Electronic Notes in Theoretical Computer Science 44 No. 4 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume44.html 12 pages

On Modifying High Level Replacement Systems ⋆

Francesco Parisi-Presicce

Dip. Scienze dell'Informazione, Univ. di Roma 'La Sapienza', Rome (ITALY)

Abstract

High Level Replacement Systems generalize the concept of graph transformation systems from graphs to other types of structures. Transformation rules to replace a structure with another structure in the same category are defined using morphisms in that category. As with other specification formalisms, replacement systems are subject to modifications during their development process. These modifications could be *local* changes (individual rules are modified), *modular* changes (two replacement systems are combined to form a larger system) and *global* changes (the replacement of a replacement subsystem with another one). These different ways are analyzed and some results are presented to express the behavior of the modified system in terms of the behavior of the original one(s). Applications range from graph transformations to diagrams in visual languages and modelling to algebraic high level nets.

Key words: rule-base specifications, algebraic formalisms, evolution.

1 Introduction

High Level Replacement Systems (HLRS) [4] have been devised as a generalization of graph grammars to structures other than graphs. Besides the original motivation of extending to graphs the notion of Chomsky grammar for strings, graph transformations have been used to model the evolution of systems where the state of the system could be described by a graph. The basic idea of the algebraic approach [1] is to use the categorical construction of pushout of graph morphisms to replace, within a graph, a subgraph with another graph. The idea of replacing a substructure with another structure of the same kind has been adopted in a rule-based approach to modular system design, in which a rule represents the visible interface of a module and a direct

⋆ Partially supported by the European Community under TMR GETGRATS and Esprit WG APPLIGRAPH

derivation the ability of the module to implement the resulting specification using the original one. This, and other attempts at avoiding the duplication of results (and proofs !) already known for graph transformations [2] led to the idea of abstracting away from the particular structures involved [4] and formulate a notion of rule and of transformation in a categorical setting, with objects and morphisms in an appropriate category in place of graphs or specifications and their morphisms. The result is one theory, axiomatized over suitable categories, with several results [4] applicable in a variety of different contexts such as software systems, Petri nets, testing, visual languages and diagrams [3]. In all cases, the underlying theory of high level replacement systems provides a conceptual tool for the specification of systems.

As with any other specification formalism, high level replacement systems must evolve during the process of developing, in a stepwise manner, a high level replacement system. Three kinds of modifications are investigated:

- *modular changes:* two distinct replacement systems are combined to form a larger system of which the original ones are subsystems
- *global changes:* a high level replacement subsystem is substituted with another one using a transformation rule itself.
- *local changes:* individual rules are modified by applying other rules to them

The rest of the paper consists of a section (2) reviewing the main aspects of high level replacement systems, a section (3) dealing with the category of high level replacement systems and its closure under pushouts to model modular changes, a section (4) on rules whose components are high level replacement systems themselves, and a section (5) where rules are applied in different ways to other rules.

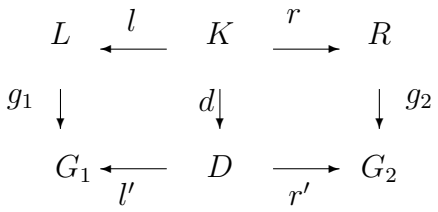
2 High Level Replacement Systems

High Level Replacement Systems (HLRS) have been formalized for an arbitrary category $\mathbf{C}$ in which a distinguished class $\mathbf{M}$of morphisms is used in the construction of the rules. In the category of graphs, the morphisms in $\mathbf{M}$are the injective graph morphisms, in the category of algebraic specifications they are injective strict specification morphisms. We review the basic notions of HLR systems, referring to [3] for details.

Definition 2.1 HLR framework
An HLR–framework *is a pair* $(\mathcal{C}, \mathbf{M})$*, where* $\mathcal{C}$ *is a category and* $\mathbf{M}$ *is a class of morphisms in* $\mathcal{C}$*.*

Definition 2.2 HLR rule and transformation
A rule over $\mathbf{M}$ *is a pair* $(l : K \to L; r : K \to R)$ *of morphisms in* $\mathbf{M}$ *with the same domain. We let Rule*$(\mathbf{M})$ *denote the class of all rules over* $\mathbf{M}$*.*

Given a rule $p = (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ *over* $\mathbf{M}$*, a* direct transformation *from* G_1 *to* G_2 *via p, denoted by* $p : G_1 \Rightarrow G_2$ *, consists of the double pushout*



Given a rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ over $\mathbf{M}$ and morphism $g_1 : L \to G_1$, the rule is applicable to G_1 if and only if
- there exists an object D (pushout complement) and relative morphisms such that the left square in the diagram above is a pushout, and
- there exists an object G_2 and relative morphisms such that the right square in the diagram above is a pushout.
Depending on the specific high level replacement framework, concrete conditions on p and g_1 can be given to guarantee the applicability of a rule [1].

The following is a variation of the definition in [4,3]. Here we add names for the rules and remove the start object.

Definition 2.3 HLR systems
A high level replacement system *is a 4-tuple* $H = (\mathcal{C}, \mathbf{M}, P, \pi)$ *where* $(\mathcal{C}, \mathbf{M})$ *is a high level replacement framework,* P *is a set (of names) and* $\pi : P \to Rule(\mathbf{M})$ *a function that associates each name to a rule over* $\mathbf{M}$.

We can think of the framework $(\mathcal{C}, \mathbf{M})$ as the "type" of H and we say that H is a High Level Replacement System over the framework $(\mathcal{C}, \mathbf{M})$.

As shown in [4,3], particular properties of an HLR framework guarantee other properties of any HLR system over that framework.

Definition 2.4 HLR conditions
In any HLR framework $(\mathcal{C}, \mathbf{M})$*, the following are called* HLR conditions

(i) *Existence of semi-*$\mathbf{M}$*-pushouts*
(ii) *Existence of* $\mathbf{M}$*-pullback*
(iii) *Inheritance of* $\mathbf{M}$*under pushouts and under pullbacks*
(iv) *Existence of binary coproducts and compatibility with* $\mathbf{M}$
(v) $\mathbf{M}$*–pushouts are pullbacks*
(vi) $\mathbf{M}$*–pushout-pullback decomposition*

Any HLR system that satisfies the HLR conditions satisfies:
Local Confluency I If $p_1 : G \Rightarrow H_1$ and $p_2 : G \Rightarrow H_2$ are parallel independent, then there exists an object X such that $p_2 : H_1 \Rightarrow X$ and $p_1 : H_2 \Rightarrow X$; furthermore $p_1 : G \Rightarrow H_1$ and $p_2 : H_1 \Rightarrow X$ are sequentially independent
Local Confluency II If $p_1 : G \Rightarrow H_1$ and $p_2 : H_1 \Rightarrow X$ are sequentially independent, then there exists an object H_2 such that $p_2 : G \Rightarrow H_2$ and $p_1 : H_2 \Rightarrow$; furthermore $p_1 : G \Rightarrow H_1$ and $p_2 : G \Rightarrow H_2$ are parallel independent
Parallelism If $p_1 : G \Rightarrow H_1$ and $p_2 : H_1 \Rightarrow X$ are sequentially independent,

then $p_1 + p_2 : G \Rightarrow X$ via the parallel rule $p_1 + p_2$

We close this section with two examples of HLR frameworks.

Example 2.5 *The first example is the framework of undirected graphs, where each edge is associated with a set of 1 or 2 nodes, its endpoints. An* **undirected graph** G *is a triple* (G_E, G_N, end)*, where* G_E *and* G_N*, are the set of edges and the set of nodes, respectively, and* $end : G_E \to \mathbf{P_2}(G_N)$ *is the function associating each edge* $\mathbf{e}$ *to a subset* $\text{END}(\mathbf{e})$ *of* G_N *of cardinality 1 or 2.*

Given undirected graphs $G = (G_E, G_N, end)$*, and* $G' = (G'_E, G'_N, end')$*, a* **U-graph morphism** $f : G \to G'$ *is a pair* $(f_E : G_N \to G'_N, f_E : G_E \to G'_E)$ *such that* $end'(f_E(e)) = f_N(end(e))$ *(where the same notation is used to denote the obvious extension of* f_N *to subsets of* N*). It is immediate to check that* **U-graphs** *and* **U-graph morphisms** *form a category closed under pushouts.*

Undirected graphs can easily be labelled with distinct sets of labels for nodes and edges, and can be typed with an undirected graph: these extensions are orthogonal to the properties presented here.

For undirected graphs, it is easy to extend the original Gluing Conditions [1] to guarantee the applicability of rules. Given $p = (L \leftarrow K \rightarrow R)$ *and* $g1 : L \to G_1$*, let*
$ID_{g1} = \{x \in L : \exists y \in L, x \neq y, g1(x) = g1(y)\}$
$DANG_{g1} = \{n \in L_N : \exists e \in G_{1E} - g1_E(L_E)$ *such that* $g1_N(n) \in end_{G_1}(e)\}$
Then the pushout complement D *exists* **if and only if**
1) $DANG_{g1} \subseteq l(K)$
2) $ID_{g1} \subseteq l(K)$

*With arguments similar to those used for directed graphs [**?**], it can be shown that with* $\mathbf{M}$*the class of* $U - morphisms$ *where both* f_N *and* f_E *are injective,* $(\mathbf{UGraph}, \mathbf{M})$ *is an HLR framework satisfying the HLR conditions.*

Example 2.6 *The second example is a recently [5] revised notion of Algebraic High Level net. An* algebraic high level net *is given by* $N = (X, A, SPEC, P, T, pre, post, cond)$ *with*

- X *: a set of variables,*
- $A \in |\mathbf{Alg}(\mathbf{SPEC})|$ *a* $SPEC$ *algebra,*
- $SPEC \in |\mathbf{SPEC}|$ *: an algebraic specification with* $SPEC = (\Sigma, E)$ *,*
- P *: a set of places,*
- T *: a set of transitions,*
- $pre, post : T \to (T_{OP}(X) \times P)^{\oplus}$ *(defining for each transition with adjacent arcs the arc inscriptions and the weight),*
- $cond : T \to \mathcal{P}_{fin}(EQNS(\Sigma))$ *(mapping each transition to a finite set of conditions representing the firing conditions).*

Morphisms in this revised version of algebraic high level nets allow the

explicit substitution of terms for variables. Given two algebraic high level nets $N_i = (X_i, A_i, SPEC_i, P_i, T_i, pre_i, post_i, cond_i)$ *for* $i = 1, 2$*, an* algebraic high level net morphism $f : N_1 \to N_2$ *is given by* $f = (f_X, f_{SPEC}, f_A, f_P, f_T)$ *with*

- $f_X : X_1 \to T_{OP_2}(X_2)$ *maps variables to terms,*
- $f_{SPEC} : SPEC_1 \to SPEC_2$ *is a specification morphism,*
- $f_A : A_1 \to V_{f_\Sigma}(A_2)$ *is a homomorphism in* $\mathbf{Alg}(\mathbf{Spec_1})$*,*
- $f_P : P_1 \to P_2$ *maps places to places in* **Set***, and*
- $f_T : T_1 \to T_2$ *maps transitions to transitions in* **Set***,*

such that the following diagram commutes componentwise

$$\begin{array}{ccccc} \mathcal{P}_{fin}(EQNS(SIG_1)) & \xleftarrow{cond_1} & T_1 & \underset{post_1}{\overset{pre_1}{\rightrightarrows}} & (T_{OP_1}(X_1) \times P_1)^{\oplus} \\ \downarrow{\scriptstyle \mathcal{P}_{fin}(f_X^\sharp)} & (1) & \downarrow{\scriptstyle f_T} & (2) & \downarrow{\scriptstyle (f_{SPEC}^\sharp \times f_P)^{\oplus}} \\ \mathcal{P}_{fin}(EQNS(SIG_2)) & \xleftarrow{cond_2} & T_2 & \underset{post_2}{\overset{pre_2}{\rightrightarrows}} & (T_{OP_2}(X_2) \times P_2)^{\oplus} \end{array}$$

A strict algebraic high level net morphism *is an algebraic high level net morphism which additionally has the following properties:*

- $f_X : X_1 \to X_2$ *is an injective mapping of variables,*
- $f_{SPEC} : SPEC_1 \to SPEC_2$ *is a strict specification morphism,*
- $f_A : A_1 \to V_{f_\Sigma}(A_2)$ *is an isomorphism in* $\mathbf{Alg}(\mathbf{Spec_1})$*,*
- f_P *and* f_T *are injective.*

As shown in [5], the category **AHL** *, of algebraic high level nets and morphisms, together with strict morphisms forms an HLR framework that satisfies the HLR conditions.*

3 Modular Transformations

A modular transformation on a HLR system consists of adding to it in an appropriate manner another HLR system, after describing what parts of the original system should be shared and not be duplicated. This combination of distinct HLR systems with a common part is obtained via the "usual" pushout construction. First we need to relate the "types" of different HLR systems via an HLR framework morphism, called **M**–compatible functor in [5].

Definition 3.1 Framework
Given HLR frameworks $(CAT_i, \mathbf{M}_i)$*, a framework morphism* $F : (CAT_1, \mathbf{M}_1) \to (CAT_2, \mathbf{M}_2)$ *is a functor* $F : CAT_1 \to CAT_2$ *such that* $F(\mathbf{M}_1) \subseteq \mathbf{M}_2$*.*

Morphisms of HLR systems associate rules to rules that are 'equal' up to retyping.

Definition 3.2 HLR morphism
An HLR morphism $h : H_1 \to H_2$ *is a pair* (h_F, h_f) *where* $h_F : (CAT_1, \mathbf{M}_1) \to (CAT_2, \mathbf{M}_2)$ *is a framework morphism and* $h_f : P_1 \to P_2$ *a function such that*

if $p \in P_1$ and $\pi_1(p) = (l, r)$ is a rule in H_1, then $\pi_2(h_f(p)) = (h_F(l), h_F(r))$ is a rule in H_2.

Being an HLR morphism does not guarantee that the derivations can also be transformed. The following result is essentially the Preservation of Derivations by Functors Theorem in [5].

Theorem 3.3 Derivation through funtors
If $h_F : CAT_1 \to CAT_2$ preserves pushouts, then $h : H_1 \to H_2$ preserve derivations.

Examples of functors with this property are the Causality and Weight functors between Elementary Nets and Place/Transition Nets, and the Data and Skeleton functors between Place/Transition Nets and Algebraic High Level Nets in [5].

HLR morphisms can be composed componentwise and the identity functor/function is obviously an HLR morphism. Furthermore,

Theorem 3.4 Category of HLR Systems
HLR systems and HLR morphisms form a category closed under pushouts.

The result of transforming an HLR system H_1 by adding to it another HLR system H_2 after specifying the shared part H_0 via HLR morphisms $h_1 : H_0 \to H_1$ and $h_2 : H_0 \to H_2$ is the pushout object of h_1 and h_2 in this category.

4 Global Transformations

Graph Transformation Systems have been devised to transform graphs by replacing a subgraph with another graph, that is, by replacing a "substructure" (determined by a morphism in the appropriate category) with another substructure. The same principle can be used to allow the replacement via a rule of a sub-graph grammar within a 'larger' graph grammar [8] or the replacement of an HLR system. The situation is comparable to that of a (string) grammar for a programming language: sets of rules describe how to generate expressions, declarations, instructions, etc., and wanting to change the concrete syntax for expressions, the corresponding subgrammar must be replaced with one reflecting the wanted new syntax.

To remain within the same DPO framework, a rule for the replacement of HLR systems consists of a span $(l : K \to L; r : K \to R)$ of HLR morphisms sharing the same HLR system as domain. Which HLR morphisms should be used ? In general there are no restrictions, but to obtain an HLR framework and therefore guarantee that the corresponding HLR system enjoys the properties mentioned earlier, the morphisms must be faithful.

Definition 4.1 Faithful
A High Level Replacement System morphism $(h_F, h_f) : HLR_1 \to HLR_2$ is faithful *if $h_f : P_1 \to P_2$ is injective and $h_F : CAT_1 \to CAT_2$ is injective on both objects and morphisms.*

This limitation is sufficient for our purpose.

Theorem 4.2 HLR framework
The category of HLR systems with distinguished morphisms **M** *the faithful morphisms of the previous definition forms a HLR framework.*

As in the case of graphs, it is possible to describe concretely the necessary and sufficient conditions for an HLR rule to be applicable to an HLR system $G = (CAT_G, M_G, P_G, \pi_G)$.

Notation For a given category CAT, $Obj(CAT)$ and $Mor(CAT)$ denote the objects and the morphisms of CAT, respectively.

Theorem 4.3 HLR Gluing Conditions
Given an HLR rule $p = (L \leftarrow K \rightarrow R)$ *and an HLR morphism* $(h_F, h_f) : (CAT_L, P_L) \rightarrow (CAT_G, P_G)$, *let*
$ID_f = \{x \in P_L : \exists y \in P_L, x \neq y, h_f(x) = h_f(y)\}$
$ID_{oF} = \{x \in Obj(CAT_L) : \exists y \in Obj(CAT_L), x \neq y, h_F(x) = h_F(y)\}$
$ID_{mF} = \{m \in Mor(CAT_L) : \exists n \in Mor(CAT_L), m \neq n, h_F(m) = h_F(n)\}$
$DANG_F = \{x \in Obj(CAT_L) : \exists m \in Mor(CAT_G) - h_F(Mor(CAT_L))$ *and* $\exists y \in Obj(CAT_G)$ *such that* $m \in Mor(h_F(x), y)$ *or* $m \in Mor(y, h_F(x))\}$

Then the pushout complement D *exists* **if and only if**
1) $DANG_F \subseteq l_F(Obj(CAT_K))$
2) $ID_{oF} \subseteq l_F(Obj(CAT_K))$
3) $ID_{mF} \subseteq l_F(Mor(CAT_K))$
4) $ID_f \subseteq l_f(P_K)$

5 Local Transformations

Besides simply replacing a rule ([8]), it is convenient to be able to define new rules by reusing existing rules. There are (at least) three possible ways of reusing a rule

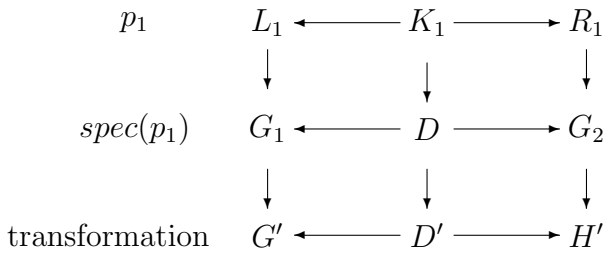
- reusing by specialization of a rule, that consists of adding properties to the rule and hence to the host object to which it can be applied (this corresponds to subtyping in which a method is refined to take advantage of the added properties of the objects of the subtype; the less specialized method can be applied whenever the specialized method can, but not viceversa)
- reusing by analogy ([9]) where a new rule is needed that "behaves like the old one" but in a different context which is 'similar' to the original (the analogy or similarity is described explicitly by the user)
- reusing by inheritance where a class with its methods is located in the object class hierarchy and is adapted by adding or extending methods to provide the desired new behavior

The approach adopted in this paper is to model all three cases of reusing in a uniform way within the same framework of the rules themselves, without

resorting to a metamodel. The idea is to use rules to modify other rules, similar to the way rules modify objects. The way the "modificator" rule is applied and the part of the rule-to-be-modified to which it is applied determines the different forms of reusing. The behavior of the modified rule can be described in terms of the behavior of the original rule and the modifying rule.

5.1 Specialization

The simplest form of modification of a rule p_1 is already present, implicitly, in the basic notion of application of a rule. When a rule p_1 is applied to an object G_1 via a matching morphism g_1, it produces not only the resulting object G_2 (see the definition of direct derivation) but also the context object D and the morphisms $D \to G_1$ and $D \to G_2$. If the morphisms of the original rule p_1 are in $\mathbf{M}$, then, by the 3rd in the list of HLR conditions, the induced morphisms $D \to G_1$ and $D \to G_2$ are also in $\mathbf{M}$. Hence $spec(p_1) = (G_1 \leftarrow D \to G_2)$ is a new rule obtained from p_1 by "specializing" the context of application via the morphism $K \to D$. Since the middle object (D in this case) represents the part of the trasformed object to be left unchanged, the new rule behaves like the old one (by removing and adding the same elements) but in a different (larger) context.



If the new rule $spec(p_1)$ is applicable to G', then there exists a context object D' so that G' is the gluing of G_1 and D' along D. By standard properties of pushouts, G' is also the gluing of L_1 and D' along K_1 and the application of p_1 to G' with matching $L_1 \to G_1 \to G'$ produces the same object H'.

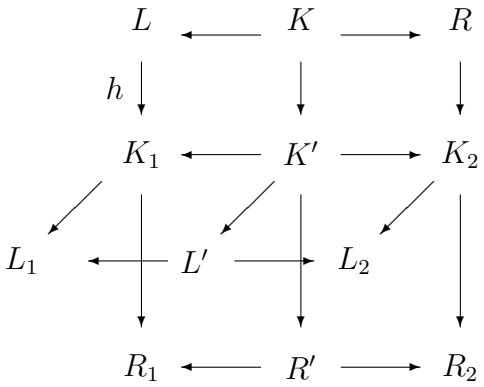
Theorem 5.1 specialization
If $(spec(p_1), m') : G' \Rightarrow H'$ *then* $(p_1, m \circ m') : G' \Rightarrow H'$.

The converse is not true since there is no guarantee that an arbitrary matching morphism $L_1 \to G'$ can be decomposed into a composition of morphisms $L_1 \to G_1 \to G'$.

5.2 Analogy

Another form of modification of a rule is obtained by analyzing the notion of analogy ([9]). While in specialization the objective is to obtain a new rule which behaves like the original one but with a 'larger' context (denoted by the morphism from the old context K to the enlarged context D), and therefore in

a smaller number of circumstances, with analogy the objective is to transform a rule indicating explicitly the similarity between the old and the new context of application. Using as example the CityTraffic model in [9], we want to express the fact that *Cars move on Roads* **like** *Trains move on Tracks*. It is the end-user responsibility to explicitly indicate that cars correspond to trains and that a road segment corresponds to a track segment. This explicit correspondence can be modelled by a rule $q = (L \leftarrow K \rightarrow R)$ which replaces an occurrence of a train and a track (represented in the left hand side L of the rule) with an occurrence of a car and a road (represented in the right hand side R of the same rule); the interface graph K is used to maintain the information about the direction of the segment (horizontal, for example) or that the moving object is on the appropriate segment. An existing "move" rule $p_1 = (L_1 \leftarrow K_1 \rightarrow R_1)$ can be modified to adapt it to the analogous situation by applying to it the "analogy" rule q. Since a move rule does not modify the segment of the move or the nature of the moving object, the information about the train and the track is represented in the part K_1 left unchanged by the rule p_1 and therefore the analogy rule q is applied to K_1 via the matching $h : L \rightarrow K_1$ as in the following diagram



The new rule $p_2 = (L_2 \leftarrow K_2 \rightarrow R_2)$ is obtained by applying q to K_1 via the matching morphism $h : L \rightarrow K_1$, to L_1 via the matching morphism $L \rightarrow K_1 \rightarrow L_1$ and to R_1 via the matching morphism $L \rightarrow K_1 \rightarrow R_1$. The applicability of q to p_1 depends on the existence of the pushout complements K', L' and R'.

For the extendibility of the derivation $q : K_1 \Rightarrow K_2$ to L_1 and R_1, there are results on the embedding of derivation sequence [1] in larger graphs. In this case of graphs, we can summarize the conditions in the following theorem.

Theorem 5.2 Application conditions
A production $q = (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ *is applicable to* $p_1 = (L_1 \stackrel{l_1}{\leftarrow} K_1 \stackrel{r_1}{\rightarrow} R_1)$ *via* $h : L \rightarrow K_1$ *if and only if*

(i) $DANG_h \cup ID_h \subseteq l(K)$

(ii) $DANG_{l_1} \cap h(L) \subseteq h(l(K))$

(iii) $DANG_{r_1} \cap h(L) \subseteq h(l(K))$

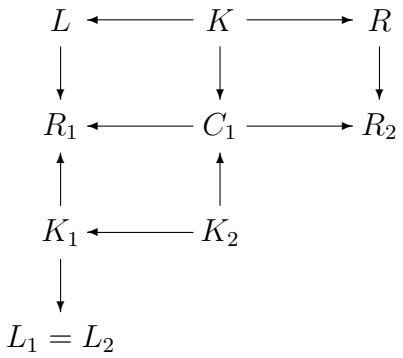
The rewriting of a production p_1 to a production p_2 via another production q is reflected in the correspondence between the objects produced by p_1 and those produced by p_2.

Theorem 5.3
If $q : p_1 \Rightarrow p_2$ *via* $h : L \to K_1$,
$p_1 : G_1 \Rightarrow H_1$ *via* $g_1 : L_1 \to G_1$,
$q : G_1 \Rightarrow G_2$ *via* $f = g_1 \circ l_1 \circ h : L \to G_1$ *and*
$p_2 : G_2 \Rightarrow H_2$ *via* $g_2 : L_2 \to G_2$
then $q : H_1 \Rightarrow H_2$.

Another way of expressing the behavior of the new rule in terms of the old one and the analogy rule is as follows: p_2 produces an object H_2 from an object G_2 if and only is G_2 is transformed by the "inverse" analogy rule q^{-1} into an object G_1 to which p_1 is applicable to produce H_1 which in turn is transformed into H_2 by the analogy rule q. Formally if p_2 is the result of applying q to p_1 as shown above, then $p_2 = q^{-1} *_L p_1 *_L q$, where $*$ is the operator of sequential composition of two rules with L determining the overlap of the right hand side of the first rule with the left hand side of the second rule [1,**?**]

5.3 Inheritance

A more general form of modification of a rule is attained via inheritance, with the underlying idea of reusing existing methods (represented in our framework by rules) by extending them to provide the desired behavior. The focus here is on a modification of a rule and not on a plain overriding. In modifying a given rule $p_1 = (L_1 \leftarrow K_1 \rightarrow R_1)$, we may want to reuse its behavior in terms of applicability of the rule and of items removed from the graph to which it is applied, while changing the items to be added: in other words, we want a new rule which coincides with p_1 in the left hand side L_1, but which has a different R_1. A modification of R_1 induces (in general) a modification of the interface part K_1. The modification of the right hand side R_1 is attained via a new rule $q = (L \leftarrow K \rightarrow R)$ applied to R_1. The result of the application is illustrated in the following diagram



where:

- R_2 is the object resulting from the direct derivation via q of R_1
- K_2 is the common part of C_1 (left unchanged by the application of q to R_1) and K_1 (left unchanged by any application of p_1); notice that it exists by the second HLR condition
- L_2 is just the unchanged left hand side of p_1

It is possible to express the new rule p_2 in terms of the old rule p_1 and the "adjustment" q. If p_2 is the rule obtained from the application of q to R_1 of p_1 as described above, then $p_2 = p_1 *_L q$. In other words, the effect of applying p_2 to an object G is the same obtained by first applying the old rule p_1 and then the modifying rule q. So the new rule p_2 "reuses" the old rule p_1 by first performing the elimination determined by p_1 and then performing the new additions.
In a similar way, a rule q can be used to modify the applicability of a rule p_1 and the items that it removes, while leaving unchanged the items added by p_1: this can be accomplished by applying q to L_1, in which case if p_2 is the rule obtained from the application of q to the left hand side L_1 of p_1, then $p_2 = q^{-1} *_L p_1$

Note that we have not discussed explicitly the applicability of a modifying rule q to p_1 since it is nothing more then the applicability of a rule to an object (R_1 in the first case, L_1 in the second case).

6 Concluding Remarks

High Level Replacement Systems describe rule based mechanisms for the substitution of a substructure by another, by using exclusively the relationship (morphism) between two objects. The lack of specific requirements on the structure of the objects that are replaced makes the theory easily usable in a variety of applications.

The specification in the different concrete cases usually evolve, either because of a better understanding of the requirements or because of changing requirements, and so it is useful to have general results at the level of HLR systems to describe the different kinds of modifications. We have modelled the integration of HLR systems by pushouts, global substitution of subsystems by rules, and "in the small" changes by individual rules.

Within the same formal framework used for rules, we have shown how to model specialization, analogy and inheritance by simply applying appropriate rules to pre–existing rules. The behavior of the "new" rules can be described in terms of the behavior of the "old" rules and the rules describing the modification. The desired adapted system can be obtained by adding to the set of pre–existing rules the "modifying" rules and by imposing a control on the order of application (described, for example, by rule expressions [6]). Local modifications can easily be viewed as global ones (where the modified rules replace the old ones) while the converse is not always true. The replacement

of a rule with its refinement (in the graph case see [6]) can be viewed as a special case of a global transformation.

A similar treatment of modifications of High Level Replacement Systems can be carried out in the Single Pushout approach.

References

[1] H.Ehrig, Introduction to the Algebraic Theory of Graph Grammars. in Lect. Notes Comp. Sci. 73, (Springer–Verlag 1979) 1–69.

[2] A.Corradini, U.Montanari, F.Rossi, H.Ehrig, R.Heckel, M.Löwe, Algebraic Approaches to Graph Grammars Part I: basic concepts and Double Pushout Approach. in G.Rozenberg ed.: *Handbook of Graph Grammars and Computing by Graph Transformations* (World Scientific ,1997) 163-246

[3] H. Ehrig, M. Gajewsky, and F. Parisi-Presicce, High-Level Replacement Systems with Applications to Algebraic Specifications and Petri Nets, in: *Handbook of Graph Grammars and Computing by Graph Transformations* volume 3: Concurrency, Parallelism, and Distribution, (World Scientific, 1999) 341–400.

[4] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce, Parallelism and concurrency in high level replacement systems. *Mathematical Structures in Computer Science*, **1** (1991) 361–404.

[5] M.Gajewsky, F.Parisi-Presicce, Formal Transformations of Petri Nets. *Techn.Rep 2000-12*, (Technische Universitat Berlin 2000)

[6] M. Große-Rhode, F. Parisi-Presicce, and M. Simeoni, Refinements of Graph Trasformation Systems via rule expressions. in: H.Ehrig, G.Engels, H.-J.Kreowski, G.Rozenberg, eds., *Theory and Applications of Graph Transformations (selected papers)* Lect. Notes Comp. Sci. 1764, (Springer–Verlag 2000) 368–382.

[7] R. Helm, and K. Marriot, Declarative Specification of Visual Languages. in *Proc. IEEE Workshop on Visual Languages*, (IEEE Comp.Sci. Press 199), 98–103.

[8] F. Parisi-Presicce, Trasformations of Graph Grammars, in *Proc. 5th Internat. Wksp. on Graph Grammars* , Lect. Notes Comp. Sci. 1073, (Springer–Verlag 1996) 426–442.

[9] C. Perrone, and A. Repenning, Graphical Rewrite Rule Analogies: avoiding the inherit or copy&paste reuse dilemma, in *Proc. IEEE Symposium on Visual Languages*, (IEEE Comp.Sci. Press 1998) 40–46.