Attributed Graph Transformations with Controlled Application of Rules*

Reiko Heckel, Jürgen Müller, Gabriele Taentzer, Annika Wagner

Technical University of Berlin
Computer Science Department
Sekr. FR 6 - 1, Franklinstr. 28/29, D-10587 Berlin
e-mail: {heckel,jolli,gabi,aw}@cs.tu-berlin.de

ABSTRACT

We present a combination of recent extensions to single pushout graph transformations, as there are attribution, application conditions and amalgamated graph transformations and add a simple transaction concept on top of this formalism. Thereby, we provide the formal basis for several examples, where these concepts are used in combination.

1 Introduction

Graph transformations [Ehr79], [Löw93] are a good means for describing the development of structured states in elementary steps. However, for the specification of more complex operations we have to provide some mechanism to control the transformation process. Several suggestion concerning different levels of control have been made up to now as there are for example

- conditional rules
- amalgamated graph transformations
- transactions.

On the level of direct transformations, applicability of rules can be controlled by application conditions. Synchronisation of parallel transformations is achieved using common subrules and amalgamation and programmed graph transformations are used to control order and frequency of rule applications.

On the other hand, most of the problems in computer science do not have only structural but also arithmetic or, more generally, data type aspects. The right means to cover this kind of applications are attributed graphs and their transformation.

Recently, an (informal) combination of these concepts has been used for specifying the AGG graph editor [CDG+94], [BT93], [Tae92], feature recognition in the area of CIM [Hof93] and other examples. Both attribution and amalgamation are used to describe aspects of the operational semantics of Troll light, which is an object-based specification language, in [WG94].

The aim of this paper is to combine all these different extensions of pure the single pushout approach to graph transformations into a single formalism that is intended to be sufficiently powerful for the specification of software systems as, for example, the AGG system.

We start in section 2 with the transformation of attributed graphs as it was introduced in [LKW93]. Then attributed graph transformations are combined with the concept of application conditions [HHT94]. Section 4 is concerned with amalgamated transformation as developed in [Tae94], [BT93], but using attributed graphs and conditional rules. Finally, section 5 provides a simple transaction concept and defines its semantics in terms of basic transactions introduced in [Sch94a]. All extension steps are illustrated by a small example of constructing Petri nets and firing transitions.

The reader is assumed to be familiar with basic notions of category theory (see for instance [HS73]) and algebraic specifications ([EM85]).

2 Attributed Graph Transformations

This section introduces the concept of attribution. Graph rewriting reaches a greater flexibility if standard operations (for example arithmetic operations for integer or real numbers) are directly applicable and need not artificially be coded into graphical structure. As running example throughout the paper we use Petri nets. Their structure is normally given by a graphical representation. But their firing mechanism is somehow based on arithmetic operations since the tokens on the places can be counted. So the idea is to assign as an attribute to each place the number of tokens laying on the place and to each pre- resp. postcondition it's weight for firing. The transitions are typed with the number of their pre- and postconditions. Figure 1 shows a sample petri net with two transitions and four places.

* This work has been partly supported by the ESPRIT Working Group 7183 "Computing by Graph Transformation (COMPUGRAPH II)"

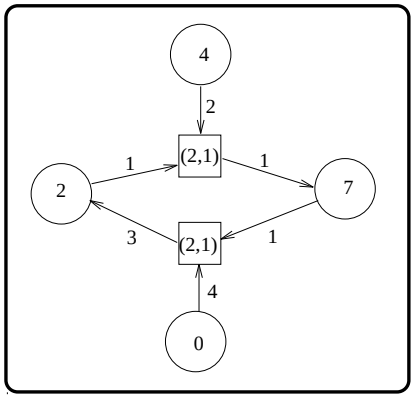

Fig. 1. Sample Petri-net

In the following we formally introduce the single pushout approach to the transformation of attributed graphs.

Definition1 (Signature). A *signature* $SIG = (S, OP)$ consists of a set of sorts S and a family of sets $OP = (OP_{w,s})_{w \in S^*, s \in S}$ of operation symbols. For $op \in OP_{w,s}$ we also write $op : w \rightarrow s$. A signature is called *graph structure signature* if it contains unary operator symbols only. An attributed graph signature consists of three parts: a graph structure signature $GS = (S1, OP1)$, an arbitrary signature $SIG = (S2, OP2)$ and a family of operator symbols called the *SIG-attribution* $ATTROP = (ATTROP_s : s \rightarrow s2_s)_{s \in S}$ where $S \subseteq S1$ and $s2_s \in S2$.

Note that the attributed graph signature $ATTR = GS + SIG + ATTROP$ is a signature. It is not really a restriction that graphical objects may only have none or one attribute. More than one attribute for the same object can be expressed using tuples.

An *algebra* A w.r.t. a signature $SIG = (S, OP)$ is an S-indexed family $(A_s)_{s \in S}$ of carrier sets together with an OP-indexed family of mappings $(op^A)_{op \in OP}$ such that $op^A : A_{s_1} \times ... \times A_{s_n} \rightarrow A_s$ if $op \in OP_{s_1 ... s_n, s}$. A special example for a SIG-algebra is the *term algebra* $T_{SIG}(X)$ *over a sort-indexed family of sets of variables* X. For this algebra the carrier sets are the sets of all terms over the variables. The operations are assigning to some argument terms the term built of the operation symbol and the argument terms.

A *morphism* between two SIG-algebras is a sort-indexed family of total mappings which is compatible with the operations. All SIG-algebras and the morphisms between them build a category, denoted $\underline{Alg(SIG)}$ in the following.

Definition2 (Attributed graph). An *attributed graph* G is an algebra with respect to an attributed graph signature. We denote the graphical part of G by G_{GS} and the attribute part by G_{SIG}, which is formally a restriction to a subsignature i.e. we use the object part of the forgetful functor associated with the signature inclusion $GS \subseteq GS + SIG + ATTROP$ resp. $SIG \subseteq GS + SIG + ATTROP$. A *morphism* between two attributed graphs $f : A \rightarrow B$ is a partial GS-morphism $f1 : A_{GS} \rightarrow B_{GS}$ together with a total SIG-morphism $f2 : A_{SIG} \rightarrow B_{SIG}$ satisfying for all operators $(attr : s1 \rightarrow s2) \in ATTROP$ and all $x \in dom(f1)_{s1}$ [1]: $f2(attr^A(x)) = attr^B(f1(x))$. We call a morphism *a- (g-) injective, surjective, isomorphic, identical or total* if it is at least injective, surjective, isomorphic, identical or total on the attribute (graphical) part. $f : A \rightarrow B$ is called *a-quasi-identical* if $A_{SIG} = T_{SIG}(X)$ and $B_{SIG} = T_{SIG}(Y)$ where $\forall x \in X : f_{SIG}(x) = y$ with $y \in Y$ s.th. $f_{SIG}(x) = f_{SIG}(x') \Longrightarrow x = x'$.

If we define composition of these morphisms by composition of the components and identities as pairs of component identities, the objects and morphisms with respect to definition 2 form a category, denoted by $\underline{AGRAPH}_{ATTR}$ in the following. If we restrict the morphisms to total ones the corresponding category is denoted by $\underline{AGRAPH}^T_{ATTR}$.

Example1 (Petri net). Each Petri net can be described as an algebra w.r.t. the following signatures:

$$
\begin{aligned}
GS = \underline{\text{Sorts}}\ & P, T, PRE, POST, Pattr, Tattr \\
\underline{\text{Opns}}\ & s_{PRE} : PRE \rightarrow P \\
& t_{PRE} : PRE \rightarrow T \\
& s_{POST} : POST \rightarrow T \\
& t_{POST} : POST \rightarrow P \\
& pattr : Pattr \rightarrow P \\
& tattr : Tattr \rightarrow T
\end{aligned}
$$

$SIG = NAT$ is the wellknown signature of natural numbers including the sort of pairs of natural numbers.

$$
\begin{aligned}
ATTROP = & attr_{token} : Pattr \rightarrow Nat \\
& attr_{weight} : PRE \rightarrow Nat \\
& attr_{weight} : POST \rightarrow Nat \\
& attr_{arity} : Tattr \rightarrow pair(Nat)
\end{aligned}
$$

Note that pre- and postconditions are attributed directly. Since attributions must be preserved by morphisms, they cannot be changed during the transformation process. In contrast the number of tokens laying on a place may change. Expressing this change means to delete the attribution to the old value and to insert an attribution to the new one. Hence attributions are objects as vertices and edges (see $Tattr$ for example).

[1] $dom(f1)$ denotes the definedness area of $f1$ which is a subalgebra of A.

For our notion of transformation based on pushouts which are special colimites it is essential to know that $\underline{AGRAPH}_{ATTR}$ is finally cocomplete. We prove this by showing that the category has an initial object and pushouts.

Proposition3 (Initial object in $\underline{AGRAPH}_{ATTR}$). *For each attributed graph signature $ATTR$ the category $\underline{AGRAPH}_{ATTR}$ has an initial object, which is the initial object in the category $\underline{Alg(ATTR)}$ of algebras and total morphisms with respect to the signature $ATTR$.*

Proof. Since $\underline{Alg(ATTR)}$ has initial objects and all objects in $\underline{AGRAPH}_{ATTR}$ are also objects in $\underline{Alg(ATTR)}$ it remains to show that for each attributed graph G the induced morphism from the initial object I in $\underline{Alg(ATTR)}$ is also a morphism in $\underline{AGRAPH}_{ATTR}$. But this is trivial due to the fact that total morphisms are special partial ones and that morphisms in $\underline{Alg(ATTR)}$ must be homomorphic for all operator symbols i.e. especially for all SIG-attributions. The uniqueness follows from the fact that every morphism from I to another attributed graph is total and from the universal property of initial objects in $\underline{Alg(ATTR)}$.

Proposition4 (Pushouts in $\underline{AGRAPH}_{ATTR}$). *Each pair $f : A \to B$, $g : A \to C$ of morphisms in $\underline{AGRAPH}_{ATTR}$ has a pushout $(D, f^* : C \to D, g^* : B \to D)$.*

For the proof of this proposition we refer to [LKW93].

Corollary5 (Colimits in $\underline{AGRAPH}_{ATTR}$). *The category $\underline{AGRAPH}_{ATTR}$ is finitely cocomplete.*

The proof follows directly from proposition 3 and 4. We do not want to construct these colimits here in general. But by defining (simple) graph transformations to be special colimits, namely pushouts, we will give a set-theoretic characterization of the resulting graph of a transformation later on.

Definition6 (Transformation). A *simple rule* $r : L \to R$ is a morphism in $\underline{AGRAPH}_{ATTR}$ whose SIG-component is an isomorphism. r is called *assignment controlled* if r_{SIG} is the identity morphism on the term algebra $T_{SIG}(X)$ over variables. A *match* $m : L \to G$ of r in an attributed graph G is a morphism whose GS-component is total. (r, m) is called *application pair*. The transformation of G with the rule r at a match m is the pushout of r and m in $\underline{AGRAPH}_{ATTR}$ written $G \overset{r,m}{\Longrightarrow} H$.

Construction 1 (Result of transformation) *Let $r : L \to R$ be a simple rule and $m : L \to G$ be a match for r in a graph G. Then the resulting attributed graph H of the transformation of G with the rule r at the match m can be constructed in three steps:*

1. *Construct the pushout of r_{GS} and m_{GS} in the category of graphs and partial morphisms $\underline{GS}$ by*
   (a) *Construction of the gluing object: Let dom be the greatest subalgebra of $dom(r_{GS})$ which is closed under the congruence induced by r_{GS} and m_{GS}.*
   (b) *Construction of the definedness area of $r^*_{GS} : G_{GS} \to H_{GS}$ and $m^*_{GS} : R_{GS} \to H_{GS}$: Let $dom(r^*_{GS})$ be the greatest subalgebra of G_{GS} whose carriers are contained in $G_{GS} - m_{GS}(L_{GS} - dom)$. Symmetrically, $dom(m^*_{GS})$ is the greatest subalgebra in $R_{GS} - r_{GS}(L_{GS} - dom)$.*
   (c) *Gluing: Let $H_{GS} = (dom(r^*_{GS}) \uplus dom(m^*_{GS})_{/\equiv}$, where $\equiv$ is the least equivalence relation containing $\sim$, with $x \sim y$ if there is $z \in dom$ with $r_{GS}(z) = x$ and $m_{GS}(z) = y$.*
   (d) *Pushout morphisms: r^*_{GS} is defined for all $x \in dom(r^*_{GS})$ by $r^*_{GS}(x) = [x]_{\equiv}$. The morphism m^*_{GS} is defined symetrically.*
2. *Taking the attribute algebra G_{SIG} as the pushout of r_{SIG} and m_{SIG} in the category $\underline{Alg(SIG)}$.*
3. *Defining the attributions: Let $(attr : s1 \to s2) \in ATTROP$. Then define $attr^H$ by*

$$attr^H(x) := \begin{cases} attr^G(y) & , \text{if } x = r^*_{GS}(y) \\ m_{SIG}(r^{-1}_{SIG}(attr^R(y))) & , \text{if } x = m^*_{GS}(y) \end{cases}$$

Note that $attr^H$ is welldefined due to properties of the pushout H_{GS} and the rule r.

In the following we give a simple example for the transformation of an attributed graph.

Example2 (Firing in Petri nets). Figure 2 shows a transformation simulating the firing of a Petri net as it is described for example in [Rei85]. By using a syntactical algebra i.e. the term algebra over variables as attribute algebra in the rule and a semantical algebra i.e. the well-known algebra of natural numbers in the graph to be transformed we get a pattern matching mechanism included into the formalism. Because the attribute algebras are not finite in general they are not represented in the graph. All variables occuring in the rule build its variable set. Attribution objects are not represented graphically. Note that although there are two possibilies to match the graphical part of the left hand side of the rule in figure 2 we have only one match because of the pattern matching.

If we want to model the firing of Petri nets in general we need a family of rules indexed by the type of the transition to make sure that a transition can only fire if all pre- and all postconditions are fullfilled.

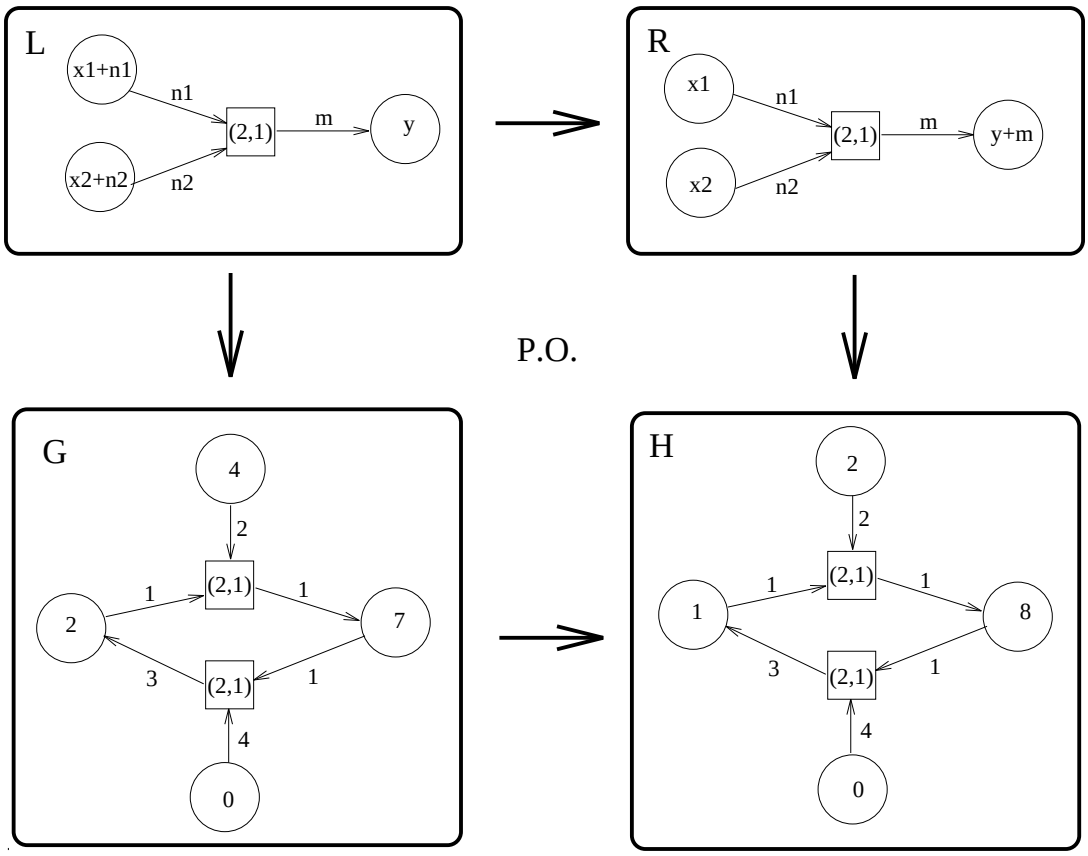

Fig. 2. Simple transformation: Firing in a Petri net

3 Conditional Rules and Transformations

In this section we extend attributed graph transformations by application conditions which are similiar to the ones, introduced in [HHT94] for the non-attributed case. Beside contextual conditions, concerning the graphical part of the match, these application conditions can express equational conditions, that have to be satisfied by its attribute part. After the basic definitions we give some technical results that are needed in the following section, extend the notion of a conditional rule-derived rule to conditional rules and show, how to get rid of positive application conditions by replacing a rule with positive and negative conditions by a set of rules with negative ones, only. Finally, we explicitly compare application conditions to equations for the attribute part and extend our Petri-net example by capacities to illustrate the new concepts.

Definition 7 (Application conditions). 1. An *application condition* $A(r) = (AP(r), AN(r))$ for a simple rule $L \xrightarrow{r} R$ consists of two finite sets of total morphisms starting from L, that contain positive and negative constraints respectively. $A(r)$ is called positive (negative) if $AN(r)$ $(AP(r))$ is empty.

2. Let $L \xrightarrow{l} \hat{L}$ be a positive or negative constraint and $L \xrightarrow{m} G$ a total morphism. We say that m *P-satisfies* l, written $m \models_P l$, if there exists a total morphism $\hat{L} \xrightarrow{n} G$ such that $n \circ l = m$. Furthermore, we say that m *N-satisfies* l if it does not P-satisfy l, i.e. $m \models_N l \iff m \not\models_P l$.
3. Let $A(r) = (AP(r), AN(r))$ be an application condition and $L \xrightarrow{m} G$ a total morphism. Then m *satisfies* $A(r)$, written $m \models A(r)$, if it P-satisfies at least one positive constraint and N-satisfies all negative constraints from $A(r)$, i.e. if

$$\exists\, l \in AP(r).\ m \models_P l \quad \wedge \quad \forall\, k \in AN(r).\ m \models_N k.$$

4. A *conditional rule* is a pair $\hat{r} = (r, A(r))$ consisting of a simple rule r and an application condition $A(r)$ for r.
5. Given a conditional rule $\hat{r} = (L \xrightarrow{r} R, A(r))$ and a match $L \xrightarrow{m} G$ for r, $(\hat{r}, m)$ is called *an application pair* if m satisfies $A(r)$. $\hat{r}$ is *applicable* if there is an application pair $(\hat{r}, m)$.
6. Given $\hat{r}$ and $L \xrightarrow{m} G$ s.t. $\hat{r}$ is applicable to G via m the direct conditional transformation $G \overset{\hat{r},m}{\Longrightarrow} H$ is the simple direct transformation $G \overset{r,m}{\Longrightarrow} H$.

In extension of the corresponding notion for simple rules, a conditional rule $\hat{r}$ is called assignment-controlled if r is assignment-controlled, i.e. $L_{SIG} = R_{SIG} = T_{OP}(X)$.

Note, that definition 7.3 implies that $\hat{r}$ is applicable, only if $AP(r)$ is not empty, i.e. if we don't want to specify extra positive conditions beside from L, we have to take $AP(r) = \{id_L\}$. It is also possible, to consider $AP(r) = \emptyset$ as an extra case in definition 7 but we want to keep this basic definition as simple as possible.

The following proposition shows, how to check applicability of conditional rules, i.e. how to find out, wether there is any match that satisfies a given application condition or not.

Proposition8 (Applicability of conditional rules). *A conditional rule $\hat{r} = (r, (AP(r), AN(r)))$ is applicable if and only if there is some $l \in AP(r)$ s.t. l N-satisfies all negative constraints in $AN(r)$.*

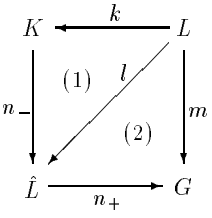

Fig. 3. Characterization of applicability.

Proof. Let $L \xrightarrow{l} \hat{L} \in AP(r)$ s.t. $l \not\models_N k$ for some $k \in AN(r)$, i.e. there is a morphism n_- s.t. (1) in figure 3 commutes. Now, if a match $L \xrightarrow{m} G \models_P l$, we have n_+ s.t. (2) commutes and therefore $(n_+ \circ n_-) \circ k = m$, i.e. $m \not\models_N k$. Conversely, if l satisfies $AN(r)$, $\hat{r}$ is applicable to $\hat{L}$ via l.

By definition 7.3. an application condition consists of a disjunction of positive constraints and a conjunction of negative ones. We show below how to express conjunctions of positive and disjunctions of negative constraints and how this can be extended to whole application conditions.

Definition9 (Conjunction and disjunction). If $L \xrightarrow{l_i} \hat{L}_i$ for $i = 1, 2$ are positive (negative) constraints, their conjunction $l_1 \wedge l_2$ (disjunction $l_1 \vee l_2$) is given by the single constraint $l = l_2^* \circ l_1$ s.t. (1) in the left diagram of figure 4 becomes a pushout.

Given two application conditions $A_i(r) = (AP_i(r), AN_i(r))$ for $L \xrightarrow{r} R$ and $i = 1, 2$, we define their conjunction by $A_1(r) \wedge A_2(r) = (\{l_1 \wedge l_2 |\, l_i \in AP_i(r)\}, AN_1(r) \cup AN_2(r))$ and their disjunction by $A_1(r) \vee A_2(r) = (AP_1(r) \cup AP_2(r), \{l_1 \vee l_2 |\, l_i \in AN_i(r)\})$.

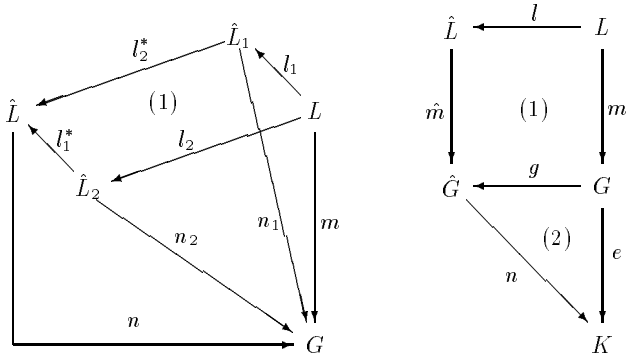

Fig. 4. Combination and translation of constraints.

Proposition10 (Conjunction and Disjunction). *Let $L \xrightarrow{l_i} \hat{L}_i$, $A_i(r)$ for $i = 1, 2$ and l be given as above. Then for all matches $L \xrightarrow{m} G$ we have*

1. $m \models_P l_1 \wedge m \models_P l_2 \iff m \models_P l$,
2. $m \models_N l_1 \vee m \models_N l_2 \iff m \models_N l$,
3. $m \models A_1(r) \wedge m \models A_2(r) \iff m \models A_1(r) \wedge A_2(r)$ *and*
4. $m \models A_1(r) \vee m \models A_2(r) \iff m \models A_1(r) \vee A_2(r)$.

Proof. 1./2.: We show that $\exists\ \hat{L} \xrightarrow{n} G.\ n \circ l = m \iff \exists\ \hat{L}_i \xrightarrow{n_i} G.\ n_i \circ l_i = m$ for $i = 1, 2$. Then proposition 10.1./2. follows from definition 7. Given n as above we have $n_1 = n \circ l_2^*$ and $n_2 = n \circ l_1^*$, where $m = n \circ l = n \circ l_2^* \circ l_1 = n_1 \circ l_1$ (and similarly for n_2). Given n_1, n_2 as above, the left hand side of the equivalence follows from the universal property of $\hat{L}$.

3./4.: $m \models AP_1(r) \vee m \models AP_2(r)$ iff $m \models AP_1(r) \cup AP_2(r)$ and $m \models AN_1(r) \wedge m \models AN_2(r)$ iff $m \models AN_1(r) \cup AN_2(r)$ by definition of satisfaction. Since $AP(r)$ represents a disjunction and $AN(r)$ a conjunction, $m \models AP_1(r) \wedge m \models AP_2(r)$ iff $m \models \{l_1 \wedge l_2 |\, l_i \in AP_i(r)\}$ $m \models AN_1(r) \vee m \models AN_2(r)$ iff $m \models \{l_1 \vee l_2 |\, l_i \in AN_i(r)\}$ by distributivity of $\vee$ over $\wedge$ and vice versa.

Definition11 (Translation). If $L \xrightarrow{m} G$ is a match and $L \xrightarrow{l} \hat{L}$ a constraint, the *translation of l along m* is given by $m^\#(l) = g$ such that (1) in the right diagram of figure 4 is a pushout. Using this, the *translation of an application condition* $A(r) = (AP(r), AN(r))$ for a rule $L \xrightarrow{r} R$ is given by $m^\#(A(r)) = (m^\#(AP(r)), m^\#(AN(r)))$, where $m^\#(A) = \{m^\#(l) |\, l \in A\}$ if A is a suitable set of constraints.

Translation of application conditions is compositional due to the compositionality of pushouts. Therefore, all application conditions build a category together with their translations where the identity is given by the translation along id_L.

Proposition12 (Translation). *Let* $\hat{r} = (L \xrightarrow{r} R, A(r))$ *be a conditional rule and* $L \xrightarrow{m} G$ *a match for* r. *Then, for all matches* $G \xrightarrow{e} K$, $e \models m^{\#}(A(r))$ *iff* $e \circ m \models A(r)$.

Proof. We show that there exists a total morphism $\hat{G} \xrightarrow{n} K$ s.t. $n \circ g = e$ if and only if there exists $\hat{L} \xrightarrow{n'} K$ s.t. $n' \circ l = m \circ e$. This implies, that for every positive (negative) constraint l we have that $e \models_{P(N)} m^{\#}(l)$ iff $e \circ m \models_{P(N)} l$.

Assume n as above. Then we have $n' = n \circ \hat{m}$ and the required commutativity by commutativity of (1) and (2) in the right diagram of figure 4. Now let n' be given as above. Then $\hat{G} \xrightarrow{n} K$ with $n \circ g = e$ exists by universal property of $\hat{G}$.

This allows us to consider the transformation morphism r^* of a direct transformation $G \overset{\hat{r},m}{\Longrightarrow} H$ together with the translated application condition $m^{\#}(A(r))$ as a conditional rule, too. Actually, we can omit the positive part of this application condition because in contrast to the general case of translation we have $m \models A(r)$ by existence of the direct conditonal derivation.

Definition13 (Conditional rule-derived rule). Let $\hat{r} = (L \xrightarrow{r} R, (AP(r), AN(r)))$ be a conditional rule and $G \overset{\hat{r},m}{\Longrightarrow} H$ a transformation via $\hat{r}$. Then, the conditional rule $\hat{r}^* = (G \xrightarrow{r^*} H, A(r^*))$ is called rule-derived via $G \overset{\hat{r},m}{\Longrightarrow} H$ if r^* is the transformation morphism of the direct transformation from G to H and $A(r^*) = (\{id_G\}, m^{\#}(AN(r))))$.

The conditional rule-derived rule behaves just like the usual rule-derived rule as defined in [Löw93].

Lemma14 (Conditional rule-derived rule). *Let* $G \overset{\hat{r},m}{\Longrightarrow} H$ *be a direct conditional transformation,* $\hat{r}^* = (G \xrightarrow{r^*} H, A(r^*))$ *the corresponding conditional rule-derived rule and* $G \xrightarrow{e} K$ *a match for* r^*. *Now,* $e \models A(r^*)$ *iff* $e \circ m \models A(r)$ *and* $K \overset{\hat{r},m \circ e}{\Longrightarrow} S$ *is isomorphic to* $K \overset{\hat{r}^*,e}{\Longrightarrow} S$.

Proof. m satisfies $A(r)$ by existence of $G \overset{\hat{r},m}{\Longrightarrow} H$, i.e. there is $L \xrightarrow{l} \hat{L} \in AP(r)$ s.t. $n \circ l = m$ for some $\hat{L} \xrightarrow{n} G$. This implies that every match $m \circ e$ also satisfies $AP(r)$. On the other hand, every match for r^* satisfies the identity constraint id_G. For the negative part equivalence follows directly from lemma 12.

Finally, (the transformation morphisms of) $K \overset{\hat{r},m \circ e}{\Longrightarrow} S$ and $K \overset{\hat{r}^*,e}{\Longrightarrow} S$ are isomorphic, due to decomposition and composition properties of pushout diagrams (see for example [HS73]).

Positive application conditions for a rule can be expressed directly by expanding this rule into a set of conditional rules, i.e. a conditional rule with positive *and* negative conditions can be seen as a rule scheme for rules with negative conditions only.

Definition15 (Expansion set of conditional rule). Given a conditional rule $\hat{r} = (r, (AP(r), AN(r)))$, we define its expansion set by

$$Exp(\hat{r}) = \{\hat{r}^* \mid \hat{r}^* \text{ is rule-derived via } \hat{L} \overset{\hat{r},l}{\Longrightarrow} \hat{R}, l \in AP(r)\}.$$

Corollary16 (Expansion set of conditional rule). *Let* $\hat{r} = (r, A(r))$ *be a conditional rule. Then for each direct transformation* $G \overset{\hat{r},m}{\Longrightarrow} H$ *there is a direct transformation* $G \overset{\hat{r}^*,n}{\Longrightarrow} H$ *for some* $\hat{r}^* \in Exp(\hat{r})$ *and vice versa such that both transformations are isomorphic.*

Proof. Starting with $G \overset{\hat{r},m}{\Longrightarrow} H$, we take $\hat{r}^*$ to be the conditional rule-derived rule of $\hat{L} \overset{\hat{r},l}{\Longrightarrow} \hat{R}$ for $L \xrightarrow{l} \hat{L} \in AP(r)$ with $m \models_P l$, i.e. there is $\hat{L} \xrightarrow{n} G$ s.t. $n \circ l = m$. Then $\hat{r}^*$ is applicable to G via n and $G \overset{\hat{r}^*,n}{\Longrightarrow} H$ is isomorphic to $G \overset{\hat{r},m}{\Longrightarrow} H$ by lemma 14. Given $G \overset{\hat{r}^*,n}{\Longrightarrow} H$ we define $m = n \circ l$ if $\hat{r}^*$ is rule-derived via $\hat{L} \overset{\hat{r},l}{\Longrightarrow} \hat{R}$. $\hat{r}$ is applicable to G and the corresponding transformations are ismorphic by lemma 14 again.

Note however, that there is no bijective correspondence between transformations by $\hat{r}$ and transformations by $\hat{r}^* \in Exp(\hat{r})$ because in general there is more than one $l \in AP(r)$ that satisfies m.

In the remaining part of this section we explain, how to use positive resp. negative constraints to express equational application conditions concerning the attribute part of the match.

Definition17 (Equations). Let $L \xrightarrow{m} G$ be an attributed graph morphism. Then, a set of *equations* E *for* L is a set of pairs $(a = b)$ s.t. a and b are elements of the same domain of L_{SIG}. Moreover, m_{SIG} is a *solution* for E if $m_{SIG}(a) = m_{SIG}(b)$ for all $(a = b) \in E$.

Proposition18 (Equational constraints). *Let* $L \xrightarrow{r} R$ *be a simple rule,* E *a set of equations for* L *and* $L \xrightarrow{c} \hat{L}$ *a g-isomorphic, a-surjective constraint s.t. the congruence* $Eq(c_{SIG})$ *induced by* c_{SIG} *and the congruence* $\equiv_E$ *generated by* E *are equal. Then, given a match* $L \xrightarrow{m} G$ *for* r, m *P-satisfies (N-satisfies)* c *if and only if* m_{SIG} *is (not) a solution for* E.

Proof. If m_{SIG} is a solution for E, $\equiv_E \subseteq Eq(m_{SIG})$. Then by the homomorphism theorem (see [EM85]) we have m^* s.t. $m^* \circ c = m$. For the converse assume an equation $l = r \in E$ s.t. $m_{SIG}(l) \neq m_{SIG}(r)$. Then $m^*_{SIG} \circ c_{SIG}(l) \neq m^*_{SIG} \circ c_{SIG}(r)$ by the required commutativity and since $c_{SIG}(l) = c_{SIG}(r)$ m^*_{SIG} is not a homomorphism.

If r is assignment-controlled, i.e. if L and R are attributed with the term algebra $T_{SIG}(X)$, $\hat{L}_{SIG}$ is (isomorphic to) the quotient term algebra $T_{SIG}(X)/_{\equiv_E}$ and l_{SIG} is the natural homomorphism.

Corollary 19 (Finite representation). *Given a simple rule $L \xrightarrow{r} R$, sets of equations E_i for L and positive (negative), g-isomorphic and a-surjective constraints $L \xrightarrow{c_i} \hat{L}_i$, s.t. $Eq((c_i)_{SIG}) = \equiv_{E_i}$ for $i = 1, 2$. Moreover, let $c = c_2^* \circ c_1$ be their conjunctive (disjunctive) combination by the pushout $\hat{L}_1 \xrightarrow{c_2^*} \hat{L} \xleftarrow{c_1^*} \hat{L}_2$ of $\hat{L}_1 \xleftarrow{c_1} L \xrightarrow{c_2} \hat{L}_2$. Then, for $L \xrightarrow{m} G$, m_{SIG} is (not) a solution for $E = E_1 \cup E_2$ if and only if m P-satisfies (N-satisfies) c, i.e. c is represented by a finite set of equations if c_1 and c_2 are.*

Proof. m_{SIG} is a solution for E iff m_{SIG} is a solution for E_1 *and* E_2 iff m P-satisfies c_1 *and* c_2 (prop.18) iff m P-satisfies c (prop.10).

Similarily, m_{SIG} is no solution for E iff m_{SIG} is no solution for E_1 *or* for E_2 iff m N-satisfies c_1 *or* c_2 (prop.18) iff m N-satisfies c (prop.10).

Example 3 (Petri nets with capacities).
In this section we extend our Petri net representation to model capacities of places. Therefore, we extend our signature NAT by a constant symbol $\omega :\rightarrow Nat$ to allow infinite capacities and introduce a new attribution $attr_P : P \rightarrow Nat$ assigning to each place its capacity.

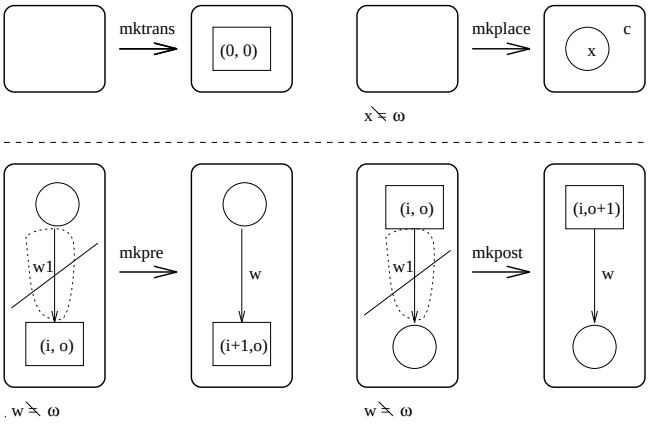

Fig. 5. Rules that build Petri nets.

The graph grammar $GG = (\emptyset_{\mathbf{N}_\omega}, \{mktrans, mkplace, mkpre, mkpost\})$ in figure 5 generates all attributed graphs that properly represent Petri nets, where $\emptyset_{\mathbf{N}_\omega}$ denotes the empty graph, attributed with the natural numbers including ω.

Negative constraints in $mkpre, mkpost$ ensure that there is at most one arc between two given nodes. The non-equations in $mkplace, mkpre$ and $mkpost$ make sure, that weights of arcs and markings of places are finite.

G- and a-injective constraints are distinguished by dotted borders, i.e all nodes and edges outside these borders form the left-hand side and the graphical component of a constraint is given by the left-hand side plus one of the dashed bordered parts and the corresponding embedding. Variables are declared for the left-hand side or for one of the constraints, depending on where they occur in the graphical representation. G-isomorphic and a-surjective constraints are written as (non-)equations at the bottom of the left-hand side of the rule. Constraints are negative if they are crossed through. If no positive constraint is given for a rule $L \xrightarrow{r} R$, $AP(r) = \{id_L\}$. The combination of both types of application conditions is graphically represented by dotted bordered parts with equations inside the border. An example is shown in the next section inside of rule *trans* in the example 4.

Using this interpretation $mkpre = (L \xrightarrow{r} R, (\{id_L\}, \{l_1, l_2\}))$. L and R are attributed with $T_{NAT}(X)$ for $X = \{i, o, w\}$ while $\hat{L}_1$ is attributed with $T_{NAT}(X \cup \{w1\})$ and $\hat{L}_2$ with $T_{NAT}(X)/_{\equiv\{w=\omega\}}$. This is equivalent to the non-equation of figure 5 due to proposition 18.

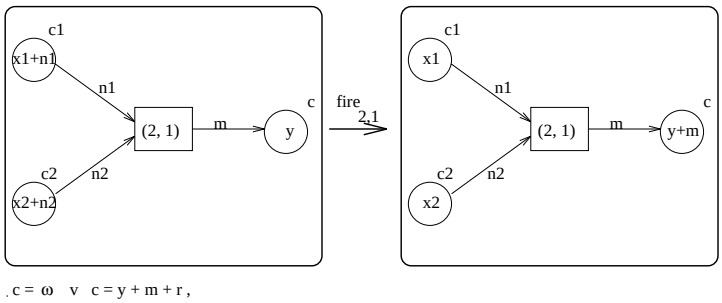

Fig. 6. Firing of transitions of arity (2,1) wrt. capacities.

Firing of transitions wrt. capacities is modeled by rules similiar to $fire_{2,1}$ in figure 6. There, we have to consider two cases. If $c = \omega$, the capacity is unbounded. Otherwise we have to make sure, that $y+m \leq c$ which is done by the equation $c = y+m+r$. These equations are expressed by positive constraints and their disjunction is reflected by the disjunction of constraints in $AP(r)$.

4 Amalgamation of rules

In this section, we describe a concept how (possibly infinite) sets of rules which have certain regularity can be described by a finite set of rules modeling the elementary actions. For the description of such rule schemata the concept of amalgamating rules at subrules is used ([BFH87]).

Graph transformations on such amalgamated rules combine parallel and sequential concepts of graph rewriting and can be regarded as an extension of the parallel graph grammar approach ([EK76]) offering additional features.

The concept of amalgamating rules can be used to describe complex embedding of rule applications in unkown context as well as parallel specification of so-called forall-operations. Moreover, it provides a general concept to describe parallel actions with the possibility of synchronization.

To model interaction between different parallel actions the concept of subrules is used. The well-known definition of a subrule as in [Löw93] is extended to conditional rules in the following definition.

Definition20 (Conditional subrule). Given two conditional, assignment-controlled rules $\hat{r} = (L \xrightarrow{r} R, A(r))$ and $\hat{s} = (L_s \xrightarrow{s} R_s, A(s))$ $\hat{s}$ is called *(conditional) subrule* of $\hat{r}$ if there are total a-quasi-identical morphisms $a : L_s \to L$ and $b : R_s \to R$ called *conditional subrule embedding* $se = (a, b)$ of $\hat{s}$ into $\hat{r}$ (short $\hat{s} \xrightarrow{se} \hat{r}$) such that $b \circ s = r \circ a$.

A so-called application pair $(\hat{s}, m_s)$ is embedded into the application pair $(\hat{r}, m)$ if, additionally, $m \circ a = m_s$.

All conditional rules on objects in $\underline{AGRAPH}_{ATTR}$ and conditional subrule embeddings describe a category $\underline{RULE}$. Furthermore, all application pairs to a graph G with their embeddings form a category $\underline{APAIR}_G$. Identities and compositions in these categories are reduced to those of morphisms. Forgetful functors $V_a, V_b : \underline{RULE} \to \underline{AGRAPH}_{ATTR}$ are defined by $V_a(L \to R) = L, V_a((a,b)) = a, V_b(L \to R) = R$ and $V_b((a,b)) = b$. $V_r : \underline{APAIR}_G \to \underline{RULE}$ is defined by $V_r((\hat{r}, m)) = \hat{r}$ and $V_r(se) = se$ whereas $V_m : \underline{APAIR}_G \to (\underline{AGRAPH}^T_{ATTR} \downarrow G)$[2] by $V_m((\hat{r}, m)) = m$ and $V_m((a,b)) = a$. In further settings, we need also application pairs of rules without conditions and their matches. The corresponding functor $V_c : \underline{APAIR}_G \to \underline{APAIR}_G$ is defined by $V_c(((r, A(r)), m)) = (r, m)$. All functors are summarized in figure 7.

For the description of a complex operation, the elementary rules which can be applied in principle are summerized in so-called presynchronisation rules. Moreover,

[2] $(\underline{AGRAPH}^T_{ATTR} \downarrow G)$ denotes the category of all arrows in $\underline{AGRAPH}^T_{ATTR}$ with codomain G.

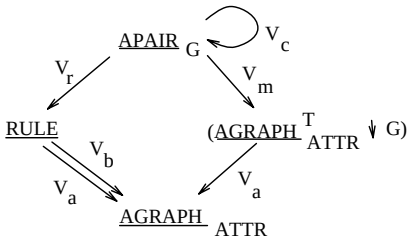

Fig. 7. Forgetful functors

all interaction possibilities with other elementary rules are described by specifying the subrule embeddings allowed. One of the interaction possibilities has to be used for synchronizing the elementary rules. If interaction is not required the empty rule has to be subrule of the corresponding rules.

Definition21 (Presynchronization rule). Let SG be a graph to the graph structure signature $GS = ((S, T, E), (s : E \to S, t : E \to T))$, called *synchronization graph*. A *presynchronization rule* $\mathcal{SR}$ is a graph SG wrt. GS where all T-nodes are labeled with rules, all S-nodes with subrules of them and all edges with corresponding subrule embeddings. Shortly, $\mathcal{SR}$ is a diagram $\mathcal{SR} : SG \to \underline{RULE}$ in the category $\underline{RULE}$. For all $x \in SG_T : I(\mathcal{SR}(x)) = \{\mathcal{SR}(e) | t^{SG}(e) = x, e \in SG_E\}$ describes the *interface* of an elementary rule $\mathcal{SR}(x)$.

$\mathcal{SR}$ is called *simple* if $\forall \hat{e} \in \mathcal{SR}(SG_T)$ the cardinality $|I(\hat{e})| = 1$.

If the embedding of subrule $\hat{s}$ into an elementary rule $\hat{e}$ is permitted to be arbitrary all possible subrule embeddings $\hat{s} \to \hat{e}$ are included in $\mathcal{SR}$.

The concept of presynchronization rules can be compared with that of tables in other approaches to parallel rewriting of graphs (f.ex. in [NA83], [DDK93]) but it is more expressive because the set of possible interfaces between productions can be specified additionally, which cannot be done using tables.

Applying a presynchronization rule to a given graph there are a lot of possibilities to get a covering of the graph, at least a partial. Such a partial covering clears up which elementary rules are applied and how often. Moreover, it determines which matches and subrules are used and what are the corresponding subrule embeddings. Note that these coverings are called partial, because, in general, not the whole graph is covered by elementary matches but these single matches have to be total also in the following.

Definition22 (Partial covering). Let CG be a graph to the graph structure signature GS in definition 21, called *covering graph* such that $\forall t_1, t_2 \in CG_T \exists! (t_1 \xleftarrow{e_1} s \xrightarrow{e_2} t_2) \in CG$.

A *partial covering* COV of an attributed graph G is then a covering graph CG where all nodes of CG_S and CG_T are labeled by application pairs to G and all edges of CG_E are labeled by subrule embeddings, shortly COV is a diagram $COV : CG \to \underline{APAIR_G}$ in category $\underline{APAIR_G}$.

Given a presynchronization rule $\mathcal{SR} : SG \to \underline{RULE}$ $\mathcal{COV}(\mathcal{SR}, G)$ denotes the *set of all coverings* of G by $\mathcal{SR}$ for which there is a graph morphism $c : CG \to SG$ such that the following diagram commutes.

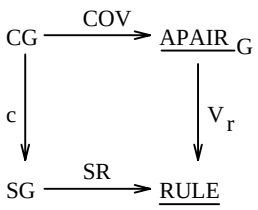

Fig. 8. Partial coverings

Some coverings which are interesting with regard to the examples that have been investigated so far are described in the following definition.

Definition 23 (Special covering constructions). Let all following partial coverings COV be defined as in definition 22. We distinguish the following special sets of coverings which are subsets of $\mathcal{COV}(\mathcal{SR}, G)$:

1. *Basic coverings:* All $COV \in \mathcal{COV}^{basic}(\mathcal{SR}, G)$ satisfy the condition: $|CG_T| = 1$.
2. *Complete coverings*: All $COV \in \mathcal{COV}^{comp}(\mathcal{SR}, G)$ satisfy the condition: The colimit of $V_a \circ V_r \circ COV$ yields G.
3. *Fully synchronized coverings*: All $COV \in \mathcal{COV}^{sync}(\mathcal{SR}, G)$ satisfy the condition: $\forall p = (t_1 \xleftarrow{e_1} s \xrightarrow{e_2} t_2) \in CG : (V_a(V_r(COV(p))))_{GS}$ is the pullback of $(V_m(COV(t_1)), V_m(COV(t_2)))_{GS}$ in the category $\underline{\mathrm{AGRAPH}^T_{GS}}$.[3]
4. *Different-match-coverings*: All $COV \in \mathcal{COV}^{diff}(\mathcal{SR}, G)$ satisfy the condition: $\neg\exists t_1, t_2 \in CG_T : V_m(COV(t_1)) \cong V_m(COV(t_2))$ if $V_r(COV(t_1)) \cong V_r(COV(t_2))$.
5. *All-match-coverings*: Let $\mathcal{COV}^{all}(\mathcal{SR}, G) \subseteq \mathcal{COV}^{diff}(\mathcal{SR}, G)$ such that $\forall COV \in \mathcal{COV}^{all}(\mathcal{SR}, G) : \neg\exists COV' \in \mathcal{COV}^{diff}(\mathcal{SR}, G)$ such that $\forall m \in V_m(COV(CG)) : m \in V_m(COV'(CG'))$ and $\exists m' \in V_m(COV'(CG')) : m' \notin V_m(COV(CG))$.
6. *Local-different-match-coverings*: Let $\mathcal{COV}^{locdiff}(\mathcal{SR}, G) \subseteq \mathcal{COV}^{diff}(\mathcal{SR}, G)$ such that for all $COV \in \mathcal{COV}^{diff}(\mathcal{SR}, G)$ $\forall e_1, e_2 \in CG_E : t^{CG}(e_1) = t^{CG}(e_2)$ implies $e_1 = e_2$.
7. *Local-all-match-coverings*: Let $\mathcal{COV}^{locall}(\mathcal{SR}, G) \subseteq \mathcal{COV}^{locdiff}(\mathcal{SR}, G)$ such that $\forall COV \in \mathcal{COV}^{locall}(\mathcal{SR}, G) : \neg\exists COV' \in \mathcal{COV}^{locdiff}(\mathcal{SR}, G)$ such that $\forall m \in V_m(COV(CG)) : m \in V_m(COV'(CG'))$ and $\exists m' \in V_m(COV'(CG')) : m' \notin V_m(COV(CG))$.

Definition 24 (Synchronization rule). A presynchronization rule $\mathcal{SR}$ together with a covering construction $cc \in \{basic, comp, sync, diff, all, locdiff, local, arb\}$ where arb allows all possible coverings, $(\mathcal{SR}, cc)$ is called *synchronization rule*. Given a graph G the set of coverings $\mathcal{COV}^{cc}(\mathcal{SR}, G)$ is described.

A basic covering is exactly one application pair consisting of a conditional rule and its match to a graph G.

Complete coverings cover the whole graph G, thus, they are *real* coverings.

In coverings which are fully synchronized, the images of all subrules in G describe exactly the overlapping of images of corresponding elementary rules.

Different-match-coverings are not allowed to contain isomorphic matches for isomorphic elementary rules.

The algorithmic way of constructing all-match-coverings for a given presynchronization rule $\mathcal{SR}$ and a graph G is the following:

Look first for all matches of all elementary rules of $\mathcal{SR}(SG_T)$ in G. For each two distinct matches of elementary productions r_1 and r_2 find then a common subrule s in $\mathcal{SR}(SG_S)$ with subrule embeddings into r_1 and r_2 such that the left hand sides of r_1 and r_2 overlap in the mapping of the left hand side of s in G which has to be covered. The covering of G consists of as many copies of all elementary rules as different matches have been found together with their matches in G. Furthermore, for each two copies of them a copy of the right subrules is joined. All elementary rules with matches in G that do not have common subrules do not belong to the covering of G.

All subrules of local-all-match-coverings have to be the same for fullfilling the correponding condition.

The types of covering given are only some examples of covering the mother graph. Other types of coverings are imaginable.

Considering somehow simplified synchronization rules, uniqueness can be stated for some covering constructions.

Corollary 25 (Uniqueness of covering construction). *Let $\mathcal{SR}$ be a presynchronization rule and G a graph.*

[3] The index 'GS' denotes the restriction of the diagram to the graph structure part. For the existence of pullbacks of total graph structure morphisms see [Löw90].

1. *If $\mathcal{SR}$ is simple $|\mathcal{COV}^{all}(\mathcal{SR}, G)| = 1$.*
2. *If $\forall \hat{e} \in \mathcal{SR}(SG_T)$: no two subrule embeddings in $I(\hat{e})$ have subrules with isomorphic left hand sides , $|\mathcal{COV}^{sync}(\mathcal{SR}, G) \cap \mathcal{COV}^{all}(\mathcal{SR}, G)| = 1$.*

Proof. These results follow directly from the assumptions chosen above.

Given a partial covering, an amalgamated graph transformation is performed in two steps. First a new rule, the so-called amalgamated rule, is generated. This rule models the synchronization of all elementary actions at their interface actions. The synchronization is done by gluing the left hand sides and right hand sides of all elementary rules at their interfaces. The amalgamated rule is then applied to the actual graph by constructing the usual single-pushout as described above.

Definition26 (Amalgamated graph transformation). Given a partial covering COV of a graph G as defined in 22, an *amalgamated graph transformation* $G \Longrightarrow G'$ over COV consists of the following two steps:

1. Construction of the *amalgamated rule* $\hat{r} = (r, A(r))$:
   (a) (r, m) is the colimit object of $V_c(COV(CG))$ with $(a_t, b_t) : COV(t) \rightarrow (r, m)$ for all $t \in CG_T$.
   (b) Let $r_t = V_r(COV(t))$ for all $t \in CG_T$ and $s_e = V_r(COV(s^{CG}(e)))$, $t_e = t^{CG}(e)$ and $a_e = V_a(V_r(COV(e)))$ for all $e \in CG_E$. The application condition of r is defined by $A(r) = \bigwedge_{t \in CG_T} a_t^{\#} A(r_t) \wedge \bigwedge_{e \in CG_E} (a_{t_e} \circ a_e)^{\#} A(s_e)$.

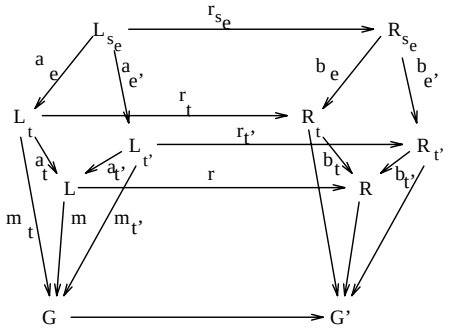

Fig. 9. Amalgamated graph transformation

2. Application of the amalgamated rule $\hat{r}$ to graph G with match m by a simple graph transformation $G \Longrightarrow G'$.

Obviously, $\hat{r}$ is a conditional rule in the sense as defined above.

The colimit of $COV(CG)$ is obtained by constructing the colimits of $cov_L = V_a(V_r(COV(CG)))$ and $cov_R = V_b(V_r(COV(CG)))$. Morphisms r and m exist uniquely by the universal property of cov_L such that $r \circ a_t = r_t$ and $m \circ a_t = m_t$, $\forall t \in CG_T$.

Definition27 (Gluing of graphs). The colimit of morphism stars $MS = (G_{t_e} \xleftarrow{g_e} G_{s_e} \xrightarrow{g_{e'}} G_{t_{e'}})_{e,e' \in CG_E}$ as they are used for the construction of colimits of cov_L and cov_R can be constructed in the following way. MS consists of attributed graphs G_{t_e}, $G_{t_{e'}}$ and G_{s_e} and total a-quasi-identical morphisms $g_e : G_{s_e} \rightarrow G_{t_e}$. The colimit of MS_{GS} in category $\underline{GS}^T$ [4] is constructed by the following steps. Let $\bar{I}_G$ be the equivalence relation generated by the relation $I_G = \{(g_e(x), g_{e'}(x)) \mid x \in G_{s_e}, e, e' \in CG_E\}$. The colimit graph G_{GS} of MS_{GS} is then defined by $G_{GS} = (\biguplus_{e \in CG_E} (G_{t_e})_{GS})_{/\bar{I}_G}$, the quotient set of the disjoint union of all $(G_{t_e})_{GS}$. Functions $(g_{t_e})_{GS} : (G_{t_e})_{GS} \rightarrow G_{GS}$ send each element of $(G_{t_e})_{GS}$ to its equivalence class in G_{GS}.

The attribute algebra $G_{SIG} = T_{SIG}(X)$ where $X = (\biguplus_{t \in CG_T} (X_t))_{/\bar{I}_G}$ if $(r_t)_{SIG} = T_{SIG}(X_t)$. All $(g_t)_{SIG}$ send each element of $T_{SIG}(X_t)$ to its equivalence class in $T_{SIG}(X)$.

Proposition28 (Special colimits in $\underline{AGRAPH}^T_{ATTR}$). *Given a morphism star MS as above, the graph G and morphisms $(g_t)_{t \in CG_T}$ constructed above form the colimit of MS.*

Proof. 1. $(G_{GS}, ((g_t)_{GS})_{t \in CG_T})$ is the colimit of MS_{GS} in $\underline{GS}^T$. $g_{t_e} \circ g_e = g_{t_{e'}} \circ g_{e'}$, $\forall e, e' \in CG_E$, follows directly from the construction of all g_{t_e}.
Given GS-morphism $k_{t_e} : (G_{t_e})_{GS} \rightarrow K_{GS}$ with $k_{t_e} \circ g_e = k_{t_{e'}} \circ g_{e'}$, $\forall e, e' \in CG_E$, there is a GS-morphism $u_{GS} : G_{GS} \rightarrow K_{GS}$ with $u([x]) = k_{t_e}(x)$ for $x \in (G_{t_e})_{GS}$. It is straight forward to check that u is well-defined and that $u \circ g_{t_e} = k_{t_e}$, $\forall e \in CG_E$. Assuming another $v : G_{GS} \rightarrow K_{GS}$ with $v \circ g_{t_e} = k_{t_e}$ it follows that v has to be defined the same way as u. Hence, u is unique.
2. All variable sets X_t, $t \in CG_T$, can be interpreted as graphs to the graph structure signature $NODES$ consisting of one sort. Therefore, X as constructed above is a colimit object in $\underline{Alg(NODES)}$. Since free constructions[5] preserve colimits $(T_{SIG}(X), ((g_t)_{SIG})_{t \in CG_T})$ as defined above is the colimit of MS_{SIG} in $\underline{Alg(SIG)}$.
3. All attributions $attr^G$ are well-defined since they are induced morphisms out of G_{GS} such that $attr^G \circ (g_t)_{GS} = (g_t)_{SIG} \circ attr^{G_t}$, $\forall t \in CG_T$.

Proposition29 (Applicability of amalgamated rule).

[4] $\underline{GS}^T$ is a subcategory of $\underline{GS}$ allowing only total morphisms.

[5] $T_{SIG}(X)$ is the free SIG-algebra over X, see [EM85].

Let COV as defined in 22 and notions r_t, s_e, t_e and a_e as in definition 26. An amalgamated rule r as constructed above is applicable to a graph G by a match m as defined above iff all rules of COV are applicable at their matches in $V_m(COV(CG))$.

Proof. Let $\forall t \in CG_T : m_t \models A(r_t)$ and $\forall e \in CG_E : m_{s_e} \models A(s_e)$. According to colimit properties of $L = V_a(r)$ this is equivalent to $\forall t \in CG_T : m \circ a_t \models A(r_t)$ and $\forall e \in CG_E : m \circ a_{t_e} \circ a_e \models A(s_e)$. Using proposition 11 we get $\forall t \in CG_T : m \models a_t^{\#} A(r_t)$ and $\forall e \in CG_E : m \models (a_{t_e} \circ a_e)^{\#} A(s_e)$. With proposition 9 this is equivalent to $m \models \bigwedge_{t \in CG_T} a_t^{\#} A(r_t) \wedge \bigwedge_{e \in CG_E} (a_{t_e} \circ a_e)^{\#} A(s_e)$.

Embedding of rule application into unknown context is modelled by a simple synchronization rule with the same subrule for all elementary rules. This subrule describes the essential action that has to be embedded into unknown context. In each elementary rule this basic action is then extended by a part of the embedding specified. All local-all-match-coverings are allowed as input for amalgamated graph transformations.

Example 4 (Firing of arbitrary transitions). The firing of arbitrary transitions is described by the following two elementary rules *in* and *out* modelling the actions on input and output places. Since one transition fires these actions have to be synchronized at this transition. Thus, there is a subrule *trans* of *in* and *out* just holding the transition. (See figure 10.) The negative application condition of *trans* prohibits its application in a context where in- and out-places do not satisfy the firing conditions. These application conditions are not necessary in example 3 since the attribute $attr_{arity}$ of transitions is used there.

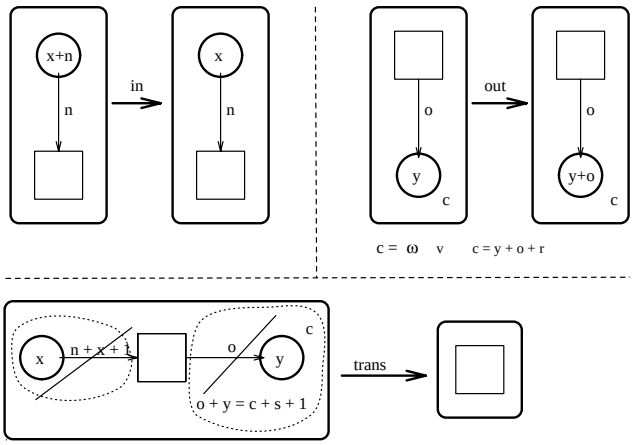

Fig. 10. Elementary rules for the firing of transitions

As conditonal subrule embeddings we allow all possible embeddings from *trans* to *in* and *out* which are exactly two, one to *in* and one to *out*. The corresponding presynchronization rule $\mathcal{SR}$ consists of rules *in*, *out* and *trans* with the obvious embeddings. To fire one transition we are interested in local-all-match-coverings. Thus, we define the synchronization rule $fire = (\mathcal{SR}, locall)$.

Given a net N as in figure 11 we find four different matches $m1, m2, m3, m4$ of *in* and *out* which overlap all in the left transition of N. Similarly, we would find two matches for our elementary rules if we want to fire the right transition. Since all subrule embeddings are clear we get the covering COV over synchronization rule $\mathcal{SR}$ shown in figure 12. Obviously, COV is a local-all-match-covering. If we would allow all-match-coverings the whole graph would be covered. Note that the corresponding amalgamated rule would not model the parallel firing of transitions.

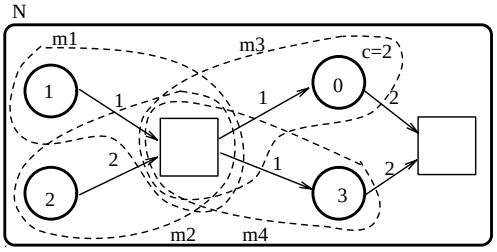

Fig. 11. Example place/transition-net N

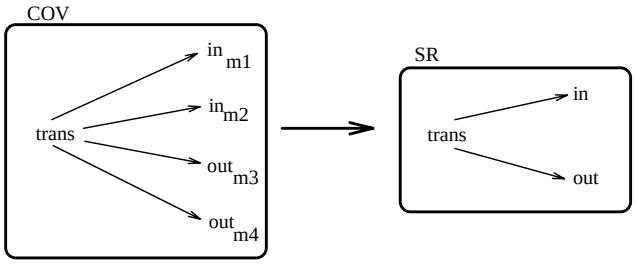

Fig. 12. Covering COV of N by $\mathcal{SR}$

Gluing all rules according to the partial covering $COV(CG)$ we get the amalgamated rule in figure 13 which models the firing of a transition with two input- and two output-places.

For-all-operations are modelled by using general all-match-coverings. If interaction should be performed only via subrules fully-synchronized-coverings are the right means.

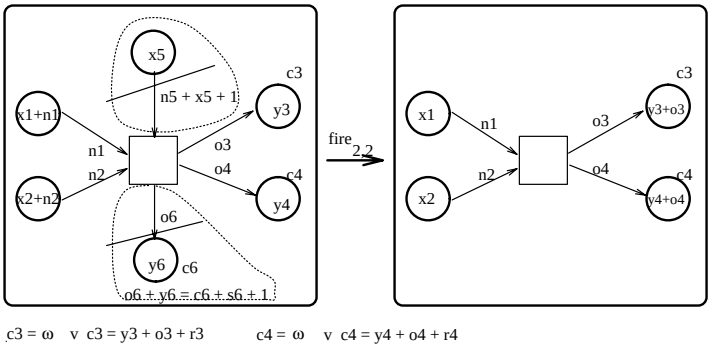

Fig. 13. Amalgamated rule for firing of transitions with arity (2,2)

5 A simple transaction concept

Up to now, we could only control the application of a graph rule by its local context; this effects only the transformation with this rule. Moreover, we can controlle the parallel application of rules by certain synchronizations which are more or less local. To control the sequentual applicability in a global context, i. e. the order and the frequency of rule application for more than one rule, we introduce the notion of a simple transaction concept. This concept allows to define imperative control structures for graph transformation using popular syntactical constructions often found in programming languages. The choice of these constructions is originated in a students project [CDG+94].

Additionally, we introduce a denotational semantic for these transactions based on the semantic defined in [Sch94b]. The application of a transaction to a graph could either succeed and will return another graph or it fails and leave the graph unmodified.

First of all, we make an abstraction of the concrete types of graph rules and graph transformations by the following definitions:

Definition30 (Rule). A rule is a simple rule or a conditional rule or a synchronization rule.

Definition31 (Transformation). A transformation $G \Longrightarrow_r H$ is a simple graph transformation or a conditional graph transformation or an amalgamated graph transformation with rule r.

Now we can define the simple transaction concept for graphs based on rules and transformations:

Definition32 (BNF for transactions). A transaction is defined by the BNF-form in table 1.

Our intention of the above defined syntactical constructors:

- $A; B$ is the sequential composition of the actions A and B
- $\underline{case}(A_1, \ldots, A_n)$ is the sequential working out of the actions A_1 to A_n, i. e. if A_i fails then try A_{i+1}
- $\{A, B\}$ is the representation of a nondeterministical choice between the action A and B
- $\square$ is an operator to indicate that an action should be active as long as possible
- $^{1,\ldots,n}$ are operators to define that an action should happen $1, \ldots, n$ times

For the definition of the denotational semantic for our transactions we will first of all introduce here the BCF-form (basic control flow) from [Sch94b] in table 2.

Now we shortly describe the intention of the BCF-form:

- $\underline{skip}$ represents the identity operator
- $\underline{loop}$ which represents the nonterminating looping construction
- $\underline{def}(A)$ returns a graph G if the action A is applicable to G
- $\underline{undef}(A)$ is the complement of $\underline{def}(A)$
- $(A; B)$ is the same as $A; B$
- $(A[]B)$ is the same as $\{A, B\}$.

In [Sch94b] there is a fixpoint semantic defined for the BCF-form on an underlying semantic domain $\mathcal{D} := 2^{\mathcal{G} \times (\mathcal{G} \cup \{\infty\})}$, whereby $\mathcal{G}$ is a set of graphs, we adapt that to $\mathcal{G} \subseteq Obj(\underline{AGRAPH}_{ATTR})$ and ∞ represents potentially nonterminating computations.

If we want to apply fixpoint theory to recursively defined transactions we have to construct a suitable partial order for our semantic domain $\mathcal{D}$. This is done in [Sch94b] and the partial order "$\leq$" for $R, R' \in \mathcal{D}$ is defined as follows:

$R \leq R' :\Leftrightarrow \forall G, G' \in \mathcal{G} : (G, G') \in R \Rightarrow (G' = \infty \wedge (G, G') \in R') \vee (G, \infty) \notin R \Rightarrow ((G, G') \in R \Leftrightarrow (G, G') \in R')$.

Obviously "$\leq$" is a partial order and it is shown that its "chains have joins" condition holds:

Let $(R^\alpha)_{\alpha \in ordinal} \subseteq \mathcal{D}$ be a chain, i. e. for any ordinals α and β holds: $R^\alpha \leq R^\beta$. Then the join of the given chain is defined as follow: $R := \cup(R^\alpha) := \{(G, G') | G' \neq \infty \vee \exists \alpha : (G, G') \in R^\alpha \wedge G' = \infty \vee \forall \alpha : (G, \infty) \in R^\alpha\}$.

Now, a semantic function $\mathcal{R}$ is defined from the syntactical domain of BCF expressions or transactions onto the semantic domain $\mathcal{D}$. Let $A, B \in < BCF - Exp >$ and $\mathcal{R} :< BCF - Exp > \rightarrow \mathcal{D}$ and $\mathcal{P}$ is a set of rules which is adapted to $\mathcal{P} \subseteq Obj(\underline{RULES})$ a set of rules.

1. $(G, G') \in \mathcal{R}[\underline{skip}] :\Leftrightarrow G = G'$.
2. $(G, G') \in \mathcal{R}[\underline{loop}] :\Leftrightarrow G' = \infty$.
3. $(G, G') \in \mathcal{R}[r] :\Leftrightarrow G \Longrightarrow_r G'$, for any rule $r \in \mathcal{P}$.
4. $(G, G') \in \mathcal{R}[\underline{def}(A)] :\Leftrightarrow \exists G'' \neq \infty : (G, G'') \in \mathcal{R}[A] \wedge G = G' \vee (G, \infty) \in \mathcal{R}[A] \wedge G' = \infty$.

$$
\begin{array}{ll}
< transaction > & ::= < transactionId > \text{ “}=\text{”} \| \text{ “} < transactionbase > \text{ “}|\|\text{”}; \\
< transactionbase > & ::= < elementary - transaction > \\
 & \quad | < elementary - transaction > \text{ “};\text{ “} < transactionbase > \\
 & \quad | \text{ “}\underline{case}\text{(“} < transactionbase > \text{ “,“} < othercase > \text{ “)”}; \\
< othercase > & ::= < transactionbase > \\
 & \quad | < transactionbase > \text{ “,“} < othercase >; \\
< elementary - transaction > & ::= \text{ “}\{\text{“} < actionset > \text{ “}\}\text{“} < applicationfrequency >; \\
< applicationfrequency > & ::= \text{ “}\square\text{“}|\text{“}^1\text{“}|\ldots|\text{“}^n\text{“}; \\
< actionset > & ::= < action > | < action > \text{ “,“} < actionset >; \\
< action > & ::= < ruleId > | < transactionId >;
\end{array}
$$

Table 1. BNF of transactions

$$
\begin{array}{ll}
< Transaction > & ::= < TransactionId > \text{ “=“} < BCF - Exp >; \\
< BCF - Exp > & ::= < BasicAction > | < ActionCall > | < BCF - Term >; \\
< BasicAction > & ::= \text{“}\underline{skip}\text{“}|\text{“}\underline{loop}\text{“}; \\
< ActionCall > & ::= < RuleId > | < TransactionId >; \\
< BCF - Term > & ::= \text{“}\underline{def}\text{““(“} < BCF - Exp > \text{“)“}| \\
 & \quad \text{“}\underline{undef}\text{““(“} < BCF - Exp > \text{“)“}| \\
 & \quad \text{“(“} < BCF - Exp > \text{“;“} < BCF - Exp > \text{“)“}| \\
 & \quad \text{“(“} < BCF - Exp > \text{“}[]\text{“} < BCF - Exp > \text{“)”};
\end{array}
$$

Table 2. BCF-form of transactions

5. $(G,G') \in \mathcal{R}[\underline{undef}(A)] :\Leftrightarrow (\neg\exists G'' : (G,G'') \in \mathcal{R}[A]) \wedge G = G' \vee (G,\infty) \in \mathcal{R}[A] \wedge G' = \infty$.
6. $(G,G') \in \mathcal{R}[(A;B)] :\Leftrightarrow \exists G'' \neq \infty : (G,G'') \in \mathcal{R}[A] \wedge (G'',G') \in \mathcal{R}[B] \vee (G,\infty) \in \mathcal{R}[A] \wedge G' = \infty$.
7. $(G,G') \in \mathcal{R}[(A[]B)] :\Leftrightarrow (G,G') \in \mathcal{R}[A] \vee (G,G') \in \mathcal{R}[B]$.

To develop a fixpoint semantics for recursively defined transactions we will use the following version of a fixpoint theorem:

Let F be a monotonic function(al) on a partially ordered set in which every chain has a join, and let f^α, for ordinal α, be defined inductively by $F^\alpha = (\cup_{\beta:\beta<\alpha} : f(f^\beta))$. Then F has a least fixpoint given by f^α, for some ordinal α.

Definition33 (Semantic for transactions).
Let $A, B \in < transactionbase >$. The semantic function $Sem :< transactionbase > \rightarrow \mathcal{D}$ is defined as follows:

- $Sem(A_1;A_2;\ldots;A_n) := \mathcal{R}[(A_1;T_1)]$ with $T_i = (A_{i+1};T_{i+1})$ for $i = 1,\ldots,n-1$ and $T_{n-1} = A_n$
- $Sem(\{A_1,A_2,\ldots,A_n\}) := \mathcal{R}[(A_1[]T_1)]$ with $T_i = (A_{i+1}[]T_{i+1})$ for $i = 1,\ldots,n-1$ and $T_{n-1} = A_n$
- $Sem(\underline{case}(A_1,A_2,\ldots,A_n)) := \mathcal{R}[((\underline{def}(A_1);A_1)[](\underline{undef}(A_1);T_1)]$ with $T_i = ((\underline{def}(A_{i+1});A_{i+1})[](\underline{undef}(A_{i+1});T_{i+1})$ for $i = 1,\ldots,n-1$ and $T_{n-1} = ((\underline{def}(A_n);A_n)[]\underline{undef}(A_n))$
- $Sem(A^\square) := \mathcal{R}[T]$ with $T = ((A;T)[]\underline{undef}(A))$
- $Sem(A^n) := \mathcal{R}[(A;T_1)]$ with $T_i = (A;T_{i+1})$ for $i = 1,\ldots,n-1$
- $Sem(r) := \mathcal{R}[r]$.

In [Sch94b] it is shown that $\mathcal{R}$ is a monotonic function. There is also shown that there exists a fixpoint semantics for transactions with and without recursion. Therefore, the semantic function $\mathcal{R}$ is expanded to $\mathcal{R}[E] : \mathcal{D}^n \rightarrow \mathcal{D}$ whereby E denotes a BCF expression containing transaction identifiers $T_1,\ldots,T_n$. Provided with the semantics $\mathcal{R}[T_1],\ldots,\mathcal{R}[T_n]$ of T_1 to T_n the semantics of E is given by the definition of the semantics for BCF expressions above and is denoted as follows: $\mathcal{R}[E][\mathcal{R}[T_1],\ldots,\mathcal{R}[T_n]]$. Then in [Sch94b] it is shown that the following construction defines fixpoint semantics for transactions: Let T_1 to T_n be transactions of the form $T_1 = E_1[T_1,\ldots T_n],\ldots,T_n = E_n[T_1,\ldots,T_n]$ then the following holds: (1) $T_1,\ldots,T_n$ have least fixpoints $\mathcal{R}[T_1],\ldots,\mathcal{R}[T_n] \in \mathcal{D}$ and (2) approximantions of these fixpoints may be constructed as follows: $R_0^i := \mathcal{D} \times \{\infty\}$ and $R_i^{k+1} := \mathcal{R}[E_i][R_1^k,\ldots R_n^k]$ for $i = 1\ldots n$ where $\mathcal{R}[E_i] : \mathcal{D}^n \rightarrow \mathcal{D}$ takes approximations for $T_1,\ldots,T_n$ and yields a new approximation for T_i by applying $\mathcal{R}$ to expression E_i.

Now we are able to define a semantic for recursively transactions in our sense:

Definition34 (Semantic for recursively transactions).
Let E be a transaction as defined in 32 which contains transaction idenitfiers T_1 to T_n. Then the semantic function $Sem[E] : \mathcal{D}^n \rightarrow \mathcal{D}$ is defined by $Sem[E] := \mathcal{R}[E]$.

Now, we are able to show that our semantic function

Sem is monoton and has a fixpoint semantics:

Corollary 35 (Fixpoint semantics for transactions). *Let T be a transaction and $Sem :< transactionbase >\rightarrow \mathcal{D}$ is the semantic function defined above, then Sem is a monotonic function and $Sem(T) \in \mathcal{D}$ has a least fixpoint.*

Proof. That *Sem* is a monotonic function follows direct from the definition of it and the fact that $\mathcal{R}$ is monoton. A least fixpoint $Sem(T) \in \mathcal{D}$ exist because the transactions defined in [Sch94b] have a fixpoint semantics.

Example 5 (Deadlocks in a Petri net). Using the rule *fire* as specified above to describe deadlocks in a Petri net we can define the following simple transaction: *Find-deadlocks* $:= \| \; fire\square \; \|$. Note, if we regard the semantics of *Find-deadlocks* the resulting set of graphs describe all situations with a deadlock in the given Petri net.

6 Conclusion

Combinig attribution concepts and control mechanisms in one graph transformation formalism the question arises if this formalism is adequate to specify software systems such as the AGG-system. Adequacy means here convient and powerful concepts without being overloaded. Comparing our formalism with logic based programmed structure rewriting presented by A. Schürr in [Sch94b, Sch94a] we may get a first answer. The extensions we made in this paper bring the algebraic graph grammar approach close to programmed structure rewriting systems. That approach functions as the underlying formalism of PROGRES ([Sch91]), a rule and graph based language for the specification of software systems. Similarily to PROGRES, it seems to be more convient to develop a useful syntax for our control concepts. For all extensions some proposals concerning graphical and textual notation are made and have to be worked out. A graphical description of control often increases clearness, consider for example Petri nets.

All extensions of the single-pushout approach to graph transformations made in this paper are relatively young, moreover, the approach itself is not old. Thus, first steps in those directions are made and a lot of extensions of each of the concept are imaginable. For example, a change of the underlying category to that of partial algebras with partial homomorphisms such that graph objects can be considered as attributes is investigated in [WG94]. Application conditions may cover also cardinality restrictions concerning the number of edges or nodes in certain parts of a graph, path expressions, etc. Amalgamation of rules should be classified according to different embedding and synchronization problems and interesting structures of synchronization and covering graphs should be examined.

A big and difficult task is to carry over theoretical results known for the pure SPO approach to our new formalism as, for example, concepts of independence and parallelism of graph transformations, embedding, concurrency, etc. First results are present for attribution, application conditions and amalgamated graph transformations but have to be extended and combined.

REFERENCESREFERENCESREFERENCES

[BFH87] P. Böhm, H.-R. Fonio, and A. Habel, *Amalgamation of graph transformations: a synchronization mechanism*, Journal of Computer and System Science **34** (1987), 377–408.

[BT93] M. Beyer and G. Taentzer, *Amalgamated graph transformation systems and their use for specifying AGG – an algebraic graph grammar system*, Accepted for Proceedings of Graph grammar workshop in Dagstuhl 93, 1993.

[CDG+94] M. Conrad, J. Demuth, M. Gajewsky, R. Holl, M. Rudolf, and S. Weber, *Spezifizieren mit Graphtransformationen und Petrinetzen am Beispiel eines Graph-Editier- und Transformations-Systems*, students project report, May 1994.

[DDK93] G. David, F. Drewes, and H.-J. Kreowski, *Hyperedge replacement with rendezvous*, to appear in proc. of 12th conf. of FST and TCS'92, 1993.

[Ehr79] H. Ehrig, *Introduction to the algebraic theory of graph grammars*, 1st Graph Grammar Workshop, Lecture Notes in Computer Science 73, 1979, pp. 1–69.

[EK76] H. Ehrig and H.-J. Kreowski, *Parallel graph grammars*, Automata,Languages, Development (A. Lindenmayer and G. Rozenberg, eds.), Amsterdam: North Holland, 1976, pp. 425–447.

[EM85] H. Ehrig and B. Mahr, *Fundamentals of algebraic specification 1: Equations and initial semantics*, EATCS Monographs on Theoretical Computer Science, vol. 6, Springer, Berlin, 1985.

[HHT94] Annegret Habel, Reiko Heckel, and Gabriele Taentzer, *Graph grammars with negative application conditions*, submitted to special issue of Fundamenta Informaticae, 1994.

[Hof93] L. Hofmann, *Anwendung algebraischer Graphgrammatiken zur Werkstückerkennung im Rahmen von CIM*, Studienarbeit, TU Berlin, 1993.

[HS73] H. Herrlich and G. Strecker, *Category Theory*, Allyn and Bacon, Rockleigh, New Jersey, 1973.

[LKW93] M. Löwe, M. Korff, and A. Wagner, *An algebraic framework for the transformation of attributed graphs*, Term Graph Rewriting: Theory and Practice (M.R. Sleep, M.J. Plasmeijer, and M.C. van Eekelen, eds.), John Wiley & Sons Ltd, 1993, pp. 185–199.

[Löw90] M. Löwe, *Extended algebraic graph transformation*, Ph.D. thesis, TU Berlin, 1990.

[Löw93] M. Löwe, *Algebraic approach to single-pushout graph transformation*, TCS **109** (1993), 181–224.

[NA83] A. Nakamura and K. Aizawa, *On a relationship between graph l-systems and picture languages*, TCS **24** (1983), 161–177.

[Rei85] W. Reisig, *Petri nets*, Springer Verlag, 1985.

[Sch91] A. Schürr, *Progress: A vhl-language based on graph grammars*, LNCS532, Springer, 1991.

[Sch94a] A. Schürr, *Logic based programmed structure rewriting systems*, submitted to special issue of Fundamenta Informaticae, 1994.

[Sch94b] A. Schürr, *Logic based structure rewriting systems*, Graph Transformations in Computer Science (H.-J. Schneider and H. Ehrig, eds.), Springer, 1994.

[Tae92] G. Taentzer, *Parallel high-level replacement systems*, Tech. Report 92/10, TU Berlin, 1992.

[Tae94] Gabriele Taentzer, *Synchronous and asynchronous graph transformations*, submitted to special issue of Fundamenta Informaticae, 1994.

[WG94] A. Wagner and M. Gogolla, *Defining operational behavior of object specifications by attributed graph transformations*, submitted to special issue of Fundamenta Informaticae, 1994.

This article was processed using the LaTeX macro package with LLNCS style