Theoretical Computer Science 83 (1991) 131–155
Elsevier

Foundations of rule-based design of modular systems

Francesco Parisi-Presicce

Dipartimento di Matematica Pura ed Applicata, Università degli Studi di L'Aquila, I-67100 L'Aquila, Italy

Abstract

Parisi-Presicce, F., Foundations of rule-based design of modular systems, Theoretical Computer Science 83 (1991) 131–155.

By treating the interfaces of a module specification as a production, we combine notions from the well known theory of algebraic graph grammars with the theory of large software system specifications to tackle the problem of designing modular systems. Given a goal specification, a library of module specifications as reusable software components and a collection of "primitive" realized data type specifications, the designing of a system consisting only of the library components to realize the goal is reduced to deriving the goal from the primitive specification using the given productions. If a derivation sequence exists, direct derivations and operations on productions are converted into the design of a modular system.

Introduction

Transformations of graphs and, more generally, of "structures" occur in many areas of computer science. Originated in the late 1960s with problems of pattern recognition, the development of graph grammars has seen its interaction with areas such as VLSI layout schemes, data bases, knowledge representation, analysis of concurrent systems, parallel computer architecture and software specification and development, among others [11, 12].

In this paper, we take as point of departure the algebraic approach based on "categorical gluing" (defined in [13] and [21] and further developed in [5, 14, 9, 8]) and modify it to apply its principles to modular software design. The modular approach to the development of large software systems we consider here is based on the formalization of a module specification as presented in [15, 2]. This notion is an extension of the notion of algebraic specification of abstract data types. It consists of four parts: an import interface, an export interface, a parameter part that they share, and a body. Its semantics is taken to be a functorial transformation from models of the import interface to models of the export interface. In the simplest framework of the basic algebraic case, the four parts are represented by algebraic

0304-3975/91/$03.50

specifications, with properties of the operations expressed as equations or positive conditional equations, and are related by specification morphisms. The formalism has been extended to include algebraic constraints and first order logical formulas [7] or specifications in more general institutions.

The stepwise modular development of software systems requires interconnection mechanisms to form the horizontal structuring of module specifications. These interconnection mechanisms have been modelled as operations on module specifications, parametrized by specification morphisms. Among them are

- union, where the corresponding parts of two modules are put together by specifying the common subpart to be identified;
- composition, where the export interface of one module is matched with the import interface of another module;
- actualization, where the parameter part is replaced by an actual specification;
- extension, recursion, product, and partial composition and actualization [2, 18, 19].

The correctness of all the resulting modules (with the exception of recursion [18]) is a consequence of the correctness of the combining ones; furthermore, their semantics can be expressed uniquely in terms of those of the original modules.

The interfaces and their shared parameter part are the only parts of a module visible from the outside (e.g., system designer, user) and provide the "gates" through which each module interacts with the other modules. The problem we discuss in this paper is the following: given a library of module specifications and the specification of a basic data type realized in the environment, is it possible to design a modular system, using the given library, which realizes a goal specification?

Our approach to tackling this problem is based on viewing the *visible* part of a module specification as a production. The library of modules consists of a set of "productions" (each realized by possibly more than one body), the basic data types are represented by the initial specification and the answer to our problem is affirmative if the goal specification can be generated from a subspecification of the initial configuration using the library productions. Each direct derivation and each combination of productions corresponds to an operation on the module specifications represented as productions by their interfaces. The body part of each module in the modular design so obtained is not relevant. What is important is that the interfaces are realized and that the operations, used in the design and synthesized from the derivation sequence, are correct. The final parallel derivation sequence $SPEC0 \Rightarrow^* SPEC1$ from a subspecification of the basic data type specification to the goal specification can then be translated into a modular design which, given an algebra of the built-in type, provides an algebra of the goal type.

Restricting our attention to the basic algebraic case for simplicity of presentation, this paper is organized as follows: after reviewing the basic terminology and main definitions of module specifications with some of their interconnections in Section 2, we introduce the notion of SPEC-production and direct derivation via a double pushout construction in Section 3. We then investigate the notion of applicability,

by showing that an Extended Gluing Condition guarantees the existence of a pushout complement, the notions of composition and amalgamation of productions and that of independence of derivations on specifications. In Section 4, we deal with the problem of translating, into system design, concepts of SPEC-productions by viewing applicability as actualization and amalgamation as union. In Section 5, we draw some conclusions, discuss some work under way and mention some problems to be solved in the next phases of this investigation.

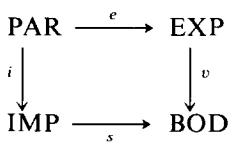
2. Module specifications

In this section we review some basic notions of algebraic specifications [10], the concept of module specification [2] and the operations of composition, actualization and union comprising the horizontal structuring mechanism of module specifications. We consider only the basic algebraic case of module specifications without constraints in the sense of [7].

Basic terminology

An *algebraic specification* SPEC $= (S, \mathrm{OP}, E)$ consists of a set S of sorts, a set OP of operation symbols and a set E of (positive conditional) equations. The three parts of a specification ACT are referred to by using the subscripts ACT_S, $\mathrm{ACT}_{\mathrm{OP}}$ and ACT_E. For $N \in \mathrm{OP}$, sorts(N) denotes the set of sorts of the signature of N. A *specification morphism* $f: \mathrm{SPEC1} \to \mathrm{SPEC2}$ is a signature morphism $(f_S, f_{\mathrm{OP}}): (S1, \mathrm{OP1}) \to (S2, \mathrm{OP2})$ such that $f^{\#}(E1)$ are provable, in some fixed calculus, from $E2$. For simplicity, we will restrict most of our discussion to *strict* morphisms for which $f^{\#}(E1)$ is contained in $E2$. Specifications and specification morphisms define a category CATSPEC, which is closed under pushouts [10]. Alg(SPEC) is the category of SPEC-algebras and SPEC-homomorphisms. A set E of equations is *minimal* in SPEC $= (S, \mathrm{OP}, E)$ if $\mathrm{Alg(SPEC)} = \mathrm{Alg}((S, \mathrm{OP}, E'))$ and $E \supset E'$ implies $E = E'$. Each specification morphism $f: \mathrm{SPEC1} \to \mathrm{SPEC2}$ defines a *forgetful functor* $V_f: \mathrm{Alg(SPEC2)} \to \mathrm{Alg(SPEC1)}$ and a *free functor* $F_f: \mathrm{Alg(SPEC1)} \to \mathrm{Alg(SPEC2)}$, the left adjoint of V_f. For more details, see [10].

Definition 2.1 (*Module specification*). (a) A *module specification* MOD consists of four algebraic specifications PAR (parameter part), EXP (export interface), IMP (import interface) and BOD (body) and four specification morphisms as in the following commutative diagram



(b) A module specification MOD is *correct* if the free functor F_S: Alg(IMP) → Alg(BOD) is strongly persistent (i.e., $V_S(F_S(A)) = A$ for all IMP-algebras A).

(c) The *semantics* SEM of MOD is the functor $V_V \cdot F_S$: Alg(IMP) → Alg(EXP).

Interpretation

The interfaces represent the only information available outside the module. Both interfaces are "contained" in the body specification, which provides an implementation of EXP by IMP. The operations and sorts in BOD but not in EXP are to be considered hidden. The semantics of MOD is a transformation between algebras: the import specifies the kind of algebra to be provided to the module to obtain an algebra which satisfies the export interface. The two interfaces share a parameter part, which, if the module is correct, is the only part of any IMP-algebra A guaranteed to be left unchanged by the semantical transformation ($V_i(A) = V_e(\text{SEM}(A))$). If PAR = IMP and EXP = BOD, this notion coincides with that of parametrized specification in [10].

Example 2.2. The example of a module for a flight schedule with flight number (f^*), destination and departure, is taken from [2]. Denoting by **bool** the standard specification of boolean values, FS-MOD = (FS-PAR, FS-EXP, FS-IMP, FS-BOD) where

FS-PAR = **bool** +
  sorts f^*, dest, dep
  opns EQF: $f^* f^* \to$ bool
    NODEP: → dep
  eqns EQF(F^*, F^*) = TRUE

FS-IMP = FS-PAR,

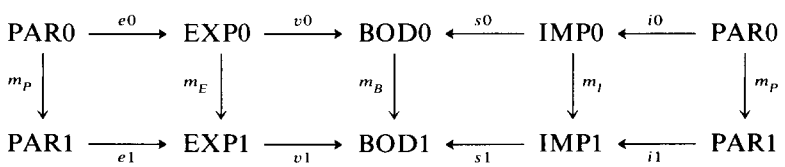
FS-EXP = FS-PAR +
  sorts fs
  opns CREATE-FS: → fs
    SEARCH-FS: f^* fs → bool
    ADD-FS: f^* dest dep fs → fs
    RETURN-FS: f^* fs → dep
    CHANGE-FS: f^* dep fs → f
  eqns SEARCH-FS(F^*, CREATE-FS) = FALSE
    SEARCH-FS(F^*, FS) = TRUE ⇒ ADD-FS(F^*, DEST, DEP, FS) = FS,
    SEARCH-FS(F^*, FS) = TRUE ⇒
      RETURN-FS(F^*, CHANGE-FS(F^*, DEP, FS)) = DEP.

and

FS-BOD = FS-EXP+
  opns TAB: f^* dest dep fs → fs
  eqns ADD-FS(F^*, DEST, DEP, CREATE-FS)
    = TAB(F^*, DEST, DEP, CREATE-FS)
  ADD-FS(F^*1, DEST1, DEP1, TAB(F^*, DEST, DEP, FS))
    = if EQF(F^*1, F^*)
    then TAB(F^*, DEST, DEP, FS)
    else TAB(F^*, DEST, DEP, ADD-FS(F^*1, DEST1, DEP1, FS))
  SEARCH-FS(F^*, CREATE-FS) = FALSE
  SEARCH-FS(F^*, TAB(F^*1, DEST, DEP, FS))
    = EQF(F^*, F^*1) or SEARCH-FS(F^*, FS)
  RETURN-FS(F^*, CREATE-FS) = NODEP
  RETURN-FS(F^*, TAB(F^*1, DEST, DEP, FS)) =
    if EQF(F^*, F^*1) then DEP else RETURN-FS(F^*, FS)
  CHANGE-FS(F^*, DEP, CREATE-FS) = CREATE-FS
  CHANGE-FS(F^*, DEP, TAB(F^*1, DEST1, DEP1, FS))
    = if EQF(F^*, F^*1)
    then TAB(F^*, DEST1, DEP, FS)
    else TAB(F^*1, DEST1, DEP1, CHANGE-FS(F^*, DEP, FS)).

The four specification morphisms e, i, s and v are all inclusions. The operation TAB in the body is used to implement the exported operation ADD but, not being present in FS-EXP, is not visible to the outside. The semantics of the module reflects the fact that the operations in the body not already in the import have only the properties specified by their (equational) definitions and the consequences (in the usual equational calculus) of these definitions. In other words, the defining equations can be seen as rewrite rules to compute the value of the functions. With this view of the body equations, the correctness of the module specification corresponds to the property that the rewrite rules just mentioned define a total function if the result is of a sort of the import algebra. No such restrictions are imposed on operations with result value of a new sort.

Definition 2.3 (*Submodule and union*). A *module specification morphism* m : MOD0 → MOD1 is a four-tuple $m = (m_P, m_E, m_I, m_B)$ of specification morphisms making each square of the following diagram commute



The morphism m is called *consistent* if $V_{m_B} \cdot F_{s1} = F_{s0} \cdot V_{m_I}$.

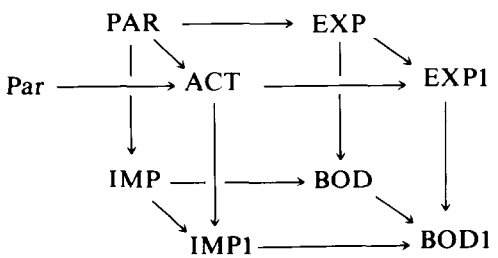
MOD0 is a *submodule* specification of MOD1 if there is an injective consistent module specification morphism $m: \text{MOD0} \to \text{MOD1}$. Given a submodule MOD0 of MOD1 and MOD2 via $m1: \text{MOD0} \to \text{MOD1}$ and $m2: \text{MOD0} \to \text{MOD2}$, the *union* of MOD1 and MOD2 w.r.t. MOD0, $m1$ and $m2$, is the module specification $\text{MOD3} = \text{MOD1} +_{\text{MOD0}} \text{MOD2}$ where each component is the pushout object (in the category CATSPEC) of the corresponding components of $m1$ and $m2$ and the morphisms are induced by the universal property of pushouts [2].

Interpretation

Each component of MOD3 is obtained by taking the disjoint union of the corresponding components of MOD1 and MOD2 and then identifying the images of MOD0 under $m1$ and $m2$. Parts in common to MOD1 and MOD2 not contained in MOD0 are duplicated. Each IMP3-algebra $A3$ is the amalgamated sum $A1 +_{A0} A2$ of IMPj-algebras [10, 1] and the semantics $\text{SEM1} +_{\text{SEM0}} \text{SEM2}$ of MOD3 transforms $A1$ and $A2$ exactly as SEM1 and SEM2 would: the consistency of $m1$ and $m2$ guarantees that the restriction of SEMi to $A0$ is exactly SEM0 (i.e. $V_{m2_E} \cdot \text{SEM2} \cdot V_{m2_I} = V_{m1_E} \cdot \text{SEM1} \cdot V_{m1_I}$).

The operation of actualization of a module specification MOD = (PAR, EXP, IMP, BOD) consists of "replacing" the parameter part PAR by a (parametrized) specification PS = (Par, ACT) with $j: \text{Par} \to \text{ACT}$ via a parameter passing morphism $h: \text{PAR} \to \text{ACT}$.

Definition 2.4 (*Actualization*). Given a module specification MOD = (PAR, EXP, IMP, BOD), a parametrized specification PS = (Par, ACT) and a parameter passing specification morphism $h: \text{PAR} \to \text{ACT}$, the *actualization of* MOD *by* PS *via* h, denoted by $\text{act}_h(\text{PS}, \text{MOD})$, is the module specification MOD1 = (Par, EXP1, IMP1, BOD1), where $\text{EXP1} = \text{ACT} +_{\text{PAR}} \text{EXP}$, $\text{IMP1} = \text{ACT} +_{\text{PAR}} \text{IMP}$ and $\text{BOD1} = \text{IMP1} +_{\text{IMP}} \text{BOD}$ and the specification morphisms are induced by the universal property of the pushouts.



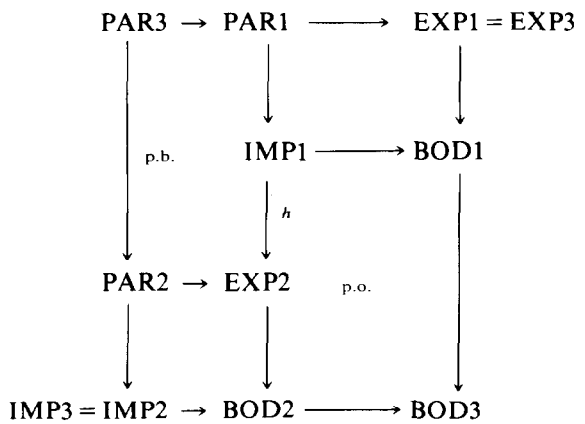
Interpretation

The parameter part PAR of MOD is replaced by ACT, whose sorts and operations are added to the interfaces and the body. The parameter part of the new module specification MOD1 is that of the parametrized specification PS (which could be

empty). The semantics SEM1 of the actualized module is given by $SEM1(A' +_P A) = A' +_P SEM(A)$, where $A \in Alg(IMP)$, $A' \in Alg(ACT)$ and $P = V_i(A) = V_h(A')$.

The third basic operation on module specifications is that of composition, where the import interface of a module specification is "matched" with the export interface of another one. The "unused" interfaces provide two of the components of the composite module specification. A specification morphism h provides the match of the import interface of MOD1 with the export interface of MOD2. The new body is the body BOD1 where IMP1 has been replaced by BOD2. Unlike the product on modules [19], the operations of EXP2 are no longer exported by the resulting module.

Definition 2.5 (*Composition*). Given module specifications $MODj = (PARj, EXPj, IMPj, BODj)$, $j = 1, 2$, and an interface morphism $h: IMP1 \to EXP2$, the *composition* $MOD1 \cdot_h MOD2$ is the module specification $MOD3 = (PAR3, EXP3, IMP3, BOD3)$ as in the following diagram



where BOD3 is the pushout object of $s1: IMP1 \to BOD1$ and $h \cdot v2: IMP1 \to BOD2$ and PAR3 is the pullback object of $e2: PAR2 \to EXP2$ and $i1 \cdot h: PAR1 \to EXP2$.

As for actualization and union, the resulting module is correct if the component modules MOD1 and MOD2 are, and its semantics SEM3 is given by the composition $SEM1 \cdot V_h \cdot SEM2$ of the respective semantics, interfaced by the "translation" via h.

It is sufficient [17] to consider composition with an identity morphism id in place of h. We will drop the subscript and just write $MOD1 \cdot MOD2$.

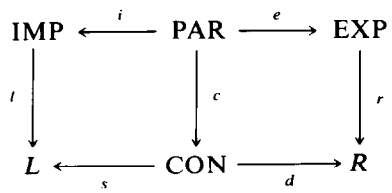
3. Modules as productions

As we have seen in the previous section, a module specification consists of four parts: two interfaces, a shared parameter part and a body. Of these parts, only the interfaces and the parameter are visible to the system designer. The interfaces and

their shared parameter part provide the "gates" through which each module interacts with the other modules, via specification morphisms. Our approach is based on viewing the interfaces of a module specification as a production and on applying these productions to the initial specification. If the goal specification is generated, then it can be realized and the modular system can be built using the way in which the productions are applied and the parallel, concurrent and amalgamated productions which may have been constructed during the derivation. How to translate these derivations into modular systems is the topic of the next section. In this section we define the basic notions of SPEC-derivation, production and other basic concepts common to graph grammars. It should be pointed out that the productions and the derivations are symmetric, and therefore no assumptions are made on possible search methods to determine the realizability of the goal specification (goal-oriented, depth-first, breadth-first, etc.).

Definition 3.1 (*SPEC-production*). (a) A *SPEC-production* is an ordered pair, denoted by PRO = (IMP ← PAR → EXP), of SPEC-morphisms i: PAR → IMP and e: PAR → EXP with a common domain.

(b) A *direct derivation* via the production PRO consists of the following two pushout diagrams in the category CATSPEC



We write PRO: $L \Rightarrow R$ and say that R *is derivable from* L *via* PRO. When it is necessary to specify the morphism c, we write (PRO, c): $L \Rightarrow R$. A production is *injective* if i and e are injective.

Interpretation

The middle specification PAR represents the part of IMP left unchanged by the production. The specification CON in the direct derivation is the part of L not affected by the derivation: it is the "context" of the derivation and it is "glued" to the right hand side EXP of the production via the common subspecification PAR, which represents the "interface" between the unchanged context and the modified part (from IMP to EXP). Notice that if PRO: $L \Rightarrow R$, then PRO^{-1}: $R \Rightarrow L$, where PRO^{-1} is the production (EXP ← PAR → IMP).

Example 3.2. The interfaces of the FS-MOD of Example 2.2 along with the obvious inclusions form a production of algebraic specifications FS-PRO. When this

production is applied to the specification $L = \textbf{nat} + \textbf{string}$ where

nat = **bool** + sorts nat
opns 0: → nat
succ: nat → nat
Eq: nat nat → bool
eqns Eq(succ(x), succ(y)) = Eq(x, y)
Eq(x, x) = TRUE
Eq(0, succ(x)) = FALSE
Eq(succ(x), 0) = FALSE

and

string = sorts alph, str
opns EMPTY: → str
MAKE: alph → str
CONC: str str → str
eqns CONC(CONC(x, y), z) = CONC(x, CONC(y, z))
CONC(EMPTY, x) = x = CONC(x, EMPTY)

via the specification morphism l: FS-IMP → **nat** + **string** with

$l_S(f^*)$ = nat = l_S(dep), l_S(dest) = str, l_{OP}(EQF) = Eq, l_{OP}(NODEP) = EMPTY,

it produces the specification

$R = L +$ sorts fs
opns CREATE-FS: → fs
SEARCH-FS: nat fs → bool
ADD-FS: nat str nat fs → fs
RETURN-FS: nat fs → dep
CHANGE-FS: nat nat fs → fs
eqns SEARCH-FS(N, CREATE-FS) = FALSE
SEARCH-FS(N, FS) = TRUE
⇒ ADD-FS(N, ST1, ST2, FS) = FS,
SEARCH-FS(N, FS) = TRUE
⇒ RETURN-FS(N, CHANGE-FS(N, ST1, FS)) = ST1.

Notice that in this example the effect of applying the production is to add to the specifications of **nat** and **string** the five operator symbols labelled with -FS and the corresponding equations. This is always the case when the production is *deductive*, i.e., when PAR = IMP. In this case, it is sufficient to have a specification morphism l: IMP → L to obtain a direct derivation via the production. In general, a production PRO = (IMP ← PAR → EXP) is *applicable* to the specification L if there exists a context specification CON and a specification morphism c: PAR → CON such that $L = \text{IMP} +_{\text{PAR}} \text{CON}$, i.e., if L can be decomposed into two parts IMP and CON which share *exactly* the PAR part of the production. PAR represents the "boundary" between the part IMP involved in the derivation via PRO and the part CON not affected by the production. The result of the derivation is then the specification

obtained by "gluing" CON and EXP via PAR. In general, the "pushout complement" $\mathrm{CON} = L -_{\mathrm{PAR}} \mathrm{IMP}$ need not exist even if $L \supseteq \mathrm{IMP}$.

Example 3.3. Let

$\mathrm{PAR} =$ sort $s0$
$\mathrm{IMP} =$ sort nat
opns $0: \to \mathrm{nat}$
$\mathrm{succ}: \mathrm{nat} \to \mathrm{nat}$
$+: \mathrm{nat}\ \mathrm{nat} \to \mathrm{nat}$
eqns $0 + n = n$
$\mathrm{succ}(n) + m = \mathrm{succ}(n + m)$

$L = \mathrm{IMP} +$ opns $\mathrm{pre}: \mathrm{nat} \to \mathrm{nat}$
eqns $\mathrm{pre}(n) + m = \mathrm{pre}(n + m)$
$\mathrm{pre}(\mathrm{succ}(n)) = n$
$\mathrm{succ}(\mathrm{pre}(n)) = n$

with $i_S(s0) = \mathrm{nat}$, i_{OP} empty, and with inclusion as the occurrence morphism l.

Any production with $\mathrm{PAR} \to \mathrm{IMP}$ as the first morphism is *not* applicable to the specification L because there is no context specification CON such that $L = \mathrm{IMP} +_{\mathrm{PAR}} \mathrm{CON}$. Such a specification $(\mathrm{CON}_S, \mathrm{CON}_{\mathrm{OP}}, \mathrm{CON}_E)$ in fact would have one sort, call it $s0$, the operator symbol $\mathrm{pre}: \mathrm{nat} \to \mathrm{nat}$ and the three equations present in L but not in IMP. But such a triple is not a well defined specification since the operator symbol pre has in its signature the sort nat not in CON_S. Adding the sort nat to CON_S would give us two distinct "nat" sorts in L_S, since nat is not a sort of PAR.

Intuitively, for the pushout complement to exist, the specification CON on the one hand should contain all the sorts and operator symbols of PAR, and the sorts and operator symbols of L not contained in IMP. Also for CON to be well defined, the operator symbols cannot use sorts not contained in the set of sorts S_{CON} (in a representation of specifications as graphs with sorts as nodes and operators as (hyper)edges, these operators would correspond to "dangling" edges, i.e., edges without source or target node). Finally, the equations E_{CON} of CON cannot be formed using operator symbols not in $\mathrm{OP}_{\mathrm{CON}}$ if we want a properly defined specification. If the "occurrence" morphism $l: \mathrm{IMP} \to L$ is injective, this is all that is needed for l to be applicable. If not, then there are some technical conditions that require the items collapsed by l to be "gluing" items.

Definition 3.4 (*Gluing Condition*). Given two specification morphisms $f1: \mathrm{SPEC0} \to \mathrm{SPEC1}$ and $g1: \mathrm{SPEC1} \to \mathrm{SPEC3}$ define

$$\mathrm{ID}(g1)_S = \{s1 \in S1: \exists s1' \in S1,\ s1' \neq s1,\ g1_S(s1) = g1_S(s1')\},$$
$$\mathrm{ID}(g1)_{\mathrm{OP}} = \{N \in \mathrm{OP1}: \exists N' \in \mathrm{OP1},\ N \neq N',\ g1_{\mathrm{OP}}(N) = g1_{\mathrm{OP}}(N')\}.$$

Then $g1$ is *injective up to* $f1$ if $f1_S(S0) \supseteq \mathrm{ID}(g1)_S$ and $f1_{\mathrm{OP}}(\mathrm{OP0}) \supseteq \mathrm{ID}(g1)_{\mathrm{OP}}$. The morphisms $f1$ and $g1$ satisfy the *Gluing Condition* if $g1$ is injective up to $f1$ and $f1_S(S0) \supseteq \mathrm{DANG}(g1)$, where $\mathrm{DANG}(g1) = \{s \in S1: \exists N \in \mathrm{OP3} - g1_{\mathrm{OP}}(\mathrm{OP1})$ and $g1_S(s) \in \mathrm{sorts}(N)\}$.

Theorem 3.5 (Pushout complement). *Given morphisms* $f1: \mathrm{SPEC0} \to \mathrm{SPEC1}$ *and* $g1: \mathrm{SPEC1} \to \mathrm{SPEC3}$, *define*

$$S2 = S3 - g1_S(S1 - f1_S(S0)) \text{ and } \mathrm{OP2} = \mathrm{OP3} - g1_{\mathrm{OP}}(\mathrm{OP1} - f1_{\mathrm{OP}}(\mathrm{OP0})).$$

If $f1$ *and* $g1$ *satisfy the Gluing Condition and* $E2 = E3 - g1^{\#}(E1 - f1^{\#}(E0))$ *is a set of equations in the signature* $(S2, \mathrm{OP2})$, *then the specification* $\mathrm{SPEC2} = (S2, \mathrm{OP2}, E2)$ *is a pushout complement of* SPEC1, *i.e., there exist specification morphisms* $f2: \mathrm{SPEC0} \to \mathrm{SPEC2}$ *and* $g2: \mathrm{SPEC2} \to \mathrm{SPEC3}$ *such that* $\mathrm{SPEC3} = \mathrm{SPEC1} +_{\mathrm{SPEC0}} \mathrm{SPEC2}$ *in* CATSPEC. *Conversely, if* $\mathrm{SPEC3} = \mathrm{SPEC1} +_{\mathrm{SPEC0}} \mathrm{SPEC2}$ *in* CATSPEC, *then there is a set of equations* $E3'$, *minimal among those equivalent to* $E3$ *and containing* $E1$, *such that* $E3' - g1^{\#}(E1 - f1^{\#}(E0))$ *is a set of equations in the signature* $(g2_S(S2), g2_{\mathrm{OP}}(\mathrm{OP2}))$.

Proof. Let $\mathrm{SPEC2} = (S2, \mathrm{OP2}, E2)$ as in the statement of the theorem; let $g2: \mathrm{SPEC2} \to \mathrm{SPEC3}$ be the obvious inclusion and $f2(x) = g1 \cdot f1(x)$ for both sorts and operator symbols.

We need to show that $f2$ and $g2$ are specification morphisms and that the diagram

$$\begin{array}{ccc} \mathrm{SPEC1} & \xleftarrow{f1} & \mathrm{SPEC0} \\ {\scriptstyle g1}\downarrow & & \downarrow{\scriptstyle f2} \\ \mathrm{SPEC3} & \xleftarrow[g2]{} & \mathrm{SPEC2} \end{array}$$

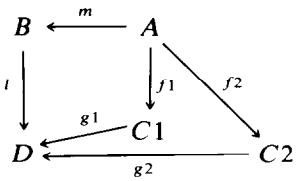
is a pushout, besides being commutative by definition.

First of all, the condition $f1_S(S0) \supseteq \mathrm{DANG}(g1)$ guarantees that all the sorts appearing in OP2 are in $S2$ and therefore, with the additional condition on the equations, that SPEC2 is a well defined specification. The inclusions $g2_S$ and $g2_{\mathrm{OP}}$ form a specification morphism since the inclusion $E3 \supseteq E2$ implies that $g2^{\#}(E2)$ is provable from $E3$. The well definedness of $f2 = (g1_S \cdot f1_S, g1_{\mathrm{OP}} \cdot f1_{\mathrm{OP}})$ as a specification morphism follows from the fact that $E2 \supseteq g1^{\#}f1^{\#}(E0)$ and thus obviously that $f2^{\#}(E0) = g1^{\#}f1^{\#}(E0)$ is provable from $E2$.

To check the universal property of the diagram above, let $h1: \mathrm{SPEC1} \to \mathrm{SPEC}$ and $h2: \mathrm{SPEC2} \to \mathrm{SPEC}$ be such that $h1 \cdot f1 = h2 \cdot f2$ and define $h3: \mathrm{SPEC3} \to \mathrm{SPEC}$ by $h3(x) = h1(x1)$ if $x = g1(x1)$ and $h3(x) = h2(x2)$ if $x = g2(x2)$. The fact that $g1$ and $g2$ are jointly surjective and the conditions $f1_S(S0) \supseteq \mathrm{ID}(g1)_S$ and $f1_{\mathrm{OP}}(\mathrm{OP0}) \supseteq \mathrm{ID}(g1)_{\mathrm{OP}}$ guarantee the well definedness and uniqueness of such morphisms. The usual commutative properties follow from the definition of $h3$.

To conclude that h is a SPEC-morphism, we need to show that $h^{\#}(E3)$ is provable from E. By definition, $h^{\#}(E3) = h^{\#}(g1^{\#}(E1) \cup g2^{\#}(E2)) = h^{\#}(g1^{\#}(E1)) \cup h^{\#}(g2^{\#}(E2)) = h1^{\#}(E1) \cup h2^{\#}(E2)$. Since $h1^{\#}$ and $h2^{\#}$ are specification morphisms, both $h1^{\#}(E1)$ and $h2^{\#}(E2)$ are provable from E and thus so is $h^{\#}(E3)$. The proof of the theorem is complete. □

Remarks 3.6. Notice that the pushout complement constructed in Theorem 3.5 is a *minimal* pushout complement, where $g2$ is injective. It is minimal in the sense that if in the diagram



$D = B +_A C1 = B +_A C2$ and $g1$ is injective, then there is $h: C2 \to C1$ such that $g2 = g1 \cdot h$. For $x \in C2$, $g2(x) \in D$ and since l and $g1$ are jointly surjective

- if $g2(x) = g1(y)$ for $y \in C1$ then $h(x) = y$
- if $g2(x) = l(z)$ then $x = f2(y)$ and $z = m(y)$ for some $y \in A$ and thus $h(x) = f1(y)$.

There may be several pushout complements of SPEC1 with respect to SPEC3 and SPEC0 (written SPEC2 = SPEC3 $-_{\text{SPEC0}}$ SPEC1), but *only one* minimal (up to isomorphism). The pushout complement is also unique, if it exists, whenever $f1$ is injective. We will assume for simplicity of presentation that all our productions have injective morphisms.

Corollary 3.7. *In the category of specifications and* strict *specification morphisms, given* $f1: \text{SPEC0} \to \text{SPEC1}$ *and* $g1: \text{SPEC1} \to \text{SPEC3}$, *there exists a pushout complement* SPEC2 = SPEC3 $-_{\text{SPEC0}}$ SPEC1 *if and only if* $f1$ *and* $g1$ *satisfy the Gluing Condition and* $E2 = E3 - g1^{\#}(E1 - f1^{\#}(E0))$ *is contained in the set of all equations over* $(S2, \text{OP2})$.

Productions can be combined to yield other productions, whose applicability and behavior can sometimes be derived from those of the component productions. In the remainder of the paper we will consider *only* strict morphisms.

Definition 3.8 (*Composite production*). The *composite* PRO1 · PRO2 of two productions PROj = (IMPj ← PARj → EXPj), $j = 1, 2$, with EXP1 = IMP2, is the production (IMP3 ← PAR3 → EXP3) where

- IMP3 = IMP1,
- EXP3 = EXP2,

- PAR3 = PAR1 $\times_{EXP1}$ PAR2, the pullback object of $e1$: PAR1 → EXP1 and $i2$: PAR2 → IMP2,
- $i3$: PAR3 → IMP3 = PAR3 → PAR1 → IMP1,
- $e3$: PAR3 → EXP3 = PAR3 → PAR2 → EXP2.

Notice that if $i1$ and $i2$ are injective, so is $i3$.

Definition 3.9 (*Matched derivations*). Two derivations PRO1: $G \Rightarrow G1$ and PRO2: $H1 \Rightarrow H$ are *matched* if EXP1 → $G1$ = IMP2 → $H1$ (i.e., same domain EXP1 = IMP2, same codomain $G1 = H1$ and same occurrence of EXP1 in $G1$).

The proof that two matched derivations can be replaced by a direct derivation via the composite production needs the following general result.

Lemma 3.10. *If the diagram*

$$\begin{array}{ccc} \text{SPEC0} & \xrightarrow{f1} & \text{SPEC1} \\ {\scriptstyle f2}\downarrow & & \downarrow{\scriptstyle g1} \\ \text{SPEC2} & \xrightarrow[g2]{} & \text{SPEC3} \end{array}$$

is a pullback in CATSPEC, $g1$ *and* $g2$ *are jointly surjective* (*i.e.*, $g1(\text{SPEC1}) \cup g2(\text{SPEC2}) \supset \text{SPEC3}$ *for sorts, operations and equations*), *and* $g1$ *and* $g2$ *are injective up to* $f1$ *and* $f2$, *respectively, then the diagram is a pushout.*

Proof. Let $h1$: SPEC1 → SPEC and $h2$: SPEC2 → SPEC be such that $h2 \cdot f2 = h1 \cdot f1$. We need to show that there exists a unique h: SPEC3 → SPEC such that $hj = h \cdot gj$, $j = 1, 2$. Let $x \in$ SPEC3 and define $h(x) = h1(y)$ if $x = g1(y)$, $y \in$ SPEC1.

To prove that h is well defined and unique, it is sufficient to show that, for $y, y' \in$ SPEC1, if $g1(y) = g1(y')$, then $h1(y) = h1(y')$. By assumption on $g1$, there exist $z, z' \in$ SPEC0 such that $y = f1(z)$ and $y' = f1(z')$. By the pullback property, $f2(z) = f2(z')$ and therefore $h1(y) = h1(f1(z)) = h2(f2(z)) = h2(f2(z')) = h1(f1(z')) = h1(y')$.

Similarly for $x = g2(w)$ for some $w \in$ SPEC2. The uniqueness of h is a consequence of the joint surjectivity of $g1$ and $g2$ and the assumption that the diagram is a pullback. □

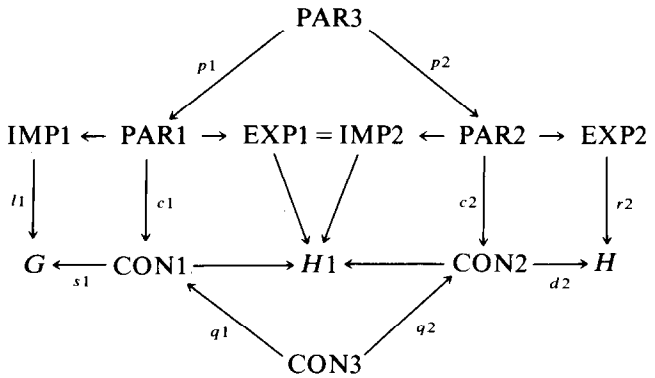
Theorem 3.11 (Composition of derivations). *Let* PROj = (IMPj ← PARj → EXPj), $j = 1, 2$, *be injective productions.*

(a) *If* PRO1: $G \Rightarrow H1$ *and* PRO2: $H1 \Rightarrow H$ *are matched derivations, then the composite* PRO1 · PRO2 *is applicable to* G *and* PRO1 · PRO2: $G \Rightarrow H$.

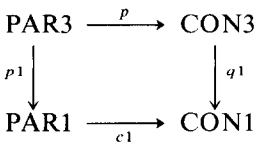
(b) *Conversely, if* PRO1 · PRO2: $G \Rightarrow H$, *then there exists a specification* $H1$ *such that*

- PRO1 *is applicable to* G *and* PRO1: $G \Rightarrow H1$,
- PRO2 *is applicable to* $H1$ *and* PRO2: $H1 \Rightarrow H$.

Proof. Define $\mathrm{CON3} = \mathrm{CON1} \times_{H1} \mathrm{CON2}$, where $\mathrm{IMP1} +_{\mathrm{PAR1}} \mathrm{CON1} = G$ and $\mathrm{IMP2} +_{\mathrm{PAR2}} \mathrm{CON2} = H1$.



The morphisms $c1: \mathrm{PAR1} \to \mathrm{CON1}$ and $q1: \mathrm{CON3} \to \mathrm{CON1}$ are jointly surjective since if $x \in \mathrm{CON1} - c1(\mathrm{PAR1})$ then $d1(x) \in H1$ but $d1(x) \notin r1(\mathrm{EXP1}) = l2(\mathrm{IMP2})$. Since $l2$ and $s2$ are jointly surjective, $d1(x) \in s2(\mathrm{CON2})$, say $d1(x) = s2(y)$. But then by definition of pullback, $x = q1(z)$ and $y = q2(z)$ for some $z \in \mathrm{CON3}$. Hence, if $x \in \mathrm{CON1} - c1(\mathrm{PAR1})$ then $x \in q1(\mathrm{CON3})$. The following diagram is a pullback



since if $h: D \to \mathrm{PAR1}$ and $h': D \to \mathrm{CON3}$ are such that $q1 \cdot h' = c1 \cdot h$, then for $x \in D$, $s2 \cdot q2 \cdot h'(x) = d1 \cdot q1 \cdot h'(x) = d1 \cdot c1 \cdot h(x) = r1 \cdot e1 \cdot h(x) = l2 \cdot e1 \cdot h(x)$ and therefore, for some $y \in \mathrm{PAR2}$, $c2(y) = q2 \cdot h'(x)$. Since $r1 \cdot e1 \cdot h(x) = l2 \cdot e2(y)$ and the pushout complements CON1 and CON2 exist, $y = p2(z)$ for a *unique* $z \in \mathrm{PAR3}$ for which $h(x) = p1(z)$. Define $k: D \to \mathrm{PAR3}$ by letting $k(x) = z$. Then $p1 \cdot k(x) = p1(z) = h(x)$ and $q2 \cdot p \cdot k(x) = q2 \cdot p(z) = c2 \cdot p2(z) = c2(y) = q2 \cdot h'(x)$. Since $q2$ is injective, $p \cdot k = h'$. The uniqueness of the morphism k follows from the uniqueness of z.

In the same diagram, $q1$ is injective (since $i2$ and $s2$ are) and therefore trivially injective up to p. Also $c1$ is injective up to $p1$ since if we let $c1(x) = c1(y)$, then $r1 \cdot e1(x) = r1 \cdot e1(y)$ and hence, by the Gluing Condition on $l2$ and $i2$, $e1(x) = i2(u)$ and $e1(y) = i2(v)$ for some $u, v \in \mathrm{PAR2}$. But then, by definition of PAR3, there exist $\alpha, \beta \in \mathrm{PAR3}$ such that $p1(\alpha) = x$, $p2(\alpha) = u$ and $p2(\beta) = y$, $p2(\beta) = v$. In particular, $x, y \in p1(\mathrm{PAR3})$.

By applying Lemma 3.10, we can conclude that $\mathrm{CON1} = \mathrm{CON3} +_{\mathrm{PAR3}} \mathrm{PAR1}$. Composition of pushouts gives us $G = \mathrm{IMP3} +_{\mathrm{PAR3}} \mathrm{CON3}$. By symmetry, $H = \mathrm{EXP3} +_{\mathrm{PAR3}} \mathrm{CON3}$.

The proof of part (b) of the theorem is direct, by *defining* $\mathrm{CON1} = \mathrm{CON3} +_{\mathrm{PAR3}} \mathrm{PAR1}$ and $\mathrm{CON2} = \mathrm{CON3} +_{\mathrm{PAR3}} \mathrm{PAR2}$ and then showing that $G = \mathrm{CON1} +_{\mathrm{PAR1}} \mathrm{IMP1}$ and that $H1 = \mathrm{EXP1} +_{\mathrm{PAR1}} (\mathrm{CON3} +_{\mathrm{PAR3}} \mathrm{PAR1})$ and $H = \mathrm{EXP2} +_{\mathrm{PAR2}} (\mathrm{CON3} +_{\mathrm{PAR3}} \mathrm{PAR2})$. □

In order to define the union of derivations, we first need the notion of subproduction.

Definition 3.12 (*Category of productions*). A *production morphism* $m: \mathrm{PRO0} \to \mathrm{PRO1}$ between two productions $\mathrm{PRO}j = (\mathrm{IMP}j \leftarrow \mathrm{PAR}j \to \mathrm{EXP}j)$, $j = 0, 1$, is a triple $m = (m_I, m_P, m_E)$ of SPEC-morphisms such that the two squares in the following diagram commute

$$\begin{array}{ccccc} \mathrm{IMP0} & \xleftarrow{i0} & \mathrm{PAR0} & \xrightarrow{e0} & \mathrm{EXP0} \\ {\scriptstyle m_I}\downarrow & & \downarrow{\scriptstyle m_P} & & \downarrow{\scriptstyle m_E} \\ \mathrm{IMP1} & \xleftarrow[i1]{} & \mathrm{PAR1} & \xrightarrow[e1]{} & \mathrm{EXP1} \end{array}$$

Production morphisms can be composed (componentwise) with (id, id, id) the identity w.r.t. this composition. The category of SPEC-productions and production morphisms is denoted by CATPROD and its morphisms referred to as PROD-morphisms.

Definition 3.13 (*Amalgamation of productions*). Given a subproduction PRO0 of PRO1 and PRO2, i.e., injective PROD-morphisms $m1: \mathrm{PRO0} \to \mathrm{PRO1}$ and $m2: \mathrm{PRO0} \to \mathrm{PRO2}$, the *amalgamation* of PRO1 and PRO2 w.r.t. PRO0 is the pushout object of $m1$ and $m2$ in CATPROD, i.e., the SPEC-production

$$\mathrm{IMP1} +_{\mathrm{IMP0}} \mathrm{IMP2} \xleftarrow[i3]{} \mathrm{PAR1} +_{\mathrm{PAR0}} \mathrm{PAR2} \xrightarrow[e3]{} \mathrm{EXP1} +_{\mathrm{EXP0}} \mathrm{EXP2}$$

where the morphisms $i3$ and $e3$ are the unique ones induced by the universal property of $\mathrm{PAR1} +_{\mathrm{PAR0}} \mathrm{PAR2}$. The amalgamation is denoted by $\mathrm{PRO1} +_{\mathrm{PRO0}} \mathrm{PRO2}$.

Remarks. The amalgamation of PRO1 and PRO2 w.r.t. PRO0 is defined along the lines of the amalgamation of Graph Productions. The construction of the amalgamated production can be iterated, satisfying the usual properties of associativity and idempotency, in addition to commutativity [17].

The result of applying an amalgamation of productions can be reconstructed from the behavior of the single productions, provided that the specification it is applied to can be decomposed as a union.

Theorem 3.14 (Union of derivations). *Let* PRO0 *be a subproduction of* PROj, $j = 1, 2$ *and* $(\mathrm{PRO}j, cj): Lj \Rightarrow Rj$ *for* $j = 0, 1, 2$. *If there exist morphisms* $mj_C: \mathrm{CON0} \to \mathrm{CON}j$ *such that* $mj_C \cdot c0 = cj \cdot mj_P$, *then*

$$(\mathrm{PRO1} +_{\mathrm{PRO0}} \mathrm{PRO2}, c1 +_{c0} c2): L1 +_{L0} L2 \Rightarrow R1 +_{R0} R2.$$

Proof. To show that $\mathrm{PRO1} +_{\mathrm{PRO0}} \mathrm{PRO2}$ is applicable to $L1 +_{L0} L2$, it is sufficient to find the pushout complement of $\mathrm{IMP3} = \mathrm{IMP1} +_{\mathrm{IMP0}} \mathrm{IMP2}$ w.r.t. $L3 = L1 +_{L0} L2$. The context needed is the specification $\mathrm{CON1} +_{\mathrm{CON0}} \mathrm{CON2}$, the object of the pushout of $m1_C$ and $m2_C$. That this is the pushout complement is a consequence of the commutativity of colimits for which

$$
\begin{aligned}
L3 &= L1 +_{L0} L2 \\
&= (\mathrm{IMP1} +_{\mathrm{PAR1}} \mathrm{CON1}) +_{(\mathrm{IMP0} +_{\mathrm{PAR0}} \mathrm{CON0})} (\mathrm{IMP2} +_{\mathrm{PAR2}} \mathrm{CON2}) \\
&= (\mathrm{IMP1} +_{\mathrm{IMP0}} \mathrm{IMP2}) +_{(\mathrm{PAR1} +_{\mathrm{PAR0}} \mathrm{PAR2})} (\mathrm{CON1} +_{\mathrm{CON0}} \mathrm{CON2}) \\
&= \mathrm{IMP3} +_{\mathrm{PAR3}} \mathrm{CON3}.
\end{aligned}
$$

The context morphism $c3 : \mathrm{PAR3} \to \mathrm{CON3}$ is the unique one induced by the universal property of $\mathrm{PAR1} +_{\mathrm{PAR0}} \mathrm{PAR2}$. That the result of the direct derivation is $R1 +_{R0} R2$, is again a consequence of the commutativity of colimits [17]. □

The applicability of the productions PROj to Lj in a compatible way ($mj_C \cdot c0 = cj \cdot mj_P$) implies the applicability of the amalgamation $\mathrm{PRO1} +_{\mathrm{PRO0}} \mathrm{PRO2}$ to the union $L1 +_{L0} L2$. The converse is not true in general, unless additional conditions are imposed on the production morphism m.

Definition 3.15. A production morphism $(m_I, m_P, m_E) : \mathrm{PRO0} \to \mathrm{PRO1}$ is *consistent* if

(1) $i0(\mathrm{PAR0}) \supseteq m_I^{-1}(i1(\mathrm{PAR1}))$, $e0(\mathrm{PAR0}) \supseteq m_E^{-1}(e1(\mathrm{PAR1}))$;

(2) m_I and $i0$ satisfy the Gluing Condition and m_E is injective up to $e0$.

The term *consistent* comes from the fact that if $m_I(x)$ is an item preserved by PRO1, then x is an item preserved by PRO0. Similarly for $m_E(x)$. In [3], for graph productions, PRO0 is a subproduction of PRO1 if there exists a consistent production morphism. The easy proof of the following results is left to the reader.

Facts 3.16. (1) *If* PRO0 *is a consistent subproduction of* PROj, $j = 1, 2$, *then* PROj *is a consistent subproduction of* $\mathrm{PRO1} +_{\mathrm{PRO0}} \mathrm{PRO2}$.

(2) *If* PRO0 *is a consistent subproduction of* PRO1 *and* PRO1 *is applicable to* L *via* $l : \mathrm{IMP1} \to L$, *then* PRO0 *is applicable to* L *via* $l \cdot m_I : \mathrm{IMP0} \to L$.

The result in Theorem 3.14 deals with the application of two productions on different parts of a specification, in the case in which their overlap is exactly the subspecification to which the common subproduction is applied. With the assumptions of this theorem, if $\mathrm{IMP0} = \mathrm{PAR0} = \mathrm{EXP0}$, then the production PRO1 is applicable to $L3$ since $L3 = L1 +_{L0} L2 = \mathrm{IMP1} +_{\mathrm{PAR1}} (\mathrm{CON1} +_{\mathrm{CON0}} L2)$ using the associativity of the union of specifications [17]. The result of applying PRO1 is $(\mathrm{CON1} +_{\mathrm{CON0}} L2) +_{\mathrm{PAR1}} \mathrm{EXP1} = R1 +_{\mathrm{CON0}} L2 = R1 +_{L0} L2$. The production PRO2 can now be applied to the resulting specification since

$$
\begin{aligned}
R1 +_{L0} L2 &= R1 +_{\mathrm{CON0}} (\mathrm{CON2} +_{\mathrm{PAR2}} \mathrm{IMP2}) \\
&= (R1 +_{\mathrm{CON0}} \mathrm{CON2}) +_{\mathrm{PAR2}} \mathrm{IMP2}
\end{aligned}
$$

to yield the specification

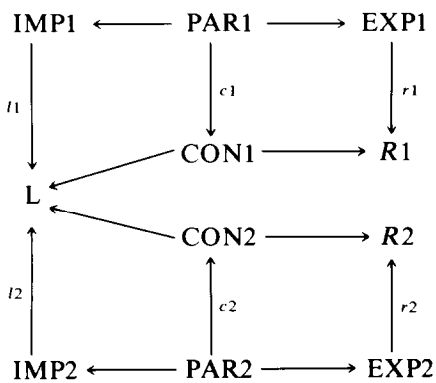
$$(R1 +_{\mathrm{CON0}} \mathrm{CON2}) +_{\mathrm{PAR2}} \mathrm{EXP2} = R1 +_{L0} R2 = R1 +_{R0} R2.$$

So the result of applying the amalgamation is the same as applying the two productions sequentially. This is the essence of the Parallelism Theorem. It relies heavily on the preservation by both productions of the common part.

Definition 3.17 (*Independence*). Two direct derivations $\mathrm{PRO1}: L \Rightarrow R1$ and $\mathrm{PRO2}: L \Rightarrow R2$ via morphism $l1$ and $l2$, respectively, are *parallel i-independent* if the intersection of IMP1 and IMP2 in L consists of common gluing items only, i.e., $l1(i1(\mathrm{PAR1})) \cap l2(i2(\mathrm{PAR2})) \supseteq l1(\mathrm{IMP1}) \cap l2(\mathrm{IMP2})$ (the inclusion in the opposite direction always holds). Parallel o-independence is defined in terms of EXPi. The derivation sequence $\mathrm{PRO1}: R1 \Rightarrow L$, $\mathrm{PRO2}: L \Rightarrow R2$ is *sequentially independent* if $\mathrm{PRO1}^{-1}$ and PRO2 are parallel i-independent.

There is an algebraic characterization of parallel independence which will be needed in the proof of the Church-Rosser Property.

Lemma 3.18. *If* $\mathrm{PRO1}: L \Rightarrow R1$ *and* $\mathrm{PRO2}: L \Rightarrow R2$ *are parallel independent, then* $l1: \mathrm{IMP1} \to L$ *and* $l2: \mathrm{IMP2} \to L$ *factor through* $s2$ *and* $s1$, *respectively.*



For an item $x \in \mathrm{IMP2}$, $l2(x) \in L$. Since $l1$ and $s1$ are jointly surjective, either

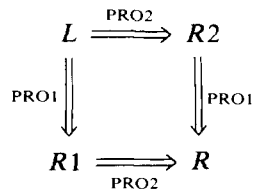
(a) $l2(x) = l1(y)$ for some $y \in \mathrm{IMP1}$ or

(b) $l2(x) = s1(z)$ for some $z \in \mathrm{CON1}$.

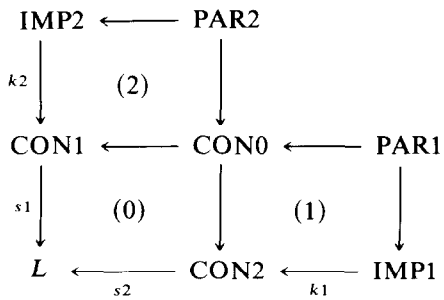
In (a), by parallel independence, there exists $w \in \mathrm{IMP1} \times_L \mathrm{IMP2}$ such that $q1(w) = y$, $q2(w) = x$. Define $k2(x) = c1(j1(w))$ with $j1: \mathrm{PAR1} \times_L \mathrm{PAR2} \to \mathrm{PAR1}$. The commutativity property is easily checked.

Theorem 3.19 (Church-Rosser property). *Given two parallel independent direct derivations* $\mathrm{PRO1}: L \Rightarrow R1$ *and* $\mathrm{PRO2}: L \Rightarrow R2$, *there exists a specification* R *such that*

PRO2: $R1 \Rightarrow R$ *and* PRO2: $R2 \Rightarrow R$ *as in the following diagram*



Proof. Consider the diagram in the previous lemma and define CON0 = CON1 $\times_L$ CON2. By the universal property of pullbacks, there exist morphisms PAR1 → CON0 and PAR2 → CON0 making the diagrams (1) and (2) commute.



By construction, (0)+(2) and (0)+(1) are pushouts. Since $s1$ and $s2$ are jointly surjective ($k2 \cdot s1 = l2$ and $s2$ are), both injective and (0) is a pullback, it is also a pushout. Therefore also (1) and (2) are pushouts.

Now define $R = (\text{EXP1} +_{\text{PAR1}} \text{CON0}) +_{\text{CON0}} (\text{CON0} +_{\text{PAR2}} \text{EXP2})$. By the associativity property of pushouts [17], we also have

$$R = (\text{EXP1} +_{\text{PAR1}} \text{CON0}) +_{\text{PAR2}} \text{EXP2}.$$

Next notice that

$$R1 = \text{EXP1} +_{\text{PAR1}} \text{CON1} = (\text{EXP1} +_{\text{PAR1}} \text{CON0}) +_{\text{CON0}} \text{CON1}$$
$$= (\text{EXP1} +_{\text{PAR1}} \text{CON0}) +_{\text{CON0}} (\text{CON0} +_{\text{PAR2}} \text{IMP2})$$
$$= (\text{EXP1} +_{\text{PAR1}} \text{CON0}) +_{\text{PAR2}} \text{IMP2}.$$

These two combined imply that PRO2: $R1 \Rightarrow R$ via the context EXP1 $+_{\text{PAR1}}$ CON0. Similarly for PRO1: $R2 \Rightarrow R$. □

Notice that PRO1 and PRO2 become parallel o-independent w.r.t. R and, by definition the derivation sequence PRO1^{-1}: $R1 \Rightarrow L$, PRO2: $L \Rightarrow R2$ is sequentially independent. Similarly for PRO1^{-1}: $R \Rightarrow R2$ and PRO2: $R1 \Rightarrow R$. The diagram obtained by replacing PRO1 with PRO1^{-1} is the graphical interpretation of the following result.

Corollary 3.20. *If* PRO: $L \Rightarrow R1$, PRO′: $R1 \Rightarrow R$ *is a sequentially independent derivation sequence, then there exists a specification* $R2$ *such that the sequence* PRO′: $L \Rightarrow R2$,

PRO: $R2 \Rightarrow R$ *is sequentially independent. Furthermore,* PRO: $L \Rightarrow R1$ *and* PRO′: $L \Rightarrow R2$ *are parallel i-independent.*

The effect of applying sequentially independent derivations can be obtained with one direct derivation using a simple amalgamation of the two productions.

Theorem 3.21 (Parallelism theorem). *If* PRO1: $L \Rightarrow R1$ *and* PRO2: $L \Rightarrow R2$ *are parallel i-independent derivations, then there exists a parallel derivation* PRO1 $+_{\text{PRO0}}$ PRO2: $L \Rightarrow R$ *(with* PRO1: $R2 \Rightarrow R$ *and* PRO2: $R1 \Rightarrow R$ *as in Theorem* 3.19) *for some subproduction* PRO0 *of* PROi, *with* IMP0 = PAR0 = EXP0.

The subproduction PRO0 of this theorem is not unique and can be chosen in the range of specifications which goes from the empty one to the pullback object of $l1$: IMP1 $\to L$ and $l2$: IMP2 $\to L$. The productions are different but, with different contexts, will produce the same specification R.

Example 3.22. Define the production PS-PRO = (PS-IMP $\leftarrow$ PS-PAR $\rightarrow$ PS-EXP) by letting

PS-PAR = PS-IMP = **bool** + sorts p^*, type, seats
opns EQP: $p^* p^* \to$ bool
eqns EQP(P^*, P^*) = TRUE

PS-EXP = PS-PAR + sorts ps
opns CREATE-PS: $\to$ ps
SEAR-PS: p^* ps $\to$ bool
RESERVE-PS: p^* type seats ps $\to$ ps
eqns SEAR-PS(P^*, CREATE-PS) = FALSE
SEAR-PS(P^*, PS) = TRUE $\Rightarrow$ RESERVE-PS(P^*, TYPE, SEATS, PS) = PS

This production PRO can be applied to the specification R of Example 3.2 via the morphism l': PS-IMP $\to R$ with $l'_S(p^*) = l'_S$(seats) = nat, l'_S(type) = str and l'_{OP}(EQP) = Eq. Since PS-IMP = PS-PAR, there is no need to check for the Gluing Conditions. The application of PS-PRO to R produces the specification

R' = **nat** + **string** + sorts fs, ps
opns CREATE-FS: $\to$ fs
SEARCH-FS: nat fs $\to$ bool
ADD-FS: nat str nat fs $\to$ fs
RETURN-FS: nat fs $\to$ dep
CHANGE-FS: nat nat fs $\to$ fs
CREATE-PS: $\to$ ps
SEAR-PS: p^* ps $\to$ bool
RESERVE-PS: p^* type seats ps $\to$ ps
eqns SEAR-PS(P^*, CREATE-PS) = FALSE

SEAR-PS(P^*, PS) = TRUE⇒RESERVE-PS(P^*, TYPE, SEATS, PS) = PS
SEARCH-FS(N, CREATE-FS) = FALSE
SEARCH-FS(N, FS) = TRUE⇒ADD-FS(N, ST1, ST2, FS) = FS
SEARCH-FS(N, FS) = TRUE
⇒RETURN-FS(N, CHANGE-FS(N, ST1, FS)) = ST1

The productions FS-PRO and PS-PRO are sequentially independent since the images of FS-EXP and PS-IMP in R overlap only in **nat**+**string**. By Corollary 3.20, they can be applied in any order. Furthermore, by the Parallelism Theorem, we can obtain R' directly from L with one direct derivation by applying the production (FS-IMP $+_{\mathbf{bool}}$ PS-IMP ← FS-PAR $+_{\mathbf{bool}}$ PS-PAR → FS-EXP $+_{\mathbf{bool}}$ PS-EXP) obtained by amalgamating PS-PRO and FS-PRO with respect to the subproduction (**bool** ← **bool** → **bool**).

4. From productions to modules

After determining, via the SPEC-productions, that the goal specification *can* be realized using the library modules, we need to specify the interconnections of the modular system, that is, we need to describe *how* the goal is realized. The modular system is to be built using the definition of applicability of a production and the way parallel, concurrent and amalgamated productions may have been constructed during the derivation.

We begin with the notion of applicability. The application of a production with a given semantical functor (such as the interfaces of a module specification) induces a unique "compatible" functor, between the semantics of the related specifications, which leaves unchanged the semantics of the context.

Theorem 4.1 (Induced functor). *Let* PRO = (IMP ← PAR → EXP) *be a* SPEC-*production and* F : Alg(IMP) → Alg(EXP) *a functor such that* $V_i = V_e \cdot F$. *If* (PRO, c) : $L \Rightarrow R$ *with* l : IMP → L *and* r : EXP → R, *then there exists a* unique *extension of* F *to a functor* $F1$: Alg(L) → Alg(R) *such that* $F \cdot V_l = V_r \cdot F1$ *and* $V_s = V_d \cdot F1$.

Proof. Let $L = \text{IMP} +_{\text{PAR}} \text{CON}$ as in Definition 3.1. By [10], every $A \in \text{Alg}(L)$ is the amalgamated sum $I +_P C$ of $I \in \text{Alg(IMP)}$, $P \in \text{Alg(PAR)}$ and $C \in \text{Alg(CON)}$ with $V_i(I) = P = V_c(C)$. Define $F1 : \text{Alg}(L) \to \text{Alg}(R)$ by letting $F1(I +_P C) = F(I) +_P C$. Then $V_d(F1(I +_P C)) = C = V_s(I +_P C)$ and $F(V_l(I +_P C)) = F(I) = V_r(F(I) +_P C)$. Such a functor is unique by the uniqueness of the amalgamated sum of algebras [10] and can be expressed as the amalgamated sum of functors $F1 = F \cdot V_l +_{V_i} V_s$. □

The applicability of a production to a specification induces a module which realizes the transformation, provided that the production consists of the interfaces of a module specification.

Theorem 4.2 (Derivation as actualization). *Let the production* PRO = (IMP ← PAR → EXP) *be composed of the interfaces and parameter part of the module specification* MOD *and let R be derivable from L via* PRO *with context* CON. *Then there exists a module specification* MOD1 *such that*

(1) *L and R are the import and export interfaces of* MOD1,

(2) *the semantics* SEM1 *of* MOD1 *is the unique functor induced as in Theorem* 4.1 *by the semantics* SEM *of* MOD.

Proof. Define PS = (CON, CON) as the parametrized specification with identity morphisms id : CON → CON and let MOD1 = act_c(PS, MOD) where c : PAR → CON is the context morphism of the derivation from L to R. By Definition 2.4, the interfaces of MOD1 are $\text{IMP} +_{\text{PAR}} \text{CON} = L$ and $\text{EXP} +_{\text{PAR}} \text{CON} = R$ while its parameter part is the parameter CON of the parametrized specification PS. Furthermore, by Definition 2.4 the semantics of MOD1 is given by $\text{SEM1}(I +_P C) = \text{SEM}(I) +_P C$, the functor induced by SEM described in Theorem 4.1. □

In order to "translate" the derivation sequences into system design, we need to be able to translate the operations on SPEC-productions into operations on the module specifications which realize the productions. The counterpart of the amalgamation of productions is the union of the modules which correspond to the single productions. The proof is immediate and is omitted.

Theorem 4.3 (Realization of amalgamation). *Let* PRO0 *be a subproduction of* PROi = (IMPi ← PARi → EXPi) *via* PROD-*morphisms* mi : PRO0 → PROi, $i = 1, 2$, *and let* PROi *be realized by the module specification* MODi. *If* MOD0 *is a submodule of* MODi *via the morphisms* mi_P, mi_I, mi_E, *and* mi_B, *then the amalgamation of productions* $\text{PRO1} +_{\text{PRO0}} \text{PRO2}$ *can be realized by the union of* MOD1 *and* MOD2 *w.r.t.* MOD0.

Theorem 4.4 (Amalgamation of induced functors). *The functor induced by the amalgamation of productions is the amalgamated sum of the individually induced functors.*

Proof. Let Fj : Alg(IMPj) → Alg(EXPj) be the functor associated with PROj = (IMPj ← PARj → EXPj) satisfying $V_{ij} = V_{ej} \cdot Fj$ and $V_{mjE} \cdot Fj = F0 \cdot V_{mjI}$. Then the functor $F3 = F1 +_{F0} F2$ of the amalgamated production induces, by Theorem 4.1, the functor

$$
\begin{aligned}
F3'(I3 +_{P3} C3) &= F3(I3) +_{P3} C3 \\
&= F3((I1 +_{I0} I2)) +_{P1 +_{P0} P2} (C1 +_{C0} C2) \\
&= [F1(I1) +_{F0(I0)} F2(I2)] +_{P1 +_{P0} P2} (C1 +_{C0} C2) \\
&= [F1(I1) +_{P1} C1] +_{[F0(I0) +_{P0} C0]} [F2(I2) +_{P2} C2] \\
&= (F1' +_{F0'} F2')(I3 +_{P3} C3)
\end{aligned}
$$

by repeated application of the subobject distributivity property of colimits [17]. □

Parallel i-independent derivations give rise to a simple special case of Theorem 4.3. If two direct derivations PRO1: $L \Rightarrow R1$ and PRO2: $L \Rightarrow R2$ are parallel i-independent, then the occurrences of IMP1 and IMP2 in L intersect only in common gluing parts, i.e., in a subspecification common to PAR1 and PAR2.

Corollary 4.5 (Union with shared subparameter). *Let* PRO1: $L \Rightarrow R1$ *and* PRO2: $L \Rightarrow R2$ *be parallel i-independent, with* PROj = (IMP$j \leftarrow$ PAR$j \rightarrow$ EXPj) *realized by the module specification* MODj, $j = 1, 2$, *and let* R *be the specification as in Theorem* 3.19. *Then there exists a specification* PAR0 *and morphisms* mj_P: PAR0 $\rightarrow$ PARj, $j = 1, 2$, *such that the derivation* $L \Rightarrow R$ *can be realized by actualizing the union* MOD1 $+_{\text{PAR0}}$ MOD2 *of* MOD1 *and* MOD2 *w.r.t. their shared subparameter part* PAR0.

The union of two module specifications with respect to a subparameter part is a special case of the general union [1]. The counterpart of the composite of two productions is the composition of the modules having the productions as interfaces.

Theorem 4.6 (Realization of composite). *Let* PRO3 *be the composite* PRO1 · PRO2 *of two productions* PROj = (IMP$j \leftarrow$ PAR$j \rightarrow$ EXPj), $j = 1, 2$. *If* MODj *realizes the production* PROj, *then the composite* PRO3 *is realized by the composition* MOD2 · MOD1.

The proof is immediate comparing the definition of composite production (Definition 3.8) with the construction of the interfaces of the composition of modules (Definition 2.5).

The main goal of this section can be stated as follows: if there is a sequence of direct derivations such that $G \Rightarrow^* H$, then there exists a module specification MOD, built *using the realizations of the productions used in the derivation*, such that INT(MOD): $G \Rightarrow H$ (where INT(MOD) is the production consisting of the interfaces and parameter part of MOD).

At the level of generality, this statement is not a theorem in that it does not take into account the correctness of the module specifications. Conditions must be imposed on the derivation sequence, similar to those imposed on module specifications to be combined correctly. The simplest case considers only derivation sequences which require union and composition (as in [2]) as the only interconnections.

Theorem 4.7. *Let* $G \Rightarrow^* H$ *via the derivation sequence* $\text{PRO}_1, \ldots, \text{PRO}_n$. *If each pair of derivations* $(\text{PRO}_i, \text{PRO}_{i+1})$ *is either sequentially independent or matched, then there exists a module specification* MOD *built using the realizations of the productions* PROi *and the operations of union, composition and actualization such that* INT(MOD): $G \Rightarrow H$.

Proof idea. For each pair of matched derivations, construct the composition of their realization, for sequentially independent pairs, their union. After replacing the sequence of productions with one production PRO (realized by the module built with composition and union of the realizations) the direct derivation PRO: $G \Rightarrow H$ provides the actual parameter CON with which to actualize the module realizing PRO. □

More general cases with partial composition and product and derivation sequences which are not sequentially independent but related via a specification will be discussed in forthcoming papers.

5. Concluding remarks

In this paper, we have proposed an approach to the development of software systems based on ideas and techniques from the algebraic theory of Graph Grammars. Given an initial specification, SPEC-productions are applied sequentially or in parallel to generate another specification: if the productions are the interfaces of module specifications from a library, then the derivation sequence can be translated into a modular system. We have seen in detail the composition and the amalgamation of productions and how to interpret direct derivations. Keeping in mind the connection between modules and productions, we should also investigate the effect that refinements, extensions and in general the vertical development of module specifications induce on the productions. The circular applicability of productions could suggest, for example, a cyclical interconnection (recursion [18]) of modules.

The problem of determining whether an arbitrary graph can be generated from an initial configuration via the productions is not only a nontrivial problem, but also an undecidable one, in general. The situation does not improve with SPEC-productions but there is hope that by restricting the class of productions (as for graphs to context-sensitive or context-free precedence grammars) the derivability problem could be solved. In any case, SPEC-productions are not intended as an automatic tool, but as an aid to the development of modular systems.

Upon discovering the impossibility of deriving the goal specification, we are interested in determining whether it is sufficient to modify one (or more) of the productions or whether a new production is needed. In the first case, we should investigate methods to detect the adequacy of a modification, to describe the altered production and to extend the modification to the rest of the module. Preliminary results have been presented in [20]. The second case presents a more difficult problem, as it requires the definition of a module, given a production. It is the essence of software development, where the interface (production) specification can be considered as a requirement specification for a module while the module specification gives its design [6]. The problem of "realizing" an interface

specification, with or without its own given semantical functor, is a problem worthy of independent investigation.

A module specification can be viewed as the description of an implementation: the body of the module provides an implementation of the sorts and operations of the export interface in terms of those of the import. The export specification is implemented (as in [22]) by the import specification via the constructor functor SEM: Alg(IMP) → Alg(EXP). The functor induced by the semantics of a module specification MOD, between Alg(SPEC) and Alg(SPEC′) when SPEC′ is derived from SPEC via the interfaces of MOD (as in Theorem 4.1), can also be viewed as an implementation functor. In general the approach is useful for other notions of implementation, using functors from Alg(IMP) to Alg(EXP) other than the semantics of modules, for the constructors in the sense of [22]. The development process using their notion of implementation corresponds to a sequence of direct derivations. The approach stresses the fact that the development is independent of the realization of the interfaces and needs only a "translation" of the sequence of direct derivations into an "interconnection" of their realizations.

Acknowledgment

I am indebted to H. Ehrig for a long and lively discussion on the contents of this paper, to A. Tarlecki for useful suggestions on an earlier draft and to the anonymous referees for their pointed criticisms. This research was partially supported by CNR-Italy under "Progetto Finalizzato: Sistemi Informativi e Calcolo Parallelo" and by M.P.I. 40%.

References

[1] E.K. Blum and F. Parisi-Presicce, The semantics of shared submodule specification, in: *Proc. CAAP 85/TAPSOFT 85*, Lecture Notes in Computer Science **185** (Springer, Berlin, 1985) 359-373.

[2] E.K. Blum, H. Ehrig and F. Parisi-Presicce, Algebraic specification of modules and their basic interconnections, *J. Comput. System Sci.* **34**(2/3) (1987) 293-339.

[3] P. Boehm, H.-R. Fonio and A. Habel, Amalgamation of graph transformations with applications to synchronization, in: *Proc. TAPSOFT 85*, Lecture Notes in Computer Science **185** (Springer, Berlin, 1985) 267-283.

[4] R.M. Burstall and J.A. Goguen, Putting theories together to make specifications, in: *Proc. 5th Internat. Conf. on Artificial Intelligence* (1977) 1045-1058.

[5] H. Ehrig, *Introduction to the Algebraic Theory of Graph Grammars*, Lecture Notes in Computer Science **73** (Springer, Berlin, 1979) 1-69.

[6] H. Ehrig, W. Fey, H. Hansen, M. Lowe and F. Parisi-Presicce, Algebraic theory of modular specification development, TUB Report 87-06.

[7] H. Ehrig, W. Fey, F. Parisi-Presicce and E.K. Blum, Algebraic theory of module specifications with constraints in: *Proc. MFCS*, Lecture Notes in Computer Science **233** (Springer, Berlin, 1986) 59-77.

[8] H. Ehrig, A. Habel and B.K. Rosen, Concurrent transformations of relational structures, *Fund. Inform.* **IX**(1) (1986).

[9] H. Ehrig, H.-J. Kreowski, A. Maggiolo-Schettini, B.K. Rosen and J. Winkowski, Transformation of structures: an algebraic approach, *Math. Systems Theory* **14** (1981) 305-334.

[10] H. Ehrig and B. Mahr, Fundamentals of algebraic specifications 1: equations and initial semantics, in: *EATCS Monographs on Theoret. Comp. Sci.*, vol 6 (Springer, Berlin, 1985).

[11] H. Ehrig, M. Nagl and G. Rozenberg, eds., *Graph Grammars and their Applications to Computer Science*, Lecture Notes in Computer Science **153** (Springer, Berlin, 1983).

[12] H. Ehrig, M. Nagl, G. Rozenberg and A. Rosenfeld, eds., *Graph Grammars and their Applications to Computer Science*, Lecture Notes in Computer Science **291** (Springer, Berlin, 1987).

[13] H. Ehrig, M. Pfender and H.J. Schneider, Graph grammars: an algebraic approach, in: *Proc. IEEE Conf. SWAT 73*, Iowa City (1973) 167-180.

[14] H. Ehrig and B.K. Rosen, Parallelism and concurrency of graph manipulations, *Theoret. Comput. Sci.* **11** (1980) 247-275.

[15] H. Ehrig and H. Weber, Algebraic specification of modules, in: E.J. Neuhold and G. Chronist, eds., *Formal Models in Programming* (North-Holland, Amsterdam, 1985).

[16] J.A. Goguen and J. Meseguer, Universal realization, persistent interconnection and implementation of abstract modules, in: *Proc. ICALP 82*, Lecture Notes in Computer Science **140** (Springer, Berlin, 1982) 265-281.

[17] F. Parisi-Presicce, Inner and mutual compatibility of operations on module specifications, in: *Proc. CAAP 88*, Lecture Notes in Computer Science **214** (Springer, Berlin, 1986) 30-44; full version, Tech. Rep. 86-06, Tech. Univ. Berlin, 1986.

[18] F. Parisi-Presicce, Partial composition and recursion of module specifications, in: *Proc. TAPSOFT 87*, Lecture Notes in Computer Science **249** (Springer, Berlin, 1987) 217-231.

[19] F. Parisi-Presicce, Product and iteration of module specifications, in: *Proc. CAAP 88*, Lecture Notes in Computer Science **299** (Springer, Berlin, 1988) 149-164.

[20] F. Parisi-Presicce, A rule-based approach to modular system design, in: *Proc. 12th ICSE*, Nice, France (1990) 202-211.

[21] B.K. Rosen, Deriving graphs by applying a production, *Acta Inform.* **4** (1975) 337-357.

[22] D. Sannella and A. Tarlecki, Toward formal development of programs from algebraic specifications: implementation revisited, in: *Proc. TAPSOFT 87*, Lecture Notes in Computer Science **249** (Springer, Berlin, 1987) 96-110.