# Strategies in Modular System Design by Interface Rewriting

S. Cicerone and F. Parisi Presicce*

Dipartimento di Matematica Pura ed Applicata,
Università de L'Aquila, I-67010 Coppito (AQ), Italy
e-mail: {cicerone, parisi}@vxscaq.aquila.infn.it

**Abstract.** The problem of designing a modular system, using a set of predefined modules, with a given import and export interface has been reduced to the problem of generating a specification in an algebraic specification grammar. Here we tackle two important problems connected with the generation: the strategy to adopt in choosing the rewrite rules and the elimination of unnecessary searches. The first is investigated using a notion of similarity of specifications and a definition of value to guide the search algorithm; the second is solved using syntactical criteria (independent of the target specification) to determine that some derivation sequences are superfluous. The latter development has been influenced by similar work on graph grammars.

## 1 Introduction

The development of large correct software systems is very difficult without the appropriate support of notions such as modularization and interconnection of components [16,11,10]. In our context, a module specification [10,1,7] consists of four parts: a parameter part PAR to model genericity and parametrization (as in Ada generics, for example); an import part IMP (containing PAR) describing what the module needs from other modules (modelling a "virtual" module to be specified at a later time); an export interface EXP (containing PAR) specifying what part of the implemented functions are visible from the outside; and a body part BOD (containing all the others) with the description of how the functionalities exported (EXP) are implemented using those imported. Interconnection mechanisms for the horizontal structuring of systems are crucial for the stepwise development of large software in a flexible manner [7]. Interpreting the interconnections as operations on module specifications [1] it is easy to give a semantics to the main ones: *union* performed componentwise by specifying the common subcomponent to be identified; *composition* where the import of one module is matched with the export of another module; and *actualization* where the parameter part is replaced by an actual specification

Module specifications designed and verified can be used via their interfaces, the only parts visible from the outside. A common problem is that of designing an interconnection of a predefined set of module specifications (of a library, for example) which realizes a given overall export interface from another given import interface. In [13,14] this problem has been addressed by viewing the visi-

* Current address: Dip. Scienze dell'Informazione , Univ. Roma "La Sapienza", via Salaria 113, I-00198 ROMA - Italy

ble part (PAR,IMP,EXP) of a module specification as a production of an algebraic specification grammar (ASG) [8], an extension of the algebraic theory of graph grammars [4] to structures other than graphs. In this approach, the applicability of a production $(IMP \leftarrow PAR \rightarrow EXP)$ to a specification $SPEC$ to obtain a new specification $SPEC'$ indicates the existence of a module specification, obtained from the one which realizes the production, which has $SPEC$ and $SPEC'$ as import and export interface, respectively. A derivation sequence $PRE \Rightarrow SPEC_1 \cdots \Rightarrow SPEC_n \Rightarrow GOAL$ can be automatically translated into the appropriate interconnection of the modules realizing the interfaces used as productions.

In general, given a specification $SPEC$ and a set $LIB$ of productions, there may be several applicable productions, each with more than one occurrence of the left hand side in $SPEC$. The combinatorial explosion of possible sequences of derivations could be contained by analyzing beforehand the productions to remove from the search tree any path which will produce specifications already generated. This reduction is addressed in section 4, where syntactical criteria are given to predict the applicability of a rule after a derivation which uses another rule, and to avoid a derivation sequence which is equivalent to another derivation produced with a different order of the same productions. Many definitions and some results in this section are inspired by [12].

Having somewhat reduced the search tree, it is still necessary to have some criteria to choose (at least temporarily, trying to avoid backtracking) which production to use and which occurrence to apply. This is the topic of section 3, where we introduce the notion of *similarity* between two specifications and use it as a guide in selecting the appropriate occurrence. The search algorithm exploits the symmetry of the notion of direct derivation to develop a strategy based on the application of deductive (i.e., strict growth) rules in a forward fashion, and of deleting (i.e., strict reduction) rules in a backward fashion, interleaved with applications of the remaining productions.

The discussion in this paper is based on a standard notion of algebraic specification and a particular choice of specification morphism, needed to exploit some results in [5]. Some of the results can be extended to other situations, a few to a framework based on arbitrary institutions. All proofs, for lack of space, are omitted and can be found in [2].

## 2 Notation and Background

In this section we briefly review some basic notions of algebraic specifications ([6]) and of algebraic specification grammars ([13,14]).

**Specifications**

An algebraic specification $(\Sigma, E)$ constists of a many sorted signature $\Sigma = (S, OP)$, and a set $E$ of (positive conditional) equations. The three parts of a specification $SPEC = (S, OP, E)$ are referred to by using a subsript $S_{SPEC}$, $OP_{SPEC}$ and $E_{SPEC}$. If $N \in OP$, $dom(N) \in S^*$ denotes the domain sorts and $cod(N) \in S$ the codomain sort of $N$. For $s \in S$, $N \in OP$, $e \in E$ we use:

- $SORT(N)$ as the union of $dom(N)$ and $cod(N)$;

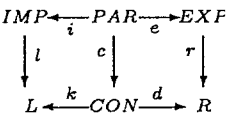
- $OPNS(s)$ as the subset of $OP$ containing the operations $N$ with $s \in SORT(N)$;
- $OPNS(e)$ as the operations in the terms of $e$;
- $EQNS(N)$ as the subset of $E$ that contains equations $e$ with $N \in OPNS(e)$

The notion of specification morphism $f : (\Sigma_1, E_2) \rightarrow (\Sigma_2, E_2)$ as a triple $(f_S, f_{OP}, f_E)$ based on the accepted definition of signature morphism $f_\Sigma : \Sigma_1 \rightarrow \Sigma_2$, assumes that the equations of $E$ are labelled, and different labels $eq_i$ may correspond to the same triple $(X, t_1, t_2)$ representing the equation $eq_i : t_1 = t_2$; for $(eq_i : t_1 = t_2) \in E_1$ we have $(f_E(eq_i) : f^\#(t_1) = f^\#(t_2)) \in E_2$. We write *Specification* to denote the set of all specifications.

**Specification Grammars**

In the well known algebraic approach to Graph Grammars [4] it is possible to replace the category of graphs by the category of some other structure, giving rise to a new rewriting theory for high level structures [5].

Among those, a **H**igh **L**evel **R**eplacement (HLR) system was introduced in [13] in order to generate algebraic specifications using productions and derivations. An *algebraic specification production*, shortly SPEC-production, is an ordered pair $Pro = (IMP \leftarrow PAR \rightarrow EXP)$ of injective specification morphisms




$i : PAR \rightarrow IMP$ and $e : PAR \rightarrow EXP$. A *direct derivation* consists of the two pushout diagrams of specifications. A production $Pro$ is *applicable* to a specification $L$ if there exist a morphism $l : IMP \rightarrow L$ and a *context* specification $CON$ such that $L$ is the pushout of $IMP$ and $CON$. The result $R$ of the derivation is pushout of $EXP$ and $CON$. In this case we have the *direct derivation* $Pro : L \Rightarrow R$ (or $L \overset{Pro}{\Rightarrow} R$), and we say that $R$ is *derivable* from $L$ via $Pro$. Notice that a direct derivation is symmetric and that if $Pro : L \Rightarrow R$, then $Pro^{-1} : R \Rightarrow L$ where $Pro^{-1} = (EXP \leftarrow PAR \rightarrow IMP)$. If $PROD$ is a set of SPEC-productions then the set of all the symmetric productions is denoted by $PROD^{-1}$. The specification morphisms that guarantee the existence of the pushout complement $CON$ in a direct derivation are called *occurrence morphisms*. A morphism $l : IMP \rightarrow L$ is an occurrence if the following *Gluing Conditions* hold:

a. $ID_S \cup DANG_S \subseteq i_S(S_{PAR})$

b. $ID_{OP} \cup DANG_{OP} \subseteq i_{OP}(OP_{PAR})$, where

- $ID_S = \{s \in S_{IMP} \mid \exists s' \in S_{IMP},\ s \neq s',\ l_S(s) = l_S(s')\}$;
- $ID_{OP} = \{N \in OP_{IMP} \mid \exists N' \in OP_{IMP},\ N \neq N',\ l_{OP}(N) = l_{OP}(N')\}$;
- $DANG_S = \{s \in S_{IMP} \mid \exists N \in OP_L \setminus l_{OP}(OP_{IMP})\ and\ l_S(s) \in SORTS(N)\}$;
- $DANG_{OP} = \{N \in OP_{IMP} \mid \exists e \in E_L \setminus l_E(E_{IMP})\ and\ l_{OP}(N) \in OPNS(e)\}$.

The occurrence morphism $l : IMP \rightarrow L$ identifies in the direct derivation $Pro : L \Rightarrow R$ the $L$-part $DEL_l = l(IMP \setminus i(PAR))$, removed from $L$ by the production $Pro$, and the $R$-part $INS_l = r(EXP \setminus e(PAR))$, glued by $Pro$ to the context $CON$ to realize the new specification $R$. Given a set $PROD$ of SPEC-productions we write $SPEC_1 \overset{*}{\underset{PROD}{\Rightarrow}} SPEC_n$ to mean a sequence of $n \geq 0$ direct derivations $SPEC_1 \overset{P_1}{\Rightarrow} SPEC_2 \overset{P_2}{\Rightarrow} \cdots \overset{P_n}{\Rightarrow} SPEC_{n+1}$ with $P_i \in PROD$ for $i = 1, \ldots, n$.

## 3 Strategy

The following notions are inspired by [12], where Graph Grammars are considered as models for rule-based systems in which solving state-space problems essentially requires searching in an exploding number of generated states which cannot be managed. The usual answer to this problem in AI is to prune the search-tree, selecting only some of the possible expansions of derivations. Any HLR system can be used to specify in a formal way many other similar problems. Algebraic specification grammars can model problems in which a transformation of a specification $PRE$ by the rules in a library $LIB$ to obtain a prefixed final specification $GOAL$, is required.

**Definition 1.** a). An *(AST-)problem* $P = (Ax, R, F)$ consists of a specification $Ax$, called axiom, a family of production rules $R$, and a unary predicate function on specifications $F$ called filter. A *solution* for $P$ is any $Ax$-reachable specification $SPEC$, i.e. $Ax \overset{*}{\underset{P}{\Rightarrow}} SPEC$, for which $F(SPEC)$ is true.

b). Given a class $PC = (\mathcal{AX}, R, F)$ of AST-problems, $P = (Ax, R, F)$ with $Ax \in \mathcal{AX}$, an algorithm $S$ is called a *search algorithm* (w.r.t. $PC$), if and only if, when supplied with $Ax \in \mathcal{AX}$, the algorithm terminates with either a specification or the message `FAIL`. The result of $S$ is correct if it is indeed a solution of $P$ or `FAIL` otherwise.

c). A *production-system* $(PC, S)$ for AST-problems, briefly an AST-PS, consists of a class of AST-problems $PC$ and a search algorithm $S$.

According to the basic execution model of production systems, derivations of a specification $SPEC$ based on a production rule $Pro$ consists of two steps: retrieving the informations of how to apply $Pro$ to $SPEC$, which can then be used to derive the new specification $SPEC'$ from $SPEC$. In terms of AST notions, this corresponds to two primitive functions:

`Recognize` : $Specification \times Rule \rightarrow OccurrenceSet$
where `Recognize`$(SPEC, IMP \leftarrow PAR \rightarrow EXP) = \{g_1, g_2, \ldots, g_m\}$ is the set of all occurrences $g_i : IMP \rightarrow SPEC$.

`Derive` : $Specification \times Occurrence \rightarrow Specification$
where `Derive`$(SPEC, g) = SPEC'$ with $SPEC \overset{g}{\Rightarrow} SPEC'$, assuming that an 'occurrence' carries the information about the corresponding rule.

Difficult problems arise in executing each of the two operations above, as well as in evaluating the filter $F$ on specifications. The cost of the latter depends on the specific problem; the cost of the operations other than `Recognize` and `Derive` is ignored. The following notion of cost covers both, the problem of a search-space-reduction, and the efforts to determine the applicability of rules.

**Definition 2.** Given an AST-PS=$(PC, S)$, the cost of the search algorithm $S$ is based on the primitive function `Recognize` and `Derive`; it is defined to be $(N_1, N_2)$ with $N_1$ and $N_2$ being the number of calls for those functions. The cost $(N_1, N_2)$ is said to be no greater than $(N_1', N_2')$ iff $N_i \leq N_i'$ for $i = 1, 2$.

Since we do not want to adopt a specific search algorithm, we aim for a notion of optimization which guarantees that every search algorithm can be improved.

**Definition 3.** Given an AST-PS=$(PC, S)$, and a search algoritm $O$ w.r.t. $PC$, we call $O$ an optimization w.r.t. $S$ iff $O$ yields the correct solution whenever $S$ does and the cost of $O$ is not greater than the cost of $S$.

The formalization can be given by the AST-problem $P = (PRE, LIB, F)$, where $PRE$ is the predefined data type, $LIB$ is the library of reusable modules defining the transformation rules, $F$ the filter defined by:
- $F(SPEC) = true$ iff $SPEC = GOAL$
An analysis of $P$ could produce a search algorithm for $PC$ (def. 1) independent of the initial axioms $PRE$. From the definition of $P$ we observe that there is only one $PRE$-reachable specification that is a solution for $P$ itself: a potential search algorithm for $P$ should select in the search-tree a path from $PRE$ to $GOAL$. If $LIB$ has reasonable dimensions, it is impossible to think of selecting such a path visiting the tree in an exaustive way; it is enough to observe that not only there exist several ways to transform each specification, but each rule in its own can generate many different results using all the occurrences selected by the `Recognize` primitive. For this reason it is difficult to work with search algorithms that exploit a backtracking going back for more than one level. Then, for each step the algorithm should check all the transformations, and then go down toward the specification that promises a 'better result'. It is necessary to find some criteria to assign a value showing the capability of each specification to lead to the final $GOAL$ specification. To this end we can assign a value to a specification according to the number of elements shared by $GOAL$.

### 3.1 Similarity

**Definition 4.** a) Let $SPEC_1 = (S_1, OP_1, E_1)$ and $SPEC_2 = (S_2, OP_2, E_2)$ be algebraic specifications and let $(S, OP, E)$ be a subspecification of $SPEC_1$ such that $S \subseteq S_1$, $\emptyset \neq OP \subseteq OP_1$, $E \subseteq E_1$. A specification morphism $f : (S, OP, E) \rightarrow SPEC_2$ having injective components $f_S : S \rightarrow S_2$, $f_{OP} : OP \rightarrow OP_2$ and $f_E : E \rightarrow E_2$, is called *sharing morphism* of $SPEC_1$ into $SPEC_2$.

b) In this case we call $SPEC_1$ *Partially Similar* to $SPEC_2$ and denote it by $SPEC_1 \overset{f}{\leq} SPEC_2$.

c) If the components of $f$ are also surjective, we call $SPEC_1$ *Totally Similar* to $SPEC_2$ and denote it by $SPEC_1 \overset{f}{=} SPEC_2$.

d) The specification $SPEC_1$ is called *Comparison Specification* and $SPEC_2$ is called *Target Specification*. The subspecification $(S, OP, E)$ and the elements in $SPEC_1 \setminus (S, OP, E)$ are denoted by $Core(SPEC_1)$ and $Remainder(SPEC_1)$ respectively, via the sharing morphism $f : (S, OP, E) \rightarrow SPEC_2$.

The set $M = \{f | SPEC_1 \overset{f}{\leq} SPEC_2\}$ containing the sharing morphisms of $SPEC_1$ into $SPEC_2$ may have more than one element. To chose one we associate to the Target Specification $SPEC_2 = (S_2, OP_2, E_2)$, a mapping $c = (c_S, c_{OP}, c_E)$ that weighs sorts, operations and equations with $c_S : S_2 \rightarrow \mathbf{N}$, $c_{OP} : OP_2 \rightarrow \mathbf{N}$, $c_E : E_2 \rightarrow \mathbf{N}$

and define a function $val : M \rightarrow \mathbf{N}$ that evaluates the resources that the Comparison and the Target specifications share. For $f : (S, OP, E) \rightarrow SPEC_2 \in M$,

$$val(f) = \sum_{s \in S} c_S(f_S(s)) + \sum_{N \in OP} c_{OP}(f_{OP}(N)) + \sum_{e \in E} c_E(f_E(e))$$

is the *comparison value* of $f$. Among all the elements in $M$ with the higher value, an arbitrary choice can selects the one we are looking for. If we choose $f^*$, it is called *main sharing morphism*, and the value $v(SPEC_1, SPEC_2, c) = val(f^*)$ denotes a *measure* of the similarity via some mapping $c$.

**Definition 5.** Let $SPEC_2$ be a Target specification with a weight mapping $c$.

**a.** the *Total Weight* of the Target is the value

$$p_T(SPEC_2) = \sum_{s \in S_2} c_S(s) + \sum_{N \in OP_2} c_{OP}(N) + \sum_{e \in E_2} c_E(e)$$

**b.** the function $Ratio_{SPEC_2,c} : Specification \rightarrow [0, 1]$ is given by

$$Ratio_{SPEC_2,c}(SPEC_1) = \frac{v(SPEC_1, SPEC_2, c)}{p_T(SPEC_2)}$$

where $SPEC_1$ denote a Comparison specification.

**c.** the specification $SPEC_*$ is the Optimal Comparison specification in $I \subset Specification$ w.r.t. the Target $SPEC_2$, when:
$Ratio_{SPEC_2,c}(SPEC^*) = max\{Ratio_{SPEC_2,c}(SPEC) | SPEC \in I\}$

**Fact 1.** *Let $SPEC_1$ be a Comparison specification and let $SPEC_2$ be a Target with a weight mapping $c = (c_S, c_{OP}, c_E)$. If $M = \{f | SPEC_1 \stackrel{f}{\simeq} SPEC_2\}$ then*

$$Ratio_{SPEC_2,c}(SPEC_1) = 1 \iff \exists\, f \in M :\ SPEC_1 \stackrel{f}{=} SPEC_2$$

The function *Ratio* could be the basis for an extension to a formal definition of a 'metric' that allows to measures the distance between specifications.

*Example 1.* . Let us suppose the following is a *Target Specification* in such a comparison between specifications with a mapping that uniformly weighs all the elements. It can be viewed as a specification of a system that represents a queue (FIFO), with some length, of weighted elements.

*QueueNat* =

sorts queue, nat, bool

opns $NEW$: → queue
$QADD$: queue nat → queue
$REMOVE$: queue → queue
$LENGTH$: queue → nat
$IS\text{-}EMPTY$: queue → bool
$ZERO$: → nat
$SUCC$: nat → nat
$TRUE$: → bool
$FALSE$: → bool

eqns For $q \in queue$, $n \in nat$,
$e_1 : REMOVE(NEW) = NEW$
$e_2 : REMOVE(QADD(q, n)) =$ IF $IS\text{-}EMPTY(q)$ THEN $NEW$ ELSE $QADD(REMOVE(q), n)$
$e_3 : IS\text{-}EMPTY(NEW) = TRUE$
$e_4 : IS\text{-}EMPTY(QADD(q, n)) = FALSE$
$e_5 : LENGTH(NEW) = ZERO$
$e_6 : LENGTH(QADD(q, n)) = SUCC(LENGTH(q))$

The following are SPEC-productions defined by two module specifications. The first is $P_{MN}$ = (MN-Imp $\xleftarrow{i_1}$ MN-Par $\xrightarrow{e_1}$ MN-Exp), where the three specifications are:

MN-Imp = MN-Par +
sorts ∅
opns *SUCC*: nat → nat
eqns ∅

MN-Par =
sorts nat
opns *ZERO*: → nat
eqns ∅

MN-Exp = MN-Imp +
sorts ∅
opns *MUL*: nat nat → nat
eqns ∅

while the second is $P_{IS}$ = (IS-Imp $\xleftarrow{i_2}$ IS-Par $\xrightarrow{e_2}$ IS-Exp), with the specifications:

IS-Imp = IS-Par+
sorts string
opns *NIL*: → string
*LADD*: data string → string
eqns ∅

IS-Par =
sorts data
opns ∅
eqns ∅

IS-Exp = IS-Imp+
sorts ∅
opns *INVERT*: string → string
eqns ∅

Both the productions have inclusions as specification morphisms. Applying the productions $P_{MN}$ and $P_{IS}$ to the specification *StackInt* we obtain the direct derivations $P_{MN} : StackInt \Rightarrow StackInt'$ and $P_{IS} : StackInt \Rightarrow InvertingStack$ respectively.

*StackInt* =
sorts stack,int
opns *ZERO*: → int
*SUCC*: int → int
*PRED*: int → int
*EMPTY*: → stack
*PUSH*: stack int → stack
eqns ∅

*InvertingStack* = *StackInt*+
sorts ∅
opns *INVERT*: stack → stack
eqns ∅

*StackInt'* = *StackInt* +
sorts ∅
opns *MUL*: int int → int
eqns ∅

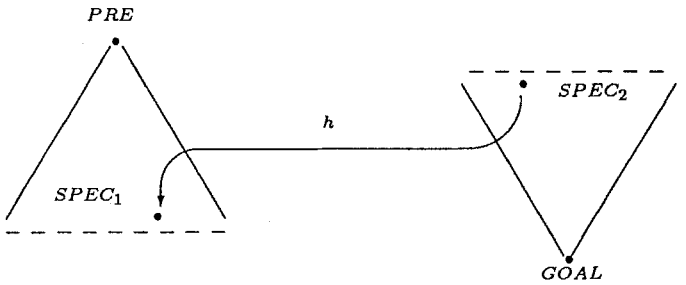
Since $v(StackInt', QueueNat, 1) = 6/18$ and $v(InvertingStack, QueueNat, 1) = 7/18$, the production $P_{IS}$ modifies *StackInt* making it more *similar* to the *target* than the result of the application $P_{MN} : StackInt \Rightarrow StackInt'$.

### 3.2 Search algorithm

In the context of similarity, the solution for $P$ coincides with the Optimal Comparison specification in the set of all the specifications generated by the grammar $(PRE, LIB, \overset{*}{\underset{LIB}{\Rightarrow}})$ w.r.t. the target $GOAL$, when it is $GOAL$ exactly. But the Optimal Comparison specification $SPEC^*$ could be different from $GOAL$, either because of the search algorithm or because of a small library. In any case $SPEC^*$ may be used in adapting the design of a partially designed modular system: instead of constructing a module with $PRE$ and $GOAL$ as interfaces, we need to implement only the elements that $GOAL$ does not share with $SPEC^*$. We could also use the target weight mapping to increase the probability that a particulary element could be in $SPEC^*$. We now give another AST-problem that is a formalization for our original problem.

**Definition 6.** Define $P'$ as the AST-problem $P' = (PRE, LIB, F')$, where $PRE$, $LIB$ and $GC$ are the same of $P$, whereas $F'$ is defined as:
$F'(SPEC_1) = true$ iff $\exists h : SPEC_2 \rightarrow SPEC_1 \ \wedge \ SPEC_2 \in (GOAL, LIB^{-1}, \overset{*}{\underset{LIB^{-1}}{\Rightarrow}})$
The symmetric AST-problem of $P'$ is $P'' = (GOAL, LIB^{-1}, F'')$, where
$F''(SPEC_2) = true$ iff $\exists h : SPEC_2 \rightarrow SPEC_1 \ \wedge \ SPEC_1 \in (PRE, LIB, \overset{*}{\underset{LIB}{\Rightarrow}})$.

This new problem is important because it can be solved even if $GOAL$ cannot be derived from $PRE$. In fact, if $F'(SPEC_1) = true$ for some $SPEC_1 \in (PRE, LIB, \overset{*}{\underset{LIB}{\Rightarrow}})$, the situation can be represented as in the following picture:




with the search tree for the problem $P'$ (the specification generated by the grammar $(PRE, LIB, \overset{*}{\underset{LIB}{\Rightarrow}})$), the tree that represents the specifications (including $SPEC_2$) generated by the grammar $(GOAL, LIB^{-1}, \overset{*}{\underset{LIB^{-1}}{\Rightarrow}})$ and the linking morphism $h$ between the trees. By theorem 4.7 in [14], the derivation $PRE \overset{*}{\underset{LIB}{\Rightarrow}} SPEC_1$ defines a module $MOD_1$, whereas the derivation $GOAL \overset{*}{\underset{LIB^{-1}}{\Rightarrow}} SPEC_2$, considered in a symmetric way, defines a module $MOD_2$. The linking morphism $h$ makes $MOD_2$ a client of $MOD_1$, and allows their composition to obtain a module $MOD$ that is a solution of the original problem.
The properties of $P'$ are now studied to define a strategy on which a search algorithm for $P'$ can be based. An initial measure of the difficulty to transform $PRE$ into $GOAL$ can be given in a way independent of the knowledge contained in the library. The value $d(PRE, GOAL) = 1 - Ratio_{GOAL,1}$ represents the percentage of the lacking resources of $PRE$ with respect to the ones in $GOAL$. There exist some ordinary productions that allow 'to deduct' both the existence of further implicit resources in $PRE$ and a surplus of elements in $GOAL$.

**Definition 7.** Let $Pro = (IMP \leftarrow PAR \rightarrow EXP)$ be a rule in $LIB$ and let $SPEC$ be a specification.

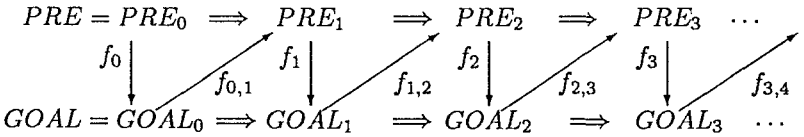
i. $App(SPEC) = \{Pro \in LIB \mid \texttt{Recognize}(SPEC, Pro) \neq \emptyset\}$
ii. $Ded(SPEC) = \{Pro \in App(SPEC) \mid PAR = IMP \neq EXP\}$

The deductive productions in the set $Ded(SPEC)$, when applied to $SPEC$ via an occurrence $l$, yield $DEL_l = \emptyset$ in forward mode and $INS_l = \emptyset$ in backward mode. So

it is possible to use a search algorithm starting with a *canonical phase* that tries to reduce the difficulty of transforming $PRE$ into $GOAL$ by means of the deductive rules of $LIB$. In fact, by these rules, a forward derivation from $PRE$ leads to an enriched specification $PRE'$, and a backward derivation from $GOAL$ to a final specification $GOAL'$ with a minimum number of elements to be implemented. An appropriate use of deductive rules assures $d(PRE', GOAL') \leq d(PRE, GOAL)$. Using the concepts of similarity we can now define the strategy for a first phase in a search algorithm for $P'$. During the enrichment of $PRE$ (*growing phase*), if we apply the rules $Pro \in Ded(SPEC_i)$ to $SPEC_i$ via the occurrence $l \in \texttt{Recognize}(SPEC_i, Pro)$, giving rise to the direct derivation $SPEC_i \Rightarrow SPEC_{i+1}$, we can consider the difference

$$Info(Pro, l, SPEC_i) = v(SPEC_{i+1}, Target, c) - v(SPEC_i, Target, c)$$

This value represents the increase of the information that $SPEC_i$ shares with target, carried by the deductive $Pro$. As target, we can assume the final specification $GOAL$ or some other, as we will see later. For each step we choose the deductive one that carries the greatest increase; if no one can bring a positive increase, the growing phase stops.Analogously, during the *shrinking phase* from the final specification $GOAL$, at each step we can apply the deductive one that cuts the greatest number of elements in the remainder of $GOAL$ via some sharing morphism w.r.t. $PRE$ (or w.r.t. the result $PRE'$ of the growing phase). Since both growing and shrinking phases require calls to primitive `Recognize` and `Derive` in a proportional way to $d(PRE, GOAL)$, choosing $GOAL'$ as target for the comparisons in the forward derivation and also choosing $PRE'$ as target for the ones in backward derivation, reduces the cost of the algorithm. Hence we prefer to adopt an *interaction* between forward and backward derivations.




In the above diagram the main sharing morphisms between the 'current' and target derivation for each derivation step are represented:

- $f_i : PRE_i \rightarrow GOAL_i$, identifying the resources that $PRE_i$ shares with the target $GOAL_i$ before deriving $PRE_{i+1}$.
- $f_{i-1,i} : GOAL_{i-1} \rightarrow PRE_i$, identifying the resources that $GOAL_{i-1}$ shares with the target $PRE_i$ before deriving $GOAL_i$.

The choice of starting the interactive derivation from $PRE_0$ is due to an immediate decrease of calls to primitive functions, rather than from $GOAL_0$, that produces benefits only on the length of forward derivation $PRE \Rightarrow PRE'$. However, for each step, the existence of a linking morphism can be tested, requiring the valuation of the filter of problem $P'$.
The following fact takes into consideration the cost of the filter valuation, allowing to focus on the current specifications.

**Proposition 8.** *If $PRE_i$ is the current derived specification in the canonical phase of a forward derivation, then:*

$$\nexists h : GOAL_{i-1} \rightarrow PRE_i \;\Rightarrow\; \nexists h' : GOAL_k \rightarrow PRE_j \;\forall k \leq i-1,\; \forall j \leq i$$

*If $GOAL_i$ is the current derived specification in the canonical phase of a backward derivation, then:*

$$\nexists h : GOAL_i \rightarrow PRE_i \;\Rightarrow\; \nexists h' : GOAL_k \rightarrow PRE_j \;\forall k \leq i,\; \forall j \leq i$$

Any search algorithm for the problem $P'$ (see def. 6) should have a Canonical Phase to construct its first part. At this point it is also necessary to take into consideration all the productions in $LIB$ to define the strategy for deriving $GOAL'$ from $PRE'$. This post-canonization phase can be based on an interactive derivation again, but we need to modify the way to choose the rule to apply. A procedure can select among all the rules in $App(PRE')$, but not in $Ded(PRE')$, the production that causes the largest increase of shared resources with respect to the current target, in spite of the removed part $DEL_l \neq \emptyset$. If $DEL_l$ contains some 'useful' elements, a new series of applications of deductive rules takes place, while another procedurecan select among all the rules in $App(GOAL')$, but not in $Ded(GOAL')$, the symmetric production that causes the largest decrease of the elements not shared (those in the Remainder) with respect to the current target, in spite of the removed part $INS_l \neq \emptyset$. If $INS_l$ contains some 'useless' elements, a new series of applications of deductive rules takes place.

## 4 Optimization

In this section we consider again the results of [12] in the theory of Graph Grammars, and present criteria that allow to optimize any search algorithm in the context of algebraic specification transformations. Our approach to optimization is based on properties of rules which must safely avoid calls to the corresponding primitive functions, thus reducing the cost of the algorithm.
The derivation bag $P_J(\mathcal{S})$ of a specification bag $\mathcal{S}$ with respect to a family of rules $(P_j)_{j \in J}$ is $P_J(\mathcal{S}) = \{SPEC' \mid \exists SPEC \in \mathcal{S},\ j \in J \text{ such that } SPEC \overset{P_j}{\Rightarrow} SPEC'\}$. We also use the notation $P_j(SPEC)$ for $J = \{j\}$ and $\mathcal{S} = \{SPEC\}$.

**Definition 9.** A rule $P_1$ is *k-monotonic* with respect to a rule $P_2$ if and only if

$$\forall SPEC:\ \mid P_2(SPEC) \mid = k \overset{implies}{\Longrightarrow} (\forall\ SPEC_1 \in P_1(SPEC) \mid P_2(SPEC_1) \mid \leq k)$$

A rule $P_1$ is called *monotonic* w.r.t. a rule $P_2$ if and only if

$$\forall SPEC:\ \ P_2(SPEC) = \emptyset \overset{implies}{\Longrightarrow} P_2P_1(SPEC) = \emptyset$$

This definition gives rise to a simple improvement whenever we find those rules where the non-applicability of the second to the result of the first can be predicted, provided the second has been non-applicable before. Thus monotonicity allows to eventually skip some `Recognize`-calls.

**Fact 2.** *Given rules* $\{P_1, P_2, \ldots, P_n\}$, *each k-monotonic w.r.t. a rule* $P_q$, *for every* $SPEC_{s_i} \in P_{s_i} \cdots P_{s_1}(SPEC)$ , *with* $i \geq 1$, $s_j \in \{1, 2, \ldots, n\}$, $j = 1, \ldots, i$:

$$\mid P_q(SPEC) \mid = k \stackrel{implies}{\Longrightarrow} \mid P_q(SPEC_{s_i}) \mid \leq k$$

In order to let a search-algoritm take advantage of a *precomputation pass* which distinguishes rules which are monotonic, *effectively computable* criteria must be found. A **syntactical monotonicity criterion** is an effectively computable binary predicate on rules telling whether these rules are monotonic.
Given two syntactical monotonicity criteria $SC$ and $SC_b$, the latter is said to be **better** if and only if $SC \subset SC_b$. A syntactical monotonicity criterion is said to be **optimal** if and only if there is no better syntactical monotonicity criterion.
Asking how an interaction of rules can effectively be characterized, we start by looking at the ways in which two rules may overlap in a derivation. To be more precise, we ask how the part in specification $SPEC_2$, defined by the intersection of the images of the occurrence morphisms $r_1 : EXP_1 \rightarrow SPEC_2$, $l_2 : IMP_2 \rightarrow SPEC_2$, can be characterized when $SPEC_1 \stackrel{p_1}{\Rightarrow} SPEC_2 \stackrel{p_2}{\Rightarrow} SPEC_3$.

**Definition 10.** (Gluing Relation Set)
Given two specification morphisms $e_1 : PAR_1 \rightarrow EXP_1$, $i_2 : PAR_2 \rightarrow IMP_2$, the **gluing relation set** is the set $\widetilde{GRS}\,(e_1, i_2)$ of relations

$$\widetilde{gr} = (\widetilde{gr}_S \subseteq S_{EXP_1} \times S_{IMP_2},\ \widetilde{gr}_{OP} \subseteq OP_{EXP_1} \times OP_{IMP_2},\ \widetilde{gr}_E \subseteq E_{EXP_1} \times E_{IMP_2})$$

on sorts, operations and equations, such that each $\widetilde{gr} \in GRS(e_1, i_2)$ also satisfies the following axioms.
For $a, a' \in EXP_1$, $b, b' \in IMP_2$, $\overline{a} \in EXP_1 \setminus e_1(PAR_1)$, $\overline{b} \in IMP_2 \setminus i_2(PAR_2)$:

**(Ax1).** $a\ \widetilde{gr}_{OP}\ b \Rightarrow (cod(a)\ \widetilde{gr}_S\ cod(b))$ and "every $x \in dom(a)$ bijectively corresponds to a $y \in dom(b)$ such that $x\ \widetilde{gr}_S\ y$"
**(Ax2).** $a\ \widetilde{gr}_E\ b \Rightarrow\ MATCH(a, b) = TRUE$
**(Ax3).** $a\ \widetilde{gr}_S\ \overline{b} \Rightarrow \forall\ N \in OPNS(a)\ \exists\ N' \in OPNS(\overline{b})$ *and* $N\ \widetilde{gr}_{OP}\ N'$
**(Ax4).** $\overline{a}\ \widetilde{gr}_S\ b \Rightarrow \forall\ N' \in OPNS(b)\ \exists\ N \in OPNS(\overline{a})$ *and* $N\ \widetilde{gr}_{OP}\ N'$
**(Ax5).** $(a\ \widetilde{gr}_S\ \overline{b})$ *and* $(a\ \widetilde{gr}_S\ b') \Rightarrow\ \overline{b} = b'$
**(Ax6).** $(\overline{a}\ \widetilde{gr}_S\ b)$ *and* $(a'\ \widetilde{gr}_S\ b) \Rightarrow\ \overline{a} = a'$
**(Ax7).** $(a\ \widetilde{gr}_{OP}\ \overline{b})$ *and* $(a\ \widetilde{gr}_{OP}\ b') \Rightarrow\ \overline{b} = b'$
**(Ax8).** $(\overline{a}\ \widetilde{gr}_{OP}\ b)$ *and* $(a'\ \widetilde{gr}_{OP}\ b) \Rightarrow\ \overline{a} = a'$
**(Ax9).** $a\ \widetilde{gr}_{OP}\ \overline{b} \Rightarrow \forall\ e_1 \in EQNS(a)\ \exists\ e_2 \in EQNS(\overline{b})$ *and* $e_1\ \widetilde{gr}_{OP}\ e_2$
**(Ax10).** $\overline{a}\ \widetilde{gr}_{OP}\ b \Rightarrow \forall\ e_2 \in EQNS(b)\ \exists\ e_1 \in EQNS(\overline{a})$ *and* $e_1\ \widetilde{gr}_{OP}\ e_2$

The function MATCH verifies that the two equations can be translated into each other via the identifications of the relation $\widetilde{gr}_{OP}$.

A relation $\widetilde{gr} \in \widetilde{GRS}$, with $\widetilde{gr} \subseteq e_1(PAR_1) \times i_2(PAR_2)$ is called an *Interface* relation.

**Proposition 11.** *Given two specification morphisms* $e : PAR_1 \rightarrow EXP$, $i : PAR_2 \rightarrow IMP$, *each element* $\widetilde{gr} \in \widetilde{GRS}\ (e, i)$ *identifies an algebraic specification* $SPEC_{\widetilde{gr}} \in SPEC_{\widetilde{GRS}}$.

There are two 'projection' morphisms $\pi_1 : SPEC_{\widetilde{gr}} \rightarrow EXP$ and $\pi_2 : SPEC_{\widetilde{gr}} \rightarrow IMP$ defined by:
- $\forall\ (x,y) \in SPEC_{\widetilde{gr}} :\ \pi_1((x,y)) = x,\ \pi_2((x,y)) = y.$
If we replace the elements $SPEC_{\widetilde{gr}}$ in $SPEC_{\widetilde{GRS}}$ by the element $(SPEC_{\widetilde{gr}}, \pi_1, \pi_2)$, we obtain a new set, denoted by $PB_{\widetilde{GRS}}$.

**Proposition 12.** *Given two rules*

$$P_1 = (IMP_1 \xleftarrow{i_1} PAR_1 \xrightarrow{e_1} EXP_1) \text{ and } P_2 = (IMP_2 \xleftarrow{i_2} PAR_2 \xrightarrow{e_2} EXP_2)$$

*the set $PB_{\widetilde{GRS}}$ contains exactly the pullbacks of all $EXP_1 \xrightarrow{r_1} SPEC_1 \xleftarrow{l_2} IMP_2$, with $r_1$ and $l_2$ occurrence morphisms.*

The following Match Theorem is a special case of the Concurrency Theorem for HLR-Systems, which holds for the particular choice of specification morphisms reviewed in section 2 ([9]).

**Theorem 13.** *Given the sequence of derivations*

*(S1)* $$SPEC_1 \overset{P_1}{\Rightarrow} SPEC_2 \overset{P_2}{\Rightarrow} SPEC_3$$

*there exists a SPEC-derivation with the match-production $P_1 *_M P_2$*

*(S2)* $$P_1 *_M P_2 : SPEC_1 \Rightarrow SPEC_3$$

*called matched derivation of the sequence (S1).*
*Viceversa each direct SPEC-derivation (S2), using $P_1 *_M P_2$ leads to a derivation sequence (S1) using $P_1$ and $P_2$.*

Now we can use the set of all gluing relations to characterize the way in which rules may overlap.

**Lemma 14.** *Given $P_1 = (IMP_1 \xleftarrow{i_1} PAR_1 \xrightarrow{e_1} EXP_1)$ and $P_2 = (IMP_2 \xleftarrow{i_2} PAR_2 \xrightarrow{e_2} EXP_2)$ and subspecifications $S_E \subseteq EXP_1$, $S_I \subseteq IMP_2$, then the proposition*

$$\forall\ SPEC_1 \overset{P_1}{\Rightarrow} SPEC_2 \overset{P_2}{\Rightarrow} SPEC_3,\quad r_1(S_E) \cap l_2(S_I) = \emptyset$$

*with $r_1 : EXP_1 \rightarrow SPEC_2$ and $l_2 : IMP_2 \rightarrow SPEC_2$, is equivalent to*

$$\forall\ \widetilde{gr} \in \widetilde{GRS}\ (e_1, i_2),\quad \widetilde{gr} \not\subseteq S_E \times S_I$$

Using this lemma, potential overlapping of occurrences, defined as universally quantified propositions over an infinite number of specifications, are effectively decidable. In fact, since each of the specifications $EXP_1 = (S_1, OP_1, E_1)$ and $IMP_2 = (S_2, OP_2, E_2)$ is finite and so are the sets $S = S_1 \times S_2$, $OP = OP_1 \times OP_2$, $E = E_1 \times E_2$ and the powersets $\mathcal{P}(S)$, $\mathcal{P}(OP)$ and $\mathcal{P}(E)$, all $\widetilde{gr}$-axioms can be checked in a finite number of steps. The classical notion of *parallel iterdependency* leads to a syntactical monotonicity criterion.

**Definition 15.** A rule $P_1$ is said to be **S-independent** of a rule $P_2$ if and only if

$$\forall\ EXP_1 \xrightarrow{r_1} SPEC \xleftarrow{l_2} IMP_2 :\ r_1(EXP_1) \cap l_2(IMP_2) \subseteq r_1(e_1(PAR_1)) \cap l_2(i_2(PAR_2))$$

where $r_1$ and $l_2$ satisfy the gluing conditions.

**Fact 3. S-indipendence** *is a syntactical k-monotonicity criterion.*

Unfortunately, s-indipendence is only a weak monotonicity criterion, since it is only a sufficient criterion: there could be a pair of rules $P_1$ and $P_2$ such that $P_1$ is monotonic w.r.t. $P_2$, altough $P_1$ is not s-indipendent of $P_2$.

**Lemma 16.** *Given rules $P_1$, $P_2$ and $P_3$,*

$$\left.\begin{array}{l} \forall\, \widetilde{gr} \in \widetilde{GRS} \text{ and } P_* = P_1 *_M P_2 = IMP_* \xleftarrow{l_*} PAR_* \xrightarrow{r_*} EXP_*) \\ \text{with } M = (SPEC_{\widetilde{gr}}, \pi_1, \pi_2) \in PB_{\widetilde{GRS}} \\ \text{there is an occurrence morphism } g_3^+ : IMP_3 \to IMP_* \\ \text{such that } (g_{3S}^+(NGl_{3S}) \subseteq NGl_{*S} \wedge g_{3OP}^+(NGl_{3OP}) \subseteq NGl_{*OP})\} \, (UCC) \end{array}\right\} (SintR)$$

*is equivalent to*

$$\left.\forall SPEC\ (\exists SPEC \overset{P_1}{\Rightarrow} SPEC_1 \overset{P_2}{\Rightarrow} SPEC_2 \overset{implies}{\Longrightarrow} \exists\, SPEC \overset{P_3}{\Rightarrow} SPEC_3)\right\} (SemR)$$

*Remark.* – In a production $P = (IMP \xleftarrow{i} PAR \xrightarrow{e} EXP)$, the elements of the set $Gl = i(PAR)$ are called *Gluing Elements*, while $NGl = IMP \setminus i(PAR)$ contains the *Non-Gluing Elements.*

– (SyntR) is a shorthand for *Syntactical Relation*, whereas (UCC) stands for *Un-Criticalness Condition.* The *Semantical Relation* (SemR) can be read as: 'if $P_1$ and $P_2$ can sequentially be applied to $SPEC$, then $P_3$ must be applicable to $SPEC$' and is equivalent to $P_3(SPEC) = \emptyset \overset{implies}{\Longrightarrow} P_2P_1(SPEC) = \emptyset$

**Definition 17.** Given two rules $P_1$ and $P_2$, $P_2$ is **M-independent** of $P_2$, if and only if for each non-Interface relation $\widetilde{gr} \in \widetilde{GRS}$ there is an occurrence morphism $g_2^* : IMP_2 \to IMP_*$ with $P_* = P_2 *_M P_1 = IMP_* \xleftarrow{i_*} PAR_* \xrightarrow{e_*} EXP_*$ and $M = (SPEC_{\widetilde{gr}}, \pi_1, \pi_2) \in PB_{\widetilde{GRS}}$, such that the *Uncriticalness Condition* $(g_{2S}^*(NGl_{2S}) \subseteq NGl_{*S} \wedge g_{2OP}^*(NGl_{2OP}) \subseteq NGl_{*OP})$ holds.

**Theorem 18.** *M-independence is an optimal syntactical monotonicity criterion.*

Any addition of correct software in any library should modify the information about the M-indipendence among all induced rules. In this way, each search algorithm could exploit search-space reduction, as well as a reduction of efforts to determine the applicability of rules. Along the lines of [12], there is a notion of *semi-commutativity* of $P_1$ w.r.t. $P_2$ which allows to interchange their applications. It can be shown that *S-independence* is a syntactical semi-commutativity criterion and that there is an optimal semi-commutativity criterion, called SC-independence. For lack of space, we refer to [2] for formal definition and proofs.

*Example 2.* . Given the specification morphism $e_1$ and $i_2$, we illustrate the elements of the set $\widetilde{GRS}\,(e_1, i_2) = \{\widetilde{gr}_0, \widetilde{gr}_1\}$, with:

– $\widetilde{gr}_0 = (\emptyset, \emptyset, \emptyset)$
– $\widetilde{gr}_1 = (\{(nat, data)\}, \emptyset, \emptyset)$.

The $\widetilde{gr}$ relation $\widetilde{gr}_2 = (\{(nat, string)\}, \emptyset, \emptyset)$ is not contained in $\widetilde{GRS}\ (e_1, i_2)$ because it does not verify **(Ax.3)**. Some $\widetilde{gr}$ with $\widetilde{gr}_{OP} \neq \emptyset$ should include the pair $(ZERO, NIL)$, but this requires the validity of $nat\ \widetilde{gr}_S\ string$. Both $\widetilde{gr}_0$ and $\widetilde{gr}_1$ are Interface relations.
The set $\widetilde{GRS}\ (e_2, i_1)$ is $\{\widetilde{gr}'_0, \widetilde{gr}'_1, \widetilde{gr}'_2\}$, in which

- $\widetilde{gr}'_0 = (\emptyset, \emptyset, \emptyset)$
- $\widetilde{gr}'_1 = (\{(data, nat)\}, \emptyset, \emptyset)$
- $\widetilde{gr}'_2 = (\{(string, nat)\}, \{(NIL, ZERO), (INVERT, SUCC)\}, \emptyset)$

and $\widetilde{gr}'_0$ and $\widetilde{gr}'_1$ are Interface relations.
The production $P_{MN}$ is M-independent of the production $P_{IS}$, because of all the relations in $\widetilde{GRS}\ (e_1, i_2)$ are Interface relations. Then, by the previous theorem:

$$\forall SPEC: \ P_{IS}(SPEC) = \emptyset \overset{implies}{\Rightarrow} P_{IS}P_{MN}(SPEC) = \emptyset$$

Viceversa, $P_{IS}$ is not M-independent of $P_{MN}$; in fact for the only not Interface relation $\widetilde{gr}'_2$ not exist such an occurrence morphism from MN-Imp to $IMP_*$ when $P_{IS} *_M P_{MN} = (IMP_* \leftarrow PAR_* \rightarrow EXP_*)$ and $M = (SPEC_{\widetilde{gr}'_2}, \pi_1, \pi_2) \in PB_{\widetilde{GRS}(e_1, i_2)}$.

## 5 Concluding Remarks

In this paper we have addressed the problem of deriving a given specification in an algebraic specifications grammar. Elsewhere [13,14], it has been shown that if the productions of the specification grammar are the interfaces of module specifications, then a derivation sequence from the initial specification of the grammar to the objective specification can be automatically translated into an interconnection of the corresponding module specifications. Even for small libraries of modules, the search space for the problem of deriving a specification can be intractably large., We have found syntactical criteria to prune the search tree by analyzing only the interaction of the different productions, independently of the specification to be generated. So, if two productions $P_1$ and $P_2$ are, say, commutative, then only one of the two sequences $P_1P_2$ and $P_2P_1$ is considered. To guide the search in the pruned tree, we have used the notion of similarity, to *measure the distance* between two specifications. In choosing the appropriate occurrence of a production, the *total weight* of a morphism is used, defined in terms of an arbitrary *importance map* defined on the goal specification.

One of the objectives of this work is to produce an "automatic helper" to assist in the design of a modular system from a library (typically a prototype to investigate the feasibility and to validate the adequacy of the goal specification). What has been developed does not depend on the notion of module specification chosen, but can be used in any context where productions of algebraic specifications are used [15]. While most of section 4 depends on syntactical criteria based on the intrinsic

structure of the algebraic specifications, the development in section 3 is based on the notion of similarity and of weight of an occurrence, both defined essentially in terms of morphisms and therefore directly extendable to institutions other than the one used (essentially to simplify the presentation).

## References


1. E. K. Blum, H. Ehrig, F. Parisi-Presicce, *Algebraic Specification of Module and their Interconnections*, J. Comp. System Sci. 34, 2/3, 1987,239-339.
2. S. Cicerone, F.Parisi-Presicce:*Strategies in Modular System Design by Interface Rewriting*, Technical Report N. 39/93, Dip. Matematica Pura ed Applicata, Univ. L'Aquila, 1993.
3. S. Cicerone, F.Parisi-Presicce:*On the Complexity of Specification Morphism*, Technical Report N.32/93, Dip. Matematica Pura ed Applicata, Univ. L'Aquila, 1993.
4. H.Ehrig: *Introduction to the Algebraic Theory of Graph Grammars*, LNCS 73, 1-69, 1979.
5. H.Ehrig, A.Habel, H.-J.Kreowski, F.Parisi-Presicce: *From Graph Grammars to High-Level Replacement System*, Proc. 4 Int. Workshop on Graph Grammars and Application to Comp. Sci., LNCS 532, 1991, 269-291.
6. H.Ehrig, B.Mahr: *Fundamentals of Algebraic Specification 1: Equation and Initial Semantics*, EATCS Monographs on Theoret. Comp. Sci., vol. 6, Springer-Verlag, 1985.
7. H.Ehrig, B.Mahr: *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*, EATCS Monographs on Theoret. Comp. Sci., vol. 21, Springer-Verlag, 1990.
8. H.Ehrig, F.Parisi-Presicce: *Algebraic Specification Grammars: A Junction Between Module Specification and Graph Grammars*, Proc. 4 Int. Workshop on Graph Grammars and Application to Comp. Sci., LNCS 532, 1991, 292-310.
9. H.Ehrig, F.Parisi-Presicce: *High-Level Replacement System for Equational Algebraic Specification*, Proc. 3rd Int. Conf. Algebraic and Logic Programming, LNCS 632, 1992, 3-20.
10. H. Ehrig, H. Weber: *Algebraic Specification of Modules*, in 'Formal Models in Programming' (E.J.Neuhold,G.Chronist,eds.), North-Holland, 1985.
11. J. A. Goguen, J. Meseguer:*Universal Realization, Persistent Interconnection an Implentation of Abstract Modules*, LNCS 140, 1982, 265-281.
12. M.Korff: *Application of Graph Grammars to Rule-Based System*, Proc. 4 Int. Workshop on Graph Grammars and Application to Comp. Sci., LNCS 532, 1991, 505-519.
13. F.Parisi-Presicce:*A Rule-Based Approach to Modular System Design*, Proc. 12 Int. Conf. Soft. Eng., Nice(France), 1990, 202-211.
14. F.Parisi-Presicce:*Foundation of Rule-Based Design of Modular System*, Theoretical Comp. Science 83, 1991, 131-155.
15. F.Parisi-Presicce:*Reusability of Specifications and Implementations*, Proc. 2nd Int. Conf. on Alg. Method and Soft. Techn., AMAST '91, (M.Nivat, T.Rus, G.Scollo, C. Rattray eds.), Springer-Verlag 1992, 43-56.
16. D. L. Parnas: *A Technique for Module Specification with examples*, Comm. ACM 15, 5, 1972, 330-336.