PUC

ISSN 0103-9741

Monografias em Ciência da Computação
nº 33/96

On Modularity under Internal and External Choice in Formal Software Development

Paulo A. S. Veloso
Sheila R. M. Veloso
José L. Fiadeiro

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, Nº 33/96
Editor: Carlos J. P. Lucena October, 1996

On Modularity under Internal and External Choice in Formal Software Development *

Paulo A. S. Veloso [1]
Sheila R. M. Veloso [2]
José L. Fiadeiro [3]

* This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

[1] On leave at Universidade Federal do Rio de Janeiro.

[2] Instituto de Matemática and Programa de Eng. Sistemas, COPPE, Univ. Federal do Rio de Janeiro.

[3] Depto. Informática, Faculdade de Ciências, Universidade de Lisboa.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio — Departamento de Informática
Rua Marquês de São Vicente, 225 — Gávea
22453-900 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br

On Modularity under Internal and External Choice in Formal Software Development

Paulo A. S. VELOSO, Sheila R. M. VELOSO AND José L. FIADEIRO

PUCRioInf MCC 33/96

Abstract. We examine some interactions between internal and external choices performed by agents in the context of modular software development of (versions of) programs from formal specifications. We emphasise implementations of collections of specifications, which may be viewed as describing versions of programs, and the use of labels to tag them and their development paths. The concept of implementation, as interpretation into a conservative extension, is generalised to labelled collections of specifications and formulated in categorical terms. We then show that the category of such collections has pushouts and that this construction preserves modularity as required for composing implementations.

Key words: Software development, formal specifications, implementation, modularity, pushouts, versions of programs, labelled collections, internal and external choices, angelic and demonic nondeterminism.

Resumo. Examinamos algumas interações entre escolhas internas e externas efetuadas por agentes no contexto de desenvolvimento modular de (versões de) programas a partir de especificações formais. Enfatizamos implementações de coleções de especificações, estas podendo ser encaradas como descrevendo versões de programas, e o emprego de rótulos para sua identificação e de suas trajetórias de desenvolvimento. O conceito de implementação, como interpretação em uma extensão conservativa, é generalizado a coleções rotuladas de especificações e formulado em termos categórios. Mostramos então que a categoria dessas coleções tem somas amalgamadas (pushouts) e que esta construção preserva modularidade conforme requirido para compor implementações.

Palavras chave: Desenvolvimento de programas, especificações formais, implementação, modularidade, somas amalgamadas (pushouts), versões de programas, coleções rotuladas, escolhas internas e externas, não determinismos angélico e demoníaco.

CONTENTS

1. Introduction

In this paper we examine some interactions between internal and external choices performed by agents in the context of modular software development of (versions of) programs from formal specifications. We emphasise implementations of collections of specifications, which may be viewed as describing versions of programs, and the use of labels to tag them and their development paths. The concept of implementation, as interpretation into a conservative extension, is generalised to labelled collections of specifications and formulated in categorical terms. We then show that the category of such collections has pushouts and that this construction preserves modularity as required for composing implementations.

Each agent in a team developing (versions of) programs faces several design decisions. One generally considers that the developer may take any particular set of design decisions as long as the requirements are met. This corresponds to an external choice (under control of the agent), and also to angelic nondeterminism (a successful path is enough). Now, consider a collection of (versions of) programs where the choice of the version is not under control of the designers. Then, every version in the collection should be developed. This corresponds to an internal choice, and also to demonic nondeterminism.

We examine here modularity in formal development of software versions and some relationships with internal and external choices. The structure of this paper is as follows. We begin by motivating collective software development and recalling some background about modular formal software development. We then proceed to formalise collections of specifications and their collective development, which motivates the introduction of labels for their versions. A categorical approach becomes then natural. These labelled collections of specifications and their interpretations form a category, which is shown to have pushouts, whose modularity (as preservation of faithfulness or conservativeness) is seen to be inherited from that of the individual specifications. Then we examine the role of labels in achieving compositionality of collective implementations and conclude with some remarks about connections with other works and possible extensions.

2. Motivation: collective software development

One often has versions of requirement specifications for a system that may lead to several versions of programs. Sometimes the one has not enough information to select between the alternative specifications or programs before developing them. This leads naturally to the question of developing collections of (versions of) specifications or programs.

Consider the following scenario. A company is in the process of producing a system. Some versions of specifications, forming a collection, are to be

developed into a corresponding collection of (versions of) programs implementing them. An agent, a developer or a team of developers, will be in charge of this development. Another agent, called the decision maker, will select which version will actually be turned into a product. We consider that this decision about which version to select is not to be made before the development (perhaps because it will depend on the development process or on its outcome).

In developing (versions of) programs the developer faces several design decisions. One generally considers that the developer may take any particular set of design decisions as long as the requirements are met. This corresponds to an external choice (under control of the developer), and also to angelic nondeterminism (a successful path is enough).

Consider a collection of (versions of) programs where the choice of the version is not under control of the designer. Then, every version in the collection must be developed. This corresponds to an internal choice, and also to demonic nondeterminism.

We examine here the interaction between these two kinds of choices in the context of modular software development from formal specifications. To emphasise that we are dealing with several versions of programs, and thus that a specification may be developed along different paths, we use labels: we will deal with labelled collection of specifications. Labels are widely used nowadays: as traces, time stamps, evaluations of confidence, etc.

3. Background: formal software development

The stepwise approach to formal software development [Maibaum, Veloso & Sadler '85; Turski & Maibaum '87] advocates that the specification of the procedures to be implemented be given as a theory presentation in a suitable logic. The starting point of this process of program development is the formal specification of the desired behaviour by a set of sentences (a theory presentation). Each development step consists in the implementation of a formal specification A in terms of another formal specification C, which is assumed to be closer to the intended programming environment.

Such an implementation step consists of an extension of C with new symbols corresponding to those of A and new axioms which specify properties of these new symbols. This gives rise to a new formal specification B. Since one does not wish to disturb the given concrete specification C, this extension B should not impose any new constraints on C. This can be formulated by requiring the extension from C to B to be conservative, in the sense that B adds no new consequence to C in the signature of the latter.

One then wishes to correlate the abstract symbols in A to corresponding ones in B. But, the properties of A are important, for instance in guaranteeing the correctness of an abstract program supported by A.

Thus, in translating from A to B, one wishes to preserve the properties of A as given by its axioms. This can be formulated by requiring the translation from A to B to be an interpretation of theories, in the sense that it translates each consequence of A into a consequence of B.

We thus arrive at the concept of an *implementation* (triangle) of A on C as an interpretation of A into a conservative extension B (a mediating specification) of C [Maibaum, Veloso & Sadler '85; Turski & Maibaum '87], as depicted in figure 3.1.

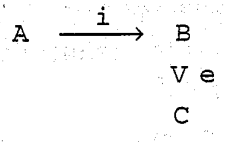


Figure 3.1: Canonical implementation step

A useful tool for program development is the structuring of a specification into a "context" and "parameter", i.e. a parameterised specification [Ehrig & Mahr '85]. The idea is that the context (e. g. SEQ[DATA]) can be plugged into different situations by appropriate choice of values (instantiation) for the parameters (e.g. giving rise to SEQ[NAT], etc.).

The simple tools of conservative extension and interpretations between theories provide us with a quite straightforward account of parameterisation. A specification S is said to be *parameterised* by a sub-specification X (called *parameter*) whenever S is a conservative extension of X, and a *parameter instantiation* is an interpretation p of X into an actual argument Y [Maibaum, Veloso & Sadler '85; Turski & Maibaum '87].

In defining the result of a parameter instantiation, one has to complete a diagram of two specifications B and D with a common source C, which is extended to B and interpreted into D, to a rectangle. The completion of the diagram is achieved by the pushout construction [Arbib & Mannes '75; Mac Lane '71], and the preservation of conservativeness is guaranteed by the Modularisation Theorem [Veloso & Maibaum '95; Veloso '96]. This situation is illustrated in figure 3.2.

Parameter instantiation Modularisation Theorem

Figure 3.2: Completing a rectangle of interpretations

In composing implementations one faces a similar situation of completing a rectangle, so as to preserve conservativeness. This construction gives a mediating specification for the composite implementation, as illustrated in figure 3.3. (Here, preservation of conservativeness is important for ensuring that the pushout specification indeed serves as a mediating

specification for the composite implementation.

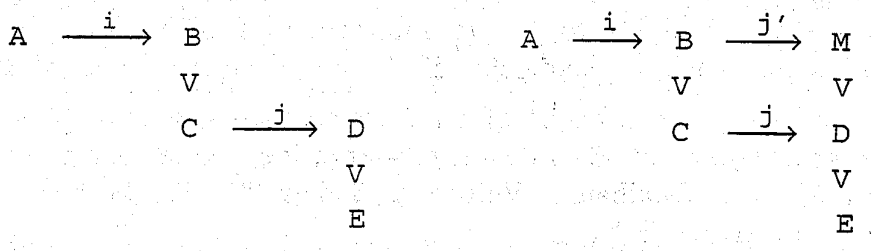


Figure 3.3: Composition of implementation steps

We now recall some concepts from formal specifications and logic [Enderton '72; Ehrig & Mahr '85; Shoenfield '67; Turski & Maibaum '87]. We consider a *signature* K as characterised by its alphabet of extra-logical (predicate and function) symbols together with syntactical declarations. A *specification* is a theory presentation, i. e. a pair $S=\langle K,\Sigma\rangle$, consisting of a signature K and a set Σ of sentences of K (its *axioms*), generating its *theory* Cn(S) (which consists of the consequences of its axioms).

Given signatures I and K, a *translation* $t:I\rightarrow K$ is a syntax-preserving signature morphism mapping each symbol of I to a corresponding symbol in K of the same kind; so each formula of I is translated to a formula of K. An *interpretation* from $P=\langle I,\Gamma\rangle$ to $R=\langle K,\Theta\rangle$ is a translation $t:I\rightarrow K$ that translates Cn(P) into Cn(R). A *faithful interpretation* $t:P\rightarrow R$ is one such that, for every sentence σ of I, $\sigma\in Cn(P)$ iff $t(\sigma)\in Cn(R)$.

We call I a *sub-signature* of K (denoted by $I\subseteq K$) when K can be obtained from I by adding some extra-logical symbols (and declarations); we then have an inclusion translation $j:I\rightarrow K$. We say that R *extends* P (*conservatively*) iff $I\subseteq K$ and the inclusion $j:I\rightarrow K$ interprets P into R (faithfully).

4. Formalisation: collections and labels

We wish to formalise implementation of versions of specifications and programs. The objects we have to deal with are collective specifications: collections of specifications over the same underlying signature.

In our scenario we assume internal choice within the collection and external choice along the development. So, an implementation of collective specifications should provide an implementation triangle for each version of the abstract specification in the collection.

Consider collective specifications $\mathcal{a}$ and $\mathcal{c}$, over the respective signatures L and K. A collective specification implementation of $\mathcal{a}$ on $\mathcal{c}$ should provide, for each version a of $\mathcal{a}$, a mediating specification b conservatively extending some version c of $\mathcal{c}$ together with an interpretation $t:a\rightarrow b$. In view of the situation, it is quite reasonable to consider mediating specifications over the same underlying signature I and interpretations $t:a\rightarrow b$ with same underlying signature translation

$t: L \to I$. Then, the mediating specifications will also form a collective specification $\mathbf{b}$ and it is natural to explicate an implementation of collective specifications in terms of collective interpretations into collective conservatively extensions.

Given collective specifications $\mathbf{a}$ and $\mathbf{b}$, with respective signatures L and I, a *collective interpretation* of $\mathbf{a}$ into $\mathbf{b}$ amounts to a translation $t: L \to I$ of their underlying signatures, such that for each version b of $\mathbf{b}$, there exists a version a of $\mathbf{a}$ which t interprets into b. A *collective extension* is a collective interpretation whose underlying signature translation is an inclusion; an extension is *collectively conservative* when each version b of $\mathbf{b}$ is a conservative extension of its corresponding version a in $\mathbf{a}$.

Now, in view of our motivation, it is quite natural to regard the various versions within a collective specification as tagged by labels identifying them. We are thus led to the idea of labelled collective specification as a family $(\mathbf{e}_\varepsilon)_{\varepsilon \in E}$ of specifications over a common signature K indexed by a set E of labels. (The set of labels, in addition to tagging the versions, can also serve to record the overall design strategies concerning the development of each version.)

A labelling consists of a set of labels together with a mapping assigning to each label a specification over a common signature. Thus, a *labelled collective specification* $\mathbf{e}$ can be described by the data (see figure 4.1):

- a set $\mathcal{L}(\mathbf{e})$ (its *underlying label set*),
- a signature $\mathcal{S}(\mathbf{e})$ (its *common underlying signature*),
- a mapping $\mathbf{e}[\_]$ assigning to each label $\varepsilon \in \mathcal{L}(\mathbf{e})$ a specification over $\mathcal{S}(\mathbf{e})$.

Label		Specification		Signature
$\varepsilon \in \mathcal{L}(\mathbf{e})$	$\Rightarrow$	$\mathbf{e}[\varepsilon]$	over	$\mathcal{S}(\mathbf{e})$

Figure 4.1: Labelled collective specification

Now, consider labelled collective specifications $\mathbf{a}$ and $\mathbf{b}$. A *labelled collective interpretation* from $\mathbf{a}$ to $\mathbf{b}$ is accordingly a pair $\langle p,s \rangle$ where

$p: \mathcal{L}(\mathbf{b}) \to \mathcal{L}(\mathbf{a})$ is a relabelling of their underlying label sets,

$s: \mathcal{S}(\mathbf{a}) \to \mathcal{S}(\mathbf{b})$ is a signature translation between their signatures;

such that (see figure 4.2)

- for every label β of $\mathcal{L}(\mathbf{b})$ translation s interprets $\mathbf{a}[p(\beta)]$ into $\mathbf{b}[\beta]$, i. e. $s: \mathbf{a}[p(\beta)] \to \mathbf{b}[\beta]$.

$$\left\langle \begin{array}{ccc} \mathcal{L}(\mathbf{a}) & & \mathcal{S}(\mathbf{a}) \\ p \uparrow & , & \downarrow s \\ \mathcal{L}(\mathbf{b}) & & \mathcal{S}(\mathbf{b}) \end{array} \right\rangle : \begin{array}{ccc} p(\beta) \in \mathcal{L}(\mathbf{a}) & & \mathbf{a}[p\,\beta)] \\ p \uparrow & \Rightarrow & s \downarrow \\ \beta \in \mathcal{L}(\mathbf{b}) & & \mathbf{b}[\beta] \end{array}$$

Figure 4.2: Labelled collective interpretation

We thus formalise collective specifications essentially as mappings from set of labels to specifications over a common signature. So, we are naturally led to a categorical approach, which is quite appropriate for

several aspects. We shall in the sequel regard labelled collective specifications as forming a category and examine some of its categorical properties and constructions.

5. Categorical approach

We consider for label sets the category of sets and functions. We thus use for label sets a category **LBL** that has pullbacks [Arbib & Mannes '75; Mac Lane '71].

A specification is a presentation over an underlying signature. So, we consider categories **SGNT**, of *signatures*, and **SPEC**, of *specifications*, together with a functor *underlying signature* $\mathcal{U}(\_)$:**SPEC**$\rightarrow$**SGNT**. Thus, each interpretation of specifications $\tau:\Gamma\rightarrow\Delta$ in **SPEC** has, as usual, an underlying signature translation $\mathcal{U}(\tau):\mathcal{U}(\Gamma)\rightarrow\mathcal{U}(\Delta)$ in **SGNT** (see figure 5.1).

$$\begin{array}{ccc} \Gamma & & \mathcal{U}(\Gamma) \\ \tau\downarrow & \xrightarrow{\ \mathcal{U}\ } & \downarrow\mathcal{U}(\tau) \\ \Delta & & \mathcal{U}(\Delta) \end{array}$$

Figure 5.1: Functor underlying signature $\mathcal{U}(\_)$:**SPEC**$\rightarrow$**SGNT**)

A *labelled (specification) object* e consists of

- a set $\mathcal{L}(e)$ (called its *underlying label set*),
- a mapping e[_] assigning to each label ε of $\mathcal{L}(e)$ a specification in **SPEC**.

We say that such an object e is *collective* when all the specifications involved are over the same signature. A *labelled collective object* is a labelled object e such that for every ε' and ε'' in $\mathcal{L}(e)$ $\mathcal{U}(e[\varepsilon'])=\mathcal{U}(e[\varepsilon''])$. In other words, the composite mapping $e;\mathcal{U}:\mathcal{L}(e)\rightarrow$**SGNT** is a constant, which we call its *common underlying signature* $\mathcal{S}(e)$ (see figure 5.2).

$$\begin{array}{lcl} \varepsilon'\in\mathcal{L}(e) & e[\varepsilon'] \xrightarrow{\ \mathcal{U}\ } & S'' \\ & \Rightarrow & \| \\ \varepsilon''\in\mathcal{L}(e) & e[\varepsilon''] \xrightarrow{\ \mathcal{U}\ } & S'' \end{array} \qquad \begin{array}{ccc} \mathcal{L}(e) & \xrightarrow{\ e\ } & \textbf{SPEC} \\ & \searrow^{\mathcal{S}} & \downarrow\mathcal{U} \\ & & \textbf{SGNT} \end{array}$$

Figure 5.2: Labelled collective object with common underlying signature

We now consider interpretations between labelled collective specifications as morphisms between labelled collective objects.

Consider labelled collective objects a and b. A *labelled collective morphism* from a to b is a pair $\langle p,s\rangle$ where (see figure 5.3)

$p:\mathcal{L}(b)\rightarrow\mathcal{L}(a)$ is a morphism in **LBL**, relabelling the underlying label sets,
$s:\mathcal{S}(a)\rightarrow\mathcal{S}(b)$ is a morphism in **SGNT**, between the underlying signatures;

subject to the following requirement:

- for every label β in $\mathcal{L}(b)$ there exists a morphism $s_\beta:a[p(\beta)]\rightarrow b[\beta]$ in **SPEC** whose underlying translation $\mathcal{U}(s_\beta)$ is $s:\mathcal{S}(a)\rightarrow\mathcal{S}(b)$.

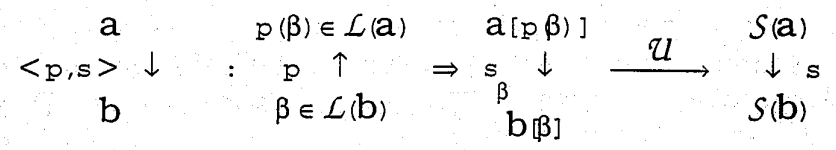


Figure 5.3: Labelled collective morphism $\langle p,s\rangle : a \to b$

The appropriate componentwise composition of labelled collective morphisms is easily seen to yield such a morphism (see figure 5.4). The composite $\langle p,s\rangle;\langle q,t\rangle := \langle q;p,s;t\rangle$ of $\langle p,s\rangle : a \to b$ and $\langle q,t\rangle : b \to c$ is a morphism from a to c (with $s;t_\gamma : a[p(q(\gamma))] \to c[\gamma]$ as the composite $s_{q(\gamma)};t_\gamma$ of $s_{q(\gamma)} : a[p(q(\gamma))] \to b[q(\gamma)]$ and $t_\gamma : b[q(\gamma)] \to c[\gamma]$, since it has underlying signature translation $\mathcal{U}(s_{q(\gamma)};t_\gamma) = \mathcal{U}(s_{q(\gamma)});\mathcal{U}(t_\gamma) = s;t$).

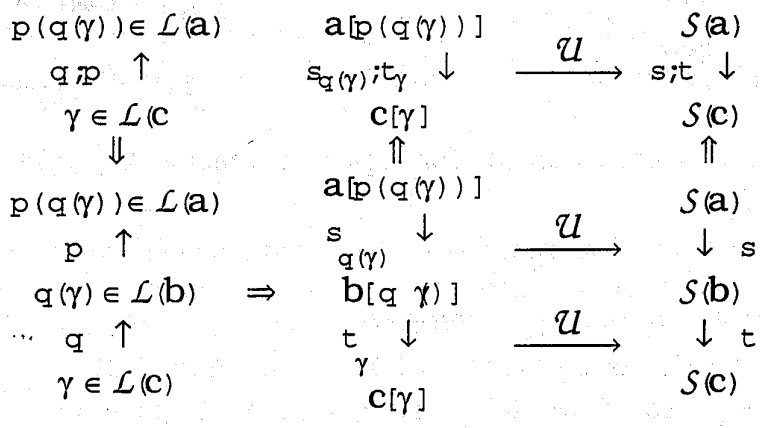


Figure 5.4: Composition of labelled collective morphisms

One can see that we thus have a category **LCS** of *labelled collective specifications*. It is clear that underlying label set and signature become functors $\mathcal{L}(\_)$ and $\mathcal{S}(\_)$ from **LCS** into $\mathbf{LBL}^{op}$ and **SGNT**, i. e the former is contravariant and the latter is covariant (see figure 5.5).

Figure 5.5: Functors $\mathcal{L}(\_):\mathbf{LCS} \to \mathbf{LBL}^{op}$ and $\mathcal{S}(\_):\mathbf{LCS} \to \mathbf{SGNT}$

6. Categorical constructions

We assume that category **SGNT** of signatures has pushouts. We shall place two requirements on the functor $\mathcal{U}(\_):\mathbf{SPEC} \to \mathbf{SGNT}$. The first one corresponds to requiring an interpretation of specifications to be characterised by its underlying signature translation. The second one enables transferring pushouts from **SGNT** to **SPEC**.

Faithfulness

Functor $\mathcal{U}(\_):\mathbf{SPEC} \to \mathbf{SGNT}$ is faithful:

if $\mathcal{U}(\mu) = \mathcal{U}(\nu)$ in **SGNT** then $\mu = \nu$ in **SPEC** (see figure 6.1).

In other words, given a pair of morphisms $\mu:\Gamma\to\Delta$ and $\nu:\Gamma\to\Delta$ with common source and target in **SPEC**, if their images under $\mathcal{U}$ are the same morphism $\mathcal{U}(\mu)=\mathcal{U}(\nu):\mathcal{U}(\Gamma)\to\mathcal{U}(\Delta)$ in **SGNT**, then $\mu=\nu:\Gamma\to\Delta$ in **SPEC**.

$$\begin{array}{ccc} \Gamma & = & \Gamma \\ \mu\downarrow & & \downarrow\nu \\ \Delta & = & \Delta \end{array} \xrightarrow{\ \mathcal{U}\ } \begin{array}{ccc} \mathcal{U}(\Gamma) & = & \mathcal{U}(\Gamma) \\ \mathcal{U}(\mu)\downarrow & = & \downarrow\mathcal{U}(\nu) \\ \mathcal{U}(\Delta) & = & \mathcal{U}(\Delta) \end{array} \quad \Rightarrow \quad \mu=\nu$$

Figure 6.1: Faithfulness of signature functor $\mathcal{U}(\_):$**SPEC**$\to$**SGNT**

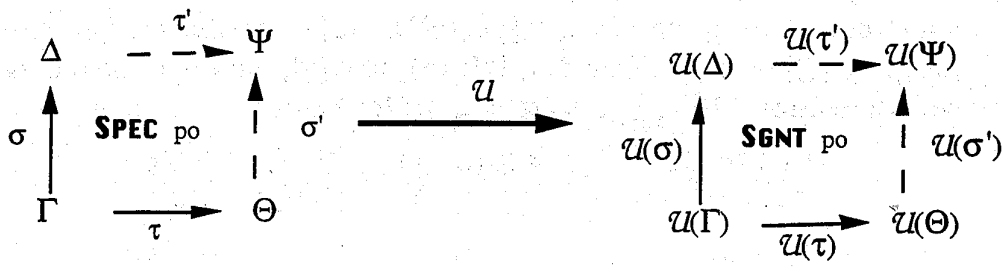


Figure 6.2: Pushout transfer for signature functor $\mathcal{U}(\_):$**SPEC**$\to$**SGNT**

Pushout transfer

In **SPEC** each pair of morphisms $\sigma:\Gamma\to\Delta$ and $\tau:\Gamma\to\Theta$ with common source Γ has a pushout $\sigma':\Delta\to\Psi$ and $\tau':\Gamma\to\Psi$ with common target Ψ, so that the rectangle $\mathcal{U}(\sigma):\mathcal{U}(\Gamma)\to\mathcal{U}(\Delta)$, $\mathcal{U}(\tau):\mathcal{U}(\Gamma)\to\mathcal{U}(\Theta)$, $\mathcal{U}(\sigma'):\mathcal{U}(\Delta)\to\mathcal{U}(\Psi)$ and $\mathcal{U}(\tau'):\mathcal{U}(\Gamma)\to\mathcal{U}(\Psi)$ is a pushout in **SGNT** (see figure 6.2).

In other words(see figure 6.3), given a pushout rectangle $s:I\to J$, $t:I\to K$, $s':K\to L$ and $t':J\to L$ in **SGNT**, for each pair of morphisms $\sigma:\Gamma\to\Delta$ and $\tau:\Gamma\to\Theta$ with common source Γ in **SPEC**, such that $\mathcal{U}(\sigma)=s$ and $\mathcal{U}(\tau)=t$, there exist an object Ψ and morphisms $\sigma':\Delta\to\Psi$ and $\tau':\Gamma\to\Psi$ with common target Ψ in **SPEC**, so that

($\mathcal{U}$) $\mathcal{U}(\sigma')=s$ and $\mathcal{U}(\tau')=t$ with $\mathcal{U}(\Psi)=L$;

(po) $\sigma:\Gamma\to\Delta$, $\tau:\Gamma\to\Theta$, $\sigma':\Delta\to\Psi$ and $\tau':\Gamma\to\Psi$ is a pushout rectangle in **SPEC**.

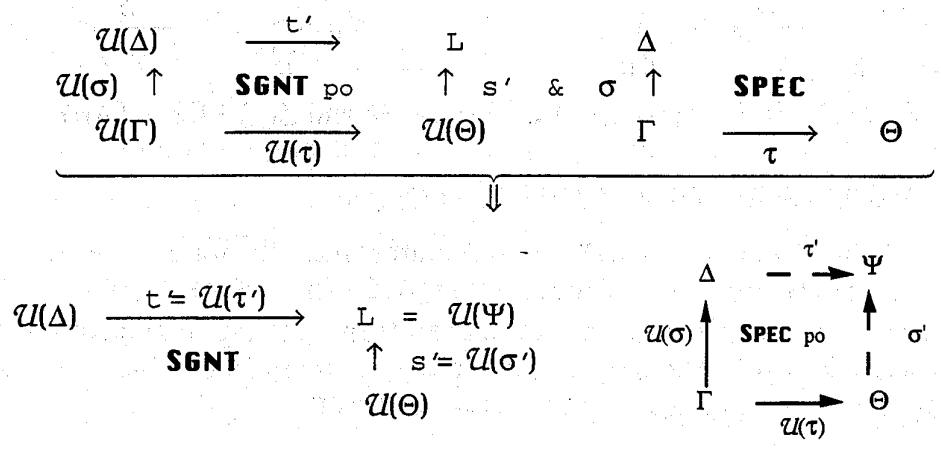


Figure 6.3: Pushout transfer from **SGNT** to **SPEC** via $\mathcal{U}(\_):$**SPEC**$\to$**SGNT**

Since **SGNT** is assumed to have pushouts, the latter requirement for faithful $\mathcal{U}$ is seen to be guaranteed by the following joint translation

transfer condition.

Joint translation transfer (see figure 6.4)

Given a pair of specifications Δ and Θ in **SPEC**, for each pair of morphisms $s':\mathcal{U}(\Delta)\rightarrow L$ and $t':\mathcal{U}(\Theta)\rightarrow L$ with common target L in **SGNT**, there exists a specification Ψ and a pair of morphisms $\sigma':\Delta\rightarrow\Psi$ and $\tau':\Theta\rightarrow\Psi$ in **SPEC**, such that

($\mathcal{U}$) $\mathcal{U}(\sigma')=s'$ and $\mathcal{U}(\tau')=t'$ with $\mathcal{U}(\Psi)=L$;

(up) given a pair of morphisms $\mu:\Delta\rightarrow\Xi$ and $\nu:\Theta\rightarrow\Xi$ in **SPEC**, for each morphism $k:L\rightarrow\mathcal{U}(\Xi)$ such that $t';k=\mathcal{U}(\mu)$ and $s';k=\mathcal{U}(\nu)$ in **SGNT**, there exists a morphism $\chi:\Psi\rightarrow\Xi$ in **SPEC** such that $\mathcal{U}(\chi)=k$.

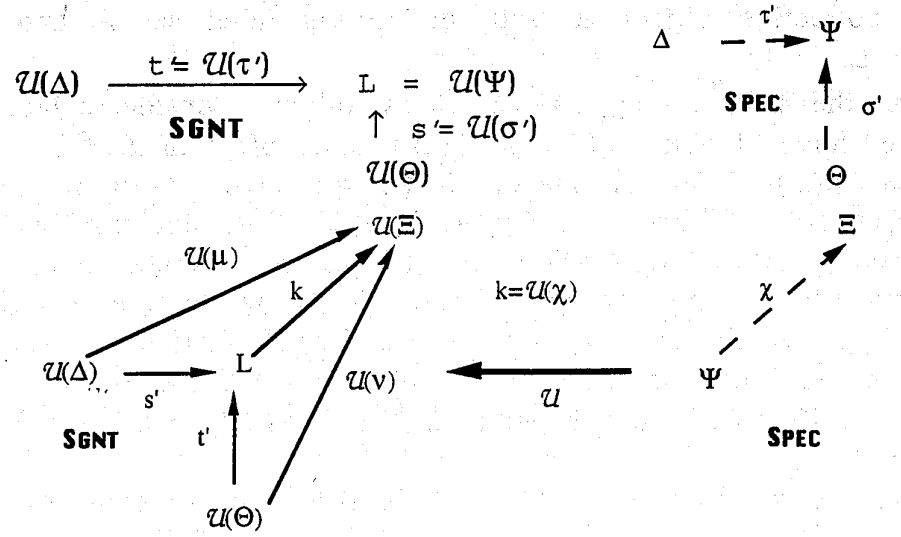


Figure 6.4: Joint translation transfer condition for $\mathcal{U}(\_)$:**SPEC**$\rightarrow$**SGNT**

We now wish to show that the category **LCS** of labelled collective specifications has pushouts. For this purpose, consider a pair of labelled collective morphisms $\langle p,s\rangle:c\rightarrow b$ and $\langle q,t\rangle:c\rightarrow d$ with common source c.

We first construct a labelled collective object a and morphisms $\langle 'p,s'\rangle:d\rightarrow a$ and $\langle 'q,t'\rangle:b\rightarrow a$, such that the rectangle $\langle p,s\rangle:c\rightarrow b$, $\langle q,t\rangle:c\rightarrow d$, $\langle 'q,t'\rangle:b\rightarrow a$, and $\langle 'p,s'\rangle:d\rightarrow a$ commutes in the category **LCS** of labelled collective specifications (see figure 6.5).

We construct them by using the pullback in **LBL** and the pushout in **SGNT**, as well as the pushout transfer from **SGNT** to **SPEC**

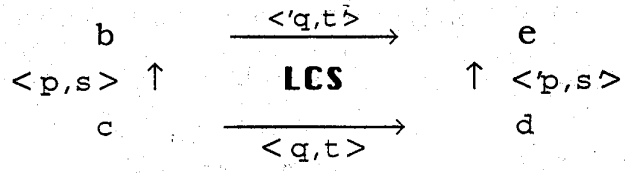


Figure 6.5: Commutative rectangle $\langle p,s\rangle;\langle 'q,t'\rangle=\langle q,t\rangle;\langle 'p,s'\rangle$ in **LCS**

In category **LBL** we have underlying morphisms $p:\mathcal{L}(b)\rightarrow\mathcal{L}(c)$ and $q:\mathcal{L}(d)\rightarrow\mathcal{L}(c)$ with common target $\mathcal{L}(c)$. We form their pullback yielding morphisms $'p:A\rightarrow\mathcal{L}(d)$ and $'q:A\rightarrow\mathcal{L}(b)$ with common source A in **LBL**.

In category **SGNT** we have underlying morphisms $s:\mathcal{S}(c)\to\mathcal{S}(b)$ and $t:\mathcal{S}(c)\to\mathcal{S}(d)$ with common source $\mathcal{S}(c)$. We form their pushout yielding morphisms $s':\mathcal{S}(d)\to L$ and $t':\mathcal{S}(b)\to L$ with common target L in **SGNT**.

$$\begin{array}{ccc} \mathcal{L}(b) & \xleftarrow{'q} & A \\ p\downarrow & \textbf{LBL pb} & \downarrow\, 'p \\ \mathcal{L}(c) & \xleftarrow[q]{} & \mathcal{L}(d) \end{array} \qquad \begin{array}{ccc} \mathcal{S}(b) & \xrightarrow{t'} & L \\ s\uparrow & \textbf{SGNT po} & \uparrow\, s' \\ \mathcal{S}(c) & \xrightarrow[t]{} & \mathcal{S}(d) \end{array}$$

Figure 6.6: Label set and signature for pushout in category **LCS**

We thus have a label set A and signature L. We now wish to construct a labelled collective object a with underlying label set A and common signature L.

We define the mapping $a[\_]:A\to$ **SPEC** as follows. Consider a label α in A. We then have labels $'q(\alpha)$ in $\mathcal{L}(b)$ and $'p(\alpha)$ in $\mathcal{L}(d)$, such that $p('q(\alpha))=q('p(\alpha))$. Thus, in category **SPEC** we have underlying morphisms $s_{'q(\alpha)}:c[\gamma]\to b['q(\alpha)]$ and $t_{'p(\alpha)}:c[\gamma]\to d['p(\alpha)]$ with common source $c[\gamma]$, where $\gamma:=p('q(\alpha))=q('p(\alpha))$. Now, by the pushout transfer requirement on functor underlying signature $\mathcal{U}(\_)$:**SPEC**$\to$**SGNT**, we have a specification Ψ and a pair of morphisms $\tau:b['q(\alpha)]\to\Psi$ and $\sigma:c['p(\alpha)]\to\Psi$ with common target Ψ in **SPEC** such that $\mathcal{U}(\Psi)=L$ as well as $\mathcal{U}(\tau)=t'$ and $\mathcal{U}(\sigma)=s'$. We now set $a[\alpha]:=\Psi$. We thus have a mapping $a[\_]:A\to$ **SPEC** (see figure 6.7).

$$\begin{array}{ccc} 'q(\alpha) & \xleftarrow{'q} & \alpha \\ p\downarrow & = & \downarrow\, 'p \\ \gamma & \xleftarrow[q]{} & 'p(\alpha) \end{array} \qquad \begin{array}{ccc} b['q(\alpha)] & \xrightarrow{\tau} & \Psi \\ s_{'q(\alpha)}\uparrow & \textbf{SPEC po} & \uparrow\, \sigma \\ c[\gamma] & \xrightarrow[t_{'p(\alpha)}]{} & d['p(\alpha)] \end{array}$$

Figure 6.7: Mapping $a[\_]:A\to$ **SPEC** for pushout in category **LCS**

The constructed a is a labelled collective object with underlying label set $\mathcal{L}(a):=A$ and common signature L, because for every label α in $\mathcal{L}(a):=A$, we have $\mathcal{U}(a[\alpha])=L$. So a has common underlying signature $\mathcal{S}(a)=L$.

We now claim that the pairs $\langle 'p,s'\rangle$ and $\langle 'q,t'\rangle$ are labelled collective morphisms $\langle 'p,s'\rangle:d\to a$ and $\langle 'q,t'\rangle:b\to a$. Indeed, given any label α in $\mathcal{L}(a):=A$, we have by construction morphisms $\tau:b['q(\alpha)]\to a[\alpha]$ and $\sigma:d['p(\alpha)]\to a[\alpha]$ with common target $a[\alpha]:=\Psi$ in **SPEC** such that $\mathcal{U}(\tau)$ is $t':\mathcal{S}(b)\to\mathcal{S}(a)$ and $\mathcal{U}(\sigma)$ is $s':\mathcal{S}(d)\to\mathcal{S}(a)$; so we set $t'_\alpha:=\tau$ and $s'_\alpha:=\sigma$.

$$\begin{array}{ccc} 'q(\alpha)\in\mathcal{L}(b) & b[\,'q(\alpha)\,] & \mathcal{S}(b) \\ 'q\uparrow & \Rightarrow\ t'_\alpha=\tau\downarrow & \xrightarrow{\mathcal{U}} \quad \downarrow t \\ \alpha\in\mathcal{L}(a) & a[\alpha] & \mathcal{S}(a) \end{array}$$

$$\begin{array}{ccc} 'p(\alpha)\in\mathcal{L}(d) & d[\,'p(\alpha)\,] & \mathcal{S}(d) \\ 'p\uparrow & \Rightarrow\ s'_\alpha=\sigma\downarrow & \xrightarrow{\mathcal{U}} \quad \downarrow s \\ \alpha\in\mathcal{L}(a) & a[\alpha] & \mathcal{S}(a) \end{array}$$

Figure 6.8: Labelled collective morphisms in **LCS**

We thus have a pair $\langle 'p,s' \rangle : d \to a$ and $\langle 'q,t' \rangle : b \to a$ of labelled collective morphisms with common target a in **LCS**.

The commutativity $\langle p,s \rangle ; \langle 'q,t' \rangle = \langle q,t \rangle ; \langle 'p,s' \rangle$ now follows from the commutativity of the component rectangles $'q;p='p;q$ in **LBL** and $s;t'=t;s'$ in **SGNT** (see figure 6.9).

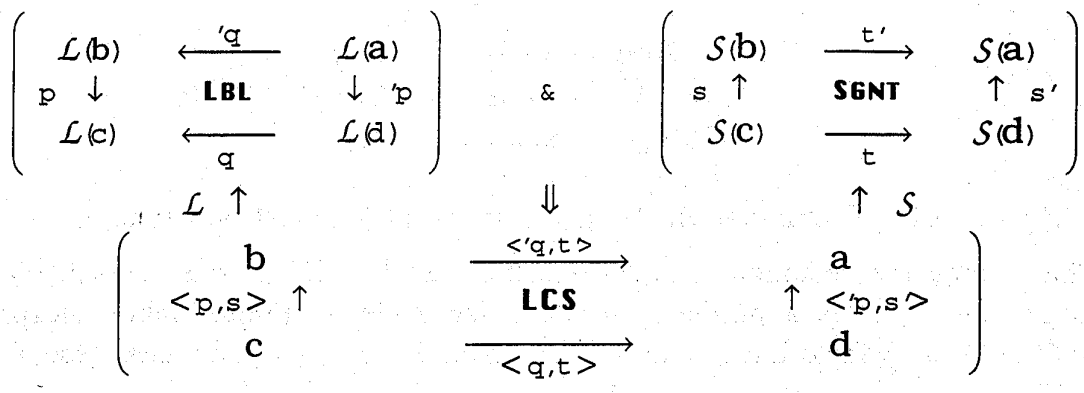


Figure 6.9: Commutativity transfer to **LCS** via functors $\mathcal{L}(\_)$ and $\mathcal{S}(\_)$

Thus, from a pair of labelled collective morphisms $\langle p,s \rangle : c \to b$ and $\langle q,t \rangle : c \to d$ with common source c in **LCS**, we have constructed a pair $\langle 'p,s' \rangle : d \to a$ and $\langle 'q,t' \rangle : b \to a$ of labelled collective morphisms with common target a in **LCS**, such that the rectangle $\langle p,s \rangle : c \to b$, $\langle q,t \rangle : c \to d$, $\langle 'q,t' \rangle : b \to a$, and $\langle 'p,s' \rangle : d \to a$ commutes in **LCS**, as shown in figure 6.5.

We now examine the universal property of the construction.

For this purpose, we consider a pair $\langle f,i \rangle : b \to e$ and $\langle g,j \rangle : d \to e$ of morphisms with common target e in **LCS** such that $\langle p,s \rangle ; \langle f,i \rangle = \langle q,t \rangle ; \langle g,j \rangle$, as depicted in figure 6.10. We wish to show the unique common factorisation of morphisms $\langle f,i \rangle$ and $\langle g,j \rangle$ via some morphism in **LCS**.

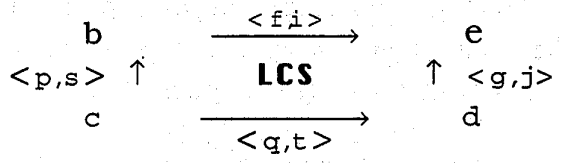


Figure 6.10: Commutative diagram $\langle p,s \rangle ; \langle f,i \rangle = \langle q,t \rangle ; \langle g,j \rangle$ in **LCS**

Towards this goal, we use the functors underlying label set and signature as well as the universal properties of our construction.

We apply the functors $\mathcal{L}(\_) : \mathbf{LCS} \to \mathbf{LBL}^{op}$ and $\mathcal{S}(\_) : \mathbf{LCS} \to \mathbf{SGNT}$ to the above rectangle in **LCS**. We will obtain commutative rectangles in **LBL** and in **SGNT**. Then the universal properties of the pullback in **LBL** and of the pushout in **SGNT**, respectively, will yield the components of the desired morphism in **LCS**.

Since underlying label set is a (contravariant) functor $\mathcal{L}(\_) : \mathbf{LCS} \to \mathbf{LBL}^{op}$, the commutativity $\langle p,s \rangle ; \langle f,i \rangle = \langle q,t \rangle ; \langle g,j \rangle$ in **LCS** yields the commutativity $f;p=g;q$ in **LBL** (see figure 6.11).

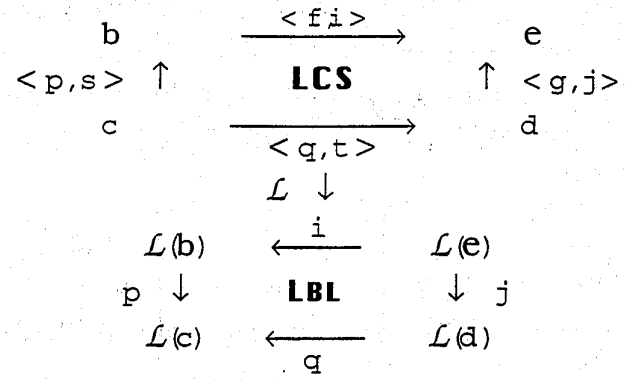


Figure 6.11: Commutativity transfer from **LCS** to **LBL** via functor $\mathcal{L}(\_)$

Thus, since the rectangle $p:\mathcal{L}(b)\rightarrow\mathcal{L}(c)$, $q:\mathcal{L}(d)\rightarrow\mathcal{L}(c)$, $'q:\mathcal{L}(a)\rightarrow\mathcal{L}(b)$ and $'p:\mathcal{L}(a)\rightarrow\mathcal{L}(d)$ is a pullback in **LBL**, we have a unique label morphism $h:\mathcal{L}(e)\rightarrow\mathcal{L}(a)$ factorizing f and g, i. e. h;'q=f and h;'p=g, in **LBL** (see figure 6.12).

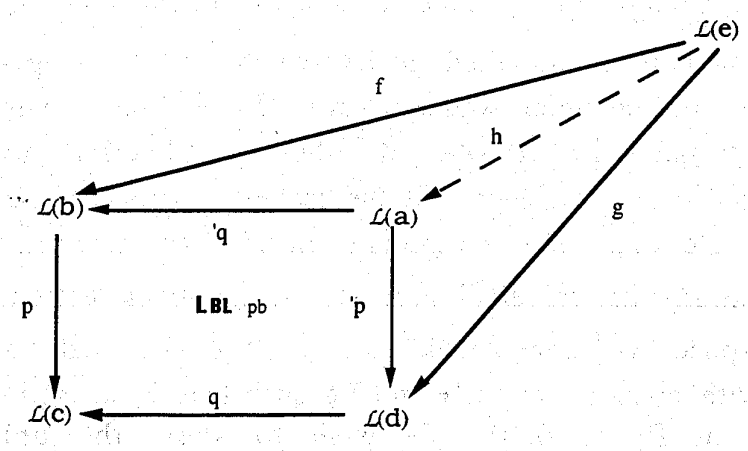


Figure 6.12: Pullback property of label set in category **LBL**

Similarly, since underlying common signature is a (covariant) functor $\mathcal{S}(\_):\mathbf{LCS}\rightarrow\mathbf{SGNT}$, the commutativity <p,s>;<f,i>=<q,t>;<g,j> in **LCS** yields the the commutativity s;i=t;j in **SGNT** (see figure 6.13).

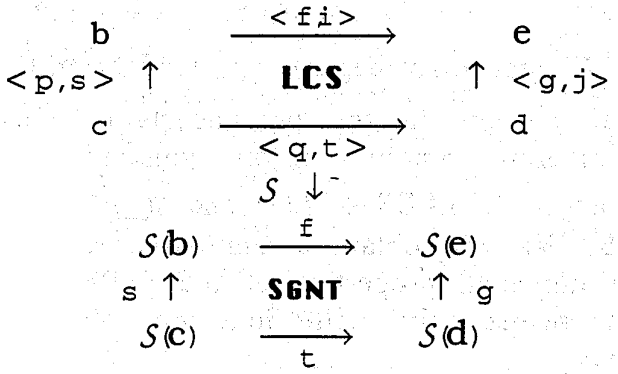


Figure 6.13: Commutativity transfer from **LCS** to **SGNT** via functor $\mathcal{S}(\_)$

Thus, since the rectangle $s:\mathcal{S}(c)\rightarrow\mathcal{S}(b)$, $t:\mathcal{S}(c)\rightarrow\mathcal{S}(d)$, $s':\mathcal{S}(d)\rightarrow\mathcal{S}(a)$ and $t':\mathcal{S}(b)\rightarrow\mathcal{S}(a)$ is a pushout in **SGNT**, we have a unique signature morphism

k:$\mathcal{S}(a)\rightarrow\mathcal{S}(e)$ factorising i and j, i. e. s';k=j and t';k=i, in **SGNT** (see figure 6.14).

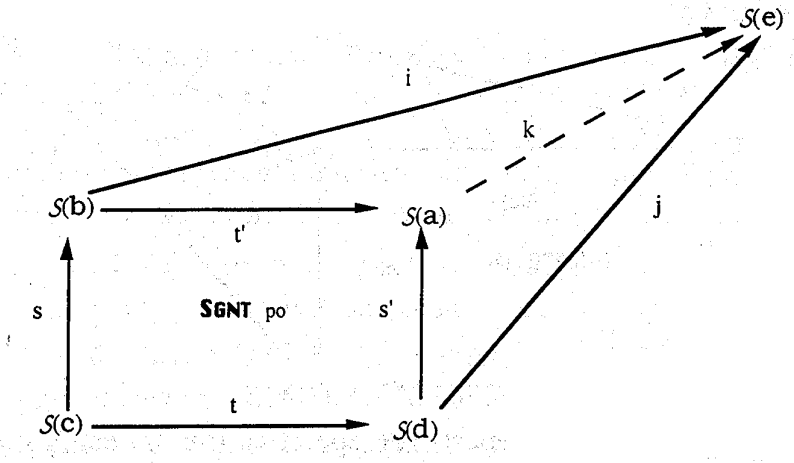


Figure 6.14: Pushout property of signature in category **SGNT**

We now claim that the pair <h,k> is a labelled collective morphism in the category **LCS** of labelled collective specifications.

To establish this claim, we must verify the requirement for every label ε in $\mathcal{L}(e)$ there exists a morphism k_ε:a[h(ε)]→e[ε] in **SPEC** whose underlying translation $\mathcal{U}(k_\varepsilon)$ is the given k:$\mathcal{S}(a)\rightarrow\mathcal{S}(e)$.

Indeed, given any label ε in $\mathcal{L}(e)$, we have i_ε:b['q(h((ε))]→e[ε] and j_ε:d['p(h((ε))]→e[ε] in **SPEC** with underlying signature translations $\mathcal{U}(i_\varepsilon)$=i and $\mathcal{U}(j_\varepsilon)$=j in **SGNT**. Now, in **SGNT**, we have the commutativity s;i=t;j i. e. $\mathcal{U}(s_{'q(h(\varepsilon))});\mathcal{U}(i_\varepsilon)=\mathcal{U}(t_{'p(h(\varepsilon))});\mathcal{U}(j_\varepsilon)$; by the faithful requirement on the functor $\mathcal{U}(\_)$:**SPEC**→**SGNT**, $\mathcal{U}(s_{'q(h(\varepsilon))};i_\varepsilon)=\mathcal{U}(t_{'p(h(\varepsilon))};j_\varepsilon)$ yields the commutativity $s_{'q(h(\varepsilon))};i_\varepsilon=t_{'p(h(\varepsilon))};j_\varepsilon$ in **SPEC** (see figure 6.15).

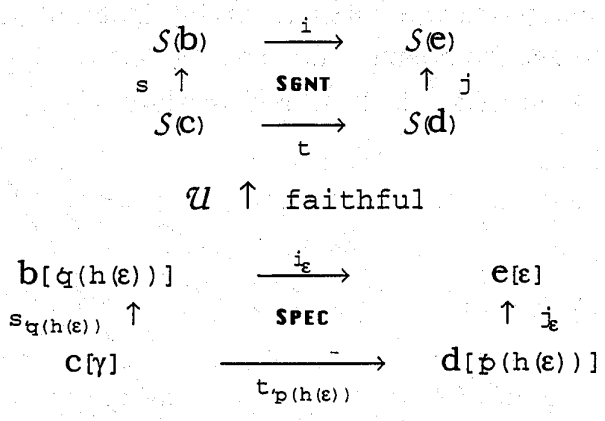


Figure 6.15: Transfer of commutativity from **SGNT** to **SPEC** via $\mathcal{S}(\_)$

Now, the rectangle with morphisms $s_{'q(h(\varepsilon))}$:c[γ]→b['q(h(ε))], $t_{'p(h(\varepsilon))}$:c[γ]→d['p(h(ε))], $s'_{h(\varepsilon)}$:d['p(h(ε))]→a[h(ε)] and $t'_{h(\varepsilon)}$:b['q(h(ε))]→a[h(ε)], where γ:=p('q(h(ε)))=q('p(h(ε))), is a pushout in **SPEC**. Thus we have a (unique) morphism χ:a[h(ε)]→e[ε] factorising i_ε and j_ε, i. e. $t'_{h(\varepsilon)};\chi=i_\varepsilon$ and $s'_{h(\varepsilon)};\chi=j_\varepsilon$, in **SPEC** (see figure 6.16).

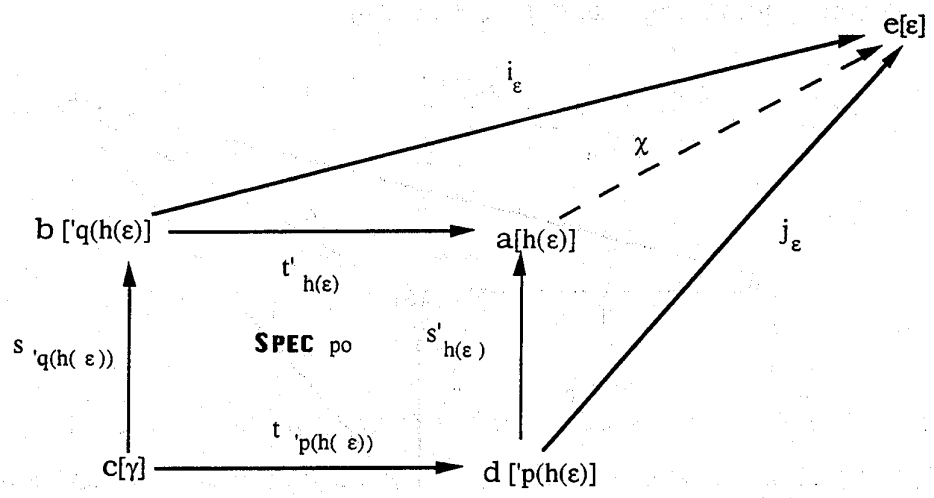


Figure 6.16: Pushout property of specification in category **SPEC**

Now, since underlying common signature is a functor $\mathcal{S}(\_)$:**LCS**$\to$**SGNT**, we also have the commutativities $\mathcal{U}(t'_{h(\varepsilon)});\mathcal{U}(\chi)=\mathcal{U}(i_\varepsilon)$ and $\mathcal{U}(s'_{h(\varepsilon)});\mathcal{U}(\chi)=\mathcal{U}(j_\varepsilon)$ i. e. $t';\mathcal{U}(\chi)=i_\varepsilon$ and $s';\mathcal{U}(\chi)=j_\varepsilon$, in **SGNT**. Thus, by the uniqueness of $k:\mathcal{S}(a)\to\mathcal{S}(e)$ as a factorisation of i and j in **SGNT**, we have $\mathcal{U}(\chi)=k$. Hence, by setting $k_\varepsilon:=\chi$ we have a morphism $k_\varepsilon:a[h(\varepsilon)]\to e[\varepsilon]$ in **SPEC** such that $\mathcal{U}(k_\varepsilon)=k:\mathcal{S}(a)\to\mathcal{S}(e)$. Thus $\langle h,k\rangle:a\to e$ is a labelled collective morphism in **LCS** (see figure 6.17).

$$\begin{array}{ccccc} h(\varepsilon)\in\mathcal{L}(a) & & a[h(\varepsilon)] & & \mathcal{S}(a) \\ h\uparrow & \Rightarrow & k_\varepsilon:=\chi\downarrow & \xrightarrow{\mathcal{U}} & \downarrow k \\ \varepsilon\in\mathcal{L}(e) & & e[\varepsilon] & & \mathcal{S}(e) \end{array}$$

Figure 6.17: Labelled collective mediator morphism

This argument shows the existence of a labelled collective mediator morphism $\langle h,k\rangle:a\to e$ factorising $\langle f,i\rangle$ and $\langle g,j\rangle$, i. e. $\langle 'q,t'\rangle;\langle h,k\rangle=\langle f,i\rangle$ and $\langle 'p,s'\rangle;\langle h,k\rangle=\langle g,j\rangle$, in **LCS**, as illustrated in figure 6.18).

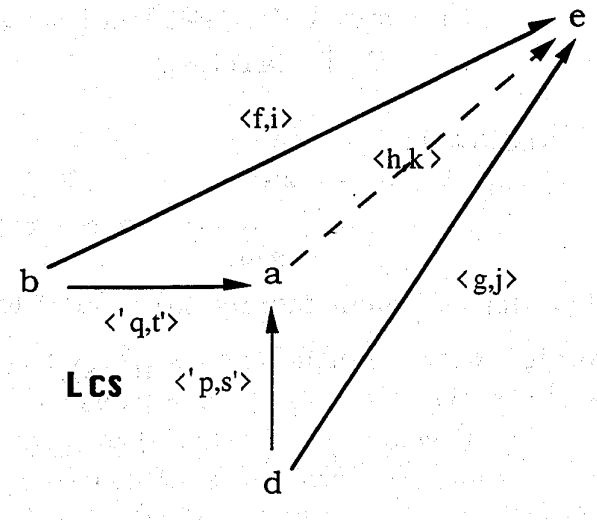


Figure 6.18: Mediator morphism $\langle h,k\rangle:a\to e$ in **LCS**

The uniqueness of such a factorisation follows from the uniqueness of the components and the functorial nature of $\mathcal{L}(\_):\mathbf{LCS}\rightarrow\mathbf{LBL}^{op}$ and $\mathcal{S}(\_):\mathbf{LCS}\rightarrow\mathbf{SGNT}$. Indeed, consider any morphism $\langle m,n\rangle:a\rightarrow e$ such that $\langle 'q,t'\rangle;\langle m,n\rangle=\langle f,i\rangle$ and $\langle 'p,s'\rangle;\langle m,n\rangle=\langle g,j\rangle$, in **LCS**. By applying the functors $\mathcal{L}(\_):\mathbf{LCS}\rightarrow\mathbf{LBL}^{op}$ and $\mathcal{S}(\_):\mathbf{LCS}\rightarrow\mathbf{SGNT}$, we have the commutativities $m;'q=f$ and $m;'p=g$ in **LBL** as well as $t';n=i$ and $s';n=j$ in **SGNT**. By uniqueness of such factorisations, the former yields $m=h$ and the latter yields $n=k$. We therefore have $\langle m,n\rangle=\langle h,k\rangle$ in **LCS** as claimed.

Summing up, in **LCS**, from a pair of labelled collective morphisms $\langle p,s\rangle:c\rightarrow b$ and $\langle q,t\rangle:c\rightarrow d$ with common source c, we can construct a pair $\langle 'p,s'\rangle:d\rightarrow a$ and $\langle 'q,t'\rangle:b\rightarrow a$ of labelled collective morphisms with common target a, such that the rectangle $\langle p,s\rangle:c\rightarrow b$, $\langle q,t\rangle:c\rightarrow d$, $\langle 'q,t'\rangle:b\rightarrow a$, and $\langle 'p,s'\rangle:d\rightarrow a$ commutes (as shown in figure 6.5) and has the universal property of unique factorisation (see figures 6.10 and 6.18).

This finishes the proof that the category **LCS** of labelled collective specifications has pushouts.

Theorem *Pushout in labelled collective specifications*
The category **LCS** of labelled collective specifications has pushouts.

7. Modularity

We now examine modularity: preservation of faithfulness or conservativeness by the pushout construction.

We endow category **LCS** of labelled collective specifications with a concept of faithful morphism naturally inherited from category **SPEC** of specifications as follows. A labelled collective morphism $\langle p,s\rangle:a\rightarrow b$ in **LCS** will be called *faithful* iff for every label β in $\mathcal{L}(b)$ the morphism $s_\beta:a[p(\beta)]\rightarrow b[\beta]$ is faithful in **SPEC** (see figure 7.1).

$$\begin{array}{ccccc} a & & p(\beta)\in\mathcal{L}(a) & & a[p\,\beta)] \\ \text{faithful}\downarrow & \langle p,s\rangle & : \quad p\ \uparrow & \Rightarrow & \text{faithful}\downarrow\ s_\beta \\ b & & \beta\in\mathcal{L}(b) & & b[\beta] \end{array}$$

Figure 7.1: Faithful labelled collective morphism $\langle p,s\rangle:a\rightarrow b$ in **LCS**

By our assumptions category **SPEC** of specifications has pushouts. We shall place a further requirement on category **SPEC** of specifications: that its pushout preserves faithfulness. (This requirement is satisfied by the presentations in classical first-order logic [Veloso '96].)

Modularity of pushouts in **SPEC** (see figure 7.3)
Given a pushout rectangle $\sigma:\Gamma\rightarrow\Delta$, $\tau:\Gamma\rightarrow\Theta$, $\sigma':\Delta\rightarrow\Psi$ and $\tau':\Gamma\rightarrow\Psi$ in **SPEC**, if morphism $\sigma:\Gamma\rightarrow\Delta$ is faithful then so is its pushout $\sigma':\Delta\rightarrow\Psi$.

We now wish to show that the pushouts in the category **LCS** of labelled collective specifications preserve faithfulness. Consider a pushout

rectangle in **LCS**: pair of morphisms $\langle p,s\rangle : c \to b$ and $\langle q,t\rangle : c \to d$ with common source c yielding a pair $\langle 'p,s'\rangle : d \to a$ and $\langle 'q,t'\rangle : b \to a$ of morphisms with common target a. Assuming that $\langle p,s\rangle : c \to b$ is faithful we claim that so is $\langle 'p,s'\rangle : d \to a$.

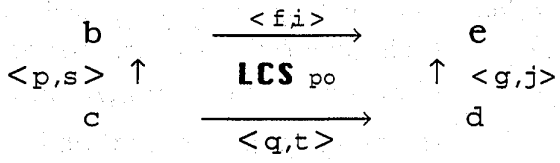


Figure 7.2: Pushout rectangle in category **LCS**

Consider a label α in $\mathcal{L}(a)$. By construction, we have a pushout rectangle in **SPEC**: a pair of morphisms $s'_{q(\alpha)} : c[\gamma] \to b['q(\alpha)]$ and $t'_{p(\alpha)} : c[\gamma] \to d['p(\alpha)]$ with common source $c[\gamma]$, where $\gamma := p('q(\alpha)) = q('p(\alpha))$, and a pair $t'_\alpha : b['q(\alpha)] \to a[\alpha]$ and $s'_\alpha : d['p(\alpha)] \to a[\alpha]$ of morphisms with common target a. Now, since $\langle p,s\rangle : c \to b$ is faithful in **LCS**, morphism $s'_{q(\alpha)} : c[\gamma] \to b['q(\alpha)]$ is a faithful interpretation in **SPEC**, and modularity in **SPEC** yields the faithfulness of $s'_\alpha : d['p(\alpha)] \to a[\alpha]$ (see figure 7.3). So, morphism $\langle 'p,s'\rangle : d \to a$ is faithful in **LCS**.

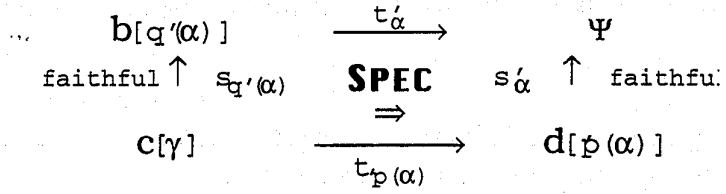


Figure 7.3: Pushout modularity in **SPEC**

Proposition *Preservation of faithfulness by pushouts in* **LCS**
Pushouts in the category **LCS** of labelled collective specifications preserve faithfulness.

A line of reasoning similar to the preceding one shows that the pushouts in the category **LCS** of labelled collective specifications inherit preservation of extensions from the underlying category **SPEC** of specifications.

We endow category **LCS** of labelled collective specifications with a concept of extension naturally inherited from category **SPEC** of specifications as before. A labelled collective morphism $\langle p,s\rangle : a \to b$ in **LCS** is called an *extension* iff for every label β in $\mathcal{L}(b)$ the morphism $s_\beta : a[p(\beta)] \to b[\beta]$ is an extension $s_\beta : a[p(\beta)] \sqsubseteq b[\beta]$ in **SPEC** (see figure 7.4).

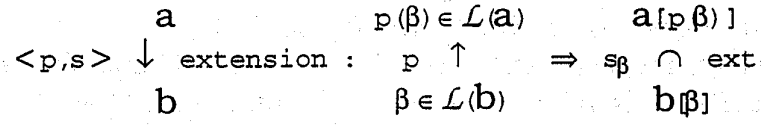


Figure 7.4: Labelled collective extension $\langle p,s\rangle : a \to b$ in **LCS**

We shall now consider a further requirement on category **SPEC** of specifications: that its pushout preserves extensions. (This requirement is satisfied by the presentations in classical first-order logic [Veloso & Maibaum '95].)

Extension modularity in **SPEC** (see figure 7.5)
Given a pushout rectangle $\sigma:\Gamma\rightarrow\Delta$, $\tau:\Gamma\rightarrow\Theta$, $\sigma':\Delta\rightarrow\Psi$ and $\tau':\Gamma\rightarrow\Psi$ in **SPEC**, if morphism $\sigma:\Gamma\rightarrow\Delta$ is an extension then so is its pushout $\sigma':\Delta\rightarrow\Psi$.

We can now argue that the pushouts in the category **LCS** of labelled collective specifications preserve extensions. Given a pushout rectangle in **LCS**: pair of morphisms $\langle p,s\rangle:c\rightarrow b$ and $\langle q,t\rangle:c\rightarrow d$ with common source c yielding a pair $\langle 'p,s'\rangle:d\rightarrow a$ and $\langle 'q,t'\rangle:b\rightarrow a$ of morphisms with common target a, with $\langle p,s\rangle:c\rightarrow b$ an extension, we claim that so is $\langle 'p,s'\rangle:d\rightarrow a$.

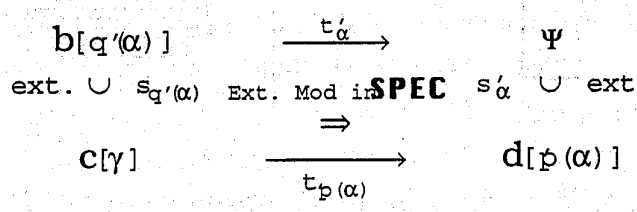


Figure 7.5: Extension modularity in **SPEC**

As before, consider a label α in $\mathcal{L}(a)$. By construction, we have a pushout rectangle in **SPEC**: a pair of morphisms $s'_{q(\alpha)}:c[\gamma]\rightarrow b['q(\alpha)]$ and $t'_{p(\alpha)}:c[\gamma]\rightarrow d['p(\alpha)]$ with common source $c[\gamma]$, where $\gamma:=p('q(\alpha))=q('p(\alpha))$, and a pair $t'_\alpha:b['q(\alpha)]\rightarrow a[\alpha]$ and $s'_\alpha:d['p(\alpha)]\rightarrow a[\alpha]$ of morphisms with common target a. Now, since $\langle p,s\rangle:c\rightarrow b$ is an extension in **LCS**, morphism $s'_{q(\alpha)}:c[\gamma]\rightarrow b['q(\alpha)]$ is an extension in **SPEC**, and extension modularity in **SPEC** yields that $s'_\alpha:d['p(\alpha)]\rightarrow a[\alpha]$ is an extension (see figure 7.5). So, morphism $\langle 'p,s'\rangle:d\rightarrow a$ is an extension in **LCS**.

Corollary *Preservation of conservative extensions by pushouts in* **LCS**
Pushouts in the category **LCS** of labelled collective specifications preserve conservative extensions.

8. The role of labels in modular development

We now examine more closely the role of labels for the modular development of labelled collective specifications.

First of all, labels are a useful tool for keeping track of the various versions and for establishing the existence of pushouts.

Now, given a collective specification e there are several ways of labelling it. A possible way is taking the collection e itself as its label set, with each specification version e of e as a tag for itself. In general, one may have other labelling schemes. Let us examine some possibilities.

Consider a labelled collective specification e with label set $\mathcal{L}(e)$. We regard each label ε of $\mathcal{L}(e)$ as tagging a specification version $e[\varepsilon]$. Thus,

labelled collective specification e with mapping e[_] describes a collection of specifications, namely its image e[$\mathcal{L}$(e)]. Let us examine their local behaviour: some specifications may have no label while others may have several labels.

A specification version v that is not in the image of the mapping e[_] has no label assigned to it, and we may view this as an indication that it need not be developed.

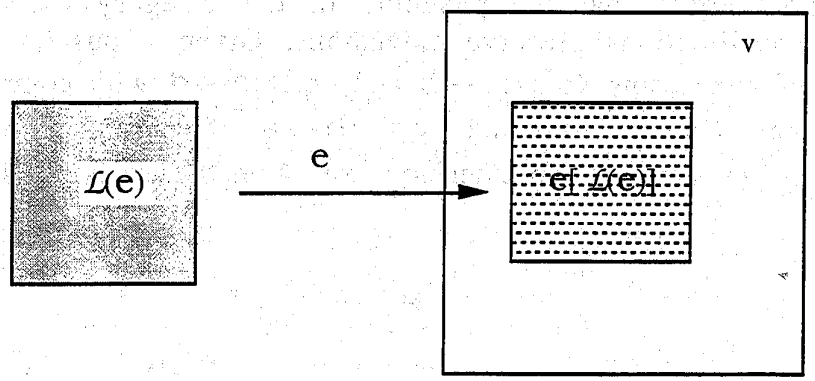


Figure 8.1: Specification version without label

A specification version v that has distinct labels assigned to it, say ε' and ε" in $\mathcal{L}$(e) with e[ε']=v=e[ε"], may be viewed as having two versions, which may, or may not, follow distinct development paths along the corresponding fibres.

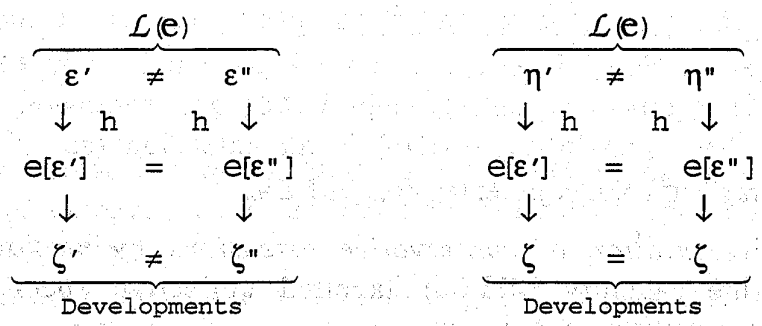


Figure 8.2: Development paths for versions with distinct labels

We now consider implementations of labelled collective specifications.

First recall that an implementation of collective specifications should provide an implementation triangle for each version of the abstract specification in the collection. Also, bear in mind that, in implementing an "abstract" a on a "concrete" c, one must produce a mediating specification.

We now wish to explicate labelled collective implementations in terms of interpretations and conservative extensions, much as in the case of implementations of individual specifications.

Consider labelled collective specifications "abstract" a and "concrete" c. To provide a labelled collective implementation of a on c one should produce a mediating labelled collective specification b with its own label

set. Thus, such a labelled collective implementation would amount to a labelled collective morphism of a into mediating labelled collective specification b, which is a conservative labelled collective extension of c. In order to guarantee the above requirement that this provides an implementation triangle for each version of the abstract specification, we place a constraint on the labelled collective interpretation: that it has a surjective relabeling. (Notice that this constraint is not so severe, since we are producing mediating b with with its own label set. Also, this surjectivity constraint means that each label for the mediating collective specification comes from a label for the "abstract" collective specification, and thus each version of the former supports a version of the latter.)

We thus arrive at the following formulation for implementation of labelled collective specifications. Given labelled collective specifications a and c, a *labelled collective implementation* of a on c consists of a mediating labelled collective object b together with

- a labelled collective morphism $\langle q,t\rangle: a \to b$ whose underlying relabelling $q: \mathcal{L}(b) \to \mathcal{L}(a)$ is a retract in **LBL**, i. e. there exists a morphism $q^*: \mathcal{L}(a) \to \mathcal{L}(b)$ such that the composite $q^*;q$ is the identity on $\mathcal{L}(a)$ in **LBL**; as well as
- a conservative labelled collective extension $\langle p,s\rangle: b \to c$.

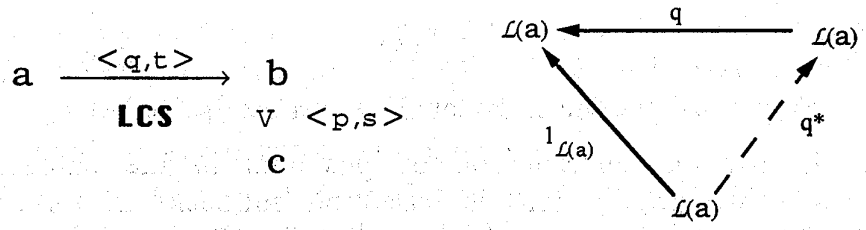


Figure 8.3: Labelled collective implementation

Let us now examine the requirement that an implementation of collective specifications should provide an implementation triangle for each version a_α of the abstract specification in the labelled collective specification a. Given a label α of $\mathcal{L}(a)$, we have a label $\beta := q^*(\alpha)$ in $\mathcal{L}(b)$ such that $q(q^*(\alpha))=\alpha$. We thus have a mediating specification $b[q^*(\alpha)]$ together with

- a morphism $t_\beta: a[q(\beta)] \to b[\beta]$ in **SPEC** whose underlying translation $\mathcal{U}(t_\beta)$ is the signature translation $t: \mathcal{S}(a) \to \mathcal{S}(b)$ in **SGNT**; as well as
- a conservative extension $s_\beta: c[p(\beta)] \subseteq b[\beta]$ in **SPEC** whose underlying translation $\mathcal{U}(s_\beta)$ is the signature inclusion $s: \mathcal{S}(c) \subseteq \mathcal{S}(b)$ in **SGNT**.

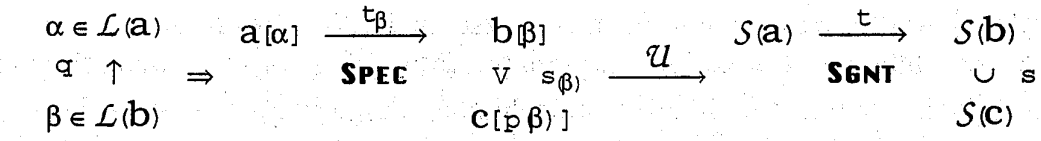


Figure 8.4: Morphisms in labelled collective implementation

Thus, for each label α of $\mathcal{L}(a)$, we have an implementation triangle for

version a_α of the abstract specification on a version of the concrete specification: an interpretation $t_{q^*(\alpha)}:a[\alpha]\rightarrow b[q^*(\alpha)]$ into a mediating specification $b[q^*(\alpha)]$ which extends conservatively $c[p(q^*(\alpha))]$.

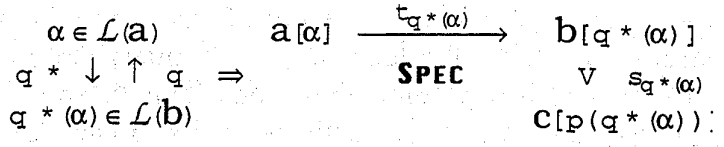


Figure 8.5: Labelled collective implementation triangle

Now, let us examine the compositionality of labelled collective implementations.

Pullbacks are known to preserve retracts, as can easily be seen (see figure 8.6 and the appendix).

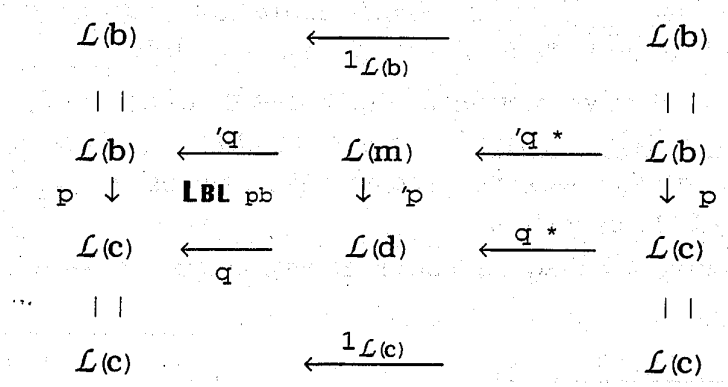


Figure 8.6: Pullback of label set retract $q:\mathcal{L}(d)\rightarrow\mathcal{L}(c)$

Now, recall that our construction of pushouts in the category **LCS** of labelled collective specifications is based on pullbacks in **LBL**. Now,Thus, this pushout construction in the category **LCS** provides a mediating collective specification for the composite implementation step.

Theorem *Composition of labelled collective implementations*
Consider labelled collective specifications a, c and e in the category **LCS**. Given labelled collective implementations

(1) of a on c, consisting of labelled collective morphisms $\langle n,g\rangle:a\rightarrow b$ and conservative extension $\langle p,s\rangle:c\rightarrow b$ into mediating b in **LCS**,

(2) of c on e, consisting of labelled collective morphisms $\langle q,t\rangle:c\rightarrow d$ and conservative extension $\langle r,h\rangle:e\rightarrow d$ into mediating d in **LCS**,

form the pushout of morphisms $\langle p,s\rangle:b\rightarrow c$ and $\langle q,t\rangle:c\rightarrow d$ with common source b in **LCS**, yielding morphisms $\langle 'q,t'\rangle:b\rightarrow m$ and $\langle 'p,s'\rangle:d\rightarrow m$ with common target m in **LCS**. We then have a composite labelled collective implementation of a on e, consisting of composite labelled collective morphisms $\langle n,g\rangle;\langle 'q,t'\rangle:a\rightarrow m$ and conservative extension $\langle r,h\rangle;\langle 'p,s'\rangle:e\rightarrow m$ into mediating m in **LCS**.

Indeed, since $\langle p,s\rangle:c\rightarrow b$ is a labelled collective conservative extension, so

is its pushout $\langle 'p,s' \rangle : d \to m$ in **LCS**, and, since relabelling $q:\mathcal{L}(d) \to \mathcal{L}(c)$ is a retract in **LBL**, so is its pullback $'q:\mathcal{L}(m) \to \mathcal{L}(b)$ in **LBL**.

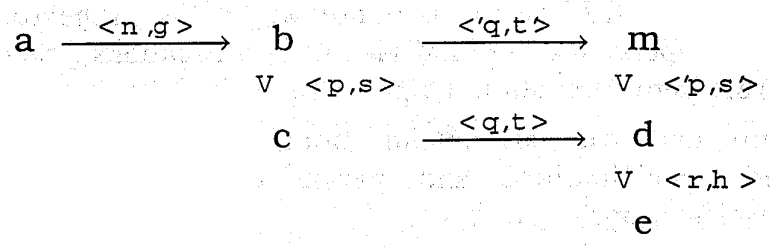


Figure 8.7: Composition of labelled collective implementations

As mentioned, labels serving to tag the various versions, can also be used to keep track of design strategies concerning the development of each version. A further use of labels in modular software development is recording information concerning the structure of a version; for instance, by indicating its components and their subordination. For this purpose, one may use structured labels sets in lieu of simple sets. This involves using another category (still with pullbacks) for labelling, but our categorical approach appears to be adaptable without much difficulty to this case as long as we have a functor $\mathcal{W}(\_):\mathbf{LBL} \to \mathbf{SET}$ mapping the structured of each structured label into sets and functions.

We would then be dealing with a category $\mathbf{LBL}[^{\mathcal{W}}\mathbf{SET}]$ of labels over sets, much as our specifications form a category $\mathbf{SPEC}[^{\mathcal{U}}\mathbf{SGNT}]$ of labels over sets, and they would give rise to a category $\mathbf{LCS}[^{\mathcal{L}}\mathbf{LBL},^{\mathcal{S}}\mathbf{SPEC}]$ of labelled collective specifications with functors $\mathcal{L}(\_):\mathbf{LCS} \to \mathbf{LBL}^{op}$ and $\mathcal{S}(\_):\mathbf{LCS} \to \mathbf{SGNT}$. Our results could be summarised as follows:

if we have pushout transfer from **SGNT** to $\mathbf{SPEC}[^{\mathcal{U}}\mathbf{SGNT}]$ and pullback transfer from **SET** to $\mathbf{LBL}[^{\mathcal{W}}\mathbf{SET}]$, then $\mathbf{LCS}[^{\mathcal{L}}\mathbf{LBL},^{\mathcal{S}}\mathbf{SPEC}]$ has pushouts;

if $\mathbf{SPEC}[^{\mathcal{U}}\mathbf{SGNT}]$ inherits modularity from **SGNT**, then $\mathbf{LCS}[^{\mathcal{L}}\mathbf{LBL},^{\mathcal{S}}\mathbf{SPEC}]$ has modularity;

in such case, implementations are composable in $\mathbf{LCS}[^{\mathcal{L}}\mathbf{LBL},^{\mathcal{S}}\mathbf{SPEC}]$.

9. Conclusions

We have examined internal and external choices and modularity in formal software development by generalising logical concepts concerning specifications to collections of specifications, so as model external choice along the development (freedom in taking design decisions) and internal choice of the particular specification (program version) within the source collection.

Internal and external choices are related to demonic and angelic nondeterminism, respectively. Their interplay mirrors the behaviour of two agents: a developer (team), responsible for carrying out the development, and a decision maker, who will select a final version.

Our formalisation generalises the concept of implementation (as interpretation into a conservative extension) to collective specifications

(of versions tagged by labels). The categorical approach extends the pushout construction to labelled collective specifications. Modularity (i. e. preservation of faithfulness) is shown to be inherited from the corresponding property of the individual specifications, thereby allowing composition of implementation steps.

It is apparent that we can extend along these lines the concepts of parameterised specifications and parameter instantiation to labelled collective specifications.

Ideas related to internal and external choices (and to demonic and angelic nondeterminism, respectively) also appear in other contexts. Some examples are distributed and parallel computing (where some execution details are outside the control of the designer) and default logic [Reiter '80] (where default leaves some choice between alternatives, giving rise to several extensions of a theory).

Labels are being widely used nowadays. Some examples of their use are as tags identifying different versions of program (in software development), as traces of deductions of a formula (in labelled deductive systems [Gabbay '94]), as time stamps (in temporal logic and temporal data bases), as numerical evaluation of data to aggregate confidence in some conclusions (in probabilistic and fuzzy systems). Moreover, the structure of a collection of labelled theories can be mirrored in the label set. For instance, when all theories in the collection are extensions of a particular theory, as in default logic, this structure may be represented in the label set as a tree.

Some ideas of our approach may be expected to be adaptable to these and other related situations.

Appendix: Detailed Proofs of Auxiliary Results

In this appendix we present some more details concerning some auxiliary results that have been used.

Lemma *Condition for pushout transfer via underlying signature*
Consider the categories **SGNT** and **SPEC**, as well as the (covariant) functor underlying signature $\mathcal{U}(\_)$:**SPEC**$\rightarrow$**SGNT**.

Assume that category **SGNT** has pushouts and that the functor $\mathcal{U}(\_)$:**SPEC**$\rightarrow$**SGNT** is faithful and satisfies the joint translation transfer condition:

" given a pair of specifications Δ and Θ in **SPEC**, for each pair of morphisms $s':\mathcal{U}(\Delta)\rightarrow L$ and $t':\mathcal{U}(\Theta)\rightarrow L$ with common target L in **SGNT**, there exists a specification Ψ and a pair of morphisms $\sigma':\Delta\rightarrow\Psi$ and $\tau':\Theta\rightarrow\Psi$ in **SPEC**, such that

($\mathcal{U}$) $\mathcal{U}(\sigma')=s'$ and $\mathcal{U}(\tau')=t'$ with $\mathcal{U}(\Psi)=L$;

(up) given a pair of morphisms $\mu:\Delta\rightarrow\Xi$ and $\nu:\Theta\rightarrow\Xi$ in **SPEC**, for each morphism $k:L\rightarrow\mathcal{U}(\Xi)$ such that $t';k=\mathcal{U}(\mu)$ and $s';k=\mathcal{U}(\nu)$ in **SGNT**,

there exists a morphism $\chi:\Psi\to\Xi$ in **SPEC** such that $\mathcal{U}(\chi)=k$ ".

Then, functor $\mathcal{U}(\_):$**SPEC**$\to$**SGNT** has the pushout transfer:

" in **SPEC** each pair of morphisms $\sigma:\Gamma\to\Delta$ and $\tau:\Gamma\to\Theta$ with common source Γ has a pushout $\sigma':\Delta\to\Psi$ and $\tau':\Gamma\to\Psi$ with common target Ψ, so that the rectangle $\mathcal{U}(\sigma):\mathcal{U}(\Gamma)\to\mathcal{U}(\Delta)$, $\mathcal{U}(\tau):\mathcal{U}(\Gamma)\to\mathcal{U}(\Theta)$, $\mathcal{U}(\sigma'):\mathcal{U}(\Delta)\to\mathcal{U}(\Psi)$ and $\mathcal{U}(\tau'):\mathcal{U}(\Gamma)\to\mathcal{U}(\Psi)$ is a pushout in **SGNT** "

Proof (se figures 6.2, 6.3 and 6.4)

Consider a pair of morphisms $\sigma:\Gamma\to\Delta$ and $\tau:\Gamma\to\Theta$ with source Γ in **SPEC**.

We then have a pair of morphisms $\mathcal{U}(\sigma):\mathcal{U}(\Gamma)\to\mathcal{U}(\Delta)$ and $\mathcal{U}(\tau):\mathcal{U}(\Gamma)\to\mathcal{U}(\Theta)$ with common source $\mathcal{U}(\Gamma)$ in **SGNT**.

$$\begin{array}{ccc} \Delta & & \mathcal{U}(\Delta) \\ \sigma\uparrow & \textbf{SPEC} \xrightarrow{\mathcal{U}} & \sigma\uparrow \quad \textbf{SGNT} \\ \Gamma & \xrightarrow[\tau]{} \Theta & \mathcal{U}(\Gamma) \xrightarrow[\tau]{} \mathcal{U}(\Theta) \end{array}$$

Since category **SGNT** has pushouts, the pair $\mathcal{U}(\sigma):\mathcal{U}(\Gamma)\to\mathcal{U}(\Delta)$ and $\mathcal{U}(\tau):\mathcal{U}(\Gamma)\to\mathcal{U}(\Theta)$ has a pushout $s':\mathcal{U}(\Theta)\to L$ and $t':\mathcal{U}(\Delta)\to L$ with common target L.

$$\begin{array}{ccc} \mathcal{U}(\Delta) & \xrightarrow{t'} & L \\ \mathcal{U}(\sigma)\uparrow & \textbf{SGNT po} & \uparrow s' \\ \mathcal{U}(\Gamma) & \xrightarrow[\mathcal{U}(\tau)]{} & \mathcal{U}(\Theta) \end{array}$$

By the joint translation transfer condition, we have a specification Ψ and a pair of morphisms $\sigma':\Delta\to\Psi$ and $\tau':\Theta\to\Psi$ in **SPEC**, satisfying ($\mathcal{U}$) and (up).

$$\begin{array}{ccccc} \mathcal{U}(\Delta) & \xrightarrow{t'} & \mathcal{U}(\Psi) & \xleftarrow{\mathcal{U}} & \Delta \xrightarrow{\tau'} \Psi \\ & \textbf{SGNT} & \uparrow s' & & \textbf{SPEC} \quad \uparrow \sigma' \\ & & \mathcal{U}(\Theta) & & \Theta \end{array}$$

By ($\mathcal{U}$), we have $\mathcal{U}(\sigma')=s'$ and $\mathcal{U}(\tau')=t'$ with $\mathcal{U}(\Psi)=L$. So, the rectangle $\mathcal{U}(\sigma):\mathcal{U}(\Gamma)\to\mathcal{U}(\Delta)$, $\mathcal{U}(\tau):\mathcal{U}(\Gamma)\to\mathcal{U}(\Theta)$, $\mathcal{U}(\sigma'):\mathcal{U}(\Delta)\to L$ and $\mathcal{U}(\tau'):\mathcal{U}(\Gamma)\to L$ is a pushout in **SGNT**.

We claim that the rectangle $\sigma:\Gamma\to\Delta$, $\tau:\Gamma\to\Theta$, $\sigma':\Delta\to\Psi$ and $\tau':\Theta\to\Psi$ is a pushout in **SPEC**.

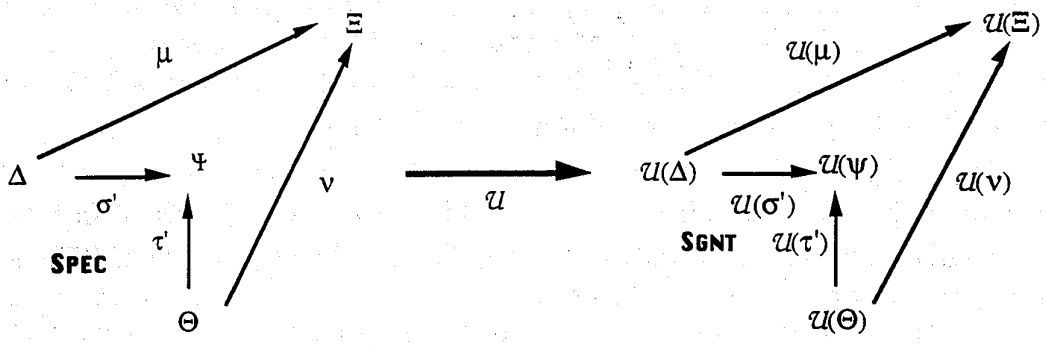
First, we see that we have the commutativity $\sigma;\tau'=\sigma';\tau$ in **SPEC**.

Indeed, in **SGNT** we have the commutativity $s;t'=t;s'$, i. e. $\mathcal{U}(\sigma);\mathcal{U}(\tau')=\mathcal{U}(\tau);\mathcal{U}(\sigma')$. So $\mathcal{U}(\sigma;\tau')=\mathcal{U}(\tau;\sigma')$, and the assumed faithfulness of functor $\mathcal{U}(\_):$**SPEC**$\to$**SGNT** yields the commutativity $\sigma;\tau'=\sigma';\tau$ in **SPEC**.

$$\begin{array}{ccccccc} \mathcal{U}(\Delta) & \xrightarrow{\mathcal{U}(\tau')} & \mathcal{U}(\Psi) & \mathcal{U} & \Delta & \xrightarrow{\tau'} & \Psi \\ \mathcal{U}(\sigma)\uparrow & \textbf{SGNT} & \uparrow \mathcal{U}(\sigma') & \leftarrow & \sigma\uparrow & \textbf{SPEC} & \uparrow \sigma' \\ \mathcal{U}(\Gamma) & \xrightarrow[\mathcal{U}(\tau)]{} & \mathcal{U}(\Theta) & \text{faithful} & \Gamma & \xrightarrow[\tau]{} & \Theta \end{array}$$

For the universal property, consider a pair $\mu:\Delta\to\Xi$ and $\nu:\Theta\to\Xi$ of morphisms with common target Ξ in **SPEC** such that $\sigma;\mu=\tau;\nu$.

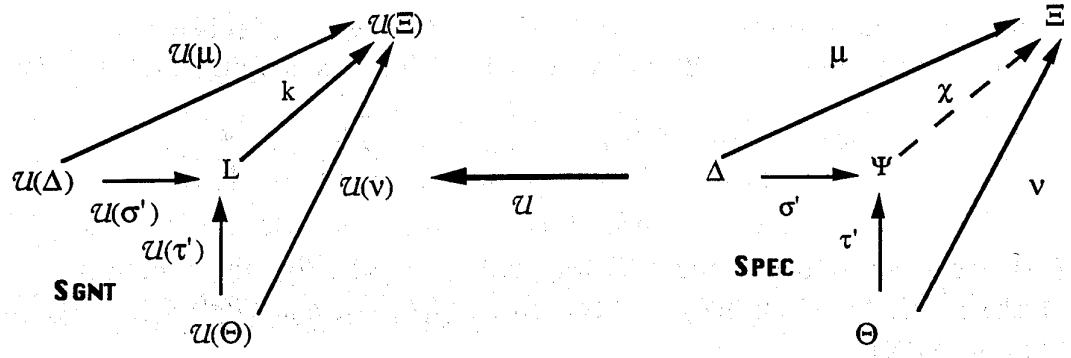
We then have a pair of morphisms $\mathcal{U}(\mu):\mathcal{U}(\Delta)\rightarrow\mathcal{U}(\Xi)$ and $\mathcal{U}(\nu):\mathcal{U}(\Theta)\rightarrow\mathcal{U}(\Xi)$ with common target $\mathcal{U}(\Xi)$ in **SGNT** such that $\mathcal{U}(\sigma);\mathcal{U}(\mu)=\mathcal{U}(\tau);\mathcal{U}(\nu)$.



By the universal property of the pushout rectangle $\mathcal{U}(\sigma):\mathcal{U}(\Gamma)\rightarrow\mathcal{U}(\Delta)$, $\mathcal{U}(\tau):\mathcal{U}(\Gamma)\rightarrow\mathcal{U}(\Theta)$, $\mathcal{U}(\sigma'):\mathcal{U}(\Delta)\rightarrow\mathcal{U}(\Psi)$ and $\mathcal{U}(\tau'):\mathcal{U}(\Gamma)\rightarrow\mathcal{U}(\Psi)$ in **SGNT**, we have a unique signature morphism $k:\mathcal{U}(\Psi)\rightarrow\mathcal{U}(\Xi)$ factorising $\mathcal{U}(\mu)$ and $\mathcal{U}(\nu)$, i. e. $\mathcal{U}(\tau');k=\mathcal{U}(\mu)$ and $\mathcal{U}(\sigma');k=\mathcal{U}(\nu)$, in **SGNT**.

So, we have in **SGNT** a morphism $k:\mathcal{U}(\Psi)\rightarrow\mathcal{U}(\Xi)$ such that $t';k=\mathcal{U}(\mu)$ and $s';k=\mathcal{U}(\nu)$, whence by the clause (up) of the joint translation transfer condition, we have a morphism $\chi:\Psi\rightarrow\Xi$ in **SPEC** such that $\mathcal{U}(\chi)=k$.

We thus have, in **SGNT**, $\mathcal{U}(\tau');\mathcal{U}(\chi)=\mathcal{U}(\mu)$ and $\mathcal{U}(\sigma');\mathcal{U}(\chi)=\mathcal{U}(\nu)$, i. e. $\mathcal{U}(\tau';\chi)=\mathcal{U}(\mu)$ and $\mathcal{U}(\sigma';\chi)=\mathcal{U}(\nu)$. So, faithfulness of functor $\mathcal{U}(\_):$**SPEC**$\rightarrow$**SGNT** yields the factorisation $\tau';\chi=\mu$ and $\sigma';\chi=\nu$ in **SPEC**.



To see the uniqueness of such factorisation $\tau';\chi=\mu$ and $\sigma';\chi=\nu$ in **SPEC**, consider a morphism $\varphi:\Psi\rightarrow\Xi$ in **SPEC** such that $\tau';\varphi=\mu$ and $\sigma';\varphi=\nu$.

We then have $\mathcal{U}(\tau';\varphi)=\mathcal{U}(\tau');\mathcal{U}(\varphi)=\mathcal{U}(\mu)$ and $\mathcal{U}(\sigma';\varphi)=\mathcal{U}(\sigma');\mathcal{U}(\varphi)=\mathcal{U}(\nu)$, in **SGNT**. So, the uniqueness of the factorisation $\mathcal{U}(\tau');k=\mathcal{U}(\mu)$ and $\mathcal{U}(\sigma');k=\mathcal{U}(\nu)$, in **SGNT**, yields $\mathcal{U}(\varphi)=k=\mathcal{U}(\chi)$; whence $\varphi=\chi$ in **SPEC**, by the assumed faithfulness of functor $\mathcal{U}(\_):$**SPEC**$\rightarrow$**SGNT**.

Therefore, the rectangle $\sigma:\Gamma\rightarrow\Delta$, $\tau:\Gamma\rightarrow\Theta$, $\sigma':\Delta\rightarrow\Psi$ and $\tau':\Theta\rightarrow\Psi$ is a pushout in **SPEC** such that its image $\mathcal{U}(\sigma):\mathcal{U}(\Gamma)\rightarrow\mathcal{U}(\Delta)$, $\mathcal{U}(\tau):\mathcal{U}(\Gamma)\rightarrow\mathcal{U}(\Theta)$, $\mathcal{U}(\sigma'):\mathcal{U}(\Delta)\rightarrow\mathcal{U}(\Psi)$ and $\mathcal{U}(\tau'):\mathcal{U}(\Gamma)\rightarrow\mathcal{U}(\Psi)$ under functor $\mathcal{U}(\_):$**SPEC**$\rightarrow$**SGNT** is a pushout rectangle in **SGNT**.

QED

Lemma *Preservation of retracts by pullbacks of label sets*

Consider a pullback rectangle in category **LBL**: morphisms

$p:\mathcal{L}(b)\rightarrow\mathcal{L}(c)$ and $q:\mathcal{L}(d)\rightarrow\mathcal{L}(c)$ with common target $\mathcal{L}(c)$ yielding morphisms $'p:\mathcal{L}(m)\rightarrow\mathcal{L}(d)$ and $'q:\mathcal{L}(m)\rightarrow\mathcal{L}(b)$ with common source $\mathcal{L}(m)$. If morphism $q:\mathcal{L}(d)\rightarrow\mathcal{L}(c)$ is a retract then so is its pullback $'q:\mathcal{L}(m)\rightarrow\mathcal{L}(b)$.

Proof (se figure 8.6)

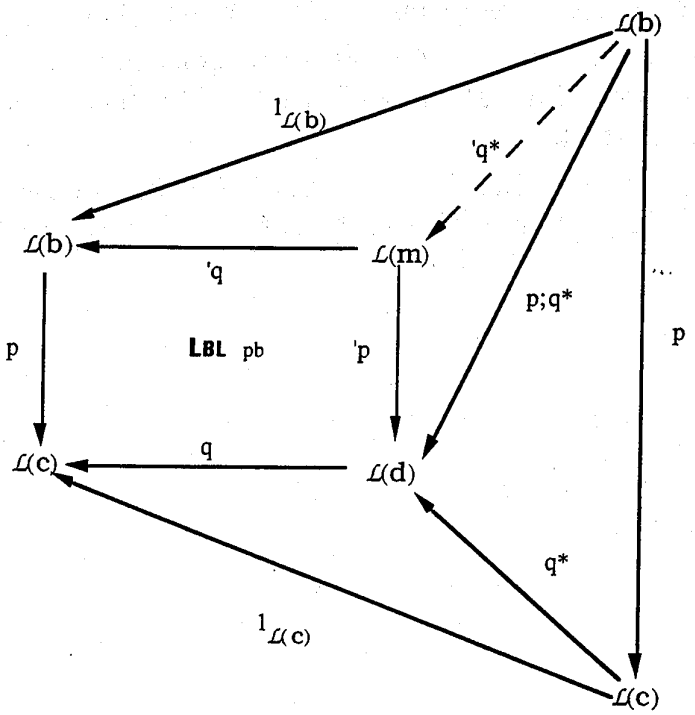
Since morphism $q:\mathcal{L}(d)\rightarrow\mathcal{L}(c)$ is a retract, we have a morphism $q^*:\mathcal{L}(a)\rightarrow\mathcal{L}(b)$ such that the composite $q^*;q$ is the identity morphism $1_{\mathcal{L}(c)}:\mathcal{L}(c)\rightarrow\mathcal{L}(c)$ on $\mathcal{L}(c)$ in **LBL**.

We now apply the universal property of the pullback.

Consider the rectangle in category **LBL**: morphisms $p:\mathcal{L}(b)\rightarrow\mathcal{L}(c)$ and $q:\mathcal{L}(d)\rightarrow\mathcal{L}(c)$ with common target $\mathcal{L}(c)$ and morphisms $1_{\mathcal{L}(b)}:\mathcal{L}(b)\rightarrow\mathcal{L}(b)$ and the composite $p;q^*:\mathcal{L}(b)\rightarrow\mathcal{L}(d)$ with common source $\mathcal{L}(b)$.

It commutes, because $p;q^*;q=p;1_{\mathcal{L}(c)}=p=1_{\mathcal{L}(b)};p$.

Thus, since the rectangle $p:\mathcal{L}(b)\rightarrow\mathcal{L}(c)$, $q:\mathcal{L}(d)\rightarrow\mathcal{L}(c)$, $'p:\mathcal{L}(m)\rightarrow\mathcal{L}(d)$ and $'q:\mathcal{L}(m)\rightarrow\mathcal{L}(b)$ is a pullback in **LBL**, we have a (unique) label morphism $'q^*:\mathcal{L}(b)\rightarrow\mathcal{L}(m)$ factorising $1_{\mathcal{L}(b)}$ (and $p;q^*$); so $'q^*;'q=1_{\mathcal{L}(b)}$, whence morphism $'q:\mathcal{L}(m)\rightarrow\mathcal{L}(b)$ is a retract in **LBL**.



QED

References

[1] Arbib, M. and Mannes, E. (1975) *Arrows, Structures and Functors: the Categorical Imperative*. Academic Press, New York.

[2] Enderton, H. B. (1972) *A Mathematical Introduction to Logic*. Academic Press, New York.

[3] Ehrig, H. and Mahr, B. (1985) *Fundamentals of Algebraic Specifications, 1: Equations and Initial Semantics.* Springer-Verlag, Berlin.

[4] Gabbay, D. M. (1994) *Labelled Deductive Systems, Vol. 1.* Oxford University Press, Oxford.

[5] Mac Lane, S. (1971) *Categories for the Working Mathematician.* Springer-Verlag, New York.

[6] Maibaum, T. S. E, Veloso, P. A. S. and Sadler, M. R. (1985) A theory of abstract data types for program development: bridging the gap?. In Ehrig, H., Floyd, C., Nivat, M. and Thatcher, J. eds. *Formal Methods and Software Development, vol. 2: Colloquium on Software Engineering.* Springer-Verlag, Berlin, 214-230.

[7] Reiter, R. (1980) A Logic for Default Reasoning. *Artificial Intelligence,* **13**, 81-132.

[8] Shoenfield, J. R. (1967) *Mathematical Logic.* Addison-Wesley, Reading.

[9] Turski, W. M and Maibaum, T. S. E. (1987) *The Specification of Computer Programs.* Addison-Wesley, Wokingham.

[10] Veloso, P. A. S. (1996) On pushout consistency, modularity and interpolation for logical specifications. *Information Processing Letters* (to appear).

[11] Veloso, P. A. S. and Maibaum, T. S. E. (1995) On the modularisation theorem for logical specifications. *Information Processing Letters,* 53, 287-293.