozenberg, G., eds.: *Handbook of Graph Grammars and Computing by Graph Transformation*. Volume 3: Concurrency, Parallelism, and Distribution. World Scientific (1999)
6. Busatto, G.: *An Abstract Model of Hierarchical Graphs and Hierarchical Graph Transformation*. PhD thesis, Universität-Gesamthochschule Paderborn (2002)
7. Ehrig, H.: Introduction to the algebraic theory of graph grammars. In: *Graph Grammars and Their Application to Computer Science and Biology*. Volume 73 of *Lecture Notes in Computer Science.*, Springer-Verlag (1979) 1–69
8. Engels, G., Heckel, R.: Graph transformation as a conceptual and formal framework for system modelling and evolution. In: *Automata, Languages, and Programming (ICALP 2000)*. Volume 1853 of *Lecture Notes in Computer Science.*, Springer-Verlag (2000) 127–150
9. Plump, D., Habel, A.: Graph unification and matching. In: *Graph Grammars and Their Application to Computer Science*. Volume 1073 of *Lecture Notes in Computer Science.*, Springer-Verlag (1996) 75–89
10. Baader, F., Nipkow, T.: *Term Rewriting and All That*. Cambridge University Press, Cambridge, UK (1998)
11. Hoffmann, B.: Abstraction and control for shapely nested graph transformation. *Fundamenta Informaticae* **58** (1) (2003) 39–65
12. Ehrig, H., Kreowski, H.J.: Parallelism of manipulations in multidimensional information structures. In: *Mathematical Foundations of Computer Science*. Volume 45 of *Lecture Notes in Computer Science.*, Springer-Verlag (1976) 284–293
13. Ehrig, H., Habel, A., Padberg, J., Prange, U.: Adhesive high-level replacement categories and systems. In: *Graph Transformation (ICGT'04)*. *Lecture Notes in Computer Science*, Springer-Verlag (2004)
14. Habel, A., Müller, J., Plump, D.: Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science* **11** (2001) 637–688
15. Ehrig, H., Habel, A., Kreowski, H.J., Parisi-Presicce, F.: Parallelism and concurrency in high level replacement systems. *Mathematical Structures in Computer Science* **1** (1991) 361–404

16. Plump, D.: Hypergraph rewriting: Critical pairs and undecidability of confluence. In: Term Graph Rewriting: Theory and Practice. John Wiley, New York (1993) 201–213
17. Plump, D.: On termination of graph rewriting. In: Graph-Theoretic Concepts in Computer Science. Volume 1017 of Lecture Notes in Computer Science., Springer-Verlag (1995) 88–100

A Use of Skeleton Interfaces in Rule Instances

The “only-if” direction of theorem 1 requires that the interface of a rule instance is its skeleton, and not its instance. Otherwise the rule instance would be applicable to graphs where the substitutive rule does not apply. This shall be illustrated by an example.

The rule $p = \langle L \leftarrow K \rightarrow R \rangle$ shown in Fig. 12 has the interface variable A . If p would be instantiated by the substitution $\sigma = \{A \mapsto \circ \boxed{a} \bullet\}$, the (extended) rule instance $p\sigma = \langle L\sigma \leftarrow K\sigma \rightarrow R\sigma \rangle$ has a direct derivation $d: G \Rightarrow_{p\sigma} H$. However, there is no way to extend σ to a substitution σ^* for the substitutive rule $p^* = \langle L^*, R^* \rangle$ so that $L^*\sigma^* \cong G$: The context variable C cannot be instantiated by the substitution pair $D \mapsto \circ \boxed{c} \bullet$ because that graph is connected to an “inner node” of ‘ A ’s substitution. (There is a substitution σ' where $\sigma'(A)$ is a single point, and $\sigma'(C) = \sigma^*(A) \cup \sigma^*(C)$, but the instance $p^*\sigma'$ does not derive H , but a subgraph of H where the right a -edge is missing.)

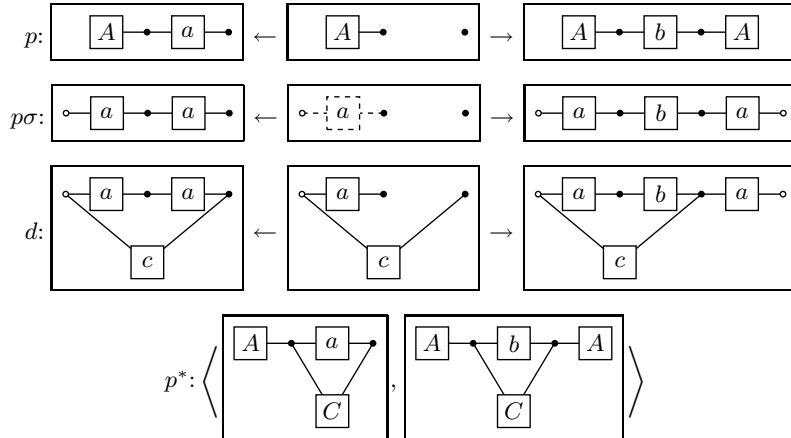


Fig. 12. Skeleton interfaces in instances