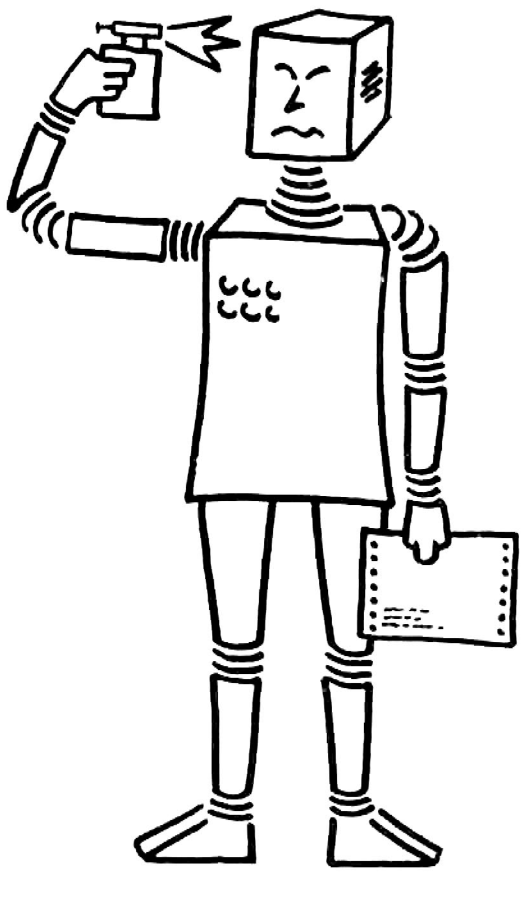
SEKI-PROJEKT

SEKI
MEMO

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany



Parameterization-by-use for hierarchically structured objects

Ch. Beierle, A. Voß

Memo SEKI-83-08

Parameterization-by-use for hierarchically structured objects

Christoph Beierle, Angi Voß

Universität Kaiserslautern
Postfach 3049
6750 Kaiserslautern
West Germany

Abstract

A formal model for hierarchical objects is presented. The hierarchical structure between objects is defined by a general notion of use relationship. Used objects may be regarded as formal parameters leading to the definition of parameter applications and a new parameterization concept called parameterization-by-use. We study hierarchies with all applications and give a canonical closure construction to generate such hierarchies. We show how these consepts can be incorporated into a specification language for hierarchically structured objects.

This research was supported by the Bundesministerium für Forschung und Technologie under contract IT.8302363.

Contents

1. Introduction 1

2. Hierarchically structured objects 2
  2.1 Appropriate categories 2
  2.2 Appropriate orders 3
  2.3 Hierarchies 4

3. Parameters and applications in hierarchies 6
  3.1 Motivation 6
  3.2 Parameter sets 8
  3.3 Applications 10

4. Hierarchies with direct applications 18
  4.1 Closed hierarchies 18
  4.2 Evaluation of direct application terms 19
  4.3 Composition of direct applications 20

5. Generalized applications 36
  5.1 Indirect applications 36
  5.2 Application terms and their evaluation 44

6. Extension and closure of hierarchies 46
  6.1 Canonical closure 46
  6.2 Prefix hierarchies 50

7. Parameterization-by-use in specification languages 58
  7.1 A general hierarchy specification language 58
  7.2 A specification language for hierarchies of signatures 61
    7.2.1 Signatures 61
    7.2.2 Hierarchically structured signatures 66
    7.2.3 Evaluating signature application terms 69
  7.3 Non-proliferic semantics for specification languages 73

References 76

1. Introduction

Parameterization is a well-established concept in many programming languages (e.g. [Na 63], [JW 76],[WLS 76], [LALS 77]). The work on specifications revealed that also for specification languages parameterization mechanisms are needed in order to reuse specifications in different contexts and to adjust them to particular situations (e.g. [BG 77], [BG 80], [GT 79], [TWW 82], [Eh 82], [Eh 81], [Ba 81], [Ga 81]).

Hierarchical structures arise in programming languages (e.g. structured programming), specification languages (e.g. hierarchical specifications) and many other areas (e.g. classifications of objects, divisions in a company, etc.).

In this paper we investigate the relationship between parameterization and hierarchical structures in a general setting. We give a general notion of hierarchical objects and propose a parameterization concept which we call parameterization-by-use. In this concept, the declaration of formal parameters can be disposed of. Instead, every object may be regarded as a formal parameter, identified only when it is associated with an actual parameter. In this way, instances or applications of hierarchical objects are generated, which are again hierarchical objects. We study the properties of hierarchies that are closed under applications and introduce a canonical closure construction.

In order to give an idea of how parameterization-by-use may be incorporated in a language for the definition of hierarchically structured objects, we define such a specification language and give various example specifications demonstrating the properties of the parameterization concept.

In Section 2, hierarchical structures are defined using the notions of appropriate category and appropriate order.

Applications in hierarchies are introduced in Section 3. In Section 4, we concentrate on a special type of application, the direct applications, while the general case is considered in Section 5. Section 6 introduces our canonical closure construction. In Section 7, a general specification language is proposed, several examples are given and it is shown how the use of closed hierarchies leads to non-proliferic (c.f. [BG 81], [Sa 81]) semantics for specification languages.

2. Hierarchically structured objects

2.1 Appropriate categories

A hierarchically structured object is an object that is based on a set of other hierarchical objects. Often, the hierarchical relationship between the objects involved is a kind of subpart-relationship, e.g. terms in a formal language, nested blocks or macros in a programming language, classifications of objects, divisions in a company, etc.

Thus, without making any further assumptions about the kind of hierarchical relationship, we will just assume that there is only one type of it. If the objects are taken from some category, a hierarchical relationship between two objects is given by a certain type of unique morphism in that category.

Definition 2.1 [appropriate category]

A category C is an appropriate category with subcategory $\vec{C}$ iff $\vec{C}$ has all objects of C and there is at most one morphism between any two objects in $\vec{C}$, i.e.

$$\forall c,c' \varepsilon /\vec{C}/ \;.\; |\mathrm{hom}(c,c')| \leq 1$$

Usually, a $\vec{C}$-morphism will be called an <u>inclusion</u> and will be denoted by $\hookrightarrow$. The obvious embedding functor from $\vec{C}$ into C is

$$\mathrm{into}_C\colon \vec{C} \to C.$$

Often we will ambiguously call the pair $AC=(C,\vec{C})$ itself an appropriate category.

2.2 Appropriate orders

Given the general notion of inclusion above, a hierarchical object is an object x together with a set B of hierarchical objects such that there is an inclusion from b to x for every $b \in B$. We say that x <u>uses</u> every $b \in B$. Since the use-relationship must not introduce any cycles, a set of hierarchical objects may be represented by an acyclic graph where the use-relationship corresponds to a path in that graph. Furthermore, if we assume that a set M of hierarchical objects has exactly one element that uses no other objects and all other objects use a non-empty but finite set of other objects, the representing acyclic graph for M defines an appropriate order:

<u>Definition 2.2</u> [appropriate order]

An <u>appropriate order</u> $AO = (O,<,\perp)$ is a well founded irreflexive partial order $(O,<)$ with minimum $\perp$ such that every element has only finitely many predecessors.

<u>Notation:</u> $\leq$ denotes the reflexive closure of $<$.

An appropriate order AO defines an acyclic graph with the desired properties. The order category induced by AO will also be denoted by AO. It has objects O and all order relations as morphisms, i.e. an arrow $a \to b$ iff $a \leq b$. Next we will introduce structure preserving maps between appropriate orders.

Definition 2.3 [appropriate order morphism]

An appropriate order morphism $f: AO \to AO'$ is a functor between the corresponding order categories such that $f(\perp) = \perp'$.

Fact 2.1

Appropriate order morphisms are determined uniquely by the object part of their defining functors.

Proof: Because there is at most one morphism (i.e. order relation) between any two objects in an order category.

2.3 Hierarchies

A hierarchy defines a set of objects that can be constructed step by step: starting with an object that uses no other objects, all other objects can be added using at least one but only a finite number of already existing objects. Thus given the definitions above, a hierarchy of objects in an appropriate category $(C, \vec{C})$ is a set of objects together with an appropriate order between them where the relationship A uses B corresponds to an inclusion $B \hookrightarrow A$.

Definition 2.4 [hierarchy, use-relationship]

Given an appropriate order AO and an appropriate category $AC=(C,\vec{C})$, a hierarchy

$H: AO \to AC$

is a functor $H: AO \to \vec{C}$. For $n \in O$ we say that $H(n)$ uses $H(m)$ for every $m \in O$ with $m<n$.

Notation: We will denote a hierarchy by $H: AO \to \vec{C}$ assuming that the appropriate category is given by $(C,\vec{C})$.

For a hierarchy $H: AO \to \vec{C}$ we will talk about the hierarchical object $H(n)$ for an $n \in O$ referring to the fact that it uses all objects $H(m)$ with $m<n$.

Some obvious facts about hierarchies are:

<u>Fact 2.2</u>

A hierarchy $H\colon AO \to \vec{C}$ is determined by the object part of the functor H.

<u>Proof:</u> Follows from the fact that there is at most one morphism between any two objects in $\vec{C}$.

<u>Fact 2.3</u>

For a hierarchy $H\colon AO \to \vec{C}$

$$\mathrm{into}_C \circ H\colon AO \to C$$

is a commutative diagram in C.

<u>Proof:</u> Because $H\colon AO \to \vec{C}$ is a functor and there is at most one morphism between any two objects in the subcategory $\vec{C}$.

3. Parameters and applications in hierarchies

3.1. Motivation

In programming languages such as ALGOL 60 [Na 63] and PASCAL [JW 76] as well as in specification languages such as Clear [BG 80] and CIP-L [Ba 81] the parameterization concepts involve two basic steps:

- Declaring formal parameters when defining a parameterized object.
- Giving a correspondence between formal and actual parameters when instantiating a parameterized object.

In general, the following points must be observed and may be regarded as drawbacks in some applications:

1. When defining a parameterized object P one has to give the complete set of P´s formal parameters. If later on one realizes that some other parts of P could be regarded as a formal parameters and one wants to substitute them by other objects it turns out to be impossible without rewriting P and extending its parameter declaration.

2. To instantiate a parameterized object P with formal parameter set F an actual parameter for each $x \in F$ must be given. Even if one actually wants a partial instantiation of P where some formal parameters $F' \subset F$ are kept unchanged one still has to give a dummy actual parameter for every $x \in F'$.

3. The distinction between non-parameterized, parameterized and parameter objects sometimes appears to be artificial, e.g. regarding a parameterized object as a non-parameterized object or vice versa may not be possible.

These three observations apply especially to specification languages. Given a specification, say, of ARRAY with parameters

INDEX and ELEMENT, a partial instantiation of ARRAY with actual parameter INTEGER for INDEX but leaving ELEMENT unchanged is usually not supported by the parameterization concept. The identification of loose specifications and parameter specifications in [BG 80] lead to a subtle error as pointed out in [Sa 81]: an actual parameter still had to contain its formal parameter. But the solution suggested in [Sa 81] introduces a distinction between the two types of specifications such that a parameter (meta theory) differs from an ordinary specification (theory) only in the keyword ´meta`.

In order to overcome these difficulties we propose a new parameterization concept for hierachical structures which is called parameterization-by-use. Given the notion of hierarchy and use relationship introduced above it is based on the following principles:

- When defining hierarchical objects no distinction whatsoever is made between parameters or used objects.
- Every object may function as a non-parameterized object, as a parameterized one or as a parameter; consequently no such distinction is made when defining an object.
- Instead of declaring parameters when defining an object X parameters are identified when some other object uses an instantiation of X.
- Every object used by some object X may be regarded as a formal parameter and may be actualized by some other object.
- Instantiation must be compatible with the hierarchical structure of the objects involved.

To make these ideas more precise we need some preparatory definitions.

3.2 Parameter sets

Definition 3.1 [base, parameter set]

Let $AO=(O,<,\perp)$ be an appropriate order, $n \in O$, $M \subseteq O$.

(1) The base of n in AO is given by

$\text{base}(n,AO) := \{n' \mid n' < n\}$

i.e. the set of all elements in AO with a path to n.

(2) M is a parameter set for n iff

- $M \subseteq \text{base}(n,AO)-\{n\}$
- $\forall m, m'' \in M \,.\, \forall m' \in O \,.\, m<m'<m'' \Rightarrow m' \in M$

(3) M is a direct parameter set for n iff

- M is a parameter set for n and
- $\forall m \in M \,.\, \forall m' \in O \,.\, m<m'<n \Rightarrow m' \in M$

(4) The set of elements between n and a parameter set M for n is given by

$\text{between}(n,M,AO) := \{o \in O-M \mid \nexists\, m \in M \,.\, m < o < n\}$.

(5) The base of n w.r.t. parameters M is given by

$\text{base}(n,M,AO) := \text{base}(n,AO)-(M \cup \{n\} \cup \text{between}(n,M,AO))$

Note that a parameter set may be empty. Figure 3.1 gives an illustration for base and parameter sets.

Fact 3.0

Let $AO = (O,<,\perp)$ be an appropriate order, $n \in O$, M a parameter set for n. Then:

- M is a direct parameter set for n iff $\text{between}(n,M,AO) = \phi$.
- $M \cup \text{between}(n,M,AO)$ is a direct parameter set for n.
- The four sets
  - $\{n\}$
  - $\text{between}(n,M,AO)$
  - M
  - $\text{base}(n,M,AO)$

  are pairwise disjoint and their union is
  - $\text{base}(n,AO)$

Proof: Immediately from Definition 3.1.

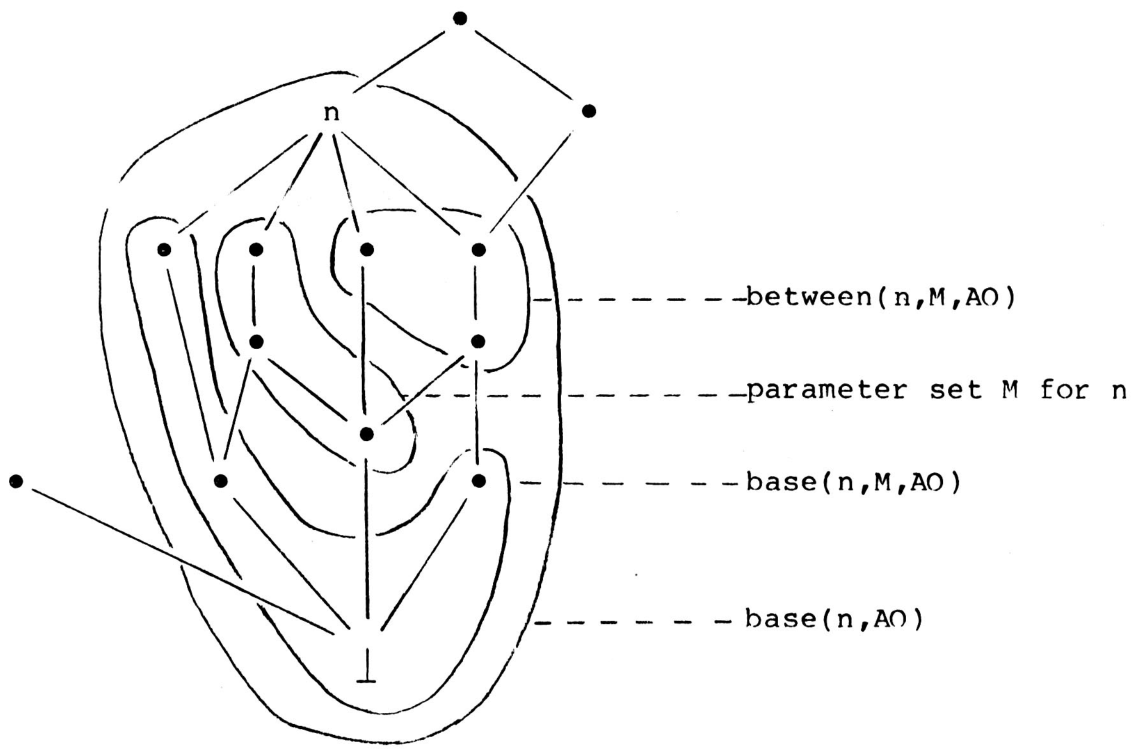


Figure 3.1: base and parameter set for a node n (c.f. Def. 3.1)

Fact 3.1

Let $AO=(O,<,\perp)$ be an appropriate order, $n\varepsilon O$, M a parameter set for n, H: $AO \to \vec{C}$ a hierarchy.

Then:

- base(n,AO) and base(n,M,AO) are appropriate orders where the order relationship is inherited from AO.
- $H_{|base(n,AO)}$ and $H_{|base(n,M,AO)}$, the restrictions of H to base(n,AO) resp. base(n,M,AO), are again hierarchies.

Proof: Immediately from Def. 3.1.

3.3 Applications

Given a hierarchy H: $AO \to \vec{C}$ and a node $n\varepsilon O$ together with a parameter set M, the hierarchical object $H(n)$ can be viewed as a parameterized object with parameters $\{H(m)|m\varepsilon M\}$. An instantiation or application may be generated by providing an object A_m for every $m\varepsilon M$ and a means of getting from $H(m)$ to A_m, i.e. a morphism $f_m:H(m) \to A_m$. The object A_m is again a hierarchical object subject to the condition that it is built upon the same objects as $H(m)$. First we concentrate on the case of direct parameter sets; the general case of parameter sets is studied in Sec. 5. For direct parameters the conditions on the morphisms f_m are captured by the following fact:

Fact 3.2

Let H: $AO \to \vec{C}$ be a hierarchy, $AO=(O,<,\perp)$, $n\varepsilon O$ and M a direct parameter set for n. Let (f,h) be a pair with:

(i) f: base$(n,\phi,AO) \to AO$
is an appropriate order morphism with $f(x)=x$ for all $x \notin M$.

(ii) h: $into_C \circ H_{|base(n,\phi,AO)} \to into_C \circ H \circ f$
is a natural transformation with $h_x=id_{H(x)}$

for all $x \notin M$.

Then for all $x,y \in O$ with $x<y<n$ the diagram

$$\begin{tikzcd}
H(y) \arrow[r, dashed, "h_y"] & H(f(y)) \\
H(x) \arrow[u, hook] \arrow[r, dashed, "h_x"] & H(f(x)) \arrow[u, hook]
\end{tikzcd}$$

commutes in C.

Proof: $x<y<n$ implies $x,y \in \text{base}(n,\phi,AO)$, and the commutativity of the diagram represents the natural transformation property of h.

Since f and h of the previous fact are determined by the two maps

$$f': M \to O$$

$$h': M \to /C/$$

where $f'(x)=f(x)$ and $h'(x)=h_x$ it suffices to supply f' and h' (c.f. Figure 3.2).

Definition 3.2 [direct application term]

Let H,n,M as before.

$T = n\{(m,h_m,f(m)) \mid m \in M\}$ is a direct application term of H iff f and h define an appropriate order morphism and a natural transformation fulfilling the conditions of Fact 3.2. T is trivial iff $M=\phi$.

The result of a direct application term $T = n\{(m,h_m,f(m)) \mid m \in M\}$ in H is constructed from the hierarchical object H(n) by removing the objects H(m) for each formal parameter $m \in M$ and by substituting the objects H(f(m)). Similar to e.g. [Eh 82], [EKTWW 80], [BG 80], [Eh 81], [Ga 81] and [Li 82] where a related process is defined using pushouts we define the result of a

$$\begin{tikzcd}
 & n & & & & & & \\
 & m_1 \arrow[u, hook] \arrow[rrrr, dashed, "h'_{m1}"] & & & & f'(m_1) & & \\
 & & & m_2 \arrow[rrrr, dashed, "h'_{m2}"] & & & & f'(m_2) \\
 & & m_3 \arrow[uul, hook] \arrow[ur, hook] \arrow[rrrr, dashed, "h'_{m3}"] & & & & f'(m_3) \arrow[uul, hook] \arrow[ur, hook] & \\
b_1 \arrow[uuuur, hook] & & & & & & & \\
 & & & & b_2 \arrow[uull, hook] \arrow[uurr, hook] & b_3 \arrow[uuull, hook] \arrow[uuurr, hook] & & b_4 \arrow[uuu, hook] \\
 & & & & \perp \arrow[uullll, hook] \arrow[u, hook] \arrow[ur, hook] \arrow[urrr, hook] & & &
\end{tikzcd}$$

Figure 3.2: f' and h' determining f and h of Fact 3.2 for the direct parameter set $M = \{m_1, m_2, m_3\}$ of n

direct application using colimits.

Definition 3.3 [application diagram, application object]

Let H: AO → $\vec{C}$ be a hierarchy.

(1) For a direct application term $T=n\{(m,h_m,f(m))|m\varepsilon M\}$ in H diagram(T) is called a direct application diagram of H. It is a diagram in C with:

nodes: $\{n_n\} \cup \{m_n|\ m\varepsilon M\} \cup \bigcup_{m\varepsilon base(n,\phi,AO)} base(f(m),AO)$

where the x_n are new nodes not in O for $x \varepsilon \{n\} \cup M$. x_n is labelled by H(x), $x\varepsilon O$ is labelled by H(x).

edges: - all edges from AO, labelled by H.

- edges between x and y_n labelled by H((x,y)) iff there is an edge between x and y in AO.
- edges between x_n and y_n labelled by H((x,y)) iff there is an edge between x and y in AO.
- edges from m_n to f(m) labelled by h_m for every $m\varepsilon M$.

(2) The set base(T) is given by $\bigcup_{m\varepsilon base(n,\phi,AO)} base(f(m),AO)$

(3) If diagram(T) has a colimit c in C such that the colimit injections from base(T) are inclusions - i.e. morphisms in $\vec{C}$ - then c is a direct application object (or just application object) for T.

In Figure 3.3 an illustration is given for an application diagram, its base and its application object. Figure 3.4 shows a further example.

Fact 3.3

Any two application objects c and c′ for a direct application term T are isomorphic in C. Moreover, an application object may be determined uniquely by giving the colimit injection from H(n) to c.

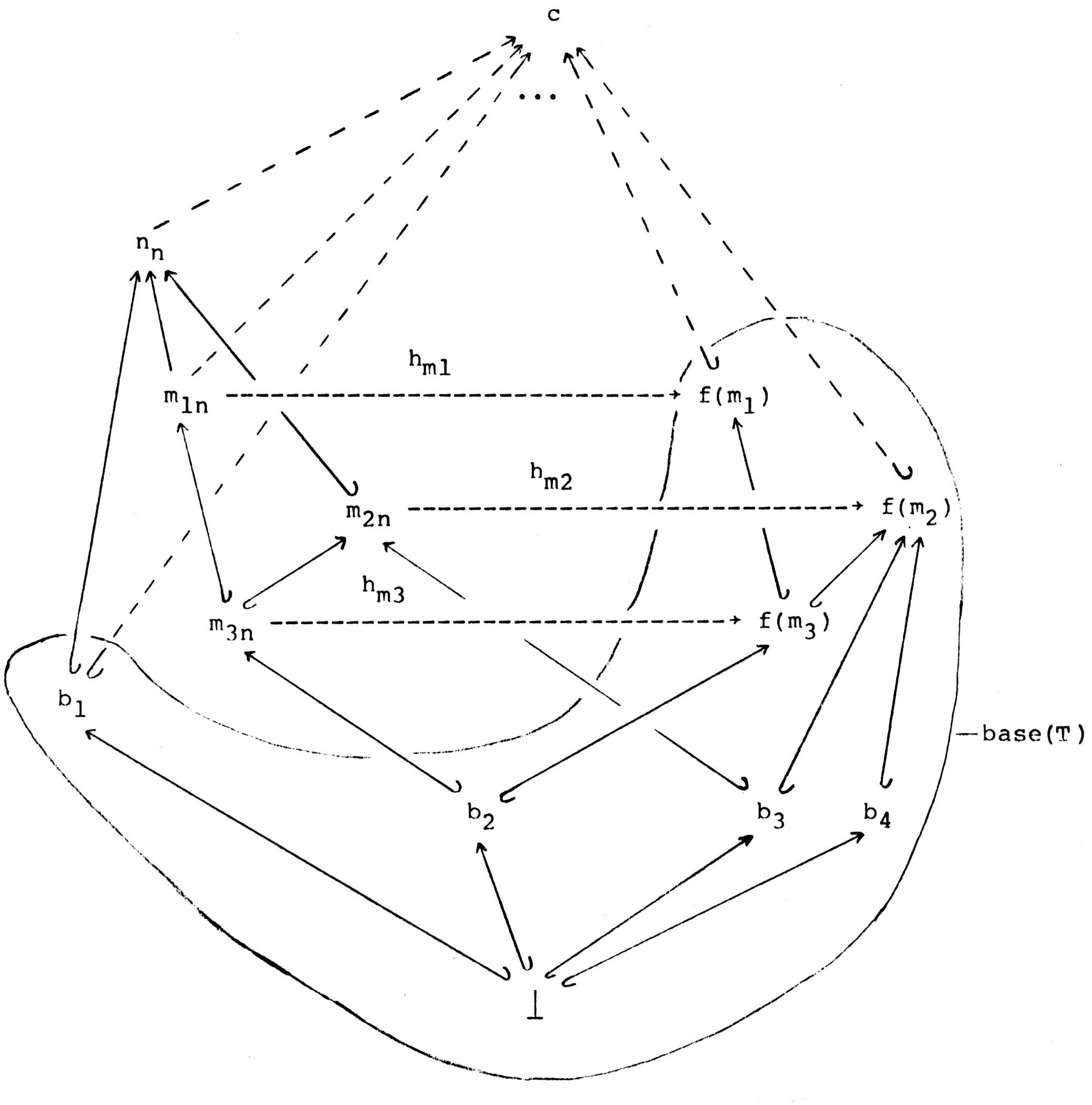


Figure 3.3: application diagram for $T = n\{(m_i, h_{m_i}, f(m_i)) \mid i \in \{1,2,3\}\}$ with application object c (c.f. Def. 3.3)

(a)

$$\begin{tikzcd}
n \\
m \arrow[u, hook] \arrow[u, bend right=60, "h_m"'] \\
b \arrow[u, hook] \\
\perp \arrow[u, hook]
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
 & c & & \\
n_n \arrow[ur, dashed] & \cdots & & \\
m_n \arrow[u, hook] \arrow[uur, dashed] \arrow[rr, dashed, "h_m"] & & n \arrow[uul, dashed] & \\
 & & m \arrow[u, hook] & \text{—base(T)} \\
 & b \arrow[uul, hook] \arrow[ur, hook] & & \\
 & \perp \arrow[u, hook] & &
\end{tikzcd}$$

Figure 3.4: (a) $T = n\{(m, h_m, n)\}$ is a direct application term for $f(m) = n$ and h_m: $H(m) \to H(n)$

(b) The application diagram for T with application object c (c.f. Def. 3.3)

Proof: Colimits are isomorphic (e.g.[McL 71], [HS 73]) and there is at most one inclusion from any object into c.

Thus we will talk about the application object of T, denoted by application-object(T). Whenever it is necessary it may be identified by giving the injection from H(n). It can be viewed as a hierarchical object that uses all objects H(x) with $x \in$ base(T).

Fact 3.4

For every trivial direct application term T = n{ } in a hierarchy H: AO $\rightarrow \vec{C}$ we have

- H(n) = application-object(T)
- base(n,ϕ,AO) = base(T).

Proof: Since M=ϕ, base(n,M,AO) = base(T) and diagram(T) results from $\text{into}_C \circ H_{|\text{base}(n,AO)}$ by replacing node name n by n_n.

Fact 3.5

Let T = n{(m,h_m,f(m) | $m \in M$} be a direct application term in H, $m_o \in M$ minimal in M, f(m_o) = m_o and $h_{mo} = id_{H(mo)}$. Then

T´= n{(m,h_m,f(m)) | $m \in M-\{mo\}$ }

is a direct application term in H with

- application-object(T) = application-object(T´)
- base(T) = base(T´).

Proof: diagram(T´) results from diagram(T) by ´merging` the two nodes m_o and m_{om}, which is justified by the fact that in diagram(T) both are labelled by H(m_o) and there is an identity morphism between them.

Definition 3.4 [reduced application term]

A direct application term T is reduced iff there is no minimal m_o as in Fact 3.5.

Successive applications of Fact 3.5 yield a function reduce taking direct application terms to reduced direct application terms.

Fact 3.6

Let $T = n\{(m,h_m,f(m)) \mid m\varepsilon M\}$ be a direct application term. Then

$$\underline{\text{reduce}(T)} := n\{(m,h_m,f(m)) \mid m\varepsilon M'\}$$

with

$$M' = M-\{m\varepsilon M \mid f(m) = m,\ h_m = id_{H(m)},\ (\forall m'\varepsilon M.\ m'<m \Rightarrow (f(m')=m' \text{ and } h_{m'}=id_{H(m')}))\}$$

is a reduced direct application term with

- application-object(T) = application-object(reduce(T))
- base(T) = base(reduce(T)).

Proof: Follows from Fact 3.5. by an easy induction.

Similarly, a direct application term T may be extended by viewing all nodes used by n (with the exception of the minimal element) as formal parameters without affecting the denoted application object.

Fact 3.7

Let T be as in Fact 3.6.

$$\underline{\text{extend}(T)} := n\{(m,h_m,f(m)) \mid m\varepsilon M\} \cup \{(m,id_{H(m)},m) \mid m \varepsilon \text{base}(n,M,AO)-\{\bot\}\}$$

is a direct application term with

- application-object(T) = application-object(extend(T))
- base(T) = base(extend(T)).

Proof: Analogously to Facts 3.5 and 3.6.

4. Hierarchies with direct applications

4.1 Closed hierarchies

Since we consider ´application-object` to be an operation on the objects in a hierarchy, we are interested in hierarchies being closed under this operation. First, we consider the case of direct applications.

Definition 4.1 [closed under direct applications]

A hierarchy $H:AO \to \vec{C}$ is closed under direct applications iff for every direct application term T in H there exists a node $n \in O$ with:

- H(n) = application-object(T)
- base(n,ϕ,AO) = base(T)

A necessary condition for a hierarchy to be closed under direct applications is:

Definition 4.2 [direct application complete]

A hierarchy $H:AO \to \vec{C}$ is direct application complete iff for every direct application term T application-object(T) exists.

It should be noted that requiring both C and $\vec{C}$ to be finitely co-complete is not a sufficient condition for direct application completeness.

Fact 4.1

There is an appropriate category $(C,\vec{C})$ with C and $\vec{C}$ being finitely co-complete, and a hierarchy $H: AO \to \vec{C}$ such that H is not direct application complete.

Proof: Let C = SET be the category of sets, $\vec{C}$ = SET with set-theoretic inclusions as morphisms. AO is given by:

$$\begin{tikzcd}
& P \arrow[dl, dash] \arrow[d, dash] & \\
E \arrow[dr, dash] & F \arrow[d, dash] & A \arrow[dl, dash] \\
& \bot &
\end{tikzcd}$$

$H(\bot) = \phi$, $H(E) = H(A) = \{a\}$, $H(F) = \{b\}$, $H(P) = \{a,b\}$. Then a colimit object of diagram(T) with

$T = P\{(F,(b \to a),A)\}$

is $\{a_E, a_A\}$. However, there is no colimit object in SET such that the colimit injections from H(E) and H(A) are set-theoretic inclusions.

Sufficient conditions for application completeness will be given in Section 6.

4.2 Evaluation of direct application terms

Definition 4.1 requires the existence of a node labelled with the application object of a direct application term. Exploiting this fact one can get a function taking direct application terms to nodes in a hierarchy.

Fact 4.2

Let H: AO $\to \vec{C}$ be a hierarchy that is closed under direct applications. Then an evaluation function

$\mathrm{eval}_H : \{T \mid T \text{ is direct appl. term in } H\} \to O$

exists such that for every direct application term T

- $H(\mathrm{eval}_H(T))$ = application-object(T)
- $\mathrm{base}(\mathrm{eval}_H(T),\phi,AO) = \mathrm{base}(T)$

Proof: Immediately from Definition 4.1.

On the other hand, an evaluation function exists only for hierarchies closed under direct applications.

Fact 4.3

Let H be a hierarchy. If there exists an evaluation function $eval_H$, then H is closed under direct applications.

Proof: Immediately from Def. 4.1 and Fact 4.2.

For a hierarchy H there might be several functions $eval_H$ fulfilling the requirements of Fact 4.2. W.r.t. a specific evaluation function the composition of direct applications may be investigated.

4.3 Composition of direct applications

Two direct application terms may be composed if the result of the first one is the source of the second term.

Fact 4.4

Let $H: AO \to \vec{C}$ be a hierarchy with evaluation function $eval_H$, $AO = (O, <, \perp)$, $n_1, n_2 \in O$ and M_1 resp. M_2 direct parameter sets for n_1 resp. n_2. For i=1,2 let

- f_i: $base(n_i, \phi, AO) \to AO$
  be an appropriate order morphism with $\forall x \notin M_i \,.\, f_i(x) = x$
- h_i: $into_C \circ H|_{base(ni,\phi,AO)} \to into_C \circ H \circ f_i$
  be a natural transformation with: $\forall x \notin M_i \,.\, h_{ix} = id_{H(x)}$
- T_i be the direct application term for n_i, f_i and h_i.

If $eval_H(T_1) = n_2$ then the following holds:

(1) $M = M_1 \cup (M_2 \cap base(n_1, \phi, AO))$
is a direct parameter set for n_1

(2) $f = f_2 \circ f_1$ is an appropriate order morphism
$f: base(n_1, \phi, AO) \to AO$
with: $\forall\, x \notin M \,.\, f(x)=x$

(3) $h = \overbrace{h_2 \circ}^{*f_1} h_1$ is a natural transformation

$$h\colon \mathrm{into}_C \circ H|_{\mathrm{base}(n1,\phi,AO)} \to \mathrm{into}_C \circ H \circ f$$

with: $\forall\, x \notin M \,.\; h_x = id_{H(x)}$

(4) Let T be the direct application term for n_1, f and h. Then:

- application-object(T) $\cong$ H($\mathrm{eval}_H(T_2)$)
- base(T) = base($\mathrm{eval}_H(T_2)$,ϕ,AO)

<u>Proof</u>:

(1) Obviously, $M \subseteq \mathrm{base}(n_1,\phi,AO)$. It remains to be shown that for $m \varepsilon M$, $m' \varepsilon O$ $m < m' < n_1$ implies $m' \varepsilon M$.

- If $m \varepsilon M_1$ then $m' \varepsilon M_1 \subseteq M$ because M_1 is a direct parameter set for n_1.
- If $m \notin M_1$, we have $m \varepsilon M_2 \cap \mathrm{base}(n_1,\phi,AO)$.
  - If now $m' < n_2$ then $m' \varepsilon M_2$ because M_2 is a direct parameter set for n_2, and together with $m' \varepsilon \mathrm{base}(n_1,\phi,AO)$ this implies $m' \varepsilon M$.
  - If on the other hand $m' \nless n_2$, we have $m' \notin \mathrm{base}(T_1)$, since $\mathrm{eval}_H(T_1) = n_2$ and therefore $\mathrm{base}(T_1) = \mathrm{base}(n_2,\phi,AO)$. But $m' \notin \mathrm{base}(T_1)$ and $m' < n_1$ imply $m' \varepsilon M_1 \subseteq M$.

(2) holds because the composition of morphisms is a morphism.

(3) holds because the composition of natural transformations is again a natural transformation.

(4) We will use the following lemmas about diagrams:

<u>Lemma 1:</u> Let D′ be a subdiagram of D such that for every node n in D there exists a D-path to some node n′ in D′. If for a cocone $C = (c,\{c_n \mid n \varepsilon D\})$ of D, $C|_{D'} = (c,\{c_n \mid n \varepsilon D'\})$ is a colimit of D′, then C is a colimit of D.

<u>Proof</u>: (c.f. the illustration in Figure 4.1). Let C′ =

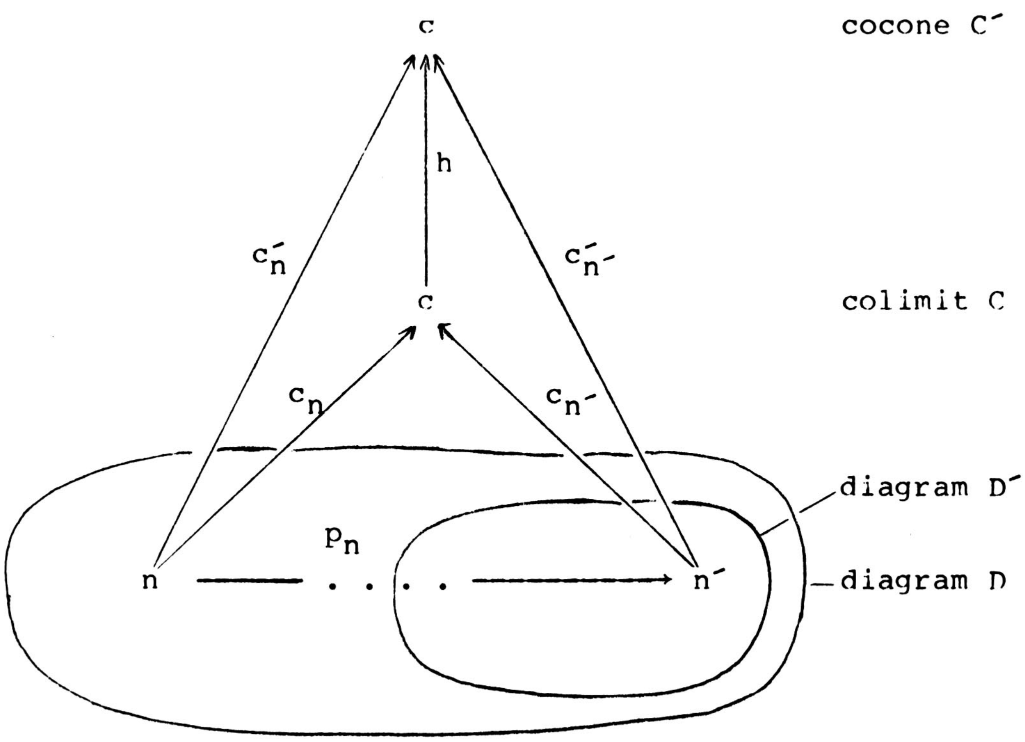


Figure 4.1: Illustration for the proof of Lemma 1 of Fact 4.4

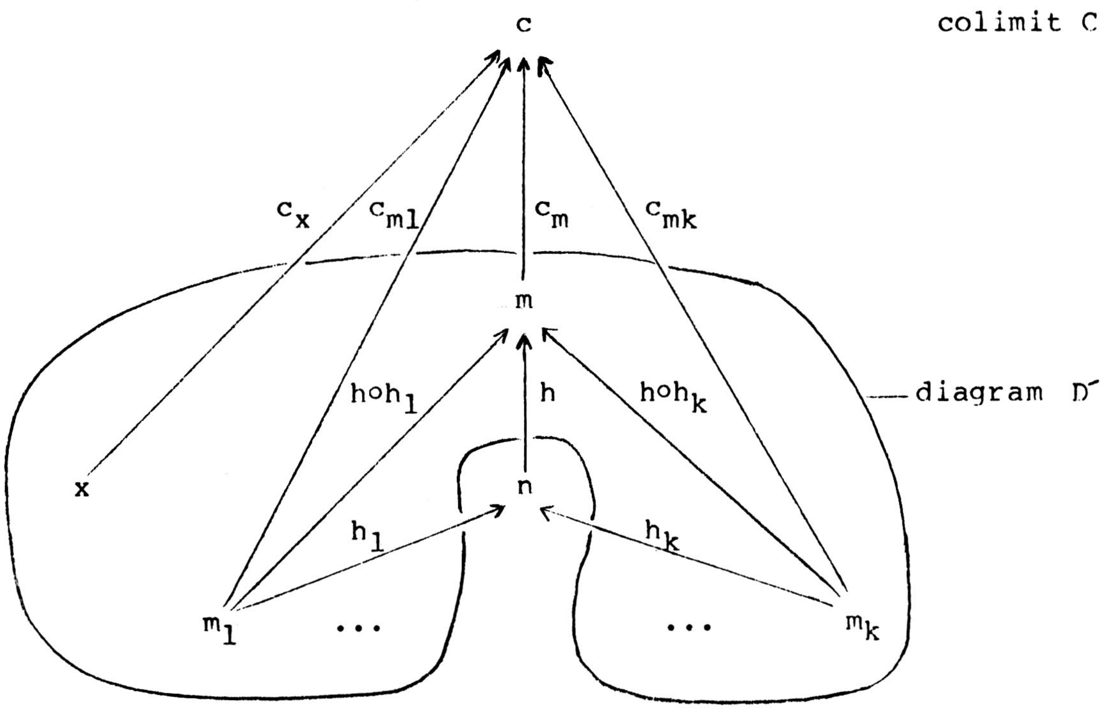


Figure 4.2: Illustration for the proof of Lemma 2 of Fact 4.4

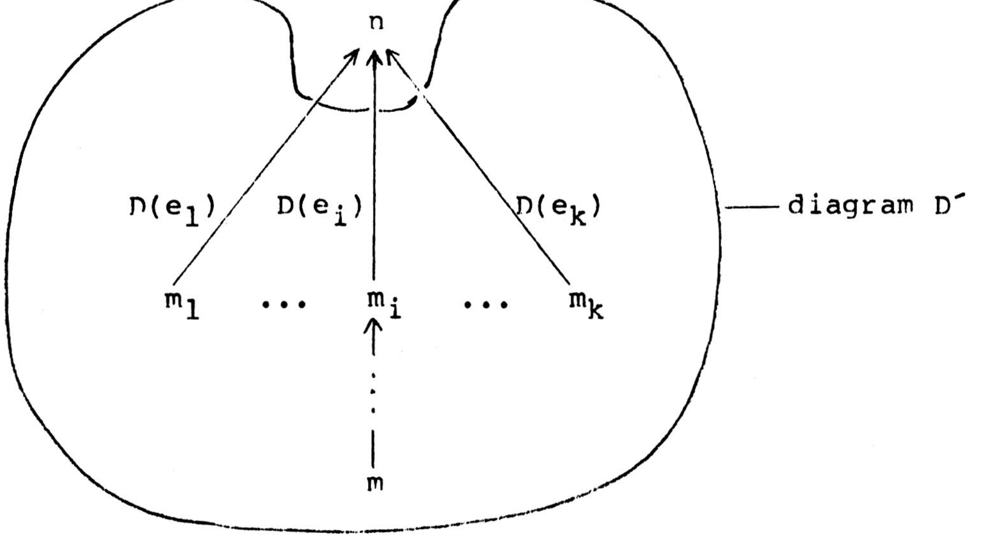


Figure 4.3: Illustration for Lemma 3 of Fact 4.4

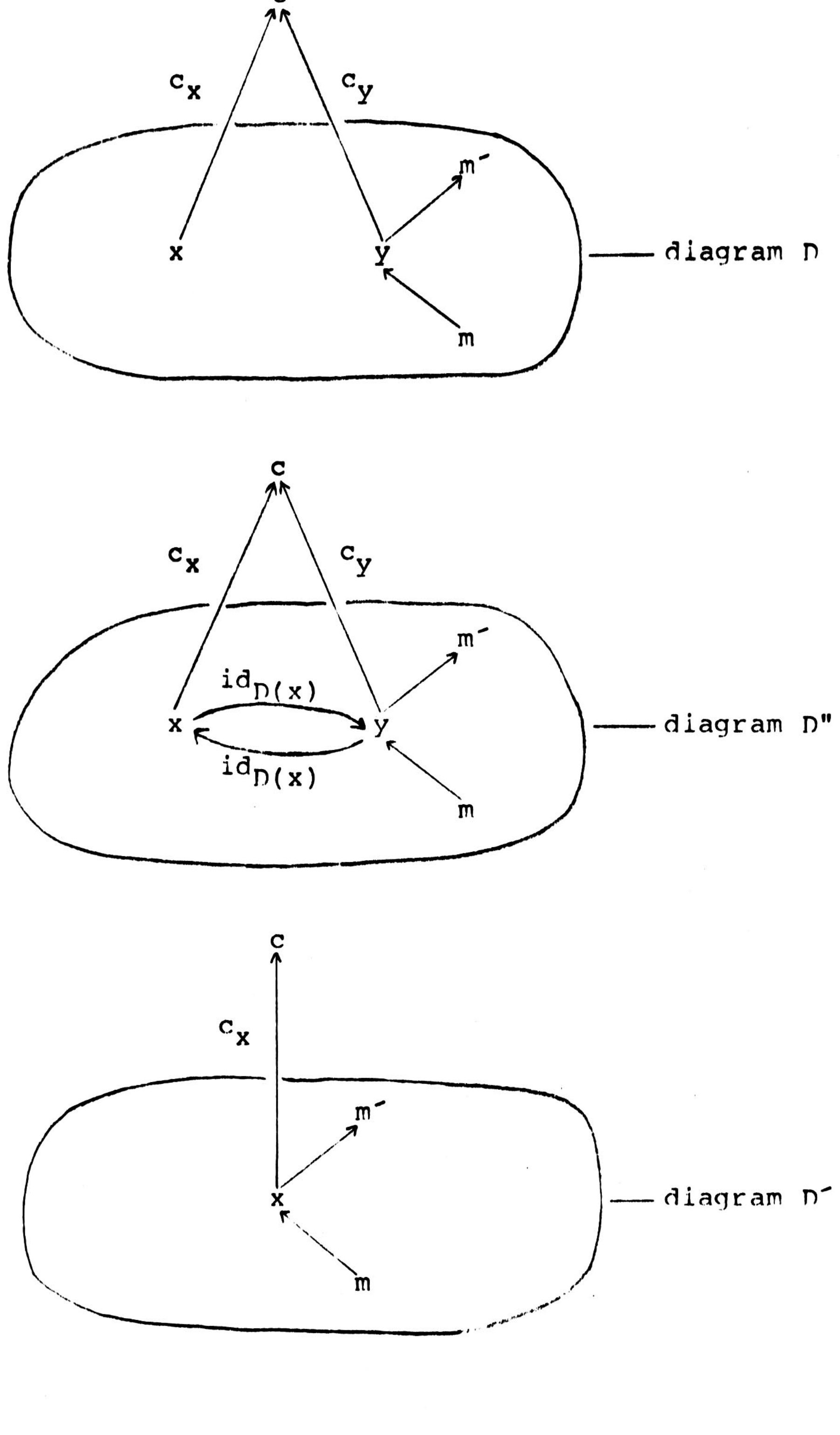


Figure 4.4: Illustration for the proof of Lemma 4 of Fact 4.4

$$\begin{tikzcd}
& n_{2n2} & & & & \\
m_{in2} \arrow[ur, hook] \arrow[rrrrr, dashed, "h_{mi}"] & & & & & f_2(m_i) \\
\cdots & & & & & \cdots \\
& & m_{jn2} \arrow[uuul, hook] \arrow[rr, dashed, "h_{mj}"] & & f_2(m_j) & \\
& & & b_j \arrow[ul, hook] \arrow[ur, hook] & & \\
& & \cdots & & & \\
& & b_i \arrow[uuuuull, hook] \arrow[uuuuurrr, hook] & & & \\
& & & \perp \arrow[ul, hook] \arrow[uuu, hook] & &
\end{tikzcd}$$

Figure 4.5: Diagram D_2 in the proof of Fact 4.4(4)

$$\begin{tikzcd}
 & n_{2n2} & & & & \\
m_{in2} \arrow[ur, hook] \arrow[rrrrr, dashed, "h_{2mi}"] & & & & & f_2(m_i) \\
\cdots & & & & & \cdots \\
 & m_{jn2} \arrow[uuu, hook] \arrow[rrr, dashed, "h_{2mj}"] & & & f_2(m_j) & \\
b_{in2} \arrow[uuu, hook] \arrow[rrrrr, dashed, "id_{H(bi)}"] & & & & & b_i \arrow[uuu, hook] \\
\cdots & & & & & \cdots \\
 & b_{jn2} \arrow[uuu, hook] \arrow[rrr, dashed, "id_{H(bj)}"] & & & b_j \arrow[uuu, hook] & \\
 & & \perp \arrow[uuull, hook] \arrow[ul, hook] \arrow[urr, hook] \arrow[uuurrr, hook] & & &
\end{tikzcd}$$

Figure 4.6: Diagram D_3 in the proof of Fact 4.4(4)

$$\begin{tikzcd}
 & n_{1n1} \arrow[rrrr, dashed] & & & & n_{2n2} & & \\
m_{rn1} \arrow[ur, hook] \arrow[urrrrr, dashed] \arrow[rrr, dashed, "h_{1mr}"] & & & f_1(m_r) & m_{in2} \arrow[ur, hook] \arrow[rrr, dashed, "h_{2mi}"] & & & f_2(m_i) \\
\cdots & m_{sn1} \arrow[uu, hook] \arrow[r, dashed, "h_{1ms}"] & f_1(m_s) & \cdots & \cdots & m_{jn2} \arrow[uu, hook] \arrow[r, dashed, "h_{2mj}"] & f_2(m_j) & \cdots \\
b_{rn1} \arrow[uu, hook] \arrow[rrr, dashed, "id_{H(br)}"] & & & b_r \arrow[uu, hook] \arrow[uuurr, dashed, hook] & b_{in2} \arrow[uu, hook] \arrow[rrr, dashed, "id_{H(bi)}"] & & & b_i \arrow[uu, hook] \\
\cdots & b_{sn1} \arrow[uu, hook] \arrow[r, dashed, "id_{H(bs)}"] & b_s \arrow[uu, hook] & \cdots & \cdots & b_{jn2} \arrow[uu, hook] \arrow[r, dashed, "id_{H(bj)}"] & b_j \arrow[uu, hook] & \cdots \\
 & \perp_{T1} \arrow[uul, hook, bend left] \arrow[u, hook] \arrow[ur, hook] \arrow[uurr, hook, bend right] & & & & \perp_{T2} \arrow[uul, hook, bend left] \arrow[u, hook] \arrow[ur, hook] \arrow[uurr, hook, bend right] & &
\end{tikzcd}$$

Figure 4.7: Diagram D_4 in the proof of Fact 4.4(4)

$$\begin{tikzcd}
 & n_{1n1} \arrow[rr, dashed] & & n_{2n2} & & \\
m_{rn1} \arrow[ur, hook] \arrow[urrr, dashed] \arrow[rr, dashed, "h_{1mr}"] & \cdots & f_1(m_r)_{n2} \arrow[ur, hook] \arrow[rrr, dashed, "h_{2f1(mr)}"] & \cdots & \cdots & f_2(f_1(m_r)) \\
 & m_{sn1} \arrow[uu, hook] \arrow[rr, dashed, "h_{1ms}"] & & f_1(m_s)_{n2} \arrow[uu, hook] \arrow[r, dashed, "h_{2f1(ms)}"] & f_2(f_1(m_s)) & \\
b_{rn1} \arrow[uu, hook] \arrow[rr, dashed, "id_{H(br)}"] & \cdots & b_{rn2} \arrow[uu, hook] \arrow[rrr, dashed, "h_{2br}"] & \cdots & \cdots & f_2(br) \arrow[uu, hook] \\
 & b_{sn1} \arrow[uu, hook] \arrow[rr, dashed, "id_{H(bs)}"] & & b_{sn2} \arrow[uu, hook] \arrow[r, dashed, "h_{2bs}"] & f_2(b_s) \arrow[uu, hook] & \\
 & & \perp \arrow[uull, hook] \arrow[ul, hook] \arrow[uu, hook] \arrow[ur, hook] \arrow[uurr, hook] \arrow[uurrr, hook] & & &
\end{tikzcd}$$

Figure 4.8: Diagrams D_5 and D_6 (eliminate n_{2n2}) in the proof of Fact 4.4(4)

$$\begin{tikzcd}
 & n_{1n1} & & & & \\
m_{rn1} \arrow[ur, hook] \arrow[rrrrr, dashed, "h_{2f1(mr)} \circ h_{1mr}"] & \cdots & & & \cdots & f_2(f_1(m_r)) \\
 & m_{sn1} \arrow[u, hook] \arrow[rr, dashed, "h_{1ms}"] & & f_1(m_s)_{n2} \arrow[r, dashed, "h_{2f1(ms)}"] & f_2(f_1(m_s)) & \\
b_{rn1} \arrow[uu, hook] \arrow[rr, dashed, "id_{H(br)}"] & & b_{rn2} \arrow[rrr, dashed, "h_{2br}"] & & & f_2(br) \arrow[uu, hook] \\
\cdots & b_{sn1} \arrow[uu, hook] \arrow[rr, dashed, "id_{H(bs)}"] & \cdots & b_{sn2} \arrow[uu, hook] \arrow[r, dashed, "h_{2bs}"] & f_2(b_s) \arrow[uu, hook] & \cdots \\
 & & \perp \arrow[uull, hook] \arrow[ul, hook] \arrow[uu, hook] \arrow[ur, hook] \arrow[urr, hook] \arrow[uurrr, hook] & & &
\end{tikzcd}$$

Figure 4.9: Diagram D_7 in the proof of Fact 4.4(4)

$$\begin{tikzcd}
n_{1n1} & & & \\
m_{rn1} \arrow[u, hook] \arrow[rrr, dashed, "h_{2f1(mr)} \circ h_{1mr}"] & & & f_2(f_1(m_r)) \\
 & \cdots & \cdots & \\
 & m_{sn1} \arrow[uuul, hook] \arrow[r, dashed, "h_{2f1(ms)} \circ h_{1ms}"] & f_2(f_1(m_s)) & \\
b_{rn1} \arrow[uuu, hook] \arrow[rrr, dashed, "h_{2f1(br)} \circ h_{1br}"] & & & f_2(f_1(b_r)) \arrow[uuu, hook] \\
 & \cdots & \cdots & \\
 & b_{snl} \arrow[uuu, hook] \arrow[r, dashed, "h_{2f1(bs)} \circ h_{1bs}"] & f_2(f_1(b_s)) \arrow[uuu, hook] & \\
 & \perp \arrow[u, hook] \arrow[uuul, hook, bend left] \arrow[ur, hook] \arrow[uuurr, hook, bend right] & &
\end{tikzcd}$$

Figure 4.10: Diagram D_8 in the proof of Fact 4.4(4)

$(c', \{c'_n | n \in D\})$ be a cocone of D. Obviously, $C'|_D = (c', \{c'_n | n \in D'\})$ is a cocone of D'. Thus, there exists a unique colimit-to-cocone morphism $h: c \to c'$ with

(1) $\forall n \in D'.\ h \circ c_n = c'_n$

For $n \in D-D'$ let p be a path from n to $n' \in D'$ with a path morphism p_n.

(2) $c'_{n'} \circ p_n = c'_n$ [C' is cocone of D]

(3) $c_{n'} \circ p_n = c_n$ [C is cocone of D]

(4) $h \circ c_{n'} \circ p_n = c'_n$ [(1) and (2)]

(5) $h \circ c_n = c'_n$ [(3) and (4)]

Thus, (1) and (5) imply that $h \circ c_n = c'_n$ for every $n \in D$. Since h is unique, c is a colimit of D.

Lemma 2: If diagram D results from D' by adding a new node n with exactly one outgoing edge e from n to m labelled with h, and incoming edges $e_1, \ldots, e_k$, $k>1$, from m_i to n labelled by h_i such that for all $i \in \{1, \ldots, k\}$ there is a path in D' from m_i to m labelled with $h \circ h_i$, then the colimit of D' with colimit injections c_x is a colimit of D as well, where the colimit injection c_n for the new node n is given by $c_m \circ h$.

Proof: (c.f. the illustration in Figure 4.2). We show that the colimit of D' together with c_n is a cocone of D; the rest follows from Lemma 1. To show the cocone property, consider paths from a node x into the colimit of D'. If x=n, every path other than c_n must go through m, and thus the path morphism must be $c_m \circ h = c_n$. If a path goes through n, it must go through some m_i, thus we may assume $x=m_i$. The path morphism must be $c_m \circ h \circ h_i$ or $c_n \circ h_i$, but these are identical since $c_m \circ h = c_n$. Finally, if a path does not go through n, its morphism is given by the D'-colimit injection c_x.

__Lemma 3:__ Let n be a node in a diagram D with no outgoing edges and incoming edges $e_1,\dots,e_k$. If D(n) together with injections $D(e_i)$ is a colimit of the subdiagram of D determined by all nodes m such that there is a path from m to n, then the colimit of D is also a colimit of D´ resulting from D by eliminating n and all incoming edges, (c.f. the illustration in Figure 4.3).

__Proof:__ Immediately.

__Lemma 4:__ If for two nodes x and y in a diagram D with D(x) = D(y) the colimit injections c_x and c_y are identical, then the colimit of D is also a colimit of D´ resulting from D by eliminating y and replacing all incoming (resp. outgoing) edges for y by incoming (resp. outgoing) edges for x.

__Proof__: (c.f. the illustration in Figure 4.4). D may be transformed into a diagram D" by adding two edges from x to y and from y to x labelled by $id_{D(x)}$ without affecting the colimit. Then merging x and y in the above sence results in D´ and obviously does not change the colimit either.

To prove (4), we show that application-object(T_2) = application-object(T) by colimit preserving transformations of D_2 = diagram(T_2) into $D_3,\dots,D_8$ and finally into diagram(T). The colimit injections for all x ε base(T_2) will be inclusions for every D_i. The diagrams D_i are illustrated in Figures 4.5 - 4.10, starting with D_2 in Figure 4.5.

1. D_3 := diagram(extend(T_2)) (c.f. Fact 3.7)
   For an illustration of D_3 see Figure 4.6.

2. D'_4 results from D_3 by taking the __disjoint__ union with diagram(extend(T_1)). (Whenever the nodes are unique,

we will omit the index T_1 resp. T_2.) D_4 results from D'_4 by additional edges from all nodes x of diagram(extend(T_1)) into n_{2n2} labelled by the colimit injection from $H(x) = D_4(x)$ into $H(n_2) = D_4(n_{2n2})$. Since the colimit of D_3 is easily shown to be a cocone of D_4 by taking as additional cocone injections the composition of the injections into $H(n_2)$ and the colimit injection from $H(n_2)$ into the colimit of D_3, Lemma 1 applies because D_3 is a subdiagram of D_4 and there is a path from every node in D_4-D_3 into n_{2n2}.
(c.f. Figure 4.7)

3. The next step consists of merging actual parameter nodes from T_1 with formal parameter nodes from T_2. Let $B = \text{base}(T_1) = \text{base}(n_2,\phi,AO)$. Since $D_4(n_{2n2})$ is an application object for T_1, the colimit injection from $D_4(b_{T1})$ into $D_4(n_{2n2})$ is an inclusion for every $b\varepsilon B$. Furthermore, since $D_4(b_{T1}) = D_4(b_{n2})$ and the morphism from $D_4(b_{n2})$ into $D_4(n_{2n2})$ is an inclusion as well, the colimit injections from $D_4(b_{T1})$ and $D_4(b_{n2})$ into the colimit of D_4 must be identical for $b \neq \perp$. The same argument applies to $D_4(\perp_{T1})$ and $D_4(\perp_{T2})$. D_5 results from D_4 by successive applications of Lemma 4 thereby eliminating all b_{T1} for $b\varepsilon B$.
Note: Since all node names in D_5 are unique, we will omit the indices T_1 resp. T_2 in the following.
(c.f. Figure 4.8)

4. D_6 results from D_5 by eliminating n_{2n2} and all incoming edges. Lemma 3 justifies this step, since $D_5(n_{2n2})$ is a colimit of the subdiagram of D_5 determined by all nodes x having a path to n_{2n2} - this subdiagram is identical to diagram(extend(T_1)) w.r.t. the renaming of nodes given by x for x_{n2}.
(c.f. Figure 4.8)

5. Let b be a node maximal w.r.t. AO such that b_{n2} is in D_6. The edge from b_{n2} to $f_2(b)$ is the only outgoing edge for b_{n2} in D_6. There is an incoming edge from node x_{n1} if $f_1(x) = b$. There is at least one incoming edge from $\perp$ resp. nodes x_{n2} for $x \in base(b,\phi,AO)$. D'_7 results from D_6 by eliminating b_{n2} according to Lemma 2. D_7 results from D'_7 by deleting the newly added edges from nodes with index n2 to $f_2(b)$. For such nodes x_{n2}, the D'_7 edge label from x_{n2} to $f_2(b)$ is given by the composition of the inclusion H(x) into H(b) and h_{2b}, and due to the natural transformation property of h_2, it is identical to the composition of h_{2x} and the inclusion $H(f_2(b)) = D'_7(f_2(b))$ into $H(f_2(b)) = D'_7(f_2(b))$. Thus, the colimits of D'_7 and D_7 are identical.
(c.f. Figure 4.9)

6. Since either there is a maximal node b_{n2} in D_7 as above or there is no node with index n2 in D_7, D_8 results from D_7 by successively applying Lemma 2 and eliminating all nodes with index n2 according to step 5.
(c.f. Figure 4.10)

7. D_8 is exactly diagram(extend(T)). The colimit preserving transformations from D_2 to D_8 imply that base(T_2) = base(extend(T)) and application-object(T_2) = application-object(extend(T). The rest follows from Fact 3.7, completing the proof of Fact 4.4.

As a result of Fact 4.4 we can define the composition of direct application terms.

Definition 4.3 [composition of direct application terms]

Let H, T_1, T_2, T as in Fact 4.4. T is called the composition of T_1 and T_2 and is denoted by $T_1 \circ T_2$.

Definition 4.4. [respecting direct application composition]

The evaluation function $eval_H$ of a hierarchy H respects direct application composition iff for all T,T´, such that T $\circ$ T´ is defined,

$$eval_H(T') = eval_H(T \circ T')$$

holds.

Fact 4.5

If H is a hierarchy with evaluation function $eval_H$ respecting direct application composition then direct application composition is associative, i.e.

$$(T_1 \circ T_2) \circ T_3 = T_1 \circ (T_2 \circ T_3)$$

for all T_i such that $T_1 \circ T_2$ and $T_2 \circ T_3$ are defined.

Proof:

Let $T_i = n_i\{(m,h_{im},f_i(m)) \mid m \in M_i\}$ for i=1,2,3.

$$(T_1 \circ T_2) \circ T_3 = n_1\{(m,h_{3f2(f1(m))} \circ (h_{2f1(m)} \circ h_{1m}), f_3(f_2(f_1(m)))) \mid m \in M_{(1,2)3}\}$$

$$T_1 \circ (T_2 \circ T_3) = n_1\{(m,(h_{3f2})_{f1(m)} \circ h_{2f1(m)}) \circ h_{1m}), f_3 \circ f_2(f_1(m))) \mid m \in M_{1(2,3)}\}$$

Since composition of morphisms and functions is associative, $M_{(1,2)3} = M_{1(2,3)}$ remains to be shown.

$$\begin{aligned} M_{1(2,3)} &= M_1 \cup ((M_2 \cap (M_3 \cup base(n_2,\phi,AO)) \cap base(n_1,\phi,AO)) \\ &= M_1 \cup (M_2 \cap base(n_1,\phi,AO)) \\ &\qquad \cup (M_3 \cap base(n_2,\phi,AO) \cap base(n_1,\phi,AO)) \\ &= [\text{since } base(n_1,\phi,AO) - base(n_2,\phi,AO) \subseteq M_1] \\ &\quad M_1 \cup (M_2 \cap base(n_1,\phi,AO)) \cup (M_3 \cap base(n_1,\phi,AO)) \\ &= M_{(1,2)3}. \end{aligned}$$

5. Generalized applications

5.1. Indirect applications

In Definition 3.1 parameter sets and direct parameter sets were introduced. As discussed in Sections 3.3 and 4 we have the following situation in the case of direct parameter sets: every node that is used by the node n is either a formal parameter and is therefore actualized by a corresponding actual parameter, or it is not affected by the application at all, i.e. it is used by the application object as well. However, for an arbitrary parameter set M a third case may arise. If n uses a node m which is not in the parameter set but uses itself a parameter node m´ in M, m is between n and M in AO. Since no actual parameter is associated to m and setting f(m) = m may be meaningless we exclude the nodes between n and M from the domain of f and h in generalizing Fact 3.2:

Fact 5.1

Let $H:AO \to \hat{C}$ be a hierarchy, $n \in O$, M a parameter set for n, $M' = \text{between}(n,M,AO)$. Let

$f: \text{base}(n,M',AO) \to AO$

be an appropriate order morphism and

$h: \text{into}_C \circ H|_{\text{base}(n,M',AO)} \to \text{into}_C \circ H \circ f$

a natural transformation with $f(x) = x$ and $h_x = id_{H(x)}$ for all $x \notin M$. Then for all $x,y \in O-M'$ with $x<y<n$ the diagram

$$\begin{tikzcd} H(y) \arrow[r, dashed, "h_y"] & H(f(y)) \\ H(x) \arrow[u, hook] \arrow[r, dashed, "h_x"] & H(f(x)) \arrow[u, hook] \end{tikzcd}$$

commutes in C.

Proof: For $x,y \in O-M'$ $x<y<n$ implies $x,y \in \text{base}(n,M',AO)$ and the commutativity of the diagram represents the natural

transformation property of h.

Since f and h of Fact 5.1 are determined by giving their values for $m \in M$ it suffices to supply f(M) and h(M), (c.f. Figure 5.1).

Definition 5.1 [indirect application term]

Let H,n,M and f,h be as in Fact 5.1. Then

$$T = n\{(m, h_m, f(m)) \mid m \in M\}$$

is an indirect application term.

An example of an indirect application term is given in Figure 5.2-a.

In specification languages the situation of what we call an indirect application term is usually not considered explicitly. The parts of a parameterized specification corresponding to the nodes above a parameter set are not viewed to be hierarchically structured; instead they are combined into a single object as are all formal resp. actual parameters. The semantics of the application is the pushout of the resulting diagram (e.g. [BG 80], [EKTWW 80], [Eh 82], [Eh 81]).

Here we choose a different approach. The structure between the nodes above the parameter set should be preserved in the result of the application. This can be achieved by providing an actual parameter for every $m \in$ between(n,M,AO). Fact 5.2 shows how a fitting actual parameter can be found for such a minimal m.

Fact 5.2

Let T be an application term as in Def. 5.1. Then for all minimal $m \in$ between(n,M,AO)

$$T_m = m\{(m', h_{m'}, f(m')) \mid m' \in M \cap \text{base}(m, AO)\}$$

is a direct application term.

Proof: Because of the minimality of m, $M' = M \cap \text{base}(m,AO)$ is a direct parameter set for m and the restrictions of h and f to

$$\begin{tikzcd}
 & n & & & & & & \\
 & & n_1 \arrow[ul, hook] & & & & & \\
 & n_2 \arrow[ur, hook] & & & & & & \\
 & & & n_3 \arrow[uul, hook] & & & & \\
 & m_1 \arrow[uu, hook] \arrow[rrrrr, dashed, "h_{m1}"] & & & & f(m_1) & & \\
 & & & m_2 \arrow[uu, hook] \arrow[rrrr, dashed, "h_{m2}"] & & & & f(m_2) \\
 & & m_3 \arrow[uul, hook] \arrow[ur, hook] \arrow[rrrr, dashed, "h_{m3}"] & & & & f(m_3) \arrow[uul, hook] \arrow[ur, hook] & \\
b_1 \arrow[uuuuuuur, hook] & & & & & & & \\
 & & & & b_2 \arrow[uull, hook] \arrow[uurr, hook] & & b_3 \arrow[uuulll, hook] \arrow[uuur, hook] & b_4 \arrow[uuu, hook] \\
 & & & & \perp \arrow[uullll, hook] \arrow[u, hook] \arrow[urr, hook] \arrow[urrr, hook] & & &
\end{tikzcd}$$

Figure 5.1: f and h for the non-direct parameter set
$M = \{m_1, m_2, m_3\}$ of n (c.f. Fact 5.1)

(a) $T = n\{(m_i, h_{mi}, f(m_i)) \mid i \varepsilon \{1,2,3\}\}$

(b) $T_{n2} = n_2\{(m_i, h_{mi}, f(m_i)) \mid i \varepsilon \{1,2\}\}$

$T_{n3} = n_3\{(m_i, h_{mi}, f(m_i)) \mid i \varepsilon \{1,3\}\}$

(c) $T_{\{n2,n3\}} = n\{(m_i, h_{mi}, f(m_i)) \mid i \varepsilon \{1,2,3\}$
$\quad u \; \{(n_i, h'_{ni}, eval_H(T_{ni})) \mid i \varepsilon \{2,3\}\}$

(d) $direct_i(T) = n\{(m_i, h_{mi}, f(m_i)) \mid i \varepsilon \{1,2,3\}\}$
$\quad u \; \{(n_i, h'_{ni}, eval_H(T_{ni})) \mid i \varepsilon \{1,2,3\}\}$

<u>where</u>:
$T_{n1} = n_1\{(m_i, h_{mi}, f(m_i)) \mid i \varepsilon \{1,2,3\}\}$
$\quad u \; \{(n_i, h'_{ni}, eval_H(T_{ni})) \mid i \varepsilon \{2,3\}\}$

Figure 5.2: Application terms for f and h as given in Figure 5.1, illustrating Def. 5.1 (a), Fact 5.2 (b), Fact 5.3 (c), and Def. 5.4 (d).

M' obviously fulfill the conditions of Fact 3.2.

Examples illustrating Fact 5.2 are given in Figure 5.2-b.

If h is closed under direct applications $T_{m'}$ may be evaluated to a fitting actual parameter for m.

Fact 5.3

Let T, T_m be as before and H be closed under direct applications. Let M' be a set of minimal elements in the set between(n,M,AO). Then

$$T_M = n\{(m,h_m,f(m)) \mid m\in M\} \cup \{(m,h'_m,eval_H(T_m)) \mid m\in M'\}$$

is an indirect application term where

$$h'_m: H(m) \to \text{application-object}(T_m)$$

is the colimit injection.

Proof: Since all elements in M' are minimal, $M \cup M'$ is a parameter set for n. Suppose $M' = \{m\}$. Extending f by sending m to $eval_H(T_m)$ is an appropriate order morphism since $\{x\in O \mid x<m\} = base(m,\phi,AO)$, $f(base(m,\phi,AO)) \subseteq base(T_m)$, and $eval_H$ is an evaluation function for H. Furthermore since h'_m is a colimit injection for diagram(T_m) and the application object of T_m has inclusions as colimit injections for all $m' \in base(T_m)$, extending h by h'_m yields a natural transformation fulfilling the conditions of Fact 5.1. The general case with M' containing more than one element follows by an easy induction on $|M'|$.

An example illustrating Fact 5.3 is given in Figure 5.2-c.

Fact 5.3 allows the definition of a function $direct_i$ taking indirect application terms to direct application terms.

Definition 5.2 [$direct_i$]

Let H, T, T_M be as in Fact 5.3. The function

$\underline{direct_i}$: $\{T \mid T$ is indirect appl. term of $H\} \to$

$\{T \mid T \text{ is direct appl. term of } H\}$

is defined by:

```
<u>direct_i(T)</u> :=
  <u>if</u> T is direct application term
     <u>then</u> T
     <u>else</u> <u>let</u> $M' = \{m \mid m \text{ is minimal in between}(n,M,AO)\}$ <u>in</u>
        $direct_i(T_{M'})$
```

As an example the direct application term $direct_i(T)$ for an indirect application term T is given in Figure 5.2-d.

The semantics of any indirect application term T is the semantics of $direct_i(T)$. Thus, the hierarchical structure of the nodes between n and M is mirrored by the corresponding actual parameters in $direct_i(T)$. On the other hand, this approach is compatible with the simpler semantics when viewing the objects as non-hierarchical:

<u>Fact 5.4</u>

Let H be a hierarchy closed under direct applications, T an indirect application term. Let diagram(T) be the C-diagram defined as in Def. 3.3. Then:

$$H(eval_H(direct_i(T))) \cong colim(diagram(T)).$$

<u>Proof:</u> For every indirect application term T as in Definition 5.1 let levels(T) be the depth of recursion in determining $direct_i(T)$. We prove 5.4 by induction on k = levels(T).

<u>k=0:</u> Since between(n,M,AO) is empty and since H is an evaluation function, we have $direct_i(T) = T$ and $H(eval_H(T))$ = application-object(T) $\cong$ colim(diagram(T)).

<u>k>0:</u> Suppose that the proposition is proved for every indirect application term T with levels(T) = k.

<u>k+1:</u> Let T be an indirect application term with levels(T) =

k+1, and let $T_{M'}$ be as in Definition 5.2. Thus

$$H(eval_H(direct_i(T))) = H(eval_H(direct_i(T_{M'})))$$

by the definition of $direct_i$. Furthermore

$$H(eval_H(direct_i(T_{M'}))) \cong colim(diagram(T_{M'}))$$

by the induction hypothesis. We consider $D' = diagram(T_{M'})$ and $D = diagram(T)$. D' results from D by adding for every $m \in M'$

- nodes m_n and $eval_H(T_m)$, and an edge between them, labelled by $H(m)$, $H(eval_H(T_m))$, h'_m, respectively, where T_m and h'_m are as in Fact 5.3.
- edges from x_n to m_n, m_n to n_n, and y to $eval_H(T_m)$ for $x \in base(m,\phi,AO)$, $y \in base(eval_H(T_m),\phi,AO)$, labelled by the respective inclusions.

D' may be transformed into D" by successively applying Lemma 3 of Fact 4.4(4), thereby deleting the nodes $eval_H(T_m)$ for $m \in M'$. D" may be transformed into D by successively applying Lemma 2 of Fact 4.4(4), thereby deleting the nodes m_n for $m \in M'$. Thus,

$$colim(diagram(T_{M'}) \cong colim(diagram(T)),$$

completing the inductive step.

As an example Figure 5.3 shows the application diagrams for an indirect application term T and its corresponding direct application term $direct_i(T)$.

As in Section 4 we are interested in hierarchies where every indirect application term can be evaluated to a node in that hierarchy. The transformation of indirect to direct application terms immediately implies:

<u>Fact 5.5</u>

If a hierarchy H is closed under direct applications then H is closed under indirect applications as well.

<u>Proof</u>: Fact 5.3 and Definition 5.2.

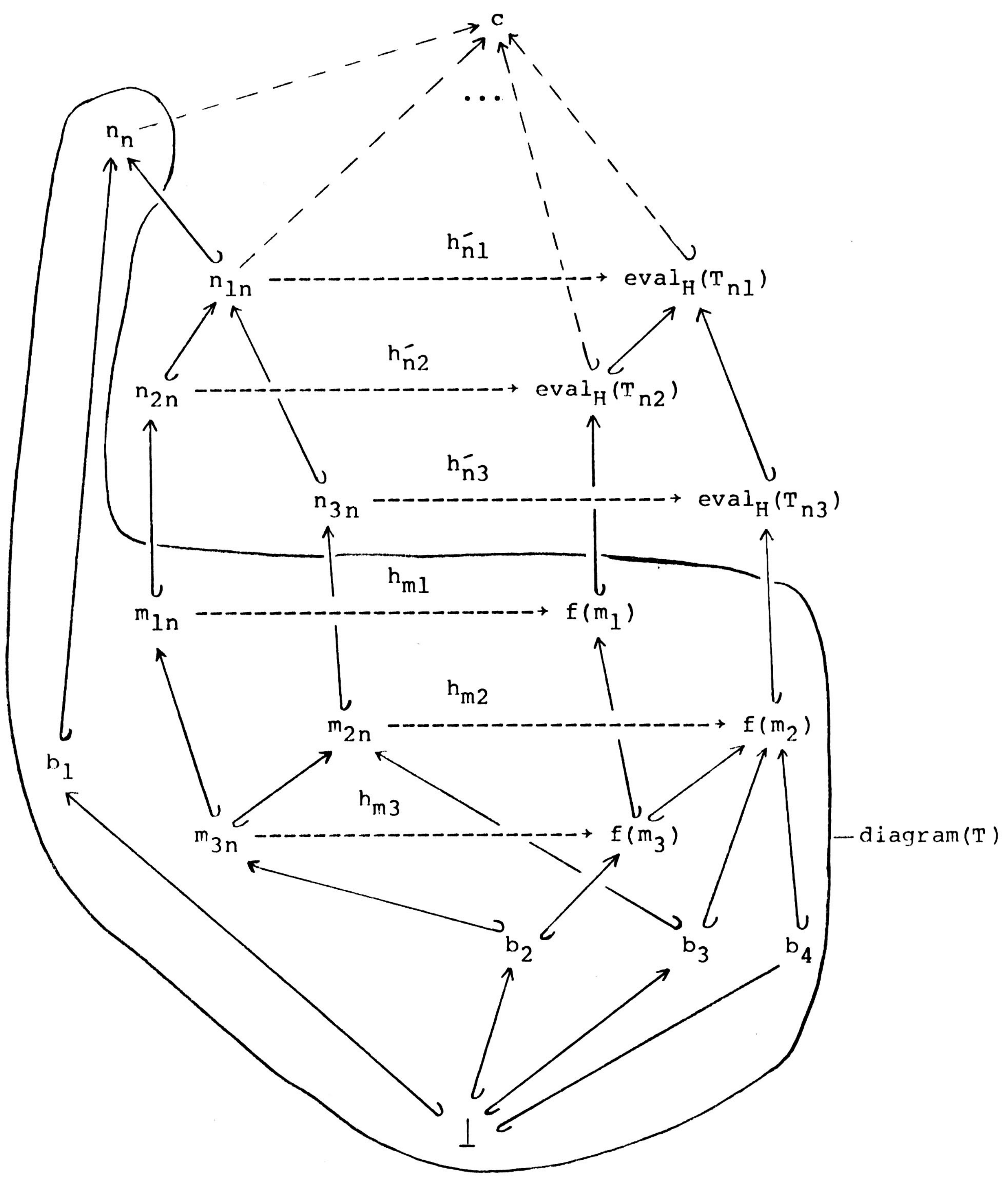


Figure 5.3: Application diagrams with application object c for the terms T and direct$_i$(T) as given in Figure 5.2 (c.f. Fact 5.4)

5.2 Application terms and their evaluation

So far we have introduced direct and indirect application terms. Besides calling every node $n \varepsilon O$ an application term as well, we will generalize the notion of application term in two directions:

1. In an application term $T = n\{...\}$ where n is a node in O, n could be substituted by an application term denoting n. By iterating this rule a chain of parameter actualization clauses can be generated yielding application terms of the form $T' = n'\{...\}...\{...\}$.

2. The observation made above for n applies to formal and actual parameter nodes m and f(m) as well. Substituting application terms denoting m resp. f(m) yields an application term of the form $n\{...,(m\{...\},h,m'\{...\}),...\}$.

In both cases the semantics of an application term is defined by the underlying simpler terms and eventually by some direct application term.

Definition 5.3 [application term]

Let H be a hierarchy closed under direct applications. 'Application term' of H and the function 'direct' taking application terms to direct application terms are defined inductively by 1.-4. below. The extension of the evaluation function $eval_H$ to all application terms is also denoted by $eval_H$ and is defined by

$eval_H(T) := eval_H(direct(T))$

for every application term T.

1. Every node $n \varepsilon O$ is an application term with

   $direct(n) := n.$

   (As before, $eval_H(n) = n$).

2. Every direct or indirect application term T is an application term with

   $\text{direct}(T) := \text{direct}_i(T)$.

3. If T, T' are application terms, $n \in O$, $T' = n\{t'_1, \ldots, t'_r\}$ and $\text{eval}_H(T) = n$ then

   $T'' = T\{t'_1, \ldots, t'_r\}$

   is an application term with

   $\text{direct}(T'') := \text{direct}(T) \circ \text{direct}(\text{eval}_H(T)\{t'_1, \ldots, t'_r\})$

   or equivalently $\text{direct}(T'') := \text{direct}(T) \circ \text{direct}(T')$.

4. If T, T_1, T_2 are application terms,

   $T = n\{t_1, \ldots, (m_1, h_{m1}, m_2), \ldots, t_r\}$

   and $\text{eval}_H(T_i) = m_i$ for $i=1,2$, then

   $T'' = n\{t_1, \ldots, (T_1, h_{m1}, T_2), \ldots, t_r\}$

   is an application term with

   $\text{direct}(T'') :=$

   $\text{direct}(n\{t_1, \ldots, (\text{eval}_H(T_1), h_{m1}, \text{eval}_H(T_2)), \ldots, t_r\})$.

Defining the semantics of arbitrary application terms by reducing them to direct application terms allows a generalization of Fact 5.5:

Fact 5.6

If H is closed under direct applications it is closed under all applications.

Proof: Fact 5.5 and Definition 5.3.

Thus it is sufficient to guarantee that a hierarchy is closed under direct applications. Therefore, when constructing the closure of a hierarchy only direct aplication terms must be considered.

6. Extension and closure of hierarchies

6.1 Canonical closure

Provided that all the necessary application objects exist, a hierarchy H could be transformed into a hierarchy H´ that is closed under direct and thus under all applications by enlarging the underlying appropriate order by new nodes and labelling them with the corresponding application objects. Enlarging a hierarchy H by a new node n with label c requires also a set of nodes determining the base of n.

Definition 6.1 [extension of a hierarchy]

Let $H: AO \to \vec{C}$ be a hierarchy, $n \notin O$, B a finite nonempty subset of O, $c \in C$ such that for every $m \in B$ there is an inclusion $H(m) \hookrightarrow c$. Then

enter(H,(n,B,c))

denotes the hierarchy $H´: AO´ \to \vec{C}$ resulting from H by adding n to AO such that

- $\forall m \in O.\ m <´ n$ iff $\exists\, m´ \in B.\ \ m \leq m´$
- $\forall m \in O.\ n \nless´ m$

and setting $H´(n) = c$.

If M is a set of triples (n,B,c) such that enter(H,(n,B,c)) is defined for all triples in M and all the nodes n in M are pairwise distinct, we will use the notation enter(H,M) for extending the hierarchy H by a set of new nodes.

Since we are interested in constructing hierarchies step by step we will distinguish the nodes of a hierarchy entered implicitly as application nodes from the other nodes: the latter ones are called extension nodes. This allows the definition of a special type of hierarchy called canonically closed hierarchy.

Definition 6.2 [canonically closed]

(1) A hierarchy $H: (\{\bot\},\phi,\bot) \to \vec{C}$ is a canonically closed

hierarchy with <u>extension set</u> $E = \{\bot\}$.
H is denoted by <u>initial-hierarchy($\bot$,c)</u> for $H(\bot) = c$.

(2) Let H'': $AO'' \to \vec{C}$ be a canonically closed hierarchy with extension set E'', $H' = \text{enter}(H'',(\text{new},B,c))$.
Then H: $AO \to \vec{C}$ is a <u>canonically closed</u> hierarchy with <u>extension set</u> $E = E'' \cup \{\text{new}\}$ where H is defined inductively:

$i=0$: $A_0 := \phi$
$H_0 := H$
$i>0$: $A_i := \{T \mid T = n\{(m,h_m,f(m) \mid m\varepsilon M\}$ is a reduced direct application term in H_{i-1}
<u>and</u> $n \varepsilon E$
<u>and</u> $i=1 \Rightarrow \text{new}\ \varepsilon\ (f(M) \cup \{n\})$
<u>and</u> $i>1 \Rightarrow \{"T'" \mid T'\varepsilon A_{i-1}\} \cap f(M) \neq \phi\}$
$H_i :=$ enter $(H_{i-1},\{("T", \text{base}(T), \text{application-object}(T)) \mid T\varepsilon A_i\})$

H is denoted by <u>closure (H')</u>.

<u>Fact 6.0</u>
Every node in a canonically closed hierarchy is either an extension node or a node of the form "n{...}" such that
- n is an extension node,
- n{...} is a reduced direct application term in H,
- H("n{...}") = application-object(n{...})
- base("n{...}",ϕ,AO) = base(n{...}).

<u>Proof</u>: Immediately from Def. 6.2.

We will show that a canonically closed hierarchy is indeed closed under direct applications. Furthermore, its evaluation function respects composition of direct application terms. Thus, application term composition is associative according to Fact 4.5.

Fact 6.1

Let H be a canonically closed hierarchy with extension set E. Then $eval_H$ defined by:

```
eval_H(T) := let T_1 = reduce (T), T_1 = n{...} in
                 if T_1 is trivial then n
                    else if n∈E then "T_1"
                    else let n ="T_2" in
                              eval_H (T_2 ∘ T_1)
```

is an evaluation function for direct application terms of H respecting composition of direct application terms.

Proof: We first show that $eval_H$ is an evaluation function. Let $T = n\{(m,h_m,f(m)) \mid m \in M\}$ be a direct application term of H. The definition of $eval_H$ implies $eval_H(T) = eval_H(reduce(T))$. Because of Fact 3.6 we can therefore assume that T is reduced. If $T = n\{\}$ is trivial, $eval_H(n) = n$ and Fact 3.4 completes the proof. Thus, let T be reduced and non-trivial. Two cases arise: (1) $n \in E$ and (2) $n \notin E$ where E is the extension set of H.

(1) Suppose $n \in E$. We will show that "T"$\in$O, H("T") = application-object(T) and base("T",ϕ,AO) = base(T). The rest follows from $eval_H$(T)="T".
Since T is non-trivial, n cannot be the minimum element of AO. Therefore, there must be some canonically closed subhierarchy H":AO" $\to \vec{C}$ in the construction of H with $n \notin$O", B $\subseteq$ O" for B=base(n,ϕ,AO), and an inclusion from H"(b) into c for every $b \in B$ and c=H(n), such that H´ = enter(H",(n,B,c)) is a subhierarchy of H.

(1.1) If f(M) $\subseteq$ O´, then T is a reduced direct application term of H´ and in the inductive step of closing H´ in Def. 6.2, T $\in A_1$. Thus, enter(H´,("T",base(T), application-object(T))) is a subhierarchy of H.

(1.2) If $f(M) \not\subseteq O'$, there must be some canonically closed subhierarchy H_1'' of H, such that

- H' is a subhierarchy of H_1''
- $f(M) \not\subseteq O_1''$
- $H_1' = \text{enter}(H_1'',(\text{new},B',c'))$ is a subhierarchy of H
- H_1, the canonically closed hierarchy of H_1', is a subhierarchy of H
- $f(M) \subseteq O_1$.

Let $A_{1,i}$ (resp. $H_{1,i}$, $O_{1,i}$) be the sets of direct application terms (resp. hierarchies, sets of nodes) generated in the process of closing H_1' according to Def. 6.2. Since $f(M) \not\subseteq O_1''$, $f(M) \subseteq O_1$, and $O_1'' \subseteq O_{1,o} \subseteq O_{1,1} \subseteq \ldots \subseteq O_1$ there must be a minimal i such that $f(M) \subseteq O_{1,i-1}$. T is a reduced direct application term in $H_{1,i-1}$. We show that $T_i \in A_{1,i}$, concluding that enter(H',("T", base(T), application-object(T))) is a subhierarchy of H. If $i=1$, then new ε f(M) since $O_{1,o} = O_1'' \cup \{\text{new}\}$; thus $T \in A_{1,1}$. If $i>1$, we have $f(M) \cap \{"T'" \mid T' \in A_{1,i-1}\} \neq \phi$, since $O_{1,i-1} = O_{1,i-2} \cup \{"T'" \mid T' \in A_{1,i-1}\}$ and $f(M) \subseteq O_{1,i-1}$, but $f(M) \not\subseteq O_{1,i-2}$ by the minimality of i. Thus, $T \in A_{1,i}$.

(2) Suppose $n \notin E$. Because of Fact 6.0, n = "T´" for some reduced direct application term of the form $T' = n'\{\ldots\}$. Since both T and T´ are direct application terms in H and $\text{eval}_H(T') = n$, the preconditions of Fact 4.4 are fulfilled and $T'' := T' \circ T$ is a direct application term in H with application-object(T") = application-object(T) and base(T") = base(T). Since $\text{eval}_H(T) = \text{eval}_H(T'')$ we only have to show that eval_H yields the proper value for T". But this follows from (1), since $n' \in E$ according to Fact 6.0, completing the proof that eval_H is an evaluation function.

To show that $eval_H$ respects direct application composition we observe that reduce $(T \circ T')$ = reduce (reduce$(T) \circ$ reduce(T')) due to Facts 3.6 and 4.4. Since $eval_H(T) = eval_H(\text{reduce}(T))$, it suffices to show $eval_H(T \circ T') = eval_H(T')$ for reduced direct application terms T and T'. If T is trivial, $T \circ T' = T'$ and thus $eval_H(T \circ T') = eval_H(T')$. If T is non-trivial, let $T = n\{...\}$, $T' = n'\{...\}$. If $n \in E$ then $eval_H(T)$ = "T" and $eval_H(T') = eval_H(T \circ T')$ since n' = "T". Otherwise, if $n \notin E$ then n = "T_2" for some reduced nontrivial direct application term T_2. Thus $eval_H(T) = eval_H(T_2 \circ "T_2"\{...\}) = n_3$ with either $n_3 \in E$ or n_3 = "$n_4\{...\}$" with $n_4 \in E$. If $n_3 \in E$, the above argumentation for the case that T is trivial applies; if n_3 = "$n_4\{...\}$", the argumentation for the case that T is non-trivial and $n \in E$ is applicable, thereby completing the proof of Fact 6.1.

Fact 6.2

A canonically closed hierarchy is closed under direct applications.

Proof: Fact 4.3. and Fact 6.1.

6.2 Prefix hierarchies

In Fact 4.1 an example was given showing that not every hierarchy H is application complete and thus a canonical closure for H may not exist. Here we will investigate conditions under which a hierarchy can be closed. Recalling the definition of application object (Def. 3.3) one condition is essential: the existence of colimits for specific diagrams with certain colimit injections being inclusions.

Fact 6.3.

Let $H: AO \to \check{C}$ be a hierarchy, $M \subseteq O$. Then restricting H to the suborder AO' with the nodeset being the union of base(m,AO)

for $m \varepsilon M$ yields again a hierarchy, called <u>subhierarchy</u> of H and M and denoted by $H_{|M}$.

<u>Proof:</u> immediately from Def. 2.5 and 3.1.

<u>Definition 6.3</u> [subhierarchy-complete]

Let $H: AO \to \vec{C}$ be a hierarchy. H is <u>subhierarchy-complete</u> iff for all finite $M \subseteq O$ there is a colimit of $H_{|M}$ that is a colimit of $into_C \circ H_{|M}$ as well.

If a hierarchy H is subhierarchy-complete, a direct application diagram can be transformed into a pushout diagram due to the following fact.

<u>Fact 6.4</u>

Let $T = n\{(m,h_m, f(m)) \mid m \varepsilon M\}$ be a direct application term in $H: AO \to \vec{C}$ with H subhierarchy-complete.

<u>formal(T)</u> := $H_{|base(n,\phi,AO)}$

<u>actual(T)</u> := $H_{|f(base(n,\phi,AO))}$

Then:

(1) H(n) is a cocone object over $into_C \circ$ formal(T) and the unique morphism from some colimit of $into_C \circ$ formal(T) to cocone H(n) is an inclusion.

(2) The colimit object of actual(T) is a cocone over $into_C \circ$ formal(T).

<u>Proof</u>:

(1) Since H is a hierarchy, H(n) is a cocone oject over formal(T) and over $into_C \circ$ formal(T) as well. Since H is subhierarchy-complete, there is a c such that c is a colimit object of both formal(T) and $into_C \circ$ formal(T). Thus, the unique morphism $c \to H(n)$ must be an inclusion.

(2) Let c = colim(actual(T)). Then the cocone injections i_m for $into_C \circ$ formal(T) are given by $i_m = incl_m \circ h_m$, where $m \varepsilon base(n,\phi,AO)$ and $incl_m$ is the inclusion $f(m) \hookrightarrow c$.

Fact 6.5 shows how the colimit of a direct application diagram corresponds to a pushout in C.

Fact 6.5

Let T, H as before. Let F resp. A be the colimits of formal(T) resp. actual(T). Then

$$\begin{tikzcd} H(n) \arrow[r, dashed, "c_n"] & c \\ F \arrow[u, hook, "h_F"] \arrow[r, dashed, "h_A"] & A \arrow[u, "c_A"'] \end{tikzcd}$$

is a pushout diagram in C with colimit-to-cocone morphisms h_F and h_A iff c is a colimit object for the application diagram of T.

Proof: Let D_1 = diagram(T). We show that the pushout diagram results from D_1 by colimit preserving transformations (c.f. Figure 6.1 where an illustration for these transformations is given).

1. According to Fact 6.4. and Lemma 2 of Fact 4.4(4), D_1 may be transformed into D_2' by adding a new node n_F labelled with F, new edges from m_n into n_F for m ε M and from n_F into n_n labelled by the respective inclusions. Since all D_1 edges are labelled by the composition of the inclusions into n_F resp. n_F into n_n, D_2 results from D_2' by deleting all D_1 edges into n_n.

2. Lemma 3 of Fact 4.4(4) allows to transform D_2 into D_3 by adding a new node n_A labelled with A and edges from x ε base(T) into n_A labelled by inclusions since actual(T) = $H|_{base(T)}$.

3. Since for every node in D_2 other than n_n there is a path

$$\begin{tikzcd}
 & n_n:H(n) \arrow[rrr, dashed, "c_n"] & & & C & \\
 & n_F:F \arrow[u, hook, "h_F"] \arrow[rrr, dashed, "h_A"] & & & n_A:A \arrow[u, dashed, "c_A"'] & \\
 & m_{1n} \arrow[u, hook] \arrow[rrr, dashed, "h_{m1}"] & & & f(m_1) \arrow[u, hook] & \\
 & & m_{2n} \arrow[uul, hook] \arrow[rrr, dashed, "h_{m2}"] & & & f(m_2) \arrow[uul, hook] \\
b_1 \arrow[uuur, hook] \arrow[uuurrrr, hook] & m_{3n} \arrow[uu, hook] \arrow[ur, hook] \arrow[rrr, dashed, "h_{m3}"] & & & f(m_3) \arrow[uu, hook] \arrow[ur, hook] & \\
 & & b_2 \arrow[ul, hook] \arrow[urr, hook] & & b_3 \arrow[uull, hook] \arrow[uur, hook] & b_4 \arrow[uu, hook] \\
 & & & \perp \arrow[uulll, hook] \arrow[ul, hook] \arrow[ur, hook] \arrow[urr, hook] & &
\end{tikzcd}$$

Figure 6.1: The application diagram of Figure 3.3 after adding the nodes n_F, n_A, the edge from n_F to n_A, and colimit C according to the proof of Fact 6.5.

to n_A or n_F and for any path morphism p_F into n_F and path morphism p_A into n_A we have $h_A \circ p_F = p_A$, the colimit of D_2 is a cocone of D_3 resulting from D_2 by adding a new edge from n_F to n_A labelled with h_A. According to Lemma 1 of Fact 4.4(4) the colimits of D_2 and D_3 are identical.

4. Let D_4 be the subdiagram of D_3 containing only the nodes n_n, n_F, n_A and the edges from n_F to n_n and n_F to n_A. Let $(c,\{c_n, c_F, c_A\})$ be a colimit of D_3. For every node $x \varepsilon D_3 - D_4$ there is a path in D_3 from x to n_A, and since all such path morphisms from x to n_A must be identical let c'_x be the composition of that path morphism p_{xA} with c_A, i.e. $c'_x = c_A \circ p_{xA}$. If there is a path from x to n_n with morphism p_{xn} it must go through n_F and so there must also be a path from x to n_A going through n_F. Thus, p_{xA} and p_{xn} decompose into $p_{xA} = h_A \circ p_{xF}$ and $p_{xn} = h_F \circ p_{xF}$, respectively, where p_{xF} is the unique path morphism from x to n_F. Since $c_A \circ h_A = c_n \circ h_F$ we have $c'_x = c_A \circ h_A \circ p_{xF} = c_n \circ h_F \circ p_{xF}$. Thus, $(c,\{c_n,c_F,c_A\} \cup \{c'_x | x \varepsilon D_3 - D_4\})$ is a cocone of D_3, and due to Lemma 1 of Fact 4.4(4) D_3 and D_4 have identical colimits. D_4 is exactly the given pushout diagram, thereby completing the proof of Fact 6.5.

The proof of Fact 6.5 exhibits a condition an appropriate category must fulfill so that the colimit object for T becomes an application object for T.

<u>Definition 6.4</u> [mixed pushouts]

An appropriate category $(C, \check{C})$ has <u>mixed pushouts</u> iff for every diagram of the form

$$\begin{tikzcd} C \\ A \arrow[u, hook] \arrow[r, dashed] & B \end{tikzcd}$$

a pushout diagram

$$\begin{tikzcd} C \arrow[r, dashed] & D \\ A \arrow[u, hook] \arrow[r, dashed] & B \arrow[u, hook] \end{tikzcd}$$

exists.

Fact 6.6

Let H: AO → $\vec{C}$ be subhierarchy-complete. If (C, $\vec{C}$) has mixed pushouts then H is direct application complete.

Proof: For a direct application term T consider the pushout diagram of Fact 6.5. According to Fact 6.4, h_F is an inclusion. Since (C, $\vec{C}$) has mixed pushouts, there exists a c such that c_A is an inclusion as well. Sinced actual(T) = $H|_{base(T)}$, c_A yields an inclusion as colimit injection for diagram(T) for every x ε base(T) according to Fact 6.5. Thus, c is an application object for T.

Subhierarchy-completeness of H is a necessary condition in Fact 6.6: the hierarchy given in Fact 4.1 does not fulfill this condition although (SET,$\vec{SET}$) has mixed pushouts. However, consider the hierarchy H´: AO → $\vec{SET}$ with AO as in Fact 4.1 and H´($\perp$) = ϕ, H´(E) = {E.a}, H´(F) = {F.b}, H´(A) = {A.a}, H´(P) = {E.a,F.b}. H´ results from H by prefixing each element of a set with the name of the node where the element is introduced. H´ is subhierarchy-complete and thus direct application complete according to Fact 6.6. This prefixing method can be generalized to other categories as well.

Definition 6.5 [prefix function]

Let (C, $\vec{C}$) be an appropriate category, N the set of all possible nodes in a hierarchy. A function

$p: N \times 2^{|C|} \times |C| \to |C|$

is a *prefix function* iff there is a function p^{-1} with the same functionality such that for all finite hierarchies H: AO → $\vec{C}$ where each node label for a node n is of the form

$p(n, \{H(b) \mid b \varepsilon base(n,AO)\}, c)$

the following holds:

(1) $\forall\, n \;\varepsilon\; N\text{-}O.\; \forall c \varepsilon |\vec{C}|.$ c is a cocone object of H =>

- c and p(n,H(O),c) are isomorphic in C
- p(n,H(O),c) is a cocone object of H
- $p^{-1}(n,H(O) \cdot p(n,H(O),c)) = c$

(2) There is a colimit of H that is a colimit of $into_C \circ H$ as well.

The first condition in the above definition guarantees that a prefix function p does not change the information in a hierarchy. Instead of labelling a node n with base nodes B by c, n can always be labelled by $c' = p(n,H(B),c)$ since $p^{-1}(n,H(B),c')$ yields the original label c. The second condition is crucial since it is a basis for subhierarchy-completeness.

Definition 6.6 [prefix hierarchy]

H: AO $\rightarrow \vec{C}$ is a prefix hierarchy with prefix p iff p is a prefix function and every node n in O is labelled with p(n,H(base(n,AO)),c) for some c.

Fact 6.7

Every prefix hierarchy is subhierarchy-complete.

Proof: Let M be a finite set, $M \subseteq O$. Since AO is an appropriate order, every node $m \;\varepsilon\; M$ has only finitely many predecessors. Thus, $M' = \{m' \mid m' \;\varepsilon\; base(m,AO)$ for some $m \;\varepsilon\; M\}$ is finite. Since $H|_{M'}$ is a finite hierarchy, according to Definition 6.5 a colimit of $H|_{M'}$ exists that is a colimit of $into_C \circ H$ as well.

In Definitions 6.1 and 6.2 extension and closure of a hierarchy were introduced. In the presence of a category with mixed pushouts and a prefix function, Facts 6.6 and 6.7 suggest a slight modification so that the canonical closure of a hierarchy always exists.

Definition 6.7

Let H be a prefix hierarchy in a category with mixed pushouts.

(1) initial-hierarchy$_p$($\perp$,c) denotes:

initial-hierarchy($\perp$,p($\perp$,ϕ,c)).

(2) enter$_p$(H,(n,B,c)) denotes enter(H,(n,B,p(n,H(B),c))).

(3) closure$_p$(H) denotes the canonical closure of H analogously to Def. 6.2 but where every ´enter´ is replaced by ´enter$_p$´.

Fact 6.8

For every prefix hierarchy H in a category with mixed pushouts closure$_p$(H) exists and is canonically closed.

Proof: Definitions 6.2 and 6.7, Facts 6.1 and 6.7.

7. Parameterization-by-use in specification languages

In this section we will give an example what a language with parameterization-by-use might look like. In 7.1 a general hierarchy specification language is introduced in such a way that only the details concerning the concepts of hierarchy and parameterization are given, and a formal semantics for this language is defined. In 7.2 an instance of this language is presented by providing all parts left open in 7.1, yielding a language for the specification of signature hierarchies. In 7.3 we show how canonically closed hierarchies lead to non-proliferic semantics of specification languages.

7.1 A general hierarchy specification language

The hierarchy specification language is suitable for the specification of hierarchies in an arbitrary appropriate category $(C,\hat{C})$ with mixed pushouts and a hierarchy prefix function p. We will assume the existence of a semantic function

Sth: something $\rightarrow$ SOMETHING

where something is a syntactic category and SOMETHING is some particular domain, and another function

build-object: $2^{|C|}$ x SOMETHING $\rightarrow$ $|C|$

that builds hierarchically structured objects such that

$\forall b \in B.$ $b \hookrightarrow$ build-object(B,s)

Since we assume an appropriate category with mixed pushouts and a hierarchy prefix function p the semantics of the language will use only prefix hierarchies H. Thus, Fact 6.8 guarantees that the canonical closure $\text{closure}_p(H)$ exists. Therefore, we may use the function eval_H of Fact 6.1 extended by Definition 5.3 as evaluation function for application terms in H.

A hierarchy specification consists of a list of specifications all but the first one having a use clause denoting a list of objects as base of the new object to be generated. The last part of a hierarchy specification is either a node name or an application term. For simplicity, we do not go into details about the syntactical form of application terms, instead the mathematical notation as introduced in the previous chapters will be used.

1. Syntactic categories

hier-spec	:	hierarchy-specifications
decl	:	declarations
use-list	:	lists-of-application-terms
application-term	:	application-terms
n	:	node-names
sth	:	something (suitable as argument to the semantic function Sth)

2. Syntax

```
hier-spec    ::= object n = sth endobject decl
decl         ::= object n = use  use-list sth endobject decl |
                 application-term
use-list     ::= application-term use-list |
                 application-term
```

3. Values

η: hierarchies
M: set-of-nodes (in a hierarchy)

4. Semantic functions

P: hierarchy-specifications → hierarchies
D: declarations → hierarchies → hierarchies
U: lists-of-application-terms → hierarchies → set-of-nodes
A: application-terms → hierarchies → hierarchies

5. Semantic equations

```
P [ object n = sth endobject decl ] =
                    let s = Sth [sth] in
                    let c = build-object($\phi$,s) in
                    let $\eta$ = initial-hierarchy$_p$(n,c) in
                        D [decl] $\eta$

D [application-term]$\eta$ = A[application-term]$\eta$

D [ object n = use use-list sth endobject  decl]$\eta$ =
                    let M = U[use-list] $\eta$ in
                    let B = {$\eta$(m) |m$\in$M} in
                    let s = Sth [sth]  in
                    let c = build-object(B,s) in
                    let $\eta'$ = enter$_p$($\eta$,(n,M,c)) in
                    let $\eta''$ = closure$_p$($\eta'$) in
                        D[decl] $\eta''$

U[application-term]$\eta$ =
                    let $\eta$:AO $\rightarrow$ $\check{C}$ in
                    let n = eval$_\eta$(application-term) in
                          base(n,AO)

U[application-term use-list]$\eta$ =
                    let $\eta$:AO $\rightarrow$ $\check{C}$ in
                    let n = eval$_\eta$(application-term) in
                    let M = base(n,AO) in
                        M $\cup$ U[use-list]$\eta$

A[application-term]$\eta$ =
                    let n = eval$_\eta$(application-term) in
                        $\eta$|{n}
```

7.2 A specification language for hierarchies of signatures

7.2.1. Signatures

A signature is a set of sorts S together with a set of operators, ceach operator having an arity in $S^* \times S$. A signature morphism is a translation of sorts to sorts and operators to operators such that the arities are preserved.

Definition 7.1 [signatures]

(1) $\Sigma=(S,F)$ is a signature iff S is a set and F is an S^*xS-indexed family of sets.

(2) $\sigma:\Sigma\to\Sigma'$ is a signature morphism iff $\Sigma=(S,F)$, $\Sigma'=(S',F')$, $\sigma=(g,h)$ and g is a map $g:S\to S'$ and h is an S^*xS-indexed family of maps $h_{ws}: F_{ws} \to F'_{g^*(w)g(s)}$ where $w\varepsilon S^*$, $s \varepsilon S$ and g^* is the extension of g to strings. σ is a signature inclusion iff g and all h_{ws} are set-theoretic inclusions.

(3) SIG is the category of signatures with signature morphisms as morphisms, $S\hat{I}G$ is the subcategory with only signature inclusions as morphisms.

For a set N, $n\varepsilon N$, a set of signatures $B = \{\Sigma_1,\ldots,\Sigma_k\}$, $\Sigma_i = (S_i,F_i)$, and a signature $\Sigma = (S,F)$ such that there is an inclusion $\Sigma_i \hookrightarrow \Sigma$ for $i \varepsilon \{1,\ldots,k\}$ let p be the function

$$p: N \times 2^{|SIG|} \times |SIG| \to |SIG|$$

defined by:

$$\begin{aligned} p(n,B,\Sigma) = {} & \underline{let}\ S' = S - (S_1 \cup \ldots \cup S_k)\ \underline{in} \\ & \underline{let}\ F'_{ws} = F_{ws} - (F_{1ws} \cup \ldots \cup F_{kws})\ \underline{in} \\ & \underline{let}\ \pi = \lambda x.\ n.x\ \underline{in} \\ & \underline{let}\ S'' = \pi(S')\ \underline{in} \end{aligned}$$

$\underline{\text{let}}\ F''_{\pi^*(w)\pi(s)} = \pi(F'_{ws})\ \underline{\text{in}}$

$((S-S') \cup S'', (F-F') \cup F'')$

The function p changes every sort and operator name in Σ not yet already contained in some $\Sigma' \in B$ into the name prefixed by n. The arities of the operators are changed accordingly.

Fact 7.1

(SIG, $S\overset{\uparrow}{I}G$) is an appropriate category with mixed pushouts and p is a hierarchy prefix function.

Proof:

1. (SIG,$S\overset{\uparrow}{I}G$) is an appropriate category since obviously there is at most one signature inclusion between any two signatures and the objects of SIG and $S\overset{\uparrow}{I}G$ are the same.

2. To prove that (SIG,$S\overset{\uparrow}{I}G$) has mixed pushouts consider the diagram

$$\begin{tikzcd}
\Sigma' & \\
\Sigma \arrow[u, hook] \arrow[r, dashed, "\sigma"] & \Sigma''
\end{tikzcd}$$

where $\Sigma = (S,F)$, $\Sigma' = (S',F')$, $\Sigma'' = (S'',F'')$. In order to apply a general pushout construction let $\equiv_S$ (resp. $\equiv_F$) be the smallest equivalence relation generated by

$\{(s,\sigma(s)) \mid s \in S\}$

resp.

$\{(f,\sigma(f)) \mid f \in F\}$

Since σ is a signature morphism $\equiv_S$ and $\equiv_F$ are compatible with the arities of the operators. Thus, $\equiv_S$ and $\equiv_F$ define a signature congruence $\equiv_\Sigma$ and

$$\Sigma_{po} = ((S' \dot{\cup} S'')/{\equiv_S},\ (F' \dot{\cup} F'')/{\equiv_F})$$

is a signature where $\dot{\cup}$ denotes the disjoint union and Σ_{po} is a pushout object of the diagram above: the pushout morphisms into Σ_{po} take any x into its equivalence class in Σ_{po}. Σ_{po} is isomorphic to the signature Σ''' that results from Σ_{po} by taking a representative for each equivalence class in Σ_{po} as defined by

- x for $[x_{\Sigma''}]$
- $x_{\Sigma'}$ for $[x_{\Sigma'}]$ iff $\forall\, y \in \Sigma''.\ x_{\Sigma'} \not\equiv_\Sigma y_{\Sigma''}$

In both cases the representatives are determined uniquely: the definition of $\equiv_\Sigma$ implies that there is at most one $x_{\Sigma''}$ in any equivalence class of Σ_{po}, and if there is no such $x_{\Sigma''}$ then there is exactly one $x_{\Sigma'}$ in that class. Thus, Σ''' is also a pushout object for the diagram. The pushout morphisms are given by the inclusion from Σ'' to Σ''' and by σ': $\Sigma' \to \Sigma'''$ as defined by:

- $\sigma'(x) := \sigma(x)$ iff $x \in \Sigma$
- $\sigma'(x) := x_{\Sigma'}$ otherwise.

Thus,

$$\begin{tikzcd}
\Sigma' \arrow[r, "\sigma'"] & \Sigma''' \\
\Sigma \arrow[u, hook] \arrow[r, "\sigma"] & \Sigma'' \arrow[u, hook]
\end{tikzcd}$$

is a pushout diagram in SIG.

3. It remains to be shown that p is a prefix function. For a set N, $n \in N$, a set of signatures $B = \{\Sigma_1, \ldots, \Sigma_k\}$, Σ_i =

(S_i, F_i), and a signature $\Sigma = (S,F)$ such that there is an inclusion $\Sigma_i \hookrightarrow \Sigma$ for $i \in \{1,\ldots,k\}$ let p^{-1} be the function

$$p^{-1}: N \times 2^{|SIG|} \times |SIG| \to |SIG|$$

defined by:

$$\begin{aligned} p^{-1}(n,B,\Sigma) = {} & \underline{\text{let}}\ S' = S - (S_1 \cup \ldots \cup S_k)\ \underline{\text{in}} \\ & \underline{\text{let}}\ F'_{ws} = F_{ws} - (F_{1ws} \cup \ldots \cup F_{kws})\ \underline{\text{in}} \\ & \underline{\text{let}}\ \rho = \lambda x.\ x{=}n.y \to y,\ T \to x\ \ \underline{\text{in}} \\ & \underline{\text{let}}\ S'' = \rho(S')\ \underline{\text{in}} \\ & \underline{\text{let}}\ F''_{\rho^*(w)\rho(s)} = \rho(F'_{ws})\ \underline{\text{in}} \\ & \quad ((S-S') \cup S'', (F-F') \cup F'') \end{aligned}$$

and let $H: AO \to \vec{SIG}$ be a finite signature hierarchy with $AO = (O,<,\perp)$, $O \subseteq N$ and where each node label for a node n is of the form $p(n, \{H(b) | b \in base(n,AO)\}, \Sigma)$.

In the following let $B = \{H(m) | m \in O\}$. To show the first condition for a prefix function (Definition 6.5) we have to show that for every $n \in N-O$ and every ($\vec{SIG}$-) cocone object Σ of H the following holds:

(1) Σ and $p(n,B,\Sigma)$ are isomorphic in SIG
(2) $p(n,B,\Sigma)$ is a cocone object of H
(3) $p^{-1}(n,B,p(n,B,\Sigma)) = \Sigma$

Let Σ_{col} be defined by $\Sigma_{col} := \bigcup_{m \in O} H(m)$

where the union of two signatures is given by the componentwise union of the sorts and operators. There is an inclusion from any $H(m)$ for $m \in O$ into Σ_{col} and for any other signature Σ'' with this property we have $\Sigma_{col} \hookrightarrow \Sigma''$. Thus, Σ_{col} is a ($\vec{SIG}$-) colimit of H, and the inclusion $\Sigma_{col} \hookrightarrow \Sigma$ is the colimit-to-cocone morphism. Since $p(n,B,\Sigma_{col}) = \Sigma_{col}$ and $\Sigma_{col} \hookrightarrow p(n,B,\Sigma)$ we conclude that $p(n,B,\Sigma)$ is a cocone object of H as well. Furthermore we can define the signature morphisms

$\sigma: \quad \Sigma \to p(n,B,\Sigma)$

$\sigma^{-1}: p(n,B,\Sigma) \to \Sigma$

by

$\sigma|_{\Sigma col} := id_{\Sigma col}$

$\sigma|_{\Sigma'-\Sigma col} := \lambda x.\ n.x$

$\sigma^{-1}|_{\Sigma col} := id_{\Sigma col}$

$\sigma^{-1}|_{p(n,B,\Sigma)-\Sigma col} := \lambda x.\ x=n.y \to y,\ T \to x$

Since every sort or operator in Σ_{col} has a prefix $m\varepsilon O$ that is distinct from $n\varepsilon N-O$ it is easy to see that σ is a signature isomorphism and that σ^{-1} is its inverse, implying that Σ and $p(n,B,\Sigma)$ are isomorphic in SIG. The functions p and p^{-1} correspond exactly to σ and σ^{-1} by taking into account that the arities of the operators have to be changed according to the newly added resp. removed prefix n, implying that p and p^{-1} are inverse to each other in the sense of condition (3) above.

In order to prove the second condition for a prefix function we show that Σ_{col} together with the signature inclusions $\{i_m | m\varepsilon O\}$ is a (SIG-)colimit of $into_C \circ H$ as well. Obviously, it is a cocone of $into_C \circ H$. Now let $(\Sigma', \{\sigma_m | m\varepsilon O\})$ be any cocone of $into_C \circ H$. Let $\sigma: \Sigma_{col} \to \Sigma'$ be given by

$\sigma(x) :=$ <u>let</u> $x = m.y$ <u>in</u>

$\sigma_m(x)$

where σ is well defined since every $x \varepsilon \Sigma_{col}$ has a unique prefix $m\varepsilon O$ and for every such x we have $x\varepsilon H(m)$. Thus, we have $\sigma \circ i_m = \sigma_m$ for every $m\varepsilon O$, and since there is no other σ with this property we conclude that Σ_{col} is a colimit of $into_C \circ H$. Thus we have shown that p is a prefix function, thereby completing the proof of Fact 7.1.

Fact 7.1 implies that SIG with subcategory $\vec{SIG}$ and the prefix function p are suitable for the language introduced in Sec. 7.1.

We just have to define the two functions

(1) Sth: something $\to$ SOMETHING
(2) build-object: $2^{|SIG|}$ x SOMETHING $\to$ $|SIG|$

(1) sth::= <u>sorts</u> $s_1, \ldots, s_r$
<u>ops</u> $f_1: s_{1,1} \ldots s_{1,n1} \to s_1$
.
.
$f_m: s_{m,1} \ldots s_{m,nm} \to s_m$

where $r, m, n, n_i > 0$

Let SOMETHING be the set of pairs (S,F) such that S is a set and F is an $S'^* \times S'$ - indexed family of sets with $S \subseteq S'$.
Let sth be as above, then:

Sth[sth] = <u>let</u> S = $\{s_1, \ldots, s_r\}$ <u>in</u>
<u>let</u> $F_{ws} = \{f_i | f_i: w \to s\ ,\ w = s_{i,1} \ldots s_{i,ni},\ s = s_i\}$ <u>in</u>
(S,F)

(2) For $B = \{\Sigma_1, \ldots, \Sigma_n\}$, $\Sigma_i = (S_i, F_i)$, define:

build-object (B, (S,F)) =
<u>let</u> $S' = S_1 \cup \ldots \cup S_n \cup S$ <u>in</u>
<u>let</u> $F' = F_1 \cup \ldots \cup F_n \cup F$ in
(S', F')

<u>7.2.2 Hierarchically structured signatures</u>

We will now give an example of a specification for a hierarchically structured signature. Again we will not define the syntax of applications terms in detail, but we will use the mathematical notation as we did in 7.1. A complete formal treatment with a special syntax for application terms should pose no difficulties and could be done similarly to the formalizations above. We will define signatures for booleans, arbitrary elements, natural numbers, stacks and sets over arbitrary elements. In the last object declaration we give a signature for various stack and set instances in order to illustrate the different aspects of parameterization-by-use in hierarchical objects.

The hierarchy specification is:

```
object BOOL =
    sorts bool
    ops   true: → bool
          false: → bool
          not: bool → bool
          and: bool bool → bool
endobject

object  ELEM =
    use BOOL
    sorts elem
endobject

object NAT =
    use BOOL
    sorts nat
    ops 0: → nat
        succ: nat → nat
        le:   nat nat → BOOL.bool
endobject

object LIMIT =
    use NAT
    ops limit: → NAT.nat
endobject
```

```
object MAX =
    use NAT
    ops maxl: → NAT.nat
        max2: → NAT.nat
endobject

object STACK =
    use BOOL, NAT, LIMIT, ELEM
    sorts stack
    ops empty: → stack
        push: ELEM.elem stack → stack
        pop:  stack → stack
        top:  stack → ELEM.elem
        depth:stack → NAT.nat
endobject

object  SET =
    use  ELEM
    sorts  set
    ops    create: → set
           insert: ELEM.elem  set → set
           remove: ELEM.elem  set → set
           has:    ELEM.elem  set → BOOL.bool
endobject

object  STACKS&SETS
    use  SET {(ELEM,$\sigma_2$,STACK )}{(ELEM,$\sigma_1$,NAT)},
         SET {(ELEM,$\sigma_1$,NAT)},
         STACK {(ELEM,$\sigma_2$,STACK)}{(ELEM,$\sigma_1$,NAT)}
         STACK {(LIMIT,$\sigma_3$,MAX)}
    ops convert: STACK{(ELEM,$\sigma_1$,NAT)}. stack
                          → SET{(ELEM,$\sigma_1$,NAT)}.set
endobject

STACKS&SETS
```

The signature morphisms used in STACKS&SETS are given by the identity except for:

σ_1(ELEM.elem) = NAT.nat
σ_2(ELEM.elem) = STACK.stack
σ_3(LIMIT.limit) = MAX.max2

Figure 7.1 shows a section of the closed hierarchy generated by this specification. If HS is the hierarchy specification and η: AO$\to\vec{\text{SIG}}$ is the hierarchy generated by HS, then the semantics of HS is given by

$$P[HS] = \eta|_{\{STACKS\&SETS\}}$$

i.e. the hierarchy of Figure 7.1 restricted to the nodes n having a path to STACKS&SETS.

7.2.3 Evaluating signature application terms

As an example for the evaluation of application terms, we will take the first use clause element of STACKS&SETS

(1) SET{(ELEM,σ_2,STACK)}{(ELEM,σ_1,NAT)}

and show that it evaluates to the same node in the hierarchy as

(2) SET{(ELEM,σ_4,STACK{(ELEM,σ_1,NAT)})}

where σ_4 is given by the identity except for

σ_4(ELEM.elem) = STACK{(ELEM,σ_1NAT)}.stack

Let T be the direct application term

$$\begin{tikzcd}
 & & \mathrm{STACKS\&SETS} & & & \\
 & & \mathrm{STACK}\{(\mathrm{ELEM},\sigma_4, \mathrm{STACK}\{(\mathrm{ELEM},\sigma_1,\mathrm{NAT})\})\} \arrow[u, hook] & & \mathrm{SET}\{(\mathrm{ELEM},\sigma_4, \mathrm{STACK}\{(\mathrm{ELEM},\sigma_1,\mathrm{NAT})\})\} \arrow[ull, hook] & \\
 \mathrm{STACK}\{(\mathrm{LIMIT},\sigma_3,\mathrm{MAX})\} \arrow[uurr, hook] & & & \mathrm{STACK}\{(\mathrm{ELEM},\sigma_1,\mathrm{NAT})\} \arrow[ul, hook] \arrow[ur, hook] & & \\
 & \mathrm{SET}\{(\mathrm{ELEM},\sigma_1,\mathrm{NAT})\} \arrow[uuur, hook] & & & \mathrm{STACK}\{(\mathrm{ELEM},\sigma_2,\mathrm{STACK})\} & \\
 & & & & & \mathrm{SET}\{(\mathrm{ELEM},\sigma_2,\mathrm{STACK})\} \\
 & & & & \mathrm{STACK} \arrow[uu, hook] \arrow[ur, hook] & \mathrm{SET} \\
 \mathrm{MAX} \arrow[uuuu, hook] & & \mathrm{LIMIT} \arrow[urr, hook] \arrow[uuuur, hook] & & & \\
 & \mathrm{NAT} \arrow[ul, hook] \arrow[ur, hook] \arrow[uuuu, hook] & & \mathrm{ELEM} \arrow[uuuuulll, hook] \arrow[uur, hook] \arrow[uurr, hook] & & \\
 & & \mathrm{BOOL} \arrow[ul, hook] \arrow[ur, hook] & & &
\end{tikzcd}$$

Figure 7.1: The hierarchy of signatures with STACKS&SETS

$T = \text{SET}\{(\text{ELEM},\sigma_2,\text{STACK})\}$.

Then (1) can be written as

(3) $T\{(\text{ELEM},\sigma_1,\text{NAT})\}$

which according to Definition 5.3 (3) evaluates to

(4) $\text{direct}(T) \circ \text{direct}(\text{eval}_\eta(T)\{(\text{ELEM},\sigma_1,\text{NAT})\})$.

Since $\text{direct}(T) = T$ and $\text{eval}_\eta(T) =$ "T", (4) is equivalent to

(5) $T \circ \text{direct}($ "T"$\{(\text{ELEM},\sigma_1,\text{NAT})\})$.

"T"$\{(\text{ELEM},\sigma_1,\text{NAT})\}$ is an indirect application term since no actual parameter is given for STACK which is used by "T" = "SET$\{(\text{ELEM},\sigma_2,\text{STACK})\}$" (c.f. Fig. 7.1). Thus,

between("T",{ELEM},AO) = {STACK},

and according to Fact 5.2

$T_{\text{STACK}} = \text{STACK}\{(\text{ELEM},\sigma_1,\text{NAT})\}$

is a direct application term. Let σ_5 be the colimit injection

σ_5: $\eta(\text{STACK}) \to \text{application-object}(T_{\text{STACK}})$

which is the obvious extension of σ_1 to STACK by sending the sort and operation names introduced in STACK identically to the sort and operation names in $\text{STACK}\{(\text{ELEM},\sigma_1,\text{NAT})\}$ where only the new prefix has to be taken into account. Furthermore, we have

$\text{eval}_\eta(T_{\text{STACK}}) =$ "STACK$\{(\text{ELEM},\sigma_1,\text{NAT})\}$"

and according to Fact 5.3

(6) $T \circ$ "T"$\{(\text{ELEM},\sigma_1,\text{NAT}), (\text{STACK},\sigma_5,$"STACK$\{(\text{ELEM},\sigma_1,\text{NAT})\}$"$)\}$

is an application term which is equivalent to (5) according to Def. 5.3. In Fact 4.4 the composition of direct application terms is introduced and Fact 6.1 says that the evaluation function eval_η respects direct application composition. Thus, (6) is equivalent to

(7) SET{(ELEM,$\sigma_5 \circ \sigma_1$, "STACK{(ELEM,σ_1,NAT)}")}.

But composing the signature morphisms σ_1 and σ_5 yields

$$\sigma_4 = \sigma_5 \circ \sigma_1$$

so that (7) is equivalent to

(8) SET{(ELEM,σ_4, "STACK{(ELEM,σ_1,NAT)}")}

which in turn is equivalent to

(9) SET{(ELEM,σ_4, STACK{(ELEM,σ_1,NAT)})}

due to the definition of application terms in Def. 5.3 (4). Since (9) is exactly the term (2) given above, we have shown that both (1) and (2) evaluate to the same node in the hierarchy.

The equivalence of (1) and (2) in the sense that they evaluate to the same node in the hierarchy is an example showing the associativity of applications (c.f. Fact 4.5): first instantiating the elements ELEM of SET by STACK and then instantiating the elements of STACK by NAT yields the same result as instantiating the elements of SET by the result of instantiating the elements of STACK by NAT. Similar associativity results in a non-hierarchical framework are given in [Eh 82], [EKTWW 80b], [Ga 81]. In such a framework a non-hierarchical semantics as in Fact 5.4 would be sufficient, whereas our hierarchical approach of taking indirect to direct application terms guarantees that the objects denoted by (1) and (2) are identical even when viewed as hierarchical objects.

7.3. Non-proliferic semantics for specification languages

In Clear ([BG 77], [BG 80]) non-parameterized and parameterized objects are distinguished, namely theories and theory procedures. The Clear equivalent to an application term like

(10) STACK{(ELEM,σ_1,NAT)}

denotes a corresponding instantiation object. However, writing down the same Clear term twice at two different places yields two distinct copies of that object: each time a theory procedure is applied to actual parameters a new object is generated. This proliferation problem of Clear ([BG 81], [Sa 81]) can be avoided by the use of canonically closed hierarchies since in all contexts the term (10) evalutes to the same node in the hierarchy.

This solution to Clear´s proliferation problem goes further than [Sa 81]. In [Sa 81], a term corresponding to (10) would always yield the same object in different contexts. But the terms

(11) STACK{(ELEM,σ_1,NAT)} {(LIMIT,σ_3,MAX)}

(12) STACK{(LIMIT,σ_3,MAX)} {(ELEM,σ_1,NAT)}

(13) STACK{(ELEM,σ_1,NAT), (LIMIT,σ_3,MAX)}

would yield three different copies of actually the same instantiation object. In a canonically closed hierarchy, however, (11), (12) and (13) all evaluate to the same node, namely

"STACK{(ELEM,σ_1,NAT), (LIMIT,σ_3,MAX)}"

and thus all three terms (11) - (13) denote the same object.

Furthermore, the evaluation of indirect application terms by transforming them into direct application terms also avoids unnecessary duplications of instantiation objects. As an example, consider the indirect application terms in STACKS&SETS's use clause. Similarly to the evaluation process shown in 7.2.3, we conclude that the use clause element of STACKS&SETS

(14) STACK{(ELEM,σ_2,STACK)} {(ELEM,σ_1,NAT)}

evaluates to the same node as

(15) STACK{(ELEM,σ_4,STACK{(ELEM,σ_1,NAT)})}

Thus both the objects denoted by (1) and (14) are based on the same object

(16) STACK{(ELEM,σ_1,NAT)}

and consequently, STACKS&SETS includes only one copy of (16).

Apparently, this reflects exactly the intuition one might have when writing a hierarchical specification based on both terms (1) and (14). Instantiating the elements ELEM of both SET and STACK by STACK , and instantiating the elements ELEM of both the resulting objects by NAT should be equivalent to instantiating the elements ELEM of both SET and STACK by the result of instantiating ELEM of STACK by NAT. Thus, the final instantiations of SET and STACK should be based upon the same instantiation of STACK. As demonstrated above, this may be achieved by the use of canonically closed hierarchies.

The examples given in this section show that in the hierarchy specification language for signatures introduced in 7.2 a Clear-like proliferation is avoided since the occuring hierarchies can be canonically closed. However, following the general development of parameterization-by-use for hierarchically structured objects in an arbitrary appropriate category it was possible to prove that mixed pushouts and a prefix function suffice to guarantee the existence of the canonical closure of a hierarchy (Fact 6.8).

These two conditions may easily be met by a specification language:

- mixed pushouts: many specification languages for the definition of abstract data types (e.g. [BG 80], [BA 81]) are based on a notion of signature as defined in 7.2. In Fact 7.1 we showed that ($\hat{SIG}$, SIG) is an appropriate category with mixed pushouts. As already pointed out in [BG 80] and further developed in the framework of institutions in [GB 83] the existence of pushouts and colimits carry over from a category of signatures to a category of specifications (called theories in [GB 83]). It is easy to prove that the same is true for the existence of mixed pushouts.

- prefix function: the prefix function given in 7.2 for hierarchies of signatures guarantees the existence of the canonical closure of signature hierarchies. Again, since in the framework of institutions colimits of signatures and also mixed pushouts carry over to colimits and mixed pushouts in a category of specifications (or theories), the existence of the canonical closure for specification hierarchies is guaranteed as well.

The concepts of parameterization-by-use and canonically closed hierarchies are incorporated in the specification language ASPIK [BV 83] that allows for axiomatic and algorithmic specifications of abstract data types; several examples demonstrating these concepts are also given in [BGV 83].

References

[Ba 81] Bauer, F.L. et al.: Report on a wide spectrum language for program specification and development. TU München, Inst.f.Informatik, Report TUM-I8104, May 1981.

[BGV 83] Beierle, Ch., Gerlach, M., Voß, A.: Parameterization without parameters - in: the history of a hierarchy of specifications. SEKI-Projekt, Univ. Kaiserslautern, FB Informatik (in preparation).

[BV 83] Beierle, Ch., Voß, A.: Canonical term functors and parameterization-by-use for the specification of abstract data types. SEKI-Projekt, MEMO SEKI-83-07, Universität Kaiserslautern FB Informatik May 1983.

[BDPPW 80] Broy, M., Dosch, W., Partsch, H., Pepper, P., Wirsing, M.: On hierarchies of abstract data types, TU München, Inst. für Informatik, TUM-I8007, May 1980.

[BG 77] Burstall, R.M., Goguen, J.A.: Putting Theories together to Make Specifications. Proc. 5th IJCAI, 1977, pp. 1045-1058.

[BG 80] Burstall, R.M., Goguen, J.A.: The semantics of Clear, a specification language. Proc. of Advanced Course on Abstract Software Specifications, Copenhagen. LNCS Vol.86, pp. 292-332.

[BG 81] Burstall, R.M., Goguen, J.A.: An informal introduction to specifications using Clear. in: The Correctness problem in Computer Science (Eds. R.S. Boyer, J.S. Moore). Academic Press 1981.

[Eh 82] Ehrich, H.-D.: On the theory of specification, Implementation and Parametrization of Abstract Data Types. JACM Vol. 29, No. 1, Jan. 1982, pp. 206-227.

[Eh 81] Ehrig, H.: Algebraic Theory of Parameterized Specification with Requirements, Proc. 6th CAAP, Genova, 1981.

[EKTWW 80] Ehrig, H., Kreowski, H.-J., Thatcher J., Wagner, E., Wright, J.: Parameterized data types in algebraic specification languages, Proc. 7th ICALP, LNCS Vol. 85, 1980, pp. 157-168.

[EKTWW 80b] Ehrig, M., Kreowski, H.-J., Thatcher, J., Wagner, E., Wright, J.: Paramter Passing in Algebraic Specification languages. Draft version, TU Berlin, March 1980.

[Ga 81] Ganzinger, H.: Parameterized specifications: parameter passing and optimizing implementation. TU München, Institut f. Informatik, Report TUM-I8110, August 1981.

[GB 83] Goguen, J.A., Burstall, R.M.: Institutions: Abstract Model Theory for Program Specification. Draft version. SRI International and University of Edinburgh, January 1983.

[GT 79] Goguen, J.A., Tardo, J.: An Introduction to OBJ: A Language for Writing and Testing Software Specifications. In: Specification of Reliable Software, IEEE 1979, pp. 170-189.

[HS 73] Herrlich, H., Strecker, G.E., Category Theory. Allyn and Bacon, Boston 1973.

[JW 76] Jensen, K. Wirth, N.: Pascal User Manual and Report. LNCS 18, Springer Verlag 1976.

[Li 82] Lipeck, U.: Ein algebraischer Kalkül für einen strukturierten Entwurf von Datenabstraktionen. Dissertation. Forschungsbericht Nr. 148, Universität Dortmund, 1983.

[LSAS 77] Liskov, B., Snyder, A., Atkinson, R., Schaffert, C.: Abstraction Mechanisms in CLU. CACM Vol 20, No. 8, August 1977, pp. 564-576.

[McL 71] MacLane, S.: Categories for the Working Mathematician. Springer Verlag, 1971.

[*Na* 63] *Naur, P. (ed): Revised Report on the Algorithmic* Language ALGOL 60. CACM 6, 1963, pp. 1-17.

[Sa 81] Sannella, D.T.: A new semantics for Clear. Report CSR -79-81, Dept. of Computer Science, Univ. of Edinburgh, 1981.

[SB 83] Sannella, D.T., Burstall, R.M.: Structured theories in LCF. Proc. CAAP 1983.

[TWW 82] Thatcher, J.W., Wagner, E.G., Wright, J.B.: Data Type Specification: Parameterization and the Power of Specification Techniques. ACM TOPLAS Vol. 4, No. 4, Oct. 1982, pp. 711-732.

[WLS 76] Wulf, W.A., London, R.L Shaw, M.: An Introduction to the Construction and Verification of Alphard Programs. IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976.