Subtypes and Subsorts in Overloaded Specifications

D. Duval, H. Kirchner, C. Lair

Rapport de Recherche IMAG-LMC
n° 1058 I
Septembre 2003

Subtypes and Subsorts in Overloaded Specifications

Dominique DUVAL
LMC-IMAG, Université Joseph Fourier, Grenoble
Dominique.Duval@imag.fr

Hélène KIRCHNER
LORIA, Nancy
Helene.Kirchner@loria.fr

Christian LAIR
Équipe Catégories et Structures, Université Denis Diderot, Paris
lairchrist@aol.com

September 10, 2003

Abstract.

A new point of view about overloading is introduced: overloaded specifications are considered as morphisms of ordinary specifications. Then it is proven that some major issues about overloading are quite simple to express and to study with this point of view. Moreover, the interaction of overloading with subtyping and subsorting gets clarified. Then, in the framework of diagrammatic specifications, it is shown that overloaded specifications and ordinary specifications can be unified, and that this study can be generalized.

1 Introduction

This paper deals with *overloading*, in the sense that distinct operations are allowed to share the same name. The focus is on specification concepts, rather than implementation issues.

A *specification* is a presentation of a logical theory. It is made of *sort symbols*, called *sorts*, and *operation symbols*, called *operations*, together with *axioms*. A sort is interpreted as a set of values, and an operation as a map between sets of values, which has to satisfy the axioms. In addition, *subsorts* are interpreted as subsets.
On the other hand, in simply typed lambda calculus, there are *type symbols*, called *types*, and *function symbols*, called *functions*. A type is interpreted as a set of values, and a function as a map between sets of values. In addition, *subtypes* do not have to be interpreted as subsets.
In this paper, the words "sorts" and "operations" are used rather than "types" and "functions", but "subsorts" and "subtypes" remain distinct notions.

There is *overloading* as soon as several operations have the same name. In this paper, the operations and their names are considered as distinct entities. More precisely, the *names* of the operations are themselves considered as operations in another specification, so that the fact of naming operations becomes a morphism of specifications. Hence, there is *overloading* as soon

as several operations get identified by this naming morphism.
It should be emphasized that the specification which is made of the names is built mainly for syntactic purposes: its models are usually irrelevant.
Two semantics, at least, can be associated with overloading, depending on whether overriding is allowed, or not. Indeed, a major issue about overloading is to choose the "right" interpretation, when several terms share the same name. If overriding is allowed, then several terms with the same name and the same profile may have distinct interpretations. Otherwise, such terms must get identified: in this paper, this is called the *identification rule*. Both semantics may coincide, under restrictive assumptions.
A *subtype* X of a type Y is such that a value of type X can be used in any context in which a value of type Y is expected. This is expressed as a *coercion*, or an implicit conversion, from X to Y, i.e., as an operation which does not appear in the notations. With our definition of overloading, the coercions are the operations such that their name is an *identity*. The interpretation of this operation can be any map, it is not required to be injective (i.e., an embedding).
The property of subtype is expressed by the *subtyping rule*, which can be used either when overriding is allowed, or in combination with the identification rule.
A *subsort* X of a sort Y is such that the interpretation of X is a subset of the interpretation of Y. This is expressed as a *sort inclusion* from X to Y, i.e., as a coercion such that its interpretation is injective. In an overloaded specification, some coercions can be declared to be sort inclusions. When the semantics of overloading satisfies the identification rule, it is required that the sort inclusions satisfy the *retraction rule*. This ensures that their interpretation is an injective map, and provides the required restrictions of operations, without involving any partial retraction operation.

In section 2, overloading, subtypes and subsorts are studied in the context of equational specifications, according to these lines.

However, looking at overloaded specifications as morphisms of ordinary specifications is not completely satisfactory: it would be better to consider that all the specifications are of the same nature. It happens that both equational and overloaded specifications are *diagrammatic specifications*, in the sense of [Duval and Lair 2002].

Hence, overloading, subtypes and subsorts are revisited in section 3 in the framework of diagrammatic specifications. In addition, this point of view is not restricted to the equational specifications. For instance, an application to subtyping of power types is presented.

This work is motivated by the decorated equational logic of [Hintermeier et al. 1998]. Other approaches are presented in the survey paper [Mosses 1992], or more recently in [Castagna 1995], [Bouhoula et al. 2000], [Casl 2003], among others.

2 Overloaded specifications as morphisms of specifications

In this section, overloaded equational specifications are considered as morphisms of equational specifications. In part 2.1, some notions about equational specifications are reminded. Then overloaded specifications and their plain semantics are defined in part 2.2, while the resolved semantics is described in part 2.3. Then subtyping is studied in part 2.4 for its plain semantics and in part 2.5 for its resolved semantics. Finally, subsorts are added in part 2.6, and the resolved semantics is enriched accordingly.

2.1 Equational specifications and their semantics

Equational specifications

An *equational signature* is made of *sorts* and *operations*. Sorts are symbols like X, Y,..., operations are symbols like f, g,..., so that distinct operations are distinct symbols. Each operation f has a *profile* $X_1, \ldots, X_k \longrightarrow Y$, made of its *source* $X_1, \ldots, X_k$ and its *target* Y; this is denoted $f : X_1, \ldots, X_k \longrightarrow Y$. When $k = 0$, the operation f is called a *constant*, it is denoted $f :\longrightarrow X$, or just $f : X$.

The *terms* of an equational signature are usually defined inductively, from sorted *variables* and operations; let us call them the *logical* terms. In this paper, another classical definition of terms is used, which avoids variables. From this point of view, the profile of a term is made of its source and target, which both are lists of sorts. The terms with target a list of length one can be identified to the logical terms. Here, the same kind of symbols X, Y,..., are used for sorts and for lists of sorts, and the list made of one sort X is also denoted X. Terms are defined inductively as follows, together with their profile.
– Each operation $f : X \longrightarrow Y$ is a term (which corresponds to the logical term $f(x)$, where x is a variable of sort X).
– For each list of sorts Y, there is a term $\mathrm{id}_Y : Y \longrightarrow Y$, called the *identity* of Y (which corresponds to a logical term y, where y is any variable of sort Y).
– If $t : X \longrightarrow Y$ and $u : Y \longrightarrow Z$ are consecutive terms, then $u \circ t : X \longrightarrow Z$ is a term.
– If $t_i : X \longrightarrow Y_i$ is a term, for each i from 1 to n, then $(t_1, \ldots, t_n) : X \longrightarrow Y_1, \ldots, Y_n$ is a term (so that, for each operation $g : Y_1, \ldots, Y_n \longrightarrow Z$, the term $g \circ (t_1, \ldots, t_n) : X \longrightarrow Z$ corresponds to the logical term $g(t_1, \ldots, t_n)$).

A term is *closed* when its source is the empty list of sorts. In the examples, when dealing with closed terms, the logical notation $g(t_1, \ldots, t_n)$ will be used instead of $g \circ (t_1, \ldots, t_n)$, and $g(t)$ instead of $g \circ t$.

An *equational specification* is made of an equational signature together with *equations*. An *equation* is made of two *parallel* terms t_l and t_r, i.e., two terms with the same profile; it is written $t_l \equiv t_r$.

It should be noted that, with these definitions, there is no kind of overloading in an equational specification. The overloading will be introduced in part 2.2.

Equational theories

The *equational theory* $F(S)$ which is generated by an equational specification S can be considered as an equational specification itself: its sorts are the lists of sorts of S, its operations are the terms of S, and its equations are all the *congruences* which are generated from the equations of S (we choose the following terminology: each element of *the congruence relation* in an equational theory is called a *congruence*).

As usual, a *deduction rule* is a property which does not have to be satisfied in the specification S, but which has to be satisfied in the generated theory $F(S)$. Several deduction rules are used in order to derive the theory $F(S)$ from the specification S, let us call them the *equational rules*.

Two major equational rules, for building the congruences, are the *substitution rule* and the *replacement rule*:
if $u_l \equiv u_r : Y \longrightarrow Z$, $t : X \longrightarrow Y$ and $v : Z \longrightarrow U$,

then $u_l \circ t \equiv u_r \circ t : X \longrightarrow Z$ and $v \circ u_l \equiv v \circ u_r : Y \longrightarrow U$.

Other equational rules deal with identities:
if $t : X \longrightarrow Y$ then $t \circ \mathrm{id}_X = t$ and $\mathrm{id}_Y \circ t = t$.
Another equational rule is the associativity of composition:
if $t : X \longrightarrow Y$, $u : Y \longrightarrow Z$ and $v : Z \longrightarrow W$, then $(v \circ u) \circ t = v \circ (u \circ t)$,
so that this term can be simply written $v \circ u \circ t$.

The generation of the congruence relation is a major issue about equational specifications, which can be solved with the help of *rewrite rules*. The generalization of rewrite rules to subsorts is addressed in [Hintermeier et al. 1998].

Morphisms and models

A *morphism* of equational specifications $\zeta : T \longrightarrow S$ maps the sorts of T to sorts (or lists of sorts) of S, the operations of T to operations (or terms) of S, in such a way that the profiles are preserved, and that the equations of T are mapped to equations (or congruences) of S. Such a morphism is canonically extended to the theories: $F(\zeta) : F(T) \longrightarrow F(S)$.

A *(set-valued) model* M of an equational signature, often called an *algebra*, interprets each sort X as a set $M(X)$ and each operation $f : X_1, \ldots, X_k \longrightarrow Y$ as a map $M(f) : M(X_1) \times \cdots \times M(X_k) \longrightarrow M(Y)$. Then, the interpretation of a term is defined recursively: an identity id_Y is interpreted as the identity map $\mathrm{id}_{M(Y)}$, such that $y \mapsto y$; a term $u \circ t$ is interpreted as the composed map $M(u) \circ M(t)$, such that $x \mapsto M(u)(M(t)(x))$; and a term $(t_1, \ldots, t_n)$ is interpreted as the map $(M(t_1), \ldots, M(t_n))$, such that $x \mapsto (M(t_1)(x), \ldots, M(t_n)(x))$.

A *(set-valued) model* M of an equational specification S is a model of the signature of S such that each equation $t_l \equiv t_r : X_1, \ldots, X_k \longrightarrow Y$ of S becomes an equality $M(t_l)(x_1, \ldots, x_k) = M(t_r)(x_1, \ldots, x_k)$ for all $(x_1, \ldots, x_k)$ in the set $M(X_1) \times \cdots \times M(X_k)$. There is a straightforward notion of morphism of models of S.

Let M be a model of S, then all the congruences which are generated from the equations of S become equalities in M: this is the *soundness property* of equational logic.

The theory and the models of S are nicely related: the semantics of S is sound and complete, and the closed terms of S, modulo the congruence relation, form an initial model of S [Goguen et al. 1978].

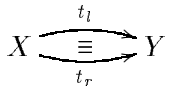
Extensions by definition

In addition, an equational specification S can be *extended by definition*, which means that a new operation f is introduced, together with a *defining equation*, i.e. an equation of the form $f \equiv t$ where t is a term which does not involve f. Such an extension does not modify the models of S, and it does not modify the theory of S, up to congruence.

The illustration of specifications and rules

An equational specification S can easily be illustrated by ways of a directed graph: the points of the graph are the sorts (and some lists of sorts) of S, its arrows are the operations (and some terms) of S; moreover, an equation $t_l \equiv t_r : X \longrightarrow Y$ of S can be illustrated by adding a

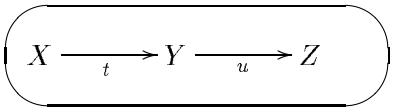
symbol "$\equiv$" to the graph:



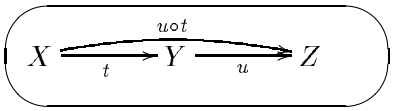
This can also be used for illustrating the deduction rules. Let us look, for instance, at the rule:

If $t : X \longrightarrow Y$ **and** $u : Y \longrightarrow Z$ **are terms of** S**,**
then $u \circ t : X \longrightarrow Z$ **is a term of** S**.**

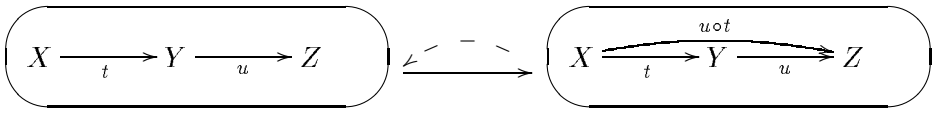
Its hypothesis "$t : X \longrightarrow Y$ and $u : Y \longrightarrow Z$ are terms of S" corresponds to an equational specification $\mathcal{H}$:



Similarly, its conclusion "$u \circ t : X \longrightarrow Z$ is a term of S" corresponds to an equational specification $\mathcal{C}$:



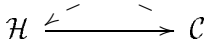
There is a straightforward morphism from $\mathcal{H}$ to $\mathcal{C}$, and the rule means that each occurrence of $\mathcal{H}$ in an equational specification S can be extended as an occurrence of $\mathcal{C}$ in the equational theory $F(S)$. This is illustrated with the help of a dashed arrow:



More generally, any deduction rule:

$$\frac{\mathcal{H}}{\mathcal{C}}$$

can be illustrated in a similar way, after replacing $\mathcal{C}$ by the conjunction $\mathcal{H} \wedge \mathcal{C}$, if necessary:



A deeper motivation for such illustrations can be found in [Duval et al. 2003].

2.2 Overloaded specifications and their plain semantics

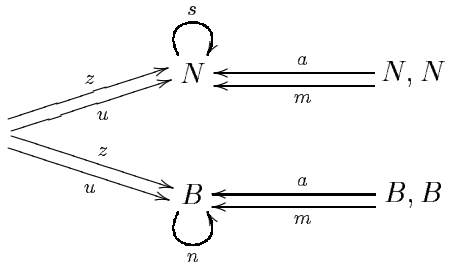
An example

An *overloaded specification* is not an equational specification, strictly speaking. In an informal way, an overloaded specification is "an equational specification where several operations are allowed to share the same name". Let us first look at a detailed simple example.

Example 2.1 In order to deal simultaneously with the set $\mathbb{N}$ of integers and with the set $\mathbb{B}$ of booleans, the following overloaded specification Z_1 can be used. It is made of two sorts N and B, two overloaded constants z and u, two non-overloaded unary operations s and n, two overloaded binary operations a and m, and some equations:

`Overloaded specification` Z_1:
`Sorts:`
N, B.
`Operations:`
$z, u : N$, $z, u : B$, $s : N \longrightarrow N$, $n : B \longrightarrow B$,
$a, m : N, N \longrightarrow N$, $a, m : B, B \longrightarrow B$.
`Equations:`
$s(z) \equiv u : N$, $n(z) \equiv u : B$, $n(u) \equiv z : B$,
$a(z, u) \equiv u : N$, $a(z, u) \equiv u : B$, $a(u, u) \equiv u : B$, ...

$Sig(Z_1)$:



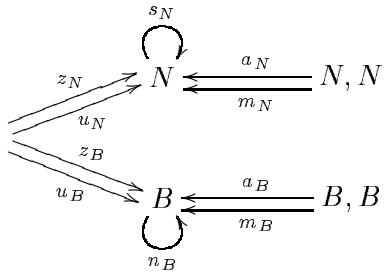
The interpretation we have in mind maps the sorts N and B to the sets $\mathbb{N}$ and $\mathbb{B}$, the constant operations z and u to the integers 0 and 1 in $\mathbb{N}$ and to the booleans 0 (or f, for *false*) and 1 (or t, for *true*) in $\mathbb{B}$, the unary operation s to the successor map *succ* of $\mathbb{N}$, the unary operation n to the negation $\neg$ in $\mathbb{B}$, the binary operations a and m to the addition $+$ and multiplication $\times$ of $\mathbb{N}$ and to the disjunction $+$ (or $\vee$) and conjunction $\times$ (or $\wedge$) of $\mathbb{B}$.

Two distinct equational specifications T_1 and S_1 can be associated to Z_1, as follows.

In the equational specification T_1, the overloading is eliminated by giving distinct names to distinct operations:

`Equational specification` T_1:
`Sorts:`
N, B.
`Operations:`
$z_N, u_N : N$, $z_B, u_B : B$, $s_N : N \longrightarrow N$, $n_B : B \longrightarrow B$,
$a_N, m_N : N, N \longrightarrow N$, $a_B, m_B : B, B \longrightarrow B$.
`Equations:`
$s_N(z_N) \equiv u_N$, $n_B(z_B) \equiv u_B$, $n_B(u_B) \equiv z_B$,
$a_N(z_N, u_N) \equiv u_N$, $a_B(z_B, u_B) \equiv u_B$, $a_B(u_B, u_B) \equiv u_B$, ...

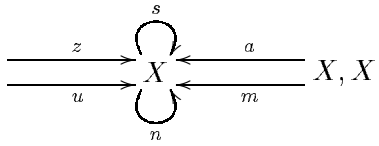
$Sig(T_1)$:



In the equational specification S_1, the overloading is eliminated by identifying the operations which have the same name. For instance, both $z_N : N$ and $z_B : B$ get identified as a unique operation $z : X$, so that the sorts N and B also get identified as a unique sort X:

`Equational specification` S_1:
`Sort:`
X.
`Operations:`
$z, u : X$, $s, n : X \longrightarrow X$,
$a, m : X, X \longrightarrow X$.
`Equations:`
$s(z) \equiv u$, $n(z) \equiv u$, $n(u) \equiv z$,
$a(z, u) \equiv u$, $a(z, u) \equiv u$, $a(u, u) \equiv u$, ...

$Sig(S_1)$:



Let $\zeta_1 : T_1 \longrightarrow S_1$ be the morphism which maps N and B to X, z_N and z_B to z, and so on. Then T_1, S_1 and ζ_1 are determined from Z_1. Conversely, Z_1 can be recovered from T_1, S_1 and ζ_1: indeed, Z_1 is a copy of T_1 where the names of the operations are modified as follows: if f' is an operation of T_1, its name in Z_1 is the image $\zeta_1(f')$ of f' via ζ_1.

Overloaded specifications

The previous example suggests the following definition:

Definition 2.2
An *overloaded equational specification* is a morphism of equational specifications $\zeta : T \longrightarrow S$.

In this section, an overloaded equational specification is called simply an *overloaded specification*.

Usually, in an overloaded specification, there is at most one operation with a given name and a given profile. Then, as in example 2.1, it is non-ambiguous to represent an overloaded specification $\zeta : T \longrightarrow S$ as a copy of T where each operation is named according to its image in S. Indeed, $f : X' \longrightarrow Y'$ is equal to $f : X'' \longrightarrow Y''$ if and only if $X' = X''$ and $Y' = Y''$.

Hence, graphically, our representation of an overloaded specification is the same one as usual. However, by defining an overloaded specification as a morphism of ordinary specifications, we avoid some ambiguities and we get a clearer view about the issues of overloading.

At first glance, the semantics of ζ is determined by T, while S is a simplified syntactic view of T, which can be used in order to organize the parsing in two steps: first on the names (i.e., in S), then on T. Indeed, usually S is much simpler than T. For instance, in example 2.1, S_1 deals only with the arity issues (i.e., the number of arguments), while T_1 adds the typing information.

In this paper, the following systematic terminology is used:

Definition 2.3 The *sorts*, *operations* and *equations* of $\zeta : T \longrightarrow S$ are those of T. Their *name* is their image in S via ζ. Two operations are *homonymous* if they have the same name. An operation *overloads* another one when both are homonymous.

This terminology is extended to *terms* and *congruences*. However, it will be seen that there can be more terms and congruences in ζ than in T, according to the chosen semantics for ζ.

Here, this terminology is chosen for its coherence. But many other choices can be found, for instance:
– in [Castagna 1995]: a *function* of ζ is an operation of S, and the operations of T with a given name f are the *branches* of f;
– in [Hintermeier et al. 1998]: a *term* is a term of S, and a *decorated term* is a term of T; the *decoration* of a term t (of S) is made of the targets of the terms (of T) with name t; a term t (of S) *exists* if is the name of at least one term (of T); in addition, *decorated rewrite rules* are used in order to generate the congruence relation of T;
– in [Bouhoula et al. 2000]: a sort of ζ is indeed a sort of T, and its name in S is called its *kind*;
– in [Bruce 2002]: an operation of ζ is indeed an operation of T, and its name is called an *overloaded operator name*;
– in [Casl 2003]: an operation of ζ is indeed an operation of T, and its *name* is made of its name in S together with its profile, so that the "same name, same thing" principle for operations means that there is at most one operation with a given name and a given profile.

The plain semantics of an overloaded specification

The previous example also suggests that the semantics of ζ is the semantics of T. Actually, this is only one of the possible semantics of ζ, we call it the *plain* semantics. This semantics allows the overridding of operations: no relation is required between the interpretations of two terms with the same name.

Definition 2.4 The *plain models* of an overloaded equational specification $\zeta : T \longrightarrow S$ are the models of the equational specification T.

Since an overloaded specification is a morphism of equational specifications $\zeta : T \longrightarrow S$, it generates, from the equational rules, a morphism of equational theories $F(\zeta) : F(T) \longrightarrow F(S)$.

Definition 2.5 The *plain theory* of an overloaded specification ζ is $F(\zeta)$.

Proposition 2.6 *The plain semantics of ζ is sound and complete. With respect to the plain semantics, the closed terms of ζ, modulo the congruence relation, form an initial model of ζ.*

Proof. Since the plain semantics of ζ is the equational semantics of T, this result follows immediately from the corresponding result about equational specifications. □

Example 2.7 In example 2.1, with respect to the plain semantics:
– the term $s_N(z_N)$ is a term of ζ_1 with name $s(z)$, and $s_N(z_N) \equiv u_N$ is an equation of ζ_1 with name $s(z) \equiv u$,
– the term $s(z)$ of S_1 it the name of exactly one term $s_N(z_N)$ of ζ_1,
– the term $a(u,u)$ of S_1 is the name of two terms $a_N(u_N,u_N)$ and $a_B(u_B,u_B)$ of ζ_1: these terms are homonymous,
– the term $s(n(z))$ of S_1 is not the name of any term of ζ_1,
– the equation $s(z) \equiv u$ of S_1 is the name of exactly one equation $s_N(z_N) \equiv u_N$ of ζ_1,
– the equation $a(z,u) \equiv u$ of S_1 is the name of two equations $a_N(z_N,u_N) \equiv u_N$ and $a_B(z_B,u_B) \equiv u_B$ of ζ_1: these equations are homonymous,
– the congruence $s(z) \equiv n(z)$ of S_1, which is derived from $s(z) \equiv u$ and $n(z) \equiv u$, is not the name of any congruence of ζ_1.

The following interpretation M_1 is a plain model of ζ_1:

`Plain model` M_1 `of` ζ_1:
`Interpretation of the sorts:`
$M_1(N) = \mathbb{N}$, $M_1(B) = \mathbb{B}$.
`Interpretation of the operations:`
$M_1(z_N) = 0 \in \mathbb{N}$, $M_1(u_N) = 1 \in \mathbb{N}$,
$M_1(z_B) = 0 = f \in \mathbb{B}$, $M_1(u_B) = 1 = t \in \mathbb{B}$,
$M_1(s_N) = succ : \mathbb{N} \longrightarrow \mathbb{N}$, $M_1(n_B) = \neg : \mathbb{B} \longrightarrow \mathbb{B}$,
$M_1(a_N) = + : \mathbb{N}^2 \longrightarrow \mathbb{N}$, $M_1(m_N) = \times : \mathbb{N}^2 \longrightarrow \mathbb{N}$,
$M_1(a_B) = + = \vee : \mathbb{B}^2 \longrightarrow \mathbb{B}$, $M_1(m_B) = \times = \wedge : \mathbb{B}^2 \longrightarrow \mathbb{B}$.

Example 2.8 In order to deal with the set $\mathbb{N}/2$ of integers modulo 2 rather than the set $\mathbb{N}$ of naturals, T_1 is enriched with the equations $s_N(u_N) \equiv z_N$ and $a_N(u_N,u_N) \equiv z_N$. Hence S_1 is enriched with the equations $s(u) \equiv z$ and $a(u,u) \equiv z$. This leads to a new overloaded specification $\zeta_2 : T_2 \longrightarrow S_2$. It follows from both equations $a(u,u) \equiv u$ and $a(u,u) \equiv z$ in S_2 that the congruence $u \equiv z$ can be derived in S_2. In T_2, there is an equation $a_B(u_B,u_B) \equiv u_B$ with name $a(u,u) \equiv u$, and an equation $a_N(u_N,u_N) \equiv z_N$ with name $a(u,u) \equiv z$, but there is no congruence with name $u \equiv z$. Hence, there is no congruence of ζ_2 with name $u \equiv z$.

The following interpretation M_2, where $succ_2$ denotes the successor map modulo 2, is a plain model of ζ_2:

`Plain model` M_2 `of` T_1:
`Interpretation of the sorts:`
$M_2(N) = \mathbb{N}/2$, $M_2(B) = \mathbb{B}$.
`Interpretation of the operations:`
$M_2(z_N) = 0 \in \mathbb{N}/2$, $M_2(u_N) = 1 \in \mathbb{N}/2$,
$M_2(z_B) = 0 = f \in \mathbb{B}$, $M_2(u_B) = 1 = t \in \mathbb{B}$,
$M_2(s_N) = succ : \mathbb{N}/2 \longrightarrow \mathbb{N}/2$, $M_2(n_B) = \neg : \mathbb{B} \longrightarrow \mathbb{B}$,
$M_2(a_N) = + : (\mathbb{N}/2)^2 \longrightarrow \mathbb{N}/2$, $M_2(m_N) = \times : (\mathbb{N}/2)^2 \longrightarrow \mathbb{N}/2$,
$M_2(a_B) = + = \vee : \mathbb{B}^2 \longrightarrow \mathbb{B}$, $M_2(m_B) = \times = \wedge : \mathbb{B}^2 \longrightarrow \mathbb{B}$.

2.3 The name-driven semantics of overloaded specifications

The disambiguation question

One major issue about overloading is the following *disambiguation question*:

If there are several homonymous parallel terms,
which one is the "right" one?

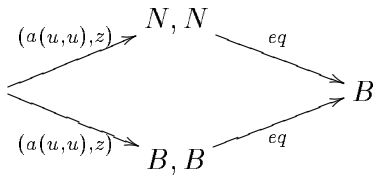
The overloading is *resolved* when an answer can be given to this question.

Example 2.9 In example 2.1, such a situation cannot occur: two homonymous terms of T_1 may have the same source, but not the same target, so that they cannot be parallel. For instance, the terms $a_N(z_N, z_N)$ and $a_B(z_B, z_B)$ have the same name $a(z, z)$, the same source (the empty list of sorts), but they have different targets, namely N and B.

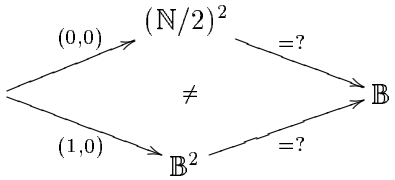
Example 2.10 Let us enrich the overloaded specification $\zeta_2 : T_2 \longrightarrow S_2$, from example 2.8, in order to deal with equalities as operations in the specification. For this purpose, two operations $eq_N : N, N \longrightarrow B$ and $eq_B : B, B \longrightarrow B$ are added to T_2. This leads to a new overloaded specification $\zeta_3 : T_3 \longrightarrow S_3$. The model M_2 of T_2 is extended as a model M_3 of T_3 by interpreting eq_N and eq_B as the equality on $\mathbb{N}$ and $\mathbb{B}$, respectively.

Then, both terms $eq_N(a_N(u_N, u_N), z_N) : B$ and $eq_B(a_B(u_B, u_B), z_B) : B$ of T_3 are parallel and have the same name $eq(a(u, u), z)$, but they are not congruent in T_3. Indeed in M_3 they have distinct interpretations, since $1 + 1 = 0$ is true in $\mathbb{N}/2$ and false in $\mathbb{B}$.

in ζ_3 :



in M_3 :



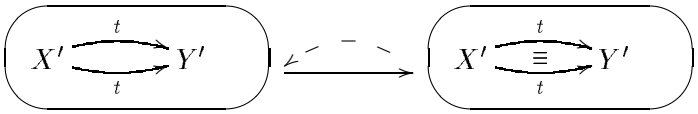
The identification rule

The disambiguation question can be solved by prescribing the following *identification rule*.

Definition 2.11 Let $\zeta : T \longrightarrow S$ be an overloaded specification. The *identification rule* is:

If two parallel terms t' and t'' are homonymous,
then $t' \equiv t''$.

This rule can be illustrated as:



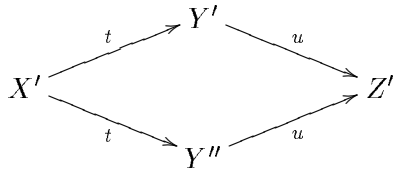
Then, homonymous parallel terms always have the same interpretation, so that any of them is "as good as" any other.

Usually, ζ is built in such a way that distinct parallel *operations* do not have the same name, but this becomes false for *terms*.

The diamond example

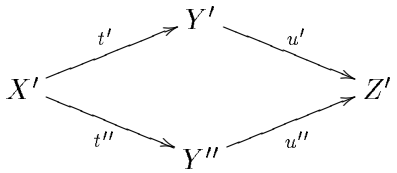
The basic example for studying the disambiguation question is the following *diamond* situation:

ζ :



which means that:

T :



S : $X \xrightarrow{t} Y \xrightarrow{u} Z$

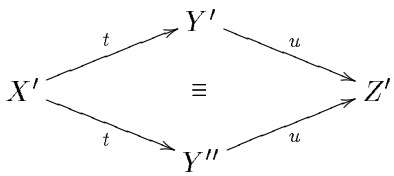
where ζ maps t', t'' to t, and u', u'' to u.

Then there are two terms of ζ with name $u \circ t$ and profile $X' \longrightarrow Z'$, which are not congruent in T. Indeed, it is easy to find a model of T where the interpretations of $u' \circ t'$ and $u'' \circ t''$ are distinct (see example 2.10).

This can be stated as follows: under the easily-checked assumption that there is at most one operation with a given name and a given profile in ζ, the operations are easily disambiguated by their name and profile, but this does not hold for terms. In this example, the notation $u \circ t : X' \longrightarrow Z'$ is ambiguous, it gets disambiguated as $(u : Y' \longrightarrow Z') \circ (t : X' \longrightarrow Y')$ or $(u : Y'' \longrightarrow Z') \circ (t : X' \longrightarrow Y'')$.

Now, let us build a new overloaded specification $\zeta^{\equiv} : T^{\equiv} \longrightarrow S$, where $T^{\equiv}$ is T together with the equation $u' \circ t' \equiv u'' \circ t''$:

$\zeta^{\equiv}$:



In any model of $T^{\equiv}$, both terms with name $u \circ t$ and profile $X' \longrightarrow Z'$ have the same interpretation. The models of $\zeta^{\equiv}$ are the models M of ζ such that $M(u') \circ M(t') = M(u'') \circ M(t'')$. Then, at the specification level, the ambiguity among both terms $u \circ t : X' \longrightarrow Z'$ does not matter.

The name-driven semantics of an overloaded specification

The identification rule is used now, in order to severely modify the semantics of overloaded specifications.

Definition 2.12 The *name-driven models* of an overloaded equational specification $\zeta : T \longrightarrow S$ are the models M of the equational specification T such that, moreover, $M(t') = M(t'')$ as soon as t' and t'' are two parallel homonymous terms of ζ.

Let $\zeta : T \longrightarrow S$ be an overloaded specification. By applying the equational and identification rules to ζ, a morphism of equational theories $F^{\equiv}(\zeta) : T^{\equiv} \longrightarrow F(S)$ is generated, which satisfies the identification property. The equational theory $T^{\equiv}$ differs from $F(T)$: it has more equations, and it depends on the whole ζ (not only on T).

Definition 2.13 The *name-driven theory* of an overloaded specification ζ is $F^{\equiv}(\zeta)$.

Then, it is easy to check that the name-driven models of ζ are the models of the equational specification $T^{\equiv}$. However, as in example 2.16, $T^{\equiv}$ can be infinite even when ζ is finite.

The name-driven semantics of ζ is usually fairly different from its plain semantics:

Proposition 2.14 *With respect to the plain models of ζ, the identification rule is unsound.*

Proof. It can happen that a plain model of ζ does not satisfy the congruence rule: this does happen in the diamond example. □

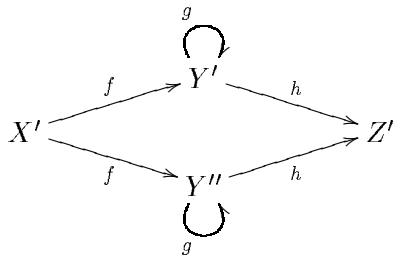
Example 2.15 In the overloaded specification $\zeta_3 : T_3 \longrightarrow S_3$ from example 2.10, both terms $eq_N(a_N(u_N, u_N), z_N) : B$ and $eq_B(a_B(u_B, u_B), z_B) : B$ are parallel and have the same name $eq(a(u, u), z)$, but they are not congruent. In $F^{\equiv}(\zeta)$, the equation $eq_N(a_N(u_N, u_N), z_N) \equiv eq_B(a_B(u_B, u_B), z_B)$ is added. Clearly, the plain model M_3 of ζ_3 is not a name-driven model of ζ_3.

Approximation of the name-driven semantics

The identification rule, like for instance the substitution rule, is a pattern for generating congruences, and it may generate an infinite number of congruences. The next example illustrates this property: even when ζ is finite, there is usually no finite overloaded specification $\zeta^{\equiv}$ such that $F^{\equiv}(\zeta) = F(\zeta^{\equiv})$. This proves that finite overloaded specifications with their name-driven semantics, are strictly more expressive than finite equational specifications.

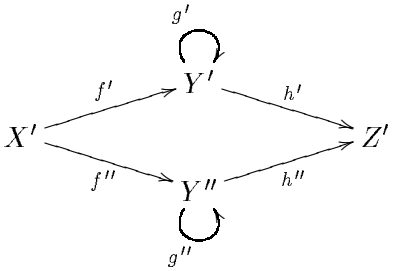
Example 2.16 Let ζ_4 be the following overloaded specification:

ζ_4 :

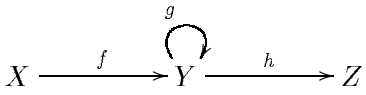


which means that:

T_4 :



S_4 :



and ζ_4 maps X' to X, Y' and Y'' to Y, Z' to Z, f' and f'' to f, g' and g'' to g, h' and h'' to h. Then, the smallest equational specification $\zeta_4^{\equiv}$ such that $F^{\equiv}(\zeta_4) = F(\zeta_4^{\equiv})$ is built by adding to T_4 the equations:

$$h' \circ g' \circ \cdots \circ g' \circ f' \equiv h'' \circ g'' \circ \cdots \circ g'' \circ f'' \ ,$$

with n copies of g' and g'', for every $n \geq 0$, so that $\zeta_4^{\equiv}$ is infinite.

Both semantics may coincide

It can happen that the plain and the name-driven semantics coincide. The following condition is well-known.

Proposition 2.17 *Let $\zeta : T \longrightarrow S$ be an overloaded specification such that:*

If two operations f' and f'' have the same source and the same name, then they have the same target and $f' \equiv f''$.

Then, the plain and the name-driven semantics of ζ coincide.

Proof. It is easy to check that if the operations of ζ satisfy this property, then its terms also satisfy it, hence two homonymous parallel terms are congruent. □

A well-known drawback of this result is that constants are not allowed to be overloaded, since they all have the same (empty) source. For instance, 0 cannot be both the name of an integer and the name of a boolean. For this reason it is not assumed, in this paper, that both semantics coincide.

Example 2.18 The overloaded specification $\zeta_3 : T_3 \longrightarrow S_3$ from example 2.10 does not satisfy the condition of proposition 2.17. It can be modified, by adding the names $t, f : X$ in S and by naming 0_B and 1_B as t and f, respectively, instead of 0 and 1. Then the terms $eq_N(a_N(u_N, u_N), z_N) : B$ and $eq_B(a_B(u_B, u_B), z_B) : B$ are still parallel, but they have distinct names $1 + 1 \equiv 0$ and $t + t \equiv f$, so that there is no ambiguity.

2.4 Subtypes and the plain semantics of subtyping

Subtypes and coercions

A *subtype* Y' of a type Y'' is such that a value of type Y' can be used in any context in which a value of type Y'' is expected: this is the *subtyping property*, sometimes called the *subsumption property* [Bruce 2002]. This means that there is some canonical way, in every model M, to identify each element of $M(Y')$ to an element of $M(Y'')$. It is viewed as a *coercion* from Y' to Y'', i.e., an implicit type conversion, so that it is an operation, or a term, which does not appear in the notations. The identification of each element of $M(Y')$ to an element of $M(Y'')$ can be done through an inclusion, like $\mathbb{N} \subseteq \mathbb{Z}$. It can also be done through a canonical surjection, like $\mathbb{Z} \longrightarrow \mathbb{Z}/2\mathbb{Z}$, where $\mathbb{Z}/2\mathbb{Z} = \{0, 1\}$, which maps each integer to 0 or to 1 according to its parity. More generally, a coercion can stand for any map, as soon as this map is considered as *canonical*, in the sense that it may be skipped from the notations.

From the point of view of notations, this means that a coercion behaves like an identity. This is easy to state in the framework of overloaded specifications:

Definition 2.19 In an overloaded specification, a *coercion* is a term such that its name is an identity.
A sort Y' is a *subtype* of a sort Y'' if there is a coercion from Y' to Y''; this is generalized to lists of sorts.

Let $\zeta : T \longrightarrow S$ be an overloaded specification, a coercion is a term $\varepsilon : Y' \longrightarrow Y''$ in T such that $\zeta(\varepsilon) = \mathrm{id}_Y : Y \longrightarrow Y$ in S, hence in ζ it is denoted $\mathrm{id}_Y : Y' \longrightarrow Y''$.

The following properties of coercions follow directly from their definition, and from the properties of identities in an equational specification.

Proposition 2.20
– reflexivity of subtyping: *every identity of T is a coercion,*
– transitivity of subtyping: *the composition of two coercions is a coercion.*
– subtyping and record types: *the product of two coercions is a coercion.*

The subtyping rules

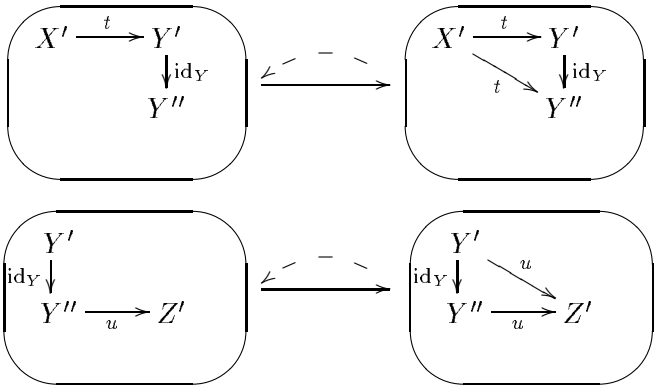
The subtyping property can be prescribed, thanks to the two following rules, which correspond to two complementary views of the subtyping property.

Definition 2.21 Let $\zeta : T \longrightarrow S$ be a specification with coercions. The *subtyping rules* are:

If there is a coercion $\mathrm{id}_Y : Y' \longrightarrow Y''$,
and if $t : X' \longrightarrow Y'$,
then t **is overloaded by** $t : X' \longrightarrow Y''$.

If there is a coercion $\mathrm{id}_Y : Y' \longrightarrow Y''$
and if $u : Y'' \longrightarrow Z'$,
then u **is overloaded by** $u : Y' \longrightarrow Z'$.

They can be illustrated as:



In [Hintermeier et al. 1998], the coercions are expressed by ways of *decoration rules*, which can be compared to our first subtyping rule.

The plain semantics of subtyping

Let $\zeta : T \longrightarrow S$ be an overloaded specification. By applying the equational and subtyping rules to ζ, a morphism of equational theories $F_{st}(\zeta) : T_{st} \longrightarrow F(S)$ is generated, which satisfies the subtyping property.

Definition 2.22 The *plain theory with subtyping* of an overloaded specification ζ is $F_{st}(\zeta)$. The *plain models with subtyping* of ζ are the models of the equational specification T_{st}.

In this way, a *canonical map* can be defined as the image of a coercion: this definition depends both on the overloaded specification and on its model.

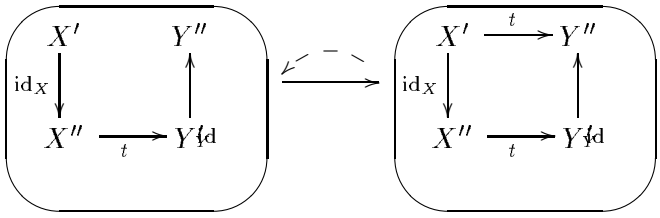
This semantics allows the overridding of operations: within the subtyping rules, no relation is required between the interpretations of both terms $t : X' \longrightarrow Y'$ and $t : X' \longrightarrow Y''$, and similarly for $u : Y'' \longrightarrow Z'$ and $u : Y' \longrightarrow Z'$.

In addition, a unique plain model of ζ, in the sense of part 2.2, may give rise to many plain models with subtyping, since there is no prescription on the interpretation of the operations which are generated by the subtyping rules.

The contravariant property of coercions

Clearly, the subtyping rules have the following consequence, which can be called the *contravariant property of coercions* [Castagna 1995]:

Proposition 2.23 *From the subtyping rules, if there are two coercions* $\mathrm{id}_X : X' \longrightarrow X''$ *and* $\mathrm{id}_Y : Y' \longrightarrow Y''$*, and if* $t : X'' \longrightarrow Y'$*, then* $t : X' \longrightarrow Y''$.



2.5 The name-driven semantics of subtyping

The coherence rules

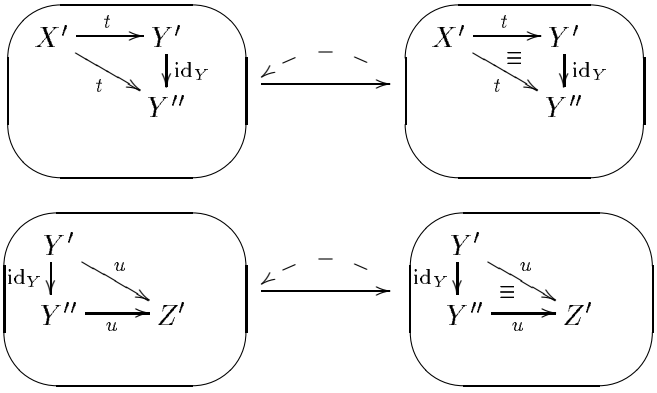
Another widely used semantics for subtyping prescribes that homonymous operations, when their profile differs only by a coercion, should get the same interpretation. More precisely, this semantics is obtained from the following rules.

Definition 2.24 Let $\zeta : T \longrightarrow S$ be an overloaded specification. The *coherence rules* are:

If there is a coercion $\mathrm{id}_Y : Y' \longrightarrow Y''$**,**
and if $t : X' \longrightarrow Y'$ **and** $t : X' \longrightarrow Y''$**,**
then $\mathrm{id}_Y \circ t \equiv t$**.**

If there is a coercion $\mathrm{id}_Y : Y' \longrightarrow Y''$
and if $u : Y'' \longrightarrow Z'$ **and** $u : Y' \longrightarrow Z'$,
then $u \circ \mathrm{id}_Y \equiv u$.

They can be illustrated as:



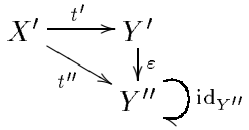
It is now proven that any one of these rules is equivalent to the identification rule. As a consequence, both coherence rules are equivalent.

Theorem 2.25 *In an overloaded specification, the identification property is satisfied if and only if one of the coherence properties is satisfied.*

Proof. For clarity, the proof is written in T rather than in ζ. Moreover, only the first coherence property is considered, since the proof for the second one is similar.

First, let us assume that the first coherence property is satisfied, and let us prove the identification property. Let $t' : X' \longrightarrow Y'$ and $t'' : X' \longrightarrow Y'$ be two parallel terms with the same name t. Since the identity $\mathrm{id}_{Y'}$ is a coercion, the first coherence property ensures that $\mathrm{id}_{Y'} \circ t' \equiv t''$. But $\mathrm{id}_{Y'} \circ t' = t'$, so that indeed $t' \equiv t''$.

Now, let us assume that the identification property is satisfied, and let us prove the first coherence property. Let $\varepsilon : Y' \longrightarrow Y''$ be a coercion, and let $t' : X' \longrightarrow Y'$ and $t'' : X' \longrightarrow Y''$ be two terms with the same name t. Then both $\varepsilon \circ t' : X \longrightarrow Y''$ and $\mathrm{id}_{Y''} \circ t'' : X \longrightarrow Y''$ are parallel and have the same name $\mathrm{id}_Y \circ t$, hence, from the identification property, $\varepsilon \circ t' \equiv \mathrm{id}_{Y''} \circ t''$. But $\mathrm{id}_{Y''} \circ t'' = t''$, so that indeed $t'' \equiv \varepsilon \circ t'$.



□

The name-driven semantics of subtyping

Let $\zeta : T \longrightarrow S$ be an overloaded specification. By applying the equational, subtyping and identification rules to ζ, a morphism of equational theories $F^{\equiv\equiv}_{st}(\zeta) : T^{\equiv\equiv}_{st} \longrightarrow F(S)$ is generated, which satisfies the subtyping and identification properties.

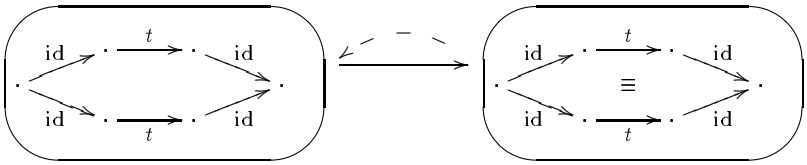
Definition 2.26 The *name-driven theory with subtyping* of an overloaded specification ζ is $F_{st}^{\equiv}(\zeta)$.
The *name-driven models with subtyping* of ζ are the models of the equational specification $T_{st}^{\equiv}$.

Some properties of coercions

The identification rule has various consequences, here are some of them.

Proposition 2.27 *From the identification rule:*
– if there are two coercions from Y' to Y'', then they are congruent,
– if there is a coercion from Y' to Y', then it is congruent to the identity of Y',
– antisymmetry of subtyping, up to congruence: *if there is a coercion from Y' to Y'' and a coercion from Y'' to Y', then they are inverses, up to congruence.*

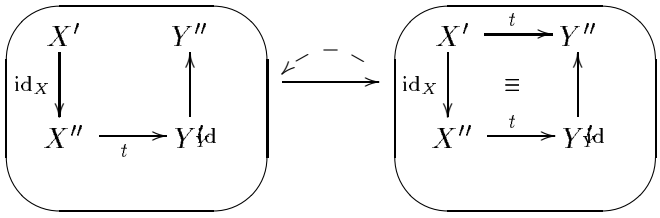
As an example, the following property is an application of the identification rule:



The contravariant property of coercions, revisited

The subtyping rules together with the coherence rules yield an enforced version of the *contravariant property of coercions*:

Proposition 2.28 *From the subtyping and coherence rules, if there are two coercions* $\mathrm{id}_X : X' \longrightarrow X''$ *and* $\mathrm{id}_Y : Y' \longrightarrow Y''$, *and if* $t : X'' \longrightarrow Y'$, *then* $t : X' \longrightarrow Y''$ *with* $\mathrm{id}_Y \circ t \circ \mathrm{id}_X \equiv t$.

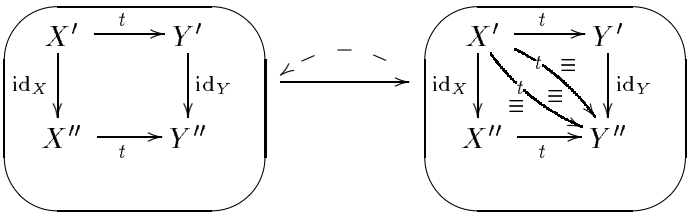


The covariant property of coercions

On the other hand, the subtyping, coherence and identification rules have the following consequence, which can be called the *covariant property of coercions* [Castagna 1995]:

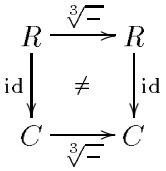
Proposition 2.29 *From the subtyping, coherence and identification rules, if there are two coercions* $\mathrm{id}_X : X' \longrightarrow X''$ *and* $\mathrm{id}_Y : Y' \longrightarrow Y''$, *and if* $t : X' \longrightarrow Y'$ *and* $t : X'' \longrightarrow Y''$, *then* $\mathrm{id}_Y \circ t \equiv t \circ \mathrm{id}_X$.

Proof. The first subtyping and coherence rules give rise to $t : X' \longrightarrow Y''$ with $t \equiv \mathrm{id}_Y \circ t$, and the second subtyping and coherence rules give rise to $t : X' \longrightarrow Y''$ with $t \equiv t \circ \mathrm{id}_X$. Then the identification rule yields the congruence $\mathrm{id}_Y \circ t \equiv t \circ \mathrm{id}_X$. □



This can be understood as follows: a coercion behaves as a morphism, with respect to the homonymous operations. Quite often, this situation occurs when the overloading comes from parametric polymorphism.

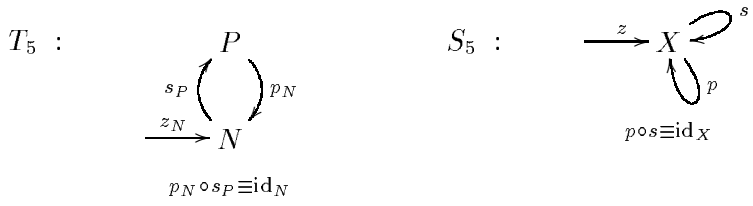
Example 2.30 Here is a well-known example, where this property is *not* satisfied. The same name $\sqrt[3]{-}$ is used for cubic roots, either over the reals, or over the complex numbers. Since it is required that $(\sqrt[3]{x})^3 = x$ for all x, there is no choice for the definition of $\sqrt[3]{-}$ over the reals. Usually, the definition of $\sqrt[3]{-}$ over the complex is such that the minimal nonnegative argument is chosen. Then, the value of $\sqrt[3]{-1}$ is -1 over the reals, but it is $\exp(i\pi/3)$ over the complex:



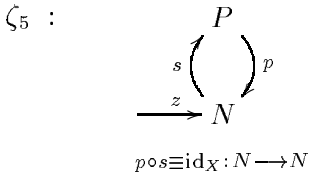
A classical example

Let us now look at the classical example of dealing with $pred(pred(succ(succ(0))))$ over the naturals: clearly this number is equal to 0, however it is not so easy to get it as the image of a well-formed term. This will be done progressively in examples 2.31, 2.32 and 2.38.

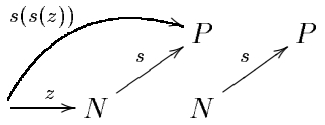
Example 2.31 Let us consider the overloaded specification $\zeta_5 : T_5 \longrightarrow S_5$:



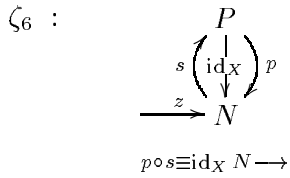
where ζ_5 maps P, N, z_N, s_P and p_N to X, X, z, s and p, respectively, so that it is illustrated as:



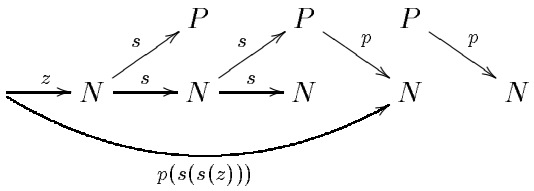
Then:
– the term $s(z) : X$ of S_5 is the name of the term $s_P(z_N) : P$,
– but the term $s(s(z)) : X$ of S_5 is not the name of any term.



Example 2.32 Now, let us add a coercion from P to N, so that ζ_5 becomes ζ_6:



Since s is the name of an operation $s_P : N \longrightarrow P$, the first subtyping rule asserts that an operation $s_N : N \longrightarrow N$ with the same name s can be added. Then:
– the term $s(z) : X$ of S_6 is the name of both terms $s_P(z_N) : P$ and $s_N(z_N) : N$,
– the term $s(s(z)) : X$ of S_6 is the name of both terms $s_P(s_N(z_N)) : P$ and $s_N(s_N(z_N)) : N$,
– and the term $p(s(s(z))) : X$ of S_6 is the name of the term $p_N(s_P(s_N(z_N))) : N$, which is congruent to the term $s_N(z_N) : N$,
– but the term $p(p(s(s(z)))) : X$ of S_6 is not the name of any term.



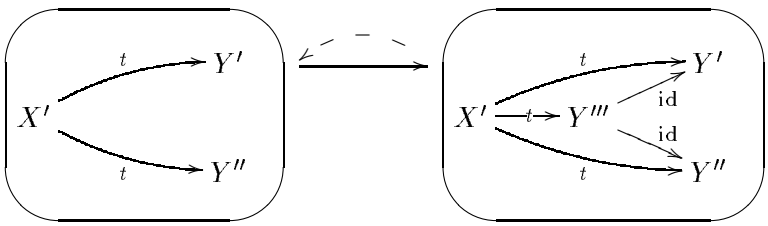
The sort-inheritance rule

In [Hintermeier et al. 1998], a *sort-inheritance* property is required, in order to find a least subsort for a term; this property is used in the decorated unification algorithm.

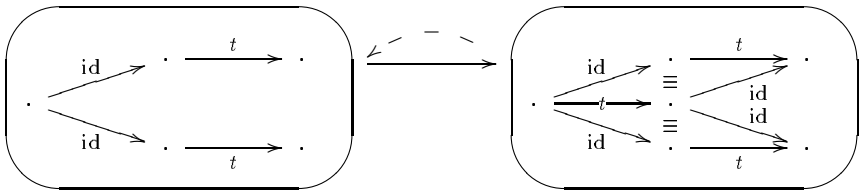
In a similar way, here, the following *sort-inheritance rule* could be added, which would change the semantics:

If $t : X' \longrightarrow Y'$ ***and*** $t : X' \longrightarrow Y''$***,***
then t ***is overloaded by*** $t : X' \longrightarrow Y'''$
with coercions $\mathrm{id}_Y : Y''' \longrightarrow Y'$ ***and*** $\mathrm{id}_Y : Y''' \longrightarrow Y''$***.***

More precisely, this property means that there is some way to choose Y''', $t : X' \longrightarrow Y'''$, $\mathrm{id}_Y : Y''' \longrightarrow Y'$ and $\mathrm{id}_Y : Y''' \longrightarrow Y''$, from $t : X' \longrightarrow Y'$ and $t : X' \longrightarrow Y''$.



As an example, the following property is an application of the subtyping, sort-inheritance and identification rules:



Extensions of overloaded specifications by definition

Before proving that the name-driven semantics of subtyping is not much more than the name-driven semantics from part 2.3, let us come back to the extensions by definition, in the case of overloaded specifications.

As for an equational specification, an overloaded specification can be *extended by definition*. This means that a new operation f' is introduced in T, together with a *defining equation* $f' \equiv t'$, where t' is a term which does not involve f'. The name of f' can be either in S, or in an extension by definition of S. Such an extension does not modify the plain semantics of ζ, up to congruence. However it can modify the name-driven semantics: for instance if t'_1 and t'_2 are non-congruent parallel terms, and if f'_1 and f'_2 are defined by $f'_1 \equiv t'_1$ and $f'_2 \equiv t'_2$, where both f'_1 and f'_2 have the same name, then the identification property adds the equation $f'_1 \equiv f'_2$, from which a new congruence $t'_1 \equiv t'_2$ is derived.

Definition 2.33 An extension of an overloaded specification $\zeta : T \longrightarrow S$ by an operation $f' : X' \longrightarrow Y'$ with name f and with a defining equation $f' \equiv t'$ is *non-ambiguous* if, before this extension, for any term $f'_1 : X' \longrightarrow Y'$ with name f, the congruence $f'_1 \equiv t'$ holds in T.

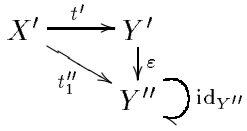
If an extension by definition is non-ambiguous, then it does not modify the name-driven semantics of ζ, up to congruence.

The subtyping rules from the identification rule

The following result proves that the subtyping rules can be derived from the identification rule.

Theorem 2.34 *In an overloaded specification $\zeta : T \longrightarrow S$, let $\varepsilon : Y' \longrightarrow Y''$ be a coercion.*
– If there is a term $t' : X' \longrightarrow Y'$ with name t, then the extension of ζ by an operation $t'' : X' \longrightarrow Y''$ with the defining equation $t'' \equiv \varepsilon \circ t'$ and with name t, is non-ambiguous.
– If there is a term $u'' : Y'' \longrightarrow Z'$ with name u, then the extension of ζ by an operation $u' : Y' \longrightarrow Z'$ with the defining equation $u' \equiv u'' \circ \varepsilon$ and with name u, is non-ambiguous.

Proof. Let us prove the first assertion, the proof of the second one is similar. Let $t''_1 : X' \longrightarrow Y''$ be a term with name t, and let us prove that $t''_1 \equiv \varepsilon \circ t'$. Both $\varepsilon \circ t' : X \longrightarrow Y''$ and $\mathrm{id}_{Y''} \circ t''_1 : X \longrightarrow Y''$ are parallel and have the same name $\mathrm{id}_Y \circ t$, hence, from the identification rule, $\varepsilon \circ t' \equiv \mathrm{id}_{Y''} \circ t''_1$. But $\mathrm{id}_{Y''} \circ t''_1 = t''_1$, so that indeed $t''_1 \equiv \varepsilon \circ t'$.



□

Theorem 2.34 means that the name-driven semantics with subtyping is essentially the same as the name-driven semantics from part 2.3.

2.6 Subsorts and the name-driven semantics of subsorting

Subsorts and sort inclusions

The notion of *subsort* refers to a more restricted property between sorts, or types, than the notion of subtype. It aims at formalizing the set-theoretic notion of *subset*. Whenever a sort Y' is a subsort of a sort Y'', then in each model, the interpretation B' of Y' should be a subset of the interpretation B'' of Y'', which means that there should be an *inclusion* from B' to B''. In fact, a *sort inclusion* is defined below as a monomorphic coercion, so that its interpretation is a *canonical injection*, i.e., an *implicit embedding*, i.e., an injection which may be skipped from the notation.

It is proven below that, with respect to the name-driven semantics for subtyping, the injectivity can be expressed as a *retraction rule*.

Specifications with subsorts

Since a canonical map can be formalized as a coercion $\mathrm{id}_Y : Y' \longrightarrow Y''$, a sort inclusion is now defined as a coercion which satisfies some additional property. Hence, the sort inclusions are some coercions among all the coercions of a given overloaded specification.

Definition 2.35 A *specification with subsorts* is an overloaded specification where some coercions are called *sort inclusions*.
In a specification with subsorts, a sort Y' is a *subsort* of a sort Y'' if there is a sort inclusion from Y' to Y''; this is generalized to lists of sorts.

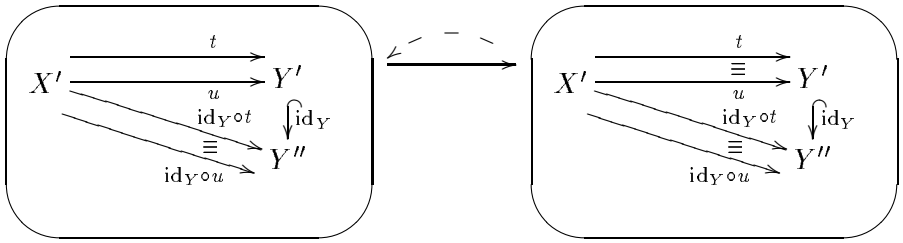
For simplicity, a specification with subsorts is still denoted $\zeta : T \longrightarrow S$, although this notation does not mention the chosen sort inclusions. In this paper, a sort inclusion is represented in T as $\varepsilon : Y' \hookrightarrow Y''$, and in ζ as $\mathrm{id}_Y : Y' \hookrightarrow Y''$

The monomorphism rule and the retraction rule

The traditional way to assert that an operation, for instance a sort inclusion $\mathrm{id}_Y : Y' \hookrightarrow Y''$, has to be interpreted as an injection, is the following *monomorphism rule*:

$$\textbf{If } t, u : X' \longrightarrow Y' \textbf{ are such that } \mathrm{id}_Y \circ t \equiv \mathrm{id}_Y \circ u\textbf{,}$$
$$\textbf{then } t \equiv u\textbf{.}$$

The monomorphism rule can be illustrated as follows:

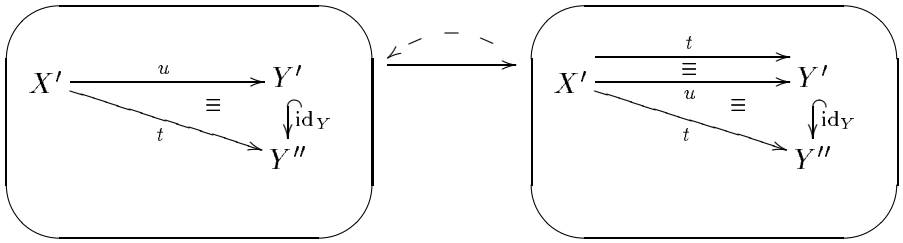


However, the rule which is used for dealing with subsorts is not the monomorphism rule, but the following *retraction rule*:

If $t : X' \longrightarrow Y''$ **is such that** $t \equiv \mathrm{id}_Y \circ u$ **for some** $u : X' \longrightarrow Y'$**,**
then t **is overloaded by** $t : X' \longrightarrow Y'$ **with** $t \equiv u$**.**

More precisely, the retraction rule means that there is some way to choose $t : X' \longrightarrow Y'$ from $t : X' \longrightarrow Y''$ and $u : X' \longrightarrow Y'$.

It can be illustrated as follows:



In [Hintermeier et al. 1998], the rule which is expressed as : "equal terms belong to the same sort", is similar to our retraction rule.

The equivalence of both rules

The next result proves the equivalence of both rules, when dealing with coercions.

Theorem 2.36 *In the name-driven theory with subtyping of an overloaded specification:*
– if a coercion satisfies the retraction rule, then it satisfies the monomorphism rule.
– if a coercion satisfies the monomorphism rule, then it satisfies the retraction rule, up to a non-ambiguous extension by definition.

Proof. For clarity, the proof is written in T rather than in ζ. The name of t', t'', t'_1, is t, and the name of u', u'', is u. Let $\varepsilon : Y' \longrightarrow Y''$ be a coercion.

First, let us assume that ε satisfies the retraction rule. Let $t', u' : X' \longrightarrow Y'$ be two terms such that $\varepsilon \circ t' \equiv \varepsilon \circ u'$, then we have to prove that $t' \equiv u'$. Since ε is a coercion, the subtyping and coherence rules add $t'' : X' \longrightarrow Y''$ with $t'' \equiv \varepsilon \circ t'$. Since it is assumed that $\varepsilon \circ t' \equiv \varepsilon \circ u'$, it follows that $t'' \equiv \varepsilon \circ u'$. Now, the retraction rule can be used: t'' can be overloaded by $t'_1 : X' \longrightarrow Y'$ with $t'_1 \equiv u'$. Since t' and t'_1 are parallel and homonymous, the identification rule adds the equation $t' \equiv t'_1$. Finally, since $t' \equiv t'_1$ and $t'_1 \equiv u'$, it follows that $t' \equiv u'$.

Now, let us assume that ε satisfies the monomorphism rule. Let $t'' : X' \longrightarrow Y''$ and $u' : X' \longrightarrow Y'$ with $t'' \equiv \varepsilon \circ u'$, then we have to prove that, up to a non-ambiguous extension by definition, t'' can be overloaded by $t' : X' \longrightarrow Y'$ with $t' \equiv u'$. Since ε is a coercion, the subtyping and coherence rules add $u'' : X' \longrightarrow Y''$ with $u'' \equiv \varepsilon \circ u'$. Since $t'' \equiv \varepsilon \circ u'$, it follows that $t'' \equiv u''$. Let us consider the extension by definition of an operation $t' : X' \longrightarrow Y'$ such that $t' \equiv u'$. We have to prove that it is non-ambiguous. Let $t'_1 : X' \longrightarrow Y'$, then both $\mathrm{id}_{Y''} \circ t''$ and $\varepsilon \circ t'_1$ are parallel and homonymous, hence the identification rule adds the equation $\mathrm{id}_{Y''} \circ t'' \equiv \varepsilon \circ t'_1$. But $\mathrm{id}_{Y''} \circ t'' = t''$, and $t'' \equiv \varepsilon \circ u'$ from the hypothesis, so that $\varepsilon \circ t'_1 \equiv \varepsilon \circ u'$. Now, the monomorphism rule can be used, it proves that $t'_1 \equiv u'$, so that indeed the extension is non-ambiguous. □

The name-driven semantics of specifications with subsorts

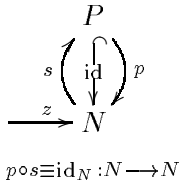
Let $\zeta : T \longrightarrow S$ be a specification with subsorts. By applying the equational, identification, subtyping and retraction rules to ζ, a specification with subsorts $F^{\equiv}_{ss}(\zeta) : T^{\equiv}_{ss} \longrightarrow F(S)$ is generated, which satisfies the identification, subtyping and retraction properties.

Definition 2.37 The *name-driven theory* of a specification with subsorts ζ is $F^{\equiv}_{ss}(\zeta)$.
The *name-driven models* of ζ are the models of the equational specification $T^{\equiv}_{ss}$.

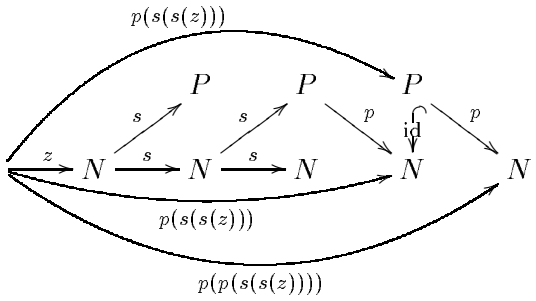
So, the interpretation of a sort inclusion in such a model is an injection.

Example 2.38 Let us build ζ_7 from ζ_6, by adding that the coercion form P to N is a sort inclusion:

ζ_7 :



Since the congruence $p(s(s(z))) \equiv s(z) : N$ can be derived from $p \circ s \equiv \mathrm{id}_N : N \longrightarrow N$, and since $s(z) : N$ is overloaded by $s(z) : P$, it follows from the retraction rule that the term $p(s(s(z))) : N$ can be overloaded by $p(s(s(z))) : P$. This means that an operation $u_P : P$ can be added to T_7, with name $\zeta_7(u_P) = p(s(s(z)))$. Hence the term $p(p(s(s(z)))) : X$ of S_7 is the name of the term $p_N(u_P) : P$.



Static vs. dynamic typing

With this approach, the introduction of partially defined *retraction operations*, as partial inverses for sort inclusions, is avoided. It is well known that retraction operations are rather difficult

to handle, mainly because of their partiality. A slightly different point of view is chosen in [Casl 2003], where a term can be *casted* from a sort to a subsort, thanks to a partial casting operation.

In this paper, the keypoint for avoiding retraction operations is that, as in the previous example, an operation can have a composed name without being itself a composed operation: the image via ζ of an operation of T can be any term of S.

Moreover, the retraction rule clearly illustrates an intrinsic difficulty in the use of subsorts: the following *typing question* cannot be solved statically in a specification with subsorts:

If t is a term of S, and $X \longrightarrow Y$ a profile in T,
is there a term with name t and profile $X \longrightarrow Y$?

Indeed, in overloaded specifications, even with the subtyping and identification rules, the rules for forming terms only depend on terms, not on congruences. It follows that the typing question can be solved statically, i.e., without any computation, i.e., without using the equations. This becomes false with the retraction rule: indeed, the hypothesis of this rule contains an equation $t \equiv \mathrm{id}_Y \circ u$, while its conclusion generates a new operation $t : X' \longrightarrow Y'$.

3 Overloaded specifications as diagrammatic specifications

In section 2, overloaded equational specifications are defined as morphisms of equational specifications. A drawback of this approach is that overloaded specifications are not of the same nature as ordinary specifications.

In the context of *diagrammatic specifications*, as introduced in [Duval and Lair 2002], it is possible to define both ordinary specifications and overloaded specifications as two families of diagrammatic specifications. Then, the deduction rules for ordinary specifications are an instance of the deduction rules for diagrammatic specifications, and the deduction rules for overloaded specifications are another instance of the deduction rules for diagrammatic specifications.

There is a general notion of *model* of a diagrammatic specification. The models of an ordinary specification coincide with its models as a diagrammatic specification. But the models of an overloaded specification are different from its models as a diagrammatic specification. It is the *zooming process*, as introduced in [Duval et al. 2003], which yields the right definition for the models.

Moreover, this new point of view on overloaded specifications is not restricted to the equational frame.

The basic features of diagrammatic specifications are presented in part 3.1. As an example, equational specifications are presented as diagrammatic specifications in part 3.2. Then, overloaded specifications, their plain semantics and their resolved semantics are revisited in parts 3.3 and 3.4. The same technique is used for subtyping and subsorting semantics in part 3.5. Finally, a non-equational example is considered in part 3.6.

3.1 About diagrammatic specifications and their semantics

Projective sketches and propagators

Diagrammatic specifications are defined below, as realizations of meta-level specifications (the word "realization" is used at the meta-level, instead of "model"). The basic tool for building diagrammatic specifications is the theory of *sketches* [Ehresmann 1966], [Coppey and Lair 1984], [Coppey and Lair 1988]. On the one hand, diagrammatic specifications generalize sketches, and on the other hand, they are defined by means of *projective sketches* at the meta-level: a diagrammatic specification *has* models, and at the same time it *is* a realization of a projective sketch. Diagrammatic specifications are introduced in [Duval and Lair 2002], as summarized below.

Definition 3.1 A *(directed) graph* is made of points and arrows, in such a way that each arrow f has a *profile* $X \longrightarrow Y$, made of its *source* X and its *target* Y; this is denoted $f : X \longrightarrow Y$.
A *morphism of graphs* $\gamma : \mathcal{G} \longrightarrow \mathcal{G}'$ maps each point G of $\mathcal{G}$ to a point $\gamma(G)$ of G', and each arrow $g : G_1 \longrightarrow G_2$ of $\mathcal{G}$ to an arrow $\gamma(g) : \gamma(G_1) \longrightarrow \gamma(G_2)$ of G'.

Definition 3.2 A *compositive graph* is a graph where some points X have an *identity arrow* $\mathrm{id}_X : X \longrightarrow X$ and some consecutive pairs of arrows $(f : X \longrightarrow Y, g : Y \longrightarrow Z)$ have a *composed arrow* $g \circ f : X \longrightarrow Z$. In a compositive graph, there is no assumption about associativity and unitary properties of composition and identities.
A *morphism of compositive graphs* is a morphism of directed graphs which preserves the identity arrows and the composed arrows.

Definition 3.3 A *cone* in a compositive graph $\mathcal{G}$ is made of a *vertex* point G, a *base* morphism $b : \mathcal{I} \longrightarrow \mathcal{G}$, where the compositive graph $\mathcal{I}$ is called the *index*, and *projection* arrows $p_I : G \longrightarrow b(I)$ for each point I in $\mathcal{I}$, such that $b(i) \circ p_I = p_{I'}$ for each arrow $i : I \longrightarrow I'$ in $\mathcal{I}$.

Definition 3.4 A *projective sketch* is a compositive graph where some cones are called *distinguished cones*.
A *morphism of projective sketches*, or *propagator*, is a morphism of compositive graphs which preserves the distinguished cones.

Definition 3.5 A *category* is a compositive graph with an identity arrow for each point, a composed arrow for each consecutive pair of arrows, which satisfies the associativity and unitary axioms:
– if (f, g) and (g, h) are consecutive, then $(h \circ g) \circ f = h \circ (g \circ f)$,
– if $f : X \longrightarrow Y$, then $f \circ \mathrm{id}_X = f$ and $\mathrm{id}_Y \circ f = f$.
A *morphism of categories*, or *functor*, is a morphism of compositive graphs.

For instance, up to some care about the size, the sets and maps form a category, which is called *the category of sets*.

In a category, some cones are called *limits*, and a limit gives rise to *factorization arrows*. For instance, a *product* is a limit; it is a cone:

$$\begin{array}{ccccc} & & P & & \\ & {\scriptstyle p_1}\swarrow & & \searrow{\scriptstyle p_n} & \\ Y_1 & & \cdots & & Y_n \end{array}$$

for some integer $n \geq 0$, such that, for each cone with the same base:

$$\begin{array}{ccccc} & & X & & \\ & {\scriptstyle f_1}\swarrow & & \searrow{\scriptstyle f_n} & \\ Y_1 & & \cdots & & Y_n \end{array}$$

there is a unique arrow $(f_1, \ldots, f_n) : X \longrightarrow P$, called the *factorization* of $f_1, \ldots, f_n$, such that $p_i \circ (f_1, \ldots, f_n) = f_i$ for $i = 1, \ldots, n$.

When $n = 0$, this property says that for each point X there is a unique arrow from X to P; this means that P is a *terminal point* in the category.

Here is another kind of limit cone:

$$\begin{array}{ccccc} & & P & & \\ & {\scriptstyle id_P}\swarrow & \vert & \searrow{\scriptstyle id_P} & \\ P & & {\scriptstyle id_P}\downarrow & & P \\ & {\scriptstyle m}\searrow & & \swarrow{\scriptstyle m} & \\ & & Q & & \end{array}$$

This limit cone means that $m : P \longrightarrow Q$ is a monomorphism, i.e. whenever $m \circ f_l = m \circ f_r$, then $f_l = f_r$. Then m is represented as $m : P \rightarrowtail Q$.

For instance, in the category of sets:
– a product is a cartesian product, which means that $P = Y_1 \times \cdots \times Y_n$, and $p_1, \ldots, p_n$ are the projections; the factorization arrow $(f_1, \ldots, f_n) : X \longrightarrow Y_1 \times \cdots \times Y_n$ builds n-uples.
– when $n = 0$, P is a singleton, i.e., a one-element set;
– a monomorphism is an injective map.

Definition 3.6 A *(set-valued) realization* S of a projective sketch $\mathcal{E}$ maps each point E of $\mathcal{E}$ to a set $S(E)$ and each arrow $e : E \longrightarrow E'$ of $\mathcal{E}$ to a map $S(e) : S(E) \longrightarrow S(E')$, in such a way that each identity arrow becomes an identity map, each composed arrow becomes a composed map, and each distinguished cone becomes a limit cone.
A *morphism* $\sigma : S_1 \longrightarrow S_2$ *of realizations* of $\mathcal{E}$ is a natural transformation. This means that σ is made of a map $\sigma_E : S_1(E) \longrightarrow S_2(E)$ for each point E of $\mathcal{E}$, in such a way that $S_1(e) \circ \sigma_E = \sigma_{E'} \circ S_2(e)$ for each arrow $e : E \longrightarrow E'$ of $\mathcal{E}$.

The category of realizations of $\mathcal{E}$ is denoted $\mathcal{R}eal(\mathcal{E})$.

Example 3.7 Let $\mathcal{E}_{Gr}$ be the projective sketch:

$$\mathcal{E}_{Gr} \ : \qquad \mathrm{Pt} \underset{\text{tgt}}{\overset{\text{sce}}{\rightleftarrows}} \mathrm{Ar}$$

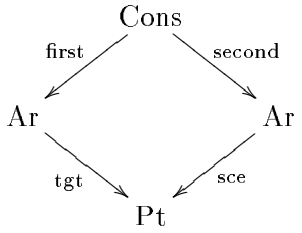
The idea is that Pt and Ar are understood as "points" and "arrows", sce and tgt as "source" and "target", so that the category of realizations of $\mathcal{E}_{Gr}$ is the category of directed graphs.

Example 3.8 The projective sketch $\mathcal{E}_{Gr}$ can be enriched in order to get a projective sketch $\mathcal{E}_{Cat}$ such that the realizations of $\mathcal{E}_{Cat}$ are the categories.

For instance, in order to prescribe that every pair of consecutive arrows can be composed, the enrichment can be performed as follows, in two steps.

First, a point Cons is added to $\mathcal{E}_{Gr}$, for "pairs of consecutive arrows". In order to ensure that it is interpreted as the set of pairs of consecutive arrows, it is accompanied with two arrows

first : Cons $\longrightarrow$ Ar and second : Cons $\longrightarrow$ Ar (first and second arrow in the pair) and with the following distinguished cone with vertex Cons (as often, the diagonal arrow sce $\circ$ second = tgt $\circ$ first : Cons $\longrightarrow$ Pt is omitted):



The realizations of this new projective sketch still are the graphs: this kind of enrichment does not modify the realizations.

Then, an arrow comp : Cons $\longrightarrow$ Ar is added, with the composed arrows:

$$\text{sce} \circ \text{comp} = \text{sce} \circ \text{first} \ , \qquad \text{tgt} \circ \text{comp} = \text{tgt} \circ \text{second} \ .$$

This restricts the realizations of the projective sketch to the graphs where any pair of consecutive arrows is composable.

In a similar way, the whole definition of a category can be expressed in the projective sketch $\mathcal{E}_{Cat}$. This enrichment defines a propagator $P_{Cat} : \mathcal{E}_{Gr} \longrightarrow \mathcal{E}_{Cat}$.

The adjunction associated to a propagator

Let us consider a propagator:

$$P : \mathcal{E} \longrightarrow \overline{\mathcal{E}} \ .$$

The *omitting functor* $G_P : \mathcal{R}eal(\overline{\mathcal{E}}) \longrightarrow \mathcal{R}eal(\mathcal{E})$ is such that $G_P(\mathcal{C}) = \mathcal{C} \circ P$ for all realization $\mathcal{C}$ of $\overline{\mathcal{E}}$.

The *freely generating functor* $F_P : \mathcal{R}eal(\mathcal{E}) \longrightarrow \mathcal{R}eal(\overline{\mathcal{E}})$ is such that for all realization S of $\mathcal{E}$ and all realization $\mathcal{C}$ of $\overline{\mathcal{E}}$:

$$\text{Hom}_{\mathcal{R}eal(\mathcal{E})}(S, G_P(\mathcal{C})) \cong \text{Hom}_{\mathcal{R}eal(\overline{\mathcal{E}})}(F_P(S), \mathcal{C}) \ .$$

$$\mathcal{R}eal(\mathcal{E}) \underset{G_P}{\overset{F_P}{\rightleftarrows}} \mathcal{R}eal(\overline{\mathcal{E}})$$

In categorical terms, this means that F_P is the *left adjoint* of G_P. The existence of this left adjoint is a major property of projective sketches [Ehresmann 1966], [Lair and Duval 2001]. When the propagator P is fixed, the subscript P can be omitted.

Example 3.9 Let $P = P_{Cat} : \mathcal{E}_{Gr} \longrightarrow \mathcal{E}_{Cat}$ be the propagator from example 3.8. The omitting functor G_P maps each category to its underlying graph. The freely generating functor F_P maps each graph to its freely generated category, by adding all the missing identities and composed arrows, and by identifying arrows when it is needed for the unitarity and associativity axioms to be satisfied.

Diagrammatic specifications and domains

Now, projective sketches and propagators can be used at the *meta-level* for specifying the diagrammatic specifications. Hence, let us consider a propagator:

$$P : \mathcal{E} \longrightarrow \overline{\mathcal{E}} \ .$$

Basically, the projective sketch $\mathcal{E}$ describes the form of the specifications, while $\overline{\mathcal{E}}$ and the propagator P describe the rules for deriving a theory from any given specification. For instance, a propagator P_{Eq} for equational logic is described in part 3.2.

Definition 3.10 A *(diagrammatic) P-specification* is a realization of $\mathcal{E}$, and a *(diagrammatic) P-domain* is a realization of $\overline{\mathcal{E}}$.

In addition, a *morphism of P-domains* is a morphism of realizations of $\overline{\mathcal{E}}$. But a *morphism of P-specifications* is somewhat more general than a morphism of realizations of $\mathcal{E}$; it is defined in the Kleisli way, as explained now.

For instance, with respect to the propagator $P_{Cat} : \mathcal{E}_{Gr} \longrightarrow \mathcal{E}_{Cat}$, a morphism of realizations of $\mathcal{E}_{Gr}$ is a graph morphism $\gamma : G_1 \longrightarrow G_2$, so that it maps each arrow of G_1 to an arrow of G_2. More generally, a morphism of P_{Cat}-specifications is allowed to map each arrow of G_1 to an arrow in the category $F(G_2)$: such an arrow can be composed from several arrows in G_2. Clearly, only a part of $F(G_2)$ is needed for describing γ; this part is a graph G_2' extending G_2 and such that $F(G_2')$ is the same as $F(G_2)$.

More generally, a *morphism of P-specifications* $\sigma : S_1 \longrightarrow S_2$ is defined as a morphism of realizations of $\mathcal{E}$ from S_1 to $F_P(S_2)$. In this way, the category of P-specifications is the Kleisli category of $\mathcal{R}eal(\mathcal{E})$ [Mac Lane 1971].

Not all of $F_P(S_2)$ is needed in order to describe σ. Actually, a morphism of P-specifications $\sigma : S_1 \longrightarrow S_2$ can be described from two morphisms of realizations of $\mathcal{E}$: a morphism $\tau : S_2 \longrightarrow S_2'$ such that $F(\tau)$ is an isomorphism, and a morphism $\sigma' : S_1 \longrightarrow S_2'$.

So, the category of P-specifications and the category of P-domains are such that:

$$\mathcal{S}pec(P) \supseteq \mathcal{R}eal(\mathcal{E}) \quad , \quad \mathcal{D}om(P) = \mathcal{R}eal(\overline{\mathcal{E}}) \ .$$

The adjunction is easily generalized:

$$\mathcal{S}pec(P) \underset{G_P}{\overset{F_P}{\rightleftarrows}} \mathcal{D}om(P)$$

Definition 3.11 The *P-theory generated by* a P-specification S is the P-domain $F_P(S)$.

Models of diagrammatic specifications

Definition 3.12 Let S denote a P-specification and $\mathcal{C}$ a P-domain. The *set of P-models of S with values in* $\mathcal{C}$ is:

$$\mathrm{Mod}_P(S, \mathcal{C}) = \mathrm{Hom}_{\mathcal{D}om(P)}(F_P(S), \mathcal{C}) \ .$$

The adjunction property yields the following bijection, which can be viewed as a kind of soundness property:

$$\mathrm{Mod}_P(S, \mathcal{C}) \cong \mathrm{Hom}_{\mathcal{S}pec(P)}(S, G_P(\mathcal{C})) \ .$$

For each morphism $\sigma : S \longrightarrow S'$, it is easy to define a map:

$$\mathrm{Mod}_P(\sigma, \mathcal{C}) : \mathrm{Mod}_P(S', \mathcal{C}) \longrightarrow \mathrm{Mod}_P(S, \mathcal{C}) \ ,$$

so that $\mathrm{Mod}_P(-, \mathcal{C})$ is a contravariant functor from the category of P-specifications to the category of sets.

In addition, it can happen, and it does happen in the most usual cases, that there is a natural notion of morphisms of P-models, so that there is a *category of P-models of S with values in $\mathcal{C}$*.

Example 3.13 Let $P = P_{Cat}$. A model M of a graph $\mathcal{G}$ with values in the category of sets interprets each point G of $\mathcal{G}$ as a set $M(G)$ and each arrow $g : G_1 \longrightarrow G_2$ of $\mathcal{G}$ as a map $M(g) : M(G_1) \longrightarrow M(G_2)$.

3.2 Equational specifications as diagrammatic specifications

A propagator for equational specifications

A propagator $P_{Eq} : \mathcal{E}_{Eq} \longrightarrow \overline{\mathcal{E}}_{Eq}$ is built now, in such a way that the P_{Eq}-specifications are the *equational specifications*, and the propagator P_{Eq} corresponds to the equational logic. The P_{Eq}-domains are called the *equational categories*.

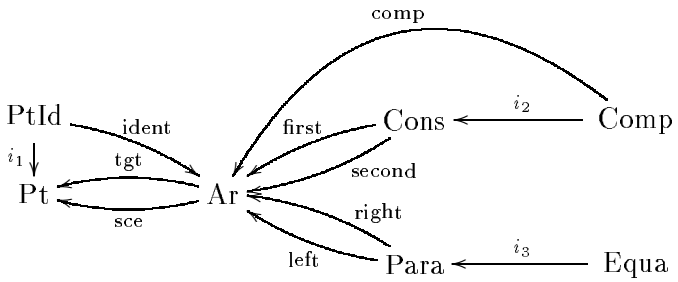
Actually, equational specifications are rather similar to projective sketches, and equational categories to categories.

Definition 3.14 An *equational specification* S can be defined as a compositive graph together with:
– some pairs of parallel arrows $(f : X \longrightarrow Y, g : X \longrightarrow Y)$ in S, called *equations* (or *potential equalities*), which are written $f \equiv g \ : X \longrightarrow Y$, or just $f \equiv g$,
– some cones $(p_i : P \longrightarrow X_i)_{i=1...n}$, called *(potential) products*; when $n = 0$, the vertex P of the cone is called a *(potential) terminal point*, which is written $P = \mathbb{1}$.

This definition of equational specifications is slightly different from the usual one, as reminded in part 2.1. Indeed, traditionally, an equational specification is built step by step. First some points, called the *sorts*, followed by some potential products of sorts, which vertices are represented as *lists of sorts*. Then some arrows, called the *operations*, followed by some potential composed operations, called *terms*, Finally the *equations* are pairs of parallel terms. In this new setting, a *point* may correspond to a sort or to a list of sorts, and an *arrow* to an operation or to a term.

It follows from the definition of equational specifications that they are the realizations of a

projective sketch $\mathcal{E}_{Eq}$, which contains the following graph:



So, the projective sketch $\mathcal{E}_{Eq}$ is an enrichment of $\mathcal{E}_{Gr}$. The points Cons and Comp stand for "pairs of consecutive arrows" and "pairs of composable arrows", Para and Equa for "pairs of parallel arrows" and "equations", PtId for "points with identity". The arrows first, second, left, right stand for the projections, i_1, i_2, i_3 for the inclusions, comp for the composition $(f, g) \mapsto g \circ f$, and ident for the selection of identities $A \mapsto \mathrm{id}_A$.

The projective sketch $\mathcal{E}_{Eq}$ also contains composed arrows like:

$$\begin{array}{ll} \mathrm{sce} \circ \mathrm{ident} = i_1 \ , & \mathrm{tgt} \circ \mathrm{ident} = i_1 \ , \\ \mathrm{sce} \circ \mathrm{comp} = \mathrm{sce} \circ \mathrm{first} \ , & \mathrm{tgt} \circ \mathrm{comp} = \mathrm{tgt} \circ \mathrm{second} \ , \end{array}$$

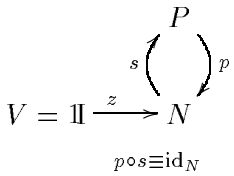
which mean that the profile of id_A is $A \longrightarrow A$, and that the profile of $g \circ f$ is $X \longrightarrow Z$ where X is the source of f and Z is the target of g.

Moreover, $\mathcal{E}_{Eq}$ contains a distinguished cone with vertex Cons, as in example 3.8, as well as a similar distinguished cone with vertex Para, and distinguished cones for monomorphisms, like:

$$\mathrm{PtId} \overset{i_1}{\rightarrowtail} \mathrm{Pt} \ , \qquad \mathrm{Comp} \overset{i_2}{\rightarrowtail} \mathrm{Cons} \ , \qquad \mathrm{Equa} \overset{i_3}{\rightarrowtail} \mathrm{Para} \ .$$

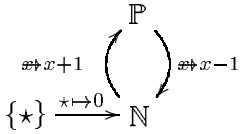
Example 3.15 Let us consider the equational specification T_5 from example 2.31. It is made of three points V, N and P, where V is a terminal point, five arrows $z : V \longrightarrow N$, $s : N \longrightarrow P$, $p : P \longrightarrow N$, $\mathrm{id}_N : N \longrightarrow N$ and $p \circ s : N \longrightarrow N$, where id_N is the identity of N and $p \circ s$ is composed of s and p, and one equation $p \circ s \equiv \mathrm{id}_N$. In the illustrations, the identities and composed arrows can be skipped, and the fact that V is a terminal point is represented as $V = \mathbb{1}$, so that the illustration is essentially the same as the usual illustration, as in example 2.31:

T_5 :

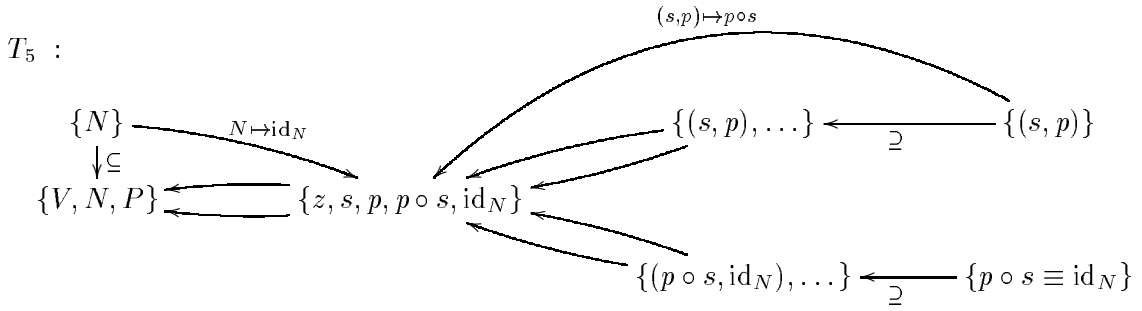


The specification T_5 has a model M_5 which interprets the sort V as a singleton $\{\star\}$, the sort N as the set $\mathbb{N}$ of non-negative integers, the sort P as the set $\mathbb{P}$ of positive integers, the constant z as the constant map $\star \mapsto 0$, and the operations s and p as the maps $x \mapsto x + 1$ and $x \mapsto x - 1$, respectively:

M_5 :



On the other hand, T_5 is a realization of $\mathcal{E}_{Eq}$, which gives rise to a totally different illustration of T_5:



Equational categories as diagrammatic domains

Definition 3.16 A *congruence relation* in a category $\mathcal{C}$ is a binary relation $\equiv$ on the arrows of $\mathcal{C}$ such that:
if $g_l \equiv g_r$ then g_l and g_r are parallel,
$\equiv$ is an equivalence relation,
$\equiv$ is compatible with the composition, which means that:
– if $g_l \equiv g_r : Y \longrightarrow Z$ and if $f : X \longrightarrow Y$, then $g_l \circ f \equiv g_r \circ f : X \longrightarrow Z$,
– if $g_l \equiv g_r : Y \longrightarrow Z$ and if $h : Z \longrightarrow U$, then $h \circ g_l \equiv h \circ g_r : Y \longrightarrow U$.

Clearly, the equality of arrows is a congruence relation.

Definition 3.17 An *equational category* is a category together with a congruence relation $\equiv$ which is compatible with the products.
A *morphism of equational categories* is a morphism of categories which preserves the congruence relation.

Any category $\mathcal{C}$ can be considered as an equational category $\mathcal{C}_{Eq}$, where the congruence relation is the equality of arrows. For instance, the category of sets *set* can be considered as an equational category set_{Eq}.

Then, it is easy to enrich $\mathcal{E}_{Eq}$ in order to get a projective sketch $\overline{\mathcal{E}}_{Eq}$ such that its realizations are the equational categories.

For instance, in order to prescribe the reflexivity of the congruence relation, the enrichment can proceed in two steps. First, a point Diag is added, together with a distinguished cone, so that Diag stands for the pairs (f, f) of identical arrows, and an arrow j : Diag $\longrightarrow$ Para is also added, which stands for the inclusion. Then, the reflexivity of the congruence relation corresponds to an arrow r : Diag $\longrightarrow$ Equa such that $i_3 \circ r = j$. Clearly, the reflexivity could be prescribed without adding the point Diag, since the interpretations of Diag and Ar are isomorphic. However, the point Diag will prove helpful later on.

Definition 3.18 The *equational propagator* is the enrichment:

$$P_{Eq} : \mathcal{E}_{Eq} \longrightarrow \overline{\mathcal{E}}_{Eq}$$

Hence, the P_{Eq}-specifications are the equational specifications and the P_{Eq}-domains are the equational categories.

Let $P = P_{Eq}$. Let S be a P-specification, i.e., an equational specification, and $F(S)$ its theory. Then the arrows of $F(S)$ are the *terms* of S, and the equations of $F(S)$ are the *congruences* of S, in the sense of part 2.1. On the one hand, in order to deal with computations issues, for instance with rewriting techniques, $F(S)$ has to be an equational category, not just a category: the equations are not equalities. On the other hand, the models of S, as defined in part 2.1, coincide with its P-models with values in set_{Eq}. Hence, the equations become equalities in these models. Actually, although the models of S with values in $\mathcal{C}$ are defined, as explained in part 3.1, for any equational category $\mathcal{C}$, most often $\mathcal{C} = \mathcal{C}'_{Eq}$ for some category $\mathcal{C}'$ (typically $\mathcal{C} = set_{Eq}$), i.e., the congruence relation of $\mathcal{C}$ is the equality of arrows.

Extensions of equational specifications by definition

It has been noted that any category $\mathcal{C}$ can be considered as an equational category $\mathcal{C}_{Eq}$, where the congruence relation is the equality of arrows. More precisely, $\mathcal{C}_{Eq} = G_{P'}(\mathcal{C})$, where $G_{P'}$ is the omitting functor associated to the propagator:

$$P' : \overline{\mathcal{E}}_{Eq} \longrightarrow \mathcal{E}_{Cat}$$

which maps the arrow $r : \mathrm{Diag} \longrightarrow \mathrm{Equa}$ of $\overline{\mathcal{E}}_{Eq}$ to the identity arrow $\mathrm{id}_{\mathrm{Diag}} : \mathrm{Diag} \longrightarrow \mathrm{Diag}$ of $\mathcal{E}_{Cat}$. Let $P'' = P' \circ P : \mathcal{E}_{Eq} \longrightarrow \mathcal{E}_{Cat}$. The following definition generalizes the one from part 2.1.

Definition 3.19 A morphism of equational specifications $\sigma : S_1 \longrightarrow S_2$ is an *extension by definition* if $F_{P''}(\sigma)$ is an isomorphism.

The definition asserts that an extension by definition does not modify the generated theory, up to congruence. The following result proves that an extension by definition does not modify the models, as long as their values are taken in a category.

Proposition 3.20 *Let $P = P_{Eq}$, and let $\sigma : S_1 \longrightarrow S_2$ be an extension by definition. Then, for every category $\mathcal{C}$, the map:*

$$\mathrm{Mod}_P(\sigma, \mathcal{C}_{Eq}) : \mathrm{Mod}_P(S_2, \mathcal{C}_{Eq}) \longrightarrow \mathrm{Mod}_P(S_1, \mathcal{C}_{Eq})$$

is a bijection.

Proof. Let S denote either S_1 or S_2. From the definition of models:

$$\mathrm{Mod}_P(S, \mathcal{C}_{Eq}) = \mathrm{Hom}_{\mathcal{D}om(P)}(F_P(S), \mathcal{C}_{Eq}) \ .$$

Since $\mathcal{C}_{Eq} = G_{P'}(\mathcal{C})$, from the adjunction property with respect to P':

$$\mathrm{Mod}_P(S, \mathcal{C}_{Eq}) \cong \mathrm{Hom}_{\mathcal{D}om(P'')}(F_{P''}(S), \mathcal{C}) \ .$$

The result follows. □

3.3 Overloaded specifications as diagrammatic specifications

Towards a propagator for overloaded specifications

Let us consider any propagator:

$$P : \mathcal{E} \longrightarrow \overline{\mathcal{E}} \ .$$

The aim of this part is to define a propagator:

$$L : \mathcal{L} \longrightarrow \overline{\mathcal{L}} \ ,$$

in such a way that, when P is the equational propagator, the L-specifications are the overloaded equational specifications and for each L-specification S, the L-theory $F_L(S)$ is the plain theory of the overloaded equational specification S, as defined in part 2.2.

A projective sketch for morphisms

Let us define, from the projective sketch $\mathcal{E}$, a new projective sketch $\mathcal{L}$, such that the realizations of $\mathcal{L}$ can be identified to the morphisms of realizations of $\mathcal{E}$ (actually, $\mathcal{L}$ is the *lax-colimit* of the identity morphism of $\mathcal{E}$, it is described for instance in [Duval et al. 2003]).

Definition 3.21 The projective sketch $\mathcal{L}$ is made of two copies $\mathcal{E}_1$ and $\mathcal{E}_0$ of $\mathcal{E}$, with an arrow $\mathrm{tr}_E : E_1 \longrightarrow E_0$ for each point E of $\mathcal{E}$, such that $e_0 \circ \mathrm{tr}_E = \mathrm{tr}_{E'} \circ e_1$ for each arrow $e : E \longrightarrow E'$ of $\mathcal{E}$:

$$\begin{array}{lccccc}
\mathcal{E}_1 : & \cdots\cdots & E_1 & \xrightarrow{e_1} & E'_1 & \cdots\cdots \\
 & \cdots\cdots & {\scriptstyle \mathrm{tr}_E}\downarrow & = & \downarrow{\scriptstyle \mathrm{tr}_{E'}} & \cdots\cdots \\
\mathcal{E}_0 : & \cdots\cdots & E_0 & \xrightarrow{e_0} & E'_0 & \cdots\cdots
\end{array}$$

The arrow $\mathrm{tr}_E : E_1 \longrightarrow E_0$ is the *transition arrow* of $\mathcal{L}$ with respect to E.

When $\mathcal{E} = \mathcal{E}_{Eq}$, the projective sketch $\mathcal{L} = \mathcal{L}_{Eq}$ contains:

$$\mathcal{E}_{Gr} : \quad \begin{array}{ccc}
 & \overset{\mathrm{sce}_1}{\underset{\mathrm{tgt}_1}{\rightleftarrows}} & \\
\mathrm{Pt}_1 & & \mathrm{Ar}_1 \\
{\scriptstyle \mathrm{tr}_{\mathrm{Pt}}}\downarrow & & \downarrow{\scriptstyle \mathrm{tr}_{\mathrm{Ar}}} \\
\mathrm{Pt}_0 & & \mathrm{Ar}_0 \\
 & \underset{\mathrm{tgt}_0}{\overset{\mathrm{sce}_0}{\rightleftarrows}} &
\end{array}$$

$$\mathrm{sce}_0 \circ \mathrm{tr}_{\mathrm{Ar}} = \mathrm{tr}_{\mathrm{Pt}} \circ \mathrm{sce}_1$$
$$\mathrm{tgt}_0 \circ \mathrm{tr}_{\mathrm{Ar}} = \mathrm{tr}_{\mathrm{Pt}} \circ \mathrm{tgt}_1$$

The next result is easy to check.

Proposition 3.22 *For each realization Z of $\mathcal{L}$, let T, S and $\zeta : T \longrightarrow S$ be defined by the following equalities, for each point E of $\mathcal{E}$:*

$$T(E) = Z(E_1) \ , \quad S(E) = Z(E_0) \ , \quad \zeta_E = Z(\mathrm{tr}_E) : T(E) \longrightarrow S(E) \ .$$

This defines a one-to-one correspondence between the realizations of $\mathcal{L}$ and the morphisms of P-specifications.

A propagator for morphisms

Similarly, the projective sketch $\overline{\mathcal{L}}$ is defined from $\overline{\mathcal{E}}$, and the propagator L is defined as the canonical propagator from $\mathcal{L}$ to $\overline{\mathcal{L}}$:

$$L : \mathcal{L} \longrightarrow \overline{\mathcal{L}} \ .$$

Overloaded P-specifications and overloaded P-domains

Definition 3.23 An *overloaded P-specification* is a L-specification, and an *overloaded P-domain* is a L-domain.

Hence, from proposition 3.22, an overloaded P-specification is a morphism of P-specifications, and an overloaded P-domain is a morphism of P-domains. So, the definition of overloaded P-specifications generalizes the definition of overloaded equational specification from part 2.2.

The plain semantics of an overloaded P-specification

Definition 2.5 is generalized as follows.

Definition 3.24 The *plain theory* of an overloaded P-specification ζ is the overloaded P-domain $F_L(\zeta)$.

Like any L-domain, $F_L(\zeta)$ is a morphism of P-domains. It is easy to check the following *reliability property*:

Proposition 3.25 *The L-domain $F_L(\zeta)$ is such that:*

$$F_L(\zeta) : F_P(T) \longrightarrow F_P(S) \ .$$

The plain models of an overloaded P-specification

When P is the equational propagator, from definition 2.4, the plain models of ζ are the P-models of T, i.e., the morphisms from $F_P(T)$. This suggests the following definition:

Definition 3.26 Let ζ be an overloaded P-specification. For any P-domain $\mathcal{C}$, the *plain P-models of ζ with values in $\mathcal{C}$* are the morphisms from $F_P(T)$ to $\mathcal{C}$:

$$\mathrm{Mod}_P(\zeta, \mathcal{C}) = \mathrm{Hom}_{\mathcal{D}om(P)}(F_P(T), \mathcal{C}) = \mathrm{Mod}_P(T, \mathcal{C}) \ .$$

Hence, the difference between an overloaded P-specification and a L-specification lies in the associated notion of model. Indeed, a L-specification has L-models in a L-domain, according to part 3.1. When it is considered as an overloaded P-specification, it has P-models in a P-domain; this definition of models stems from the notion of zoom [Duval et al. 2003].

This paragraph can be skipped. A zoom, as defined in [Duval et al. 2003], is associated to three specifications: a *far* specification S, an *intermediate* specification T, and a *near* specification U; starting from S, a *decoration* process builds T, then an *expansion* process builds U. For dealing with overloading, the expansion process is trivial, so that $T = U$, and the decoration process is simply the morphism $\zeta : T \longrightarrow S$. The unique non-trivial feature of zooms which is used here is the definition of models, and the related notion of reliability.

3.4 The name-driven semantics, diagrammatically

The identification rule

The propagator L from part 3.3 does not take into account the identification rule: two homonymous parallel terms do not have to be congruent. It is now enriched, in order to take this rule into account. Part 2.3 suggests that the new propagator:

$$L^{\equiv} : \mathcal{L}^{\equiv} \longrightarrow \overline{\mathcal{L}^{\equiv}}$$

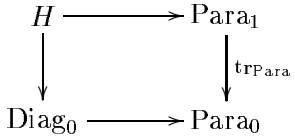
should be such that $\mathcal{L}^{\equiv} = \mathcal{L}$ and the $L^{\equiv}$-domains are the L-domains which satisfy the identification rule.

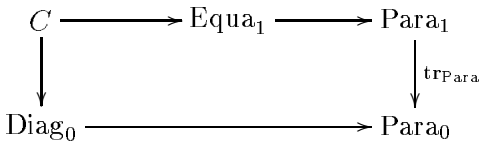
From definition 2.11, the *identification rule* is:

If two parallel terms t' and t'' have the same name,
then $t' \equiv t''$.

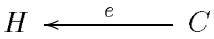
The construction of $\overline{\mathcal{L}^{\equiv}}$ from $\overline{\mathcal{L}}$ can be made in two steps.

First, as for the point Cons in part 3.2, a point can be added to $\overline{\mathcal{L}}$, without changing its realizations, if it is the vertex of a distinguished cone with its base in $\overline{\mathcal{L}}$ and with new projections. In this way, let us now add two points H and C to $\overline{\mathcal{L}}$, so that $\zeta(H)$ is the set of pairs of parallel homonymous terms (t', t'') (the hypothesis of the rule) and $\zeta(C)$ is the set of pairs of equations $t' \equiv t''$ made of parallel homonymous terms (the conclusion of the rule).



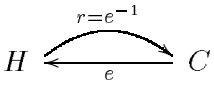


Then, an arrow $e : C \longrightarrow H$ can also be added to $\overline{\mathcal{L}}$, together with the obvious equations, so that $\zeta(e)$ is the inclusion of $\zeta(C)$ in $\zeta(H)$:



Until now, the resulting projective sketch has the same realizations as $\overline{\mathcal{L}}$.

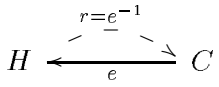
Then, a new arrow $r : H \longrightarrow C$ is added, which is the inverse of e, i.e., $e \circ r = \mathrm{id}_H$ and $r \circ e = \mathrm{id}_C$, or simply $r = e^{-1}$:



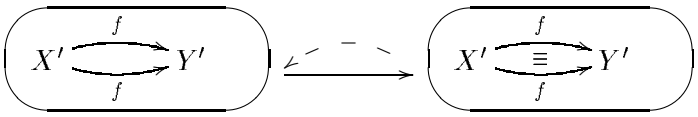
Then, the realizations of $\overline{\mathcal{L}^{\equiv}}$ are the realizations ζ of $\overline{\mathcal{L}}$ such that $\zeta(C) = \zeta(H)$: this means that they are the realizations of $\overline{\mathcal{L}}$ which satisfy the identification rule, as required.

An illustration of the identification rule

In order to illustrate this construction, the propagator $L^{\equiv} : \mathcal{L}^{\equiv} \longrightarrow \overline{\mathcal{L}^{\equiv}}$ can be represented as a copy of $\mathcal{L}$ together with H, C, e, which do not modify the realizations, and with r as a dashed arrow. So that $L^{\equiv}$ contains:



As explained in [Duval et al. 2003], the contravariant Yoneda morphism maps this piece of $L^{\equiv}$ to a morphism of $L^{\equiv}$-specifications, which is the one from part 2.3:



The name-driven models of an overloaded P-specification

The new propagator $L^{\equiv} : \mathcal{L}^{\equiv} \longrightarrow \overline{\mathcal{L}^{\equiv}}$ does take into account the identification rule, but it does not satisfy the reliability property: this means that, if $\zeta : T \longrightarrow S$ is any overloaded P-specification, the source of $F_{L^{\equiv}}(\zeta)$ is different from $F_P(T)$. More precisely, $F_{L^{\equiv}}(\zeta)$ is a morphism of P-domains $F_{L^{\equiv}}(\zeta) : T^{\equiv} \longrightarrow F_P(S)$, but its source $T^{\equiv}$ is not isomorphic, in general, to $F_P(T)$. Actually, the P-domain $T^{\equiv}$ does not depend only on T, but on the whole ζ. This is easily seen from the diamond example, as described in part 2.5: the equation $u' \circ t' \equiv u'' \circ t''$ is in $T^{\equiv}$, although it is not in $F_P(T)$.

Definition 3.27 Let ζ be an overloaded P-specification. For any P-domain $\mathcal{C}$, the *name-driven P-models of ζ with values in $\mathcal{C}$* are the morphisms from $T^{\equiv}$ to $\mathcal{C}$:

$$\mathrm{Mod}^{\equiv}_P(\zeta, \mathcal{C}) = \mathrm{Hom}_{\mathcal{D}om(P)}(T^{\equiv}, \mathcal{C}) \ .$$

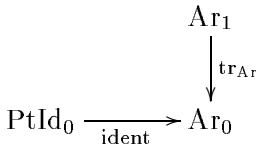
It follows, because $L^{\equiv}$ does not satisfy the reliability property, that the name-driven P-models of ζ with values in $\mathcal{C}$ are different from the P-models of T with values in $\mathcal{C}$. Actually, there is a canonical map from $\mathrm{Mod}^{\equiv}_P(\zeta, \mathcal{C})$ to $\mathrm{Mod}_P(T, \mathcal{C})$, which corresponds to the fact that the name-driven P-models of ζ are the models of T which satisfy the identification property.

3.5 Subtypes and subsorts

About the propagators for dealing with subtypes and subsorts

In this part, it is assumed that the propagator $P : \mathcal{E} \longrightarrow \overline{\mathcal{E}}$ is such that there is some reasonable notion of arrows and identity arrows; typically, there is in $\mathcal{E}$ a copy of a sketch of compositive graphs, with a point PtId and an arrow ident : PtId $\longrightarrow$ Ar, and there is in $\overline{\mathcal{E}}$ a copy of a sketch of categories, so that the unitarity properties of identity arrows are prescribed. For instance, P_{Eq} satisfies this assumption.

Then, from its definition, the projective sketch $\mathcal{L}$ contains:



Each of the semantics from section 2 corresponds to a propagator L', which is obtained by some enrichment of the propagator L. More precisely, the correspondence is such that:
– the specifications are the L'-specifications,
– the theory of a specification ζ is its freely generated L'-domain $F_{L'}(\zeta)$,
– and the models of a specification ζ with values in a P-domain $\mathcal{C}$, are the morphisms from T' to $\mathcal{C}$, where T' is the source of $F_{L'}(\zeta)$, when $F_{L'}(\zeta)$ is considered as a morphism of P-domains.

This correspondence holds for the plain and the name-driven semantics of overloaded specifications, as explained in parts 3.3 and 3.4. It is now checked that it does hold for the plain and the name-driven semantics of subtyping, as well as the semantics for subsorting. These propagators do not satisfy the reliability condition.

Subtypes

Let us add to $\mathcal{L}$ a point Coer, together with three arrows from Coer to Ar_1, Ar_0 and PtId_0, respectively, and with the distinguished cone with vertex Coer:

$$\begin{array}{ccc} \mathrm{Coer} & \longrightarrow & \mathrm{Ar}_1 \\ \downarrow & & \downarrow{\scriptstyle \mathrm{tr}_{\mathrm{Ar}}} \\ \mathrm{PtId}_0 & \xrightarrow[\mathrm{ident}]{} & \mathrm{Ar}_0 \end{array}$$

This projective sketch is called $\mathcal{L}_{st}$, its realizations are still the overloaded P-specifications: indeed, if ζ is a P-specification, then $\zeta(\mathrm{Coer})$ is the set of operations of ζ such that their name is an identity arrow. Hence, the following definition generalizes definition 2.19:

Definition 3.28 In an overloaded P-specification ζ, a *coercion* is an element of $\zeta(\mathrm{Coer})$. A point Y' is a *subtype* of a point Y'' if there is a coercion from Y' to Y''.

The semantics of subtyping

Then, the projective sketch $\overline{\mathcal{L}_{st}}$ is built from $\overline{\mathcal{L}}$ by adding the subtyping rules, in a way similar to the construction of $\overline{\mathcal{L}^{\equiv}}$ in part 3.4. This yields a new propagator:

$$L_{st} : \mathcal{L}_{st} \longrightarrow \overline{\mathcal{L}_{st}} \ ,$$

which corresponds to the plain semantics of subtyping, from part 2.4.

The projective sketch $\overline{\mathcal{L}_{st}^{\equiv}}$ is built from $\overline{\mathcal{L}_{st}}$ by adding the coherence rules, so that the new propagator:

$$L_{st}^{\equiv} : \mathcal{L}_{st} \longrightarrow \overline{\mathcal{L}_{st}^{\equiv}}$$

corresponds to the name-driven semantics of subtyping, from part 2.5.

Subsorts

Let us add to $\mathcal{L}_{st}$ a point Incl, together with a monomorphism $m :$ Incl $\rightarrowtail$ Coer . This projective sketch is called $\mathcal{L}_{ss}$. The following definition generalizes definition 2.35:

Definition 3.29 A *P-specifications with subsorts* is a realization of $\mathcal{L}_{ss}$.

The semantics of subsorting

Finally, the projective sketch $\overline{\mathcal{L}^{\equiv}_{ss}}$ is built from $\overline{\mathcal{L}^{\equiv}_{st}}$ by adding the retraction rule. This new propagator:

$$L^{\equiv}_{ss} : \mathcal{L}_{ss} \longrightarrow \overline{\mathcal{L}^{\equiv}_{ss}}$$

corresponds to the semantics of subsorting, from part 2.6.

3.6 An example over non-equational specifications

Functional specifications

Section 2 is restricted to equational specifications, while section 3 deals with more general specifications. In this last part, a non-equational application is considered. It deals with subtyping when operations and terms are considered as first-order objects: this is the context of simply-typed lambda-calculus.

The *functional rule* states that, for all points X and Y, there is a point Y^X, called a *power sort*, which stands for the set of functions from X to Y, usually called a *power set*. The power sort Y^X is such that, for any point Z, the terms from $Z \times X$ to Y can be identified with the terms from Z to Y^X. More precisely, for all points Y and X, there is an *application* operation:

$$\mathrm{app}_{Y,X} : Y^X \times X \longrightarrow Y \ ,$$

and for all point Z, each term $t : Z \times X \longrightarrow Y$ corresponds bijectively to a term $\widehat{t} : Z \longrightarrow Y^X$ such that:

$$\mathrm{app}_{Y,X}(\widehat{t}(z), x) = t(z, x) \ .$$

The *functional propagator* P_{Fu} is obtained by adding the functional rule to the equational propagator.

Coercions between power sorts

The usual *functional subtyping rule* is:

If there are coercions $X' \longrightarrow X''$ and $Y' \longrightarrow Y''$,
then there is a coercion $(Y')^{(X'')} \longrightarrow (Y'')^{(X')}$.

Now, it is proven that this rule derives from our definition of coercions.

Theorem 3.30 *In an overloaded P_{Fu}-domain, the functional subtyping property is satisfied.*

Proof. Let $\varepsilon_X : X' \longrightarrow X''$ and $\varepsilon_Y : Y' \longrightarrow Y''$ be two coercions. Let us consider the following composed arrow (where id denotes the identity of $(Y')^{(X'')}$):

$$f : (Y')^{(X'')} \times X' \longrightarrow Y'' \ :$$

$$\begin{array}{ccc} (Y')^{(X'')} \times X' & \xrightarrow{f} & Y'' \\ {\scriptstyle \mathrm{id}\times\varepsilon_X}\downarrow & = & \uparrow{\scriptstyle \varepsilon_Y} \\ (Y')^{(X'')} \times X'' & \xrightarrow[\mathrm{app}_{Y',X''}]{} & Y' \end{array}$$

It corresponds to an arrow:

$$\widehat{f} : (Y')^{(X'')} \longrightarrow (Y'')^{(X')} .$$

Let us now check that $\widehat{f}$ is a coercion. The image of f in S is the following composed arrow (where id denotes the identity of Y^X):

$$\begin{array}{ccc} Y^X \times X & \xrightarrow{\zeta(f)} & Y \\ {\scriptstyle \mathrm{id}\times\mathrm{id}_X}\downarrow & = & \uparrow{\scriptstyle \mathrm{id}_Y} \\ Y^X \times X & \xrightarrow[\mathrm{app}_{Y,X}]{} & Y \end{array}$$

Hence, the name of f is:

$$\mathrm{app}_{Y,X} : Y^X \times X \longrightarrow Y .$$

On the other hand, it is easily checked that:

$$\widehat{\mathrm{app}_{Y,X}} = \mathrm{id}_{Y^X} .$$

It follows that $\widehat{\zeta(f)} = \mathrm{id}_{Y^X}$. Since $\widehat{\zeta(f)} = \zeta(\widehat{f})$, it follows that the name of $\widehat{f}$ is an identity arrow of S, i.e., $\widehat{f}$ is a coercion, as required. □

About other propagators

The equational propagator P_{Eq} generates new points $X \times Y$ ("record types") from points X and Y. The functional propagator P_{Fu}, moreover, generates new points Y^X ("power types") from points X and Y. Other propagators may generate pull-backs, for instance; then points (the vertices of the pull-backs) are generated from points *and arrows* (the base of the pull-backs). In such a case, the congruence on arrows generates a congruence on points. Such situations are considered in [Cury 1991].

4 Conclusion

In this paper, overloaded specifications are defined as morphisms of ordinary specifications. Thanks to this new point of view on overloading, we get a clear description of the disambiguation issue and of the name-driven semantics for solving it. The interaction of overloading with subtyping and subsorting also gets clearer. From a more operational point of view, it

would be worthwhile to interpret papers like [Hintermeier et al. 1998], [Bouhoula et al. 2000] and [Casl 2003] in this framework.

The more abstract level offered by the diagrammatic specifications allows to consider that overloaded specifications and ordinary specifications are of the same nature. Moreover, diagrammatic specifications can also deal with other major issues, like for instance the treatment of exceptions [Duval et al. 2003].

References

[Bouhoula et al. 2000] A. Bouhoula, J.-P. Jouannaud and J. Meseguer. Specification and Proof in Membership Equational Logic. *Theoretical Computer Science* **236**, 35-132 (2000).

[Bruce 2002] K.B. Bruce. *Foundations of Object-Oriented Languages, Types and Semantics.* MIT (2002).

[Casl 2003] M. Bidoit and P.D. Mosses. *The CASL Book.* http://www.cofi.info/ (2003).

[Castagna 1995] G. Castagna. Covariance and covariance: conflict without a cause. *ACM Transactions on Programming Languages and Systems* **17** 431-447 (1995).

[Coppey and Lair 1984] L. Coppey and C. Lair. Leçons de Théorie des esquisses (I). *Diagrammes* **12** (1984).

[Coppey and Lair 1988] L. Coppey and C. Lair. Leçons de Théorie des esquisses (II). *Diagrammes* **19** (1988).

[Cury 1991] F. Cury. Catégories lax-localement cartésiennes et catégories localement cartésiennes : un exemple de suffisante complétude connexe de sémantiques initiales. *Diagrammes* **25** (1991).

[Duval and Lair 2002] D. Duval. Diagrammatic specifications. To appear in *Mathematical Structures in Computer Science.* A preliminary version is available as: D. Duval and C. Lair. Diagrammatic specifications. *Rapport de recherche IMAG-LMC 1043.* http://www-lmc.imag.fr/lmc-cf/Dominique.Duval/ (2002).

[Duval et al. 2003] D. Duval, C. Lair, J.-C. Reynaud, C. Oriat. A zooming process for specifications. *Rapport de recherche IMAG-LMC 1055.* http://www-lmc.imag.fr/lmc-cf/Dominique.Duval/ (2003).

[Ehresmann 1966] C. Ehresmann. Introduction to the theory of structured categories. Report **10**, University of Kansas, Lawrence (1966).

[Goguen et al. 1978] J.A. Goguen, J.W. Thatcher and E.G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In *Current trends in Programming Methodology*, ed. R.T. Yeh, **4**, Prentice-Hall (1978).

[Hintermeier et al. 1998] C. Hintermeier, C. Kirchner, and H. Kirchner. Dynamically-Typed Computations for Order-Sorted Equational Presentations. *Journal of Symbolic Computation* **25**, 455–526 (1998).

[Lair and Duval 2001] Lair, C. and Duval, D. (2001) Esquisses et Spécifications, Manuel de Référence, 4ème partie : Fibrations et Eclatements, Lemmes de Yoneda et Modèles Engendrés. *Rapport de Recherche du LACO* 2001-**03**.
http://www.unilim.fr/laco/rapports/ (2003).

[Mac Lane 1971] Mac Lane, S. (1971) *Categories for the working mathematician.* Springer-Verlag.

[Mosses 1992] P.D. Mosses. The Use of Sorts in Algebraic Specifications. In *Recent Trends in Data Type Specification*, ed. M. Bidoit and C. Choppy, LNCS 655, Springer-Verlag (1992).