Spatial and Temporal Refinement of Typed Graph Transformation Systems *

Martin Große–Rhode[1], Francesco Parisi–Presicce[2], and Marta Simeoni[2]

[1] Dip. di Informatica, Università di Pisa, Corso Italia, 40, I – 56125 Pisa, Italy,
mgr@di.unipi.it
[2] Università di Roma *La Sapienza*, Dip. Scienze dell'Informazione,
Via Salaria 113, I-00198 Rome, Italy,
{parisi,simeoni}@dsi.uniroma1.it

Abstract. Graph transformation systems support the formal modeling of dynamic, concurrent, and distributed systems. States are given by their graphical structure, and transitions are modeled by graph transformation rules. In this paper we investigate two kinds of refinement relations for graph transformation systems in order to support the development of a module concept for graph transformation systems. In a spatial refinement each rule is refined by an amalgamation of rules, in a temporal refinement it is refined by a sequence of rules.

1 Introduction

Graph grammars and graph transformation systems, in their different variations, have become a well accepted approach to the formal modeling of systems. (For a survey see [Roz97].) In this paper we investigate refinement relations between graph transformation systems, a question that has been addressed only few in the literature up to now (see [CH95,HCEL96,Par96,Rib96]). Our main concern are refinement relations that preserve the full behaviour of graph transformation systems, as opposed to [CH95,HCEL96] for instance, whose refinement relation guarantees only the existence of specialised transformations in the refining system, not the whole behaviour. Using typed graph transformation systems ([CEL$^+$96]) refinement also supports the *implementation* of a more abstract system by another more concrete one. Thereby type restriction corresponds to the hiding of implementation details.

A possible application of refinement is the development of a module concept for graph transformation systems. Well investigated in the field of programming languages module concepts have been carried over also to formal specification approaches, as for instance algebraic specification of abstract data types (see e.g. [BEP87,EM90]). Basically, a module is given by an export and an import interface, and a body that *implements* the features offered at the export interface, possibly using the features required at the import interface. A necessary

* This research has been supported by the TMR Network GETGRATS, ERB-FMRX-CT960061.

Luboš Brim et al. (Eds.): MFCS'98, LNCS 1450, pp. 553–561, 1998.

formal means to define such modules for formal specifications are *morphisms* between the specification units for the three parts, that model these relationships appropriately. That means, morphisms are required that model the inclusion of the imported features into the body, and morphisms that model the relation between the exported features and their implementation in the body. Since the latter task is of more general nature there should be an embedding of morphisms of the first kind (inclusions) into morphisms of the second kind (implementations). In [EM90] *horizontal* composition operations have been introduced, such as union and composition via import–export interface matching. The essential requirement on the category of specification units to support these horizontal operations is that pushouts (more generally colimits) of specifications exist. For the special and most important case of import–export interface matching it suffices already, if pushouts of inclusions and implementations exist.

The first kind of morphisms between graph transformation systems, corresponding to inclusions, are mappings between the name sets that are compatible with the associated rules. In a refinement morphism names are mapped to *instructions* that indicate how a rule is refined to a composition of rules of the refining system. In a spatial refinement, several rules of the refining system are glued together in parallel (amalgamated) to obtain the effect of the original rule. That means, the different rules of the refining system must be applied at the same time to different, possibly overlapping parts of the actual graph (state), and their simultaneous application yields the same successor graph as the original rule. In a temporal refinement, a sequential composition of rules refines a given one, i.e. the sequential computation steps are refined.

The paper is organized as follows. In the next two sections graph transformation systems and refinements are introduced for the untyped case. Although this case is not very meaningful for applications, the separated presentation makes the presentation easier. In section 2 basic definitions and facts of graph transformation systems and their behaviour are revisited. In section 3 spatial and temporal refinements are introduced. In section 4 types for graph transformation systems and the extension and restriction constructions associated with type morphisms are revisited. Finally in section 5 the results of the previous sections are put together to obtain the results we consider useful for applications. Full proofs and further examples can be found in the technical reports [GPS97a,GPS97b].

2 Graph Transformation Systems

In this section we briefly review the standard definitions and facts of graph transformation systems. A *graph* $G = (N, E, src, tar)$ is given by a set N of nodes, a set E of edges, and functions $src, tar : E \to N$ that assign source and target nodes to each edge. Thus graphs are unlabeled directed graphs that may have multiple edges and loops. A *graph morphism* $f = (f_N, f_E) : G \to G'$ is given by functions $f_N : N \to N'$ and $f_E : E \to E'$ such that $src' \circ f_E = f_N \circ src$ and

$tar' \circ f_E = f_N \circ tar$. With identities and composition being defined component wise this defines the category **Graph**.

A *graph transformation rule* $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ is given by a left graph L, that is matched to the actual state graph when the rule is applied, a right graph R by which the occurrence of L is replaced, and a span $L \leftarrow K \rightarrow R$, given by a gluing graph K and graph morphisms to L and R. The span expresses which items of L are related to which items of R. Intuitively, items related in this way are preserved when the rule is applied, and items in $L - K$ are deleted. A *rule morphism* $mp = (mp_L, mp_K, mp_R) : p \rightarrow p'$ is given by graph morphisms $mp_L : L \rightarrow L'$, $mp_K : K \rightarrow K'$, and $mp_R : R \rightarrow R'$, that commute with l and l', and r and r' respectively, i.e. $mp_L \circ l = l' \circ mp_K$ and $mp_R \circ r = r' \circ mp_K$. With component wise identities and composition this defines the category **Rule**. The *amalgamation* of two rules w.r.t. a common subrule is their pushout in **Rule**.

A *graph transformation system* $\mathbf{G} = (P, \pi)$ is given by a set P of names, that is considered as its signature, and a mapping $\pi : P \rightarrow |\mathbf{Rule}|$ that assigns to each name a rule, thus specifying the behaviour. A *morphism of graph transformation systems*, $f : \mathbf{G} \rightarrow \mathbf{G}'$ is a mapping $f : P \rightarrow P'$ between the sets of rule names that is compatible with π and π', i.e. $\pi' \circ f = \pi$. With composition and identity inherited from **Set** , this defines the category **GTS**.

Since **Graph** and **Rule** are (isomorphic to) functor categories to **Set** and **GTS** is a comma category to **Set** all three categories are cocomplete.

Given a graph transformation system $\mathbf{G} = (P, \pi)$ a *direct derivation* $p/m : G \Rightarrow H$ over $\mathbf{G}$ from a graph G via a rule p and a matching morphism $m : L \rightarrow G$ is a pair (p, S), where $p \in P$, S is a double pushout diagram

$$\begin{array}{ccccc} L & \xleftarrow{\ l\ } & K & \xrightarrow{\ r\ } & R \\ {\scriptstyle m}\downarrow & (po) & {\scriptstyle k}\downarrow & (po) & \downarrow{\scriptstyle h} \\ G & \xleftarrow[\ \bar{l}\ ]{} & D & \xrightarrow[\ \bar{r}\ ]{} & H \end{array}$$

in **Graph**, and $\pi(p) = (L \xleftarrow{l} K \xrightarrow{r} R)$. G is called the *input*, and H the *output* of $p/m : G \Rightarrow H$. A *derivation* $p_1/m_1; \ldots; p_n/m_n : G \Rightarrow H$ over $\mathbf{G}$ from a graph G via rules $p_1, \ldots, p_n$ and matching morphisms $m_1, \ldots, m_n$ is a sequence of direct derivations over $\mathbf{G}$, such that the output of the i'th direct derivation is the input of the $(i+1)$'st direct derivation. The set of all derivations over $\mathbf{G}$ is denoted $Der(\mathbf{G})$. Using amalgamated rules for derivations allows to prescribe synchronized derivations. The expressive power of amalgamated rules is in general higher than sequential composition, see [BFH87]. For a derivation with an amalgamated rule q we use the notation $\bar{q}/n : G \Rightarrow H$. Note that q is a rule here, whereas p in $p/m : G \Rightarrow H$ is a rule name. The set of all derivations over $\mathbf{G}$ with amalgamated rules is denoted $ADer(\mathbf{G})$.

Considering $Der(\mathbf{G})$ as the behaviour of a graph transformation system, morphisms $f : \mathbf{G} \rightarrow \mathbf{G}'$ preserve behaviour. I.e., for each derivation $d : G \Rightarrow H$ with $d = (p_1/m_1; \ldots; p_n/m_n)$ in $Der(\mathbf{G})$ there is a derivation $f(d) : G \Rightarrow H$ in $Der(\mathbf{G}')$, where $f(d) = (f(p_1)/m_1; \ldots; f(p_n)/m_n)$. The same holds for $ADer(\mathbf{G})$.

3 Untyped Refinements

As mentioned in the introduction a refinement of a graph transformation system is given by a mapping that associates with each rule name an instruction how to implement the associated rule as a composition of rules of the refining system. In a spatial refinement this composition is an amalgamation, in a temporal refinement a sequence.

Definition 1 (Refinement Instructions). *Let* $\mathbf{G} = (P, \pi)$ *be a graph transformation system. A* spatial refinement instruction *si* on $\mathbf{G}$ *is defined by:*

$$si = (p_1 \dots p_k, (\pi(p_i) \xleftarrow{m_{ij}} r_{ij} \xrightarrow{m'_{ij}} \pi(p_j))_{1 \le i < j \le k})$$

where $p_1, \dots, p_k \in P$, $r_{ij} \in |\mathbf{Rule}|$ *and* $\pi(p_i) \xleftarrow{m_{ij}} r_{ij} \xrightarrow{m'_{ij}} \pi(p_j)$ *for* $1 \le i < j \le k$ *is a span of morphisms in* **Rule**.

A temporal refinement instruction $ti = p_1 \dots p_k$ on $\mathbf{G}$ *is a string of sort names* $p_1, \dots, p_k \in P$, *such that* $R_i = L_{i+1}$ *for* $i \in \{1, \dots, k-1\}$ *if* $\pi(p_j) = (L_j \leftarrow K_j \rightarrow R_j)$.

The sets of spatial refinement instructions and temporal refinement instructions on $\mathbf{G}$ *are denoted* $SRI(\mathbf{G})$ *and* $TRI(\mathbf{G})$ *respectively.*

The rule result(si) of a spatial refinement instruction si is defined as the colimit in **Rule** *of the diagram given by all spans of si with their adjacent rules. The result of a temporal refinement instruction* $ti = p_1 \dots p_k$ *is given by* $result(ti) = (L_1 \xleftarrow{l} K \xrightarrow{r} R_k)$, *where* K *is the limit of the diagram*

$$L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1 = L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} \cdots \xleftarrow{l_k} K_k \xrightarrow{r_k} R_k$$

in **Graph**, *and* $l : K \to L_1$ *and* $r : K \to R_k$ *are the corresponding projections.*

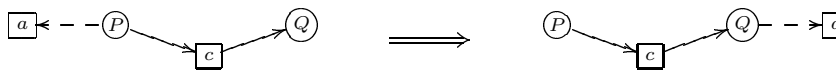
Definition 2 (Refinement Morphisms). *Let* $\mathbf{G} = (P, \pi)$ *and* $\mathbf{G}' = (P', \pi')$ *be two graph transformation systems. A* spatial (temporal) refinement morphism $ref : \mathbf{G} \to \mathbf{G}'$ *is a mapping from the set of rule names* P *to the set of spatial (temporal) refinement instructions* $SRI(\mathbf{G}')$ *(resp.* $TRI(\mathbf{G}')$*) such that, for* $p \in P$, $result(ref(p)) \cong \pi(p)$.

Two refinement morphisms $ref : \mathbf{G} \to \mathbf{G}'$ *and* $ref' : \mathbf{G} \to \mathbf{G}'$ *are* equivalent *if, for all* $p \in P$, $result(ref(p)) \cong result(ref'(p))$ *in* ***Rule***.

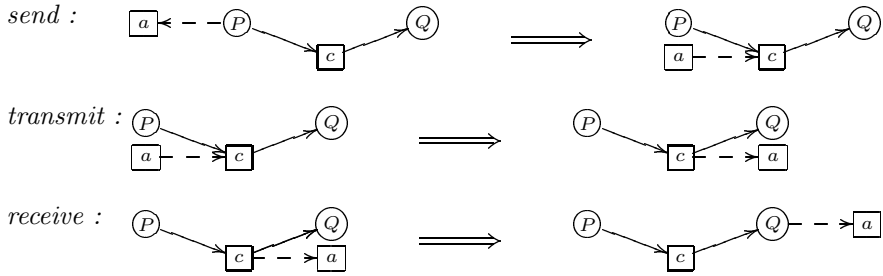
From this definition follows immediately that any two refinement morphisms $ref : \mathbf{G} \to \mathbf{G}'$ and $ref' : \mathbf{G} \to \mathbf{G}'$ are equivalent.

Example 1 (Asynchronous Communication). Consider an asynchronous communication of agents P and Q, with writing and reading access to a common channel c. Agent P holds a value a, that it sends to Q. Thus the abstract view of the communication is

asynch-com :



Refined into intermediate steps this communication would look as follows.



First P sends a to c, then a is transmitted from the input port of c to its output port, where Q can receive it.

The spatial and temporal refinement relations are transitive. Compositions of refinement morphisms are constructed via pullbacks and pushouts respectively ([GPS97b]). Thus it cannot be expected that composition is strictly associative. Moreover, it is impossible to obtain pushouts on this level, that were required for horizontal composition of modules. This is due to the fact that a refinement of a single rule may have arbitrary many component rules, that may be connected in many ways. Thus a common refinement of two given ones, that were required to construct a pushout, does not exist. Therefore we abstract from the concretely given refinement instructions at this point, and proceed with *existence* of spatial refinements alone.

Definition 3 (Categories of Refinements). *The* spatial (temporal) refinement category $\mathbf{SR}_{\equiv}$ *(resp.* $\mathbf{TR}_{\equiv}$*) has graph transformation systems as objects and equivalence classes of spatial (temporal) refinements as morphism.*

Since all diagrams in these categories commute, all colimits of refinements exist. They can be constructed as disjoint union (i.e. *coproducts*) of the rule name sets of the components and the induced mappings to the rules. Note that the diagram of morphisms on the underlying sets of names of a colimit diagram in $\mathbf{SR}_{\equiv}$ or $\mathbf{TR}_{\equiv}$ need not commute.

Proposition 1 (Colimits of Refinements). *The categories* $\mathbf{SR}_{\equiv}$ *and* $\mathbf{TR}_{\equiv}$ *have colimits.*

As mentioned in the introduction pushouts of inclusion morphisms and spatial refinements are required for the horizontal composition of modules. The obvious embedding of morphisms of graph transformation systems into the refinement categories yields a more intuitive way to construct a pushout of (the embedding of) an injective $\mathbf{GTS}$–morphism $f : \mathbf{G}_0 \to \mathbf{G}_1$ and a spatial refinement morphism $sr : \mathbf{G}_0 \to \mathbf{G}_2$. In this case the set of names of the pushout can in fact be taken as a pushout in **Set** with the induced mappings to the rules, thus it also commutes.

Since spatial refinements use amalgamations a refining system must be able to use amalgamated rules. This is reflected in the preservation properties for the two kinds of refinements.

Theorem 1 (Preservation of Behaviour). *Let $sr : \mathbf{G} \to \mathbf{G}'$ be a spatial refinement morphism of graph transformation systems. For each derivation $d : G \Rightarrow H$ with $d = (p_1/m_1; \ldots; p_n/m_n)$ in $Der(\mathbf{G})$ there is an amalgamated derivation $\bar{d} : G \Rightarrow H$ in $ADer(\mathbf{G}')$, where $\bar{d} = (q_1/m_1; \ldots; q_n/m_n)$ and $q_i = result(sr(p_i))$.*

Let $\mathbf{G}'$ be a graph transformation system with injective rules, i.e. for each rule name $p' \in P'$ the graph morphisms l'_i, r'_i of $\pi'(p'_i) = (L'_i \xleftarrow{l'_i} K'_i \xrightarrow{r'_i} R'_i)$ are injective, and $tr : \mathbf{G} \to \mathbf{G}'$ be a temporal refinement morphism. Then for each derivation $d : G \Rightarrow H$ with $d = (p_1/m_1; \ldots; p_n/m_n)$ in $Der(\mathbf{G})$ there is a derivation $d' : G \Rightarrow H$ in $Der(\mathbf{G}')$, where $d' = (p'_{11}/m_{11}; \ldots; p'_{nk_n}/m_{nk_n})$ and $tr(p_i) = p'_{i1} \ldots p'_{in_i}$ for $i = 1, \ldots, n$.

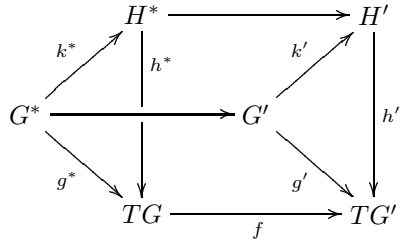
4 Typed Graph Transformation Systems

In [CMR96] typed graphs have been introduced as a technical means for the construction of graph processes. This typing, however, may also be considered as a structuring means for graph transformation systems in the usual sense of typing. Morphisms and refinements of graph transformation systems with type graphs as structuring means have been introduced in [CH95,HCEL96,Rib96].

Definition 4 (Categories of Typed Graphs and Typed Rules). *Given a graph TG, a TG–*typed graph *g is a graph morphism $g : G \to TG$, and a TG–*typed graph morphism *$k : g \to h$ is a graph morphism with $h \circ k = g$. This defines the category $\mathbf{Graph}_{TG}$.*

The category $\mathbf{Rule}_{TG}$ of TG–typed rules is given by TG–typed graph spans and morphisms, as for untyped rules.

Definition 5 (Retyping, Forgetful Functor and Free Functor). *Let $f : TG \to TG'$ be a graph morphism. f induces a* backward retyping functor *$f^< : \mathbf{Graph}_{TG'} \to \mathbf{Graph}_{TG}$, $f^<(g') = g^*$ and $f^<(k' : g' \to h') = k^* : g^* \to h^*$ by pullbacks and mediating morphisms as in the following diagram,*



and a forward retyping functor *$f^> : \mathbf{Graph}_{TG} \to \mathbf{Graph}_{TG'}$, $f^>(g) = f \circ g$ and $f^>(k : g \to h) = k$ by composition.*

Forward and backward retyping functors are left and right adjoints, i.e. for each $f : TG \to TG'$ we have $f^> \dashv f^< : \mathbf{Graph}_{TG} \to \mathbf{Graph}_{TG'}$. Moreover, forward retyping is a functor $\_^> : \mathbf{Graph} \to \mathbf{Cat}$, and backward retyping is a

pseudo functor $\_^< : \mathbf{Graph} \to \mathbf{Cat}^{op}$, i.e. $(id_{TG})^< \cong id_{\mathbf{Graph}_{TG}}$ and $(e \circ f)^< \cong f^< \circ e^<$.

Typing a graph transformation system means to define a type system as a type graph TG, and have all rules typed w.r.t. TG.

Definition 6 (Typed Graph Transformation System). *A* typed graph transformation system $\mathbf{TG} = (TG, P, \pi)$ *consists of a type graph* TG*, a set of rule names* P*, and a mapping* $\pi : P \to |\mathbf{Rule}_{TG}|$*, associating with each rule name its* TG*–typed rule.*

A morphism of typed graph transformation systems must first of all relate their type systems, i.e. it must contain a type graph homomorphism. Forward and backward retyping then induce translations to compare the rules of both systems. In general, however, compatibility with the forgetful functor (backward retyping) is too weak to preserve derivations.

Definition 7. *A* morphism of typed graph transformation systems, $f = (f_P, f_{TG}) : \mathbf{TG} \to \mathbf{TG}'$ *is given by a mapping* $f_P : P \to P'$ *between the sets of rule names and an injective type graph morphism* $f_{TG} : TG \to TG'$*, such that* $f^>_{TG}(\pi(p)) = \pi'(f_P(p))$ *for all* $p \in P$ *.*

Typed graph transformation systems and morphisms form a category, called **TGTS**.

Proposition 2 (Colimits). *The category* ***TGTS*** *has colimits.*

The preservation theorem for morphisms of typed graph transformation systems compares the behaviour of the systems again in both directions. Each derivation of the first system gives rise to a corresponding derivation in the second system, and the forgetful image of the derivation coincides with the original one.

Theorem 2 (Preservation of Behaviour). *Let* $f = (f_P, f_{TG}) :$ $\mathbf{TG} \to \mathbf{TG}'$ *be a TGTS–morphism. For each derivation* $d : G \Rightarrow H$ *with* $d = (p_1/m_1; \ldots; p_n/m_n)$ *in* $Der(\mathbf{TG})$ *there is a derivation* $f(d) : f^>_{TG}(G) \Rightarrow f^>_{TG}(H)$ *in* $Der(\mathbf{TG}')$*, where* $f(d) = (f(p_1)/f^>_{TG}(m_1); \ldots; f(p_n)/f^>_{TG}(m_n))$*. Furthermore* $f^<(f(d) : f^>_{TG}(G) \Rightarrow f^>_{TG}(H)) = (d : G \Rightarrow H)$*.*

5 Typed Refinements

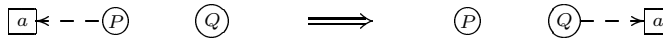
Having defined refinements and typing we are now in a position to combine the results and obtain the constructions that we consider appropriate for the module concept discussed in the introduction, and other applications. In the following definition the sets $TySRI(\mathbf{G}')$ and $TyTRI(\mathbf{G}')$ of *typed spatial and temporal refinement instructions* are given by replacing all rules by TG'–typed rules. Their *results* are the corresponding constructions in $\mathbf{Rule}_{TG'}$.

Definition 8 (Typed Refinements). *Let* $\mathbf{TG} = (TG, P, \pi)$ *and* $\mathbf{TG}' = (TG', P', \pi')$ *be typed graph transformation systems. A* typed spatial (temporal) refinement morphism $tr = (r, f_{TG}) : \mathbf{TG} \to \mathbf{TG}'$ *is given by a mapping* $r : P \to TySRI(\mathbf{G}')$, *(resp.* $r : P \to TyTRI(\mathbf{G}')$*) and an injective type graph morphism* $f_{TG} : TG \to TG'$ *, such that* $result(r(p)) \cong f^{>}_{TG}(\pi(p))$ *for all* $p \in P$.

Theorem 3 (Preservation of Behaviour). *Let* $ref = (r, f_{TG}) : \mathbf{TG} \to \mathbf{TG}'$ *be a typed refinement. For each derivation* $d : G \Rightarrow H$ *with* $d = (p_1/m_1; \ldots; p_n/m_n)$ *in* $Der(\mathbf{TG})$ *there is a derivation* $ref(d) : f^{>}_{TG}(G) \Rightarrow f^{>}_{TG}(H)$ *over* $Der(\mathbf{TG}')$, *where* $ref(d) = (q_1/f^{>}_{TG}(m_1); \ldots; q_n/f^{>}_{TG}(m_n))$ *and* $q_i = result(r(p_i))$. *Furthermore* $f^{<}_{TG}(ref(d) : f^{>}_{TG}(G) \Rightarrow f^{>}_{TG}(H)) = (d : G \Rightarrow H)$.

Example 2. The channel used for the asynchronous communication of agents P and Q in example 1 should be considered as an internal communication infrastructure for the group that contains P and Q, and should not be visible from the outside. With the type graphs for the abstract system (left) and the refined system (right)

(that we implicitely already used in example 1) we obtain as visible (abstract) behaviour the rule



which is given by the backward retyping along the type inclusion of the rule *asynch-com*. Then the rules *send*, *transmit*, and *receive* given in example 1, together with rules to connect and disconnect agents P and Q via channel c, form a typed temporal refinement of this abstract behaviour, using the internal communication channel.

6 Conclusion

In this paper we have started the investigation of refinements of graph transformation systems. For two cases we have shown how these can be defined in a categorical framework. The background has been the use of refinement morphisms as relations between the interfaces and the body of a graph transformation module, similar to algebraic specification modules. The pattern has been the same in both cases, and the investigation of further possibilities of refinement, as e.g. the combination of temporal and spatial refinement as discussed here, would also pursue this pattern. First the shape of the refinement instructions is defined, basing on operations of rules like amalgamation and sequential composition. It must be shown then, that these refinement instructions can be composed vertically in order to obtain transitivity of refinement and to allow for a categorical treatment. The latter proceeds by taking equivalence classes

of refinement morphisms, i.e. a pre–order of refinement relationships is considered. This reflects the fact that the vertical composition in most cases is not associative, and pushouts do not exist. However, passing to the abstract level of existence of refinements instead of concrete refinement constructions yields the framework for the investigation and general formulation of further properties, like the existence of pushouts for the general case, and the special case of pushout of a refinement with an inclusion morphism. This particular result is especially important for the application to graph transformation module composition, since it corresponds to the composition of modules via their export and import interfaces.

References

BEP87. E.K. Blum, H. Ehrig, and F. Parisi-Presicce. Algebraic specification of modules and their basic interconnections. *JCSS*, 34(2-3):293–339, 1987.

BFH87. P. Böhm, H.-R. Fonio, and A. Habel. Amalgamation of graph transformations: a synchronization mechanism. *JCSS*, 34:377–408, 1987.

CEL$^+$96. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and J. Padberg. The category of typed graph grammars and its adjunctions with categories of derivations. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94,* Springer LNCS 1073, pages 240–256. 1996.

CH95. A. Corradini and R. Heckel. A compositional approach to structuring and refinement of typed graph grammars. *Proc. of SEGRAGRA'95 "Graph Rewriting and Computation", Electronic Notes of TCS*, 2, 1995. http://www.elsevier.nl/locate/entcs/volume2.html .

CMR96. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26(3,4):241–266, 1996.

Ehr79. H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *1st Graph Grammar Workshop* Springer LNCS 73, pages 1–69. 1979.

EM90. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*, volume 21 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin, 1990.

GPS97a. M. Große–Rhode, F. Parisi-Presicce, and M. Simeoni. Concrete spatial refinement constructions for graph transformation systems. Technical Report 97-10, Università di Roma *La Sapienza*, 1997.

GPS97b. M. Große–Rhode, F. Parisi-Presicce, and M. Simeoni. Spatial and temporal refinement of typed graph transformation systems. Technical Report 97-11, Università di Roma *La Sapienza*, 1997.

HCEL96. R. Heckel, A. Corradini, H. Ehrig, and M. Löwe. Horizontal and vertical structuring of typed graph transformation systems. *MSCS*, pages 1–35, 1996.

Par96. F. Parisi-Presicce. Transformation of graph grammars. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94*, Springer LNCS 1073, 1996.

Rib96. L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars*. PhD thesis, TU Berlin, 1996.

Roz97. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific Publishing, 1997.