# An algebra of graph derivations using finite (co–) limit double theories

Andrea Corradini[1], Martin Große–Rhode[2], and Reiko Heckel[3] *

[1] Dip. Informatica, Università di Pisa, andrea@di.unipi.it,
[2] TU Berlin, FB Informatik, mgr@cs.tu-berlin.de
[3] Universität–GH Paderborn, FB Mathematik–Informatik, reiko@uni-paderborn.de

**Abstract.** Graph transformation systems have been introduced for the formal specification of software systems. States are thereby modeled as graphs, and computations as graph derivations according to the rules of the specification. Operations on graph derivations provide means to reason about the distribution and composition of computations. In this paper we discuss the development of an algebra of graph derivations as a descriptive model of graph transformation systems. For that purpose we use a categorical three level approach for the construction of models of computations based on structured transition systems. Categorically the algebra of graph derivations can then be characterised as a free double category with finite horizontal colimits.
One of the main objectives of this paper is to show how we used algebraic techniques for the development of this formal model, in particular to obtain a clear and well structured theory. Thus it may be seen as a case study in theory design and its support by algebraic development techniques.

## 1 Introduction

Our main concern with this paper is to document a case study on the use of algebraic specifications or, more generally, algebraic development techniques. However, the application in this case is not the development of a software system or its specification, but rather the development of a theory that shall then be used for system specification and design. We are convinced that in order for a theory to be applicable it must itself have a clear and manageable structure, which can be obtained by a good theory design. This is particularly important for complex theories that aim at a wide range of aspects of formal system specification. Carrying over the *separation of concerns* principle from software engineering it means that the different aspects or levels of the theory must be well separated and encapsulated in order to allow for a focussing on the particular aspect concerned.

Algebraic development techniques provide powerful tools for the formal specification, structuring, model construction, and formal verification for a wide range

* This work has been carried out during a stay of the second and third author at the University of Pisa, supported by the EEC TMR network GETGRATS (General Theory of Graph Transformation Systems) ERB-FMRX-CT960061.

of systems. To discuss the applicability of algebraic techniques to theory design let us briefly sketch the formal model we are developing, the algebra of graph derivations, and its intended meaning.

Graph grammars are a further development of usual string or tree grammars for formal languages, that allow to handle more complex structures — graphs instead of words — as intuitive representations of system, program, or data states for example. Using this approach for the specification of the dynamic behaviour of systems, i.e. their computations, the emphasis shifts from the consideration of the language generated by a grammar to its *derivations*, that model the computations. Thus, under this perspective, *graph transformation systems* are considered rather than graph grammars. A semantical model of graph transformation systems can then be given by an algebra whose elements are graph derivations and whose operations describe how graph derivations can be composed sequentially (in time) and in parallel (in space). The main feature of the theory we are developing is the precise definition of this algebra of derivations.

A general method for the development of models of computations from given formal models of programs that supports the above mentioned algebraic structure has been introduced in [Cor90,CM92]. Models of computations are there given by categories with algebraic structure on objects and morphisms. The categorical composition operation models the sequential composition of computations, whereas the algebraic structure is used to express the distribution of states and computations. In order to apply this method to a particular domain, such as graph transformation systems, only the algebraic structure of the states and the atomic state transitions have to be given. The model of computations is fully determined by these data. Thus the purpose is to find the right algebraic structure of graphs to obtain the appropriate notion of computations.

Altogether both the domain we address and the method we use are well suited for an algebraic specification approach: the domain is an *algebra* of graph derivations, the method makes use of *algebraic* and categorical (*essentially algebraic*) structures.

In the following sections we introduce the method, the domain, and the application of the method to the domain and discuss some problems and their consequences for the development and structure of our theory. Then we sketch the algebra of graph derivations in this framework, and discuss the benefits of this model. The full formal definitions and results are presented in [Hec98].

## 2 The Method: Structured Transition Systems

In [Cor90,CM92] a method for the construction of models of computations from given formal models of programs has been introduced.

The first level is given by the **programs**, i.e. the syntactic entities that represent the behaviour of the system. A program is given by a transition system whose set of states comes equipped with an algebraic structure. The state operations are composition operations that indicate how larger states can be put together from smaller ones. To reduce the complexity the whole algebra of states is considered already on the level of programs, although it is in general infinite. The transitions however are an unstructured set on this level, that describe the atomic steps the systems can perform. This set can be assumed to be finite.

Consider as examples of programs in this sense P/T–nets $N = (P, T, pre, post)$, given by sets $P$ and $T$ of places and transitions respectively, and functions $pre, post : T \to P^{\oplus}$ that assign to each transition multisets of places, the pre and post conditions ([MM90]). The states are then given by multisets of places, i.e. markings of the net, with addition and empty set as algebraic operations. Note that this algebra of states is the free commutative monoid over the set $P$ of places. Distribution of states is directly modelled via the markings; correspondingly addition of multisets is the composition operation for (distributed) states. The transition system associated to a P/T–net has the set $P^{\oplus}$ of multisets of places as set of states. Its transitions are given by $t : pre(t) \to post(t)$ for each $t \in T$ . In this case, as in many other examples, the algebra of states is a free algebra over a given set. For the general methodology this assumption however is not necessary.

In the second level, given by **structured transition systems**, distributed computation steps, corresponding to the distribution of the states, are introduced. First the labeled transition system given on the first level is extended by idle steps for all states. Then the state operations are extended to the transitions. Formally this construction is obtained by replacing the set of transitions by the free algebra generated by this set, including the idle transitions. For the P/T–net example this means to construct the free commutative monoid over transitions and places, with set $(T \cup P)^{\oplus}$ of multisets of transitions and places as carrier. The functions *pre* and *post* are then extended accordingly, i.e., the pre and post conditions of a multiset are given by the sums of the pre and post conditions of its elements. The composed transitions correspond to the parallel composition of the atomic transitions.

Finally the third level, introducing the **models of computations**, is given by the categories freely generated by the structured transition systems. These categories inherit the algebraic structure both on objects (states) and arrows (computations), and have implicit compatibility laws for sequential and parallel composition given by the categorical axioms. The model of computations of a P/T–net for example is given by the set of multisets of places (=markings) as states as before, and sequential compositions of parallel transitions as computations. These are subject to the categorical axioms, such as associativity and neutrality of the empty set, and the interchange law for sequential and parallel composition. This means e.g. $(t_1+t_2);(t_3+t_4) = (t_1;t_3)+(t_2;t_4)$ , provided both sides are well defined (where ; and + denote sequential and parallel composition respectively). More precisely, the algebraic structure of the model of computations of a net is a *strict symmetric, strict monoidal category*, and as such it is perfectly consistent with the corresponding models as defined in [MM90].

The STS–method just described has been applied successfully to various computational formalisms just by changing the basic algebraic structure of states. As we just outlined, in the case of P/T Petri nets the relevant structure is that of commutative monoids; other applications include Phrase Structure Grammars, for which the algebraic structure is that of monoids, Term Rewriting Systems, where states are represented as arrows of a small cartesian category, Term Graph Rewriting, where *gs-monoidal* categories are used [CG97], and pure Logic Programming, where the models turn out to be *indexed monoidal* categories [CA93]. Concerning our application the STS–method has three main advantages:

1. Both steps, the construction of the structured transition system from the program and the construction of the model of computations from the structured transition system, are *free constructions*. Since free constructions preserve colimits the construction of the model of computations is also compositional w.r.t. all composition operations for programs that can be modelled by colimits. (Recall that colimits are the general categorical construction to 'put things together' [Gog91], thus the restriction is acceptable.)
2. Applying the STS–method to a suitable algebraic presentation of graphs we obtain an algebra of graph derivations with sequential and parallel composition operations for free, given by the model of computations delivered by the method.
3. The categorical axioms induce an equivalence (actually a congruence) on computations, that coincides with important given notions of *behavioural equivalence*, i.e., an equivalence up to the rearrangement of independent steps, in the concrete applications of the method. This yields a clear semantical interface for notions of behavioural equivalence that appear more complicated in the special instances.

## 3 The Domain: DPO–Graph Rewriting

In this section we briefly review the main definitions and facts of graph rewriting in the double pushout (DPO) approach ([Ehr79,Roz97]). 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**. The approach can be easily extended to labelled or typed graphs, or arbitrary *graph structures*. These are total algebras to a signature that contains only unary operation symbols. Such a signature is called *hierarchical* if it does not yield loops of operation symbols.

A *graph rewriting 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 $L \leftarrow K \to R$, given by a gluing graph $K$ and injective graph morphisms to $L$ and $R$. The span expresses which items of $L$ are related to which items of $R$ , thus defining how $R$ will be connected to the host graph $G$ to which the rule is applied. Intuitively, items related by the span are preserved in a rewriting, and items in $L - K$ are deleted. A *rule morphism* $mp = (mp_L, mp_K, mp_R) : p \to p'$ is given by graph morphisms $mp_L : L \to L'$, $mp_K : K \to K'$, and $mp_R : R \to R'$, 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 $\pi : P \to |\mathbf{Rule}|$ that assigns to each name a rule, thus specifying the atomic steps. A *morphism of graph transformation systems*, $f : \mathbf{G} \to \mathbf{G}'$ is a mapping $f : P \to P'$ between the sets of rule names that is compatible with $\pi$ and $\pi'$, 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* $p/m : G \Rightarrow H$ from a graph $G$ via a rule $p$ and a matching morphism $m : L \to G$ is a pair $(p, S)$, where $p \in P$, $S$ is a double pushout diagram

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(po)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] \arrow[dr, phantom, "(po)"] & R \arrow[d, "n"] \\
G & D \arrow[l, "g"] \arrow[r, "h"'] & H
\end{tikzcd}$$

in **Graph**, and $\pi(p) = (L \xleftarrow{l} K \xrightarrow{r} R)$. $G$ is called the *input*, and $H$ the *output* of $p/m : G \Rightarrow H$. Note that, due to the pushout construction, $D$ and $H$ are only determined up to isomorphism by $G$, $p$ and $m$. A *derivation* $p_1/m_1; \ldots; p_n/m_n : G \Rightarrow H$ 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 w.r.t. $\mathbf{G}$, such that the output of the $i$'th direct derivation is the input of the $(i+1)$'st direct derivation. The span relating the input $G$ and the output $H$ of two consecutive direct derivations $p/m : G \Rightarrow G'$ and $p'/m' : G' \Rightarrow H$ is defined via the pullback of the embeddings of the context graphs $D$ and $D'$ of the single steps into the intermediate graph $G'$ .

$$\begin{tikzcd}
G & D \arrow[l] \arrow[r] & G' \arrow[d, phantom, "(pb)"] & D' \arrow[l] \arrow[r] & H \\
 & & D'' \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

*Behavioural equivalence* of graph derivations in the DPO–approach is given by the so called *shift equivalence*. Consider again two consecutive direct graph derivations $p/m : G \Rightarrow G'$ and $p'/m' : G' \Rightarrow H$ .

$$\begin{tikzcd}
L_1 \arrow[d] & K_1 \arrow[l] \arrow[r] \arrow[d] & R_1 \arrow[dr] \arrow[drrr, dashed] & & L_2 \arrow[dl] \arrow[dlll, dashed] & K_2 \arrow[l] \arrow[r] \arrow[d] & R_2 \arrow[d] \\
G & D \arrow[l] \arrow[rr] & & G' & & D' \arrow[ll] \arrow[r] & H
\end{tikzcd}$$

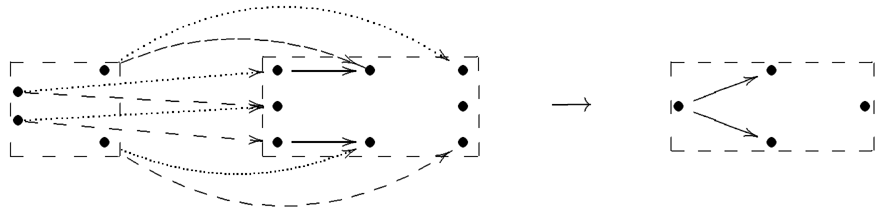
If there is a morphism $L_2 \to D$ such that $L_2 \to D \to G' = L_2 \to G'$, then this states that the part of $G'$ to which $p'$ is applied is already present in $G$, i.e., $p'$ can be applied to $G$. Vice versa, if $R_1 \to G'$ factors through $D'$ , then all of the right hand side $R_1$ of the first rule application is still present in the final output graph $H$. In this case the direct derivations $p/m : G \Rightarrow G'$ and $p'/m' : G' \Rightarrow H$ are called *sequentially independent*. The *parallelism theorem* of DPO–graph rewriting states that under these conditions the order of the derivations can be reversed, i.e., there are a graph $G''$ and matchings $n' : L_2 \to G$ and $n : L_1 \to G''$ such that $p'/n' : G \Rightarrow G''$ and $p/n : G'' \Rightarrow H$ are derivations. Moreover, the same effect can be obtained by applying immediately the parallel rule $p+p'$ to $G$, i.e., there is a parallel direct derivation $p + p'/k : G \Rightarrow H$ . The essential idea of the proof is, that two sequentially independent rule applications only overlap in the context that both derivations leave unchanged. That means, the input $G$ can be decomposed as a pushout of two local graphs overlapping in a common part, such that each rule applies to one local graph and both are identity transformations on the common part. Finally two *derivations are shift equivalent*, if one can be obtained from the other by repeatedly exchanging the order of sequentially independent consecutive direct derivations. Normal forms can be obtained by applying each rule in a derivation as early as possible.

The three levels of the STS–method roughly correspond to the elements of DPO–graph rewriting as follows. (1) Programs are graph transformation systems, (2) structured transition systems are parallel direct derivations (i.e. double pushouts), and (3) models of computations are shift equivalence classes of derivations. We will make this correspondences more precise in section 5.

## 4 The Application of the STS–Method to DPO–Graph Rewriting

To apply the STS–method now the algebraic structure of graphs has to be made explicit, i.e. the composition (and construction) operations for graphs — as representatives of distributed states — have to be defined.

Many papers in the literature propose some (essentially) algebraic presentation of suitables classes of graphs (see for example [BC87,EV97,GH98]). Clearly, the choice of the right algebraic presentation of graphs is fundamental for our goals, because the STS–method will lift via a free construction this structure to the transitions and then to the computations of a system. We do not discuss here what would be the result of applying the method to the various presentations of graphs proposed in the literature, because this goes beyond the scope of the paper. Instead, we will introduce directly the algebraic structure of graphs that we shall use, and that will be shown to induce correct structure on transtitions and computations. For that purpose the following observation is crucial. Graphs are total algebras to the algebraic specification $GRAPH$ with two sorts $E$ and $N$ and two operations $src, tar : E \to N$ . More generally (hierarchic) graph structures are total algebras to a (hierarchic) graph signature $GS$. The specification generates a *finite limit theory* $FL(GS)$, which is a category with finite limits that is universal among the categories with finite limits containing the specification $GS$. Its objects are basically sets of equations, and its morphisms are equivalence classes of tuples of terms. Now the dual $FL(GS)^{op}$ of $FL(GS)$ is equivalent to the subcategory of the finitely generated $GS$–algebras, i.e., those that are isomorphic to quotients of term algebras with finitely many variables, factorized by a finite set of equations [UG71]. If $GS$ is hierarchic the finitely generated algebras are exactly the finite ones. More concretely this observation means that each finite graph can be constructed as a finite colimit of instances of the atomic graphs $\bullet$ (one node, no edges) and $\bullet \to \bullet$ (two nodes, one edge). To construct a graph with $n$ nodes and $m$ edges first an $n$–fold coproduct of $\bullet$ and $m$-fold coproduct of $\bullet \to \bullet$ is taken, then the edges are connected appropriately by a coequalizer. The graph with four nodes and two edges as depicted on the right below for example is the coequalizer of the graph with four isolated nodes and two isolated edges (in the middle), glued at four points via the two morphisms indicated by the dashed and the dotted arrows respectively on the left.



To consider finite colimits, or more specifically binary coproducts, coequalizers, pushouts, and initial objects, as operations distinguishes the algebraic from the categorical approach. Categorically colimits are determined only up to isomorphism. That means, an object, together with a family of morphisms, may have the *property* of being a colimit, but each isomorphic object then has the same property. Thus being a colimit object does not define an operation. On the other hand each category with finite colimits can be made an algebra by defining appropriate operations that choose *designated* colimits. Let $FCCAT$ be a partial equational specification with conditional existence equations, introducing operations for binary coproducts, coequalizers, pushouts, and initial objects. (A partial equational specification of categories with finite limits can be found in [CGW95].) Partial $FCCAT$–algebras are then (small) categories with designated finite colimits, and there is clearly an embedding from the category $\mathbf{PAlg}(FCCAT)$ of partial $FCCAT$–algebras to the category **FCCat** of small categories with finite colimits. However, these categories are not equivalent, due to the morphisms. In **FCCat** it is required that *functors preserve finite colimits*, i.e. $F(A+B) \cong F(A) + F(B)$ etc. In $\mathbf{PAlg}(FCCAT)$ homomorphisms must preserve *the designated colimits*; i.e. $h(A+B) = h(A) + h(B)$ etc. Thus there are fewer $FCCAT$–homomorphisms than **FCCat** functors. Nevertheless, universal properties of free algebras can be carried over to weakened universal properties of pseudo free categories, where the uniqueness of the mediating homomorphism is replaced by uniqueness up to isomorphism of the mediating functor. Correspondingly, free categories are only defined up to equivalence. Roughly speaking, this possibility of carrying over the universal property is due to the fact that all possible choices of colimits yield isomorphic objects, thus the different mediating functors induced by the different choices are all isomorphic.

As a consequence for our model construction we must distinguish between an *algebraic* and a *categorical* definition of the state space. In the first case we consider the *algebra of finite graphs* as state space, with composition/construction operations given by designated finite colimits. The advantage is that we can use the partial equational specification $FCCAT$ to obtain an inductive definition of the state space via the construction of the finite limit theory $FL(GS)$ as free partial $FCCAT$–algebra generated by $GS$. In the second case we consider the *category of finite graphs* as state space. Here the operations are given — via finite colimits — only up to isomorphism. This conception is technically easier to deal with, because it does not require concrete, fixed colimit constructions.

In any case the state space is given by the category of finite graphs, be it considered as an algebra or not. In the sequel we call this category the *horizontal category* (of states). The model of computations that is to be constructed is also given by a category, called the *vertical category* (of computations). Combining these two we end up with a *double category with finite horizontal colimits* as structure representing the algebra of graph derivations. We will denote this categorical structure as DFHC–categories for short. So we could summarize the construction by the slogan:

> *The algebra of graph derivations of a given graph transformation system* **G** *is the double category with finite horizontal colimits freely generated by* **G** *.*

However, there are some problems with this "definition".

1. What, exactly, is an DFHC–category? This is, first of all, of course a question of understanding and getting an intuition of this categorical definition. However, also going into details, there are different possibilities to extend the definition of colimits from ordinary categories to double categories. Thus it has to be clarified which definition of horizontal colimit is the right one in this case.
2. One major advantage of the structured transition system approach is that there is a free construction relating programs and their models of computations. Passing from categories with algebraic structure to DFHC–categories we have to make sure that the corresponding free constructions still exist. Moreover, we have to make precise what "generated by" means, i.e., the embedding of the graph transformation system into the DFHC–category.
3. Beyond the categorical (external) description of the model we need a concrete algebra of graph derivations in order to compare it with the original definitions and constructions in the DPO–approach. Since the definition of a DFHC–category is rather complex the question arises, how to practically check whether the algebra of graph derivations is indeed a DFHC–category.

As mentioned in the begining we used algebraic specification techniques to tackle these problems. To obtain a (compositional) definition of DFHC–categories *ten-*

*sor products* of specifications can be used ([Gra89,Cor90]). These are presentations of products of theories, and reflect the definition of double categories by internalisation. We will explain this technique briefly below. The solution to the second problem is given for free if parameterized algebraic specifications are used for all construction steps, since they yield free functors on their model categories that induce the pseudo free categorical constructions. Due to the stepwise definition and the compositionality properties of algebraic specifications also concrete models can be constructed and checked stepwise. Moreover, this structure of the model realizes the separation of concerns principle mentioned in the introduction.

Before coming to the concrete model construction a few words on double categories and internalisation are in order. A double category can be defined as *a category in* **Cat**, the category of categories. That means, a double category is given by a category of objects, a category of arrows, functors *src* and *tar* that assign source and target objects to arrows, and identity and composition functors. The category of objects is often called the *horizontal category*, and the objects of the category of arrows, with the composition induced by the composition functor, form the *vertical category*. The arrows of the category of arrows can then be naturally represented as *squares*.

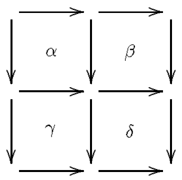
$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "u"'] \arrow[dr, phantom, "m"] & B \arrow[d, "v"] \\ C \arrow[r, "g"'] & D \end{tikzcd}$$

Here $f : A \to B$ and $g : C \to D$ are arrows of the horizontal category of objects, the vertical arrows $u : A \to C$ and $v : B \to D$ are objects in the category of arrows, with $src(u) = A$, $tar(u) = C$, $src(v) = B$, $tar(u) = D$, and the square $m$ is an arrow $m : u \to v$ in the category of arrows. Using a geographical metaphor $u$ and $v$ are called the west and the east border, and $f$ and $g$ the north and south border of $m$ respectively.

The theory of double categories is thus given by the product of the theory of categories with itself, and a specification $DCAT$ of double categories can be obtained by the tensor product of a specification $CAT$ with itself. Given partial equational specifications $SP_i = (S_i, OP_1, CE_i)$, $(i = 1, 2)$ their tensor product $SP_1 \otimes SP_2$ is given as follows. Its sorts are the cartesian product $S_1 \times S_2$ , its operations are given by the union of $OP_1 \times S_2$ and $S_1 \times OP_2$ . If $ob$ and $mor$ are the sorts of objects and arrows in $CAT$ this yields the sorts $(ob, ob)$ ,$(mor, ob)$ ,$(ob, mor)$ , and $(mor, mor)$ corresponding to objects, horizontal arrows, vertical arrows and squares of a double category respectively. Among the operations are $(comp, ob)$ , $(comp, mor)$ , $(ob, comp)$ , and $(mor, comp)$ , where $comp$ is the composition operation of $CAT$, corresponding to horizontal composition of horizontal arrows, horizontal composition of squares, vertical composition of vertical arrows, and vertical composition of squares respectively. We will denote both horizontal compositions by $\circ$ and both vertical compositions by $\bullet$ in the sequel. Finally the axioms of $SP_1 \otimes SP_2$ are given by the embeddings of the axioms $CE_1$ and $CE_2$ , and the *homomorphism equations*, that state the compatibility of the operations from $SP_1$ and $SP_2$ . For each pair $f : s_1 \dots s_n \to s \in OP_1$ and $g : s'_1 \dots s'_k \to s' \in OP_2$ the homomorphism property

$$\begin{aligned}&(f,s')((s_1,g)(x_{s_1 s'_1},\dots,x_{s_1 s'_k}),\dots,(s_n,g)(x_{s_n s'_1},\dots,x_{s_n s'_k})) = \\ &\quad = (s,g)((f,s'_1)(x_{s_1 s'_1},\dots,x_{s_n s'_1}),\dots,(f,s'_n)(x_{s_1 s'_k},\dots,x_{s_n s'_k}))\end{aligned}$$

is introduced, with a condition stating that both sides must be defined. Instantiating this equation with the composition operations for double categories we obtain the well known interchange law $(\delta \circ \gamma) \bullet (\beta \circ \alpha) = (\delta \bullet \beta) \circ (\gamma \bullet \alpha)$ for all composable quadruples of squares.




The tensor product allows to construct (and understand) the definition of DFHC–categories step by step, and supports the corresponding stepwise construction of the models, with stepwise correctness proofs. Moreover, partial equational specifications of programs and structured transition systems can be obtained as fragments of this specification, which yields the desired specification morphisms (inclusion) that induce the free functors.

To summarize the discussion so far, we end up with three levels of description of the model.

1. The first level is given by the *categorical description*, i.e. the fixation of the appropriate categorical structure (DFHC–categories) and the universal property of the free construction. It could be called the *semantical interface*, because it contains very compact descriptions via the universal properties and a terse charcterization of the overall structure, without showing the concrete constructions and its details.
2. The second level is given by the algebraic specification of the categorical structure. It can be considered as the *technical description*, because it is only used internally to construct and check the model and guarantee the existence of free constructions. Moreover, it yields a concrete syntax (the terms of the specification) to reason about the constructions.
3. The third level is given by the *concrete model*, i.e. the algebra of graph derivations in terms of concrete graphs, graph morphisms, derivations etc. This level is needed to establish the relation with the given notions of the DPO–approach, i.e. to be able to show that our model in fact realizes the notion of computation defined there.

## 5 The Algebra of Graph Derivations

In this section we now describe the development of the algebra of graph derivations within the structured transition system approach in more detail.

### 5.1 Programs

The entities corresponding to programs in the DPO–graph rewriting approach are graph transformation systems $\mathbf{G} = (P, \pi)$ . These are now represented as transition systems with algebraic structure on the states. An important observation at this point is that graph rewriting rules as well as direct derivations are not given by pairs of graphs alone, but by *graph spans.* These spans are necessary in order to express the connections between the concerned graphs. In a derivation the output is only determined up to isomorphism. Thus the preservation of identities through the state change must be made explicit by a relation between input and output. In a rule it is necessary to define a relation between its left hand side $L$ and its right hand side $R$ in order to prescribe the connection of $R$ to arbitrary input graphs to which $L$ might be matched. Both relations are given by graph spans. Thus the transitions will not relate graphs, but the graph morphisms the spans consist of. Anticipating the double category representation of the model of computations we represent the graph morphisms, that yield the states of the transition system, as horizontal arrows, and the transitions, representing the rewrite rules, as squares.

A rule $p : (L \leftarrow K \rightarrow R)$ of a graph transformation system $\mathbf{G}$ is then represented by a square $[p]$ as follows.

$$\begin{tikzcd} K \arrow[r, "l"] \arrow[d, "K"'] \arrow[dr, phantom, "{[p]}"] & L \arrow[d, "p"] \\ K \arrow[r, "r"'] & R \end{tikzcd}$$

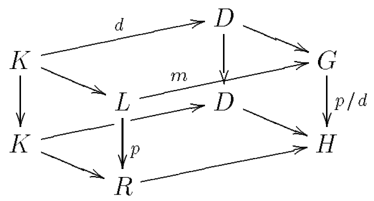
The morphisms $l : K \rightarrow L$ and $r : K \rightarrow R$ of the rule yield the north and south borders respectively of the square. Its west border is given by the identity morphism $K : K \rightarrow K$ . The east border is given by the rule name $p$ , considered as a new formal arrow between the graphs $L$ and $R$ . The whole square is denoted here by the rule name in square brackets, just to have a notational distinction between squares and (vertical formal) arrows. There is then a functional dependency from squares to their borders: The north border of $[p]$ is the first graph morphism of the graph span $\pi(p)$ etc. The program (in the STS sense) generated by $\mathbf{G}$ is thus given by the horizontal category of finite graphs, and the set of squares representing the rules of $\mathbf{G}$ .

### 5.2 Structured Transition Systems

The first step in the construction of the structured transition system from the program is to add idle steps for all states, that will yield the vertical identities in the double category in the next step. Thus we introduce squares as depicted on the left below for all graph morphisms $f : G \to H$. As for graphs the (vertical) identity on $f$ is denoted by the same letter.

$$
\begin{tikzcd}
G \arrow[r, "f"] \arrow[d, "G"'] \arrow[dr, phantom, "f"] & H \arrow[d, "H"] \\
G \arrow[r, "f"'] & H
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
L \arrow[d, "m"'] & K \arrow[l] \arrow[r] \arrow[d, "d"] & R \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}
$$

Now recall that a single step rewriting $p/m : G \Rightarrow H$ in the DPO–approach is given by the double pushout diagram given on the right above and the rule name $p$ . In the transition system we have already the left front square $[p]$ representing the rule $p : (L \leftarrow K \to R)$, and the vertical identity on $d : K \to D$ , back left.



Now the horizontal category of states has pushouts. Extending these pushouts to transitions, i.e., the squares, we can construct the following completion. The graph $G$ is the pushout object of $d : K \to D$ and $l : K \to L$ in the horizontal category of graphs, $H$ is the pushout of $d : K \to D$ and $r : K \to R$ . Thus these are the input and output of the derivation. Extending pushouts to squares this yields a pushout object of the squares $[p]$ and $d$ , a vertical arrow that we denote by $p/d : G \to H$ . The left pushout injection, denoted by $[p/d]$ , given by the square $G \leftarrow D \to D \to H$ represents the whole span $G \leftarrow D \to H$ .

Using coproducts of the horizontal category parallel rules are obtained, given by the coproduct of two squares representing rules. The pushout with a vertical identity $d$ of a context morphism $d : K + K' \to D$ then represents a parallel direct derivation $p + p'/m : G \Rightarrow H$ . Thus the structured transition system generated by $\mathbf{G}$ contains all parallel direct derivations of $\mathbf{G}$ as squares.

### 5.3 Model of Computations

Sequential composition of parallel graph derivations is obtained by the free construction of the double category with finite horizontal colimits generated by the structured transition system. Consider the sequential composition of (parallel) graph derivations $p/m : G \Rightarrow G'$ and $p'/m' : G' \Rightarrow H$ . The single steps are represented in the structured transition system by the upper right and lower right squares in the following diagram.

$$\begin{tikzcd}
D_0 \arrow[r] \arrow[dd] & D \arrow[r] \arrow[d] & G \arrow[dd] \\
& D \arrow[dr] & \\
D_0 \arrow[ur] \arrow[dr] \arrow[dd] & = & G' \arrow[dd] \\
& D' \arrow[ur] \arrow[d] & \\
D_0 \arrow[r] & D' \arrow[r] & H
\end{tikzcd}$$

Obviously these squares are not vertically composable, because the south border of the first one does not coincide with the north border of the second one. Constructing the pullback $D_0 \to D, D_0 \to D'$ of $D \to G'$ and $D' \to G'$ the single steps can be composed horizontally with the vertical identities on $D_0 \to D$ and $D_0 \to D'$ respectively. These yield now vertically composable squares $G' \leftarrow D_0 \to D_0 \to G$ and $H \leftarrow D_0 \to D_0 \to G'$ representing the sequential composition of the steps as defined in section 3.

### 5.4 Interchange Laws

Consider now a derivation $p/m : G_1 \Rightarrow H_1$ , represented by a square $[p/d]$ . Via a pushout with the vertical identity on a morphism $g_2 : D \to G_2$ it can be embedded in a larger context, and yields a derivation $p/e : G \Rightarrow G'$ . Suppose furthermore a square $[p'/d']$ representing a derivation $p'/m' : G_2 \Rightarrow H_2$ and its extension by a pushout with a vertical identity on $h_1 : D \to H_1$ , represented by the vertical arrow $p'/e' : G' \to H$ . These two steps overlap only in the common context $D$ that both leave unchanged, thus they are sequentially independent (cf. section 3).

$$\begin{tikzcd}
G_2 \arrow[d] \arrow[r, phantom, "g_2"] & D \arrow[l] \arrow[r] \arrow[d] \arrow[r, phantom, "{[p/d]}"] & G_1 \arrow[d] & G \arrow[d, "p/e"] \\
G_2 \arrow[d] \arrow[r, phantom, "{[p'/d']}"] & D \arrow[l] \arrow[r] \arrow[d] \arrow[r, phantom, "h_1"] & H_1 \arrow[d] & G' \arrow[d, "p'/e'"] \\
H_2 & D \arrow[l] \arrow[r] & H_1 & H
\end{tikzcd}$$

$$\begin{tikzcd}
G_2 \arrow[d] \arrow[r, phantom, "{[p'/d']}"] & D \arrow[l] \arrow[r] \arrow[d] \arrow[r, phantom, "{[p/d]}"] & G_1 \arrow[d] & G \arrow[d, "p+p'/f"] \\
H_2 & D \arrow[l] \arrow[r] & H_1 & H
\end{tikzcd}$$

Since $g_2$ and $h_1$ are vertical identities vertical composition yields $[p'/d'] \bullet g_2 = [p'/d']$ and $h_1 \bullet [p/d] = [p/d]$ . Their pushout is the vertical arrow $p + p'/f : G \to H$ . Vice versa, computing first the horizontal pushouts in the upper diagram yields the vertical composition $p'/e' \bullet p/e : G \to H$ . The corresponding interchange law states that the order of the operations (pushout and composition) can be exchanged, i.e. $p + p'/f = p'/e' \bullet p/e$ .

The pushout decomposition of the input graph $G$ into local graphs $G_1$ and $G_2$ in the upper diagram corresponds exactly to the decomposition that is used to prove the parallelism theorem of DPO–graph rewriting. Using this idea it can be shown that the representations of shift equivalent derivations as vertical arrows are identified by the interchange laws. Moreover, the axioms of DFHC–categories are faithful w.r.t. this representation, i.e., non–equivalent derivations are represented by distinct vertical arrows.

## 6 Conclusion

We have presented the application of the STS–method for the definition of models of computations to DPO–graph rewriting. Its aim has been the construction of an algebra of graph derivations as a descriptive model of the semantics of graph transformation systems according to the DPO–aproach. The main point in this application is to find the right algebraic structure of graphs, that model the states in this case. The model of computations is then fully determined. The criterion for the *right* structure is thereby to check, whether the so defined model of computations covers the semantics, including the notion of behavioural equivalence, that had been defined in the DPO–approach before. Since the shift equivalence classes of a graph transformation system $\mathbf{G}$ are faithfully represented by the squares of the model of computations generated by $\mathbf{G}$, and all squares have an interpretation in terms of the semantics of $\mathbf{G}$, this criterion is fulfilled.

There are several advantages of constructing the algebra of a graph derivations as a model of computations according to the STS–method. First all advantages of the STS–method are inherited (see section 2). Second we obtain in this way an inductive definition not only of graphs, but also of graph derivations and shift equivalence classes, due to the free construction of the model of computations. Finally, the free construction also yields a very compact characterization of the semantics via the universal property. Basically it is given by the slogan in section 4.

An algebra of graph derivations as semantical model has several advantages. As already discussed above, it can be used to reason about the distribution of states and computations, due to the operations of the algebra. Moreover, it can be used to define refinement or implementation relations between graph transformation systems. These would be given categorically be Kleisli–morphisms, which means that rules of one system can be mapped to derivations of the other one, i.e., the rules are refined/implemented by sequential and parallel compositions and/or instances of the rules of the more concrete system.

The application of the STS–method to DPO–graph rewriting showed moreover, that the states of a graph derivation are graphs with interfaces (i.e. graph morphisms) rather than graphs, which results in a double category with algebraic structure, induced by finite colimits, as model of computations. This yields a certain complexity of the model. For that reason the development of the algebra of graph derivations required a careful reflection on the methods we could use to reach our goal. As documented here algebraic development techniques could be applied successfully. Although the final model need not necessarily be presented in this algebraic framework, the use of algebraic techniques in the development has been necessary in order to obtain the right definitions at all. Secondly the use of double categories could be a starting point for a comparison with the tile model as general model of computations, that is also based on double categories and obviously had a great impact on the work presented here. This comparison, however, is outside the scope of this paper.

## References


[BC87] M. Bauderon and B. Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20:83–127, 1987.

[CA93] A. Corradini and A. Asperti. A categorical model for logic programs: Indexed monoidal categories. In *Proceedings REX Workshop, Beekbergen, The Netherlands, June 1992*, Springer LNCS 666, 1993.

[CG97] A. Corradini and F. Gadducci. A 2-categorical presentation of term graph rewriting. In *Proceedings CTCS'97*, Springer LNCS 1290, 1997.

[CGW95] I. Claßen, M. Große-Rhode, and U. Wolter. Categorical concepts for parameterized partial specifications. *MSCS*, 5(2):153–188, 1995.

[CM92] A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its application to logic programs. *TCS*, 103:51–106, 1992.

[Cor90] A. Corradini. *An algebraic semantics for transition systems and logic programming*. PhD thesis, Dip. Informatica, Università di Pisa, 1990.

[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.

[EV97] J. Engelfriet and J.J. Vereijken. Context-free graph grammars and concatenation of graphs. *Acta Informatica*, 34:773–803, 1997.

[GH98] F. Gadducci and R. Heckel. An inductive view of graph transformation. In *Proceedings WADT'97*, Springer LNCS 1376, 1998.

[Gog91] J.A. Goguen. A categorical manifesto. *MSCS*, 1, 1991.

[Gra89] J.W. Gray. The category of sketches as a model for algebraic semantics. *Contemporary Mathematics*, 92:109–135, 1989.

[Hec98] R. Heckel. *Compositional Development of Reactive Systems using Graph Transformation Systems with Loose Semantics*. PhD thesis, TU Berlin, 1998.

[Law63] F. W. Lawvere. Functorial semantics of algebraic theories. In *Proc. National Academy of Science, U.S.A., 50*, pages 869–872. Columbia University, 1963.

[MM90] J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Computation*, 88:105–155, 1990.

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

[UG71] F. Ulmer and P. Gabriel. *Lokal präsentierbare Kategorien*, Springer LNM 221, 1971.