Weakest Liberal Preconditions relative to HR* Graph Conditions

Hendrik Radke
hendrik.radke@informatik.uni-oldenburg.de

Carl v. Ossietzky Universität Oldenburg, Germany*

Abstract. Graph conditions are very important for graph transformation systems and graph programs in a large variety of application areas. With HR* graph conditions, non-local graph properties like "there exists a path of arbitrary length" or "the graph is cycle-free" can be expressed. Together with graph programs, these conditions form a framework for writing programs over graphs, and specifying invariants and properties for these graphs. This paper takes a step towards automating the verification of graph programs with pre- and postconditions. Using Dijkstra's approach, the postcondition is transformed "over the program" to a weakest precondition. The correctness problem is thus reduced to the problem whether or not the precondition implies the weakest precondition, which can be tackled with a theorem prover.

1 Introduction

Formal methods, like the verification of programs with respect to formal system properties, play an important role in the development of trustworthy systems. In our approach, we use graphs to model real-world states and double-pushout graph transformation rules [EEPT06] to describe state changes. Structural properties of the system are described by graph conditions. In [HP09,Pen09], nested graph conditions have been discussed as a formalism to describe structural properties. Nested conditions are expressively equivalent to first-order graph formulas and can express local properties in the sense of Gaifman [Gai82]. In [HR10], HR^+ graph conditions are introduced which are more expressive than monadic second-order graph formulas [Cou97]. These conditions make it possible to express non-local properties like the existence of a path of arbitrary length, the connectedness or circle-freeness of a graph. In this paper, we propose HR^* conditions generalizing HR and HR^+ conditions. Our goal is to check the correctness of graph programs relative to a HR^* pre- and postcondition. Following Dijkstra's approach [Dij76], the correctness of a program relative to pre- and postconditions can be shown by constructing a weakest precondition from the program and the postconditon. A weakest precondition is constructed by first transforming the postcondition into a right HR^* application condition for the program,

* This work is supported by the German Research Foundation (DFG), grants GRK 1076/1 (Graduate School on Trustworthy Software Systems).

then a transformation from the right to a left application condition and finally, from the left application condition to the weakest precondition. This is a generalization of the transformations from [HPR06], where the variables and the corresponding replacement systems have to be regarded. This way, the correctness problem can be reduced to the problem whether the precondition implies the weakest precondition.

Example 1. A small example may illustrate how HR conditions look and how they can be used to form non-local conditions.

$$c_x = \exists(\bullet_1 \xrightarrow{+} \bullet_2), \text{ with } + ::= \bullet_1 \longrightarrow \bullet_2 \mid \bullet_1 \longrightarrow \bullet \xrightarrow{+} \bullet_2$$

The condition c_x has the meaning "There is a path of arbitrary length from the image of node 1 to the image of node 2". The paths of arbitrary length are represented by the hyperedge labeled +. Replacing the hyperedge according to the replacement rules, we gain a series of graphs without hyperedges $\bullet_1 \longrightarrow \bullet_2$, $\bullet_1 \longrightarrow \bullet \longrightarrow \bullet_2$, $\bullet_1 \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet_2$, ..., i.e. paths of arbitrary length from node 1 to node 2.

The paper is organized as follows. Section 2 introduces HR* conditions and graph transformation rules. In Section 3, basic transformations for HR* conditions are defined. These transformations are used in Section 4 to define the transformation of programs and postconditions into weakest preconditions. The paper is closed with Section 5, where the conclusion is drawn and further work is suggested.

2 HR* conditions

Graphs with variables consist of nodes, edges, and hyperedges. Edges have one source and one target and are labeled by a symbol of an alphabet; hyperedges have an arbitrary sequence of attachment nodes (indicated by *tentacles* between the hyperedge and the attachment node) and are labeled by variables.

Definition 1 (Graphs with variables). *Let* $\mathrm{C} = \langle \mathrm{C_V}, \mathrm{C_E}, \mathrm{X} \rangle$ *be a fixed, finite label alphabet where* X *is a set of variables with a mapping* $\mathrm{rank}\colon \mathrm{X} \to \mathbb{N}_0$ *defining the rank of each variable. A* graph (with variables) *over* C *is a system* $G = (\mathrm{V}_G, \mathrm{E}_G, \mathrm{Y}_G, \mathrm{s}_G, \mathrm{t}_G, \mathrm{att}_G, \mathrm{lv}_G, \mathrm{le}_G, \mathrm{ly}_G)$ *consisting of finite sets* V_G, E_G, *and* Y_G *of* nodes *(or* vertices*)*, edges*, and* hyperedges, source *and* target functions $\mathrm{s}_G, \mathrm{t}_G\colon \mathrm{E}_G \to \mathrm{V}_G$, *an* attachment function $\mathrm{att}_G\colon \mathrm{Y}_G \to \mathrm{V}_G^*$[1], *and* labeling functions $\mathrm{lv}_G\colon \mathrm{V}_G \to \mathrm{C_V}$, $\mathrm{le}_G\colon \mathrm{E}_G \to \mathrm{C_E}$, $\mathrm{ly}_G\colon \mathrm{Y}_G \to \mathrm{X}$ *such that, for all* $y \in \mathrm{Y}_G$, $|\mathrm{att}_G(y)| = \mathrm{rank}(\mathrm{ly}_G(y))$. *The set of all graphs with variables we call* $\mathcal{G}_X$, *while* $\mathcal{G}$ *denotes the set of all graphs without variables, i.e. with* $\mathrm{Y}_G = \emptyset$ *for any* $G \in \mathcal{G}$.

[1] This also includes hyperedges with zero tentacles.

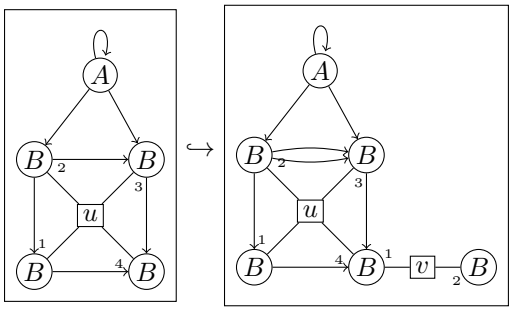
Remark 1. The definition extends the well-known definition of graphs [Ehr79] by the concept of hyperedges in the sense of [Hab92]. Graphs with variables also may be seen as special hypergraphs where the set of hyperedges is divided into a set of edges labelled with terminal symbols and a set of hyperedges labelled by nonterminal symbols.

We extend the definition of graph morphisms to the case of graphs with variables.

Definition 2 (Graph morphisms with variables). *A* (graph) morphism (with variables) $g\colon G \to H$ *consists of functions* $g_{\mathrm{V}}\colon \mathrm{V}_G \to \mathrm{V}_H$, $g_{\mathrm{E}}\colon \mathrm{E}_G \to \mathrm{E}_H$, *and an injective*[2] *function* $g_{\mathrm{Y}}\colon \mathrm{Y}_G \to \mathrm{Y}_H$ *that preserve sources, targets, attachment nodes, and labels, that is,* $\mathrm{s}_H \circ g_{\mathrm{E}} = g_{\mathrm{V}} \circ \mathrm{s}_G$, $\mathrm{t}_H \circ g_{\mathrm{E}} = g_{\mathrm{V}} \circ \mathrm{t}_G$, $\mathrm{att}_H = g_{\mathrm{V}}^* \circ \mathrm{att}_G$, $\mathrm{lv}_H \circ g_{\mathrm{V}} = \mathrm{lv}_G$, $\mathrm{le}_H \circ g_{\mathrm{E}} = \mathrm{le}_G$, *and* $\mathrm{ly}_H \circ g_{\mathrm{Y}} = \mathrm{ly}_G$. *(For a mapping* $g\colon A \to B$, *the free symbolwise extension* $g^*\colon A^* \to B^*$ *is defined by* $g^*(a_1 \dots a_k) = g(a_1) \dots g(a_k)$ *for all* $k \in \mathbb{N}$ *and* $a_i \in A$ $(i = 1, \dots, k)$.*) We call* $\mathrm{Dom}(g) = G$ *the* domain *of* g *and* $\mathrm{Ran}(g) = H$ *the* codomain *of* g. *The term* $g(G)$ *denotes the result of the application of the functions in* g *on graph* G.

A morphism g *is* injective *(*surjective*) if* g_{V}, g_{E}, *and* g_{Y} *are injective (surjective), and an* isomorphism *if it is both injective and surjective. In the latter case* G *and* H *are* isomorphic, *which is denoted by* $G \cong H$. *The* composition $h \circ g$ *of* g *with a graph morphism* $h\colon H \to M$ *consists of the composed functions* $h_{\mathrm{V}} \circ g_{\mathrm{V}}$, $h_{\mathrm{E}} \circ g_{\mathrm{E}}$, *and* $h_{\mathrm{Y}} \circ g_{\mathrm{Y}}$. *For a graph* G, *the* identity $\mathrm{id}_G\colon G \to G$ *consists of the identities* $\mathrm{id}_{G_{\mathrm{V}}}$, $\mathrm{id}_{G_{\mathrm{E}}}$, *and* $\mathrm{id}_{G_{\mathrm{Y}}}$ *on* G_{V}, G_{E}, *and* G_{Y}, *respectively.*

Example 2. Consider graphs G, H over the label alphabet $\mathrm{C} = \langle \{A, B\}, \{\square\}, \mathrm{X} \rangle$ where the symbol $\square$ stands for the invisible edge label and is not drawn and $\mathrm{X} = \{u, v\}$ is a set of variables that have rank 4 and 2, respectively. The graph G contains five nodes with the labels A and B, respectively, seven edges with label $\square$ which is not drawn, and one hyperedge of rank 4 with label u. Additionally, the graph H contains a node, an edge, and a hyperedge of rank 2 with label v.



The drawing of graphs with variables combines the drawing of graphs in [Ehr79] and the drawing of hyperedges in [Hab92,DHK97]: Nodes are drawn by

[2] Injectivity of g_Y ensures that replacement morphisms (defined later) can be composed properly, see also [HR10].

circles carrying the node label inside, edges are drawn by arrows pointing from the source to the target node and the edge label is placed next to the arrow, and hyperedges are drawn as boxes with attachment nodes where the i-th tentacle has its number i written next to it and is attached to the i^{th} attachment node and the label of the hyperedge is inscribed in the box. For visibility reasons, we sometimes write $\bullet \xrightarrow{x} \bullet$ instead of $\bullet \overset{1}{-} \boxed{x} \overset{2}{-} \bullet$. Arbitrary graph morphisms are drawn by the usual arrows "$\to$"; the use of "$\hookrightarrow$" indicates an injective graph morphism. The actual mapping of elements is conveyed by indices, if necessary.

Hyperedges do not only play a static part as building blocks of graphs with variables, but also a dynamic part as place holders for graphs. While a hyperedge is attached to a sequence of attachment nodes, a graph that should replace it must be equipped with an equally long sequence of nodes. This node sequence controls which attachment point of the hyperedge is fused with which node from the replacing graph.

Definition 3 (Pointed graphs with variables). *A* pointed graph with variables $\langle G, \mathrm{pin}_G \rangle$ *is a graph with variables G together with a sequence* $\mathrm{pin}_G = v_1 \dots v_n$ *of pairwise distinct nodes from G. We write* $\mathrm{rank}(G)$ *for the number n of nodes. For $x \in \mathrm{X}$ with* $\mathrm{rank}(x) = n$, $x^\bullet$ *denotes the pointed graph with the nodes $v_1, \dots, v_n$, one hyperedge attached to $v_1 \dots v_n$, and sequence $v_1 \dots v_n$.*

In [PH96,Pra04], variables are substituted by arbitrary graphs. In this paper, variables are replaced by graphs generated by a hyperedge replacement system.

Definition 4 (HR system). *A* hyperedge replacement (HR) system $\mathcal{R}$ *is a finite set of replacement pairs of the form x/R where x is a variable and R a pointed graph with* $\mathrm{rank}(x) = \mathrm{rank}(R)$.

Given a graph G, the application of the replacement pair x/R to a hyperedge y with label x proceeds in two steps:

1. *Remove the hyperedge y from G, yielding the graph $G - \{y\}$.*
2. *Construct the disjoint union $(G - \{y\}) + R$ and fuse the i^{th} node in* $\mathrm{att}_G(y)$ *with the i^{th} attachment point of R, for $i = 1, \dots, \mathrm{rank}(y)$, yielding the graph H.*

Then G directly derives *H by x/R applied to y, denoted by $G \Rightarrow_{x/R,y} H$ or $G \Rightarrow_{\mathcal{R}} H$ provided $x/R \in \mathcal{R}$. A sequence of direct derivations $G \Rightarrow_{\mathcal{R}} \dots \Rightarrow_{\mathcal{R}} H$ is called a* derivation *from G to H, denoted by $G \Rightarrow^*_{\mathcal{R}} H$. For every variable x, $\mathcal{R}(x) = \{G \in \mathcal{G}_X \mid x^\bullet \Rightarrow^*_{\mathcal{R}} G\}$ denotes the set of all graphs derivable from $x^\bullet$ by $\mathcal{R}$.*

Example 3. The hyperedge replacement system $\mathcal{R}$ with the rules given in Backus-Naur form $+ ::= \bullet_1 \longrightarrow \bullet_2 \mid \bullet_1 \longrightarrow \bullet \xrightarrow{+} \bullet_2$ generates the set of all directed paths from node 1 to node 2.

In the following, let $\mathcal{R}$ be a fixed HR-system.[3] Hyperedge replacement systems define replacements.

Definition 5 (Replacement). *Let $\mathcal{G}_X^\bullet$ denote the set of all pointed graphs. Let G be a graph and $\mathrm{Y} \subseteq \mathrm{Y}_G$ be a set of hyperedges to be replaced. A mapping* $\mathrm{repl}\colon \mathrm{Y} \to \mathcal{G}_X^\bullet$ *is a* base for replacement *in G if, for all $y \in \mathrm{Y}$,* $\mathrm{repl}(y) \in \mathcal{R}(\mathrm{ly}_G(y))$. $\mathrm{Dom}(\mathrm{repl}) = \mathrm{Y}$ *is the* domain *of* repl. *The replacement of* Y *in G by* repl*, denoted by* $\mathrm{repl}(G)$*, is obtained from G by simultaneously replacing all hyperedges y in Y by* $\mathrm{repl}(y)$. *We write $G \Longrightarrow^{\mathrm{repl}} H$ if $H \cong \mathrm{repl}(G)$.*

Replacement morphisms consist of a base for replacement and a injective graph morphism. For the composition of graph and replacement morphisms, see [HR10].

Definition 6 (Replacement morphisms). *A replacement morphism $\langle \mathrm{repl}, g \rangle$ consists of a base for replacement* repl *in G and an injective graph morphism $G \hookrightarrow^g H'$. It is* injective *if g is injective.*

Notation 1 *In order to discern different types of morphisms in graphs, different types of arrows are used. Replacement morphisms are symbolized by double tips (——») and replacements by double shafts ($\Longrightarrow$).*

We now define HR^* conditions, a slight extension of HR^+ conditions [HR10]. Instead of having conditions $X \sqsubseteq \boxed{\mathrm{Y}}$, HR^* conditions have "splitting" conditions of the form $\exists(\boxed{\mathrm{Y}} \stackrel{\circ}{=} X \oplus \boxed{\mathrm{Y'}}, c)$ that decompose the hyperedge Y into X and a new hyperedge Y'. This allows us to scope in on a part X of the replacement of Y, while at the same time keeping an injective representation of Y.

Definition 7 (union of graphs).

For two graphs with variables G_1, G_2, the union $G_1 \oplus G_2$ is defined as the pushout object G' constructed from $g_1\colon G_0 \to G_1$ and $g_2\colon G_0 \to G_2$, where G_0 constitutes the common parts of G_1 and G_2 (as indicated by indices in G_1 and G_2).

$$\begin{array}{ccc} G_0 & \hookrightarrow & G_1 \\ \downarrow & (PO) & \downarrow \\ G_2 & \hookrightarrow & G' \end{array}$$

Definition 8 (HR^* (graph) condition). HR^* conditions *are inductively defined as follows. For a graph with variables P,* true *is a condition over P. For every morphism $a\colon P \to C$ and every condition c over C, $\exists(a, c)$ is a condition over P. For every graph with variables C and every condition c over C, $\exists(P \stackrel{\circ}{=} C, c)$ is a condition over P, where $P = P' \oplus Q$, $C = C' \oplus Q$, P' is a graph*

[3] This is to prevent the tedious inclusion of $\mathcal{R}$ in every condition and improve readability.

induced by a set of hyperedges in P, and C' is an arbitrary graph in $\mathcal{G}$. Boolean formulas over conditions over P are conditions over P: For a condition c over P, $\neg c$ is a condition over P, and for an index set J and HR^* *conditions $(c_j)_{j\in J}$ over P, $\wedge_{j\in J} c_j$ is a* HR^* *condition over P. A* HR^* *condition is* finite *if every index set J in this* HR^* *condition is finite.*

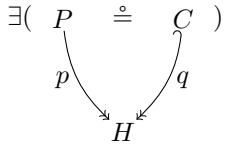
Furthermore, the following abbreviations are used: false abbreviates $\neg$true, $\exists a$ abbreviates $\exists(a, \text{true})$, $\forall(a, c)$ abbreviates $\neg\exists(a, \neg c)$, $\vee_{j\in J} c_j$ abbreviates $\neg \wedge_{j\in J} \neg c_j$, $\forall(P' \stackrel{\circ}{=} C', c)$ abbreviates $\neg\exists(P' \oplus Q \stackrel{\circ}{=} C' \oplus Q, \neg c)$ (i.e. identical parts can be omitted), and $\forall(P \stackrel{\circ}{=} C, c)$ abbreviates $\neg\exists(P \stackrel{\circ}{=} C, \neg c)$.

Remark 2. In the following, HR^* conditions are shortly called *conditions*. Conditions in the context of graphs are called *constraints*, and conditions in the context of rules are called *application conditions*.

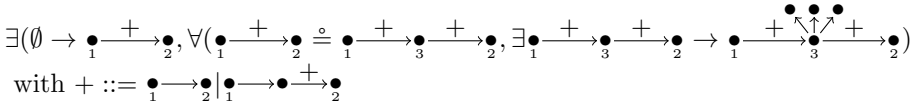
We define the satisfaction of HR^* conditions, as an extension of [Pen09].

Definition 9 (satisfaction of HR^* conditions). *A replacement morphism $p\colon P \twoheadrightarrow H$* satisfies *$\exists(a, c)$ iff there is an injective replacement morphism q such that $q \circ a = p$ and q satisfies c. A replacement morphism p* satisfies *$\exists(P \stackrel{\circ}{=} C, c)$ if there is an injective replacement morphism $q\colon C \hookrightarrow\!\!\!\!\!\rightarrow H$ such that $p(P) = q(C)$ and q satisfies c.*
The satisfaction is extended to Boolean formulas over conditions in the usual way, i.e., a replacement morphism p satisfies *true, p* satisfies *$\neg c$ iff p does not satisfy c, and p* satisfies *$\wedge_{i\in I} c_i$ iff p satisfies all c_i $(i \in I)$. A graph G* satisfies *the condition c, if c is a condition over $\emptyset$ and the morphism $\emptyset \to G$ satisfies c. We write $p \models c$ [$G \models c$] to denote that a replacement morphism p [a graph G] satisfies c.*

Remark 3. For non-injective replacement morphisms, conditions of the form $\exists(P \stackrel{\circ}{=} C, c)$ are not satisfied.



Example 4. The following example extends upon the one from the introduction. It shows a HR^* condition expressing "There is a path from the image of node 1 to the image of node 2, and all images of nodes on this path (i.e. that can be reached by splitting the path into two subpaths) have at least three outgoing edges". Note that while the nodes 1 and 2 are common to the left- and right-hand side of the splitting condition, the hyperedges on both sides are not identified.



HR* conditions include HR+ conditions, i.e. for every HR+ condition c, there is an equivalent HR* condition that satisfies the same morphisms. A condition $\overset{}{\underset{x}{\bullet}} \sqsubseteq \boxed{\text{Y}}$ is transformed into a HR* condition $\exists(\boxed{\text{Y}} \overset{\circ}{=} \overset{}{\underset{x}{\bullet}}\!\!-\!\!-\boxed{\text{Y'}}, \text{true})$, where the replacement system for Y' is the same as for Y, with the addition that node x can exactly once be identified with a derived node. For HR+ conditions with edges, $\overset{}{\underset{1}{\bullet}} \xrightarrow{\text{x}} \overset{}{\underset{2}{\bullet}} \sqsubseteq \boxed{\text{Y}}$ is transformed into $\exists(\boxed{\text{Y}} \overset{\circ}{=} \overset{}{\underset{1}{\bullet}} \overset{\text{x}}{\underset{2}{\bullet}}\!\!-\!\!-\boxed{\text{Y'}}, \text{true})$, where the two nodes 1, 2 together with edge x can exactly once be identified with two nodes and a derived edge.

The advantage of HR+ conditions over HR*conditions is that they allow expressions over parts of replacement while retaining a fully injective semantics. In contrast, the semantics for HR* conditions is based on replacement morphisms that are injective up to replacement, i.e. in the replacements for the hyperedges, nodes and edges may be joined. The author deems the fully injective semantics more comfortable and intuitive.

3 Basic transformations of HR* conditions

In this section, we define rules and generalize the basic transformations for nested graph conditions in [HPR06] to HR* graph conditions.

Definition 10 (rules). *A* plain rule $p = \langle L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R \rangle$ *is a pair of injective graph morphisms l, r with common domain K called* interface*. L is called the* left-hand side *and R the* right-hand side*. A* left (right) application condition *is a condition over L (R). A* rule $\rho = \langle p, \text{ac}_L, \text{ac}_R \rangle$ *consists of a plain rule p together with a left and a right application condition* ac_L *and* ac_R*, respectively.*

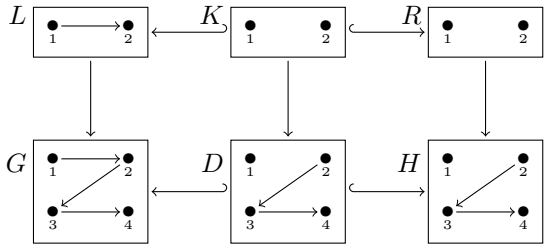
$$\begin{array}{ccccc} L & \hookleftarrow & K & \hookrightarrow & R \\ {\scriptstyle m}\downarrow & (1) & \downarrow & (2) & \downarrow{\scriptstyle m^*} \\ G & \hookleftarrow & D & \hookrightarrow & H \end{array}$$

Given a plain rule p and a morphism $m\colon L \to G$, a direct derivation *consists of two pushouts (1) and (2). We write $G \Rightarrow_{p,m,m^*} H$, $G \Rightarrow_p H$, or short $G \Rightarrow H$. The morphism m is called* match *and m^* is called* comatch*. Given a rule $\rho = \langle p, \text{ac} \rangle$ and a morphism $K \to D$, there is a direct derivation $G \Rightarrow_{p,m,m^*} H$ if $G \Rightarrow_{p,m,m^*} H$, $m \models \text{ac}_L$, and $m^* \models \text{ac}_R$.*

Pushout (1) dictates that the match must satisfy the *dangling condition*. This means that any node to be deleted (i.e. in $G - D$) must not have an edge to a node which is not deleted (i.e. a node in D). Otherwise, the pushout construction (1) would leave D with "dangling" edges which have no source or target node and D would not be a proper graph in $\mathcal{G}$.

Example 5. The (plain) rule $\rho_x = \langle \overset{}{\underset{1}{\bullet}} \longrightarrow \overset{}{\underset{2}{\bullet}} \hookleftarrow \overset{}{\underset{1}{\bullet}} \quad \overset{}{\underset{2}{\bullet}} \hookrightarrow \overset{}{\underset{1}{\bullet}} \quad \overset{}{\underset{2}{\bullet}} \rangle$ deletes an edge between two nodes 1 and 2. The nodes may be identified, so the rule could also be applied on a single node with a loop and delete the loop.

The application of this rule on a graph is shown below.



Now, we define a transformation that converts a constraint into a right application condition for a given rule.

Lemma 1 (transformation of constraints into application conditions). *For every* HR^* *constraint* c *over* P *and every graph morphism* $m\colon P \to P'$*, there is an application condition* $\mathrm{A}(m,c)$ *such that, for every replacement morphism* $p\colon P' \twoheadrightarrow P''$*,* $p \models \mathrm{A}(m,c) \iff p \circ m \models c$*.*

This transformation is used to transform a postcondition (over $\emptyset$) into a right application condition (over the right-hand side R of the rule).

Construction 1 *For a morphism* $m\colon P \to P'$*, transformation* A *is inductively defined as follows:*

$$\begin{array}{ccc} P & \xrightarrow{a} & C \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle m'} \\ P' & \xrightarrow{a'} & C' \end{array} \qquad \mathrm{A}(m, \exists(P \xrightarrow{a} C, c)) = \bigvee_{(a',m') \in \mathcal{F}} \exists\,(a', \mathrm{A}(m', c)),$$

where $\mathcal{F} = \{(a',m') \mid (1)$ *commutes,* m' *injective,* (a',m') *jointly epimorphic}*

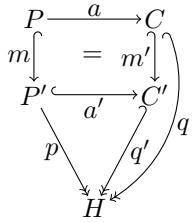
For $\mathrm{A}(m, \exists(P \stackrel{\circ}{=} C, c))$*, where* $P = Q \oplus P_1$*,* $C = Q \oplus C_1$ *let* $\mathrm{A}(m, \exists(P \stackrel{\circ}{=} C, c)) = \exists(\mathrm{Ran}(m) \stackrel{\circ}{=} m'(C), \mathrm{A}(m', c))$ *if the pushout complement* (1) *can be constructed and* (2) *is a pushout, and* false *otherwise.*

$$\begin{array}{ccccc} P & \hookleftarrow & Q & \hookrightarrow & C \\ {\scriptstyle m}\downarrow & (1) & \downarrow & (2) & \downarrow{\scriptstyle m'} \\ P' & \hookleftarrow & Q' & \hookrightarrow & C' \end{array}$$

For Boolean formulas over conditions and true*, the construction of* A *is straightforward:* $\mathrm{A}(m, \neg c) = \neg \mathrm{A}(m,c)$*,* $\mathrm{A}(m, \wedge_{j \in J} c_j) = \wedge_{j \in J} \mathrm{A}(m, c_j)$*, and* $\mathrm{A}(m, \mathrm{true}) =$ true*.*

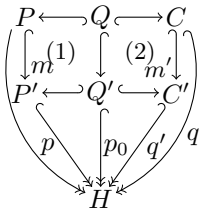
Proof. By structural induction over conditions.
Basis. For $c = \mathrm{true}$, we have $\mathrm{A}(m,c) = \mathrm{true} = c$.
Hypothesis. Assume that $p \models \mathrm{A}(m,c') \Leftrightarrow p \circ m \models c'$ holds for condition c'.
Step. We proceed by case distinction over the structure of c. For the Boolean conditions $c = \neg c', c = c' \wedge c''$, the proof is straightforward. For details, see [Pen09].

Case 1: $c = \exists(a, c')$. Assume that p satisfies $\mathrm{A}(m, c) = \bigvee_{(a',m')\in\mathcal{F}} \exists\,(a', \mathrm{A}(m', c'))$. By definition of $\models$, there is an injective morphism q' such that $p = q' \circ a'$ and $q' \models \mathrm{A}(m', c')$. Let $q = q' \circ m'$. Then we have $p \circ m = q \circ a$ and $q' \circ m' \models c'$. By the induction hypothesis, $q' \circ m' \models c' \Leftrightarrow q \models \mathrm{A}(m', c')$, completing this part of the proof.



Assume that $p \circ m \models \exists(a, c')$. By definition of $\models$, there is an injective morphism q such that $p \circ m = q \circ a$ and $q \models c'$. We construct jointly epimorphic graph morphisms $\langle m', a' \rangle$ such that $m' \circ a = a' \circ m$, and an injective replacement morphism q' with $q' \circ m' = q$. By the induction hypothesis, $q \models c' \Leftrightarrow q' \models \mathrm{A}(m', c')$.

Case 2: $c = \exists(P \stackrel{\circ}{=} C, c)$. Assume that p satisfies $\mathrm{A}(m, c) = \exists(P' \stackrel{\circ}{=} C', \mathrm{A}(m', c'))$ where (1) and (2) are pushouts. We let $q = q' \circ m'$, yielding $(p \circ m)(P) = q(C)$ and, by the induction hypothesis, $q' \models \mathrm{A}(m', c') \Leftrightarrow q' \circ m' \models c'$, implying $p \circ m \models \exists(P \stackrel{\circ}{=} C, c')$.



Assume that $p \circ m \models c$. Since m' is injective, we can construct a $q' \colon C' \to H$ such that $q = q' \circ m'$ and the diagram above commutes, yielding $p(P) = q'(C)$. By the induction hypothesis, $q' \circ m' \models c' \Leftrightarrow q' \models \mathrm{A}(m', c')$, thus $p \models \mathrm{A}(m, c')$. □

Example 6. We now show the application of A on rule ρ_x from example 5 and condition c_x from introductory example 1. The right morphism of ρ_x is $r_x = \bullet_a \ \bullet_b \hookrightarrow \bullet_a \ \bullet_b$.

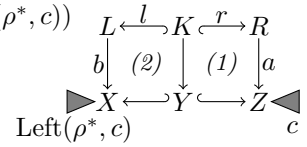
$$\mathrm{A}(r_x, \exists(\bullet_1 \xrightarrow{+} \bullet_2)) \;=\; \exists(\bullet_1 \xrightarrow{+} \bullet_2 \ \bullet_a \ \bullet_b) \vee \exists(\bullet_{1=a} \xrightarrow{+} \bullet_{2=b}) \vee \exists(\bullet_{1=b} \xrightarrow{+} \bullet_{2=a}) \vee$$
$$\exists(\bullet_b \ \bullet_{1=a} \xrightarrow{+} \bullet_2) \vee \exists(\bullet_a \ \bullet_{1=b} \xrightarrow{+} \bullet_2) \vee \exists(\bullet_1 \xrightarrow{+} \bullet_{2=a} \ \bullet_b) \vee \exists(\bullet_1 \xrightarrow{+} \bullet_{2=b} \ \bullet_a)$$

In a next step, we transform the right application condition of a rule ρ into a left one. Basically, this encompasses the reverse application of ρ to the condition itself.

Lemma 2 (from right to left application conditions). *There is a transformation* Left *such that, for every* HR^* *application condition* ac *of a rule* $\rho = \langle L \hookleftarrow K \hookrightarrow R \rangle$ *and for all direct derivations* $G \underset{\rho,m,m^*}{\Longrightarrow} H$, $m \models \mathrm{Left}(\rho, \mathrm{ac}) \Leftrightarrow m^* \models \mathrm{ac}$.

Construction 2 *The transformation* Left *applies the reverse of the rule to each morphism and object in the condition, yielding* false *whenever the dangling condition is not met.*

For a condition $\exists(a,c)$, $\mathrm{Left}(\rho,\exists(a,c)) = \exists(b,\mathrm{Left}(\rho^*,c))$ *if* $\langle r,a\rangle$ *has a pushout complement (1) and* $\rho^* = \langle X \hookleftarrow Y \hookrightarrow Z\rangle$ *is the rule derived by constructing pushout (2). Otherwise,* $\mathrm{Left}(\rho,\exists(a,\mathrm{ac})) = \mathrm{false}$.



For a condition $\exists(R \triangleq C_R,c)$, $\mathrm{Left}(\rho,\exists(R \triangleq C_R,c)) = \exists(L \triangleq C_L,\mathrm{Left}(\rho^*,c))$, *where* $R = R' \oplus P'$, $K = K' \oplus P'$, $L = L' \oplus P'$, *and* $\rho^* = \langle L' \oplus C' \hookleftarrow K' \oplus C' \hookrightarrow R' \oplus C'\rangle$ *is constructed from* ρ *by replacing* P' *with* C'. *Since* P' *contains only hyperedges,* P' *is unchanged by the rule.*

For Boolean formulas over conditions, the construction is straightforward: $\mathrm{Left}(\rho,\mathrm{true}) = \mathrm{true}$, $\mathrm{Left}(\rho,\neg c) = \neg\,\mathrm{Left}(\rho,c)$ *and* $\mathrm{Left}(\rho,\bigwedge_{j\in J} c_j) = \bigwedge_{j\in J}\mathrm{Left}(\rho,c_j)$.

Proof. By induction over the structure of ac.
Basis. For $\mathrm{ac} = \mathrm{true}$, we have $m \models \mathrm{Left}(\rho,\mathrm{true}) = \mathrm{true} \Leftrightarrow \mathrm{true} \Leftrightarrow m^* \models \mathrm{true}$.
Hypothesis. Assume that $m \models \mathrm{Left}(\rho,c) \Leftrightarrow m^* \models c$ holds for application condition c.
Step. We proceed by case distinction over the structure of ac. For Boolean formulas over conditions and $\exists(a,c)$, the proof is only sketched; for details, refer to [HPR06].

Case 1: $\mathrm{ac} = \neg c$. Then $m \models \mathrm{Left}(\rho,\neg c) \Leftrightarrow m \models \neg\,\mathrm{Left}(\rho,c) \Leftrightarrow \neg m \models \mathrm{Left}(\rho,c) \Leftrightarrow \neg m^* \models c \Leftrightarrow m^* \models \neg c$.

Case 2: $\mathrm{ac} = c \wedge c'$. Then $m \models \mathrm{Left}(\rho,c) \wedge c' \Leftrightarrow m \models \mathrm{Left}(\rho,c) \wedge m \models \mathrm{Left}(\rho,c') \Leftrightarrow m^* \models c \wedge m^* \models c' \Leftrightarrow m^* \models c \wedge c'$.

Case 3: $\mathrm{ac} = \exists(a,c)$. Assume that (m,a) has a pushout complement. Then $m \models \mathrm{Left}(\rho,\mathrm{ac}) \Leftrightarrow m \models \exists(b,\mathrm{Left}(\rho^*,c))$. For the derivation, we can decompose the pushouts of the derivation $G \underset{\rho,m,m^*}{\Longrightarrow} H$ such that $m^* = q \circ a$ and $m = q' \circ b$. By the hypothesis, $q' \models \mathrm{Left}(\rho^*,c) \Leftrightarrow q \models c$, thus $m^* \models \exists(a,c)$. If (m,a) has no pushout complement, $m \models \mathrm{Left}(m,\mathrm{ac}) \Leftrightarrow m \models \mathrm{false}$ and there is no pushout such that $m^* \models c$, i.e. $m^* \models \mathrm{false}$. Case 4: $\mathrm{ac} = \exists(R \oplus P' \triangleq R \oplus C',c)$. Assume that $m \models \mathrm{Left}(\rho,\exists(R' \oplus P' \triangleq R' \oplus C',c))$. Then $m \models \exists(L' \oplus P' \triangleq L' \oplus C',\mathrm{Left}(\rho^*,c)) \Leftrightarrow \exists q.m(L' \oplus P') = q(L' \oplus C')$ and $q \models \mathrm{Left}(\rho^*,c)$. By the induction hypothesis, $q \models \mathrm{Left}(\rho^*,c) \Leftrightarrow q^* \models c$, thus $m \models \exists(L \oplus P' \triangleq L \oplus C',\mathrm{Left}(\rho^*,c))$. □

Example 7. We use example 6 to demonstrate the workings of transformation Left.

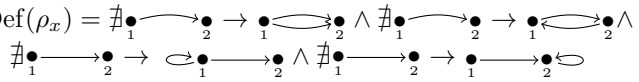
$$\mathrm{Left}(A(r_x,c_x)) = \exists(\bullet_1 \xrightarrow{+} \bullet_2 \quad \bullet_a \to \bullet_b) \vee \exists(\bullet_{1=a} \overset{+}{\rightrightarrows} \bullet_{2=b}) \vee \exists(\bullet_{1=b} \overset{+}{\rightrightarrows} \bullet_{2=a}) \vee$$
$$\exists(\bullet_b \leftarrow \bullet_{1=a} \xrightarrow{+} \bullet_2) \vee \exists(\bullet_a \to \bullet_{1=b} \xrightarrow{+} \bullet_2) \vee \exists(\bullet_1 \xrightarrow{+} \bullet_{2=a} \to \bullet_b) \vee \exists(\bullet_1 \xrightarrow{+} \bullet_{2=b} \leftarrow \bullet_a)$$

Furthermore, we need a transformation that expresses the applicability of a rule.

Lemma 3 (applicability of a rule [HP09]). *There is a transformation* Def *from rules into application conditions such that, for every rule* ρ *and every morphism* $m\colon L \to G$, $m \models \mathrm{Def}(\rho) \Leftrightarrow \exists H.G \underset{\rho,m,m^*}{\Longrightarrow}{}^* H$.

Construction 3 *For a plain rule* $p = \langle L \hookleftarrow K \hookrightarrow R \rangle$*, let* $\mathrm{Def}(p) = \wedge_{a \in A} \nexists a$*, where* A *is the set of all graph morphisms* $a\colon L \to L'$*, where* L' *is obtained from* L *by adding an edge with both end points in* L*, and for which* $\langle l, a \rangle$ *has no pushout complement. For a rule* $\rho = \langle p, \mathrm{ac} \rangle$ *with application condition, let* $\mathrm{Def}(\rho) = \mathrm{Def}(p) \wedge \mathrm{ac}_L \wedge \mathrm{Left}(\rho, \mathrm{ac}_R)$*.*

Example 8. For our example rule ρ_x, we have

$\mathrm{Def}(\rho_x) =$ 

Now, a transformation C is defined that transforms left application conditions into constraints over $\emptyset$. As in [HPR06], this is done by ensuring that the left application condition is valid for every morphism $\emptyset \to L$.

Lemma 4 (From application conditions to constraints). *For every application condition* ac *over* L*, there is a condition* $\mathrm{C}(\mathrm{ac})$ *such that for every graph* G*,* $G \models \mathrm{C}(\mathrm{ac}) \Leftrightarrow \forall m\colon L \to G.m \models \mathrm{ac}$*.*

Construction 4 *Let* $\mathcal{E}(P)$ *denote the set of all epimorphisms with domain* P*. Define* $\mathrm{C}(\mathrm{ac}) := \wedge_{e \in \mathcal{E}(L)} \forall(e \circ i, \mathrm{C}_e(\mathrm{ac}))$*, where* $i : \emptyset \to L$ *is the unique morphism from* $\emptyset$ *to* L*.* $\mathrm{C}_e(\mathrm{ac})$ *is defined over the structure of conditions as* $\mathrm{C}_e(\exists(a, \mathrm{ac})) = \exists(a', \mathrm{ac})$ *if there is a factorization* $a = a' \circ e$*, where* a' *is an injective morphism and* e *an epimorphism, and* false *otherwise, and* $\mathrm{C}_e(\exists(P \stackrel{\circ}{=} C, c)) = \exists(P \stackrel{\circ}{=} C, c)$*. For Boolean conditions,* C_e *is defined the usual way.*

Proof. By induction over the structure of ac. Let $\mathrm{C}_i(\mathrm{ac}) = \wedge_{e \in \mathcal{E}(L)} \forall(e \circ i, \mathrm{ac})$.
Basis. For $\mathrm{ac} = \mathrm{true}$, we have $\mathrm{C}(a, \mathrm{true}) = \mathrm{C}_i(\mathrm{true}) = \mathrm{true} = \mathrm{ac}$.
Hypothesis. Assume that $G \models \mathrm{C}(\mathrm{ac}) \Leftrightarrow \forall m\colon L \to G.m \models \mathrm{ac}$ holds for application condition c.
Step. We proceed by case distinction over the structure of ac.

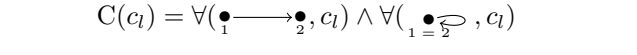
Case 1: $\mathrm{ac} = \neg c$. $G \models \mathrm{C}(\rho, \mathrm{ac}) \Leftrightarrow G \models \mathrm{C}_i(\neg \mathrm{C}_e(c)) = \neg \mathrm{C}_i(\mathrm{C}_e(c)) \Leftrightarrow m \models \mathrm{ac}$.

Case 2: $\mathrm{ac} = c \wedge c'$. Then $G \models \mathrm{C}(c \wedge c') \Leftrightarrow G \models \mathrm{C}_i(\mathrm{C}_e(c) \wedge \mathrm{C}_e(c'))$
$\Leftrightarrow m \models c \wedge c'$.

Case 3: $\mathrm{ac} = \exists(a, c)$. Then $G \models \mathrm{C}(\mathrm{ac}) \Leftrightarrow G \models \mathrm{C}_i(\mathrm{C}_e(\mathrm{ac}))$
$\Leftrightarrow \mathrm{C}_i(\exists a'.a = a' \circ e \wedge G \models \exists(a', c)) \vee \mathrm{false})$
$\Leftrightarrow \mathrm{C}_i(\exists a'.a = a' \circ e \wedge m \models c \vee \mathrm{false}) \Leftrightarrow m \models \mathrm{ac}$.

Case 4: $\mathrm{ac} = \exists(P \stackrel{\circ}{=} C, c)$. $G \models \mathrm{C}(\mathrm{ac}) \Leftrightarrow G \models \mathrm{C}_i(\mathrm{C}_e(\mathrm{ac}))$
$\Leftrightarrow \mathrm{C}_i(G \models \mathrm{ac}) \Leftrightarrow \mathrm{C}_i(m \models c) \Leftrightarrow m \models \mathrm{ac}$. □

Example 9. We continue with the term from example 7 and apply transformation C to it. Let $c_l = \mathrm{Left}(A(\rho_x, c_x))$.

$\mathrm{C}(c_l) =$ 

4 Weakest liberal preconditions of HR* conditions

We now use the transformations defined in the last section to transform conditions over rules and programs. Furthermore, we show that the transformed conditions are weakest liberal preconditions.

Graph programs are defined as in [HP01,HP09].

Definition 11 (Graph program). *Graph programs are inductively defined as follows:*

1. *Every rule is a program.*
2. *Every finite set $\mathcal{S}$ of programs is a program.*
3. *Given programs P and Q, sequential composition $(P;Q)$ and iteration $P\downarrow$ are programs.*

We can now define the semantics of graph programs.

Definition 12 (Semantics of graph programs). *The semantics of a program is a binary relation $[\![P]\!] \subseteq \mathcal{G} \times \mathcal{G}$. For every rule ρ, every set $\mathcal{S}$ of programs, and every pair of programs P and Q, $[\![p]\!] = \{\langle G, H\rangle \,|\, G \Rightarrow_p H\}, [\![\mathcal{S}]\!] = \bigcup_{P\in\mathcal{S}} [\![P]\!]$ for finite sets $\mathcal{S}$ of programs, $[\![(P;Q)]\!] = [\![Q]\!] \circ [\![P]\!], [\![P\downarrow]\!] = \{\langle G, H\rangle \,|\, \langle G, H\rangle \in [\![P]\!]^* \wedge \nexists M. \langle H, M\rangle \in [\![P]\!]\}$, where $[\![P]\!]^*$ is the reflexive-transitive closure of $[\![P]\!]$.*

Definition 13 (liberal precondition). *For a program P and a condition d, a condition c is a* liberal precondition relative to *d if for all graphs G satisfying c, $\langle G, H\rangle \in [\![P]\!]$ implies $H \models d$ for all H. A liberal precondition c is a* weakest liberal precondition *of P relative to d, denoted by* $\mathrm{wlp}(P, d)$*, if any precondition of P relative to d implies c. A (weakest) liberal precondition is a* (weakest) precondition *if $G \Rightarrow_\rho H$ for some H and P terminates for G.*

Combining the basic transformations from Section 3, we can now define a transformation of a program and a postcondition into a weakest (liberal) precondition similar to [HPR06].

Theorem 1 (weakest liberal precondition). *There is a transformation* Wlp *such that for every program P and for every condition* post*, $\mathrm{Wlp}(P, \mathrm{post})$ is a weakest liberal precondition of P and* post*.*

Construction 5 *For any postcondition d, any rule ρ, any set $\mathcal{S}$ of programs, and any programs P, Q, let*

$$\begin{aligned}
\mathrm{Wlp}(\rho, d) &= C(\mathrm{Def}(p) \Rightarrow \mathrm{Left}(p, A(r, d))) \\
\mathrm{Wlp}(\mathcal{S}, d) &= \bigwedge_{P\in\mathcal{S}} \mathrm{Wlp}(P, d) \\
\mathrm{Wlp}((P;Q), d) &= \mathrm{Wlp}(P, \mathrm{Wlp}(Q, d)) \\
\mathrm{Wlp}(P\downarrow, d) &= \mathrm{Wlp}(d \vee \bigwedge_{i=1}^{\infty} \mathrm{Wlp}(P^i, d), \mathrm{Wlp}(P, \mathrm{false}) \Rightarrow d)
\end{aligned}$$

where P^i is defined inductively as $P^1 = P$ and $P^{i+1} = (P^i; P)$.

Proof. For rules ρ, we show that $\mathrm{Wlp}(P, d)$ is a weakest liberal precondition. For all objects G, we have

$$
\begin{aligned}
& G \models Wlp(p,d) \Leftrightarrow G \models^m \mathrm{C}(\mathrm{Def}(\rho) \Rightarrow \mathrm{Left}(\rho, \mathrm{A}(\rho, d))) && \text{(Def. Wlp)} \\
& \Leftrightarrow \forall L \to G.m \models^m \mathrm{Def}(\rho) \Rightarrow \mathrm{Left}(\rho, \mathrm{A}(\rho, d)) && \text{(Def. C)} \\
& \Leftrightarrow \forall L \to G.m \models^m \mathrm{Def}(\rho) \Rightarrow m \models^{m^*} \mathrm{Left}(\rho, \mathrm{A}(\rho, d)) && \text{(Def. } \models\text{)} \\
& \Leftrightarrow \forall L \to G, R \to H.m \models^m \mathrm{Def}(\rho) \Rightarrow m^* \models^{m^*} \mathrm{A}(\rho, d) && \text{(Def. Left)} \\
& \Leftrightarrow \forall L \to G, R \to H.(G \Rightarrow_{\rho,m,m^*} H) \Rightarrow H \models d && \text{(Def. A, Def)} \\
& \Leftrightarrow \forall H.G, H \in \rho \Rightarrow H \models d && \text{(Application of } \rho\text{)} \\
& \Leftrightarrow G \text{ is a weakest liberal precondition.}
\end{aligned}
$$

For composed graph programs, see the proof in [HPR06].

Similar to [HPR06], a weakest liberal precondition can be transformed into a weakest precondition. It remains to be shown that the additional requirements for weakest preconditions can be met.

5 Conclusion

We have presented a transformation of HR^* postconditions over graph programs to weakest liberal HR^* preconditions. These conditions are an extension of HR and HR^+ conditions of [HR10] which allows the partitioning of graphs. We have shown that a HR^* constraint can be transformed into a right application condition, that a right application condition can be transformed into a left, and that left application conditions, including the implicit dangling condition, can be transformed into a precondition in the form of a HR^* constraint. These transformations are used to construct a weakest liberal precondition for a program respective to a postcondition. Thus the problem whether a program is weakly correct relative to its pre- and postcondition is reduced to the problem whether the weakest liberal precondition implies the precondition.

As further work, the result for weakest liberal preconditions shall be generalized for weakest preconditions. Furthermore, a theorem prover similar to ProCon [Pen09] shall be developed that can check whether a HR^* condition implies another HR^* condition.

References

[Cou97] Bruno Courcelle. On the expression of graph properties in some fragments of monadic second-order logic. In *Descriptive Complexity and Finite Models: Proceedings of a DIMACS Workshop, Chapter 2*, 1997.

[DHK97] Frank Drewes, Annegret Habel, and Hans-Jörg Kreowski. Hyperedge replacement graph grammars. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1, pages 95–162. World Scientific, 1997.

[Dij76] Edsger Wybe Dijkstra. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, NJ, 1976.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs of Theoretical Computer Science. Springer, Berlin, 2006.

[Ehr79] Hartmut Ehrig. Introduction to the algebraic theory of graph grammars. In *Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *LNCS*, pages 1–69. Springer, 1979.

[Gai82] H. Gaifman. On local and non-local properties. In J. Stern, editor, *Proceedings of the Herbrand symposium: Logic Colloquium'81*, pages 105–135. North Holland Pub. Co., 1982.

[Hab92] Annegret Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *LNCS*. Springer, Berlin, 1992.

[HP01] Annegret Habel and Detlef Plump. Computational completeness of programming languages based on graph transformation. In *Proc. Foundations of Software Science and Computation Structures (FOSSACS 2001)*, volume 2030 of *LNCS*, pages 230–245. Springer, 2001.

[HP09] Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, pages 1–52, 2009.

[HPR06] Annegret Habel, Karl-Heinz Pennemann, and Arend Rensink. Weakest preconditions for high-level programs. In *Graph Transformations (ICGT 2006)*, volume 4178 of *Lecture Notes in Computer Science*, pages 445–460. Springer, 2006.

[HR10] Annegret Habel and Hendrik Radke. Expressiveness of graph conditions with variables. In *Int. Colloquium on Graph and Model Transformation on the occasion of the 65th birthday of Hartmut Ehrig*, volume 30 of *Electronic Communications of the EASST*, 2010. to appear.

[Pen09] Karl-Heinz Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, Universität Oldenburg, 2009.

[PH96] Detlef Plump and Annegret Habel. Graph unification and matching. In *Graph Grammars and Their Application to Computer Science*, volume 1073 of *LNCS*, pages 75–89. Springer, 1996.

[Pra04] Ulrike Prange. Graphs with variables as an adhesive HLR category. Private communication, 2004.