PUC

ISSN 0103-9741

Monografias em Ciência da Computação
nº 33/94

# Logical Specifications: 3. Extensions of Specifications

Paulo A. S. Veloso
Thomas S. E. Maibaum

Departamento de Informática

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

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, Nº 33/94

Editor: Carlos J. P. Lucena Novembro, 1994

# Logical Specifications: 3. Extensions of Specifications *

Paulo A. S. Veloso

Thomas S. E. Maibaum**

* Research partly sponsored by the Brazilain agencies CNPq, RHAE and FAPERJ, British agency SERC and European Community agencies.

** Dept. of Computing, Imperial College of Science, Technology and Medicine, London, UK.

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

# LOGICAL SPECIFICATIONS: 3. EXTENSIONS OF SPECIFICATIONS

**Paulo A. S. VELOSO** and **Thomas S. E. MAIBAUM**
{e-mail: veloso@inf.puc-rio.br and tsem@doc.ic.ac.uk}

PUCRioInf MCC 33 /94

**Abstract.** We present basic concepts and results concerning extensions of logical specifications and illustrate their use in constructing specifications. We start by briefly examining the idea of hidden symbols in specifications. Then, we introduce some concepts concerning extensions of languages and of presentations, as well as properties such as conservativeness and eliminability. We then examine separately extensions by the introduction of predicates, operations, constants and sorts, with emphasis on criteria for conservativeness. In addition to classical definitions, we also consider. and compare, constraints (inner, outer, Skolemisation, and pre and post conditions), as well as inductive predicates. The methods we consider for introducing new sorts are product, discriminated union, subsort, quotient and enumeration, as found in many programming languages. Finally, we apply these ideas to the treatment of errors and to construction of specifications. This report is a draft of the third section of a handbook chapter. Other reports cover the remaining sections.

**Key words:** Formal specifications, program development, axiomatic specifications, extensions of specifications, conservative extensions, eliminability, introduction of predicate, operations, constants and sorts, definitions, constraints, inductive predicates, Skolemisation, pre and post conditions, product, discriminated union, subsort, quotient, enumeration, treatment of errors, construction of specifications.

**Resumo.** Apresentamos conceitos e resultados básicos sobre extensões de especificações lógicas ilustrando seu emprego em construção de especificações. Começamos com um breve exame da idéia de símbolos ocultos em especificações. A seguir, introduzimos alguns conceitos relativos a extensões de linguagens e de apresentações, além de propriedades como conservatividade e eliminabilidade. Passamos então a examinar separadamente extensões por introdução de predicados, operações, constantes e sortes, enfatizando critérios para conservatividade. Além de definições clássicas, consideramos e comparamos restrições (internas, externas, skolemização, e pré e pós condições), bem como predicados indutivos. Os métodos considerados para a introduzir novos sortes são produto, união discriminada, subsorte, quociente e enumeração, como encontrados em várias linguagens de programação. Finalmente, aplicamos essas idéias ao tratamento de erros e a construção de especificações. Este relatório é um esboço da terceira seção de um capítulo de um manual. Outros relatórios cobrem as demais seções.

**Palavras chave:** Especificações formais, desenvolvimento de programas, especificações axiomáticas, extensões de especificações, extensões conservativas, eliminabilidade, introdução de predicados, operações, constantes e sortes, definições, restrições, predicados indutivos, skolemização, pré e pós condições, produto, união discriminada, subsorte, quociente, enumeração, tratamento de erros, construção de especificações.

## NOTE

This report is a draft of the third section of a chapter in a forthcoming volume of the Handbook of Logic in Computer Science

Other reports, corresponding to the remaining sections, have been issued or are in preparation. The plan of the chapter - and series of reports Logical Specifications - is as follows.

1. Introduction and Overview — MCC 13/94, June 1994, (v+11 p)
2. Specifications as Presentations — MCC 26/94, July 1994, (vi+24 )
3. Extensions of Specifications
4. Interpretation of Specifications
5. Implementation of Specifications
6. Parameterised Specifications
7. Conclusion: Retrospect and Prospects.

The chapter - and series of reports - is intended to provide an account of the logical approach to formal specification development.

Any comments or criticisms will be greatly appreciated.

The next report in this series is planned to be Logical Specifications: 4. Interpretations of Specifications, covering:

unsorted and many-sorted translations and interpretations;
composition, decomposition and internalisations of translations;
variations of translations and interpretations, relativisation, translation of equality, sort tupling;
interpolation and modularity of extensions and of interpretations.

## ACKNOWLEDGEMENTS

Research reported herein is part of an on-going research project. Partial financial support from British, European Community and Brazilian agencies is gratefully acknowledged. The hospitality and support of the institutions involved have been very helpful. Collaboration with Martin R. Sadler, Sheila R. M. Veloso and José L. Fiadeiro was instrumental in sharpening many ideas. The authors would like to thank the following for many fruitful discussions on these and related topics: Carlos J. P. de Lucena, Samit Khosla, Atendolfo Pereda Bórquez, Douglas R. Smith, Haydée W. Poubel and M. Claudia Meré. Special thanks go to Tarcísio H. C. Pequeno and Roberto Lins de Carvalho

## CONTENTS *

* See the preceding note for an explanation of the numbering system

## List of Figures

## List of Example Specifications

## 3 Extensions of Specifications

Specifications as theory presentations are not given whole, they have to be constructed to suit the purpose at hand. One might claim that the subject matter of specification engineering is the study and practice of presentation construction - and, as we shall see later, a variant of it, translation. (Even management of software projects can be seen in terms of theory construction: how can one control the construction of very large theories.)

Important tools for the construction of specifications are related to extensions: one can construct a, possibly large, specification by starting from a small one and extending it in steps. In this process of presentation construction, one generally wishes to preserve - as oposed to revise - certain properties. This is the basic idea underlying modularity.

The structure of this section is as follows. We start in 3.1 (Introduction) by presenting some simple examples, indicating what extensions are and some of their uses. Then we briefly illustrate the use of extensions in constructing specifications and comment on the role of hidden symbols in 3.2 and examine some concepts related to extensions of languages and specifications in 3.3. In 3.4 we examine the concepts of conservativeness and eliminability. The next four subsections consider separately the addition of predicates, operations, constants and sorts. In 3.5 we examine extensions by addition of predicate symbols: by definitions, by outer and inner constraints and by induction. Extensions by operations are considered in 3.6, which examines, and later compares, definitions, constraints (outer, inner, Skolemisation, and pre and post conditions). In 3.7 we briefly examine extensions by constants, with emphasis on the simultaneous introduction of several constants by Skolemisation. Extensions by sorts, which are important but more complex, are dealt with in 3.8. We examine constructs similar to those found in many programming languages: four constructs that introduce a new sort (together with some conversion operations) as a product, a subsort, a quotient, or a discriminated union of existing sorts, as well as the introduction of a new sort by simply enumerating its elements. Some applications We also comment on these constructs and their use in program development. We then examine some applications illustrating the use of extensions in development of specifications. The first one, more general, will address the issues related to errors. The second one will consist of case studies showing the construction of extensions of integers to naturals and rationals, which illustrate the use of extensions for building specifications, hinting at some desirable features of systems providing support for such task. Our Example

Specifications of this section are collected in 3.10.

## 3.1 Introduction

A specification 'describes' (properties of) some objects expressed by sentences of a specific language. In dealing with specifications it is often useful to compare a specification with parts of it in smaller languages. Also, a useful approach to constructing specifications is to start from somewhat simple specifications which will be extended.

An extension is obtained by adding new symbols (with their syntactical declarations) and new properties (expressed by new axioms). A simple example will indicate what extensions consist of as well as how they can be used in constructing larger specifications.

Assume that we wish to specify the Bolean values with negation and ordering. We may start by specifying the Bolean values, and then extend it by adding negation and less (symbol for them and their properties).

So, we may start form the specification BOOL for the Bolean values given as Spec. 3.1 in 3.10. We now extend it by adding operation negation and predicate less?, obtaining Spec. 3.2: BOOL_EXT_NEG&LESS in 3.10.

Extensions also provide a way of comparing presentations. For a simple example, assume that we have two presentations of the ordering of the natural numbers: one based on lt (less than) and another one based on le (less than or equal). As such, they are quite different, even linguistically. But, we can extend them to a common language, where they can be compared. We can extend the former by introducing le by means of the (defining) axiom $\forall x,y[le(x,y)\leftrightarrow(lt(x,y)\vee x\approx y)]$. Similarly, we can extend the latter by means of $\forall x,y[lt(x,y)\leftrightarrow(le(x,y)\wedge\neg x\approx y)]$, which defines lt in terms of le. These extensions to the common language turn out to be equivalent, as one might expect. (Here, since we have a single sort, we have left it implicit in equality and quantitifications, a simplification we shall often resort to.)

## 3.2 Specification construction and hidden symbols

We shall now briefly illustrate the use of extensions in constructing specifications and comment on the role of hidden symbols.

A simple example illustrating the idea of specification construction may be obtained by examining how to formalise the notion of natural numbers with a "less than" relation defined on them. Here, the language consists of the sort Nat, and the symbols succ (unary operation), zero (constant) and $<$ (infix binary predicate).

One may arrive at an entire specification somehow; for instance, by discovering in Enderton (1972, Section 3.2, p.184) the following axiomatisation $A_L$:

$\forall y(\neg y \approx zero \rightarrow \exists x\ y \approx succ(x))$ (S3)

$\forall x,y(x<succ(y) \leftrightarrow x<y \vee x \approx y)$ (L1)

$\forall x \neg x<zero$ (L2)

$\forall x,y(x<y \vee x \approx y \vee y<x)$ (L3)

$\forall x,y(x<y \rightarrow \neg y<x)$ (L4)

$\forall x,y,z[x<y \rightarrow (y<z \rightarrow x<z)]$ (L5)

If, on the other hand, we had to construct such a presentation from scratch, it might be wise to start from a simpler one, say that of the ordering with only <, and later extend it by introducing the remaining symbols.

Thus, we may start by describing < as a strict ordering that is linear, discrete, has a beginning but no end. In other words, we start from the following axiomatisation $G_<$ (involving only <):

$\forall x \neg x<x$ (ir)

$\forall x,y,z[(x<y \wedge y<z) \rightarrow x<z]$ (tr)

$\forall x,y(x<y \vee x \approx y \vee y<x)$ (ln)

$\forall x \exists y x<y$ (nl)

$\neg \forall x \exists y y<x$ (ft)

$\forall x,y\{x<y \rightarrow \exists u[x<u \wedge \neg \exists z(x<z \wedge z<u)]\}$ (dc+)

$\forall x,y\{x<y \rightarrow \exists v[v<y \wedge \neg \exists z(v<z \wedge z<y)]\}$ (dc-)

We now extend the above presentation $G_<$ by definitions for zero and succ:

$\forall y(y \approx zero \leftrightarrow \neg \exists x\ x<y)$ (Dz)

$\forall x,y[y \approx succ(x) \leftrightarrow x<y \neg \exists z\ (x<z \wedge z<y)]$ (Ds)

thereby obtaning presentation $G_L$.

Note that (Dz) is consistent with (ft) because of the existence and uniqueness of those values which are minimal with respect to <, and (Ds) is consistent with (dc+).

Now $G_L$ was obtained in steps, and is more trustworthy than $A_L$, if we had written the latter in a single swoop. In fact, they are equivalent since either one generates Th(<**N**,S,0,<>). Which specification is preferred depends very much on factors to do with intended use, starting point for formalisation, even history of modifications during formalisation (Maibaum *et al.* 1991).

In this example we have so far assumed that we wished to specify the naturals with zero and successor, as wel as their ordering. Assume now that we actually wished to specify the naturals with zero and successor. Notice that Th(<**N**,S,0>) is not finitely axiomatisable (Enderton 1972, p. 178-187), in contrast with Th(<**N**,S,0,<>). Thus, it might be interesting to retain the previous axiomatisation $G_L$ (or $A_L$) for Th(<**N**,S,0,<>) and add the proviso that < is not really part of the language. This is part of the idea of a hidden (predicate) symbol.

Hidden symbols are often found in the literature (Guttag and Horning 1978). The idea is that some symbols of a specification are flagged as *hidden*. This serves two distinct, but related, purposes

To the user of the specification, it signals that such a symbol may be used in reasoning, but should not be invoked by a program.

To the implementor of the specification, it signals that he needs not to bother with implementing such a symbol by procedures.

In short, a hidden symbols is just like an ordinary symbol from the viewpoint of declarative reasoning, but it is not guaranteed to have a procedural realisation. The motivaton is simplifying specifications without burdening the implementation. Like hidden predicates, illustrated in the above example, hidden operations and hidden sorts are often found useful.

## 3.3 Extensions of languages and presentations

Extending specifications involves extending the languages and the sets of axioms. We shall now examine some concepts related to extensions of languages and specifications.

Given languages L' and L", we say that L' is a *sub-language* of L", or that L" *extends* L', (notation L'⊆L") iff each extra-logical symbol (sort, operation, or predicate) of L' is a symbol of the same kind in L" with the same declaration, which also is to occur with the variables and their declarations as well. In other words, L" can be thought of as obtained from L' by the addition of new symbols and their syntactical declarations; so each formula of L' is a formula of L", bu the latter may have new formulae not in L'. Thus, if L'⊆L" then Srt(L')⊆Srt(L"), Prd(L')⊆Prd(L"), Opr(L')⊆Opr(L"), and Var(L')⊆Var(L"). Also, Trm(L')⊆Trm(L") and Frml(L')⊆Frml(L").

For theories T' and T", we say that T' is a *sub-theory* of T", or that T" *extends* T' iff T'⊆T" as sets of sentences. For specifications P'=<L',G'> and P"=<L",G">, we say that P" is an *extension* of P', which we denote by P'⊆P", iff L'⊆L" and Cn[P']⊆Cn[P"], i. e., we have inclusion of languages and theories generated.

The important point to note is that extension amounts to inclusion between theories and not between axiomatic presentations of theories, even though it is defined as a relationship between specifications. When constructing an extension one generally adds new axioms to the existing ones, so that one also gets inclusion between axiomatisations.

The relation of extension between specifications is clearly reflexive and transitive. Transitivity is of special interest in view of our idea of using extensions to construct specifications in steps.

Given a theory T" over language L" and a sub-language $L' \subseteq L''$, the *restriction* of T" to sub-language L' is the theory $T''\surd L' := \{\sigma \in Sent(L') \backslash T'' \models \sigma\}$ consisting of the sentences of L' that are in T" (Shoenfield 1967, p. 95). Similarly, the *restriction* of specification $P''=<L'',G''>$ to sub-language $L' \subseteq L''$ is the specification $P''\surd L' := <L',Cn[P'']\surd L'>$. Notice that restriction does not necessarily preserves finite axiomatisability. As mentioned in 3.2, $Th(<\mathbf{N},S,0,<>)$ is finitely axiomatisable, but its restriction by forgetting <, which is $Th(<\mathbf{N},S,0>)$, is not so (Enderton 1972, p. 178-187).

After constructing a specification, by adding new symbols and axioms, some details of the construction history may be considered no longer relevant, then one can use restriction to hide them. For instance, upon constructing the signed integers as pairs $\langle s,n \rangle$ of signs and naturals, one may decide to hide the original sorts, together with some operations. Also, a combination of extension with restriction can be used to 'rename' symbols. For a simple example, reconsider our discussion of the presentation of the ordering of the natural numbers based on lt (less than) in 3.1. Assume that we wish a more user-friendly infix <. We can clearly extend our presentation introducing < by the (defining) axiom $\forall x,y[x<y \leftrightarrow lt(x,y)]$. The effect of now restricting by forgetting lt will be that of renaming lt as <.

If L' is a sub-language of L", then a structure for L" consists of realisations for the symbols of L' and for the symbols only of L". Thus, given a structure $\mathfrak{M}$ for language L", if we restrict it to L', by forgetting the realisations for the symbols of L', we obtain a structure $\mathfrak{M}\downarrow L'$ for L', called the *reduct* of $\mathfrak{M}$ to sub-language $L' \subseteq L''$. We also say that $\mathfrak{M}$ is an *expansion* of $\mathfrak{M}\downarrow L'$ to language L" (Shoenfield 1967, p. 43; van Dalen 1989, p. 116). Since satisfaction of a sentence depends only on the realisations of its extra-logical symbols, for a sentence $\sigma$ of L': $\mathfrak{M} \models \sigma$ iff $\mathfrak{M}\downarrow L' \models \sigma$.

## 3.4 Conservative and eliminable extensions

We now briefly examine two properties that an extension may or may not have: conservativeness and eliminability. The former is specially important in stepwise refinement, and both properties are characteristic of extensions by definitions.

When performing an extension one expects an increase in both deductive and expressive powers. On the other hand, one often does not wish to disturb the smaller theory. When the extension only adds properties of the new symbols, without disturbing the old ones, one has a conservative extension, which does not increase the deductive power for the properties of the old symbols. Similarly, the increase in expressive power may be inessential, in which case our extension has the property of eliminability.

Versions of conservativeness and eliminability are important also in the algebraic approach, which emphasises equations between ground terms (Ehrig and Mahr 1985). Here we consider general versions by means of relativisations to sets of formulas.

Consider theories T' and T" as well as a set $\Sigma$ of sentences common to both underlying languages. We say that T" *$\Sigma$-extends* T' ($T"\supseteq T'[\Sigma]$) iff for each sentence $\sigma\in\Sigma$, $T"\models\sigma$ whenever $T'\models\sigma$; and that T" *$\Sigma$-conserves* T' (($T"\geq T'[\Sigma]$)) iff for each sentence $\sigma\in\Sigma$, $T'\models\sigma$ whenever $T"\models\sigma$. Now, consider a theory T and sets of formulae $\Psi$ and $\Theta$ of its language. We say that T *eliminates $\Psi$ in favour of* $\Theta$ ($T:\Psi\Rightarrow\Theta$) iff for every formula $\psi\in\Psi$ there exists a formula $\theta\in\Theta$ such that $T\models(\psi\leftrightarrow\theta)$

Some special cases of these concepts are found in the literature. For instance, if we specialise $\Sigma$ to equalities between ground terms of L' and $\Psi$ and $\Theta$, respectively, to equalities of the form $v\approx_s t'$ and $v\approx_s t"$ (with $t'\in Nm(L')$ and $t"\in Nm(L")$), we get the versions used for initial algebras (Ehrig and Mahr 1985). Also, quantifier elimination (Enderton 1972, p. 181; Shoenfield 1967, p. 83) can be regarded as a special case of $\Psi$-$\Theta$-eliminability.

For specifications P'=<L',G'> and P"=<L",G"> we are generally interested in special versions of the above concepts. By a *conservative extension* (denoted $P'\leq P"$) we mean an extension $P'\subseteq P"$ such that, for every sentence $\sigma$ of L', $\sigma\in Cn[P"]$ iff $\sigma\in Cn[P']$, i. e. Cn[P"] Sent(L')-conserves Cn[P']. By an *eliminable extension* (denoted $P'\Leftarrow P"$) we mean an extension $P'\subseteq P"$ such that for every formula $\psi\in Frml(L")$ there exists a formula $\theta\in Frml(L')$ with $(\psi\leftrightarrow\theta)\in Cn[P"]$, i. e. Cn[P"] eliminates Frml(L") in favour of Frml(L').

Both relations ≤, of conservative extension, and ⇐, of eliminable extension, between specifications are reflexive and transitive.

Eliminability can be reduced to elimination of the atomic formulae, from which the other are constructed. Indeed a simple charaterisation of eliminability is as follows. Extension P'⊆P" is eliminable iff for each atomic formula α∈(Atfm(L")-Atfm(L')) there exists a formula θ of L' such that (α↔θ)∈Cn[P"].

Conservative extensions play important roles in developing specifications, in particular in parameterisation and implementation. Part of this importance stems from the fact that they preserve consistency: a conservative extension of a consistent specification is consistent. In the case of maximal consistency, the converse also hols: for a maximally consistent P', an extension P"⊇P' is conservative iff P" is consistent.

Now, consider a specification P'=<L',G'> and assume we wish to extend it to P"=<L",G">, where G"=G'∪N with N being the set of new axioms. There are basically two ways to proceed, depending on whether the first step extends the language or keeps it fixed, as illustrated in figure 3.1.

1. We first extend the languages, from L' to L", and then add the set N of new axioms.
2. Within language L', we first extend G' to some set G⊇ G' of sentences of L', and then extend the languages, from L' to L", and add a set N" of sentences of L".

$$\begin{array}{ccc|ccc}
<L',G'> & \subset & <L'',G''> & <L',G'> & & \\
 & & \cap & \cap & & \\
 & & <L'',G' \cup N> & <L',G> & \subset & <L'',G \cup N''>
\end{array}$$

Fig. 3.1: Two ways of extending presentations

In the second approach above, the first step involves selecting a set of axioms in the smaller language. A natural choice for this intermediate step is the restriction P"√L' of P" to L'. Then we get the bonus that the second extension, from P"√L' to P" is conservative. In fact, this provides a simple, albeit not very useful, characterisation of conservativeness: an extension P'⊆P" of specifications is conservative iff P' is equivalent to the restriction P"√L' of P" to sub-language L'=Lang(P') (P'≡P"√L').

Checking conservativeness a-posteriori is in general not easy. We shall later examine some ways of guaranteeing it by construction. Some useful criteria for conservativeness are given in the next result (Shoenfield 1967, p. 65, 95), which uses model-oriented counterparts of conservativeness. An extension $P' \subseteq P''$ is *expansive* (notation $P' \preceq P''$) iff every model $\mathfrak{M}' \in Mod[P']$ has an expansion to a model $\mathfrak{M}'' \in Mod[P'']$, i. e. $\mathfrak{M}'' \downarrow L' = \mathfrak{M}'$. A less restrictive variant is elementary expansiveness: extension $P' \subseteq P''$ is *elementarily expansive* iff for every $\mathfrak{M}' \in Mod[P']$ there exists some model $\mathfrak{M}'' \in Mod[P'']$ such that $\mathfrak{M}'' \downarrow L'$ is elementarily equivalent to $\mathfrak{M}'$ ($\mathfrak{M}'' \downarrow L' \equiv \mathfrak{M}'$).

**Proposition** Model-theoretic criterion for conservativeness

An extension $P' \subseteq P''$ of specifications is conservative iff it is elementarily expansive. In particular, an expansive extension is conservative.

In the first approach above, the first step amounts to simple addition of new symbols without any new axiom. This is an extremely simple case of extension, which we may call an extension by symbols. An *extension by symbols* is an extension $P' \subseteq P''$ where Axm(P'')=Axm(P'). Clearly, since the nothing is required of the new symbols, we may realise them as we please. Thus, an extension by symbols is expansive, hence conservative.

Property-oriented versions of the above criterion for conservativeness are provided in the next proposition, which generalises the idea that conservative extensions preserve consistency.

**Proposition** Property-oriented criteria for conservativeness

For an extension $P' \subseteq P''$ of specifications the following are equivalent.

a) Extension $P' \subseteq P''$ is conservative.

b) For any set of sentences $\Sigma$ of Lng(P'), if Axm(P')$\cup\Sigma$ is consistent, then so is Axm(P')$\cup\Sigma$.

c) For any set of sentences $\Sigma$ of Lng(P'), if Axm(P')$\cup\Sigma$ is maximally consistent, then Axm(P')$\cup\Sigma$ is consistent.

We view assertion (c) as a property-oriented version of elementary expansiveness. This underlines an important methodological point in our approach: we do not care to distinguish elementary equivalent structures, because they have the same properties, as expressed in the language.

As mentioned, conservative extensions play a crucial role in our approach, but conservativeness is generally not easy to check a-posteriori. So it is interesting to have constructs for extension that guarantee this property. Two extreme cases of conservative extensions are extensions by symbols and extensions by definitions. The former is not very interesting in practice because nothing is said about the new symbols, which renders them unlikely to be useful in implementations. The latter are quite useful, but are sometimes too restrictive, since they do not leave room for further refinement.

In constructing an extension one may proceed in steps, each step adding a few symbols and axioms. Each such step, as seen above, may in turn be decomposed into first adding the new symbols and then the new axioms. The first step is an extension by symbols, hence conservative (and generally not eliminable). The properties of the composite extension will heavily depend on the second step, addition of the new axioms.

In the sequel we shall examine a few methods for extending a specification, with emphasis on criteria for conservativeness. Some of these methods are traditional in logic, whereas others have their origin in program development.

We shall consider separately the addition of predicates, operations, constants and sorts. The reason for this separation is the distinct effects that these additions have on the language. For the case of sorts, we shall examine sort introduction constructs akin to those found in many programming languages. For the other cases, we shall examine classical explicit definitions, as well as some slightly more liberal constructs, which will be of particular interest in the case of introducing new operations. Extensions where no new sort is introduced are often called enrichments (Ehrig and Mahr 1985; Goguen *et al.* 1978).

In examining extensions it is sometimes convenient to consider the removal of symbols. Removal illustrates why sorts are more difficult to handle. One can remove a single predicate, operation or constant symbol. But, when removing a sort, one must also remove the other symbols that touch it (include it in their profiles) for obtaining a language.

## 3.5 Extensions by predicates

The introduction of predicate symbols is in general not very problematic. When one adds new predicate symbols to a language one has new (atomic) formulae, but no new terms or variables.

A case of particular importance is that of extension by definition of a new predicate. This occurs in our example BOOL_EXT_NEG&LESS, where the introduction of predicate symbol less? in BOOL is by means of the defining axiom $(\forall x,y{:}Bool)[less?(x,y) \leftrightarrow (x \approx_{Bool} fl \wedge y \approx_{Bool} tr)]$ (see Spec. 3.2 in 3.10). The idea is that the effect of the new predicate less? is explained by the old formula $(x \approx_{Bool} fl \wedge y \approx_{Bool} tr)$. Another simple example stems from the interdefinability between lt and le mentioned in 3.1.

We call language L" an *extension* of L' *by predicate symbols* iff the only new symbols in L" are predicate symbols over sorts of L', i. e. Srt(L')=Srt(L"), Opr(L')=Opr(L"), Var(L')=Var(L"), and Prd(L')$\subseteq$Prd(L"). Then, Trm(L')=Trm(L") and Frml(L')$\subseteq$Frml(L").

Let us now examine the introduction of a new predicate symbol. The situation is as follows. We have a specification P'=<L',G'> and a symbol r not in language L'. We wish to extend P' to P"=<L",G"> where r is introduced as a predicate symbol over given sorts $s_1,\dots,s_m$, already in L'. For this purpose we first form language L" := L'$\cup\{r(s_1,\dots,s_m)\}$, called the *extension* of L' by the *addition of predicate symbol* r over sorts $s_1,\dots,s_m$.

The next step will be the addition of some sentence(s) of language L" as new axiom(s), presumably connecting r to the other symbols. We shall examine separately various forms such new axioms may have and their consequences, mainly in connection with conservativeness. These forms are called (classical, explicit) definition, constraints and inductive closure.

## 3.5.1 Extensions by definitions of predicates

The introduction of a new (predicate) symbol by classical definition occurs when the new axiom, in this case called defining axiom, provides an explicit definition for this new symbol in terms of the old ones.

Our example BOOL_EXT_NEG&LESS shows the defining axiom $(\forall x,y{:}Bool)[less?(x,y)\leftrightarrow(x\approx_{Bool}fl\wedge y\approx_{Bool}tr)]$ for predicate symbol less?. The idea is that one may view the new atomic formula less?(x,y) as an abbreviation for the old formula $(x\approx_{Bool}fl\wedge y\approx_{Bool}tr)$.

Let us first review the nature of a definition for a predicate.

Let L be a language with a predicate symbol r over sorts $s_1,\dots,s_m$. A *definition of predicate symbol* r is a sentence $\delta(r\backslash\rho)$ of the form:

$$(\forall v_1{:}s_1)\dots(\forall v_m{:}s_m)[r(v_1,\dots,v_m)\leftrightarrow\rho] \qquad (\delta(r\backslash\rho))$$

where $v_1,\dots,v_m$ are variables ranging over sorts $s_1,\dots,s_m$ and $\rho$ is a formula, called the *definiens* of r, with no free occurrences of variables other than $v_1,\dots,v_m$. The specification $Def(r\backslash\rho) := <L,\delta(r\backslash\rho)>$ is called the *definition of predicate symbol* r *in terms of formula* $\rho$.

The reason for the name for specification $Def(r\backslash\rho)=<L,\delta(r\backslash\rho)>$ is the following model-oriented property: $\mathfrak{M}\in Mod[Def(r\backslash\rho)]$ iff $\mathfrak{M}[r]=\mathfrak{M}[\rho]$. Thus, predicate r is indeed explained by formula $\rho$.

Now, consider the *sub-language* $L^*$=L-{r} of L obtained by *removal* of symbol r, and assume that the definiens $\rho$ is in $L^*$. We then have eliminability: for every formula $\psi$ of L there exists a formula $e(\psi)$ of $L^*$ such that $\delta(r\backslash\rho)\models[\psi\leftrightarrow e(\psi)]$. Another basic property of definition is unique expandability: each structure $\mathfrak{M}$ for sub-language $L^*$ has a unique expansion to a model $\mathfrak{M}^r\in Mod[Def(r,\rho)]$. Thus, definitions are conservative and eliminable, properties which characterise them.

**Proposition** Characterisation of definition of predicate

Consider the sub-language $L^* = L\text{-}\{r\}$ of L obtained by removal of a predicate symbol r. Then, an extension $P=<L,G>$ of $P^*=<L^*,G^*>$ is both conservative and eliminable iff P is equivalent to $<L,G^*\cup\{\delta(r\backslash\rho)\}>$ for some formula $\rho$ of $L^*$.

**Proof.**

($\Leftarrow$) From the preceding well-known remarks.

($\Rightarrow$) Let r be over sorts $s_1,\ldots,s_m$.

Consider the atomic formula $r(v_1,\ldots,v_m)$ of L. By eliminability, we have a formula $\rho$ of $L^*$ such that $(\forall v_1:s_1)\ldots(\forall v_m:s_m)[r(v_1,\ldots,v_m)\leftrightarrow\rho]\in Cn[P]$.

Let $\delta(r\backslash\rho)$ be the sentence $(\forall v_1:s_1)\ldots(\forall v_m:s_m)[r(v_1,\ldots,v_m)\leftrightarrow\rho]$ of L and consider $P^\# := <L,G^*\cup\{\delta(r\backslash\rho)\}>$. We claim that $P^\#$ is equivalent to P.

Since $\delta(r,\rho)\in Cn[P]$, we have $P^\#\subseteq P$.

For $P\subseteq P^\#$, consider a sentence $\tau\in G$. We will show that $\tau\in Cn[P^\#]$.

By eliminability, we have a sentence $\sigma=e(\tau)$ of $L^*$ such that $\delta(r\backslash\rho)\models[\tau\leftrightarrow\sigma]$.

Now, since $\tau\in G$, $\sigma\in Cn[P]$, and conservativeness yields $\sigma\in Cn[P^*]$. Thus, $\sigma\in Cn[G^*\cup\{\delta(r\backslash\rho)\}]$ and $\tau\in Cn[P^\#]$, as claimed.

*QED*

Finally, specification $P''=<L'',G''>$, with

language $L''=L'\cup\{r(s_1,\ldots,s_m)\}$ being the extension of L' by the addition of predicate symbol r over some sorts $s_1,\ldots,s_m$ of L', and

axiomatisation $G''=G'\cup\{\delta(r\backslash\rho)\}$ the extension of G' by the definition predicate symbol r with definiens $\rho$ a formula of sub-language L',

is called the *extension* of $P'=<L',G'>$ *by definition* of the *predicate symbol* r (with *defining axiom* $\delta(r,\rho)$). We shall sometimes use the notation PD_DEF$[r(s_1,\ldots,s_m)\backslash\rho](P')$ for $<L'\cup\{r(s_1,\ldots,s_m)\},G'\cup\{\delta(r\backslash\rho)\}>$.

The introduction of a single new predicate by definition can be generalised to the case of several predicates. We call specification $P''=<L'',G''>$

an *extension* of $P'=\langle L',G'\rangle$ *by definitions of predicate symbols* iff L" is an extension of L' by predicate symbols and $G'' := G' \cup \{\delta(r) \backslash r \in Prd(L'')-Prd(L')\}$, where, for each $r \in Prd(L'')[s_1 \dots s_m]-Prd(L')$, $\delta(r)$ is a defining axiom of the form $(\forall v_1{:}s_1)\dots(\forall v_m{:}s_m)[r(v_1,\dots,v_m) \leftrightarrow \rho]$, where $\rho$ is a formula $\rho$ of L' with no free occurrences of variables other than $v_1,\dots,v_m$, called the *definiens* of r. The basic properties of unique expandability, conservativeness and eliminability also generalise to this case.

### 3.5.2 Extensions by constraints on predicates

The defining axiom of a new predicate is a biconditional, a crucial feature for guaranteeing eliminability. Its effect is forcing the realisation of the new predicate to be equal to that of its definiens. It is sometimes convenient to relax this equality to inclusion. This leads to the idea of inner and outer constraints, which will be more interesting in introducing new operations.

Let L be a language with a predicate symbol r over sorts $s_1,\dots,s_m$, and consider a formula $\rho$ with no free occurrences of variables other than $v_1,\dots,v_m$ (over sorts $s_1,\dots,s_m$). An *outer constraint* for predicate r is a sentence $o(r\backslash\rho)$ of the form

$$(\forall v_1{:}s_1)\dots(\forall v_m{:}s_m)[r(v_1,\dots,v_m) \rightarrow \rho] \qquad (o(r\backslash\rho))$$

where $\rho$ is called the *outer constraint* on r. The specification $Out(r\backslash\rho) := \langle L, o(r,\rho)\rangle$ is called the *outer constraint of predicate symbol* r *under formula* $\rho$. Similarly, an *inner constraint* for predicate r is a sentence $\iota(r\backslash\rho)$ of the form

$$(\forall v_1{:}s_1)\dots(\forall v_m{:}s_m)[\rho \rightarrow r(v_1,\dots,v_m)] \qquad (\iota(r\backslash\rho))$$

where $\rho$ is called the *inner constraint* on r. The specification $Inn(r\backslash\rho) := \langle L, \iota(r,\rho)\rangle$ is called the *inner constraint of predicate symbol* r *over formula* $\rho$.

The reason for the names of specifications $Out(r\backslash\rho)=\langle L, o(r\backslash\rho)\rangle$ and $Inn(r\backslash\rho)=\langle L, \iota(r\backslash\rho)\rangle$ is the following model-oriented property: $\mathfrak{M} \in Mod[Out(r\backslash\rho)]$ iff $\mathfrak{M}[r] \subseteq \mathfrak{M}[\rho]$ and $\mathfrak{M} \in Mod[Inn(r\backslash\rho)]$ iff $\mathfrak{M}[r] \supseteq \mathfrak{M}[\rho]$.

Now, specification $P''=\langle L'',G''\rangle$, where

language $L''=L' \cup \{r(s_1,\dots,s_m)\}$ is the extension of L' by the addition of predicate symbol r over some sorts $s_1,\dots,s_m$. of L', and

axiomatisation $G''=G' \cup \{o(r\backslash\rho)\}$ is the extension of G' by an outer constraint on predicate symbol r with constraint $\rho$ a formula of sub-language L',

is called the *extension* of $P'=\langle L',G'\rangle$ *by outer constraint* on the *predicate* *symbol* r (with *outer constraint* $o(r\backslash\rho)$). Similarly, the *extension of* $P'=<L',G'>$ *by inner constraint* $\rho$ *on predicate symbol* r is the specification $P''=<L'\cup\{r(s_1,\dots,s_m)\},G'\cup\{\iota(r\backslash\rho)\}>$, where constraint $\rho$ is a formula of sub-language L'. We shall sometimes use for these extensions the notations $PD\_OUT[r(s_1,\dots,s_m)\backslash\rho](P')$ and $PD\_INN[r(s_1,\dots,s_m)\backslash\rho](P')$.

The connections established by constraints are rather loose. The idea behind constraints is splitting the biconditional of a defining axiom into two conditionals. Indeed a defining axiom $\delta(r\backslash\rho)$ of the form $(\forall v_1:s_1)\dots(\forall v_m:s_m)[r(v_1,\dots,v_m)\leftrightarrow\rho]$ is equivalent to the conjunction $o(r\backslash\rho)\wedge\iota(r\backslash\rho)$ of its corresponding outer and inner constraints. So we no longer have eliminability but we still have expansiveness.

**Proposition** Properties of extension by constraint on predicate

Assume that specification $P''=<L'',G''>$ is an extension of $P'=<L',G'>$ by outer or inner constraint on a predicate symbol. Then, extension $P'\subseteq P''$ is expansive, hence conservative.

### 3.5.3 Extensions by inductive predicates

When dealing with graphs one often considers reachability predicates. For a unsorted example, consider unary predicate r under axiom $\forall y\{r(y)\leftrightarrow[y\approx c\vee\exists x(r(x)\wedge f(x)\approx y)]\}$, where c is a constant and f a unary operation. This illustrates the introduction of a new predicate r by a "recursive definition", which is available even without considering inductive constructors.

Let L be a language with a predicate symbol r over sorts $s_1,\dots,s_m$ and consider two formulae: $\beta$ with no free variables other than $y_1,\dots,y_m$ (over sorts $s_1,\dots,s_m$), and $\psi$ with no free variables other than $x_1,\dots,x_m,y_1,\dots,y_m$ (over sorts $s_1,\dots,s_m,s_1,\dots,s_m$). An *inductive constraint* for predicate r is a sentence $*(r\backslash\beta,\psi)$ of the form

$$(\forall y_1:s_1)\dots(\forall y_m:s_m)\{r(y_1,\dots,y_m)\leftrightarrow[\beta\vee(\exists x_1:s_1)\dots(\exists x_m:s_m)(r(x_1,\dots,x_m)\wedge\psi)]\}$$

where $\beta$ and $\psi$ are called the *inductive basis* and *inductive step* for r. The specification $Ind\_Cl(r\backslash\beta,\psi) := <L,*(r\backslash\beta,\psi)>$ is called the *inductive closure of predicate symbol* r *under basis* $\beta$ and *step* $\psi$.

The reason for the name for specification $Ind\_Cl(r\backslash\beta,\psi)=<L'',*(r;\beta,\psi)>$ is as follows, considering a unsorted unary predicate to simplify the notation. The inductive constraint $*(r\backslash\beta,\psi)$ is equivalent to the conjunction of the following sentences:

$\forall y[\beta(y)\rightarrow r(y)]$ $\{r \text{ includes } \beta\}$,

$\forall x\forall y[(r(x)\wedge\psi(x,y))\rightarrow r(y)]$ $\{r \text{ closed under } \psi\}$,

$\forall y[r(y)\rightarrow[\beta(y)\vee\exists x(r(x)\wedge\psi(x,y))]]$.

Thus, a structure $\mathfrak{M}$ for L where $\mathfrak{M}[r]$ is the closure of $\mathfrak{M}[\beta]$ under $\mathfrak{M}[\psi]$ is a model of Ind_Cl($r\backslash\beta,\psi$).

A caveat for the unattentive reader: care must be exercised not to read into the inductive constraint $*(r\backslash\beta,\psi)$ more than is written; it does not force $\mathfrak{M}[r]$ to be exactly the closure of $\mathfrak{M}[\beta]$ under $\mathfrak{M}[\psi]$. Indeed, reconsider our introductory example $\forall y\{r(y)\leftrightarrow[y\approx c\vee\exists x(r(x)\wedge f(x)\approx y)]\}$. Consider a nonstandard model $\mathfrak{M}$ of the naturals, i. e. a copy of the naturals plus a few copies of the integers (Enderton 1972; Ebbinghaus *et al.* 1984; van Dalen 1989), with $\mathfrak{M}[c]=0\in\mathbf{N}$ and $\mathfrak{M}[f]$ being successor in $\mathbf{N}$ and within each copy of $\mathbf{Z}$ . Then, the elements reachable from $\mathfrak{M}[c]$ via $\mathfrak{M}[f]$ constitute exactly the standard copy $\mathbf{N}$. A possible model for $*(r\backslash y\approx c, y\approx f(x))$ is obtained by taking $\mathfrak{M}[r]$ to consist of $\mathbf{N}$ plus some copies of $\mathbf{Z}$. (If we have an inductive axiom like $r(c)\wedge\forall x[r(x)\rightarrow r(f(x))]\rightarrow\forall y r(y)$, it will force $\mathfrak{M}[r]$ to be all of the domain.) In any case, $\mathfrak{M}[r]$ does not consist only of the elements reachable from $\mathfrak{M}[c]$ via $\mathfrak{M}[f]$. Notice, however, that this problem affects only those with model-oriented eyes; our approach does not even perceive it, since we do not discriminate elementarily equivalent models.

As before, we are mainly interested in the case where the basis and step formulae $\beta$ and $\psi$ are in sub-language L'. Consider the case where $L''=L'\cup\{r(s_1,\dots,s_m)\}$ is the extension of L' by the addition of predicate symbol r over sorts $s_1,\dots,s_m$. Then, specification $P''=\langle L'\cup\{r(s_1,\dots,s_m)\},G'\cup*(r\backslash\beta,\psi)\}\rangle$ is called the *extension of* $P'=\langle L',G'\rangle$ *by inductive closure of the predicate symbol* r (with *basis* $\beta$ and *step* $\psi$) and will sometimes be denoted by PD_INDCL[$r(s_1,\dots,s_m)\backslash\beta,\psi$](P').

The basic properties of these extensions are similar to those of extensions by constraints.

**Proposition** Properties of extension by inductive closure of predicate

Assume that specification $P''=\langle L'',G''\rangle$ is an extension of $P'=\langle L',G'\rangle$ by inductive closure of a predicate symbol. Then, extension $P'\subseteq P''$ is expansive, hence conservative.

As before, expandability is not unique and we cannot ensure eliminability.

## 3.6 Extensions by operations

The introduction of operation symbols requires some care. When one adds new operation symbols to a language one has new terms and (atomic) formulae, but no new variables. The basic idea is extracting functions from relations. A formula with free variables denotes, on a structure, a relation. We wish to "extract" a function from such a relation.

A case of particular importance is that of extension by definition of a new operation. This appears in our example BOOL_EXT_NEG&LESS (see Spec. 3.2 in 3.10), where operation symbol neg is introduced into BOOL by means of the new axiom $(\forall x,y{:}Bool)(\neg neg(x)\approx_{Bool}x)$, which has the effect of explaining the new atomic formula $neg(x)\approx_{Bool}y$ by means of the old formula $[(x\approx_{Bool}fl\wedge y\approx_{Bool}tr)\vee(x\approx_{Bool}tr\wedge y\approx_{Bool}fl)]$.

We call language L" an *extension* of L' *by operation symbols* iff the only new symbols in L" are operation symbols between sorts of L', i. e. Srt(L')=Srt(L"), Prd(L')=Prd(L"), Var(L')=Var(L"), and Opr(L')$\subseteq$Opr(L"). Then, Trm(L')$\subseteq$Trm(L") and Frml(L')$\subseteq$Frml(L").

Let us now examine the introduction of a new operation symbol. The situation is as follows. We have a specification P'=<L',G'> and a symbol f not in language L'. We wish to extend P' to P"=<L",G"> where f is introduced as an operation symbol involving given sorts $s_1,\dots,s_n$ and s, already in L'. For this purpose we first extend L' to L" := $L'\cup\{f(s_1,\dots,s_n)\rightarrow s\}$, called the *extension* of L' by the *addition of operation symbol* f from sorts $s_1,\dots,s_n$ to s.

The next step will be the addition of some sentence(s) of language L" as new axioms. In the sequel we shall examine various forms such new axioms may take and their consequences, mainly in connection with conservativeness.

If conservativeness is to be achieved, the introduction of a new operation symbol imposes requirements on new axioms with respect to the starting specification. These requirements may be too restrictive in the case of definitional extensions, which renders constraints more interesting in this case.

### 3.6.1 Extensions by definitions of operations

The introduction of a new operation symbol by classical definition occurs when its behaviour is explained in terms of the the old symbols. In other words, its defining axiom should describe its graph by means of the available concepts.

Let L be a language with an operation symbol f from sorts $s_1,\ldots,s_n$ to s. A *definition of operation symbol* f is a sentence $\delta(f\backslash\theta)$ of the form:

$$(\forall x_1{:}s_1)\ldots(\forall x_n{:}s_n)(\forall y{:}s)[f(x_1,\ldots,x_n)\approx_s y\leftrightarrow\theta] \qquad (\delta(f\backslash\theta))$$

where $x_1,\ldots,x_n,y$ are variables ranging over sorts $s_1,\ldots,s_n,s$ and $\theta$ is a formula, called the *definiens* of f, with no free variables other than $x_1,\ldots,x_n,y$. The specification $Def(f\backslash\theta) := \langle L,\delta(f\backslash\theta)\rangle$ is called the *definition of operation symbol* f *in terms of formula* $\theta$.

The reason for the name of specification $Def(f\backslash\theta)=\langle L'',\delta(f\backslash\theta)\rangle$ is, as in the case of predicates, the property: $\mathfrak{M}\in Mod[Def(f\backslash\theta)]$ iff $\mathfrak{M}[f]=\mathfrak{M}[\theta]$. Thus, the graph of operation f is the relation realising $\theta$. The defining axiom equates the graph of operation f with a relation, so this relation should the graph of some function. These requirements are expressed by the existence and uniqueness conditions. The *existence condition* $\exists(\theta)$ for formula $\theta$ is the following sentence of L

$$(\forall x_1{:}s_1)\ldots(\forall x_n{:}s_n)(\exists y{:}s)\theta(x_1,\ldots,x_n,y) \qquad (\exists(\theta))$$

and the *uniqueness condition* $!(\theta)$ for formula $\theta$ is the sentence

$$(\forall x_1{:}s_1)\ldots(\forall x_n{:}s_n)(\forall y',y''{:}s)\{[\theta(x_1,\ldots,x_n,y')\wedge\theta(x_1,\ldots,x_n,y'')]\rightarrow y'\approx_s y''\} \qquad (!(\theta))$$

These requirements indicate the restricitve nature of classical definitions of operation symbols, because $\delta(f\backslash\theta)\models\exists(\theta)\wedge!(\theta)$.

But, first let us examine eliminability. For this purpose, let us consider, as before, the *sub-language* $L^*=L-\{f\}$ of L obtained by *removal* of symbol f, and assume that the definiens $\theta$ is in $L^*$. We have eliminability (of formulae, even if not of terms): the operation symbol f can be eliminated from any formula. This is the content of the following well-known result.

**Proposition** Eliminability of defined operation symbol

Consider the sub-language $L^*=L-\{f\}$ of L obtained by removal of a operation symbol f. If $\theta$ is a formula of $L^*$, then, for every formula $\psi$ of L there exists a formula $e(\psi)$ of $L^*$ such that $\delta(f\backslash\theta)\models[\psi\leftrightarrow e(\psi)]$

**Proof outline.**

For each term $f(t_1,\ldots,t_n)$, in view of the defining axiom
$(\forall x_1{:}s_1)\ldots(\forall x_n{:}s_n)(\forall y{:}s)[f(x_1,\ldots,x_n)\approx_s y\leftrightarrow\theta(x_1,\ldots,x_n,y)]$, we have
$(\forall y{:}s)[f(t_1,\ldots,t_n)\approx_s y\leftrightarrow\theta(t_1,\ldots,t_n,y)]$.

If terms $t_1,\ldots,$ and $t_n$ have no occurrence of f, we can eliminate $f(t_1,\ldots,t_n)\approx_s y$ in terms of formula $\theta(t_1,\ldots,t_n,y)$ of L.

This indicates how to eliminate the innermost occurrences of f.
*QED*

Now, specification P"=<L",G">, with
language L" := $L'\cup\{f(s_1,\dots,s_n)\rightarrow s\}$ being the extension of L' by the addition of operation symbol f from sorts $s_1,\dots,s_n$ to s of L', and
axiomatisation $G''=G'\cup\{\delta(f\backslash\theta)\}$ the extension of G' by the definition operation symbol f with definiens θ a formula of sub-language L'
is called the *extension of* P'=<L',G'> *by definition of the operation symbol* f (with *defining axiom* $\delta(f\backslash\theta)$). In other words, an extension by definition of a new operation symbol f has a single new axiom of the form $(\forall x_1{:}s_1)\dots(\forall x_n{:}s_n)(\forall y{:}s)[f(x_1,\dots,x_n)\approx_s y\leftrightarrow\theta]$, with $\theta\in \mathrm{Frml}(L')$. We shall sometimes use the notation $\mathrm{OP\_DEF}[f(s_1,\dots,s_n)\rightarrow s\backslash\theta](P')$ for this extension $<L'\cup\{f(s_1,\dots,s_n)\rightarrow s\},G'\cup\{\delta(f\backslash\theta)\}>$ of P'=<L',G'>.

Actually, our concept of extension by definition of an operation symbol is not exactly the usual one (Shoenfield 1967, p. 59; van Dalen 1989 , p. 147). This is due to the fact that, in introducing an operation symbol by definition, one has to take some care in order to obtain a conservative extension. These precautions are expressed by the existence and uniqueness conditions. The next proposition shows that they are indeed necessary, and sufficient, for the extension to be conservative.

**Proposition** Properties of extensions by definition of operation

Assume that specification P"=<L",G"> is an extension of P'=<L',G'> by definition of an operation symbol in terms of formula θ of L'. Then, the following are equivalent.

a) Extension $P'\subseteq P''$ is conservative.

b) The existence $\exists(\theta)$ and uniqueness $!(\theta)$ conditions for θ are in Cn[P'].

c) Each model $\mathfrak{M}\in \mathrm{Mod}[P']$ has a unique expansion to a model $\mathfrak{M}^f\in \mathrm{Mod}[P'']$.

**Proof.**

($a\Rightarrow b$) Because $P''\vDash\exists(\theta)\wedge!(\theta)$ and the latter is a sentence of L'.

($b\Rightarrow c$) Given $\mathfrak{M}\in \mathrm{Mod}[P']$, since $\mathfrak{M}\vDash\exists(\theta)\wedge!(\theta)$, the relation $\mathfrak{M}[\theta]$ is the graph of a function. By setting $\mathfrak{M}^f[f]=\mathfrak{M}[\theta]$, we obtain the expansion $\mathfrak{M}^f\in \mathrm{Mod}[P'']$, which is the only one satisfying $\delta(f\backslash\theta)$.

($c\Rightarrow a$) By the model-theoretic criterion for conservativeness.
*QED*

As in the case of predicates, conservativeness together with eliminability characterise these extensions.

**Proposition** Characterisation of extension by definition of predicate

Assume that L" is the extension of L' by the addition of an operation symbol. Then, the extension from P'=<L',G'> to P"=<L",G"> is both conservative and eliminable iff P" is equivalent to an extension of P' by definition of the new operation symbol.

### 3.6.2 Extensions by constraints on operations

The restrictive nature of classical definitions leads one to consider two generalisations. The basic idea is relaxing the tight connection imposed by classical definitions - forcing equality of realisations - to less stringent ones with looser connections on realisations (Veloso and Veloso 1990). We shall now consider four such methods: inner and outer constraints, Skolem functions and pre and post conditions.

Inner and outer constraints require the graph of the new function to include, or to be included in, a given relation. The requirements for conservativeness are weaker than in the case of definition, namely uniqueness, for inner constraint, and existence, for outer constraint.

Outer constraints are similar to the traditional method of extending a theory by means of Skolem functions, which has many applications in automated theorem proving. The requirement for conservativeness is again existence, as suggested by the very idea of eliminating existential quantifiers in favour of new function symbols.

We then consider another variation motivated by programming, namely the specification of the input-output behaviour of a program by pre and post conditions. The requirement for conservativeness turns out to be akin to existence. An example of pre and post conditions on an operation is provided by the axiom $(\forall s{:}Set)[\neg empty?(s) \rightarrow blng(sel(s),s)]$ of selection in Sets of Elements (see Spec. 2.4: SET[ELEMENT] in 2.9).

After examining these four extension methods, we shall compare them, among themselves and with definitions, with respect to their power in 3.6.3.

A. *Inner constraints on operations*

We begin by examining inner constraints: the idea is extracting a function whose graph contains a given relation. We have a formula $\theta$, representing a relation and wish to introduce a function whose graph is contrained to include the relation represented by $\theta$. We illustrate this underlying idea (for the case of unary operation) by means of the geometrical visualisation in figure 3.2, where we have marked on the $x_1$-axis the domain of the relation denoted by formula $\theta$.

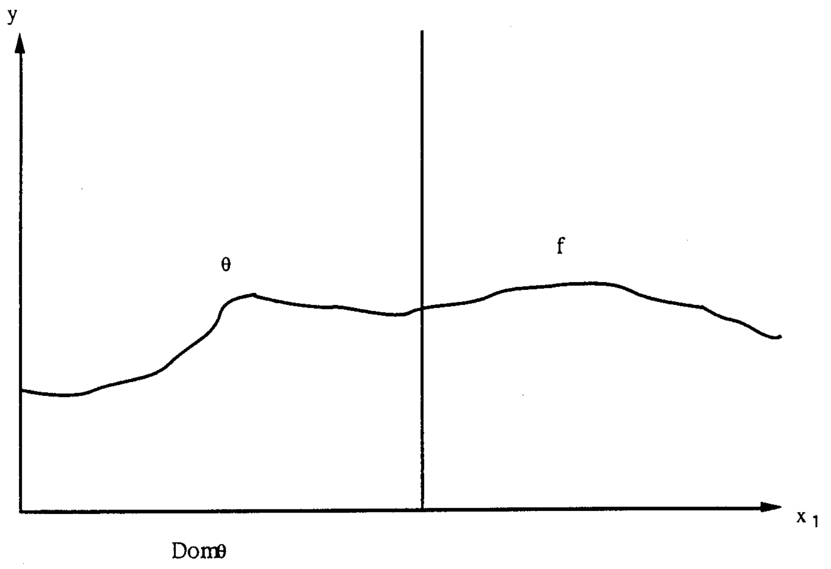

Fig. 3.2: Inner constraint on operation

Let L be a language with an operation symbol f from sorts $s_1,\dots,s_n$ to s, and consider a formula $\theta$ with no free occurrences of variables other than $x_1,\dots,x_n,y$ (over sorts $s_1,\dots,s_n,s$). An *inner constraint* for operation f is a sentence $\iota(f\backslash\theta)$

$$(\forall x_1{:}s_1)\dots(\forall x_n{:}s_n)(\forall y{:}s)[\theta(x_1,\dots,x_n,y)\rightarrow f(x_1,\dots,x_n)\approx_s y] \qquad (\iota(f\backslash\theta))$$

where $\theta$ is called the *inner constraint* on f. Specification $\mathrm{Inn}(f\backslash\theta) := <L,\iota(f\backslash\theta)>$ is called the *inner constraint of operation symbol* f *over formula* $\theta$. The reason for this name is as for the case of predicates: $\mathfrak{M}\in \mathrm{Mod}[\mathrm{Inn}(f\backslash\theta)]$ iff $\mathfrak{M}[f]\supseteq\mathfrak{M}[\theta]$

We are particularly interested in the case where the constraint $\theta$ on f is in sub-language L' and language $L''=L'\cup\{f(s_1,\dots,s_n)\rightarrow s\}$ is the extension of L' by the addition of operation symbol f from sorts $s_1,\dots,s_n$ to s. We then call specification $P''=<L'\cup\{f(s_1,\dots,s_n)\rightarrow s\},G'\cup\{\iota(f\backslash\theta)\}>$, sometimes denoted $\mathrm{OP\_INN}[f(s_1,\dots,s_n)\rightarrow s\backslash\theta](P')$, the *extension* of $P'=<L',G'>$ *by inner constraint* $\theta$ *on operation symbol* f.

The basic properties of these extensions are given in the next result, which shows that a necessary and sufficient requirement for conservativeness is presence in Cn[P'] of the uniqueness condition $(\forall x_1{:}s_1)\ldots(\forall x_n{:}s_n)(\forall y',y''{:}s)\{[\theta(x_1,\ldots,x_n,y')\wedge\theta(x_1,\ldots,x_n,y'')]\rightarrow y'\approx_s y''\}$.

**Proposition** Properties of extension by inner constraint on operation

Assume that specification P"=<L",G"> is an extension of P'=<L',G'> by inner constraint $\theta$ on an operation symbol. Then, the following are equivalent.

a) Extension $P'\subseteq P''$ is conservative.

b) The uniqueness condition $!(\theta)$ is in Cn[P'].

c) Extension $P'\subseteq P''$ is expansive.

**Proof.**

($a\Rightarrow b$) and ($c\Rightarrow a$) are as in the case of extension by definition.

($b\Rightarrow c$) Given $\mathfrak{M}'\in$ Mod[P'], since $\mathfrak{M}'\models !(\theta)$, the relation $\mathfrak{M}'[\theta]$ is the graph of a function on its domain. Take $\mathfrak{M}''[f]$ to be an arbitrary extension of the latter to a total function.

*QED*

B. *Outer constraints on operations*

Let us now move on to the dual of inner constraints: outer constraints. Here the idea is extracting a function whose graph is included in a given relation. A geometrical visualisation of the effect of an outer constraint (for the case of unary operation) is provided in figure 3.3.

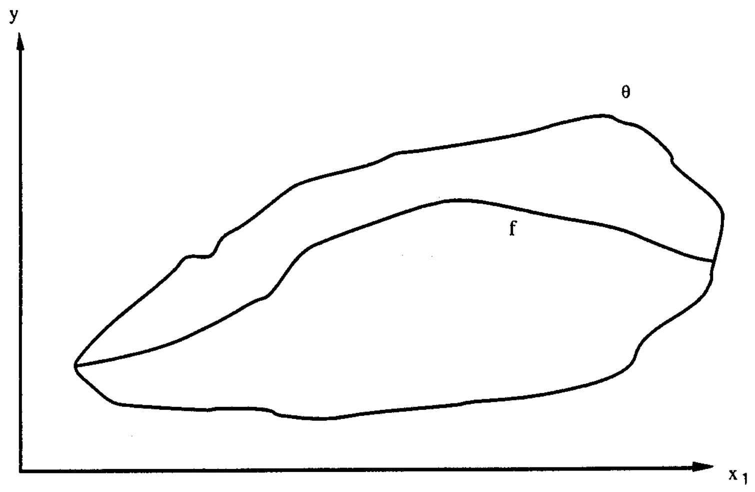

Fig. 3.3: Outer constraint on operation

As before, let L be a language with an operation symbol f from sorts $s_1,\ldots,s_n$ to s, and consider a formula $\theta$ with no free occurrences of variables other than $x_1,\dots,x_n,y$ (over sorts $s_1,\dots,s_n,s$). An *outer constraint* for operation f is a sentence $o(f\backslash\theta)$

$$(\forall x_1:s_1)\dots(\forall x_n:s_n)(\forall y:s)[f(x_1,\dots,x_n)\approx_s y\rightarrow\theta(x_1,\dots,x_n,y)] \qquad (o(f\backslash\theta))$$

where $\theta$ is called the *outer constraint* on f. Specification $\mathrm{Out}(f\backslash\theta) := <L,o(f\backslash\theta)>$ is called the *outer constraint of operation symbol* f *under formula* $\theta$. The reason for this name is as before: $\mathfrak{M}\in\mathrm{Mod}[\mathrm{Out}(f,\theta)]$ iff $\mathfrak{M}[f]\subseteq\mathfrak{M}[\theta]$

In the case where the constraint $\theta$ on f is in sub-language L' and $L''=L'\cup\{f(s_1,\dots,s_n)\rightarrow s\}$ is the extension of L' by the addition of operation symbol f from sorts $s_1,\dots,s_n$ to s, we call specification $P''=<L'\cup\{f(s_1,\dots,s_n)\rightarrow s\},G'\cup\{o(f,\theta)\}>$, denoted $\mathrm{OP\_INN}[f(s_1,\dots,s_n)\rightarrow s\backslash\theta](P')$, the *extension* of $P'=<L',G'>$ *by outer constraint* $\theta$ *on operation symbol* f.

The basic properties of these extensions are given in the next result, showing that a necessary and sufficient requirement for conservativeness is presence of the existence condition $(\forall x_1:s_1)\dots(\forall x_n:s_n)(\exists y:s)\theta(x_1,\dots,x_n,y)$ in Cn[P'].

**Proposition** Properties of extension by outer constraint on operation

Assume that specification $P''=<L'',G''>$ is an extension of $P'=<L',G'>$ by outer constraint $\theta$ on an operation symbol. Then, the following are equivalent.

a) Extension $P'\subseteq P''$ is conservative.

b) The existence condition $\exists(\theta)$ is in Cn[P'].

c) Extension $P'\subseteq P''$ is expansive.

**Proof.**

($a\Rightarrow b$) and ($c\Rightarrow a$) are as before.

($b\Rightarrow c$) Given $\mathfrak{M}'\in\mathrm{Mod}[P']$, the Axiom of Choice gives a function $\mathfrak{M}''[f]$ whose graph is included in the relation $\mathfrak{M}'[\theta]$ with the same domain. Since $\mathfrak{M}'\models\exists(\theta)$, this function is total.

*QED*

C. *Skolem functions*

The geometrical visualisation in figure 3.3 reminds one of the important logical idea of Skolemisation. Indeed, the intuitive role of a Skolem function is selecting a value.

Informal arguments often involve the following line of reasoning: given that for each a there exists some b such that q(a,b), pick one such b and call it f(a)) (to record the fact that the chosen b may very well depend on a). This choice function is what is called a Skolem function. This idea generalises the process of introducing constants as witnesses in Henkin's proof of the completeness theorem (van Dalen 1989, p. 145). It is also instrumental for the process of eliminating existential quantifiers in favour of new function symbols, which is quite important for several methods of automated theorem proving (Manna 1974, p. 125, 126).

Consider a language L with an operation symbol f from sorts $s_1,\dots,s_n$ to s, and a formula $\theta$ of L with no free occurrences of variables other than $x_1,\dots,x_n,y$ (over sorts $s_1,\dots,s_n,s$). A *Skolem axiom* for operation f is a sentence $\sigma(f\backslash\theta)$

$$(\forall x_1{:}s_1)\dots(\forall x_n{:}s_n)\theta(x_1,\dots,x_n,f(x_1,\dots,x_n)) \qquad (\sigma(f\backslash\theta))$$

where $\theta$ is called the *Skolemised formula.* Specification $Skl(f\backslash\theta) := <L,\sigma(f\backslash\theta)>$ is called the *Skolemisation of formula* $\theta$ *by operation symbol* f. The reason for this name should be clear.

In case the Skolemised formula $\theta$ is in sub-language L' and $L''=L'\cup\{f(s_1,\dots,s_n)\rightarrow s\}$ is the extension of L' by the addition of operation symbol f from sorts $s_1,\dots,s_n$ to s, we call specification $P''=<L'\cup\{f(s_1,\dots,s_n)\rightarrow s\},G'\cup\{\sigma(f\backslash\theta)\}>$ the *extension* of $P'=<L',G'>$ *by operation symbol* f as *Skolemisation* for $\theta$ is the specification $P''=<L'\cup\{f(s_1,\dots,s_n)\rightarrow s\},G'\cup\{\sigma(f\backslash\theta)\}>$. When convenient we use for such Skolem extension the notation $OP\_SKL[f(s_1,\dots,s_n)\rightarrow s\backslash\theta](P')$.

The basic properties of Skolem extensions are those of outer constraint: an extension P" of P' by an operation symbol as Skolemisation is conservative iff the existence condition $\exists(\theta)$ is in Cn[P']

D. *Pre and post conditions for operations*

Let us now consider another method for introducing a function symbol, one directly suggested by programming considerations.

The specification of the input-output behaviour of a program usually consists of two formulas, namely a pre-condition $\varphi(x)$ and a post-condition $\psi(x,y)$. The pre-condition is intended to describe the inputs one is interested in, whereas the post-condition describes how the outputs are to be related to the inputs (Manna 1974, p. 164). If we use f to denote the function computed by the program, then what is required is that whenever input a satisfies $\varphi(x)$ then $\psi(a,f(a))$ holds.

This situation can be given a geometrical visualisation in the same spirit as our previous ones. In the present case, pre-condition $\varphi(x)$ describes a subset of the x-axis, and post-condition $\psi(x,y)$ describes a region of the x-y plane as before. So, function f represents a plane curve contained in the region corresponding to the post-condition at least within the vertical stripe over the set corresponding to the pre-condition.

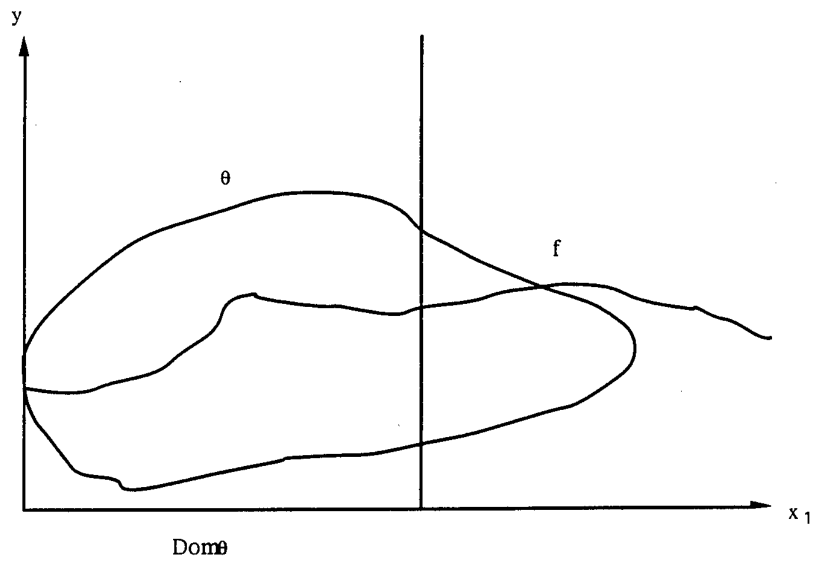

Fig. 3.4: Pre and post condition on operation

These considerations suggest another method of introducing an operation symbol: by means of pre and post conditions.

Let L be a language with an operation symbol f from sorts $s_1,\dots,s_n$ to s, and consider two formulae: $\varphi$ with no free variables other than $x_1,\dots,x_n$ (over sorts $s_1,\dots,s_n$), and $\psi$ with no free variables other than $x_1,\dots,x_n,y$ (over sorts $s_1,\dots,s_n,s$). A *pre-post condition* for operation f is a sentence $\pi(\varphi\backslash f\backslash\psi)$ of the form

$$(\forall x_1:s_1)\dots(\forall x_n:s_n)(\forall y:s)[\varphi(x_1,\dots,x_n)\rightarrow\psi(x_1,\dots,x_n,f(x_1,\dots,x_n))] \qquad (\pi(\varphi\backslash f\backslash\psi))$$

where $\varphi$ is called the *pre-condition* and $\psi$ the *post-conditon* for f. The *viability condition* for *pre-condition* $\varphi$ and *post-conditon* $\psi$ is the following sentence $\omega(\varphi,\psi)$

$$(\forall x_1:s_1)\dots(\forall x_n:s_n)[\varphi(x_1,\dots,x_n)\rightarrow(\exists y:s)\psi(x_1,\dots,x_n,y)] \qquad (\omega(\varphi,\psi)).$$

Specification $PPC(\varphi\backslash f\backslash\psi) := \langle L,\pi(\varphi\backslash f\backslash\psi)\rangle$ is called *operation symbol* f *under pre-condition* $\varphi$ *and post-conditon* $\psi$ The reason for this name is as before:

$\mathfrak{M} \in \mathrm{Mod}[PPC(\varphi\backslash f\backslash\psi)]$ iff the restriction of $\mathfrak{M}[f]$ to $\mathfrak{M}[\varphi]$ is included in $\mathfrak{M}[\psi]$. Notice that $\pi(\varphi\backslash f\backslash\psi) \models \omega(\varphi,\psi)$.

In case the pre and post conditions $\varphi$ and $\psi$ for f are in sub-language L' and $L''=L'\cup\{f(s_1,\ldots,s_n)\to s\}$ is the extension of L' by the addition of operation symbol f from sorts $s_1,\ldots,s_n$ to s, we call specification $P''=<L'\cup\{f(s_1,\ldots,s_n)\to s\},G'\cup\{\pi(\varphi\backslash f\backslash\psi)\}>$ the *extension* of $P'=<L',G'>$ *by operation symbol* f *under pre-condition* $\varphi$ *and post-conditon* $\psi$. A notation for such extension is $\mathrm{OP\_PPC}[f(s_1,\ldots,s_n)\to s\backslash\varphi,\psi](P')$.

The basic property of an extension of $P'=<L',G'>$ by an operation symbol under pre and post conditions is the equivalence of the three conditions: (a) conservativeness, (b) the viability condition $\omega(\varphi,\psi)$ in Cn[P'], and (c) expansiveness.

### 3.6.3 Comparison of constructs for adding operations

We have examined five methods for introducing a new operation symbols and the corresponding requirements for conservativeness. Now, let us examine some relationships among these various ways of extending a theory by the introduction of an operation symbol.

We are considering the process of extending a specification $P'=<L',G'>$" to a new specification $P''=<L'\cup\{f(s_1,\ldots,s_n)\to s\},G'\cup\{\alpha\}>$, where $\alpha$ is the new axiom introducing operation symbol f on the basis of a formula $\theta$ of L'. We have examined the following five methods.

Definition: $\mathrm{OP\_DEF}[f(s_1,\ldots,s_n)\to s\backslash\theta](P')$, with $\alpha$ of the form

$$(\forall x_1{:}s_1)\ldots(\forall x_n{:}s_n)(\forall y{:}s)[f(x_1,\ldots,x_n)\approx_s y\leftrightarrow\theta] \qquad (\delta(f\backslash\theta))$$

and condition for conservativeness $P'\models\exists(\theta)\wedge!(\theta)$.

Inner constraint: $\mathrm{OP\_INN}[f(s_1,\ldots,s_n)\to s\backslash\theta](P')$, with $\alpha$ of the form

$$(\forall x_1{:}s_1)\ldots(\forall x_n{:}s_n)(\forall y{:}s)[\theta(x_1,\ldots,x_n,y)\to f(x_1,\ldots,x_n)\approx_s y] \qquad (\iota(f\backslash\theta))$$

and condition for conservativeness $P'\models !(\theta)$.

Outer constraint: $\mathrm{OP\_OUT}[f(s_1,\ldots,s_n)\to s\backslash\theta](P')$, with $\alpha$ of the form

$$(\forall x_1{:}s_1)\ldots(\forall x_n{:}s_n)(\forall y{:}s)[f(x_1,\ldots,x_n)\approx_s y\to\theta(x_1,\ldots,x_n,y)] \qquad (o(f\backslash\theta))$$

and condition for conservativeness $P'\models\exists(\theta)$.

Skolem extension: $\mathrm{OP\_SKL}[f(s_1,\ldots,s_n)\to s\backslash\theta](P')$, with $\alpha$ of the form

$$(\forall x_1{:}s_1)\ldots(\forall x_n{:}s_n)\theta(x_1,\ldots,x_n,f(x_1,\ldots,x_n)) \qquad (\sigma(f\backslash\theta))$$

and condition for conservativeness $P'\models\exists(\theta)$.

Pre and post conditions: OP_PPC$[f(s_1,\dots,s_n)\rightarrow s\backslash\varphi,\psi](P')$, with $\alpha$ of the form

$$(\forall x_1{:}s_1)\dots(\forall x_n{:}s_n)(\forall y{:}s)[\varphi(x_1,\dots,x_n)\rightarrow\psi(x_1,\dots,x_n,f(x_1,\dots,x_n))] \qquad (\pi(\varphi\backslash f\backslash\psi))$$

and condition for conservativeness $P'\vDash\omega(\varphi,\psi)$.

Let us first compare classical (explicit) definition with constraints. It seems intuitively clear that the former is a special case of the latter. Indeed, a motivation for considering constraints was the quest for a method that is less restrictive than definitions. The next proposition, illustrated in figure 3.5, states this intuition in a precise form.

$$\begin{tikzcd}
\text{Outer}\wedge !(\theta) \arrow[r, Leftrightarrow] \arrow[d, Rightarrow] & \text{Definition} \arrow[r, Leftrightarrow] \arrow[d, Rightarrow] & \text{Inner}\wedge\exists(\theta) \arrow[d, Rightarrow] \\
\exists(\theta)\wedge !(\theta) & \exists(\theta)\wedge !(\theta) & !(\theta)\wedge\exists(\theta)
\end{tikzcd}$$

Fig. 3.5: Definition vs. inner and outer constraint on operation

**Proposition** Definition vs. inner and outer constraint on operation

Assume that specification P"=<L",G"> is a conservative extension of P'=<L',G'>. If L'=L"-{f} then the following are equivalent.

a) Specification P" is equivalent to an extension of P' by definition.

b) P" is equivalent to an extension of P' by outer constraint and $P'\vDash !(\theta)$.

c) P" is equivalent to an extension of P' by inner constraint and $P'\vDash\exists(\theta)$.

**Proof outline.**

($a\Rightarrow b\&c$) Because $\delta(f\backslash\theta)\vDash o(f,\theta)\wedge\iota(f,\theta)$.

($b\Rightarrow a$) The axiom of outer constraint forces the graph of f to be included in $\theta$, and the uniquess condition forces their equality.

($c\Rightarrow a$) is dual to ($b\Rightarrow a$).

*QED*

Let us now compare the constraints for operation among themselves. As we have already hinted at, by properties of equality, we have the equivalence between outer constraint

$$(\forall x_1{:}s_1)\dots(\forall x_n{:}s_n)(\forall y{:}s)[f(x_1,\dots,x_n)\approx_s y\rightarrow\theta(x_1,\dots,x_n,y)] \qquad (o(f\backslash\theta))$$

and Skolem axiom

$$(\forall x_1{:}s_1)\dots(\forall x_n{:}s_n)\theta(x_1,\dots,x_n,f(x_1,\dots,x_n)) \qquad (\sigma(f\backslash\theta))$$

Moreover, each one of these two is equivalent to pre and post conditions, as shown in the next result.

**Proposition** Outer constraint, Skolem and pre and post condition

Assume that specification P"=<L",G"> is an extension of P'=<L',G'>. If L' = L"-{f}, then the following are equivalent.

a) P" is equivalent to an extension of P' by outer constraint.

b) P" is equivalent to a Skolem extension of P'.

c) P" is equivalent to an extension of P' by pre and post conditions.

In each case, the requirements for conservativeness are equivalent.

**Proof outline.**

(a⇔b) As noted $o(f\backslash\theta)$ and $\sigma(f\backslash\theta)$ are equivalent.

(b⇒c) Take $\varphi(x_1,\ldots,x_n)$ as $x_1{\approx_{s_1}}x_1\wedge\ldots\wedge x_n{\approx_{s_n}}x_n$ and $\psi(x_1,\ldots,x_n,y)$ as $\theta(x_1,\ldots,x_n,y)$. Then $\sigma(f\backslash\theta)$ and $\pi(\varphi\backslash f\backslash\psi))$ are equivalent, as are $\exists(\theta)$ and $\omega(\varphi,\psi)$.

(c⇒a) Given $\varphi(x_1,\ldots,x_n)$ and $\psi(x_1,\ldots,x_n,y)$, take $\theta(x_1,\ldots,x_n,y)$ as $[\varphi(x_1,\ldots,x_n)\rightarrow\psi(x_1,\ldots,x_n,y)]$. Then $\pi(\varphi\backslash f\backslash\psi))$ and $\sigma(f\backslash\theta)$ are equivalent, as are $\omega(\varphi,\psi)$ and $\exists(\theta)$.

*QED*

Also, inner constraints can be viewed as special cases of outer constraints.

**Proposition** Inner vs. outer constraint on operation

Any extension P"=<L",G"> of P'=<L',G'> by inner constraint on operation symbol f is equivalent to an extension P of P' by outer constraint on f. Moreover, P" is a conservative extension of P' iff P is so.

**Proof outline.**

Given $\theta(x_1,\ldots,x_n,y)$, select a new variable z of the same sort as y, and take $\theta'(x_1,\ldots,x_n,y)$ as $(\forall z{:}s)[\theta(x_1,\ldots,x_n,z)\rightarrow z\approx_s y]$. Then $\iota(f\backslash\theta)$ and $\sigma(f\backslash\theta')$ are equivalent, as are $!(\theta)$ and $\exists(\theta')$.

*QED*

The comparisons among constraints for operation are illustrated in figure 3.6. More details can be found in (Veloso and Veloso 1990).

pre&post ⇔ Skolem ⇔ outer ⇐ inner

Fig. 3.6: Comparison among constraints for operation

## 3.7 Extensions by constants

We regard constants as nullary operations (with no arguments, only results). So, extensions by constants are special cases of those by operations. But for the case of constants one can get more information (Veloso and Veloso

1991). An example of definition of a constant appears in Sets of Elements (see Spec. 2.4: SET[ELEMENT] in 2.9), where axiom $(\forall s{:}Set)[empty?(s) \leftrightarrow s \approx_{Set} void]$ defines constant void in terms of predicate empty?.

The case of constants is particularly simple, due to the absence of arguments. We can consider the simultaneous introduction of several symbols. We call language L" an *extension by constants* of L' iff the only new symbols in L" are constant symbols involving sorts of L'. Thus, we have some new names (ground terms). In particular, the *extension of language* L' *by constant symbols* $c_1, \dots, c_k$ over sorts $s_1, \dots, s_k$ will be denoted by $L' \cup \{c_1{:}s_1, \dots, c_k{:}s_k\}$.

We shall now examine the simultaneous introduction of constants via Skolemisation. This uses the idea of Skolem constant, a simple special case of Skolem function (van Dalen 89, p.144-146). We shall simplify the notation by restricting ourselves to the unsorted case. since the many-sorted one adds no conceptual difficulty.

Consider a formula $\phi$ of language L of the form $\phi(y_1, \dots, y_k)$, with no variables other than the displayed ones occurring free in it. Let language $L^\#$ be obtained from L by the addition of some new constants, $c_1, \dots, c_k$ being among them. Replace every free occurrence of each variable $y_i$ in $\phi(y_1, \dots, y_k)$ by the new constant $c_i$; this yields the sentence $\phi(c_1, \dots, c_k)$ of $L^\#$. This sentence $\phi(c_1, \dots, c_k)$ is called a *Skolemisation* of the sentence $\exists y_1 \dots \exists y_k \phi(y_1, \dots, y_k)$ of L.

Now, consider theories T over L, and $T^\#$ over $L^\#$. We call $T^\#$ an *extension by Skolem constants* of T iff every new axiom of $T^\#$ is a Skolemisation of some consequence of T. Notice that, in this case, every new constant c of $L^\#$ occurs in a single new axiom of $T^\#$, called its *constraining axiom.*

The following lemma recalls a simple result concerning these concepts. Its proof is simpler than its well-known counterpart for Skolem functions (presented for outer constraints in 3.6.2), in that it does not require the Axiom of Choice.

**Lemma** Expansiveness of extension by Skolem constants

If $T^\#$ is an extension by Skolem constants of T then $T^\#$ is an expansive, hence conservative, extension of T.

**Proof outline.**

Consider a model $\mathfrak{M}$ of T. Let $\phi(c_1, \dots, c_k)$ be a new axiom of $T^\#$. It is the Skolemisation of $\exists y_1 \dots \exists y_k \phi(y_1, \dots, y_k)$, which is a consequence of T. Thus, $\mathfrak{M}$

can be expanded to a structure which is a model of $\phi(c_1,\dots,c_k)$. Since each new constant occurs in a single new axiom, we can thus expand $\mathfrak{M}$ to a model $\mathfrak{M}^\#$ of $T^\#$.

*QED*

We can now characterise the conservative extensions by a set of new constants formed by the addition of only finitely many new axioms.

**Proposition** Finite conservative extensions by Skolem constants

Consider a language L', and let L" be obtained from L' by the addition of a set C of new constants. Consider a theory T' over L' and its extension T" over L" obtained by adding to T' a finite set $\Sigma$ of sentences of L". Then, the following are equivalent.

a) The extension $T' \subseteq T"$ is conservative.

b) T" is equivalent to an extension by Skolem constants of T'.

c) The extension $T' \subseteq T"$ is expansive.

**Proof outline.**

($a \Rightarrow b$) Let $\sigma$ be the conjunction of the new axioms in $\Sigma$. Clearly, $T' \cup \{\sigma\}$ is equivalent to T". We shall now show that $T' \cup \{\sigma\}$ is equivalent to an extension by Skolem constants of T'. First, $\sigma$ has only finitely many new constants, say k of them. By resorting to an alphabetic variant, if necessary, we may assume that variables $y_1,\dots,y_k$ do not occur in $\sigma$. Now, let $\sigma^*$ be obtained from $\sigma$ by replacing each new constant $c_i$ by a corresponding new variable $y_i$. Then, $\sigma$ will be equivalent to a Skolemisation of $\exists y_1 \dots \exists y_k \sigma^*$. But, the latter is a sentence of L' in Cn(T"), whence in Cn(T'), by conservativeness.

($b \Rightarrow c$)&($c \Rightarrow a$) By the preceding lemma.

*QED*

This result shows that extensions by Skolem constants turn out to be the most general conservative extensions by addition of constants.

These ideas apply to the many-sorted case as well.

Consider a language L with constant symbols $c_1,\dots,c_k$ over sorts $s_1,\dots,s_k$, and a formula $\phi$ of L with no free occurrences of variables other than $y_1,\dots,y_k$ (over sorts $s_1,\dots,s_k$). A *(joint) Skolem axiom* for constants $c_1,\dots,c_k$ is a sentence $\sigma(c_1,\dots,c_k \backslash \phi)$ of the form $\phi(c_1,\dots,c_k)$, where $\phi$ is called the *Skolemised formula*. Specification $Skl(c_1,\dots,c_k \backslash \phi) := \langle L, \sigma(c_1,\dots,c_k \backslash \phi) \rangle$ is called the *Skolemisation of formula* $\phi$ *by constant symbols* $c_1,\dots,c_k$.

When the Skolemised formula $\phi$ is in sub-language L' and $L"=L' \cup \{c_1:s_1,\dots,c_k:s_k\}$ is the extension of L' by the addition of constant symbols $c_1,\dots,c_k$ over sorts $s_1,\dots,s_k$, we call specification $P"=<L' \cup \{c_1:s_1,\dots,c_k:s_k\},G' \cup \{\sigma(c_1,\dots,c_k \backslash \phi)\}>$ the *extension* of $P'=<L',G'>$ *by constant symbols* $c_1,\dots,c_k$ as *Skolemisation* for $\phi$. A convenient notation for such extension is $CN\_SKL[c_1:s_1,\dots,c_k:s_k \backslash \phi](P')$. This extension is conservative iff $(\exists y_1:s_1)\dots(\exists y_k:s_k)\phi$ is a consequence of P'.

## 3.8 Extensions by sorts

We shall now consider the introduction of new sorts. This is of importance because it occurs often in implementing formal specifications, when new sorts are "constructed" from the concrete ones. In this context, one generally introduces some new sort(s) and then some new predicate, operation and constant symbol(s) that require these sort(s).

For a simple example consider the overused implementation of stacks by arrays with indices. Here one wishes to represent a stack by means of an array, containing the elements stored in the stack, together with an index, indicating the position of the topmost element. Now, the specification of arrays has three sorts: Arrays, Indices, and Elements. One then wishes to extend it by adding a product sort Arrays×Indices. But, not every pair <a,i> will represent a stack; only a subsort (consisting of those with $i \le size(a)$). Further, pairs with arrays differing only above their indices are to be identified (for they represent the same stack), so one wishes a quotient sort.

The cases of extensions we have examined so far account only for the introduction of new predicate, operation and constant symbols, i. e. cases of enrichment. We call language L" an *enrichment* of L' iff the only new symbols in L" are predicate, operation and constant symbols involving sorts of L', i. e., $Prd(L') \subseteq Prd(L")$, $Opr(L') \subseteq Opr(L")$, but $Var(L')=Var(L")$. Then, $Trm(L') \subseteq Trm(L")$ and $Frml(L') \subseteq Frml(L")$, but $Srt(L')=Srt(L")$.

The introduction of a new sort is more complex than the previous ones. A reasons for this has already been indicated by examing removal. One can remove a single predicate, operation or constant symbol, by itself. But, when removing a sort, one must also remove the other symbols that touch it (include it in their profiles) if a language is to be obtained.

Now looking from the viewpoint of addition, if one wishes to add a symbol involving some sort not already in the language, one must introduce this new sort as well. And when one adds a new sort to a language one gets not only new (atomic) formulae, but new variables as well. For, upon adding a new sort t, one also has:

a new equality symbol $\approx_t$ over sort t, and

new variables over sort t.

We thus have new variables, and hence new terms, in addition to new (atomic) formulae.

The simplest case would be the addition of a single new sort by itself. This is a symbol extension, always conservative, but of little interest. For, one generally wishes to have some information, provided by new axioms, about the behaviour of the new elements.

In the sequel we shall examine a few methods for extending a specification by sorts, with emphasis on criteria for conservativeness. These sort introduction constructs are akin to those found in many programming languages (Ghezzi and Jazayeri 1982; Hoare 1974; Wirth 1973).

We shall consider four constructs that introduce a new sort as a product, a subsort, a quotient, or a discriminated union of existing sorts (Meré and Veloso 1991). These introduce a new sort together with some conversion operations. Another construct, also found in programming languages, introduces a new sort by simply enumerating its elements.

### 3.8.1 Extensions by product of sorts

Assume that we wish to extend the naturals to the (signed) integers. Then it is natural to consider introducing a product sort Bool×Nat. Here Bool stands for the two signs: plus and minus. Also, a product sort can be used to represent regular polygons by pairs: number of edges (a natural) and edge size (a real).

The sort construct product introduces a new sort consisting of the records (ordered pairs) of objects of the given sorts together with the corresponding selectors. Thus, its realisations have a new set $T=S_1\times S_2$, together with its two projections $p_k:T\rightarrow S_k$, with $p_k(\langle a_1,a_2\rangle)=a_k$.

Consider language $L^\times$ consisting of

sorts t, $s_1$ and $s_2$, as well as

operation symbols $p_1$, from sort t to $s_1$, and $p_2$, from t to $s_2$.

We call this language $L^\times$ the *language for for sort* t *as product of sorts* $s_1$ and $s_2$.

An *explication of sort* t as the *product of sorts* $s_1$ and $s_2$ is the set $\times(t\backslash p_1\rightarrow s_1,p_2\rightarrow s_2)$ consisting of the following two sentences of $L^\times$:

$(\forall x_1:s_1)(\forall x_n:s_n)(\exists y:t)[p_1(y)\approx_{s_1}x_1\wedge p_2(y)\approx_{s_2}x_2]$ (pjs)

$(\forall y,y':t)[(p_1(y)\approx_{s_1}p_1(y')\wedge p_2(y)\approx_{s_2}p_2(y'))\to y\approx_t y']$ (pji)

Sentence (pjs) states that the projections are jointly surjective, and (pji) states that the projections are jointly injective. Specification $Prod(t\backslash p_1{:}s_1,p_2{:}s_2) := <L^{\times},\{(pjs),(pji)\}>$ is called the *explication of sort* t as the *product of sorts* $s_1$ and $s_2$, with *projections* $p_1(t)\to s_1$ and $p_2(t)\to s_2$.

The reason for the name of specification $Prod(t\backslash p_1\to s_1,p_2\to s_2) = \langle L^{\times},\times(t\backslash p_1\to s_1,p_2\to s_2)\rangle$ stems from the properties of its models. Clearly, $Prod(t\backslash p_1\to s_1,p_2\to s_2)$ has models $\mathfrak{P}$ such that $\mathfrak{P}[t]=\mathfrak{P}[s_1]\times\mathfrak{P}[s_2]$; such models, where sort t is realised as the set of ordered pairs of elements of $\mathfrak{P}[s_1]$ and $\mathfrak{P}[s_2]$, may be called *canonical models*. Now, given a structure $\mathfrak{M}$ for language $L^{\times}$, $\mathfrak{M}\in Mod[Prod(t\backslash p_1\to s_1,p_2\to s_2)]$ iff $\mathfrak{M}$ is isomorphic to some canonical model $\mathfrak{P}$. Thus, for $\mathfrak{M}\in Mod[Prod(t\backslash p_1\to s_1,p_2\to s_2)]$ the realisation $\mathfrak{M}[t]$ of sort t has indeed the behaviour of the cartesian product of $\mathfrak{M}[s_1]$ and $\mathfrak{M}[s_2]$.

Now, given a language L extending $L^{\times}$, we may remove from L sort t and operations $p_1$ and $p_2$, thereby obtaining its sub-language $L_{\times} := L\text{-}L^{\times}$. We then have expandability: each structure $\mathfrak{M}$ for sub-language $L_{\times}$ has a unique, up to isomorphism, expansion to a model $\mathfrak{M}^{\times}\in Mod[Prod(t\backslash p_1\to s_1,p_2\to s_2)]$.

Eliminability is to be expected. Indeed, in a realisation, an object a of sort T can be regarded as a pair $\langle a_1,a_2\rangle$. So, it is reasonable to replace a relation R, involving T by its 'paired version' $R_{\times}$, involving $S_1$ and $S_2$, so that $\langle\ldots,a,\ldots\rangle\in R$ iff $\langle\ldots,a_1,a_2,\ldots\rangle\in R_{\times}$.

Formally we have to face the fact that we now have variables over the product sort t. This can be overcome by means of a connection, which indicates how to translate variables ranging over t as pairs of variables over $s_1$ and $s_2$. The idea is as follows. Consider variables y, over t, and $v_1,\ldots,v_m$, over sorts other than t. Select two new variables, x', over $s_1$, and x", over $s_2$, distinct from $v_1,\ldots,v_m$. Now, consider the *pairing connection* $p(y\text{»}x',x'')$ established by formula $p_1(y)\approx_{s_1}x'\wedge p_2(y)\approx_{s_2}x''$. When one has several variables $y_1,\ldots,y_k$ over sort t, one uses a *conjunct connection* $p(y_1\text{»}x_1',x_1'')\wedge\ldots\wedge p(y_k\text{»}x_k',x_k'')$. Under such connection, we can eliminate sort t and projections $p_1$ and $p_2$ from formulae of L: we have eliminability under connection.

**Proposition** Eliminability of product sort under connection

Given a language L extending $L^{\times}$, consider the sub-language $L_{\times} := L\text{-}L^{\times}$ of L obtained by removal of sort t and operations $p_1$ and $p_2$.

Given a set of variables $y_1,\dots,y_k$, over t, and $v_1,\dots,v_m$, over sorts other than t; select, for each variable $y_i$, two new variables, $x_i'$, over $s_1$, and $x_i''$, over $s_2$, distinct from $v_1,\dots,v_m$, and consider the pairing connection $p(y_1 \gg x_1',x_1'',\dots,y_k \gg x_k',x_k'')$ established by conjunction $[p_1(y_1) \approx_{s_1} x_1' \wedge p_2(y_1) \approx_{s_2} x_1''] \wedge \dots \wedge [p_1(y_k) \approx_{s_1} x_k' \wedge p_2(y_k) \approx_{s_2} x_k'']$. Then, for each formula $\psi$ of L, whose free variables are among $y_1,\dots,y_k,v_1,\dots,v_m$, there exists a formula $\psi_\times$ of $L_\times$ (whose free variables are among $x_1',x_1'',\dots,x_k',x_k'',v_1,\dots,v_m$) such that the formula, with implicit universal quantification, $p(y_1 \gg x_1',x_1'',\dots,y_k \gg x_k',x_k'') \rightarrow [\psi \leftrightarrow \psi_\times]$ is a consequence of $Prod(t \backslash p_1 \rightarrow s_1, p_2 \rightarrow s_2)$.

**Proof outline.**

We proceed by induction on the structure of formula $\psi$ of L.

An atomic formula of L is either an equality between variables of sort t, or else the result $\alpha$ of replacing, in an atomic formula $\alpha'$ of $L_\times$, occurrences of variables x', over $s_1$, and x'', over $s_2$, by $p_1(y)$ or $p_2(y)$.

Setting $\alpha_\times := \alpha'$, we have $p_1(y) \approx_{s_1} x' \wedge p_2(y) \approx_{s_2} x'' \models [\alpha \leftrightarrow \alpha_\times]$.

Quantifying over sorts other than t causes no problem.

For $\psi$ of the form $(\forall y{:}t)\theta$ we set $\psi_\times := (\forall x'{:}s_1)(\forall x''{:}s_2)\theta_\times$. Then, since the projections are jointly bijective, we apply the inductive hypothesis.

*QED*

This proposition can be read as: any relation definable in language L corresponds to a relation definable in the sub-language $L_\times = L - \{t,p_1,p_2\}$ of L obtained by removal of sort t and projections $p_1$ and $p_2$. In particular, for each formula $\theta$ of L, with no variable ranging over sort t, there exists a formula $\theta_\times$ of $L_\times$ called its *paired version*, such that $\times(t \backslash p_1 \rightarrow s_1, p_2 \rightarrow s_2) \models [\theta \leftrightarrow \theta_\times]$.

Now, consider a specification P'=<L',G'> where language L' has
sorts $s_1$ and $s_2$,
but neither sort t nor operations $p_1$ or $p_2$.

Then, specification P''=<L'',G''> with
language L'' := $L' \cup \{t, p_1(t) \rightarrow s_1, p_2(t) \rightarrow s_2\}$ being the extension of L' by the addition of sort t and operation symbols $p_1$, from sort t to $s_1$, and $p_2$, from sort t to $s_2$, and
axiomatisation $G'' = G' \cup \{\times(t \backslash p_1 \rightarrow s_1, p_2 \rightarrow s_2)\}$ the extension of G' by the explication of sort t as the product of sorts $s_1$ and $s_2$, with projections

$p_1(t) \rightarrow s_1$ and $p_2(t) \rightarrow s_2$,
is called the *extension of* $P' = \langle L', G' \rangle$ *by introduction of sort* t as the *product of sorts* $s_1$ and $s_2$, with *projections* $p_1(t) \rightarrow s_1$ and $p_2(t) \rightarrow s_2$, and is denoted by SR_PROD$[t \backslash p_1 \rightarrow s_1, p_2 \rightarrow s_2](P')$.

Extensions by introduction of a sort as product of existing ones are clearly conservative. They are convenient, but dispensable in the sense that a relation r, involving a product sort, can be replaced by a relation $r_\times$, <u>not</u> involving the product sort, which does the same job. This situation is similar to the case of lt and le, mentioned in 3.1: predicates r and $r_\times$ are interdefinable in the context of the product explication, as illustrated in figure 3.7. We shall comment some more on this point later on in 3.8.6.

$$\begin{tikzcd}
P' \arrow[r, "t=s1\times s2"] \arrow[d, equal] & P'' \arrow[r, "add\ r"] & P''+r \arrow[r, "define\ r_\times/r"] & (P''+r)+r_\times \arrow[d, Leftrightarrow] \\
P' \arrow[r, "add\ r_\times"] & P^* \arrow[r, "t=s1\times s2"] & P^*+prd \arrow[r, "define\ r/r_\times"] & (P^*+prd)+r
\end{tikzcd}$$

Fig. 3.7: Dispensing with product sort.

The point is that extensions by product sort behave very much like classical extensions by explicit definition of predicates. The situation is reminiscent of that of matrices vs. systems of linear equations: the former provide conceptual convenience, but can in principle be dispensed with in favour of the latter.

### 3.8.2 Extensions by sum of sorts

Some programming languages provide a construct like "variant record". Since records are already handled by the product construct, we can concentrate on the "variant" part. This is the idea of sum, or discriminated union: having a sort with objects of diverse natures. This may be useful, for instance, in representing graphs: one may use distinct representations for sparse and non-sparse graphs.

The sort construct *sum* (or *discriminated union*) introduces a new sort consisting of alternatives - objects of either one of the given sorts together - with the corresponding constructors. Since the elements of this new set need not be represented in the same way as their originators, one also introduces conversions. Thus, its realisations have a new set $T=S_1+S_2$ (disjoint union), together with its two insertions $i_k : S_k \rightarrow T$.

Consider language $L^+$ consisting of
sorts t, $s_1$ and $s_2$, as well as
operation symbols $i_1$, from sort $s_1$ to t, and $i_2$, from $s_2$ to t.

We call this language $L^+$ the *language for for sort* t *as sum of sorts* $s_1$ and $s_2$.

An *explication of sort* t as the *sum of sorts* $s_1$ and $s_2$ is the set $+(t\backslash s_1 \Rightarrow i_1 | s_2 \Rightarrow i_2)$ consisting of the following four sentences of $L^+$:

$(\forall y{:}t)[(\exists x_1{:}s_1) y \approx_t i_1(x_1) \vee (\exists x_2{:}s_2) y \approx_t i_2(x_2)]$ (ijs)

$(\forall x_1{:}s_1)(\forall x_2{:}s_2) \neg i_1(x_1) \approx_t i_2(x_2)$ (idi)

$(\forall x_1,u_1{:}s_1)[i_1(x_1) \approx_t i_1(u_1) \rightarrow x_1 \approx_{s_1} u_1]$ (ii1)

$(\forall x_2,u_2{:}s_2)[i_2(x_2) \approx_t i_2(u_2) \rightarrow x_2 \approx_{s_2} u_2]$ (ii2)

Sentence (ijs) states that the images of the insertions cover the new sort, whereas sentence (idi) states that these images are disjoint. Sentences (ii1) and (ii2) express the injectivity of each insertion. Specification $Sum(t\backslash s_1 \Rightarrow i_1 | s_2 \Rightarrow i_2) := \langle L^+, +(t\backslash s_1 \Rightarrow i_1 | s_2 \Rightarrow i_2) \rangle$ is called the *explication of sort* t as the *sum of sorts* $s_1$ and $s_2$ with *insertions* $i_1(s_1) \rightarrow t$ and $i_2(s_2) \rightarrow t$.

The reason for the name of specification $Sum(t\backslash s_1 \Rightarrow i_1 | s_2 \Rightarrow i_2) = \langle L, +(t\backslash s_1 \Rightarrow i_1 | s_2 \Rightarrow i_2) \rangle$ stems from its model-oriented properties.. Clearly, $Sum(t\backslash s_1 \Rightarrow i_1 | s_2 \Rightarrow i_2)$ has models $\mathfrak{A}$ such that $\mathfrak{A}[t] = \mathfrak{A}[s_1] + \mathfrak{A}[s_2]$; such models, where sort t is realised as the disjoint union of $\mathfrak{A}[s_1]$ and $\mathfrak{A}[s_2]$, may be called *canonical models*. Now, given a structure $\mathfrak{M}$ for language $L^+$, $\mathfrak{M} \in Mod[Sum(t\backslash s_1 \Rightarrow i_1 | s_2 \Rightarrow i_2)]$ iff $\mathfrak{M}$ is isomorphic to some canonical model $\mathfrak{A}$. Thus, for $\mathfrak{M} \in Mod[Sum(t\backslash s_1 \Rightarrow i_1 | s_2 \Rightarrow i_2)]$ the realisation $\mathfrak{M}[t]$ of sort t has indeed the behaviour of the discriminated union of $\mathfrak{M}[s_1]$ and $\mathfrak{M}[s_2]$.

Now, consider a specification $P' = \langle L', G' \rangle$ where language L' has
sorts $s_1$ and $s_2$,
but neither sort t nor operations $i_1$ or $i_2$.

Then, specification $P'' = \langle L'', G'' \rangle$ with
language $L'' := L' \cup \{t, i_1(s_1) \rightarrow t, i_2(s_2) \rightarrow t\}$ being the extension of L' by the addition of sort t and operation symbols $i_1$, from sort $s_1$ to t, and $i_2$, from sort $s_2$ to t,
axiomatisation $G'' = G' \cup \{+(t\backslash s_1 \Rightarrow i_1 | s_2 \Rightarrow i_2)\}$ the extension of G' by the explication of sort t as the sum of sorts $s_1$ and $s_2$, with insertions $i_1(s_1) \rightarrow t$ and $i_2(s_2) \rightarrow t$,

is called the *extension of* $P' = \langle L', G' \rangle$ *by introduction of sort* t as the *sum of sorts* $s_1$ and $s_2$, with *insertions* $i_1(s_1) \rightarrow t$ and $i_2(s_2) \rightarrow t$, and is denoted by $SR\_SUM[t\backslash s_1 \Rightarrow i_1 | s_2 \Rightarrow i_2](P')$.

Extensions by introduction of a sort as sum of existing sorts are clearly conservative. Also, in a realisation an object a of sort T can be regarded as an alternative: either $a_1$ of sort $S_1$, or $a_2$ of sort $S_2$. So, a unary relation R over T can be viewed as a (disjoint) union $R_1+R_2$ of two relations $R_1$, over $S_1$, and $R_2$, over $S_2$. In general, a relation R, involving sort T, can be regarded as a (disjoint) union of relations, its 'alternant' versions, each one involving sort $S_1$ or $S_2$ where R involved T.

The preceding argument indicates that a relation is definable in language L" iff it is definable in the sub-language L'. Extensions by sum of sorts have a kind of eliminability under conversion, slightly more complex than in the case of product. For a formula $\theta$ of L", with no variable ranging over sort t, there exists a formula $\theta_\times$ of L' called its *alternant version*, such that $+(t;s_1{:}i_1|s_2{:}i_2)\vDash[\theta\leftrightarrow\theta_\times]$.

Thus, extensions by sum sort behave very much like classical extensions by explicit definition of predicates. They are, like products, a dispensable convenience, from the purely logical viewpoint.

### 3.8.3 Extensions by subsorts

For many purposes it is convenient to deal with a subsort, rather than a full sort. For instance, in talking about division, the non-zero naturals are the ones to be safely used as denominators.

The sort construct *subsort* introduces a new sort consisting of the objects of the given sort S satisfying the relativisation predicate r, in the spirit of the set-theoretical specification axiom. Thus, its realisations have a new set $T=\{a\in S{:}r(a)\}$, together with its the insertion $j{:}T\rightarrow S$ (since the elements of this new set T need not be represented in the same way as those of S).

Consider language $L^{\subseteq}$ consisting of
sorts t and s, as well as
a unary predicate symbol r over sort s, and
a unary operation symbol j from sort t to s.

We call this language $L^{\subseteq}$ the *language for for sort* t *as* the *subsort of sort* s *relativised to* r.

An *explication of sort* t as the *subsort of sort* s *relativised to* r (with *insertion* $j(t)\rightarrow s$) is the set $\subseteq(t\backslash s{:}r,j)$ consisting of the following two sentences of $L^{\subseteq}$:

$(\forall x{:}s)[r(x)\leftrightarrow(\exists y{:}t)x\approx_s j(y)]$ (jr)

$(\forall y,y'{:}t)[j(y)\approx_s j(y')\rightarrow y\approx_t y']$ (ij)

Sentence (jr) states that the image of operation j is (the extension of) the relativisation predicate (thus defining r in terms of j), whereas sentence (ij) expresses the injectivity of operation j qualifying it as an insertion. Specification Sbst(t\s:r,j) := $<L^{\subseteq},\{(jr),(ij)\}>$ is called the *explication of sort* t as the *subsort of sort* s *relativised to relativisation predicate* r, with *insertion* $j(t)\rightarrow s$.

The reason for the name of specification Sbst(t\s:r,j) := $<L^{\subseteq},\subseteq(t\backslash s{:}r,j)>$ comes from its model-oriented properties. Clearly, Sbst(t\s:r,j) has models $\mathfrak{S}$ such that $\mathfrak{S}[t] = \{a\in \mathfrak{S}[s]{:}\mathfrak{S}\models r(x)\ [a]\}$; such models, where sort t is realised as the set of the objects of the given sort s satisfying the relativisation predicate r, may be called *canonical models*. Also, given a structure $\mathfrak{M}$ for language $L^{\subseteq}$, $\mathfrak{M}\in$ Mod[Sbst(t\s:r,j)] iff $\mathfrak{M}$ is isomorphic to some canonical model $\mathfrak{S}$. Thus, for $\mathfrak{M}\in$ Mod[Sbst(t\s:r,j)] the realisation $\mathfrak{M}[t]$ of sort t has indeed the desired behaviour, since it can be regarded as consisting of the objects of the given sort s satisfying the relativisation predicate r.

Consider the sub-language $L_{\top}{:=}L^{\subseteq}\text{-}\{t,j\}$ obtained by removing from $L^{\subseteq}$ sort t and operation j, thus consisting only of sort s and unary predicate r over s.. Now, a consequence of the reflexivity of equality is $(\exists y{:}t)y\approx_t y$ (model-theoretically $\mathfrak{M}[t]\neq\emptyset$). Thus $\{(jr)\}\models(\exists x{:}s)r(x)$. Notice that the latter is a sentence $\nu(r)$ of $L_{\subseteq}$ expressing that (the extension of) the predicate r must be nonempty ($\mathfrak{M}[r]\neq\emptyset$). We shall call this sentence $\nu(r)$ of $L_{\subseteq}$ the *non-voidness requirement for relativisation predicate* r.

Now, consider a language L' extending $L_{\top}$, and assume that sort t and operation j are not in L' We shall say that $\subseteq(t\backslash s{:}r,j)$ is *applicable* to a specification P'=<L',G'> iff the non-voidness requirement $\nu(r)$ is in Cn[P']. In such case, we shall call specification $P''=<L'\cup\{t,j(t)\rightarrow s\},G'\cup\{\subseteq(t\backslash s{:}r,j)\}>$ the *extension* of P'=<L',G'> *by introduction of sort* t as the *subsort of sort* s under *relativisation predicate* r (with *insertion* $j(t)\rightarrow s$) - and denote it by SR_SBST[t\s:r,j](P').

The reason for imposing the above requirement is that it is a necessary condition for conservativeness of the extension. That it is sufficient as well can be seen because of the following characterising property.

**Proposition** Characterisation of models of extension by subsort

Let $P''=<L'',G'\cup\{\subseteq(t\backslash s{:}r,j)\}>$ be the extension of P'=<L',G'> by introduction of sort t as the subsort of sort s under relativisation predicate r. Given a structure $\mathfrak{M}$" for language L", consider its reduct $\mathfrak{M}$' to language L'. Then $\mathfrak{M}$"$\in$Mod[P"] iff $\mathfrak{M}$'$\in$Mod[P'] and, up to isomorphism, $\mathfrak{M}$"[t] = $\{a\in \mathfrak{M}'[s]:\mathfrak{M}'\models r(x)[a]\}$.

Thus the realisation of the new sort t has indeed the desired behaviour. We also have expandability: each model $\mathfrak{M}\in$ Mod(P') has an expansion $\mathfrak{M}^{\subseteq}\in$ Mod[P"]. Moreover, in a realisation, a relation M involving sort T, can be replaced by a relation $M_{\subseteq}$, its 'relativisation', doing the same job.

So, we have eliminability under connection, as in the case of product. The idea is similar: we use a connection for translating variables ranging over t as variables over s. Consider variables y, over t, and $v_1,\dots,v_m$, over sorts other than t. Select a new variable x over s, distinct from $v_1,\dots,v_m$. Now, consider the *insertion connection* j(y»x) established by formula $x\approx_s j(y)$. When one has several variables $y_1,\dots,y_k$ over sort t, one uses a *conjunct connection* $j(y_1»x_1)\wedge\dots\wedge j(y_k»x_k,)$. Under such connection, we can eliminate sort t and insertion j from formulae of L".

**Proposition** Eliminability of subsort under connection

Let P"=<L",G'$\cup\{\subseteq(t\backslash s:r,j)\}$> be the extension of P'=<L',G'> by sort t as the subsort of sort s under relativisation predicate r (with insertion $j(t)\rightarrow s$).

Given a set of variables $y_1,\dots,y_k$, over t, and $v_1,\dots,v_m$, over sorts other than t; select, for each variable $y_i$, a new variable $x_i$, over s, distinct from $v_1,\dots,v_m$, and consider the insertion connection $j(y_1»x_1,,\dots,y_k»x_k)$ established by conjunction $j(y_1)\approx_s x_1\wedge\dots\wedge j(y_k)\approx_s x_k$. Then, for each formula $\psi$ of L", whose free variables are among $y_1,\dots,y_k,v_1,\dots,v_m$, there exists a formula $\psi_{\subseteq}$ of L' (whose free variables are among $x_1,\dots,x_k,v_1,\dots,v_m$) such that the formula, with implicit universal quantification, $j(y_1»x_1,\dots,y_k»x_k)\rightarrow[\psi\leftrightarrow\psi_{\subseteq}]$ is a consequence of P".

**Proof outline.**

We proceed by induction on the structure of formula $\psi$ of L".

An atomic formula of L" is either an equality between variables of sort t, or else the result $\alpha$ of replacing, in an atomic formula $\alpha$' of L', occurrences of variables x, over s, by j(y).

Setting $\alpha_{\subseteq}:=\alpha'$, we have $x\approx_s j(y)\models[\alpha\leftrightarrow\alpha_{\subseteq}]$.

Quantifying over sorts other than t causes no problem.

For $\psi$ of the form $(\forall y{:}t)\theta$ we set $\psi_{\subseteq}:=(\forall x{:}s)[r(x)\rightarrow\theta_{\subseteq}]$. Then, since the insertion is bijective onto r, we apply the inductive hypothesis.

*QED*

So, a relation definable in language L' corresponds to a relation definable in the sub-language L". In particular, for each formula $\theta$ of L", with no variable ranging over sort t, there exists a formula $\theta_{\subseteq}$ of L', called its *relativised version*, such that $\subseteq(t\backslash s{:}r,j) \models [\theta \leftrightarrow \theta_{\subseteq}]$.

Thus, extensions by subsort, being conservative under requirement and eliminable, are very much like classical extensions by explicit definition of operations.

### 3.8.4 Extensions by quotient of sorts

In some cases it is convenient to identify objects of a given sort. For instance, in specifying the (signed) integers by extending the naturals by a product sort Bool×Nat, one may wish to identify minus zero with plus zero, i. e. ‹fl,zero› with ‹tr,zero›.

The sort construct *quotient* introduces a new sort consisting of the equivalence classes of the objects of the given sort S under a given equivalence relation q over it, a usual set-theoretical construction. Thus, its realisations have a new set $T=\{[a/q]:a\in S\}$, together with its canonical projection $p:S\rightarrow T$ mapping each element a of S to its q-class [a/q].

Consider language $L^{/}$ consisting of
sorts t and s, as well as
a binary predicate symbol q over sort s, and
a unary operation symbol p from sort s to t.

We call this language $L^{/}$ the *language for for sort* t *as quotient of sort* s under *predicate* q.

An *explication of sort* t as the *quotient of sort* s under *equivalence predicate* q (with *canonical projection* $p(s)\rightarrow t$) is the set $/(t\backslash s/q,p)$ consisting of the following two sentences of $L^{/}$:

$(\forall y{:}t)(\exists x{:}s)y\approx_t p(x)$ (sp)

$(\forall x,x'{:}s)[p(x)\approx_t p(x')\leftrightarrow q(x,x')]$ (pq)

Sentence (sp) expresses the surjectivity of operation p, whereas sentence (pq) states that the q is the kernel of operation p (thus defining q in terms of p). Specification $Quot(t\backslash s/q,p) := \langle L^{/},\{(sp),(pq)\}\rangle$ is called the *explication of sort* t as the *quotient of sort* s under *equivalence predicate* q with *canonical projection* $p(s)\rightarrow t$.

As before, the reason for the name of specification

$\mathrm{Quot}(t\backslash s/q,p):=<L^{/},/(t\backslash s/q,p)>$ stems from properties of its models. For, $\mathrm{Quot}(t\backslash s/q,p):=<L^{/},/(t\backslash s/q,p)>$ has models $\mathfrak{Q}$ such that $\mathfrak{Q}[t] = \{[a/\mathfrak{Q}[q]:a\in\mathfrak{Q}[s]\}$; such models, where sort t consists of the q-classes of the elements of sort s under equivalence relation $\mathfrak{M}[q]$, may be called *canonical models.* Also, given a structure $\mathfrak{M}$ for language $L^{/}$, $\mathfrak{M}\in \mathrm{Mod}[\mathrm{Quot}(t\backslash s/q,p)]$ iff $\mathfrak{M}$ is isomorphic to some canonical model $\mathfrak{Q}$. Thus, for $\mathfrak{M}\in \mathrm{Mod}[\mathrm{Quot}(t\backslash s/q,p)]$ the realisation $\mathfrak{M}[t]$ of sort t has indeed the desired behaviour, since it can be regarded as consisting of the q-classes of the objects of sort s under equivalence relation $\mathfrak{M}[q]$.

Indeed, consider the sub-language $L_q:=L^{/}-\{t,p\}$ obtained by removing from $L^{/}$ sort t and operation p, thus consisting only of sort s and binary predicate q over s.. In view of the equality axioms, some consequences (pq) are $(\forall x:s)q(x,x)$, $(\forall x,x':s)[q(x,x')\to q(x',x)]$ and $(\forall x,x',x'':s)[(q(x,x')\wedge q(x',x'')\to q(x,x'')]$. Notice that these are sentences of $L_q$ expressing that (the extension of) the predicate q must be an equivalence relation. We shall call their conjunction $\varepsilon(q)$ the *equivalence requirement for predicate* q.

Now, consider a language L' extending $L_q$, and assume that sort t and operation p are not in L' We shall say that $/(t\backslash s/q,p)$ is *applicable* to a specification $P'=<L',G'>$ iff the non-equivalence requirement $\varepsilon(q)$ is in Cn[P']. In such case, we shall call specification $P''=<L'\cup\{s,p(s)\to t\},G'\cup\{/(t\backslash s/q,p)\}>$ the *extension* of $P'=<L',G'>$ *by introduction of sort* t as the *quotient of sort* s under *equivalence predicate* q (with *canonical projection* $p(s)\to t$) - and denote it by SR_QUOT[t\s/q,p](P') .

The above requirement is imposed because it is a necessary condition for conservativeness of the extension. That it is sufficient as well follows from the following characterising property.

**Proposition** Characterisation of models of extension by quotient

Let $P''=<L'',G'\cup\{/(t\backslash s/q,p)\}>$ be the extension of $P'=<L',G'>$ by sort t as the quotient of sort s under equivalence predicate q. Given a structure $\mathfrak{M}''$ for language L'', consider its reduct $\mathfrak{M}'$ to language L'. Then $\mathfrak{M}''\in \mathrm{Mod}[P'']$ iff $\mathfrak{M}'\in \mathrm{Mod}[P']$ and, up to isomorphism, $\mathfrak{M}''[t] = \{[a/\mathfrak{M}'[q]:a\in\mathfrak{M}'[s]\}$.

This property also shows that the realisation of the new sort t indeed behaves as consisting of the equivalence classes under q of the objects of the given sort s.

We have eliminability under connection, as in the case of subsort. The idea is similar: we use a connection for translating variables ranging over t as variables over s. Consider variables y, over t, and $v_1,\dots,v_m$, over sorts other than t. Select a new variable x over s, distinct from $v_1,\dots,v_m$. Now, consider the *projection connection* $p(y»x)$ established by formula $y{\approx_t}p(x)$. When one has several variables $y_1,\dots,y_k$ over sort t, one uses a *conjunct connection* $p(y_1»x_1)\wedge\dots\wedge p(y_k»x_k,)$. Under such connection, we can eliminate sort t and projection p from formulae of L".

**Proposition** Eliminability of quotient sort under connection

Let P"=<L",G'$\cup$\{/(t\s/q,p)\}> be the extension of P'=<L',G'> by introduction of sort t as the quotient of sort s under equivalence predicate q (with canonical projection $p(s)\rightarrow t$).

Given a set of variables $y_1,\dots,y_k$, over t, and $v_1,\dots,v_m$, over sorts other than t; select, for each variable $y_i$, a new variable $x_i$, over s, distinct from $v_1,\dots,v_m$, and consider the projection connection $p(y_1»x_1,,\dots,y_k»x_k)$ established by conjunction $y_1{\approx_t}p(x_1)\wedge\dots\wedge y_k{\approx_t}p(x_k)$. Then, for each formula $\psi$ of L", whose free variables are among $y_1,\dots,y_k,v_1,\dots,v_m$, there exists a formula $\psi_/$ of L' (whose free variables are among $x_1,\dots,x_k,v_1,\dots,v_m$) such that the formula, with implicit universal quantification, $p(y_1»x_1,\dots,y_k»x_k)\rightarrow[\psi\leftrightarrow\psi_/]$ is a consequence of P".

**Proof outline.**

Clearly the terms of L" with sort in L' are terms of L'. Also, each term of L" with sort t is either a variable over sort t, or p(t) for some term t of L' with sort s. With this classification we can handle the three cases of new atomic formulae of L": $p(t){\approx_t}p(t')$, $y{\approx_t}p(t)$ and $y{\approx_t}y'$.

Quantifying over sorts other than t causes no problem.

For $\psi$ of the form $(\forall y{:}t)\theta$ we set $\psi_/ := (\forall x{:}s)\theta_/$, and use the inductive hypothesis together with $P"\models(\forall y{:}t)(\exists x{:}s)p(y{:}x)\wedge(\forall x{:}s)(\exists y{:}t)p(y{:}x)$.

*QED*

So, a relation definable in language L' corresponds to a relation definable in the sub-language L". In particular, for each formula $\theta$ of L", with no variable ranging over sort t, there exists a formula $\theta_/$ of L', its *projected version*, such that $/(t\backslash s/q,p)\models[\theta\leftrightarrow\theta_/,]$.

Thus, extensions by quotient sort, like those by subsort, are conservative under requirement and eliminable, much as classical extensions by explicit definition of operations.

### 3.8.5 Extensions by sort enumeration

In extending the naturals to the (signed) integers, we suggested in 3.8.1 that it was natural to consider a product sort Bool×Nat, where Bool stands for the two signs: plus and minus. Instead of adding the last comment, it may be better to use more mnemonic names for the signs, i. e. use Sign×Nat, where Sign={plus, minus}. Another example, along the same line is a sort day_of_week enumerated by its constants:Sun, Mon, Tue, Wed, Thu, Fri, Sat. This is the idea of introducing a (finite) sort by enumeration of its elements.

The sort construct *enumeration* introduces a new sort consisting of the objects named by the constants, a usual way to decribe a finite set. Thus, their realisations have a new set $T=\{a_1,\dots,a_k\}$.

Consider language $L^e$ consisting of
sort t, as well as
constants $c_1,\dots,c_k$ over sort t.

We call this language $L^e$ the *language for for sort* t *as* tthe *enumeration by constants* $c_1,\dots,c_k$.

An *explication of sort* t as the *enumeration by constants* $c_1,\dots,c_k$ is the set $=(t\backslash\{c_1,\dots,c_k\})$ consisting of the following two sentences of $L^e$:

$$[(\neg c_1 \approx_t c_2 \wedge \dots \wedge \neg c_1 \approx_t c_k) \wedge (\neg c_2 \approx_t c_3 \wedge \dots \wedge \neg c_2 \approx_t c_k) \wedge \dots \wedge \neg c_{k-1} \approx_t c_k] \qquad \text{(dc)}$$

$$(\forall y{:}t)[y \approx_t c_1 \vee \dots \vee y \approx_t c_k] \qquad \text{(ce)}$$

Sentence (dc) states that the constants are pairwise distinct, and (ce) that they exhaust sort t. Specification $\mathrm{Enum}(t\backslash\{c_1,\dots,c_k\}) := \langle L^e,\{(dc),(ce)\}\rangle$ is called the *explication of sort* t as the *enumeration by constants* $c_1,\dots,c_k$.

The reason for this name for specification $\mathrm{Enum}(t\backslash\{c_1,\dots,c_m\}) = \langle L^e,=(t\backslash\{c_1,\dots,c_k\})\rangle$ is clear: $\mathfrak{M}\in \mathrm{Mod}[\mathrm{Enum}(t\backslash\{c_1,\dots,c_k\})]$ iff $\mathfrak{M}[t] = \{\mathfrak{M}[c_1],\dots,\mathfrak{M}[c_k]\}$. Thus, the realisation $\mathfrak{M}[t]$ of sort t exhibits the desired behaviour.

Now, given a language L extending $L^e$, we may remove from L sort t and constants $c_1,\dots,c_k$, thereby obtaining its sub-language $L_e{:=}L-L^x$. We then have expandability: each structure $\mathfrak{M}$ for sub-language $L_e$ has a unique, up to isomorphism, expansion to a model $\mathfrak{M}^e\in \mathrm{Mod}[\mathrm{Enum}(t\backslash\{c_1,\dots,c_k\})]$.

Assume that sort t and constants $c_1,\dots,c_k$ are not in L', and consider the language extension $L''=L'\cup\{t,c_1,\dots,c_k{:}t\}$ of L' by the addition of sort t and constants $c_1,\dots,c_k$ over sort t. Now, given a specification $P'=\langle L',G'\rangle$, we call specification $P''=<L'\cup\{t,c_1,...,c_k:t\},G'\cup\{=(t\backslash\{c_1,...,c_k\})\}>$ the *extension* of $P'=<L',G'>$ *by introducton of sort* t as the *enumeration by constants* $c_1,...,c_k$ - and denote it by $SR\_ENUM[t\backslash\{c_1,...,c_k\}](P')$. Clearly, any such extension is conservative.

### 3.8.6 Comments on sort constructs

We have been examining five methods for extending a specification by sorts, so as to guarantee conservativeness. These sort introduction constructs are formalisations of counterparts found in many programming languages. We shall now examine some further aspects of these constructs, related to their use in program development.

A. *Some simple variations*

Let us first consider some simple variations and comparisons, mainly extensions of enumerations and iterated constructs.

We have considered four constructs for introducing a new sort connected to existing ones by means of conversions. As such, they can be given parameterised specifications (Meré and Veloso 1991). Two of them, product (introducing records) and sum or discriminated union (introducing alternatives) are similar to classical extensions by definition of predicate, in that they are conservative and eliminable. The other two, namely subsort and quotient are eliminable but have requirements for conservativeness, and resemble classical extensions by definition of operation.

Another construct that we have considered introduces a new sort by simply enumerating its elements. Let us take a closer look at enumeration.

First consider the case of introducing a sort with a single element. For $Enum(t\backslash\{c\})$ the two new axioms reduce to just one, namely $(\forall y:t)y\approx_t c$, which is the Skolemisation of the logically valid sentence $(\forall y:t)(\exists z:t)y\approx_t z$. Thus, $Enum(t\backslash\{c\})$ is equivalent to an extension by Skolem constant (see 3.7) of a symbol extension (see 3.4) by a new sort. This is the only enumeration needed, for we can reduce $Enum(t\backslash\{c_1,...,c_m\})$ to $Enum(t\backslash\{c\})$ by means of discriminated union: $Enum(t\backslash\{c_1,...,c_m\})$ is a restriction of the sum $Enum(t\backslash\{c_1\})+...+Enum(t\backslash\{c_m\})$.

In programming languages, the enumeration construct sometimes introduces an 'ordered enumeration', with a linear ordering $c_1<_t...<_t c_i<_t c_{i+1}<_t...<_t c_k$. One can obtain such ordered enumeration as an extension of $Enum(t\backslash\{c_1,...,c_k\})$ by the definition of predicate $<_t$ with the defining axiom

$(\forall y{:}t)(\forall y'{:}t)\{y<_t y' \leftrightarrow [(y\approx_t c_1 \wedge y'\approx_t c_2)\vee\ldots\vee(y\approx_t c_1\wedge y'\approx_t c_k)]\vee\ldots\vee[(y\approx_t c_{k-1}\wedge y'\approx_t c_k)]\}$
One can then, if so desired, introduce successor and predecessor operations by definition.

Let us now consider subsort and quotient, which bear some similarity with each other. In each case the conversion is introduced by relying on a given predicate. Given the extension, this predicate is related to the conversion via a definition: the relativisation predicate as the image of the insertion and the equivalence predicate as the kernel of the canonical projection. We shall shortly comment on more resemblances between them.

Consider now product and sum, which take two argument sorts. These binary constructs can be iterated to yield n-ary iterated versions. For instance, iterated product gives $T=S_1\times(S_2\times S_3)$, with typical element $a=\langle a_1,\langle a_2,a_3\rangle\rangle$ where $a_2=p_1(p_2(a))$ and $a_3=p_2(p_2(a))$. Alternatively, one can obtain flat n-fold versions. For product, one has $T=S_1\times S_2\times S_3$ with typical element $a=\langle a_1,a_2,a_3\rangle$ where $a_2=p_2(a)$ and $a_3=p_3(a)$. One can easily generalise the given specifications to n sorts: products with n projections $Prod(t\backslash p_1\rightarrow s_1,\ldots,p_n\rightarrow s_n)$ and sums with n insertions $Sum(t\backslash s_1\Rightarrow i_1|\ldots|s_2\Rightarrow i_2)$. Of course, these n-fold versions are conservative as well as eliminable under connections.

B. *New conversion operations*

Now, let us examine the appropriateness of these sort constructs for program development. We shall be mostly concerned with their repertoires of conversion operations.

We have mentioned at the end of 3.8.1 that extensions by introduction of a sort as product of existing sorts are convenient, but dispensable. Let us now elaborate on this point (see figure 3.7). Assume that we have extended P'=<L',G'> to P"=<L",G"> by defining the new sort t as the product of sorts $s_1$ and $s_2$, already in L', with projections $p_1$ and $p_2$. We now wish to further extend P" by adding some new symbols. For definiteness, let us say that we wish to introduce a new predicate symbol r involving the new sort t. So, we have specification P=<L,G> extending P"=<L",G">. Instead of introducing this predicate, we can introduce predicate $r_x$, its profile being the result of replacing, in the profile of r, occurrences of t by $s_1,s_2$. We can obtain an extension $P^*=<L^*,G^*>$ extending P'=<L',G'> by predicate symbol $r_x$, so without product sort, which does the same job as P=<L,G>. For, predicates r and $r_x$ are interdefinable in the context of the product sort. A similar situation occurs with sum of existing sorts, because a sum sort adds no really new definable relations: a relation R, involving a sum sort, can be regarded as a (disjoint) union of relations, not involving it, as argued in 3.8.2.

The preceding arguments indicate that product and sum are dispensable as supporting sorts for new relations. Let us now examine the analogous situation for operations and more specifically how sort constructs are employed in program development.

The usage of sort constructs in programming is related to the idea of program development methods on the basis of data structures, such as Jackson's method (Jackson 1980). The analysis of this usage will suggest the convenience of having some extra - inverse-like conversion operations - in our four parameterised sort constructs

Let us start with the case of product, which introduces records as ordered pairs. Immediately upon introducing this new sort, one has no operations, besides the projections, involving it. Consider a procedure; its input parameters have to be processed and an output value is to be produced as a result.

If some of its input parameters ranges over a product sort t, no operation is supplied for this. Indeed, the very purpose of the projections is decomposing an object of the product sort t into its components, which can be processed by the operations available on the component sorts $s_1$ and $s_2$. Thus, projections $p_1$ and $p_2$ enable one to define an operation f' with argument in a product sort t on the basis of available operations with arguments in the component sorts $s_1$ and $s_2$, as illustrated in figure 3.8.

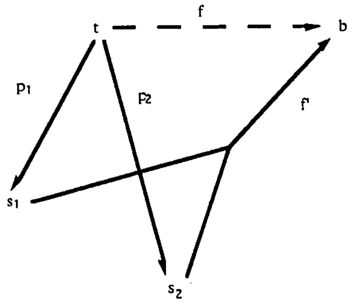

Fig. 3.8: Operation with argument in product sort.

Consider now the case where the output value of the procedure ranges over a product sort t. Assuming that the components of this output value have been obtained, one has to form a record from these components. So, one needs a binary operation from $s_1$ and $s_2$ into product sort t. In other words, with such an operation $m(s_1,s_2) \rightarrow t$, one can define an operation g with result in a product sort t on the basis of available operations $g_1$ and $g_2$ with results in the component sorts $s_1$ and $s_2$, as illustrated in figure 3.9.

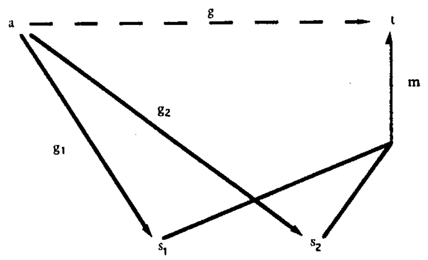

Fig. 3.9: Operation with result in product sort.

Our specification $\mathrm{Prod}(t\backslash p_1\rightarrow s_1,p_2\rightarrow s_2)$ in 3.8.1, for the explication of sort t as the product of sorts $s_1$ and $s_2$, does not provide such an operation $m(s_1,s_2)\longrightarrow t$. But, thanks to axioms (pjs) and (pji), it can easily be introduced by the following defining axiom

$$(\forall x_1{:}s_1)(\forall x_2{:}s_2)(\forall y{:}t)[m(x_1,x_2)\approx_t y\leftrightarrow(p_1(y)\approx_{s_1}x_1\wedge p_2(y)\approx_{s_2}x_2)].$$

We thus obtain a specification $\mathrm{Prod\_cnv}(t\backslash p_1\rightarrow s_1,p_2\rightarrow s_2,(s_1,s_2)\longrightarrow m)$ for product with conversions as an extension by definition of $\mathrm{Prod}(t\backslash p_1\rightarrow s_1,p_2\rightarrow s_2)$.

Consider now the case of quotient. We already have the canonical projection $p(s)\longrightarrow t$, which enables us to produce an operation g' with result in the quotient sort t from an available operation g with result in the old sort s. For likewise producing an operation f' with argument in the quotient sort t from an available operation f with argument in sort s, we would need a conversion $j(t)\longrightarrow s$, which chooses representatives. These situations are illustrated in figure 3.10.

$$\begin{tikzcd}
a \arrow[r, "g"] \arrow[d, equal] \arrow[dd, phantom, ""] & s \arrow[r, "p"] & t \arrow[d, equal] & t \arrow[r, "j"] \arrow[d, equal] & s \arrow[r, "f"] & b \arrow[d, equal] \\
a \arrow[rr, "g'"] & & t & t \arrow[rr, "f'"] & & b
\end{tikzcd}$$

Fig. 3.10: Operations with result and argument in quotient sort.

This situation will be clarified by examining the dual case: that of subsort. We already have a insertion $j(t)\longrightarrow s$ and we need a conversion $p(s)\longrightarrow t$, in the opposite direction. We can introduce it by the new axiom $(\forall y{:}t)p(j(y))\approx_t y$. Although it is <u>not</u> a definition, it is a conservative extension, being equivalent to one by pre and post conditions (see 3.6.2). It guarantees that the new conversion p behaves as inverse to the insertion j over the (extension of the) relativisation predicate, which is enough: its behaviour outside this region is irrelevant.

Now consider a sum t of sorts $s_1$ and $s_2$. By means of the insertions $i_1$ and $i_2$, a pair of operations $g_1$ and $g_2$, with results in the old sorts $s_1$ and $s_2$, generates an operation g with result in the sum sort t. For likewise defining operations with arguments in a sum sort t one would resort to a case-like akin to variant record. For this, one needs operations $a_1(t)\rightarrow s_1$ and $a_2(t)\rightarrow s_2$, for selecting alternatives "depending where the object came from". One can conservatively introduce them by $(\forall x_k{:}s_k)a_k(i_k(x_k)\approx_{s_k}x_k$.

These new constructs with (programming-motivated) conversions are conservative extensions of the simpler constructs for sort introduction. With these new conversions, subsort and quotient become very similar, having the same repertoire of operations with equivalent definitional extensions. This duality is akin to that between equivalence relations and partitions.

C. *Other sort constructs*

Finally, let us briefly mention other constructs for sort introduction. For the needs of implementation it may convenient to have additional sort constructs, as in some programming languages.

A property shared by the constructs we have examined is that of producing only finite sorts (from finite sorts). In particular, enumeration generates finite sorts with elements named by the constants provided.

Some programming languages provide additional constructs for sort introduction. Some such constructs, like powerset and mapping (similar to "array"), still preserve finiteness, but others, like sequences, generate (denumerably) infinite sorts from finite sorts. We defer these constructs to the section on parameterised specifications.

## 3.9 Applications of extensions

We shall now examine some applications illustrating the use of extensions in development of specifications. The first one, more general, will address the issues related to errors. The second one will consist of case studies showing the construction of extensions of integers to naturals and rationals. These case studies will illustrate the use of extensions for building specifications, hinting at some desirable features of systems providing support for such task.

### 3.9.1 Treatment of errors

We shall now examine some issues concerning the treatment of errors, mainly error introduction and propagation, in formal specifications. These issues are motivated by programming considerations.

We have not been paying much attention to the question of errors. This is due to our emphasis on 'liberal' specifications (see 1.6, 2.6 and 2.7). But, when one thinks of an abstraction of a data structure, some questions, which we dismissed in 2.6 as semantical, model-oriented ones, concerning empty structures crop up: what is the top of an empty stack, what is the root of a null tree? Since such objects are usually in a parameter sort, there seems to be no natural element to be returned. A usual solution amounts to returning an error constant (see 2.6). Now, we shall take a closer look at these questions.

The pragmatical reason for the lack of emphasis on errors is based on the following idea: a correct, well done program never asks for the top of a stack without first testing whether it is empty (see 2.6). This appears reasonable, once one has a good program on hand. But, and during its development and "tuning"? In these phases error messages may be helpful. In addition, in the case of 'public' specifications in a library, it may be interesting to warn the user against potentially dangerous situations. We shall now briefly examine some aspects concerning specifications with errors.

Consider stacks of elements. A formal specification P=<L,G> for its error-free verson was given in 2.9 as Spec. 2.3: STACK[ELEMENT]. We now wish to introduce a new constant (nullary operation) symbol in sort Elm to denote the top of the empty stack. This may be regarded as an extension of P=<L,G> to P'=<L',G'>. We obtain it by adding the new constant err_elm of sort Elm together with the axiom $\mathrm{top(crt)} \approx_{\mathrm{Elm}} \mathrm{err\_elm}$. This new axiom is (equivalent to) a definition of err_elm. Hence, the extension $P \subseteq P'$ is expansive and conservative (see 3.7 and 3.6.1).

We now have an error constant for designating the top of the empty stack. But, note that this new constant err_elm is of sort Elm; hence it may be pushed onto a stack. In other words, language L' has new terms, such as push(u, err_elm), of sort Stk, that are not in L. If we ask for the the values of these new terms, it seems natural to say that one has an error situation: only it is a error concerning stacks rather than one concerning elements as before. So, let us introduce a new constant symbol err_stk to denote the stack obtained from the empty one by pushing err_elm. Once more, this is an extension from P'=<L',G'> to P"=<L",G">, with $L''=L' \cup \{\mathrm{err\_stk:Stk}\}$ the extension of L' by the addition of constant err_stk of sort Stk and $G''=G' \cup \{\mathrm{push(crt,err\_elm)} \approx_{\mathrm{Stk}} \mathrm{err\_stk}\}$. Notice that term push(crt,top(crt)) was already in language L. In going from P to P' we provided a new name, namely push(cr ,err_elm ) for it; we now have another new name for it in L":

err_stk. Once again, we have an extension $P' \subseteq P''$ by definition of a constant, hence expansive and conservative.

The extension from P' to P" dealt with part of the question of error propagation, by introducing a constant symbol err_stk to denote the result of pushing err_elm onto the empty stack. We have not yet explicitly stated what happens when one pushes err_elm onto other stacks. Again, it seems natural to say that such actions result in erroneous stacks. The idea that pushing an err_elm results in an erroneous stack can be captured by an error propagation axiom like $(\forall v{:}Stk)push(v,err\_elm) \approx_{Stk} err\_stk\_pae$. Let us call this extension $P^*$. The error propagation axiom is the Skolemisation of the sentence $(\exists x{:}Elm)(\exists u{:}Stk)(\forall v{:}Stk)push(v,x) \approx_{Stk} u$ of L. The lattter sentence is a consequence of $P^*$ but <u>not</u> of P. Thus, the extension $P \subseteq P^*$ is <u>not</u> conservative, and neither are $P' \subseteq P^*$ or $P'' \subseteq P^*$. A similar problem would arise if we identified err_stk_pae with err_stk or if wished to introduce an error constant for pushing any element onto err_stk.

In general, an extension by error propagation axioms is not conservative. One way to recover conservativeness amounts to relativising the old axioms to error-free situations. This appears to be the idea behind the so-called "OK equations" (Goguen *et al.* 1978). A disadvantage of this approach is its poor legibility: it clutters up the "normal" axioms, expressing the FIFO behaviour of stacks, because of the possible anomalous behaviour.

Also, notice that error propagation is not so important from the standpoint of program development. For, one is not so interested in the behaviour of a program after entering an error state which it cannot leave. In such situations one generally expects the system to take over, either aborting program execution or invoking a special routine (see 2.6).

The problem appears to reside not so much in error propagation per se, but rather in insisting on error constants. From the programming veiwpoint this gives rise to 'sink' states. From the specification standpoint one sacrifices conservativeness or legibility.

One might consider using error operations, rather than constants, which appears more in line with error handling and recovery. The very notation err_stk_pae, with 'a' for "arbitrary", suggests replacing its axiom by one like $(\forall v{:}Stk)push(v,err\_elm) \approx_{Stk} err\_stk\_pae(v)$, where err_stk_pae(v) is now an operation from Stk to Stk that names the erroneous stacks tagged with the cause for error.

A version of this suggestion, more tuned with the 'liberal' spirit, uses error predicates in lieu of constants or operatons. We would replace constant err_elm by predicate is_err_elm and its axiom top(crt)$\approx_{Elm}$err_elm by the more liberal, inner constraint, version is_err_elm(top(crt)). Similarly, we could use predicate is_err_stk for errors of sort Stk. The axiom push(crt,err_elm)$\approx_{Stk}$err_stk of constant err_stk would be replaced by is_err_stk(push(crt,err_elm)) and error propagation for push would be expressed by an axiom like ($\forall$u:Stk)($\forall$x:Elm)[(is_err_stk(u)$\vee$is_err_elm(x))$\rightarrow$is_err_stk(push(u,x))], with ($\forall$u:Stk)[is_err_stk(u)$\rightarrow$is_err_elm(top(u)] for top. Thus such error predicate are like (simultaneously) inductive predicates (see 3.5.3)

### 3.9.2 Case studies

We shall now present two case studies showing construction of extensions of integers to naturals and rationals, which illustrate the application of our extension methods for constructing specifications.

A data type for integers is built into many programming languages. We shall now indicate how to construct specifications for representing the naturals and the rationals, the latter corresponding to the data type reals, also built into several programming languages. We shall examine two case studies showing how to use our extension constructs, mainly sort constructs,

to express the naturals as the subsort of non-negative integers, and

to mimic the usual construction of the rationals as equivalence classes of fractions with integer numerator and positive integer denominator.

A. *Naturals as subsort of integers.*

We wish to construct a specification for the naturals with zero and successor. We start from the specification INT for the integers with zero, successor and predecessor ordered by < (see Spec. 2.5 in 2.9).

We first introduce into INT the unary predicate non_neg over sort Int by the definition ($\forall$v:Int)[non_neg(v)$\leftrightarrow$zr$\leq$v]. We can then establish the non-voidness requirement ($\exists$v:Int)non_neg(v) (see 3.8.3). We can thus conservatively introduce a new sort Nat as the subsort of Int with relativisation predicate non_neg, together with corresponding insertion insrt(Nat)$\rightarrow$Int as well as conversion retr(Int)$\rightarrow$Nat in the opposite direction, the latter being constrained by a pre-post-condition axiom (see 3.8.6).

We can now introduce constant zero of sort Nat and operation succ from Nat to Nat by the definitions (see 3.8.6) zero$\approx_{Nat}$retr(zr) and succ(x)$\approx_{Nat}$retr(sc(insrt(x))). Notice that this populates sort Nat with names (for the naturals).

Among the consequences of this extended specificaton, we have

$(\forall y{:}\mathrm{Nat})[\neg y\approx_{\mathrm{Nat}}\mathrm{zero}\leftrightarrow(\exists x{:}\mathrm{Nat})y\approx_{\mathrm{Nat}}\mathrm{succ}(x)]$,

$(\forall x,y{:}\mathrm{Nat})[\mathrm{succ}(x)\approx_{\mathrm{Nat}}\mathrm{succ}(y)\rightarrow x\approx_{\mathrm{Nat}}y]$, and

for each n>0 $(\forall x{:}\mathrm{Nat})\neg x\approx_{\mathrm{Nat}}\mathrm{succc}^{n}(x)$, where

$\mathrm{succ}^{n}(x)$ stands for succ(...(succ(x))...) [n times].

Thus, we can conclude (see, e. g. Enderton 1972, p. 178-187) that our specification is an extension of NAT_ZR_SUCC given by Spec. 2.2 in 2.9.

So, we have constructed a series of specifications

$P_1$:=PD_DEF[non_neg(Int)\z r≤v](INT);

$P_2$:=SR_SBST[Nat\Int:non_neg,insrt]($P_1$),

$P_3$:=OP_PPC[retr(Int)→Nat\non_neg(x),insrt(y)$\approx_{\mathrm{Int}}$x→retr(x)$\approx_{\mathrm{Nat}}$y]($P_2$);

$P_4$:=CN_DEF[zero:Nat\y$\approx_{\mathrm{Nat}}$retr(zr)]($P_3$),

$P_5$:=OP_DEF[succ(Nat)→Nat\y$\approx_{\mathrm{Nat}}$retr(sc(insrt(x)))]($P_4$);

each one conservatively extending the preceding one. We can now take the restriction of $P_5$ to the sub-language consisting only of zero and succ. We then obtain a specification NATL:=RESTR[{zero,succ}]($P_5$) for naturals with zero and successor, such that

NATL is an expansive, so conservative, extension of INT, and

NATL is equivalent to NAT_ZR_SUCC.

B. *Rationals from integer arithmetic.*

We now wish to construct a specification for the rationals, by mimicking their usual representation as equivalence classes of fractions with integer numerator and positive integer denominator.

We will proceed in successive steps as displayed in figure 3.11.

$$\begin{tikzcd}
\text{INT\_ARITHM} \arrow[d, "\textit{add}\ \text{sub}"'] & & \\
\text{POS\_INT} \arrow[r, "\textit{add}\ \text{prod}"] & \text{FRTN} \arrow[r, "\textit{add}\ \text{quot}"] & \text{RATL}
\end{tikzcd}$$

Fig. 3.11: Sequence of specifications leading to rationals.

0. We start from the specification INT_ARITHM for the integers with arithmetic operations and predicates (see Spec. 2.6 in 2.9).

1. Our first step is the introduction of a sort for the positive integers.

This construction is analogous to the preceding one. From relativisation predicate is_pos, we obtain, as subsort, a new sort Pos_int with injective insertion ins(Pos_int)→Int as well as opposite surjective conversion rtr(Int)→Pos_int. We can then introduce constant one and operation scc, which populates sort Pos_int with names.

2. Our next step is the introduction of a sort for the fractions.

We construct sort Frtn as the product of sorts Int and Pos_int with projections nm(Frtn)$\rightarrow$Int and dn(Frtn)$\rightarrow$Pos_int as well as conversion mkfr(Int,Pos_int)$\rightarrow$Frtn in the opposite direction (see 3.8.6).

3. The third step is the introduction of equivalence of fractions.

We have the familiar idea: fractions n/d and n'/d' represent the same rational when n$*$d' = n'$*$d. We code this idea of equivalence of fractions into a definition for binary predicate eqv over sort Frtn: $(\forall$z,z':Frtn)[eqv(z,z')$\leftrightarrow$nm(z)$*$ins(dn(z'))$\approx_{Int}$nm(z')$*$ins(dn(z))].

We can then establish the equivalence requirement $\varepsilon$(eqv) (see 3.8.4). We can thus conservatively introduce a new sort Ratl as the quotient of Frtn under equivalence predicate eqv, with corresponding canonical projection val(Frtn)$\rightarrow$Ratl as well as injective opposite conversion rpr(Ratl)$\rightarrow$Frtn (see 3.8.6).

4. Our fourth step is the introduction of arithmetic on rationals.

We will illustrate this step with a few simple cases; other arithmetic operations and predicates can be similarly introduced (see 3.8.6).

First, recall that equality of rationals is equivalence of the corresponding fractions: u$\approx_{Ratl}$u'$\leftrightarrow$eqv(rpr(u),rpr(u')).

We can introduce constant 0/1 on sort Ratl by the definition 0/1$\approx_{Ratl}$val(mkfr(zr,one)). Similarly, each pair of names for Int and Pos_int generates a name for a rational: m/n$\approx_{Ratl}$val(mkfr(m,n)).

An operation unary minus on sort Ratl can be introduced by a definition like mns(u)$\approx_{Ratl}$val(mkfr(zr-nm(rpr(u)),dn(rpr(u)))).

So, starting from $P_0$=INT_ARITHM, we have constructed a sequence of specifications

1. $P_1$:=PD_DEF[is_pos(Int)\zr<v]($P_0$),
   P'$_1$:=SR_SBST_CNV[Pos_int\Int:is_pos,ins,rtr]($P_1$),
   P"$_1$:=CN_DEF[one:Pos_int\y$\approx_{Pos_int}$rtr(sc(zr))](P'$_1$)
   P$^*_1$:=OP_DEF[scc(Pos_int)$\rightarrow$Pos_int\y$\approx_{Pos_int}$rtr(sc(ins(x)))](P"$_1$);
2. $P_2$:=SR_PROD[Frtn\nm$\rightarrow$Int,dn$\rightarrow$Pos_int](P$^*_1$),
   P'$_2$:=OP_DEF[mkfr(Int,Pos_int)$\rightarrow$Frtn\nm(y)$\approx_{s_1}x_1\wedge$dn(y)$\approx_{s_2}x_2$]($P_2$);
3. $P_3$:=PD_DEF[eqv(Frtn,Frtn)\nm(z)$*$ins(dn(z'))$\approx_{Int}$nm(z')$*$ins(dn(z)](P'$_2$),
   P'$_3$:=SR_QUOT_CNV[Ratl\Frtn/eqv,val,rpr]($P_3$);
4. $P_4$:=CN_DEF[0/1:Ratl\y$\approx_{Ratl}$val(mkfr(zr,one))](P'$_3$),

$P'_4$:=OP_DEF[mns(Ratl)Ratl\y$\approx_{Ratl}$vl(mkfr(zr-nm(rpr(u)),dn(rpr(u))))]($P_4$)
each one conservatively extending the preceding one.

Figure 3.12 provides a bird's eye view of the construction history of this specification for rationals.

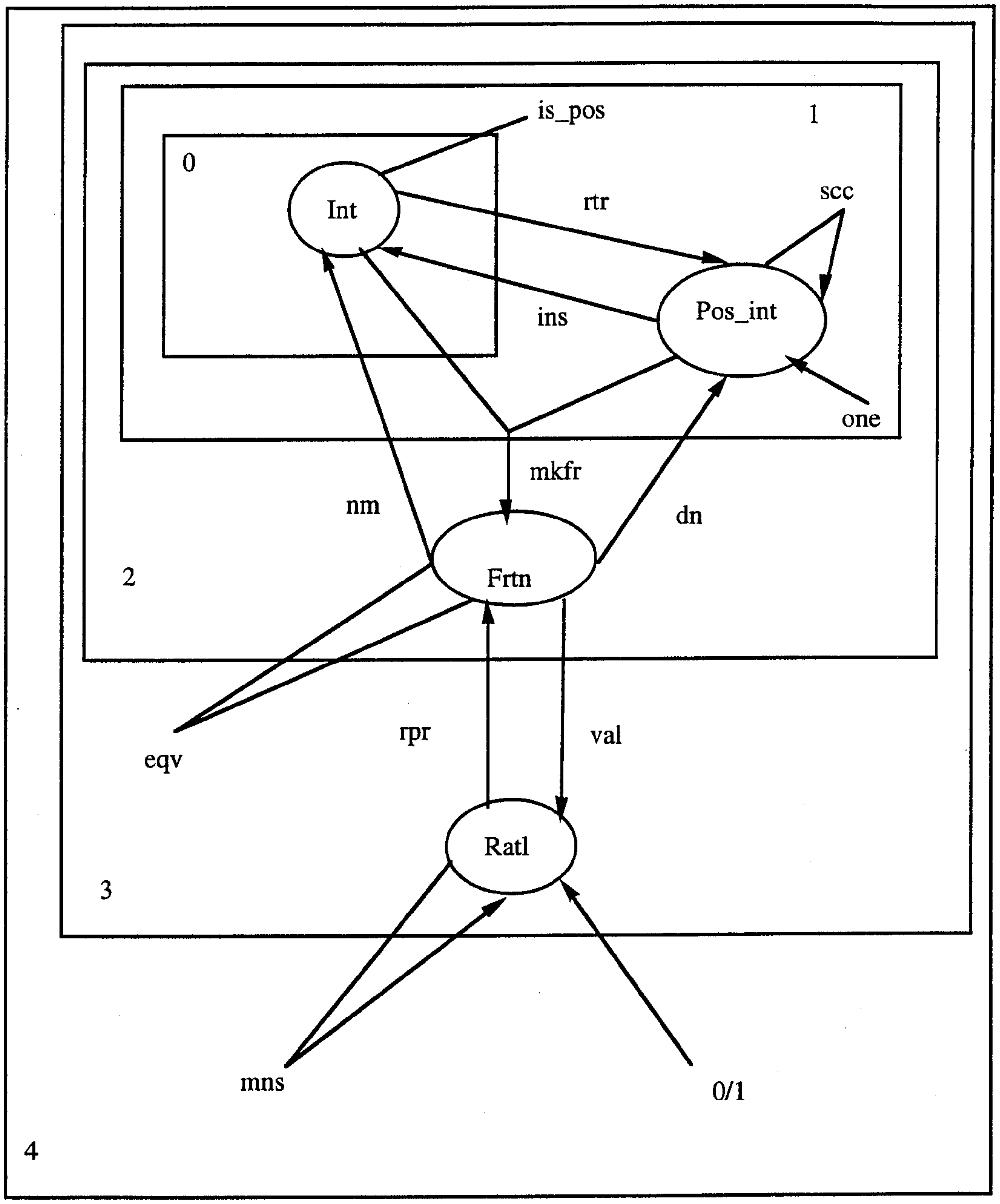

Fig. 3.12: Construction history of specification for rationals.

The final specification constructed for rationals has the following features

we can use it for reasoning about rational arithmetic,
we still have access to the representation of the rational as fractions,
its expression as extensions records the trace of its construction.

If we wish to hide details, such as representation and history of construction, we can now take its restriction to the sub-language consisting only of sort Ratl and the desired arithmetic operations and predicates. We then obtain a specification RATL for rational arithmetic that is a conservative extension of INT_ARITHM.

## 3.10 Example Specifications

Spec. 3.1. BOOL: Boolean values

```
SPEC BOOL                              {Specification of Bool(ean values)}
  DECLARATIONS
    Sorts
      Bool                                         {The (only) sort is Bool}
    Operations                              {No (non-nullary) operations}
    Constants
      tr,fl: Bool                         {tr and fl are constants of Bool}
    Predicates                          {No predicates (except equality)}
  AXIOMS
   (∀x:Bool)[x≈_Bool tr∨x≈_Bool fl]              {tr and fl exhaust Bool},
   ¬tr≈_Bool fl                                     {tr and fl distinct}.
END_SPEC BOOL
```

Spec. 3.2. BOOL_EXT_NEG&LESS: Boolean extended by negation and ordering

```
SPEC BOOL_EXT_NEG&LESS := EXT of BOOL by            {Extension of BOOL}
  DECLARATIONS                              {Description of new symbols}
    Sorts                                                 {No new sort}
    Operations                       {List of new (non-nullary) operations}
      neg (Bool)→Bool                       {neg is from Bool into Bool}
    Constants                                         {No new constant}
    Predicates                                  {List of new predicates}
      less? (Bool,Bool)                     {less? over Bool and Bool}
  AXIOMS                                            {List of new axioms}
   (∀x,y:Bool)[¬neg(x)≈_Bool x]                       {a property of neg},
```

$(\forall x,y{:}\mathrm{Bool})[\mathrm{less?}(x,y)\leftrightarrow(x\approx_{\mathrm{Bool}}\mathrm{fl}\wedge y\approx_{\mathrm{Bool}}\mathrm{tr})]$ {a definition of less?},

END_SPEC BOOL_EXT_NEG&LESS

## 3.11 References

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

Barwise, J. ed. (1977) *Handbook of Mathematical Logic.* North-Holland, Amsterdam.

Bauer, F., L. and Wössner, H. (1982) *Algorithmic Language and Program Development.* Springer Verlag, Berlin.

Broy, M. (1983) Program construction by transformations: a family of sorting programs. In Breuman, A. W. and Guiho, G. (eds) *Automatic Program Constructiom*, Reidel, Dordrecht.

Broy, M., Pair, C. and Wirsing, M (1981) A systematic study of models of abstract data types. Centre de Recherche en Informatique de Nancy Res. Rept. 81-R-042, Namcy.

Broy, M. and Pepper, P. (1981) Program development as a formal activity. *IEEE Trans. Software Engin.*, **SE-7** (1)14-22.

Broy, M. and Wirsing, M (1982) Partial abstract data types. *Acta Informatica*, **18** 47-64.

Byers, P. and Pitt, D. (1990) Conservative extensions: a cautionary note. *Bull. EATCS*,41, 196-201.

Darlington, J. (1978) A synthesis of several sorting algorithms. *Acta Informatica*, **11** (1), 1-30.

Dahl, O., Dijkstra, E. and Hoare, C. (1972) *Structured Programming.* Academic Press, New York.

Ebbinghaus, H. D., Flum, J. and Thomas, W. (1984) *Mathematical Logic.* Springer-Verlag, Berlin.

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

Ehrich, H.-D. (1982) On the theory of specification, implementation and parameterization of abstract data types. *J. ACM*, **29** (1), 206-227.

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

Gehani, A., D., McGettrick (1986) *Software Specifications Techniques.*

Addison-Wesley, Reading.

Ghezzi, C., Jazayeri, M. (1982) *Programming Languages Concepts*. Wiley, New York.

Goguen, J. A.; Thatcher, J. W. & Wagner, E. G. (1978) An initial algebra approach to the specification, correctness and implementation of abstract data types. In Yeh, R. T. (ed.) *Current Trends in Programming Methodology*: Prentice Hall, Englewood Cliffs, . 81-149.

Guttag, J. V (1977) Abstract data types and the development of data structures. *Comm. Assoc. Comput. Mach.*, **20** (6), 396-404.

Guttag, J. V (1980) Notes on type abstraction. *IEEE Trans. Software Engin.*, **6** (1),.

Guttag, J. V. and Horning, J. J. (1978) The algebraic specification of abstract data types. *Acta Informatica*, **10** (1), p. 27 - 52.

Hoare, C. A. R. (1972) Proof of correctness of data representations. *Acta Informatica*, **4**, 271-281.

Hoare, C. A. R. (1974) Notes on data structuring. In Dahl *et al.* 1(974); 83-174.

Hoare, C. A. R. (1978) Data Structures. In Yeh, R. (ed.) *Current Trends in Programming Methodology, Vol IV.* Prentice Hall, Englewood Cliffs, 1-11.

Jackson, M., A. (1980) *Principles of Program Design*. Academic Press, London.

Ledgard, H. and Taylor, R. W. (1977) Two views on data abstraction. *Comm. Assoc. Comput. Mach.*, **20** (6), 382-384.

Maibaum, T. S. E. (1986) The role of abstraction in program development. In Kugler, H.-J. ed. *Information Processing '86*. North-Holland, Amsterdam, 135-142.

Maibaum, T. S. E., Sadler, M. R. and Veloso, P. A. S. (1984) Logical specification and implementation. In Joseph, M. and Shyamasundar R. eds. *Foundations of Software Technology and Theoretical Computer Science*. Springer-Verlag, Berlin, 13-30.

Maibaum, T. S. E. and Turski, W. M. (1984) On what exactly is going on when software is developed step-by-step. *tProc. 7h Intern. Conf. on Software Engin.* IEEE Computer Society, Los Angeles, 528-533.

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

Maibaum, T. S. E, Veloso, P. A. S. and Sadler, M. R. (1991) A logical approach to specification and implementation of abstract data types. Imperial College of Science, Technology and Medicine, Dept. of Computing Res. Rept. DoC 91/47, London.

Manna, Z. (1974) *The Mathematical Theory of Computation.* McGraw-Hill, New York.

Meré, M. C. ; Veloso, P. A. S. (1992) On extensions by sorts.. PUC - RJ, Dept. Informática, Res. Rept. MCC 38/92, Rio de Janeiro..

Pair, C. (1980) Sur les modèles des types abstraites algébriques. Centre de Recherche en Informatique de Nancy Res. Rept. 80-p-042, Namcy.

Parnas, D. L. (1979) Designing software for ease of extension and contraction. *IEEE Trans. Software Engin.*, **5** (2), 128-138.

Pequeno, T. H. C. and Veloso, P. A. S. (1978) Do not write more axioms than you have to. *Proc. Intern. Computing Symposium*, Taipei, 487-498.

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

Smirnov, V. A. (1986) Logical relations between theories. *Synthese*, **66**, p. 71 - 87.

Smith, D. R. (1985) The Design of Divide and Conquer Algorithms. *Science Computer Programming*, **5** 37-58.

Smith, D. R. (1990) Algorithm theories and design tactics. *Science of Computer Programming.*, **14**, 305-321.

Smith, D. R. (1992) Constructing specification morphisms. Kestrel Institute, Tech. Rept. KES.U.92.1, Palo Alto.

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

van Dalen, D. (1989) *Logic and Structure* (2nd edn, 3rd prt). Springer-Verlag, Berlin.

Veloso, P. A. S. (1984) Outlines of a mathematical theory of general problems. *Philosophia Naturalis*, **21** (2/4 ), 354-362.

Veloso, P. A. S. (1985) On abstraction in programming and problem solving. *2nd Intern. Conf. on Systems Research, Informatics and Cybernetics*. Baden-Baden.

Veloso, P. A. S. (1987) *Verificação e Estruturação de Programas com Tipos de Dados*. Edgard Blücher, São Paulo.

Veloso, P. A. S. (1987) On the concepts of problem and problem-solving method. *Decision Support Systems*, **3** (2), 133-139.

Veloso, P. A. S. (1988) Problem solving by interpretation of theories. In Carnielli, W. A. ; Alcântara, L. P. eds. *Methods and Applications of Mathematical Logic*. American Mathematical Society, Providence, . 241-250.

Veloso, P. A. S. (1991) A computing-like example of conservative, non-expansive, extension. Imperial College of Science, Technology and Medicine, Dept. of Computing, Res. Rept. DoC 91/36, London.

Veloso, P. A. S. (1992) Yet another cautionary note on conservative extensions: a simple example with a computing flavour. *Bull. EATCS*, 46, 188-192.

Veloso, P. A. S. (1992) On the modularisation theorem for logical specifications: its role and proof. PUC - RJ, Dept. Informática Res. Rept. MCC 17/92, Rio de Janeiro.

Veloso, P. A. S. (1992) Notes on interpretations of logical specifications. COPPE-UFRJ Res. Rept. ES-277/93, Rio de Janeiro.

Veloso, P. A. S. (1993) The Modularization Theorem for unsorted and many-sorted specifications. COPPE-UFRJ Res. Rept. ES-284/93, Rio de Janeiro.

Veloso. P. A. S. and Maibaum, T. S. E. (1984) What is wrong with errors: incomplete specifications for abstract data types. UFF, ILTC, Res. Rept., Niterói.

Veloso, P. A. S., Maibaum, T. S. E. and Sadler, M. R. (1985) Program development and theory manipulation. In *Proc. 3rd Intern. Workshop on Software Specification and Design*. IEEE Computer Society, Los Angeles, 228-232.

Veloso, P. A. S. and Maibaum, T. S. E. (1992) On the Modularisation Theorem for logical specifications. Imperial College of Science, Technology & Medicine, Dept. of Computing Res. Rept. DoC 92/35, London.

Veloso, P. A. S. and Veloso. S. R. M. (1981) Problem decomposition and reduction: applicability, soundness, completeness. In Trappl, R.; Klir, J. ; Pichler, F. eds. *Progress in Cybernetics and Systems Research.* Hemisphere, Washington, DC, 199-203.

Veloso, P. A. S. and Veloso. S. R. M. (1990) On extensions by function symbols: conservativeness and comparison. COPPE-UFRJ Res. Rept. ES-288/90, Rio de Janeiro.

Veloso, P. A. S. and Veloso, S. R. M. (1991) Some remarks on conservative extensions: a Socratic dialogue. *Bull. EATCS*, 43, 189-198.

Veloso, P. A. S. and Veloso, S. R. M. (1991) On conservative and expansive extensions. *O que no faz pensar: Cadernos de Filosofia*, 4, 87, 106.

Veloso, P. A. S. and Veloso, S. R. M. (1991) On conservative and expansive extensions: why and how they differ. Imperial College of Science, Technology & Medicine, Dept. of Computing Res. Rept. DoC 91/30, London.

Wirsing, M., Pepper, P. and Broy, M. (1983) On hierarchies of abstract data types. *Acta Informatica* **20** (1) 1-33.