$$
\begin{array}{ll}
(\lambda x.M)N & \rightarrow_{op} M[N/x] \\
YM & \rightarrow_{op} M(YM) \\
succ(n) & \rightarrow_{op} n+1 \\
pred(n+1) & \rightarrow_{op} n \\
zero?(0) & \rightarrow_{op} tt \\
zero?(n+1) & \rightarrow_{op} ff \\
if\ tt\ then\ N\ else\ P & \rightarrow_{op} N \\
if\ ff\ then\ N\ else\ P & \rightarrow_{op} P
\end{array}
$$

$$
\frac{M \rightarrow_{op} M'}{MN \rightarrow_{op} M'N} \qquad \frac{M \rightarrow_{op} M'}{if\ M\ then\ N\ else\ P \rightarrow_{op} if\ M'\ then\ N\ else\ P}
$$

$$
\frac{M \rightarrow_{op} M'}{succ(M) \rightarrow_{op} succ(M')} \qquad \frac{M \rightarrow_{op} M'}{pred(M) \rightarrow_{op} pred(M')} \qquad \frac{M \rightarrow_{op} M'}{zero?(M) \rightarrow_{op} zero?(M')}
$$

Figure 6.3: Operational semantics for PCF

Definition 6.3.5 (PCF program) *We call programs the terms of* PCF *which are closed and of basic type.*

For example, $(\lambda x.x)3$ and $add_l\, 4\, 3$ (cf. exercise 6.3.3) are programs.

Theorem 6.3.6 (adequacy) *Any standard model* $\mathbf{C}$ *of* PCF *is adequate, i.e., for all programs of type* ι *(and similarly for type* o*):*

$$(\exists\, n \;\; P \to^{\star}_{op} n) \Leftrightarrow [\![P]\!] = n.$$

PROOF. ($\Rightarrow$) Follows by soundness of the continuous model.

($\Leftarrow$) The key idea is to decompose the problem into two subproblems, one which will be proved by induction on types, the other by induction on terms. We use the notation of section 4.5, and write $\underline{D^\sigma} = \mathbf{C}[1, D^\sigma]$. The induction on types comes into play by a definition of a family of relations $\mathcal{R}^\sigma \subseteq \underline{D^\sigma} \times PCF^o_\sigma$, for each type σ, where PCF^o_σ is the set of closed terms of type σ. Here is the definition of these (logical-like) relations ($\mathcal{R}^o$ is analogous to $\mathcal{R}^\iota$):

$$\begin{array}{lcl} \mathcal{R}^\iota & = & \{(x, M) \mid x = \bot \text{ or } (x = n \text{ and } M \to^{\star}_{op} n)\} \\ \mathcal{R}^{\sigma\to\tau} & = & \{(f, M) \mid \forall\, e, N \;\; (e\; \mathcal{R}^\sigma\; N \Rightarrow ev \circ \langle f, e\rangle\; \mathcal{R}^\tau\; MN)\}\ . \end{array}$$

The statement is a part of the following claim. For each provable judgement $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \sigma$, for each n-tuple $(d_1, N_1), \ldots, (d_n, N_n)$ such that $d_i\; \mathcal{R}^{\sigma_i}\; N_i$ for $i = 1, \ldots, n$, we have

$$[\![\vec{x} : \vec{\sigma} \vdash M]\!] \circ \langle d_1, \ldots, d_n\rangle\; \mathcal{R}^\sigma\; M[N_1/x_1, \ldots, N_n/x_n].$$

We set $M' = M[N_1/x_1, \ldots, N_n/x_n]$, etc... We proceed with the simplest cases first.

$M = x_i$: Then $[\![M]\!] \circ \langle d_1, \ldots, d_n\rangle = d_i$, and $M[N_1/x_1, \ldots, N_n/x_n] = N_i$, hence the sought result is $d_i\; \mathcal{R}^{\sigma_i}\; N_i$, which is among the assumptions.

$M = NQ$: By induction $[\![N]\!] \circ \langle d_1, \ldots, d_n\rangle\; \mathcal{R}^{\sigma\to\tau}\; N'$ and $[\![Q]\!] \circ \langle d_1, \ldots, d_n\rangle\; \mathcal{R}^\sigma\; Q'$. By definition of $\mathcal{R}^{\sigma\to\tau}$, $ev \circ \langle [\![N]\!] \circ \langle d_1, \ldots, d_n\rangle, [\![Q]\!] \circ \langle d_1, \ldots, d_n\rangle\rangle\; \mathcal{R}^\tau\; N'Q'$, i.e. $[\![M]\!] \circ \langle d_1, \ldots, d_n\rangle\; \mathcal{R}^\tau\; M'$.

$M = \lambda x.Q$: We have to show, for each $d\; \mathcal{R}^\sigma\; N$:

$$ev \circ \langle [\![M]\!] \circ \langle d_1, \ldots, d_n\rangle, d\rangle\; \mathcal{R}^\tau\; M'N \;\; i.e. \;\; [\![Q]\!] \circ \langle d_1, \ldots, d_n, d\rangle\; \mathcal{R}^\tau\; (\lambda x.Q')N.$$

By induction we have

$$[\![Q]\!] \circ \langle d_1, \ldots, d_n, d\rangle\; \mathcal{R}^\tau\; Q[N_1/x_1, \ldots, N_n/x_n, N/x].$$

Since $(\lambda x.Q')N \to_{op} Q[N_1/x_1, \ldots, N_n/x_n, N/x]$, we can conclude provided the following property holds, for all σ:

$$(Q_1) \quad f\,\mathcal{R}^\sigma\,M \text{ and } M' \to_{op} M \quad \Rightarrow \quad f\,\mathcal{R}^\sigma\,M'.$$

$M = n$: In this case, $n\,\mathcal{R}^\iota\,M$ holds trivially. Similarly for tt and ff.

$M = succ$: Let $d\,\mathcal{R}^\iota\,P$. We have to show $ev \circ \langle succ, d\rangle\,\mathcal{R}^\iota\,succ(P)$. There are two cases:

$$\begin{array}{ll} d = \bot: & \text{Then } ev \circ \langle succ, d\rangle = ev \circ \langle succ, \bot\rangle = \bot \\ d = n: & \text{Then } ev \circ \langle succ, d\rangle = n + 1\ . \end{array}$$

In both cases $ev \circ \langle succ, d\rangle\,\mathcal{R}^\iota\,succ(P)$. The reasoning is similar for *pred*, *zero*?, and *if then else* .

$M = Y$: We have to show $[\![Y]\!]\,\mathcal{R}^{(\sigma\to\sigma)\to\sigma}\,Y$, that is, $ev \circ \langle [\![Y]\!], g\rangle\,\mathcal{R}^\sigma\,YM$, for all $g\,\mathcal{R}^{\sigma\to\sigma}\,M$. We assume the following properties (cf. inclusive predicates), for all σ:

(Q_2) $\bot\,\mathcal{R}^\sigma\,M$

(Q_3) $\{f_n\}_{n<\omega}$ non decreasing implies $(\forall\, n\ \ f_n\,\mathcal{R}^\sigma\,M) \Rightarrow (\bigvee_{n<\omega} f_n)\,\mathcal{R}^\sigma\,M$.

By (Q_3), the conclusion follows if we show:

$$ev \circ \langle [\![\lambda f.f^n\Omega]\!], g\rangle\,\mathcal{R}^\sigma\,YM \quad \text{(for all } n\text{)}.$$

We set $d_n = ev \circ \langle [\![\lambda f.f^n\Omega]\!], g\rangle$. Since $d_n = [\![f^n\Omega]\!] \circ g$, we have $d_{n+1} = ev \circ \langle g, d_n\rangle$ for all n. Therefore, we only have to show:

1. $d_0\,\mathcal{R}^\sigma\,YM$: Since $d_0 = [\![\Omega]\!] \circ g$, this follows from (Q_2) and from the left strictness of composition.

2. $(d\,\mathcal{R}^\sigma\,YM) \Rightarrow (ev \circ \langle g, d\rangle\,\mathcal{R}^\sigma\,YM)$: Since $g\,\mathcal{R}^{\sigma\to\sigma}\,M$ by assumption, we have $ev \circ \langle g, d\rangle\,\mathcal{R}^\sigma\,M(YM)$, and the conclusion then follows by (Q_1).

Properties (Q_1) and (Q_2) are obvious at basic types. For a type $\sigma \to \tau$, (Q_1) follows by induction from the inference: $(M' \to_{op} M) \Rightarrow (M'N \to_{op} MN)$ and (Q_2) follows from the strictness equation $ev \circ \langle \bot, d\rangle = \bot$. (Q_3) follows at basic types from the fact that non-increasing sequences are stationary in a flat domain, and at functional types from the preservation of limits by continuity. This completes the proof of the claim. □

6.4 The Full Abstraction Problem for PCF

In general, given a programming language, the specification of the operational semantics is given in two steps:

1. Evaluation: a collection of *programs* is defined, usually a collection of closed terms, on which a partial relation of evaluation is defined. The evaluation is intended to describe the dynamic evolution of a program while running on an abstract machine.

2. Observation: a collection of admissible observations is given. These observations represent the only mean to record the behavior of the evaluation of a program.

In this fashion, an observational equivalence can be defined on arbitrary terms M and N as follows: M is observationally equivalent to N if and only if whenever M and N can be plugged into a piece of code P, so to form correct programs $P[M]$ and $P[N]$, then M and N are not separable (or distinguishable) by any legal observation. On the other hand any interpretation of a programming language provides a theory of program equivalence. How does this theory compare to observational equivalence? We will say that an interpretation (or a model) is adequate whenever it provides us with a theory of equivalence which is contained in the observational equivalence. Moreover we call an adequate model (equationally) fully abstract if the equivalence induced by the model coincides with the observational equivalence.

In this section we discuss the situation for PCF. We have defined the programs as the closed terms of base type. We have defined an evaluation relation $\rightarrow_{op}$. What can be observed of a program is its convergence to a natural number or to a boolean value. The principal reason for focusing on programs is that they lead to observable results. This stands in contrast with expressions like $\lambda x.x$, which are just code, and are not evaluated by $\rightarrow_{op}$ unless they are applied to an argument, or more generally unless they are plugged into a program context. A program context for a PCF term is a context C (cf. definition 2.1.6) such that $C[M]$ is a program.

Definition 6.4.1 (observational preorder) *We define a preorder $\leq_{obs}$, called observational preorder, between* PCF *terms M, N of the same type, as follows:*

$$M \leq_{obs} N \Leftrightarrow \forall C\ (C[M] \rightarrow_{op}^{\star} c\ \Rightarrow\ C[N] \rightarrow_{op}^{\star} c)$$

where C ranges over all the contexts which are program contexts for both M and N, and where $c ::= n \mid tt \mid ff$.

Remark 6.4.2 *By exercise 6.3.4 and by theorem 6.3.6, equivalent definitions for $\leq_{obs}$ are:*

$$M \leq_{obs} N \Leftrightarrow \forall C \text{ program context } (C[M] \rightarrow^{\star} c\ \Rightarrow\ C[N] \rightarrow^{\star} c)$$
$$M \leq_{obs} N \Leftrightarrow \forall C \text{ program context } (\llbracket C[M] \rrbracket \leq \llbracket C[N] \rrbracket)\,.$$

Definition 6.4.3 (fully abstract) *A cpo-enriched CCC is said to yield an inequationally fully abstract (fully abstract for short) model of* PCF *if the following equivalence holds for any* PCF *terms of the same type:*

$$M \leq_{obs} N \Leftrightarrow [\![M]\!] \leq [\![N]\!].$$

It is a consequence of the adequacy theorem that the direction ($\Leftarrow$) holds for the continuous model (and in fact for any standard model). But the converse direction does not hold for the continuous model. There are several proofs of this negative result, all based on a particular continuous function $por : \mathbf{B}_\perp \times \mathbf{B}_\perp \to \mathbf{B}_\perp$ defined by:

$$por(x,y) = \begin{cases} tt & \text{if } x = tt \text{ or } y = tt \\ ff & \text{if } x = ff \text{ and } y = ff \\ \perp & \text{otherwise .} \end{cases}$$

1. Plotkin first proved that the continuous model is not fully abstract. He gave the following terms:

$$M_1 = \lambda g.\mathit{if}\ P_1\ \mathit{then}\ \mathit{if}\ P_2\ \mathit{then}\ \mathit{if}\ P_3\ \mathit{then}\ \Omega\ \mathit{else}\ tt\ \mathit{else}\ \Omega\ \mathit{else}\ \Omega$$
$$M_2 = \lambda g.\mathit{if}\ P_1\ \mathit{then}\ \mathit{if}\ P_2\ \mathit{then}\ \mathit{if}\ P_3\ \mathit{then}\ \Omega\ \mathit{else}\ ff\ \mathit{else}\ \Omega\ \mathit{else}\ \Omega$$

where $P_1 = g\,tt\,\Omega$, $P_2 = g\,\Omega\,tt$, and $P_3 = g\,ff\,ff$. These terms are designed in such a way that

$$tt = [\![M_1]\!](por) \neq [\![M_2]\!](por) = ff.$$

On the other hand $M_1 =_{obs} M_2$. This is proved thanks to two key syntactic results:

(a) Milner's context lemma [Mil77]. This lemma, proposed as exercise 6.4.4, states that in the definition of $\leq_{obs}$ it is enough to let C range over so-called applicative contexts, of the form $[\ ]N_1 \ldots N_p$. Applying this lemma to M_1, M_2, we only have to consider contexts $[\ ]N$. By the definition of $\to_{op}$, we have for $i = 1, 2$:

$$[M_i]N \to_{op}^\star c \Rightarrow \begin{cases} N\,tt\,\Omega \to_{op}^\star tt \\ N\,\Omega\,tt \to_{op}^\star tt \\ N\,ff\,ff \to_{op}^\star ff\ . \end{cases}$$

(b) The second syntactic result that we use is that there is no N such that

$$N\,tt\,\Omega \to_{op}^\star tt \quad N\,\Omega\,tt \to_{op}^\star tt \quad N\,ff\,ff \to_{op}^\star ff\ .$$

This result is a consequence of the following more general result. PCF is a *sequential* language, in the following sense: If C is a closed program context with several holes, if

$$[\![\vdash C[\Omega, \ldots, \Omega]]\!] = \perp \quad \text{and} \quad \exists\, M_1, \ldots, M_n \ [\![\vdash C[M_1, \ldots, M_n]]\!] \neq \perp$$

then there exists an i called sequentiality index, such that

$$\forall\, N_1, \ldots, N_{i-1}, N_{i+1}, \ldots, N_n \quad [\![\vdash C[N_1, \ldots, N_{i-1}, \Omega, N_{i+1}, \ldots, N_n]]\!] = \bot\ .$$

This result is an easy consequence of (the PCF version of) Berry's syntactic sequentiality theorem 2.4.3 (see exercise 6.4.5) and of the adequacy theorem 6.3.6. Here, it is applied to $C = N[\ ][\ ]$, observing that we can use $N\ ff\ ff \to^\star_{op} ff$ to deduce that there is no c such that $M\Omega\Omega \to^\star_{op} c$.

Another way to prove the non-existence of N is by means of logical relations. We have treated essentially the same example in section 4.5.

2. Milner has shown that in an extensional standard fully abstract model of PCF, the interpretations of all types are algebraic, and their compact elements must be definable, i.e. the meaning of some closed term. This is called the definability theorem (for a proof, we refer to [Cur86]). One can use this result to cut down the path followed in (1) and go directly to step (b). In reality, there is no cut down at all, since the proof of the definability theorem uses the context lemma, and exploits terms in the style of M_1, M_2.

Exercise 6.4.4 (context lemma) ** Let M and M' be two closed* PCF *terms of the same type such that, for all closed terms $N_1, \ldots, N_n$ such that $MN_1 \cdots N_n$ is of basic type, the following holds:*

$$MN_1 \cdots N_n \to^\star_{op} c \quad \Rightarrow \quad M'N_1 \cdots N_n \to^\star_{op} c.$$

Show that $M \leq_{obs} M'$. Hint: proceed by induction on (length of the reduction $C[M] \to^\star_{op} c$, size of $C[M]$).

Exercise 6.4.5 (syntactic sequentiality for PCF**)** *Prove the* PCF *version of theorem 2.4.3, and show the corresponding corollary along the lines of exercise 2.4.4.*

The converse of the definability theorem also holds, and is easy to prove.

Proposition 6.4.6 *If* **C** *is an order-extensional standard model of* PCF *in which all cpo's interpreting all types are algebraic and are such that all their compact elements are definable, then* **C** *is fully abstract.*

PROOF. Suppose that $M \leq_{obs} M'$. It is enough to check $[\![M]\!](\vec{d}) \leq [\![M']\!](\vec{d})$ for all compact $\vec{d} = d_1 \cdots d_n$. Then the conclusion follows using contexts of the form $[\ ]N_1 \cdots N_n$. □

Exercise 6.4.7 (uniqueness) *Show, as a consequence of proposition 6.4.6 and of the definability theorem, that all order-extensional standard models of* PCF *are isomorphic (in a suitable sense).*

In fact, this (unique) fully abstract model exists, and was first constructed by Milner as a quotient of the term model of PCF. Since then, a lot of efforts have been made to provide more "semantic" constructions of this model (this is known as the full abstraction problem for PCF). In particular, the non-definability of *por* prompted the study of sequentiality, which is the subject of section 6.5 and of chapter 14. A weaker notion, stability, appeared on the way, and is the subject of chapter 12.

Remark 6.4.8 *Gunter has proposed a simple semantic proof of $M_1 =_{obs} M_2$. In the stable model of* PCF*, to be defined in chapter 12, we have $[\![M_1]\!] = [\![M_2]\!]$. In the stable model, one retains only functions which satisfy the following property (specified here for a type like $o \times o \to o$):*

$$\forall x \;\; f(x) \neq \bot \quad \Rightarrow \quad \exists y \text{ minimum} \quad (y \leq x \text{ and } f(y) \neq \bot).$$

In particular, por is rejected (take $x = (tt, tt)$, then $(\bot, tt)$ and $(tt, \bot)$ are both minimal, but there is no minimum), and this is why we have $[\![M_1]\!] = [\![M_2]\!]$. Now, because the direction $\Leftarrow$ holds for the stable model, which is standard, we have $M_1 =_{obs} M_2$.

PCF **Böhm trees.** In the rest of this section, we sketch a construction of the fully abstract model of PCF, based on a notion of Böhm tree for PCF, and due independently to Hyland and Ong, and to Abramsky, Jagadeesan and Malacaria [HO94, AJM95]. Often, our definitions are given for types built over ι only. The extension of the constructions to the full PCF type hierarchy is straightforward.

Definition 6.4.9 (PCF Böhm tree) *We define the set $\mathcal{T}^{raw}$ of raw* PCF *Böhm trees, and the auxiliary set $\mathcal{B}^{raw}$ as follows (T ranges over $\mathcal{T}^{raw}$, and B ranges over $\mathcal{B}^{raw}$):*

$$\begin{array}{lll} T ::= \lambda \vec{x} : \vec{\sigma}.B & & \\ B ::= \Omega \parallel n \parallel case\, x\vec{T}\,[F] & (n \in \omega) & \\ F : \omega \rightharpoonup \mathcal{B}^{raw} & (dom(F) \text{ finite}) & . \end{array}$$

We endow $\mathcal{T}^{raw}$ with a subtree ordering, which is the least congruence satisfying:

$$\Omega \leq T \;\; (\text{for any } T) \qquad \frac{F(n)\downarrow \Rightarrow F(n) \leq F'(n) \text{ for any } n \in \omega}{F \leq F'}$$

The set $\mathcal{T}^{raw}$ can be viewed as a subset of the set of (raw) terms in $\Lambda(C)$, with

$$C = \{\Omega, n, case_X \mid n \in \omega, X \subseteq_{fin} \omega\}$$

taking $case_{dom(F)}$ to encode $\lambda \vec{y}.case\, xT_1 \cdots T_n\,[F]$. The constants are typed as follows:

$$\vdash \Omega : \iota \quad \vdash n : \iota \quad \vdash case_X : \iota^{\sharp X + 1}.$$

There are no constants $\vdash \Omega : \sigma$ at non-basic types. A correctly typed raw PCF *Böhm tree is called a finite Böhm tree. The sets of correctly typed terms of $\mathcal{T}^{raw}$ and $\mathcal{B}^{raw}$ are denoted $\mathcal{T}$ and $\mathcal{B}$. We denote with $\mathcal{T}^\infty$ the ideal completion of $(\mathcal{T}, \leq)$ (cf. proposition 1.1.21). We use P, Q to range over T^∞, while S, T, B always denote finite trees. The completion is done at every type, and we write $\Gamma \vdash P : \tau$ whenever $\Gamma \vdash S : \tau$ for any finite approximation of P.*

Next we define a category whose morphisms are trees of $\mathcal{T}^\infty$.

Definition 6.4.10 *The category $\mathbf{BT}_{\mathrm{PCF}}$ has the following objects and morphisms:*

- *The objects of $\mathbf{BT}_{\mathrm{PCF}}$ are the sequences $\vec{\sigma}$ of* PCF *types.*
- *$\mathbf{BT}_{\mathrm{PCF}}[\vec{\sigma}, \vec{\tau}]$, with $\vec{\tau} = \tau_1, \ldots, \tau_n$, consists of a vector of trees $\vec{x} : \vec{\sigma} \vdash P_i : \tau_i$ in $\mathcal{T}^\infty$, for $i = 1, \ldots, n$.*

Given $\vec{\sigma}$ and σ in the list $\vec{\sigma}$, we define a projection morphism $\vec{x} : \vec{\sigma} \vdash \pi_{\vec{\sigma},\sigma} : \sigma$ by induction on $\sigma = \tau_1 \to \cdots \to \tau_p \to \iota$, as follows:

$$\pi_{\vec{\sigma},\sigma} \quad = \quad \lambda \vec{y}.\, case\ x(\pi_{\vec{\tau},\tau_1}) \cdots (\pi_{\vec{\tau},\tau_p})\ [id]$$

where id is the identity function mapping $n \in \omega$ to $\vdash n : \iota$. If $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$, then the identity morphism $id : \vec{\sigma} \to \vec{\sigma}$ is defined by: $id = \pi_{\vec{\sigma},\sigma_1}, \ldots, \pi_{\vec{\sigma},\sigma_n}$.

Remark 6.4.11 *The projection and identity morphisms are infinite trees, due to the presence of the identity function $\lambda n.n$ in their definition, which introduces infinite horizontal branching.*

In order to define composition, we proceed in two stages. First, we define the composition of finite morphisms, i.e. finite trees. Given $\vec{T} \in \mathbf{BT}_{\mathrm{PCF}}[\vec{\sigma}, \vec{\sigma'}]$ and $S \in \mathbf{BT}_{\mathrm{PCF}}[\vec{\sigma'}, \sigma'']$, we form $(\lambda \vec{x}.S)\vec{T}$, and reduce it to its normal form R, applying β, as well as the following rules:

$$\begin{array}{ll} (\delta) & case\ n\ [F] \to \begin{cases} F(n) & \text{if } F(n)\downarrow \\ \Omega & \text{otherwise} \end{cases} \\ (\Omega) & case\ \Omega\ [F] \to \Omega \\ (\gamma) & case\ (case\ M\ [F])\ [G] \to case\ M\ [H]\ . \end{array}$$

where H has the same domain as F and $H(n) = case\ F(n)\ [G]$. We set

$$S \circ (\vec{T}) = R.$$

Finally, composition is extended to infinite trees by continuity:

$$P \circ (\vec{Q}) = \bigvee \{S \circ (\vec{T}) \mid S \leq P, \vec{T} \leq \vec{Q}\}.$$

We now have to justify all this carefully. We have to show that:

1. R always exists,
2. $R \in \mathcal{T}$,
3. The lub in the definition of $P \circ (\vec{Q})$ exists,
4. composition satisfies the monoid laws.

As for (1), we rely on the following theorem due to Breazu-Tannen and Gallier [BTG91].

Theorem 6.4.12 *Let $\Lambda(C)$ be a simply-typed [1] λ-calculus with constants whose type has rank at most 1. Let R be a set of strongly normalising rewriting rules for first-order terms written with the (uncurried) signature C. Then the rewriting system $\beta + R$ (with the curried version of R) over $\Lambda(C)$ is strongly normalizing.*

We instantiate R as $(\delta) + (\Omega) + (\gamma)$.

Proposition 6.4.13 *The system $(\delta) + (\Omega) + (\gamma)$, considered as a first-order rewriting system, is strongly normalizing.*

PROOF. We use a technique inspired from exercise 2.2.23. We call Φ the set of first-order terms built over the uncurried signature $C = \{\Omega, n, case_X \mid n \in \omega, X \subseteq_{fin} \omega\}$. We define a subset of Ψ defined as the least set closed under the following rules, where $F \in \Psi$ stands for $\forall n \ (F(n)\downarrow \Rightarrow F(n) \in \Psi)$:

1. $s \in \Psi$ if $s = \Omega, n,$ or x,
2. $case\ s\ [F] \in \Psi$ if $s = \Omega, n,$ or x and if $F \in \Psi$,
3. $case\ (case\ s\ [F])\ [G] \in \Psi$ if $G \in \Psi$ and if $case\ s\ [H] \in \Psi$, where H is as in rule (γ).

We claim that for all s and F, if $s \in \Phi$ and $G \in \Psi$, then $case\ s\ [G] \in \Psi$. We prove this by induction on the size of s only:

- If $s = \Omega, n,$ or x, then $case\ s\ [G] \in \Psi$ by (2).
- If $s = case\ t\ [F]$, then we have to prove $case\ t\ [H] \in \Psi$, which holds by induction, provided we prove first $H \in \Psi$. But this holds by induction too, since $H(n) = case\ F(n)\ [G]$.

The claim a fortiori implies that Ψ is closed under $case\ \_\ [\ ]$, hence $\Psi = \Phi$. The interest of the presentation of Φ as Ψ is that we can prove strong normalisation of s by induction on the proof of $s \in \Psi$, as follows:

(1) Then s is in normal form.

[1] This theorem is actually proved in [BTG91] for the polymorphic λ-calculus.

(2) Then we know by induction that $F(n)$ is strongly normalizing whenever $F(n)$ is defined, and we conclude by noticing that a reduct of s is either $case\ s\ [F']$ (where F pointwise reduces to F'), or Ω, or t, where t is a reduct of $F(n)$ for some n.

(3) We know by induction that G is pointwise strongly normalizing, and that $case\ s\ [H]$ is strongly normalizing. In particular, s is strongly normalizing, and, by the definition of H, F is pointwise strongly normalizing. Therefore an infinite reduction from $case\ (case\ s\ [F])\ [G]$ can only be of the form

$$case\ (case\ s\ [F])\ [G] \to^* case\ (case\ s'\ [F'])\ [G'] \to case\ s'\ [H']$$

where H' is defined from F' and G' as H is defined from F and G. It follows that $case\ s'\ [H']$ is a reduct of $case\ s\ [H]$, and is therefore strongly normalizing. □

Exercise 6.4.14 ** Prove directly that $\beta\delta\Omega\gamma$ is strongly normalizing, by adapting the proof of theorem 3.5.20. Hint: prove by contradiction that the set of $\beta\delta\Omega\gamma$ strongly normalisable terms is closed under case $[]$, exploiting the strong normalisation of β alone, and proposition 6.4.13: the two kinds of reduction "do not mix".*

To establish that $R \in \mathcal{T}$, we define a subset Ξ of $\Lambda(C)$ (with C as above), within which all the reductions which interest us take place. The syntax of raw terms of Ξ is defined as follows:

$$\begin{array}{lll} T ::= \lambda\vec{x}.B \parallel (\lambda\vec{x}.T)\vec{S} & (length(\vec{S}) = length(\vec{x})) \\ B ::= \Omega \mid n \parallel case\ A\ [F] & (n \in \omega) \\ A ::= xT_1 \cdots T_n \parallel B \parallel (\lambda\vec{x}.B)\vec{S} & (length(\vec{S}) = length(\vec{x})) \\ F : \omega \rightharpoonup \mathcal{B} & (dom(F) \text{ finite}) . \end{array}$$

We define the following multiple version $\vec{\beta}$ of β-reduction:

$$(\vec{\beta}) \quad (\lambda\vec{x}.T)\vec{S} \to T[\vec{S}/\vec{x}].$$

The following properties are easily checked:

- The set Ξ is stable under the reductions $\vec{\beta}$, γ, and δ.
- The $\vec{\beta}\delta\Omega\gamma$ normal form of a term of Ξ is a $\beta\delta\Omega\gamma$-normal form, and belongs to T.

Hence $R \in T$. The fact that $P \circ (\vec{Q})$ is well-defined is a consequence of the following property, which is easy to check: if $S \to S'$ and if $S \leq T$, then $T \to T'$, where $\to$ is $\vec{\beta}\delta\Omega\gamma$ reduction. It follows from the claim that $\{S \circ (\vec{T}) \mid S \leq P, \vec{T} \leq \vec{Q}\}$ is directed.

We now show that the monoid laws hold. We examine associativity first. By definition, $(S \circ (T_1 \cdots T_n)) \circ (\vec{T'})$ is the normal form of

$$(\lambda \vec{x'}.(\lambda \vec{x}.S)T_1 \cdots T_n)\vec{T'}$$

while $S \circ (T_1 \circ (\vec{T'}) \cdots T_n \circ (\vec{T'}))$ is the normal form of

$$(\lambda \vec{x}.S)(((\lambda \vec{x'}.T_1)\vec{T'}) \cdots ((\lambda \vec{x'}.T_n)\vec{T'}))$$

and these two terms are $\vec{\beta}$ equal to $(\lambda \vec{x}.S)(T_1[\vec{T'}/\vec{x'}] \cdots T_n[\vec{T'}/\vec{x'}])$. Hence associativity holds for finite trees, which implies the associativity for infinite trees by continuity.

As for the identity laws, consider, say, $S \circ id$. We construct by induction on S a finite subtree $id_S \leq id$ such that $S \circ id_S = S$. We only examine the essential case $S = case\ x_i\vec{T}\ [F]$. We choose id_S (least) such that $id_T \leq id_S$ for each $T \in \vec{T}$ and such that the i-th component of id_S has the form $case\ \_\ [G]$ with $dom(F) \subseteq dom(G)$ (and of course $G(n) = n$ whenever $G(n)\downarrow$). One reasons similarly for the other identity law.

The product structure is trivial by construction, since the morphisms of the category are vectors: products of objects and pairing of arrows are their concatenations, while projection morphisms are defined with the help of the morphisms $\pi_{\vec{\sigma},\sigma}$. Finally, the exponent structure is also obvious. We set

$$\vec{\sigma} \to (\tau_1 \cdots \tau_n) = (\vec{\sigma} \to \tau_1 \cdots \vec{\sigma} \to \tau_n)$$

and use multiple abstraction to define currying.

Theorem 6.4.15 *The category* $\mathbf{BT}_{\mathrm{PCF}}$ *is a standard model of* PCF*, in which all compact elements of the interpretations of all types are definable (by terms without* Y*).*

PROOF HINT. We have already sketched the proof that $\mathbf{BT}_{\mathrm{PCF}}$ is a CCC. The homsets are obviously cpo's, and it is easy to check that $\mathbf{BT}_{\mathrm{PCF}}$ is a cpo-enriched CCC. The only closed trees of basic type are the trees n and Ω. The PCF constants are given their obvious interpretation, e.g. $[\![succ]\!] = \lambda x.case\ x\ [succ]$, where the second occurrence of *succ* is the usual successor function on ω. The fact that all compact elements are definable is tedious, but easy, to verify, with arguments similar to the ones we have used to justify the identity laws. For example, the tree $case\ x\ [F]$ where $F(1) = 4$ and $F(3) = 1$ is defined by

$$if\ pred\ x\ then\ 4\ else\ (if\ pred(pred(pred\ x))\ then\ 1\ else\ \Omega).$$

□

We have thus obtained a standard model whose compact elements are all definable. What we lack is extensionality. By extensional collapse (cf. exercise 4.5.6), we can obtain a category $[\mathbf{BT}_{\mathrm{PCF}}]$ with enough points. It remains to see whether this category is cpo-enriched. It turns out that it has enough limits to make it possible to interpret Y and the (Y)-rule, and thus to obtain a fully abstract model of PCF because the category $[\mathbf{BT}_{\mathrm{PCF}}]$ inherits from $\mathbf{BT}_{\mathrm{PCF}}$ the property that all its "compact" elements are definable (see exercises 6.4.16 and 6.4.17). However $[\mathbf{BT}_{\mathrm{PCF}}]$ is (a priori) not the unique order-extensional model of PCF (cf. exercise 6.4.7). To obtain the latter, we go through a slightly more involved construction. We build a logical-like collapse relation over compact PCF Böhm trees, and we then perform an ideal completion of the quotient. This guarantees by construction that the resulting category $[\mathbf{BT}_{\mathrm{PCF}}]^{\infty}$ is cpo-enriched. However there is still a subtle point in showing that extensionality is preserved by the completion. For this, we have to resort to finite projections (cf. section 5.2). We give more details in exercises 6.4.18 and 6.4.19.

Exercise 6.4.16 *Let* $\mathbf{C}$ *be a cpo-enriched CCC, and let* $[\mathbf{C}]$ *be its extensional collapse (cf. exercise 4.5.6), whose homsets are ordered pointwise. (1) Show that* $[\mathbf{C}]$ *is rational (in the terminology of [AJM95]), i.e. satisfies the following properties: (1) all homsets have a* $\bot$*; (2) for any* A, B *and any* $f : A \times B \to A$*, the sequence* $\{f^n\}_{n<\omega}$ *defined by* $f^0 = \bot$ *and* $f^{k+1} = f \circ \langle id, f^k \rangle$ *has a lub; (3) those lub's are preserved by left and right composition. (2) Show that if* $\mathbf{C}$ *is a standard model of* PCF*, then* $[\mathbf{C}]$ *is an order-extensional model of* PCF*.*

Exercise 6.4.17 *Show that* $[\mathbf{BT}_{\mathrm{PCF}}]$ *is a fully abstract model of* PCF*. Hints: use exercise 6.4.16, and adapt the proof of proposition 6.4.6 to the rational case.*

Exercise 6.4.18 *Let* $\mathbf{C}$ *be a cpo-enriched CCC whose homsets are all algebraic, and which satisfies:*

1. Compact morphisms are closed under composition, currying, and uncurrying.

2. For any compact f *there exists a compact morphism* $id_f \leq f$ *such that* $f \circ id_f = id_f \circ f = f$*.*

3. For any type σ*, interpreted by* D^σ*, there exists a sequence of compact morphisms* $\psi_n^{D^\sigma} : \mathbf{C}[D^\sigma, D^\sigma]$ *such that* $\bigvee_{n<\omega} \psi_n^{D^\sigma} = id$ *and (for all* σ, τ*)* $\psi_n^{D^{\sigma\to\tau}} = [\![\lambda f x.g(f(h(x)))]\!] \circ \langle \psi_n^{D^\sigma}, \psi_n^{D^\tau} \rangle$*.*
4. Moreover, at base types, $\{\psi_n^{D^\kappa} \circ h \mid h : 1 \to D^\kappa\}$ *is finite, for all* n*.*

Define a logical-like relation R *on* compact *morphisms (hence such that* $\mathcal{R}^\sigma \subseteq \mathcal{K}(\underline{D^\sigma})$*, where* D^σ *is the interpretation of* σ *and where* $\underline{A} = \mathbf{C}[1, A]$*), by setting*

$$\mathcal{R}^\kappa = \{(d, d) \mid d \in \mathcal{K}(\underline{D^\kappa})\}$$

and by extending R *to all types as in definition 4.5.1. Define a category* $[\mathbf{C}]^\infty$ *whose objects are the types and whose homsets are the ideal completions of the sets of* $\mathcal{R}^{\sigma,\tau}$ *equivalence classes (in the terminology of definition 4.5.1), ordered pointwise. Show that* $[\mathbf{C}]^\infty$ *is an order-extensional cpo-enriched CCC, and that there is a functor from*

$\mathbf{C}$ *to* $[\mathbf{C}]^\infty$ *which preserves the cartesian closed structure. Show that the functor maps compact morphisms surjectively onto compact morphisms. Hint: Show that* $[\psi_n]$ *has a finite image (cf. proof of proposition 5.2.4), and exploit the fact that for any compact* f *there exists* n *such that* $\psi_n \circ f = f$*, by (2) and (3).*

Exercise 6.4.19 *Show that* $\mathbf{BT}_{\mathrm{PCF}}$ *satisfies the conditions stated in exercise 6.4.18, and that* $[\mathbf{BT}_{\mathrm{PCF}}]^\infty$ *is the unique fully abstract model of exercise 6.4.7.*

Remark 6.4.20 *What we have done to construct the fully abstract model can be summarised as: "complete, then quotient (the base), and finally complete". Originally, Milner had not gone through the first of these steps. An advantage of the presentation chosen here is that: (1) it is reasonable (and simpler than Milner's original construction) to stop at the second stage (exercise 6.4.17); (2) it singles out some general conditions to obtain an extensional least fixpoint model out of least fixpoint model.*

The category $\mathbf{BT}_{\mathrm{PCF}}$ is a full subcategory of two categories of games, constructed recently by Hyland and Ong, and by Abramsky, Jagadeesan, and Malacaria [HO94, AJM95]. The striking point about these categories of games is that their construction does not refer to the syntax.

It is presently unknown whether the fully abstract model of PCF can be effectively presented, i.e. whether its elements can be recursively enumerated. A related open problem is whether the observational equivalence is decidable for Finitary PCF. A positive answer for this problem would follow from a positive answer to the definability problem for Finitary PCF (cf. section 4.5).

Exercise 6.4.21 *Using Statman's 1-section theorem 4.5.9, show that for any simply typed* λ*-terms* M, N*, considered as* PCF *terms,* $M =_{obs} N$ *iff* $M =_{\beta\eta} N$*. Hint: proceed as in the proof of Friedman's theorem via the 1-section theorem.*

6.5 Towards Sequentiality

We have already pointed out that λ-calculus is sequential (theorem 2.4.3). In section 4.5, we have exhibited an example of an inherently parallel function which is not definable in a (finitary version of) PCF. In this section, we give further evidence of the sequential nature of PCF. We define sequential functions in a restricted setting, which will be later extended in chapter 14. We show that the compact definable first-order functions of the continuous model of PCF are exactly the (compact) first-order sequential functions.

Definition 6.5.1 (sequential function (Vuillemin)) *Let* $D, D_1, \ldots, D_n$ *be flat cpo's, and let* $f : D_1 \times \cdots \times D_n \to D$ *be monotonic (hence continuous). Let* $x = (x_1, \ldots, x_n) \in D_1 \times \cdots \times D_n$*, and suppose that* $f(x) = \bot$*. We say that* f *is*

sequential at x if either $f(z) = \bot$ for all $z \geq x$, or there exists i such that $x_i = \bot$ and

$$\forall\, y = (y_1, \ldots, y_n)\ \ (y > x \text{ and } f(y) \neq \bot) \Rightarrow y_i \neq \bot.$$

We say then that i is a sequentiality index for f at x.

The above definition goes back to [Vui74]. The following easy proposition offers an alternative definition of sequential functions over flat cpo's.

Proposition 6.5.2 *Let $D = X_\bot$ be a flat cpo. The sets of sequential functions from products of flat domains to D are alternatively defined as follows, by induction on their arity n:*

Arity 1 : Any monotonic function $f : D \to D$ is sequential.

Arity $n \geq 2$: Given n, $i \leq n$, $X \subseteq \omega$, and a set

$$\{f_x : D_1 \times \cdots D_{i-1} \times D_{i+1} \cdots \times D_n \to D \mid x \in X\}$$

then the following function f is sequential:

$$\begin{array}{lcl} f(x_1, \ldots, x_{i-1}, \bot, x_{i+1}, \ldots, x_n) & = & \bot \\ f(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n) & = & f_x(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)\ . \end{array}$$

Moreover, the compact sequential functions are exactly the functions obtained as above, with X finite at each induction step.

PROOF. In both directions, the proof goes by induction on the arity. The two parts of the statement are proved together. If f is sequential, then we pick a sequentiality index i at $\bot$, and we define the f_x's by the second equation in the statement. They are clearly sequential, hence induction applies to them. If f is compact, X cannot be infinite, as otherwise f would be the lub of an infinite sequence of functions obtained by cutting down X to its finite subsets. Conversely, let f constructed as in the statement and x such that $f(x) = \bot$ and $f(z) \neq \bot$ for some $z > x$. There are two cases:

$x_i = \bot$: Then i is a sequentiality index at x.

$x_i = j \neq \bot$: Then $j \in X$ and by induction f_j has a sequentiality index at $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$, which is a sequentiality index of f at x.
If the X's are all finite, then the description of f is finite, from which compactness follows easily. □

Exercise 6.5.3 *Show that the C-logical relations, where C is the set of all the constants of* PCF *(including Y), are exactly the Sieber sequential relations of definition 4.5.16. Hint: show that at each type σ, the set of invariant elements of a Sieber sequential relation forms an inclusive predicate.*

Hence (compact) sequential functions on flat domains can be described by sequential "programs", which can actually be written as PCF terms, as the following proposition shows.

Theorem 6.5.4 *For a compact first-order function f of the continuous model of* PCF *(and more generally for a function from a product of flat domains to a flat domain), the following properties are equivalent:*

1. f is sequential.

2. f is definable in the following restriction $\Lambda(C')$ of PCF *(with Ω of base type):*

$$C' = \{\Omega, n, tt, ff, pred, zero?, if \quad then \quad else\ \}.$$

3. f is definable in PCF*.*

4. f is invariant under all $k+1$-ary relations S_{k+1} ($k \geq 1$) defined at ground type by

$$(x_1, \ldots, x_{k+1}) \in S_{k+1} \quad \Leftrightarrow \quad (\exists\, j \leq k \;\; x_j = \bot) \text{ or } (x_1 = \ldots = x_{k+1} \neq \bot)).$$

These relations are special cases of Sieber sequential relations, cf. definition 4.5.16. More precisely: $S_{k+1} = S^{k+1}_{\{1,\ldots,k\},\{1,\ldots,k+1\}}$.

PROOF. (1) $\Rightarrow$ (2) It is easy to check by induction on terms that the functions defined by the restricted syntax are monotonic. It is also easy to see that the restricted syntax allows to encode all compact sequential functions, as characterised in proposition 6.5.2 (cf. proof of theorem 6.4.15). Hence the interpretation function is a surjection from the restricted syntax to the set of compact first-order sequential functions.

(2) $\Rightarrow$ (3) Obvious by inclusion.

(3) $\Rightarrow$ (4) This follows from lemma 4.5.3 and from exercise 6.5.3.

(4) $\Rightarrow$ (1) Suppose that f is not sequential. Then there exists $x = (x_1, \ldots, x_n)$ such that:

$f(x) = \bot$
$J = \{j \leq n \mid x_j = \bot\} \neq \emptyset$
$\forall\, j \in J \;\; \exists\, y_j = (y_{1j}, \ldots, y_{nj}) \;\; ((\forall\, i \notin J \;\; y_{ij} = x_i)$ and $y_{jj} = \bot$ and $f(y_j) \neq \bot)$.

Without loss of generality, we can assume that $J = \{1, \ldots, k\}$ for some $k \geq 1$ (and $\leq n$). We claim that $(y_{i1}, \ldots, y_{ik}, x_i) \in S_{k+1}$ for all $i \leq n$. This follows from the following easy case analysis.

$i \notin J$: then $y_{i1} = \ldots = y_{ik} = x_i$.
$i \in J$: then $y_{ii} = \bot$.

Hence, by invariance: $(f(y_1), \ldots, f(y_k), f(x)) \in S_{k+1}$, which contradicts the definition of S_{k+1}, since we have assumed $f(y_j) \neq \bot$ for all $j \leq k$ and $f(x) = \bot$. $\Box$

We don't know how to extend this correspondence to higher orders. Both the model of sequential algorithms and the strongly stable model, which is built in chapter 14 and section 13.3, respectively, contain non PCF-definable functionals such as the functionals presented in exercises 4.5.18 and 4.5.19.

Exercise 6.5.5 *Show that every* PCF *definable first-order function in a standard model of* PCF *is sequential. Hint: given a closed term M, call M_n the term obtained by replacing Y by $\lambda f.f^n\Omega$, and let P_n be the normal form of M_n. Show that the P_n's form a directed set, and exploit this to show that they all contribute to a single sequential function defined as in proposition 6.5.2.*

Chapter 7

Domain Equations

This chapter presents general techniques for the solution of domain equations and the representation of domains and functors over a universal domain. Given a category of domains $\mathbf{C}$ we build the related category $\mathbf{C}^{ip}$ (cf. chapter 3) that has the same objects as $\mathbf{C}$ and *injection-projection pairs* as morphisms (section 7.1). It turns out that this is a suitable framework for the solution of domain equations. The technique is applied in section 7.2 in order to solve a *predicate equation*. The solution of the predicate equation is used in proving an adequacy theorem for a simple declarative language with dynamic binding. The category of injection-projection pairs is also a suitable framework for the construction of a *universal homogeneous object* (section 7.3). The latter is a domain in which every other domain (not exceeding a certain size) can be embedded. Once a universal object U is built, it is possible to *represent* the collection of domains as the domain $FP(U)$ of *finitary projections* over U, and functors as continuous functions over $FP(U)$. In this way, one obtains a poset-theoretical framework for the solution of domain equations that is more manageable than the general categorical one (section 7.4).

A third approach to the solution of domain equations consists in working with concrete representations of domains like information systems, event structures, or concrete data structures (introduced in definitions 10.2.11, 12.3.3 and 14.1.1, respectively). At this level, domain approximation can be modeled by means of inclusions relating the representing structures, and domain equations can be then solved as ordinary fixpoint equations. As in the finitary projections approach the solutions obtained are exact solutions ($F(D) = D$, and not merely $F(D) \cong D$). This was first remarked by Berry in the framework of concrete data structures. We do not detail this approach here (a good reference is [Win93][Chapter12]). See however exercises 10.2.14 and 14.1.13.

7.1 Domain Equations

One of the earliest problems in denotational semantics was that of building a model of the untyped $\lambda\beta\eta$-calculus. This boils down to the problem of finding a non-trivial domain D isomorphic to its functional space $D \to D$ (cf. chapter 3). Following work by Wand, Smyth and Plotkin [Wan79, SP82], we present a generalization of the technique proposed by Scott [Sco72] for the solution of domain equations.

An ω-chain is a sequence $\{B_n, f_n\}_{n\in\omega}$ such that $f_n : B_n \to B_{n+1}$ for all n. We write $f_{n,m} = f_{m-1} \circ \cdots \circ f_n$ for $m > n$. The general categorical definition of colimit (cf. section B.2) specializes to ω-chains as follows. A *cocone* $\{B, g_n\}_{n\in\omega}$ of the ω-chain $\{B_n, f_n\}_{n\in\omega}$ is given by an object B, and a sequence $\{g_n : B_n \to B\}_{n\in\omega}$ satisfying $g_{n+1} \circ f_n = g_n$ for all n. A cocone $\{B, g_n\}_{n\in\omega}$ is a *colimit* if it is an initial object in the category of cocones, that is if for any other cocone $\{C, h_n\}_{n\in\omega}$ there exists a unique morphism $k : B \to C$ such that $k \circ g_n = h_n$ for all n.

Let $T : \mathbf{K} \to \mathbf{K}$ be an endo-functor. We outline some rather general results that guarantee the existence of an *initial* solution for the equation $TX \cong X$. It will be shown next that these results can be usefully applied to the solution of domain equations.

Definition 7.1.1 (T-algebra) *Let $T : \mathbf{K} \to \mathbf{K}$ be an endo-functor. A T-algebra is a morphism $\alpha : TA \to A$. T-algebras form a category. If $\alpha : TA \to A$ and $\beta : TB \to B$ are T-algebras then a morphism from α to β is a morphism $f : A \to B$ such that $f \circ \alpha = \beta \circ Tf$.* [1]

Lemma 7.1.2 *Every initial T-algebra is an isomorphism.*

PROOF. Let $\alpha : TA \to A$ be initial. Then $T\alpha : TTA \to TA$ is also a T-algebra and by initiality there is $i : A \to TA$ such that:

$$i \circ \alpha = T\alpha \circ Ti = T(\alpha \circ i) \ . \tag{7.1}$$

We observe that α is a morphism (of T-algebras) from $T\alpha$ to α. By composition and initiality we get $\alpha \circ i = id$. By the equation 7.1 above we derive:

$$T(\alpha \circ i) = T(id) = id = i \circ \alpha \ .$$

So i is the inverse of α. □

The following proposition will appear natural if one thinks of categories as cpo's and of functors as continuous functions.

[1] A stronger notion of T-algebra is given in definition B.8.3 in the case T is the functor component of a monad.

Proposition 7.1.3 *Let* $\mathbf{C}$ *be a category with initial object and* ω*-colimits and* $T : \mathbf{C} \to \mathbf{C}$ *be a functor that preserves* ω*-colimits. Then there is an initial* T*-algebra.*

PROOF. Let 0 be the initial object. Consider the uniquely determined morphism $z : 0 \to T0$. By iterating T on this diagram we get an ω-diagram $D = \{T^i 0, T^i z\}_{i<\omega}$. By assumption there is an ω-colimit of D, say $C = \{A, f_i\}_{i<\omega}$, satisfying $f_i = f_{i+1} \circ T^i z$, for all i.

Now consider $TC = \{TA, Tf_i\}_{i<\omega}$. By assumption TC is an ω-colimit of $TD = \{TT^i 0, TT^i z\}_{i<\omega}$. Since we can restrict C to a cocone of TD we have determined a unique morphism $h : TC \to C$.

Moreover, we want to prove that the T-algebra $h : TA \to A$ is initial. This goes in three steps:

(1) Observe that any T-algebra, $\beta : TB \to B$, gives rise to a cocone $\{B, g_i^B\}_{i<\omega}$ where:

$$g_i^B = \beta \circ T\beta \circ TT\beta \circ \cdots \circ T^{i-1}\beta \circ T^i z_B \quad i < \omega,\ z_B : 0 \to B\ .$$

It is enough to check that $g_{i+1}^B \circ T^i z = g_i^B$, which follows from $\beta \circ Tz_B \circ z = z_B$.

(2) Any morphism of T-algebras $u : \alpha \to \beta$, where $\alpha : TA' \to A'$ and $\beta : TB \to B$, induces a morphism between the related cocones over D, as defined in (1). Suppose $\beta \circ Tu = u \circ \alpha$. Then:

$$\begin{aligned} u \circ g_i^{A'} &= u \circ \alpha \circ T\alpha \circ \cdots \circ T^{i-1}\alpha \circ T^i z_{A'} \\ &= \beta \circ Tu \circ T\alpha \circ \cdots \circ T^{i-1}\alpha \circ T^i z_{A'} \\ &= \cdots \\ &= \beta \circ T\beta \circ \cdots \circ T^{i-1}\beta \circ T^i z_B \\ &= g_i^B\ . \end{aligned}$$

Hence there is at most one T-algebra morphism $u : h \to \beta$.

(3) To prove existence we relate morphisms in $\mathbf{Cocone}\{T^i 0, T^i z\}_{i<\omega}$ to morphisms of T-algebras. Given $h : TA \to A$, $\beta : TB \to B$ there is a uniquely determined morphism $l : A \to B$ on the induced cocones over D. We observe that Tl is a morphism from $\{TA, Tf_i\}_{i<\omega}$ to $\{TB, Tg_i^B\}_{i<\omega}$ of cocones over TD. Moreover, β is a morphism from $\{TB, Tg_i^B\}_{i<\omega}$ to $\{B, g_i^B\}_{i\geq 1}$ of cocones over TD as $g_{i+1}^B = \beta \circ Tg_i^B$. By initiality of TC on TD it follows that $l \circ h = \beta \circ Tl$. □

When solving domain equations, we may wish to start the construction of the ω-diagram with some morphism $z : X \to TX$, where X is not necessarily an initial object (cf. definition 3.1.6). In the poset case this corresponds to looking for the least fixed point of a function $f : D \to D$, above a given point d such that $d \leq f(d)$. If D is an ω-dcpo, and f is ω-continuous then we can compute

the solution as $\bigvee_{n<\omega} f^n(d)$. This is the least element of the set $\{e \in D \mid f(e) \leq e \text{ and } d \leq e\}$.

We provide a categorical generalization of this fact. Suppose that the category $\mathbf{C}$ and the functor F satisfy the conditions in proposition 7.1.3. Given a morphism $z : X \to TX$ we can build an ω-diagram $D = \{T^iX, T^iz\}_{i<\omega}$. Using the hypotheses we can build its colimit $\{A, f_i\}_{i<\omega}$ and a morphism $h : TA \to A$.

The problem is now to determine in which framework h is initial. In first approximation it is natural to consider T-algebras $\beta : TB \to B$ together with a morphism $z_B : X \to B$ (as B has to be "bigger" than X). If we mimic step (1) in the proof of proposition 7.1.3, that builds a cocone out of a T-algebra, we see that we need the following property:

$$z_B = \beta \circ Tz_B \circ z \tag{7.2}$$

Generalizing step (2) presents a new difficulty. It appears that a T-algebra morphism $l : \beta \to \gamma$, where $\beta : TB \to B$, and $\gamma : TC \to C$, should also satisfy:

$$l \circ z_B = z_C \tag{7.3}$$

The following categorical formalization shows that this is just an instance of the problem we have already solved, but with respect to a related category $\mathbf{C} \uparrow X$, and a related functor T_z.

Definition 7.1.4 *Given a category* $\mathbf{C}$ *and an object* $X \in \mathbf{C}$*, we define the* slice *category* $\mathbf{C} \uparrow X$ *as follows (there is a related slice category* $\mathbf{C} \downarrow X$ *which is introduced in example B.1.5)*

$$\mathbf{C} \uparrow X = \{f : X \to B \mid B \in \mathbf{C}\} \qquad (\mathbf{C} \uparrow X)[f, g] = \{h \mid h \circ f = g\} \ .$$

Also given a functor $T : \mathbf{C} \to \mathbf{C}$*, and a morphism* $z : X \to TX$ *we define a new functor* $T_z : \mathbf{C} \uparrow X \to \mathbf{C} \uparrow X$ *as follows:*

$$T_z(f) = Tf \circ z \quad T_z(h) = Th \ .$$

Proposition 7.1.5 *Let* $\mathbf{C}$ *be a category with initial object and* ω*-colimits and* $T : \mathbf{C} \to \mathbf{C}$ *be a functor that preserves* ω*-colimits. The category* $\mathbf{C} \uparrow X$ *has initial object and* ω*-colimits, moreover given a morphism* $z : X \to TX$*, the functor* T_z *preserves* ω*-colimits.*

PROOF HINT. The commutation conditions displayed in equations 7.2 and 7.3 arise as a consequence of the abstract definitions. We show that there is a morphism $h : TA \to A$ which is the initial T_z-algebra. □

Consider the functor $\Rightarrow : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{C}$, defined in every CCC, that given objects A, B returns the exponent $A \Rightarrow B$ (with the standard extension to

morphisms). We would like to find solutions to equations such as $X = X \Rightarrow D$, or $X = X \Rightarrow X$. We recall from chapter 3 that there is no way we can look at $\lambda X.X \Rightarrow D$, or $\lambda X.X \Rightarrow X$ as (covariant) functors. We will introduce new structures that allow us to see the problem as an instance of the one solved by proposition 7.1.3. In the first place we present the notion of injection-projection pair in an *O-category* [Wan79].

Definition 7.1.6 (O-category) *A category* $\mathbf{C}$ *is called an O-category if (1) every hom-set is an* ω*-directed complete partial order, and (2) composition of morphisms is a continuous operation with respect to the orders of the hom-sets.*

Next we formulate some familiar notions (cf. chapter 3) in the framework of O-categories.

Definition 7.1.7 (retraction, injection, projection) *Let* $\mathbf{C}$ *be an O-category, and let* $A, B \in \mathbf{C}$.

(1) *A retraction from* A *to* B *is a pair* (i, j) *such that* $i : A \to B$, $j : B \to A$, $j \circ i = id_A$ *(we write then* $A \lhd B$*).*

(2) *An injection-projection from* A *to* B *is a pair* (i, j) *which is a retraction as above and such that* $i \circ j \leq id_B$ *(we write then* $A \underline{\lhd} B$*).*

(3) *A projection on* A *is a morphism* $p : A \to A$ *such that* $p \circ p = p$ *and* $p \leq id_A$.

Example 7.1.8 Cpo *is an O-category, ordering the morphisms pointwise. We note that in an injection-projection pair, injection and projection are strict (cf. definition 1.4.17) functions.*

Definition 7.1.9 *Let* $\mathbf{C}$ *be an O-category. The category* $\mathbf{C}^{ip}$ *has the same objects as* $\mathbf{C}$ *and injection-projection pairs as morphisms:*

$$\mathbf{C}^{ip}[A, B] = \{(i, j) \mid i : A \to B, j : B \to A, j \circ i = id_A, i \circ j \leq id_B\} \ .$$

Composition is given by $(i, j) \circ (i', j') = (i \circ i', j' \circ j)$*, identities by* (id, id).

Proposition 7.1.10 *Let* $\mathbf{C}$ *be an O-category. Then:*

(1) $\mathbf{C}^{ip}$ *is a category in which all morphisms are monos.*

(2) *If* $\mathbf{C}$ *has a terminal object, if each hom-set* $\mathbf{C}[A, B]$ *has a least element* $\perp_{A,B}$*, and if composition is left-strict (i.e.* $f : A \to A'$ *implies* $\perp_{A',A''} \circ f = \perp_{A,A''}$*), then* $\mathbf{C}^{ip}$ *has an initial object.*

PROOF. (1) Suppose: $(i, j) \circ (i', j') = (i, j) \circ (i'', j'')$. That is $(i \circ i', j' \circ j) = (i \circ i'', j'' \circ j)$. Since i is a mono, $i \circ i' = i \circ i''$ implies $i' = i''$. Therefore, by proposition 3.1.3, $j' = j''$.

(2) Let 1 be the terminal object in $\mathbf{C}$. We show that 1 is initial in $\mathbf{C}^{ip}$. Given $A \in \mathbf{C}$, we first show $(\perp_{1,A}, \perp_{A,1}) \in \mathbf{C}^{ip}[1, A]$. On one hand, $\perp_{A,1} \circ \perp_{1,A} = id_1$

since 1 is terminal, on the other hand $\perp_{1,A} \circ \perp_{A,1} = \perp_{A,A} \leq id_A$ since composition is left strict. There are no other morphisms in $\mathbf{C}^{ip}[1, A]$ since $\perp_{A,1}$ is the unique element of $\mathbf{C}[A, 1]$. □

We are now in a position to suggest what the category of injection-projection pairs is good for. Given a functor $F : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{C}$, we build a functor $F^{ip} : \mathbf{C}^{ip} \times \mathbf{C}^{ip} \to \mathbf{C}^{ip}$ which coincides with F on objects. In particular the exponent functor is transformed into a functor which is covariant in both arguments. We then observe that $F^{ip}(D, D) \cong D$ in $\mathbf{C}^{ip}$ implies $F(D, D) \cong D$ in $\mathbf{C}$.

In other words, we build a related structure, $\mathbf{C}^{ip}$, and we consider a related problem, $F^{ip}(D, D) \cong D$, whose solutions can be used for the initial problem. The advantage of the related problem is that we only have to deal with covariant functors and therefore we are in a favorable position to apply proposition 7.1.3. Towards this goal, it is natural to look for conditions on $\mathbf{C}$ that guarantee that $\mathbf{C}^{ip}$ has ω-colimits (we already know that under certain conditions it has an initial object) as well as for conditions on F that guarantee that F^{ip} is ω-cocontinuous, i.e. it preserves ω-cochains (cf. section B.2).

Definition 7.1.11 (locally continuous) *Let $\mathbf{C}$ be an O-category and $F : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{C}$ be a functor (the generalization to several arguments is immediate). We say that F is locally monotonic (continuous) if it is monotonic (continuous) w.r.t the orders on the hom-sets.*

Exercise 7.1.12 *Verify that the product and exponent functors on $\mathbf{Cpo}$ are locally continuous.*

There is a standard technique to transform a covariant-contravariant monotonic functor on $\mathbf{C}$ into a covariant functor on $\mathbf{C}^{ip}$.

Definition 7.1.13 *Given $F : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{C}$ define $F^{ip} : \mathbf{C}^{ip} \times \mathbf{C}^{ip} \to \mathbf{C}^{ip}$ as follows:*

$$\begin{array}{ll} F^{ip}(c, c') & = F(c, c') \\ F^{ip}((i, j), (i', j')) & = (F(j, i'), F(i, j')) \ . \end{array}$$

Exercise 7.1.14 *Verify that F^{ip} as defined above is a functor.*

The following result points out that the ω-colimit of $\{D_n, (i_n, j_n)\}_{n\in\omega}$ in $\mathbf{C}^{ip}$ can be derived from the ω^{op}-limit of $\{D_n, j_n\}_{n\in\omega}$ in $\mathbf{C}$. One often refers to this situation as limit-colimit coincidence.

Theorem 7.1.15 (limit-colimit coincidence) *Let $\mathbf{C}$ be an O-category. If $\mathbf{C}$ has ω^{op}-limits then $\mathbf{C}^{ip}$ has ω-colimits.*

PROOF. Consider an ω-chain $\{D_n, f_n\}_{n\in\omega}$ in $\mathbf{C}^{ip}$ where we denote with $f_n^+ : D_n \to D_{n+1}$ the injection and with $f_n^- : D_{n+1} \to D_n$ the embedding.

Let $\{C, g_n^-\}_{n\in\omega} = lim_{\mathbf{C}}\{D_n, f_n^-\}_{n\in\omega}$. We show that D_m can be made into a cone for $\{D_n, f_n^-\}_{n\in\omega}$, for all m. There is a natural way to go from D_m to D_n via the morphism $h_{m,n} : D_m \to D_n$ which is defined as follows:

$$h_{m,n} = \begin{cases} id & \text{if } m = n \\ f_n^- \circ \cdots \circ f_{m-1}^- & \text{if } m > n \\ f_{n-1}^+ \circ \cdots \circ f_m^+ & \text{if } m < n \ . \end{cases}$$

It is enough to check that $f_n^- \circ h_{m,n+1} = h_{m,n}$. Hence a unique cone morphism $g_m^+ : D_m \to C$ is determined such that $g_n^- \circ g_m^+ = h_{m,n}$, for all n. We note that $g_m^- \circ g_m^+ = id$, since $h_{m,m} = id$. And we observe:

$$g_m^+ \circ g_m^- = g_{m+1}^+ \circ f_m^+ \circ f_m^- \circ g_{m+1}^- \leq g_{m+1}^+ \circ g_{m+1}^- \ .$$

Hence $\{g_m^+ \circ g_m^-\}_{m\in\omega}$ is a chain and we write $k = \bigvee_{m\in\omega} g_m^+ \circ g_m^-$. We claim that (1) $k = id$, and (2) if $\{C, g_n\}_{n\in\omega}$ is a cocone of $\{D, f_n\}_{n\in\omega}$ in $\mathbf{C}^{ip}$ such that $\bigvee_{m\in\omega} g_m^+ \circ g_m^- = id$ then the cocone is a colimit.

(1) It is enough to remark that k is a cone endomorphism over $\{C, g_m^-\}_{m\in\omega}$ as:

$$\begin{aligned} g_m^- \circ k &= g_m^- \circ (\bigvee_{i\in\omega} g_i^+ \circ g_i^-) = g_m^- \circ (\bigvee_{i\geq m} g_i^+ \circ g_i^-) \\ &= \bigvee_{i\geq m} g_m^- \circ g_i^+ \circ g_i^- = \bigvee_{i\geq m} h_{i,m} \circ g_i^- = g_m^- \ . \end{aligned}$$

(2) Let $\{B, l_m\}_{m\in\omega}$ be another cocone. We define:

$$p^+ = \bigvee_{m\in\omega} l_m^+ \circ g_m^- : C \to B \quad p^- = \bigvee_{m\in\omega} g_m^+ \circ l_m^- : B \to C \ .$$

It is easy to check that $p : C \to B$ in $\mathbf{C}^{ip}$. Moreover p is a morphism of cocones between $\{C, g_m\}_{m\in\omega}$ and $\{B, l_m\}_{m\in\omega}$. Finally suppose q is another morphism with this property. Then:

$$\begin{aligned} (q^+, q^-) &= (q^+ \circ (\bigvee_{m\in\omega} g_m^+ \circ g_m^-), (\bigvee_{m\in\omega} g_m^+ \circ g_m^-) \circ q^-) \\ &= (\bigvee_{m\in\omega} q^+ \circ g_m^+ \circ g_m^-, \bigvee_{m\in\omega} g_m^+ \circ g_m^- \circ q^-) \\ &= (\bigvee_{m\in\omega} l_m^+ \circ g_m^-, \bigvee_{m\in\omega} g_m^+ \circ l_m^-) \\ &= (p^+, p^-) \ . \end{aligned}$$

□

We can extract from the previous proof the following useful information.

Proposition 7.1.16 *Let* $\mathbf{C}$ *be an O-category, and let* $\{D_n, f_n\}_{n\in\omega}$ *be an* ω*-chain in* $\mathbf{C}^{ip}$*, with a cocone* $\{C, g_n\}_{n\in\omega}$*. Then* $\{C, g_n\}_{n\in\omega}$ *is a colimit iff* $\bigvee_{n\in\omega} g_n^+ \circ g_n^- = id$.

We now show how to build ω^{op}-limits in the category **Cpo**.

Proposition 7.1.17 *The category* **Cpo** *has* ω^{op}*-limits.*

PROOF. Consider an ω^{op}-chain in **Cpo** $\{D_n, f_n\}_{n\in\omega}$ where $f_n : D_{n+1} \to D_n$. We define:

$$D = \{\alpha : \omega \to \bigcup_{n\in\omega} D_n \mid \alpha(n) \in D_n \text{ and } f_n(\alpha(n+1)) = \alpha(n)\}$$

with the pointwise ordering $\alpha \leq_D \beta$ iff $\forall n \in \omega\,(\alpha(n) \leq_{D_n} \beta(n))$. It is easy to verify that this makes D into a cpo. Now $\{D, g_n\}_{n\in\omega}$ is a cone with $g_n(\alpha) = \alpha(n)$. Suppose $\{E, h_n\}_{n\in\omega}$ is another cone. Then a continuous function $k : \{E, h_n\}_{n\in\omega} \to \{D, g_n\}_{n\in\omega}$ is completely determined by the equation $k(e)(n) = (g_n \circ k)(e) = h_n(e)$. □

Therefore, as an instance of theorem 7.1.15, we obtain:

$$colim_{\mathbf{Cpo}^{ip}}\{D_n, f_n\}_{n\in\omega} = lim_{\mathbf{Cpo}}\{D_n, f_n^-\}_{n\in\omega}\ .$$

This result is applied to bifinite domains in the following.

Proposition 7.1.18 *The category* $\mathbf{Bif}^{ip}$ *has* ω*-colimits.*

PROOF. Given an ω-chain in $\mathbf{Bif}^{ip}$ $\{D_n, f_n\}_{n\in\omega}$ let $\{D, g_n\}_{n\in\omega}$ be its ω-colimit in $\mathbf{Cpo}^{ip}$ which exists by proposition 7.1.17 and theorem 7.1.15. It remains to verify that D is bifinite. Since D_n is bifinite for any $n \in \omega$, we have $\bigvee_{i\in I_n} p_{n,i} = id$ where $\{p_{n,i}\}_{i\in I_n}$ is a directed set of finite projections over D_n. We compute:

$$\begin{aligned} id = \bigvee_{n\in\omega}(g_n^+ \circ g_n^-) &= \bigvee_{n\in\omega}(g_n^+ \circ (\bigvee_{i\in I_n} p_{n,i}) \circ g_n^-) \\ &= \bigvee_{n\in\omega}\bigvee_{i\in I_n}(g_n^+ \circ p_{n,i} \circ g_n^-) \\ &= \bigvee_{n\in\omega, i\in I_n}(g_n^+ \circ p_{n,i} \circ g_n^-)\ . \end{aligned}$$

We note that $g_n^+ \circ p_{n,i} \circ g_n^-$ is a finite projection and that the set $\{g_n^+ \circ p_{n,i} \circ g_n^-\}_{n\in\omega, i\in I_n}$ is directed by proposition 5.2.3. □

We now turn to functors. The following result relates local continuity and preservation of ω-colimits.

Proposition 7.1.19 *Let* $\mathbf{C}$ *be an O-category with* ω^{op}*-limits. If* $F : \mathbf{C}^{ip} \times \mathbf{C} \to \mathbf{C}$ *is a locally continuous functor then* $F^{ip} : \mathbf{C}^{ip} \times \mathbf{C}^{ip} \to \mathbf{C}^{ip}$ *preserves* ω*-colimits.*

PROOF. We have already observed that if F is locally monotonic then F^{ip} is a functor. Let $\{(D_n, E_n), (f_n, g_n)\}_{n\in\omega}$ be an ω-diagram in $\mathbf{C}^{ip} \times \mathbf{C}^{ip}$ with colimit $\{(D, E), (h_n, k_n)\}_{n\in\omega}$ built as in the previous theorem 7.1.15. To show that the

cocone $\{F^{ip}(D,E), F^{ip}(h_n,k_n)\}_{n\in\omega}$ is a colimit for $\{F^{ip}(D_n,E_n), F^{ip}(f_n,g_n)\}_{n\in\omega}$ it is enough to verify that (cf. proposition 7.1.16):

$$\bigvee_{n\in\omega} F(h_n^-,k_n^+)\circ F(h_n^+,k_n^-) = id_{F(D,E)} \ .$$

This is proven as follows:

$$\begin{array}{ll} \bigvee_{n\in\omega} F(h_n^-,k_n^+)\circ F(h_n^+,k_n^-) & = \bigvee_{n\in\omega} F(h_n^+\circ h_n^-, k_n^+\circ k_n^-) \\ & = F(\bigvee_{n\in\omega} h_n^+\circ h_n^-, \bigvee_{n\in\omega} k_n^+\circ k_n^-) \\ & = F(id_D,id_E) = id_{F(D,E)} \ . \end{array}$$

□

To summarize the method, we suppose given:

- An O-category $\mathbf{C}$ such that the hom-sets have a least element, composition is left strict, and $\mathbf{C}$ has (certain) ω^{op}-limits.
- A locally continuous functor $F : \mathbf{C}^{op}\times\mathbf{C}\to\mathbf{C}$.

We can apply the previous constructions and build:

- The category $\mathbf{C}^{ip}$ which has an initial object and ω-colimits.
- The functor $F : \mathbf{C}^{ip}\times\mathbf{C}^{ip}\to\mathbf{C}^{ip}$ which preserves ω-colimits.

Therefore we find an initial solution for $F^{ip}(X,X)\cong X$ in $\mathbf{C}^{ip}$. The initial solution also gives a solution for the equation $F(X,X)\cong X$ in $\mathbf{C}$.

Exercise 7.1.20 *Show the existence of a non-trivial domain D such that $D\cong D\times D\cong D\Rightarrow D$. Hint: consider the system $D\cong D\Rightarrow E$ and $E\cong E\times E$.*

Exercise 7.1.21 *Let $(\ )_\perp$ be the lifting functor (see definitions 1.4.16 and 8.1.5). Show that the equations $D\cong D\Rightarrow(D)_\perp$ and $D\cong(D)_\perp\Rightarrow(D)_\perp$ have a non-trivial initial solution in $\mathbf{Cpo}^{ip}$.*

Exercise 7.1.22 *Explain how to build two non-isomorphic, non-trivial solutions of the equation $D\cong D\Rightarrow D$. Hint: one can start the construction with a cpo which is not a lattice.*

Combining theorem 7.1.15 and proposition 7.1.3, we get the following so-called minimal invariant property, which gives a powerful tool for reasoning in recursively defined domains [Pit95].

Proposition 7.1.23 (minimal invariant) *Let $\mathbf{C}$ be an O-category with a terminal object and ω^{op}-limits, and such that each hom-set has a least element, and composition is left-strict. Let $F : \mathbf{C}^{op}\to\mathbf{C}$ be locally continuous. Let $i : F(C)\to C$ be an order-isomorphism constructed as indicated in the proof of proposition 7.1.3. We define $\delta : \mathbf{C}[C,C]\to\mathbf{C}[C,C]$ as follows:*

$$\delta(f) = i\circ F(f)\circ i^{-1}$$

Then i is a minimal invariant, by which is meant that the function δ is continuous and has id as least fixed point.

PROOF. The statement follows from proposition 7.1.16 and from the following claim:

$$\forall\, n \geq 0 \;\; \delta^n(\bot) = g_n^+ \circ g_n^-$$

where $\{C, g_n\}_{n\in\omega}$ is constructed as in the proof of theorem 7.1.15. The base case follows from left-strictness of composition. The induction case follows from the fact that (i, i^{-1}) is an iso from $\{F(C), h_n\}_{n\in\omega}$ to $\{C, g_{n+1}\}_{n\in\omega}$, with $h_n^+ = F(g_n^-)$ and $h_n^- = F(g_n^+)$. □

In the cpo case, we have seen that least fixed points are actually least prefixpoints (proposition 1.1.7). The following exercise gives a version of this for a contravariant functor [Pit95].

Exercise 7.1.24 *(1) Under the assumptions of proposition 7.1.23, suppose that* $f : A \to F(B)$ *and* $g : F(A) \to B$ *are given (think of the functor* $H : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{C}^{op} \times \mathbf{C}$ *defined by* $H(A, B) = (F(B), F(A))$*). Show that there exists a unique pair of morphisms* $h : A \to C$ *and* $k : C \to B$ *such that:*

$$F(k) \circ f = i^{-1} \circ h \quad \text{and} \quad g \circ G(h) = k \circ i$$

(2) Show that id is in fact the unique fixpoint of δ*. (3) Prove a version of (1) and (2) and of proposition 7.1.23 for a functor* $F : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{C}$*. Hints: For uniqueness, proceed as in the proof of theorem 7.1.15: take* $f = i^{-1}, g = i$*. Consider again, as a heuristics, an associated functor* $H' : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{C}^{op} \times \mathbf{C}$*.*

We have seen (almost) a minimal invariant at work in proposition 3.2.7. We shall use another one to prove an adequacy result in section 7.2.

We conclude this section with a version of Cantor's theorem on spaces of monotonic functions. Cantor's theorem states that there is no surjection from D to $\mathcal{P}(D)$ in the category of sets. It follows that the problem $(D \Rightarrow D) \lhd D$ has no non-trivial solution in this category (otherwise $\mathcal{P}(D) = (D \Rightarrow 2) \lhd (D \Rightarrow D) \lhd D$). This result can be generalized to the category of partially ordered sets and monotonic morphisms (cf. [GD62]) (a posteriori, this provides a justification for jumping directly from set-theoretic to continuous functions). In the following $\mathbf{O} = \{\bot, \top\}$ is the two points poset with $\bot < \top$, and $[D \to E]$ denotes the poset of monotonic morphisms, with D, E posets.

Proposition 7.1.25 *Let* P *be a poset. There is no monotonic surjection* $e : P \to [[P \to \mathbf{O}] \to \mathbf{O}]$*.*

PROOF. First we build a monotonic surjection $e_1 : [[P \to \mathbf{O}] \to \mathbf{O}] \to [P^{op} \to \mathbf{O}]$. To this end we define:

$$f : P^{op} \to [P \to \mathbf{O}] \qquad fxy = \bigvee\{\top \mid x \leq y\} \;.$$

We observe that f is monotonic and injective as:

$$fx \leq fz \text{ iff } \forall y\,(x \leq y \Rightarrow z \leq y) \text{ iff } z \leq x \ .$$

Next we define $e_1(F) = F \circ f$. We verify the surjectivity. Suppose $d : P^{op} \to \mathbf{O}$, let:

$$H_d : [P \to \mathbf{O}] \to \mathbf{O} \qquad H_d(h) = \bigvee\{dx \mid fx \leq h\} \ .$$

H_d is clearly monotonic. Surjectivity follows from the computation:

$$e_1(H_d)(z) = H_d(fz) = \bigvee\{dx \mid fx \leq fz\} = \bigvee\{dx \mid z \leq x\} = dz \ .$$

Suppose by contradiction that there is a monotonic surjection $e : P \to [[P \to \mathbf{O}] \to \mathbf{O}]$. By composition with e_1 we derive the existence of a surjection $s : P \to [P^{op} \to \mathbf{O}]$. We apply a diagonalization trick defining:

$$c, c' : P^{op} \to \mathbf{O} \quad c(x) = \overline{s(x)(x)} \quad c'(x) = \bigvee\{c(y) \mid x \leq y\}$$

where $\overline{\bot} = \top$ and $\overline{\top} = \bot$. Note that c' is monotonic. Let w be such that $c' = s(w)$. We claim that there is a $y \geq w$ such that $c(y) = \top$. If $c(w) = \top$ take w. Otherwise, if $c(w) = \bot$ then $s(w)(w) = \top$, that is $c'(w) = \top$. Hence $c(y) = \top$ for some $y \geq w$.

Suppose then $c(y) = \top$. We derive a contradiction as follows:

- $s(y)(y) = \bot$ by definition of c.
- $s(y)(y) = \top$ because:

$$\begin{array}{lll} c(y) = \top & \Rightarrow c'(y) = \top & \text{by definition of } c' \\ & \Rightarrow s(w)(y) = \top & \text{since } c' = s(w) \\ & \Rightarrow s(y)(y) = \top & \text{by left monotonicity of } s \text{ and } y \geq w \ . \end{array}$$

□

Corollary 7.1.26 *If $[P \to P] \triangleleft P$ in the category of posets and monotonic morphisms then $\sharp P = 1$.*

PROOF. Since the empty poset is not a solution suppose $\sharp P \geq 2$. If all elements in P are incomparable then Cantor's theorem applies. Otherwise, let $x_1 < x_2 \in P$. Then the pair $(i, \mathbf{O} \to P, j : P \to \mathbf{O})$ defined by:

$$i(y) = \begin{cases} x_2 & y = \top \\ x_1 & y = \bot \end{cases} \qquad j(x) = \begin{cases} \top & x_2 \leq x \\ \bot & \text{otherwise} \end{cases}$$

shows $\mathbf{O} \triangleleft P$. We observe that $D \triangleleft D'$ and $E \triangleleft E'$ implies $[D \to E] \triangleleft [D' \to E']$. By $[P \to \mathbf{O}] \triangleleft [P \to P] \triangleleft P$ we derive $[[P \to \mathbf{O}] \to \mathbf{O}] \triangleleft P$, contradicting the previous proposition 7.1.25. □

$$\begin{array}{lcl} n[\sigma] & \rightarrow & n \\ x[\sigma] & \rightarrow & \sigma(x)[\sigma] \\ (\text{let } x \text{ bedyn } M \text{ in } N)[\sigma] & \rightarrow & N[\sigma[M/x]] \end{array}$$

Figure 7.1: Operational semantics of DYN

7.2 Predicate Equations *

In proving properties of programs, one is often faced with predicates. We have seen their use in chapter 6, in particular for the proof of the adequacy theorem 6.3.6. If the semantics of a language involves recursively defined domains, then proving properties of programs may involve recursively defined predicates, and the existence of the solutions to these predicate equations may be troublesome, just as we had troubles with contravariance in solving domain equations. We treat an example of Mulmuley [Mul89], borrowing our techniques from Pitts [Pit95], to which we refer for a general treatment. Our example consists in proving an adequacy theorem for a simple declarative language DYN, based on dynamic binding, whose syntax is given by:

$$\begin{array}{lll} \mathit{Ide} & ::= x \mid y \mid \ldots \\ M & ::= n \mid \mathit{Ide} \mid \text{let } \mathit{Ide} \text{ bedyn } M \text{ in } M \ . \end{array}$$

where n ranges over natural numbers and where $x, y \ldots$ range over a set Ide of identifiers. The intended value of:

$$\text{let } x \text{ bedyn } 3 \text{ in let } y \text{ bedyn } x \text{ in let } x \text{ bedyn } 5 \text{ in } y$$

is 5, because in computing the value of y it is the last value of x, namely 5, which is used. In contrast, the λ-term $(\lambda x.(\lambda y.(\lambda x.y)5)x)3$ evaluates to 3. We say that λ-calculus is static. In the static discipline, the declaration of x which is used when evaluating y is the one which is immediately above y in the program text.

The operational semantics of DYN is described via rewriting rules on pairs (M, σ), written $M[\sigma]$, until eventually a constant n is reached. In the pairs $M[\sigma]$, M ranges over the set Exp of terms and σ ranges over the set of syntactic environments which are functions from Ide to Exp. The rules are given in figure 7.1. These rules should be contrasted with the rules for the environment machines described in section 8.3. In both cases a natural implementation relies on a stack to pile up unevaluated expressions. However, in dynamic binding we just save the code, whereas in static binding we memorise the code with its environment (a closure).

A denotational semantics of this language can be given with the help of a semantic domain D satisfying the equation $D = \mathit{Ide} \rightarrow (D \rightharpoonup \omega)$. The meaning $[\![M]\!]$ of a term M is as a partial function from D (ranged over by ρ) to ω, defined in figure 7.2 (without an explicit mention of the isomorphism $i : D = \mathit{Ide} \rightarrow (D \rightharpoonup \omega)$).

$$\begin{array}{lcll} [\![n]\!](\rho) & = & n & \\ [\![x]\!](\rho) & = & \rho(x)(\rho) & (\rho(x)(\rho)\downarrow) \\ [\![\text{let } x \text{ bedyn } M \text{ in } N]\!](\rho) & = & [\![N]\!]([\rho[[\![M]\!]/x]) & \end{array}$$

Figure 7.2: Denotational semantics of DYN

These semantic equations look "the same" as the rules defining the operational semantics. It requires however a non-trivial proof to show the following adequacy property of the denotational semantics with respect to the operational semantics:

$$\text{If } M \text{ is a closed term of DYN, then } M[] \to^* n \text{ iff } [\![M]\!](\bot) = n,$$

where [] is the identity syntactic environment and $\bot$ is the constant $\bot$ function.

We first need to formulate adequacy for any term. We define a semantic mapping from syntactic environments to semantic environments in the following way:

$$[\![\sigma]\!](x) = [\![\sigma(x)]\!] \ .$$

The general adequacy result that we want to prove is:

$$\text{For any } M \text{ and } \sigma,\ M[\sigma] \to^* n \text{ iff } [\![M]\!]([\![\sigma]\!]) = n \ .$$

($\Rightarrow$) We proceed by induction on the length of the derivation of $M[\sigma]$ to n:

- n: We have $[\![n]\!]([\![\sigma]\!]) = n$ by the first semantic equation.
- x:
$$\begin{array}{rcll} [\![x]\!]([\![\sigma]\!]) & = & [\![\sigma]\!](x)([\![\sigma]\!]) & \\ & = & [\![\sigma(x)]\!]([\![\sigma]\!]) & \\ & = & n & \text{by induction} \ . \end{array}$$
- let x bedyn M in N:
$$[\![\text{let } x \text{ bedyn } M \text{ in } N]\!]([\![\sigma]\!]) = [\![N]\!]([\![\sigma]\!][[\![M]\!]/x]) = [\![N]\!]([\![\sigma[M/x]]\!]) = n \ .$$

($\Leftarrow$) The proof involves a predicate $\Theta \subseteq (D \rightharpoonup \omega) \times Exp$ satisfying the following mutually recursive specification:

$$\begin{array}{rcl} \Theta & = & \{(f, M) \mid \forall\, (\rho, \sigma) \in \Pi \ \ (f(\rho)\uparrow \ \text{or} \ M[\sigma] \to^* f(\rho))\} \\ \Pi & = & \{(\rho, \sigma) \in D \times (Ide \to Exp) \mid \forall\, x \in Ide \ \ (\rho(x), \sigma(x)) \in \Theta\} \ . \end{array}$$

The whole point of this section is to prove that Θ exists. Meanwhile, assuming its existence, we end the proof of adequacy. We prove by induction on the size of M that $([\![M]\!], M) \in \Theta$.

- n: This case holds vacuously since we always have $[\![n]\!](\rho) = n$ and $n[\sigma] \to n$, regardless of what ρ and σ are.
- x: Let $(\rho, \sigma) \in \Pi$. In particular, $(\rho(x), \sigma(x)) \in \Theta$. By the specification of Θ, we have thus:
$$\rho(x)(\rho) \uparrow \text{ or } \sigma(x)[\sigma] \to^* \rho(x)(\rho)$$
which by the definition of the two semantics can be rephrased as:
$$[\![x]\!](\rho) \uparrow \text{ or } x[\sigma] \to^* [\![x]\!](\rho) \ .$$
- let x bedyn M in N: Let $(\rho, \sigma) \in \Pi$. First, exploiting induction on M, we get $([\![M]\!]\rho, M\sigma) \in \Pi$. The conclusion follows by applying induction to N.

We are now left with the proof of the existence of Θ. We set:

$$H(E) = Ide \to (E \rightharpoonup \omega) \qquad G(E) = (E \rightharpoonup \omega) \times Exp \ .$$

We have $H : \mathbf{Cpo}^{op} \to \mathbf{Cpo}$. The ordering on $H(E)$ is obtained as follows: $E \rightharpoonup \omega$ is isomorphic to $E \to \omega_\perp$ (cf. definitions 1.4.16 and 1.4.17), and given a domain D', $Ide \to D'$ is the product of copies of D' indexed over Ide, ordered pointwise. Remark that $\{\perp\} \rightharpoonup \omega$ (and hence $H(\{\perp\})$) has infinitely many elements, which makes the initial solution of $H(D) = D$ non-trivial.

We "extend" H to predicates as follows. For $R \subseteq G(E)$, we define $H(R) \subseteq G(H(E))$ as the set of pairs (f, M) such that:

$$\forall \rho \in H(E), \sigma \in Ide \to Exp \ \ (\forall x \ \ (\rho(x), \sigma(x)) \in R) \Rightarrow (f(\rho) \uparrow \text{ or } M[\sigma] \to^* f(\rho)) \ .$$

The predicate Θ is a fixpoint for the following function $K : \mathcal{P}(G(D)) \to \mathcal{P}(G(D))$:

$$K(R) = \{(f, M) \mid (f \circ i, M) \in H(R)\}$$

where $i : H(D) \to D$ is the minimal invariant (cf. proposition 7.1.23).

The trouble is that, because H is contravariant in E, the function K is anti-monotonic. The sequence $\{K^n(\emptyset)\}_{n<\omega}$ is a zigzag $\emptyset \subseteq K(\emptyset) \supseteq K^2(\emptyset) \cdots$ instead of being an increasing sequence, and therefore we cannot build a fixpoint for K right away. However, K gives rise to a continuous function:

$$L : ((\mathcal{P}(G(D)), \supseteq) \times (\mathcal{P}(G(D)), \subseteq)) \to ((\mathcal{P}(G(D)), \supseteq) \times (\mathcal{P}(G(D)), \subseteq))$$

defined by $L(S_1, S_2) = (K(S_2), K(S_1))$, which has a fixed point (R_1, R_2) (cf. exercise 7.1.24). For reasons linked with the particular K we have at hand, we in fact have $R_1 = R_2$, as we shall prove now. It is enough to establish $R_1 \subseteq R_2$, by the symmetric specification of R_1 and R_2.

We introduce more ingredients. Since H acts on relations as well as on objects and morphisms, we are led to examine the relationships between morphisms and relations more closely. Given $f : E \to E'$, $R \subseteq G(E)$ and $R' \subseteq G(E')$, we write:

$$f : R \to R' \Leftrightarrow \forall (g, M) \in R' \ \ (g \circ f, M) \in R \ .$$

The following are easily established facts:

$(R1)$ If $f : R \to R'$ and $f' : R' \to R''$, then $f' \circ f : R \to R''$.

$(R2)$ $id : R \to R'$ if and only if $R' \subseteq R$.

$(R3)$ If $f : R \to R'$, then $H(f) : H(R') \to H(R)$.

Moreover, we restrict our attention to predicates R satisfying the following properties.

$(I1)$ Closure under directed lub's: $(\forall\, \delta \in \Delta\ \ (\delta, M) \in R) \Rightarrow (\bigvee \Delta, M) \in R$.

$(I2)$ $\forall M\ \ (\bot, M) \in R$.

$(I3)$ $\forall\, n \in \omega\ \ ((\lambda\rho.n, M) \in R \Leftrightarrow M[] \to^* n)$.

We denote with $I(E)$ (I for "inclusive", cf. section 6.2) the collection of predicates over $G(E)$ satisfying properties (I1) through (I3). Clearly, $I(E)$ has a bottom element, which is:

$$\{(\lambda\rho.n, M) \mid M[] \to^* n\} \cup \{(\bot, M) \mid M \in Exp\}\ .$$

Moreover, H is compatible with (I1) through (I3).

$(R4)$ H maps $I(E)$ to $I(H(E))$.

We only check that $H(R)$ satisfies (I3). If $(\lambda\rho.n, M) \in H(R)$, then since $(\bot, x) \in R$ for all x, we have $M[] \to^* n$. The converse direction follows from the fact that $M[] \to^* n$ implies $M[\sigma] \to^* n$ for any σ.

From now on, we shall assume that all predicates satisfy (I1) through (I3). The following further facts will be needed.

$(R5)$ For any directed $\Delta \subseteq (E \to E')$, $(\forall\, \delta \in \Delta\ \ \delta : R \to R') \Rightarrow \bigvee \Delta : R \to R'$.

$(R6)$ $\bot : R \to R'$, for any R, R'.

Fact (R5) is a consequence of (I1), by the continuity of the composition operation. Properties (I2) and (I3) serve to establish (R6), as we show now. Let $(d, M) \in R'$. There are two cases. If d is strict, then $d \circ \bot = \bot$, and $(d \circ \bot, M) \in R$ follows by (I2). If $d = \lambda\rho.n$, the conclusion follows by (I3). We now have all the needed material.

By property $(R4)$, the function K restricts to a function from $I(D)$ to $I(D)$. Hence we can take the solution (R_1, R_2) in $(I(D), \supseteq) \times (I(D), \subseteq)$. By (R2), by the minimal invariant property of i, and by (R5), our goal can be reformulated as:

$$\forall\, n \geq 0\ \ \delta^n(\bot) : R_2 \in R_1\ .$$

The base case of our goal holds by (R6). By fixpoint induction (cf. section 6.2), we are left to show:

$$f : R_2 \to R_1 \Rightarrow \delta(f) : R_2 \to R_1$$

which by (R1) is proved as follows:

$$\begin{array}{ll} i^{-1} : R_2 \to H(R_1) & (K(R_1) = R_2) \\ H(f) : H(R_1) \to H(R_2) & (\text{by } (R3)) \\ i : H(R_2) \to R_1 & (K(R_2) = R_1)\ . \end{array}$$

Remark 7.2.1 *Our proof uses only the fact that (R_1, R_2) is a fixpoint (in $I(D)$), not that this is the least one. So, in the end, we get not only that Θ exists, but also that it is the "unique" solution of $K(R) = R$ (cf. exercise 7.1.24).*

7.3 Universal Domains

We discuss a technique for the construction of a *universal* domain and we apply it to the category of bifinite domains and continuous morphisms. In this section by bifinite domain we intend the ω-bifinite (or SFP domains) described in chapter 5. In the first place, we introduce the notion of *algebroidal* category (cf. [BH76]) which generalizes to categories the notion of algebraicity already considered for domains.

Definition 7.3.1 (category of monos) *A category* $\mathbf{K}$ *is called a category of monos if every morphism of* $\mathbf{K}$ *is mono.*

Example 7.3.2 *Sets with injections form a category of monos.*

Definition 7.3.3 (compact object) *Let* $\mathbf{K}$ *be a category of monos. An object* $A \in \mathbf{K}$ *is* compact *if, for each* ω*-chain* $\{B_n, f_n\}_{n\in\omega}$ *with colimit* $\{B, g_n\}_{n\in\omega}$ *and any* $h : A \to B$*, there exists* n *and* $k_n : A \to B_n$ *such that* $h = g_n \circ k_n$*. We denote with* $\mathcal{K}(\mathbf{K})$ *the collection of compact objects.*

Remark 7.3.4 *We note that for any* n *there is at most one* k_n*, as* $\mathbf{K}$ *is a category of monos and therefore* $g_n \circ k_n = g_n \circ k'_n = h$ *implies* $k_n = k'_n$*. Moreover, if* $g_n \circ k_n = h$ *then we can set* $k_{n+1} = f_n \circ k_n$ *as* $g_{n+1} \circ f_n \circ k_n = g_n \circ k_n = h$*.*

Definition 7.3.5 (algebroidal category) *A category of monos* $\mathbf{K}$ *is* algebroidal *if:*

(1) $\mathbf{K}$ *has an initial object.*

(2) *Every* ω*-chain of compact objects has a colimit.*

(3) *Every object is the colimit of an* ω*-chain of compact objects.*

An algebroidal category is ω*-algebroidal if the collection of compact objects,* $\mathcal{K}(\mathbf{K})$*, is countable up to isomorphism and so is the hom-set between any two compact objects.*

Remark 7.3.6 *The categories of* ω*-algebraic dcpo's considered in this book are* ω*-algebroidal. The category of ordinals is a notable example of non* ω*-algebroidal category (non-limit ordinals cannot be enumerated up to isomorphism).*

Exercise 7.3.7 *Let* $\mathbf{S}$ *be the category of Scott domains (see definition 1.4.9). Show that* $\mathbf{S}^{ip}$ *is not an algebroidal category. How would you modify the definition in order to include* $\mathbf{S}^{ip}$ *among the algebroidal categories? Hint: A directed diagram in a category* $\mathbf{C}$ *is a functor* $D : I \to \mathbf{C}$*, where* I *is a directed set. Show that: (1)* $\mathbf{S}^{ip}$ *has colimits of directed diagrams. (2) Each object is the colimit of a directed diagram of compact objects.*

Next we define the notion of universal object. In particular we will be interested in universal, *homogeneous* objects, as they are determined up to isomorphism. In this section we follow quite closely [DR93] (see also [GJ90]). More generally, the terminology and the techniques used in this section are clearly indebted to model theory.

Definition 7.3.8 *Let U be an object in a category $\mathbf{K}$ of monos, and let $\mathbf{K}^*$ be a full subcategory of $\mathbf{K}$. Then we say that:*

(1) *U is* $\mathbf{K}^*$-universal *if* $\forall A \in \mathbf{K}^* \, \exists f : A \to U$.

(2) *U is* $\mathbf{K}^*$-homogeneous *if:*

$$\forall A \in \mathbf{K}^* \, \forall f : A \to U \, \forall g : A \to U \, \exists h : U \to U \, (h \circ g = f) \ .$$

(3) *U is* $\mathbf{K}^*$-saturated *if:*

$$\forall A, B \in \mathbf{K}^* \, \forall f : A \to U \, \forall g : A \to B \, \exists k : B \to U \, (k \circ g = f) \ .$$

(4) $\mathbf{K}^*$ *has the* amalgamation property *if:*

$$\begin{array}{l} \forall A, B, B' \in \mathbf{K}^* \, \forall f : A \to B \, \forall f' : A \to B' \\ \exists C \in \mathbf{K}^* \, \exists g : B \to C \, \exists g' : B' \to C \, (g \circ f = g' \circ f') \ . \end{array}$$

Remark 7.3.9 *Definition 7.3.8 requires the existence of certain morphisms but* not *their uniqueness.*

Proposition 7.3.10 *Let $\mathbf{Bif}^{ip}$ be the category of ω-bifinite domains and injection-projection pairs. $\mathbf{Bif}^{ip}$ is an ω-algebroidal category and the collection of compact objects has the amalgamation property.*

PROOF. To check that $\mathbf{Bif}^{ip}$ is a category of monos with initial object it is enough to verify that $\mathbf{Bif}$ has a terminal object, the hom-sets have a least element and composition is left strict (cf. proposition 7.1.10).

Let $D \in \mathbf{Bif}$. By proposition 5.2.3, D is compact in $\mathbf{Bif}^{ip}$ iff the cardinality of D is finite. Moreover, by definition of bifinite domain, each object in $\mathbf{Bif}^{ip}$ is an ω-colimit of compact objects. By proposition 7.1.18, each ω-diagram of (compact) objects in $\mathbf{Bif}^{ip}$ has a colimit.

Next we verify that $\mathbf{Bif}^{ip}$ has the amalgamation property. Let us consider three finite posets $(E, \leq)$, $(D_1, \leq_1)$, $(D_2, \leq_2)$ with morphisms $h_i : E \to D_i$, $i = 1, 2$, in $\mathbf{Bif}^{ip}$. Without loss of generality we assume $E = D_1 \cap D_2$, then:

$$\forall e, e' \in E \, (e \leq e' \ \text{ iff } \ e \leq_1 e' \ \text{ iff } \ e \leq_2 e') \ .$$

Now we define the amalgam as the set $F = E \cup (D_1 \backslash E) \cup (D_2 \backslash E)$ where $f \leq_F f'$ iff

$$\begin{array}{l} \exists i \in \{1, 2\} \, (f, f' \in D_i, \ f \leq_i f') \text{ or} \\ \exists e \in E \, (f \leq_i e \leq_j f') \text{ for } i \neq j, i, j \in \{1, 2\} \ . \end{array}$$

It is easy to verify that $\leq_F$ is a partial order. We are left with the definition of the morphisms $k_i : D_i \to F$, $i = 1, 2$. We take the inclusions for k_i^+, and we define:

$$k_1^-(f) = \begin{cases} f & f \in D_1 \\ h_2^-(f) & \text{otherwise} . \end{cases}$$

k_2^- is defined symmetrically. It can be easily checked that k_i is a morphism in $\mathbf{Bif}^{ip}$, and that $k_1 \circ h_1 = k_2 \circ h_2$. □

Theorem 7.3.11 *Let* $\mathbf{K}$ *be an* ω*-algebroidal category of monos. The following properties are equivalent:*

(1) *There is a* $\mathbf{K}$*-universal,* $\mathcal{K}(\mathbf{K})$*-homogeneous object (universal homogeneous for short).*

(2) *There is a* $\mathcal{K}(\mathbf{K})$*-saturated object.*

(3) $\mathcal{K}(\mathbf{K})$ *has the amalgamation property.*

Moreover a $\mathbf{K}$*-universal,* $\mathcal{K}(\mathbf{K})$*-homogeneous object is uniquely determined up to isomorphism.*

PROOF. The proof of this theorem is an immediate consequence of the following lemmas. The main difficulty lies in the proof of (3) ⇒ (2) (see lemma 7.3.14). □

Lemma 7.3.12 *Let* $\mathbf{K}$ *be an algebroidal category of monos and let* U, V *be* $\mathcal{K}(\mathbf{K})$*-saturated. Then:*

$$\forall A \in \mathcal{K}(\mathbf{K})\, \forall f : A \to U\, \forall g : A \to V\, \exists i : U \to V \text{ iso } (i \circ f = g) .$$

PROOF. Let $\{(U_i, f_i)\}_{i\in\omega}$ and $\{(V_j, g_j)\}_{j\in\omega}$ be ω-diagrams of compact objects whose colimits are $\{U, l_i\}_{i\in\omega}$ and $\{V, l_i'\}_{i\in\omega}$ respectively. Given $f : A \to U$ and $g : A \to V$, by compactness of A we have

$$\begin{array}{ll} \exists f_{n_0}^* : A \to U_{n_0}\, (l_{n_0} \circ f_{n_0}^* = f) & \text{(by compactness of } A) \\ \exists p_{n_0} : U_{n_0} \to V\, (p_{n_0} \circ f_{n_0}^* = g) & \text{(by saturation)} \\ \exists h_0^* : U_{n_0} \to V_{n_1}\, (l_{n_1}' \circ h_0^* = p_{n_0}) & \text{(by compactness)} . \end{array}$$

We show how to iterate this construction once more. By saturation $\exists p_{n_1} : V_{n_1} \to U\, (p_{n_1} \circ h_0^* = l_{n_0})$. By compactness $\exists h_1^* : V_{n_1} \to U_{n_2}\, (l_{n_2} \circ h_1^* = p_{n_1})$. We proceed inductively building $V_{n_3}, U_{n_4}, \ldots$ We may suppose $n_0 < n_1 < \ldots$ We observe $l_{n_2} \circ h_1^* \circ h_0^* = p_{n_1} \circ h_0^* = l_{n_0}$. It is then possible, using the h_i^*, to see V as (the object of) a cocone for $\{(U_i, f_i)\}_{i\in\omega}$ and U as (the object of) a cocone for $\{(V_j, g_j)\}_{j\in\omega}$, by which the existence of the isomorphisms h and k which commute with f and g follows. □

Lemma 7.3.13 *Let* $\mathbf{K}$ *be an algebroidal category of monos. The following properties hold:*

(1) *For any object U the following are equivalent: (a) U is $\mathbf{K}$-universal and $\mathcal{K}(\mathbf{K})$-homogeneous. (b) U is $\mathcal{K}(\mathbf{K})$-universal and $\mathcal{K}(\mathbf{K})$-homogeneous. (c) U is $\mathcal{K}(\mathbf{K})$-saturated.*

(2) *A $\mathbf{K}$-universal, $\mathcal{K}(\mathbf{K})$-homogeneous object is determined up to isomorphism.*

(3) *If there is a $\mathbf{K}$-universal and $\mathcal{K}(\mathbf{K})$-homogeneous object then $\mathcal{K}(\mathbf{K})$ has the amalgamation property.*

PROOF. (1) We prove the equivalence as follows:

$(a) \Rightarrow (b)$ Immediate, by definition.

$(b) \Rightarrow (c)$ Let $A, B \in \mathcal{K}(\mathbf{K})$, $f : A \to U$, $g : A \to B$. By $\mathcal{K}(\mathbf{K})$-universality $\exists g' : B \to U$. By $\mathcal{K}(\mathbf{K})$-homogeneity $\exists h : U \to U(h \circ g' \circ g = f)$. So $h \circ g'$ gives saturation.

$(c) \Rightarrow (a)$ Since there is an initial object 0, U is $\mathcal{K}(\mathbf{K})$-universal by saturation applied to the (unique) morphisms $f : 0 \to U$ and $g : 0 \to A$. U is also $\mathcal{K}(\mathbf{K})$-homogeneous by lemma 7.3.12. It remains to show that U is $\mathbf{K}$-universal. Let $A \in \mathbf{K}$ and let $\{(A_i, f_i)\}_{i \in \omega}$ be an ω-chain in $\mathcal{K}(\mathbf{K})$ whose colimit is A. Take advantage of $\mathcal{K}(\mathbf{K})$-saturation to build a cocone with object U for such ω-chain. Then there is a morphism from A to U.

(2) Apply lemma 7.3.12 with $A = 0$.

(3) Let $A, B, B' \in \mathcal{K}(\mathbf{K})$, $f : A \to B$, $f' : A \to B'$. By $\mathcal{K}(\mathbf{K})$-universality $\exists h : B \to U$. By $\mathcal{K}(\mathbf{K})$-saturation $\exists h' : B' \to U(h \circ f = h' \circ f')$. Now consider an ω-chain in $\mathcal{K}(\mathbf{K})$ whose colimit is U and use the compactness of B and B' to factorize h and h' along some element of the ω-chain. □

In the next lemma we use (for the first time) the countability conditions that distinguish an ω-algebroidal category from an algebroidal one.

Lemma 7.3.14 *Let $\mathbf{K}$ be an ω-algebroidal category of monos. If $\mathcal{K}(\mathbf{K})$ has the amalgamation property then it is possible to build a $\mathcal{K}(\mathbf{K})$-saturated onbject.*

PROOF. We use the hypothesis that $\mathbf{K}$ is ω-algebroidal to build an enumeration up to isomorphism of the compact objects $H_o = \{A_i\}_{i \in \omega}$ and an enumeration of all quintuples $M_o = \{(B_i, C_i, g_i, h_i, j_i)\}_{i \in \omega}$, where $B_i, C_i \in H_o$, $g_i, h_i : B_i \to C_i$, and $j_i \in \omega$, such that each quintuple occurs infinitely often. We build an ω-chain $\{(U_i, f_i)\}_{i \in \omega}$ such that $U_i \in H_o$ and the following properties hold, where we set $f_{j,i} : U_j \to U_i$, and $f_{j,i} = f_{i-1} \circ \cdots \circ f_j$:

(1) $\forall i \in \omega \, \exists k_i : A_i \to U_i$.

(2) Given i consider the corresponding quintuple in the enumeration. If $j = j_i \leq i$ and $U_j = C_i$ then

$$\exists k : U_i \to U_{i+1}(k \circ f_{j,i} \circ h_i = f_i \circ f_{j,i} \circ g_i) \ .$$

A consequence of (1) is that, for all $C \in H_o$ and j sufficiently large, we can find $g : C \to U_j$. We also note that if $g, h : B \to C$ with $B, C \in H_o$, and $C = U_j$, then (B, C, g, h, j) will appear infinitely often in the enumeration, so we can find an i such that $(B, C, g, h, j) = (B_i, C_i, g_i, h_i, j_i)$ and $(j =)j_i \leq i$.

Then we define U as the colimit of the ω-chain $\{(U_n, f_n)\}_{n\in\omega}$. While condition (1) is natural, condition (2) may seem rather obscure. First observe that if we just want to build a $\mathcal{K}(\mathbf{K})$-universal object, that is satisfy condition (1), then it is enough to set $U_0 = A_0$ and proceed inductively using the amalgamation property on the (uniquely determined) morphisms $f : 0 \to U_n$ and $g : 0 \to A_{n+1}$. So, given lemma 7.3.13, condition (2) has to do with the fact that we want U to be $\mathcal{K}(\mathbf{K})$-saturated. Let us see how this is used. Let $B, C \in H_o$ and $g : B \to C, h : B \to U$. By (1) and $B \in \mathcal{K}(\mathbf{K})$ we have:

$$\exists j \, (g' : C \to U_j, h^* : B \to U_j, h = f_{j,\infty} \circ h^*) \; .$$

where $f_{j,\infty} : U_j \to U$. Let $g^* = g \circ g'$. Choose i large enough so that:

$$j \leq i \text{ and } (B, U_j, g^*, h^*, j) = (B_i, C_i, g_i, h_i, j_i) \; .$$

By (2), $\exists k : U_i \to U_{i+1} \, (k \circ f_{j,i} \circ h^* = f_i \circ f_{j,i} \circ g^*)$. From this, saturation follows.

Finally we show how to build the ω-chain $\{(U_i, f_i)\}_{i\in\omega}$. Set $U_0 = A_0$, the first element in the enumeration H_o. Next suppose to have built U_i and consider A_{i+1}. As observed above there are $f : 0 \to U_i$ and $g : 0 \to A_{i+1}$. By amalgamation we get, for some U'_i, two morphisms $f' : U_i \to U'_i$ and $g' : A_{i+1} \to U'_i$.

- If $j = j_i \leq i$ and $U_j = C_i$ then apply amalgamation to $f' \circ f_{j,i} \circ h_i$ and $f' \circ f_{j,i} \circ g_i$ obtaining $k : U'_i \to U'_{i+1}$ and $k' : U'_i \to U'_{i+1}$. It just remains to select U_{i+1} isomorphic to U'_{i+1} and in H_o.
- Otherwise it is enough to choose an object U_{i+1} in H_o isomorphic to U'_i. The morphism from A_{i+1} to U_{i+1} is then immediately obtained by composition. □

Corollary 7.3.15 *The category* $\mathbf{Bif}^{ip}$ *has a universal homogeneous object.*

PROOF. We have shown in proposition 7.3.10 that $\mathbf{Bif}^{ip}$ is an ω-algebroidal category with the amalgamation property. Hence theorem 7.3.11 can be applied. □

Figure 7.3 draws a rough correspondence between domain theoretical and category theoretical notions.

7.4 Representation

We are interested in the problem of representing *subdomains* of a domain D as certain functions over D. In particular we concentrate on retractions and

in a cpo D	in $\mathbf{Cpo}^{ip}$	in $\mathbf{Cpo}$
$\perp$	initial object	terminal object
directed lub	ω-colimits	ω^{op}-limits
monotonic	functor F^{ip}	functor F (not always)
continuous	ω-cocontinuous	locally continuous
algebraic	algebroidal	

Figure 7.3: Domain-theoretical versus category-theoretical notions

projections, the idea being that subdomains are represented by the image of such morphisms. When working with continuous cpo's, not every retraction (or projection) corresponds to a domain (i.e. an ω-algebraic cpo). For this reason, one focuses on the collection of *finitary* retractions, which are by definition those retractions whose image forms a domain.

The theory is simpler when dealing with (finitary) projections. Then it is not difficult to show that the collection of finitary projections $FP(D)$ over a bifinite domain D is again a bifinite. In other words the collection of subdomains of a bifinite domain can be given again a bifinite domain structure. Having found a representation of domains, we address the problem of representing domain constructors, e.g. product, exponent, sum, lifting. It turns out that the basic domain constructors we have considered so far can be represented in a suitable technical sense.

The collection $Ret(D)$ of retractions on a cpo D, is the collection of fixpoints of the functional $\lambda f.f \circ f$, and the image $r(D)$ of a retraction r on D, coincides with the collection of its fixpoints. Hence general results on fixpoints can be immediately applied. We will see that under suitable hypotheses $Ret(D)$ and $r(D)$ enjoy certain algebraic properties.

Definition 7.4.1 *Let D be an algebraic cpo and $r \in Ret(D)$. We say that r is* finitary *if $r(D)$ with the induced order is an algebraic cpo. We say that r is a* closure *if $id \leq r$.*

Proposition 7.4.2 *Let D be a cpo. Then:*

(1) *If $f : D \to D$ is a continuous morphism then $Fix(f) = \{d \in D \mid f(d) = d\}$ is a cpo.*

(2) *$Ret(D) = Fix(\lambda f : D \to D.f \circ f)$ is a cpo.*

(3) *If $r \in Ret(D)$ then $r(D) = Fix(r)$ is a cpo.*

Proposition 7.4.3 *Let D be an algebraic cpo and $r \in Ret(D)$. Then $\mathcal{K}(r(D))$, the collection of compacts in $r(D)$, can be characterized as follows:*

$$\mathcal{K}(r(D)) = \{rd \mid d \in \mathcal{K}(D) \text{ and } d \leq rd\} \ .$$

In particular, if p is a projection then $\mathcal{K}(p(D)) = p(D) \cap \mathcal{K}(D)$, and if c is a closure then $\mathcal{K}(c(D)) = c(\mathcal{K}(D))$.

PROOF. Suppose $ry \in \mathcal{K}(r(D))$. Since D is algebraic $ry = \bigvee\{x \in \mathcal{K}(D) \mid x \leq ry\}$. So:

$$ry = r(ry) = r(\bigvee\{x \in \mathcal{K}(D) \mid x \leq ry\}) = \bigvee\{rx \mid x \in \mathcal{K}(D) \text{ and } x \leq ry\} \ .$$

Since $ry \in \mathcal{K}(r(D))$, we have $\exists z\,(ry = rz$ and $z \in \mathcal{K}(D)$ and $z \leq ry)$. This z gives the desired representation of ry. Vice versa suppose $d \in \mathcal{K}(D)$ and $d \leq rd$. Let $\Delta \subseteq r(D)$ directed. Then:

$$d \leq rd \leq \bigvee \Delta \quad \Rightarrow \quad \exists y \in \Delta\,(rd \leq ry = y) \ .$$

The statements concerning projections and closures are an immediate corollary of this characterization of the compact elements. □

Proposition 7.4.4 *If D is a bounded complete cpo and $r \in Ret(D)$ then $r(D)$ is bounded complete.*

PROOF. Let $X \subseteq r(D)$ and suppose $y \in r(D)$ is an upper bound for X. Then X is bounded in D and therefore $\bigvee_D X$ exists. We show $\bigvee_{r(D)} X = r(\bigvee_D X)$.

- $\forall x \in X\,(x \leq \bigvee_D X)$ implies $\forall x \in X\,(x = rx \leq r(\bigvee_D X))$. So $r(\bigvee_D X)$ is an upper bound.
- If y is an upper bound for X in $r(D)$ then it is also an upper bound for X in D, so $\bigvee_D X \leq y$. This implies $r(\bigvee_D X) \leq ry = y$. So $r(\bigvee_D X)$ is the lub in $r(D)$. □

Let D be an (ω-)algebraic cpo and $r \in Ret(D)$. Can we conclude that $r(D)$ is again an (ω-)algebraic cpo? The answer is *no*. In general it can only be shown that $r(D)$ is a continuous cpo (see chapter 5).

Example 7.4.5 *Let $\mathcal{Q}$ and $\mathcal{R}$ be the rational and real numbers, respectively. Let $D = D_0 \cup D_1$ where $D_0 = \{[0,q] \mid q \in \mathcal{Q}\}$, and $D_1 = \{[0,r[\mid r \in \mathcal{R} \cup \{\infty\}\}$, ordered by inclusion. Consider the projection p defined by:*

$$p([0,q]) = [0,q[\qquad p([0,r[) = [0,r[\ .$$

The domain D is an ω-algebraic complete total order with D_0 as compact elements. On the other hand $im(p)$ fails to be algebraic.

For the collection $Ret(D)$ things get even worse. For example it has been shown by Ershov (see exercise 18.4.10 in [Bar84]) that the collection of retractions over $\mathcal{P}(\omega)$ is not a continuous lattice, hence a fortiori not the image of a retraction. This also shows that the collection of fixpoints of a continuous function does not need to be a continuous cpo, as $Ret(D) = Fix(\lambda f.f \circ f)$.

We will consider retractions again in the context of stable domain theory (section 12.4). For the time being we will concentrate on the simpler case of *finitary projections*. Let D be a bifinite domain. The notion of finitary projection over D provides an adequate representation of the idea of subdomain, moreover the collection of finitary projections over D, $FP(D)$, is again a bifinite domain. This is a powerful result that has applications for instance to the interpretation of higher-order calculi (see section 11.3). The following notion of normal subposet is useful in studying projections.

Definition 7.4.6 (normal subposet) *Let $(P, \leq)$ be a poset. A subset $N \subseteq P$ is called a* normal subposet *if $\forall x \in P\,(\downarrow x) \cap N$ is directed. We denote with $N(P)$ the collection of normal subposets of P ordered by inclusion.*

Theorem 7.4.7 *Let $D \in$ **Bif**. Then:*

(1) *There is an isomorphism between the collection of normal subposets of the compact elements and the finitary projections over D: $N(\mathcal{K}(D)) \cong FP(D)$.*

(2) *$FP(D)$ is an ω-algebraic complete lattice.*

PROOF. We remark that if p is a projection and $x \in D$ then

$$(\downarrow x) \cap p(D) = (\downarrow p(x)) \cap p(D)\ .$$

Moreover, if p is a finitary projection then $\mathcal{K}(p(D)) \in N(\mathcal{K}(D))$. We use the hypothesis that $p(D)$ is algebraic to show $\forall x \in D\,(\downarrow x) \cap \mathcal{K}(p(D)) = (\downarrow p(x)) \cap \mathcal{K}(p(D))$ that is directed.

We now proceed with the proof of statement (1) while leaving (2) as an exercise. If p is a finitary projection then define $N_p = \mathcal{K}(p(D))$. This is a normal subposet of $\mathcal{K}(D)$ by the remark above. Vice versa, if $N \in N(\mathcal{K}(D))$ we define:

$$p_N(d) = \bigvee((\downarrow d) \cap N)\ .$$

This is well defined because $(\downarrow d) \cap N$ is directed. These two functions define an isomorphism between $FP(D)$ and $N(\mathcal{K}(D))$. □

Exercise 7.4.8 *Show that if D is a Scott domain then $FP(D) \trianglelefteq (D \to D)$. Hint: given $f : D \to D$ consider $X_f = \{x \in \mathcal{K}(D) \mid x \leq fx\}$ and define $N_f = U^*(X_f)$. The set N_f corresponds to a finitary projection p_{N_f}. Set $\pi : (D \to D) \to (D \to D)$ as $\pi(f) = p_{N_f}$. Note that this property fails for bifinite domains (cf. exercise 12.4.21).*

Let U be a universal domain for some relevant category of domains, say **Bif**. Then every domain is isomorphic to the image of a finitary projection over U. Furthermore, it can be shown that certain basic operators over domains can be adequately represented as continuous functions over $FP(U)$. As a fall-out, one gets a technique to solve domain equations via the standard Kleene least-fixed point theorem.

Observe that there is an injective function Im from the poset $FP(U)$ to the category $\mathbf{Bif}^{ip}$ (this is immediately extended to a functor):

$$Im = \lambda p \in FP(U).p(U) \ .$$

Let $F : \mathbf{Bif}^{ip} \times \mathbf{Bif}^{ip} \to \mathbf{Bif}^{ip}$ be a binary functor. The representation problem for F consists in finding a continuous function $R_F : FP(U) \times FP(U) \to FP(U)$ such that the following holds, modulo order-isomorphism:

$$F(p(U), q(U)) = R_F(p, q)(U) \ .$$

Proposition 7.4.9 *Product and Exponent are representable.*

PROOF. In showing that **Bif** is a CCC (proposition 5.2.4), one uses the fact that if $p \in FP(D)$ and $q \in FP(D)$ then $\lambda(d, e).(p(d), q(e)) \in FP(D \times E)$ and $\lambda f.(q \circ f \circ p) \in FP(D \to E)$. If U is a universal (homogeneous) domain for $\mathbf{Bif}^{ip}$ then we may assume the existence of the injection-projection pairs:

$$(\lambda(u, u').\langle u, u'\rangle, \lambda u.(\pi_1(u), \pi_2(u))) : (U \times U) \to U \quad (i, j) : (U \to U) \to U \ .$$

It just remains to combine the two ideas to define the operators representing product and exponential:

$$\lambda(p, q).\lambda u.\langle p(\pi_1(u)), q(\pi_2(u))\rangle \quad \lambda(p, q).\lambda u.i(q \circ j(u) \circ p) \ .$$

For instance, in the case of the exponential, we compose $\lambda(p, q).\lambda u.(q \circ u \circ p) : FP_U \times FP_U \to FP_{U \to U}$ with $\lambda r.i \circ r \circ j$. □

Remark 7.4.10 *It is good to keep in mind that Im is not an equivalence of categories between $FP(U)$ and $\mathbf{Bif}^{ip}$, as $FP(U)$ is just a poset category. This point is important when one interprets second order types (see section 11.3).*

Exercise 7.4.11 *(1) Verify in detail that we can apply the fixed point proposition 1.1.7 to the domain $FP(U)$ in order to get initial solutions of domain equations in $\mathbf{Bif}^{ip}$. (2) Consider the representation problem for the operators of coalesced sum and lifting. (3) Consider the representation problem in the case we replace (finitary) projections with (finitary) retractions.*

Finally, we point out that our results about the limit-colimit coincidence (theorem 7.1.15) and the existence of a universal homogeneous object (theorem 7.3.11) can be also applied to categories of cpo's and stable functions (cf. exercise 12.4.23).

Chapter 8

Values and Computations

When considering the λ-calculus as the kernel of a programming language it is natural to concentrate on *weak* reduction strategies, that is strategies where evaluation stops at λ-abstractions. In presenting the semantic counterpart of these calculi it is useful to emphasize the distinction between *value* and *computation*. A first example coming from recursion theory relies on the notions of total and partial morphism. In our jargon a total morphism when given a value always returns a value whereas a partial morphism when given a value returns a possibly infinite computation. This example suggests that the denotation of a partial recursive algorithm is a morphism from values to computations, and that values are particular kinds of computations.

In domain theory the divergent computation is represented by a bottom element, say $\perp$, that we add to the collection of values. This can be seen as the motivation for the shift from sets to flat domains. More precisely, we have considered three categories (cf. definition 1.4.17).

- The category **Dcpo** in which morphisms send values to values, say $D \to E$. This category is adapted to a framework where every computation terminates.
- The category **pDcpo** which is equivalent to the one of cpo's and strict functions, and in which morphisms send values to computations, say $D \to (E)_\perp$. This category naturally models *call-by-value* evaluation where functions' arguments are evaluated before application.
- The category **Cpo** in which morphisms send computations to computations, or $(D)_\perp \to (E)_\perp$. In the models of the untyped λ-calculus that we have presented the distinction value-computation can actually be hidden by regarding $\perp$ as an element with the same status of a value.

Another framework where the distinction between values and computations is useful is that of fixpoint extensions of typed λ-calculi. Consider for example a simply typed λ-calculus and its *Curry-Howard correspondence* with the minimal propositional logic of implication (cf. chapter 4). Suppose that we want to enrich the calculus with a fixed point combinator on terms, say Y, allowing for

fully recursive definitions. Which type should we assign to Y? One possibility considered in chapter 6 is to introduce a family of combinators Y^σ of type $(\sigma \to \sigma) \to \sigma$. Then the correspondence with the logic is blurred as $Y^\sigma(\lambda x : \sigma.x)$ has type σ for any type/proposition σ, i.e. every type is inhabited/provable. Another possibility is to regard $Y^\sigma(\lambda x : \sigma.x)$ as a *computation of a proof*, that is to assign to Y^σ the type $(c(\sigma) \to c(\sigma)) \to c(\sigma)$, where $c(\sigma)$ is the type representing the computations over σ. Then, at the cost of a complication of the formal system, we may keep a correspondence between propositions and a subset of types.

In these examples, we have roughly considered computations as values enriched with an element denoting the divergent computations. There are however other possible notions of computations that arise in the study of programming languages. For instance, if we wish to model non-determinism then a computation may consist of a collection of values representing the possible outcomes of a program.

Which are then the common properties of these notions of computation? The notion of monad that we describe in section 8.1 seems to provide a good general framework. We present a general technique to produce a monad out of a category of partial morphisms. In particular the familiar category of dcpo's is revisited in this perspective. In section 8.2 we introduce a call-by-value version of the language PCF studied in chapter 6 which reflects the properties of the function space in a category of partial morphisms. By a variant of the technique presented in theorem 6.3.6, we prove the *adequacy* of the semantic interpretation with respect to the operational semantics. In section 8.3 we describe a class of abstract machines, known as environment machines, for the mechanical evaluation of weak λ-calculi. In section 8.4 we consider the full abstraction problem for the call-by-value λ-calculus. We show that a canonical filter model is fully abstract for the calculus enriched with a parallel join operator. In section 8.5 we revisit the continuation based semantics introduced in section 1.6 from a monadic viewpoint. We introduce a typed call-by-value λ-calculus enriched with *control operators* for the manipulation of the execution flow and study its *Continuation Passing Style* (CPS for short) translation into a standard λ-calculus. The typing of control operators allows to push from intuitionistic to classical logic the Curry-Howard correspondence between typed λ-calculi and propositional calculi. In this respect CPS translations can be regarded as a way to extract an effective content form a classical proof. We also discuss simple variants of environment machines which can handle control operators.

8.1 Representing Computations as Monads

In this section, following [Mog89], we present the notion of computation-as-monad. The monads of *partial computations*, *continuations*, and *non-deterministic computations* will be our leading and motivating examples.

Monads (or triples) are an important category-theoretical notion, we refer to section B.8 for some basic constructions and to [BW85, ML71] for a deeper analysis. What is important here, is to state which are the basic computational properties we wish to formalize. Suppose that $\mathbf{C}$ is our category of data types. An endofunctor $T : \mathbf{C} \to \mathbf{C}$ defines how to go from a certain collection of values to the computations over such values. A natural transformation $\eta : id_{\mathbf{C}} \to T$ determines how a value can be seen as a computation. Another natural transformation $\mu : T^2 \to T$ explains how to flatten a computation of a computation to a computation. These requirements plus certain natural commutation properties are expressed by the following equations (cf. definition B.8.1):

$$\mu_A \circ \eta_{TA} = \mu_A \circ T\eta_A = id_{TA} \qquad \mu_A \circ \mu_{TA} = \mu_A \circ T\mu_A \ .$$

We say that a monad satisfies the *mono requirement* if η_A is a mono, for any object A.

Example 8.1.1 *We give three basic examples of monads with a computational flavour in the category of sets. We leave to the reader the needed verifications (these monads satisfy the mono requirement).*

- *Partial computations. Define* $(\_)_\perp : \mathbf{Set} \to \mathbf{Set}$ *as:*

$$\begin{array}{lll} (X)_\perp & = X \cup \{\perp_X\}, \text{ where } \perp_X \notin X & \\ (f)_\perp(z) & = \left\{ \begin{array}{ll} f(z) & \text{if } z \in X \\ \perp_Y & \text{otherwise} \end{array} \right. & \text{where } f : X \to Y \\ \eta_X(x) & = x & \\ \mu_X(z) & = \left\{ \begin{array}{ll} z & \text{if } z \in X \\ \perp_X & \text{otherwise} \ . \end{array} \right. & \end{array}$$

- *Non-deterministic computations. Define* $P : \mathbf{Set} \to \mathbf{Set}$ *as:*

$$\begin{array}{ll} P(X) = \mathcal{P}_{fin}(X) & P(f)(a) = f(a), \text{ where } f : X \to Y \\ \eta_X(x) = \{x\} & \mu_X(z) = \bigcup z \ . \end{array}$$

- *Continuations. We suppose given a set of* results, *R, containing at least two elements. In order to understand the basic trick behind the notion of computation, one should think of the* double negation *interpretation of classical logic into intuitionistic logic [TvD88]. Let* $\neg X \equiv (X \to R)$, *and define* $C : \mathbf{Set} \to \mathbf{Set}$ *as:*

$$\begin{array}{ll} C(X) & = \neg\neg X \\ C(f) & = \lambda g \in \neg\neg X.\lambda h \in \neg Y.g(h \circ f), \text{ where } f : X \to Y \\ \eta_X(x) & = \lambda h \in \neg X.h(x) \\ \mu_X(H) & = \lambda h \in \neg X.H(\lambda g \in \neg\neg X.g(h)) \ . \end{array}$$

First, let us concentrate on the monads of continuations and non-deterministic computations. We introduce two variants of the imperative language studied in chapter 1, and analyse their interpretations in suitable monads (for the sake of simplicity we leave out recursion and expressions).

$$\begin{array}{ll} L_C & s ::= a \mid skip \mid s;s \mid stop \\ L_N & s ::= a \mid skip \mid s;s \mid s + s \ . \end{array}$$

In L_C we have introduced a statement *stop* whose intuitive effect is that of terminating immediately the execution of a program and return the current state. As already discussed in section 1.6 the "direct" semantics used in section 1.5 is not adequate to interpret commands which alter in some global way the control flow. For instance we should have $[\![stop;s]\!] = [\![stop]\!]$, for any s, which is hopeless if we insist in stating $[\![stop;s]\!] = [\![s]\!] \circ [\![stop]\!]$. The notion of continuation was introduced in section 1.6 precisely to model operators that explicitly manipulate the control flow.

Let Σ be the collection of states. It is natural to take Σ as the collection of results. Then the monad of continuations is given by:

$$C(\Sigma) = \Sigma \to (\Sigma \to \Sigma) \ .$$

The semantics of a program is a morphism from Σ to $C(\Sigma)$. The interpretation for L_C is defined as follows: [1]

$$\begin{array}{ll} [\![skip]\!] = \eta_\Sigma & [\![a]\!] = \eta_\Sigma \circ \underline{a}, \text{ for } \underline{a} : \Sigma \to \Sigma \\ [\![s_1;s_2]\!] = \mu_\Sigma \circ C([\![s_2]\!]) \circ [\![s_1]\!] & [\![stop]\!] = \lambda\sigma.\lambda f.\sigma \ . \end{array}$$

Exercise 8.1.2 *Verify that* $[\![a;b]\!] = \lambda\sigma.\lambda f.f(\underline{b}(\underline{a}\sigma))$, *and* $[\![stop;s]\!] = [\![stop]\!]$.

In L_N we have introduced an operator $+$ for the non-deterministic composition of two statements. The intuition is that the statement $s_1 + s_2$ can choose to behave as either s_1 or s_2. It is then natural to consider the interpretation of a statement as a morphism from Σ to $\mathcal{P}_{fin}(\Sigma)$, where Σ is the collection of states. Hence, using the monad of non-deterministic computations we define:

$$\begin{array}{ll} [\![skip]\!] = \eta_\Sigma & [\![a]\!] = \eta_\Sigma \circ \underline{a}, \text{ for } \underline{a} : \Sigma \to \Sigma \\ [\![s_1;s_2]\!] = \mu_\Sigma \circ P([\![s_2]\!]) \circ [\![s_1]\!] & [\![s_1 + s_2]\!] = \lambda\sigma.[\![s_1]\!]\sigma \cup [\![s_2]\!]\sigma \ . \end{array}$$

An obvious remark is that the interpretations for L_C and L_N are formally identical but for the fourth clause. As a matter of fact we have been using a general pattern in these interpretations which goes under the name of *Kleisli category*. Given a monad (T, η, μ) over a category $\mathbf{C}$ the Kleisli category $\mathbf{K}_T$ is formed as follows:

$$\begin{array}{lll} \mathbf{K}_T = \mathbf{C} & \mathbf{K}_T[d,d'] = \mathbf{C}[d,Td'] & \\ id_d = \eta_d : d \to Td & f \circ g = \mu_{d''} \circ Tf \circ g & \text{for } g : d \to d', f : d' \to d'' \text{ in } \mathbf{K}_T \ . \end{array}$$

[1] This definition is slightly more abstract, but equivalent to the one presented in section 1.6.

The reader will find in [Mog89] more information on this construction, and on its use in the interpretation of a meta-language where the notion of computation is treated abstractly, as a monad with certain desirable properties. Going back to the monads of power-sets, we hint to an application to the modelling of parallel computation. We illustrate the idea on yet another variant of the imperative language considered above:

$$L_P \quad s ::= a \mid skip \mid s; s \mid s \parallel s \ .$$

The intuitive semantics of $s_1 \parallel s_2$ is that of a parallel execution of the state transformations performed by s_1 and by s_2. Since s_1 and s_2 share the same state different orders of execution might generate different final results, as is clear, for instance, in the program $x := 1; x := 0 \parallel x := 1$, which upon termination can associate to x either 0 or 1.

In defining the semantics one has to establish what modifications of the state are *atomic*, i.e. are executed as non-interruptible operations. For instance if we assume that assignment is an atomic operation then the program $x := 0 \parallel x := 1$ will terminate with x having value 0 or 1, and nothing else. The semantics of a program is a collection of sequences of state transformations. For instance we can take:

$$[\![s]\!] \in \mathcal{P}_{fin}((\Sigma \to \Sigma)^+)$$

where $(\Sigma \to \Sigma)^+$ are non-empty finite sequences of functions. In this case it is clear that we can distinguish the interpretations of $x := 0; x := x + 1$ and $x := 1$. The interpretation of a parallel composition is an operator that *shuffles* the sequences in all possible combinations.

Exercise 8.1.3 *Define an interpretation of the language L_P in $\mathcal{P}_{fin}((\Sigma \to \Sigma)^+)$.*

In the presence of divergent programs things are a bit more complicated. What is needed is an analogous of the power-set construction in a category of domains. Various solutions to this problem will be presented in chapter 9. Let us provisionally call $\mathcal{P}_D$ the *powerdomain* operator. The interpretation of the imperative language with recursion is given in a domain of *resumptions* (see, e.g., [Plo83]) which is the least solution of the following equation:

$$R = \Sigma \to \mathcal{P}_D(\Sigma + (\Sigma \times R)) \ .$$

A resumption is a function that takes a state and returns a collection of elements that can be either a state or a pair (state, resumption). Intuitively, a program is interpreted as a possibly infinite sequence of state transformations (cf. exercise 8.1.3) each state transformation in the sequence models an operation that the program can perform atomically on the memory.

Partial morphisms. In example 8.1.1 we have defined the monad of partial computations over **Set**. We show next that the monad of partial computations can be derived in a systematic way from a general notion of *partial morphism*. We then apply this connection between partial morphisms and monads of partial computations to the categories of domains introduced in the previous chapters.

It is standard to consider an equivalence class of monos on an object as a generalized notion of subset. A partial morphism from a to b can then be represented as a total morphism from a subset of a to b. In most interesting examples the domain of convergence of a partial morphism is not arbitrary. For instance it is open (as in **Dcpo**), recursively enumerable, etcetera. It is then reasonable to look for a corresponding categorical notion of admissible mono as specified by the following definition.

Definition 8.1.4 (admissible family of monos) *An admissible family of monos $\mathcal{M}$ for a category $\mathbf{C}$ is a collection $\{\mathcal{M}(a) \mid a \in \mathbf{C}\}$ such that:*
(1) If $m \in \mathcal{M}(a)$ then m is a mono $m : d \to a$.
(2) The identity on a is in $\mathcal{M}(a)$: $id_a \in \mathcal{M}(a)$.
(3) $\mathcal{M}$ is closed under composition i.e.

$$m_1 : a \to b \in \mathcal{M}(b), m_2 : b \to c \in \mathcal{M}(c) \quad \Rightarrow \quad m_2 \circ m_1 : a \to c \in \mathcal{M}(c) \ .$$

(4) M is closed under pullbacks i.e.

$$m : d \to b \in \mathcal{M}(b), f : a \to b \quad \Rightarrow \quad f^{-1}(m) \in \mathcal{M}(a) \ .$$

An admissible family of monos $\mathcal{M}$ on $\mathbf{C}$ enjoys properties which are sufficient for the construction of a related category of partial morphisms $\mathbf{pC}$. [2] A *representative* for a partial morphism from a to b is a pair of morphisms in $\mathbf{C}$, (m, f), where $m : d \to a \in \mathcal{M}(a)$ determines the domain and $f : d \to b$ the functional behavior. The category $\mathbf{pC}$ has the same objects as $\mathbf{C}$ and as morphisms equivalence classes of representatives of partial morphisms, namely:

$$\mathbf{pC}[a, b] = \{[m, f] \mid m : d \to a \in \mathcal{M}(a), f : d \to b\}$$

where $(m : d \to a, f : d \to b)$ is *equivalent* to $(m' : d' \to a, f' : d' \to b)$ iff there is an iso $i : d \to d'$ in $\mathbf{C}$ such that $m' \circ i = m$ and $f' \circ i = f$.

To specify domain and codomain of a partial morphism, we write $[m, f] : a \rightharpoonup b$, and we write $(m, f) : a \rightharpoonup b$ for a representative. Given $(\mathbf{C}, \mathcal{M})$ there is a canonical embedding functor, $Emb : \mathbf{C} \to \mathbf{pC}$, defined as:

$$Emb(a) = a, \quad Emb(f) = [id, f] \ .$$

[2] We refer to [CO88, Mog88, RR88] for extended information on the origins and the development of the theory. The definition of pCCC can already be found in [LM84].

Definition 8.1.5 (lifting) *Given a category enriched with a collection of admissible monos, say* $(\mathbf{C}, \mathcal{M})$ *and an object* a *in* $\mathbf{C}$ *the lifting of* a *is defined as a partial morphism,* $open : (a)_\perp \rightharpoonup a$*, such that (cf. definition 1.4.16):*

$$\forall b \in \mathbf{C}\, \forall f : b \rightharpoonup a\, \exists! f' : b \to (a)_\perp (f = open \circ f') \ . \tag{8.1}$$

The following theorem characterizes the lifting as the right adjoint of the embedding functor and shows that it induces a monad (cf. section 1.4).

Theorem 8.1.6 *(1) The partial category* $(\mathbf{C}, \mathcal{M})$ *has liftings iff the embedding functor has a right adjoint. (2) The lifting functor induces a monad over* $\mathbf{C}$*.*

PROOF HINT. If $f : b \rightharpoonup a$ then we define $f' : b \to (a)_\perp$ according to condition 8.1. (1) ($\Rightarrow$) We define a lifting functor, $Lift : \mathbf{pC} \to \mathbf{C}$, as:

$$Lift(a) = (a)_\perp, \quad Lift(f) = (f \circ open_a)', \text{ where } f : a \rightharpoonup b \ .$$

Next we define a natural iso:

$$\tau : \mathbf{pC}[\_, \_] \to \mathbf{C}[\_, Lift\_], \quad \tau_{a,b}(f) = f' \ .$$

($\Leftarrow$) Given the natural iso τ, we define:

$$(a)_\perp = Lift(a), \quad open_a = \tau^{-1}(id_{(a)_\perp}) \ .$$

(2) This is a mechanical construction of a monad out of an adjunction (cf. section B.8). We define $\eta_a = (id_a)'$, and $\mu_a = \tau_{((a)_\perp)_\perp, a}(open_a \circ open_{(a)_\perp})$. □

Exercise 8.1.7 *Find a notion of admissible mono in* **Set** *that generates the monad of partial computations defined in the example 8.1.1(1).*

The notion of partial cartesian closed category (pCCC) arises naturally when requiring closure under the partial function space.

Definition 8.1.8 (pCCC) *Let* $\mathcal{M}$ *be an admissible collection of monos on the category* $\mathbf{C}$*. The pair* $(\mathbf{C}, \mathcal{M})$ *is a pCCC (partial cartesian closed category) if* $\mathbf{C}$ *is cartesian and for any pair of objects in* $\mathbf{C}$*, say* a, b*, there is a pair*

$$(pexp(a,b), pev_{a,b} : pexp(a,b) \times a \rightharpoonup b)$$

(pev for short) with the universal property that for any $f : (c \times a) \rightharpoonup b$ *there exists a unique* $h : c \to pexp(a,b)$ *(denoted* $p\Lambda_{a,b,c}(f)$*, or* $p\Lambda(f)$ *for short) such that* $pev \circ (h \times id_a) = f$*.*

In other words, for any object b there is a functor *partial exponent on* b, say $b \rightharpoonup \_ : \mathbf{pC} \to \mathbf{C}$, that is right adjoint to the product functor $\_ \times b : \mathbf{C} \to \mathbf{pC}$:

$$\mathbf{pC}[\_ \times b, \_] \cong \mathbf{C}[\_, b \rightharpoonup \_] \ .$$

By instantiating this natural isomorphism, we obtain the following version of *currying*: $a \times b \rightharpoonup c \cong a \to (b \rightharpoonup c)$. By virtue of this isomorphism we can safely confuse $b \rightharpoonup c$ with $\mathbf{pC}[b, c]$. We remark that in any pCCC the lifting can be defined as $(a)_\perp = 1 \rightharpoonup a$, with the morphism $open = pev \circ \langle id, ! \rangle$.

Every pCCC has an object Σ, called *dominance*, that *classifies* the *admissible subobjects* (in the same sense as the object of truth-values Ω classifies arbitrary subobjects in a topos).

Proposition 8.1.9 (dominance) *In every pCCC the object* $\Sigma = (1)_\perp = 1 \rightharpoonup 1$, *called* dominance, *classifies the admissible monos in the following sense, where* $\top = p\Lambda(!) : 1 \to \Sigma$*:*

$$\forall a \, \forall m \in \mathcal{M}(a) \, \exists! \chi : a \to \Sigma \text{ such that } (m, !) \text{ is a pullback for } (\chi, \top) \qquad (8.2)$$

Exercise 8.1.10 *Given a partial category define an* admissible subobject *functor* $\mathcal{M}(\_) : \mathbf{C}^{op} \to \mathbf{Set}$. *Show that the classifier condition 8.2 can be reformulated by saying that there is a natural isomorphism between the functor* $\mathcal{M}(\_)$, *and the hom-functor* $\mathbf{C}[\_, \Sigma]$.

Exercise 8.1.11 *Show that in a pCCC the following isomorphism holds:* $a \to \Sigma \cong a \rightharpoonup 1$.

In order to practice these definitions, let us consider the familiar category of directed complete partial orders and continuous morphisms (**Dcpo**). In **Dcpo** we can choose as admissible monos (i.e. subobjects) the ones whose image is a Scott open. Then the dominance is represented by Sierpinski space $\mathbf{O}$, the two points cpo. The dominance $\mathbf{O}$ classifies the admissible monos because any Scott open U over the dcpo D determines a unique continuous morphism, $f : D \to \mathbf{O}$ such that $f^{-1}(\top) = U$ (this point was already discussed in section 1.2 and it will be fully developed in section 10.1).

Definition 8.1.12 *Let* **Dcpo** *be the category of dcpo's and continuous morphisms. We consider the following class of monos in* **Dcpo***:*

$$m : D \to E \in \mathcal{M}_S \quad \textit{iff} \quad im(m) \in \tau_S(E) \ .$$

We leave to the reader the simple proof of the following proposition.

Proposition 8.1.13 *(1) The class* $\mathcal{M}_S$ *is an admissible family of monos for the category* **Dcpo**. *(2) The related category of partial morphisms is a pCCC.*

We conclude by relating various categories of dcpo's. Let D, E be dcpo's. A *partial continuous* morphism $f : D \rightharpoonup E$ is a partial morphism such that its domain of definition is a Scott open ($Dom(f) \in \tau_S(D)$) and its total restriction, $f_{|Dom(f)} : Dom(f) \to E$, is Scott continuous. We denote with **pDcpo** the category of dcpo's and partial continuous morphisms.

Let D, E be cpo's. Recall from definition 1.4.17 that a strict continuous morphism $f : D \to E$ is a (Scott) continuous morphism such that $f(\bot_D) = \bot_E$. We denote with **sCpo** the category of cpo's and strict continuous morphisms.

Exercise 8.1.14 *(1) Calculate the dominance of* $(\mathcal{M}_S, \mathbf{Dcpo})$. *(2) Define the equivalences among the category of partial morphisms generated by* $(\mathcal{M}_S, \mathbf{Dcpo})$, *the category* **sCpo**, *and the category* **pDcpo**.

8.2 Call-by-value and Partial Morphisms

We apply the idea of distinguishing between total and divergent computations which is implicit in the monad of *partial computations* to the design of a variant of the language PCF (see chapter 6). This gives us the opportunity to revisit the general problem of relating the interpretation of a programming language with the way the programming language is executed.

We may start from the following question (reversing the historical evolution of the topic): for which kind of simply typed λ-calculus does a pCCC provide an adequate interpretation? A crucial point is that we follow a call-by-value evaluation discipline, hence in an application the evaluator has to diverge if the argument diverges. To be more precise, we have to fix the rules of evaluation and observation. We stipulate the following:

(1) The evaluator has to stop at λ-abstractions.

(2) It is possible to observe the termination of a computation of a closed term at all types, equivalently one may say that programs have arbitrary, possibly functional, types.

Contrast these design choices with the definition of the evaluator $\to_{op}$ in chapter 6. There evaluation followed a call-by-name order and observation of termination was allowed only at ground types. As in chapter 6, we wish to relate operational and denotational semantics. The technical development of the adequacy proof goes through three main steps.

(1) A language based on a fixed point extension of the simply typed λ-calculus is introduced and a call-by-value evaluation of closed terms is defined.

(2) A standard interpretation of the language in the pCCC **pDcpo** is specified.

(3) A notion of *adequacy relation* is introduced which allows to relate closed terms and denotations.

It is first proved that the evaluation of a closed term converges to a *canonical* term iff its denotation is a total morphism. As a corollary, a result of adequacy

$$(*)\quad \frac{}{\Gamma \vdash * : 1} \qquad (Asmp)\quad \frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

$$(\multimap_I)\quad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma . M : \sigma \multimap \tau} \qquad (\multimap_E)\quad \frac{\Gamma \vdash M : \sigma \multimap \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$$

$$(Y)\quad \frac{\Gamma \vdash M : (1 \multimap \sigma) \multimap \sigma}{\Gamma \vdash Y^{\sigma} M : \sigma}$$

Figure 8.1: Typing rules for the call-by-value typed λ-calculus

of the interpretation with respect to a natural observational preorder is obtained. The related proof technique introduces a family of *adequacy relations* indexed over types that relate denotations and closed terms. These adequacy relations are a variant of the relations already defined in the adequacy proof for PCF (chapter 6). They combine ideas from the computability technique (a technique used for proofs of strong normalization, see theorems 3.5.20 and 11.5.18) with the *inclusive predicates* technique discussed in chapter 6.

Call-by-value λY-calculus. We consider a variant of the λY-calculus defined in chapter 6 suited to the call-by-value viewpoint. Types and raw terms are defined by the following grammars. We distinguish a special type 1 which is inhabited by the constant $*$. This type corresponds to the terminal object and it is used to define a lifting operator, according to what can be done in every pCCC.

$$\begin{array}{lll} \text{Type Variables} & tv & ::= t \parallel s \parallel \ldots \\ \text{Types} & \sigma & ::= 1 \parallel tv \parallel (\sigma \multimap \sigma) \\ \text{Term Variables} & v & ::= x \parallel y \parallel \ldots \\ \text{Terms} & M & ::= * \parallel v \parallel (\lambda v : \sigma . M) \parallel (MM) \parallel (Y^{\sigma} M) \ . \end{array}$$

Contexts Γ are defined as in chapter 4. Provable typing judgments are inductively defined in figure 8.1 (in the following we often omit the type label from the Y combinator).

The types of the Y clause may seem a bit puzzling at a first glance. One can give a semantic justification by recalling that in a pCCC we define the lifting as $(a)_{\perp} = (1 \multimap a)$, on the other hand the partial function space, say $\multimap$, relates to the total function space, say $\to$, as $a \multimap b = a \to (b)_{\perp}$. So $(1 \multimap \sigma) \multimap \sigma$ is the "same" as $((\sigma)_{\perp} \to (\sigma)_{\perp})$ and the implicit type we give to Y is $((\sigma)_{\perp} \to (\sigma)_{\perp}) \to (\sigma)_{\perp}$,

$$(*) \qquad \frac{}{* \mapsto *}$$

$$(\rightharpoonup_I) \qquad \frac{}{\lambda x : \sigma.M \mapsto \lambda x : \sigma.M}$$

$$(\rightharpoonup_E) \qquad \frac{M \mapsto \lambda x : \sigma.M' \quad N \mapsto C' \quad M'[C'/x] \mapsto C}{MN \mapsto C}$$

$$(Y) \qquad \frac{M(\lambda x : 1.YM) \mapsto C}{YM \mapsto C} \quad (x \text{ fresh})$$

Figure 8.2: Evaluation rules for the call-by-value typed λ-calculus

that is the usual type of a fixed-point combinator over $(\sigma)_\perp$. One good reason to restrict recursion to lifted objects is that these objects do have a least element! A continuous function over a directed complete partial order without a least element does not need to have a fix-point.

Evaluation. In chapter 6 we have defined the reduction relation as the reflexive, transitive closure of a one-step reduction relation $\rightarrow_{op}$. In the following we follow a different style of presentation in which evaluation is presented as a relation between programs, i.e. closed terms, and *canonical forms*. In the case considered here, the canonical forms are the closed, well-typed terms $C, C', \ldots$ that are generated by the following grammar (other examples of definition of the evaluation relation can be found in section 8.3):

$$C ::= * \parallel (\lambda v : \sigma.M) \ .$$

The evaluation relation $\mapsto$ relates closed terms and canonical forms of the same type. Its definition is displayed in figure 8.2.

We write $M \downarrow$ if $\exists C\,(M \mapsto C)$. Note that the definition of the relation $\mapsto$ gives directly a deterministic procedure to reduce, if possible, a closed term to a canonical form. In particular, canonical forms evaluate to themselves.

Interpretation. In order to define an interpretation of our call-by-value λ-calculus we concentrate on the category of directed complete partial orders and partial continuous morphisms. Then, as usual, there is a least fixed point operator over lifted objects that is calculated as the lub of an inductively defined chain.

Let *Dcpo* be the collection of dcpo's. We give a type interpretation that

$(*)$	$[\![\Gamma \vdash * : 1]\!]$	$= !_{[\![\Gamma]\!]}$
$(Asmp)$	$[\![(x_1 : \sigma_1), \ldots, (x_n : \sigma_n) \vdash x_i : \sigma_i]\!]$	$= \pi_{n,i}$
$(\rightharpoonup_I)$	$[\![\Gamma \vdash \lambda x : \sigma . M : \sigma \rightharpoonup \tau]\!]$	$= p\Lambda([\![\Gamma, x : \sigma \vdash M : \tau]\!])$
$(\rightharpoonup_E)$	$[\![\Gamma \vdash MN : \tau]\!]$	$= pev \circ \langle [\![\Gamma \vdash M : \sigma \rightharpoonup \tau]\!], [\![\Gamma \vdash N : \sigma]\!] \rangle$
(Y)	$[\![\Gamma \vdash YM : \sigma]\!]$	$= \bigvee_{n<\omega} f(n)$

Figure 8.3: Interpretation of the call-by-value λ-calculus in **pDcpo**

depends on an assignment $\eta : tv \to Dcpo$ as follows:

$$\begin{array}{lll} [\![1]\!] & = 1 & \text{(the terminal object)} \\ [\![t]\!] & = \eta(t) & \\ [\![\sigma \rightharpoonup \tau]\!] & = [\![\sigma]\!] \rightharpoonup [\![\tau]\!] & \text{(the partial exponent)} . \end{array}$$

The interpretation of a judgment $(x_1 : \sigma_1), \ldots, (x_n : \sigma_n) \vdash M : \sigma$ is a partial morphism of type: $[\![1]\!] \times [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!] \rightharpoonup [\![\sigma]\!]$ ($\times$ associates to the left) as defined in figure 8.3.

- If $\vdash M : \sigma$, that is the term is closed, then the interpretation $f \in [\![1 \rightharpoonup \sigma]\!]$ is either a divergent morphism or a point in $[\![\sigma]\!]$. We write $f \Uparrow$ in the first case and $f \Downarrow$ in the second case. We also write $M \Downarrow$ if $[\![M]\!] \Downarrow$, and we denote with $\perp$ the diverging morphism.
- In $(*)$, $!_{[\![\Gamma]\!]}$ is the unique total morphism into 1.
- In $(\rightharpoonup_E)$, the operation $\langle \_, \_ \rangle$ is a *partial* pairing, that is it is defined only if its arguments are both defined.
- In (Y), let g be $[\![\Gamma \vdash M : (1 \rightharpoonup \sigma) \rightharpoonup \sigma]\!]$, $f(0)$ be the divergent morphism, and $f(n+1) = pev \circ \langle g, \underline{id} \circ f(n) \rangle$. The morphism $\underline{id} : a \rightharpoonup pexp(1, a)$ is uniquely determined by the identity over a, and the morphism $open_a : pexp(1, a) \rightharpoonup a$.

As in chapter 4 we can proceed by induction on the size of the typing proof to establish the following properties of substitution.

Lemma 8.2.1 (substitution) *If $\Gamma, x : \sigma \vdash M : \tau$, and $\Gamma \vdash C : \sigma$ then (1) $\Gamma \vdash M[N/x] : \tau$. (2) $[\![\Gamma \vdash M[N/x] : \tau]\!] = [\![\Gamma, x : \sigma \vdash M : \tau]\!] \circ \langle id, [\![\Gamma \vdash C : \sigma]\!] \rangle$.*

Adequacy. We want to prove that given a well typed closed term M, $M \downarrow$ iff $M \Downarrow$. It is easy to show that if $M \downarrow$ then $M \Downarrow$ as the interpretation is invariant under evaluation and the interpretation of a canonical form is a total morphism. In the other direction the naive attempt of proving ($M \Downarrow \Rightarrow M \downarrow$) by induction on the typing of M does not work. Therefore, we associate to every type σ an

adequacy relation $\mathcal{R}_\sigma$ relating denotations and closed terms of type σ (cf. chapter 6). Adequacy relations enjoy the property

$$(f\,\mathcal{R}_\sigma\,M \text{ and } f\Downarrow)\ \Rightarrow\ M\downarrow$$

moreover they enjoy additional properties so that a proof by induction on the typing can go through.

Definition 8.2.2 (adequacy relation) *A relation $S \subseteq [\![1 \rightharpoonup \sigma]\!] \times \Lambda^o_\sigma$ is an adequacy relation of type σ if it satisfies the following conditions:*

$(C.1)$ $(fSM$ *and* $f\Downarrow) \Rightarrow M\downarrow$
$(C.2)$ $(fSM$ *and* $M \mapsto C$ *and* $M' \mapsto C) \Rightarrow fSM'$
$(C.3)$ $\bot SM$, *for any* $M \in \Lambda^o_\sigma$
$(C.4)$ $(\{f_n\}_{n<\omega}$ *directed in* $[\![1 \rightharpoonup \sigma]\!]$ *and* $\forall n\, f_n\, S\, M) \Rightarrow (\bigvee_{n<\omega} f_n)SM$.

We denote with AR_σ the collection of adequacy relations of type σ. For any type σ, the relation $\{(\bot, M) \mid M \in \Lambda^o_\sigma\}$, is an adequacy relation of type σ.

It is interesting to observe certain geometric properties of adequacy relations. To this end we make explicit a cpo structure on the collection of closed terms. Define an equivalence relation, say $\approx$, on terms by stating that

$$M \approx N \ \text{ iff } \ (M\uparrow \text{ and } N\uparrow) \text{ or } \exists C\,(M \mapsto C \text{ and } N \mapsto C)\ .$$

Given a type σ consider the quotient $\Lambda^o_\sigma/\approx$, with a flat order obtained by assuming that the equivalence class of diverging terms is the least element, and all other equivalence classes are incomparable.

We can now consider $E = [\![1 \rightharpoonup \sigma]\!] \times (\Lambda^o_\sigma/\approx)$ as the product cpo. By definition, a set $P \subseteq E$ is an *admissible predicate* (cf. inclusive predicates in section 6.3) if it is closed under directed sets. Note that any admissible predicate P determines a relation S_P over $[\![1 \rightharpoonup \sigma]\!] \times \Lambda^o_\sigma$ as follows:

$$(f, M) \in S_P \ \text{ iff } \ (f, [M]_\approx) \in P\ .$$

Adequacy relations can be seen as a particular case of admissible predicates.

Exercise 8.2.3 *Let $U = \{(f, [M]_\approx) \mid f\Downarrow$ implies $M\downarrow\}$ and $L = \{(\bot, [M]_\approx) \mid M$ closed$\}$. Verify that U and L are admissible predicates. Next show that the admissible predicates included between L and U are in bijective correspondence with the adequacy relations.*

Definition 8.2.4 *Given an assignment $\theta : tv \to \bigcup_{t\in tv} AR_t$, such that $\theta(t) \in AR_t$ for any t, we associate to every type σ a relation $\mathcal{R}_\sigma \subseteq [\![1 \rightharpoonup \sigma]\!] \times \Lambda^o_\sigma$ as follows:*

$$\begin{array}{ll}
\mathcal{R}_1 & = \{(f, M) \mid f \Uparrow \text{ or } (f\Downarrow \text{ and } M\downarrow)\} \\
\mathcal{R}_t & = \theta(t) \\
\mathcal{R}_{\sigma\to\tau} & = \{(f, M) \mid (f\Downarrow \Rightarrow\ M\downarrow) \text{ and } \forall d, N\,(d\,\mathcal{R}_\sigma\,N \Rightarrow (pev \circ \langle f, d\rangle)\,\mathcal{R}_\tau\,MN)\}\ .
\end{array}$$

Proposition 8.2.5 *The relation $\mathcal{R}_\sigma$ is an adequacy relation of type σ, for any type σ.*

PROOF. We proceed by induction on the structure of σ.

1 By definition of $\mathcal{R}$.

t By definition of θ.

$\sigma \rightharpoonup \tau$ We verify the four conditions.
(C.1) By definition of $\mathcal{R}_{\sigma\rightharpoonup\tau}$.
(C.2) Suppose ($f R_{\sigma\rightharpoonup\tau} M$ and $M \mapsto C$ and $M' \mapsto C$). First observe:

$$(M \mapsto C \text{ and } M' \mapsto C \text{ and } MN \mapsto C') \text{ implies } M'N \mapsto C' \tag{8.3}$$

The interesting case arises if $pev \circ \langle f, d\rangle \Downarrow$. Then we have to show:

$$pev \circ \langle f, d\rangle \,\mathcal{R}_\tau\, MN \text{ implies } pev \circ \langle f, d\rangle \,\mathcal{R}_\tau\, M'N$$

that follows by induction hypothesis on τ and property 8.3.
(C.3) $\perp \mathcal{R}_{\sigma\rightharpoonup\tau} M$ because $pev \circ \langle \perp, d\rangle \cong \perp$, and $d \,\mathcal{R}_\sigma\, N$ implies, by induction hypothesis on τ, $\perp \mathcal{R}_\tau MN$.
(C.4) $pev \circ \langle \bigvee_{n<\omega} f_n, d\rangle = \bigvee_{n<\omega} pev \circ \langle f_n, d\rangle$, but $\forall n\,(f_n \,\mathcal{R}_{\sigma\rightharpoonup\tau}\, M)$ and $d \,\mathcal{R}_\sigma\, N$ implies $\forall n\,(pev \circ \langle f_n, d\rangle \,\mathcal{R}_\tau\, MN)$. The thesis follows by (C.4) over $\mathcal{R}_\tau$. □

Theorem 8.2.6 *If $\Gamma \vdash M : \sigma$, $\Gamma \equiv (x_1 : \sigma_1), \ldots, (x_n : \sigma_n)$, and $d_i \,\mathcal{R}_\sigma\, C_i$, $i = 1, \ldots, n$ then $(\llbracket \Gamma \vdash M : \sigma \rrbracket \circ \langle d_1, \ldots, d_n\rangle) \,\mathcal{R}_\sigma\, M[C_1/x_1, \ldots, C_n/x_n]$.*

PROOF. By induction on the length of the typing judgment. We adopt the following abbreviations: $\langle d_1, \ldots, d_n\rangle \equiv \vec{d}$ and $[C_1/x_1, \ldots, C_n/x_n] \equiv [\vec{C}/\vec{x}]$.

$(*)$ $(\llbracket \Gamma \vdash * : 1 \rrbracket \circ \vec{d}) \,\mathcal{R}_1\, *$, by definition of $\mathcal{R}_1$.

$(Asmp)$ $d_i \,\mathcal{R}_{\sigma_i}\, C_i$, by assumption.

$(\rightharpoonup_I)$ We show $(p\Lambda(\llbracket \Gamma, x : \sigma \vdash M : \tau \rrbracket) \circ \vec{d}) \,\mathcal{R}_{\sigma\rightharpoonup\tau}\, (\lambda x : \sigma.M[\vec{C}/\vec{x}])$. The first condition that defines $\mathcal{R}_{\sigma\rightharpoonup\tau}$ follows by the fact that $\lambda x : \sigma.M[\vec{C}/\vec{x}] \downarrow$. For the second suppose $d \,\mathcal{R}_\sigma\, N$, $N \mapsto C$, and the application is defined, then by inductive hypothesis we have:

$$(\llbracket \Gamma, x : \sigma \vdash M : \tau \rrbracket \circ \langle \vec{d}, d\rangle) \,\mathcal{R}_\tau\, M[\vec{C}/\vec{x}][C/x] \ .$$

We observe:
(1) $pev \circ \langle p\Lambda(\llbracket \Gamma, x : \sigma \vdash M : \tau \rrbracket) \circ \vec{d}, d\rangle = (\llbracket \Gamma, x : \sigma \vdash M : \tau \rrbracket) \circ \langle \vec{d}, d\rangle$.
(2) $M[\vec{C}/\vec{x}][C/x] \mapsto C'$ implies $(\lambda x : \sigma.M[\vec{C}/\vec{x}])N \mapsto C'$
(3) Hence by condition (C.2) follows:

$$(pev \circ \langle p\Lambda(\llbracket \Gamma, x : \sigma \vdash M : \tau \rrbracket) \circ \vec{d}, d\rangle) \,\mathcal{R}_\tau\, (\lambda x : \sigma.M[\vec{C}/\vec{x}])N \ .$$

$(\multimap_E)$ We show $(pev \circ \langle [\![\Gamma \vdash M : \sigma \multimap \tau]\!], [\![\Gamma \vdash N : \sigma]\!] \rangle \circ \vec{d})\, \mathcal{R}_\tau\, (MN)[\vec{C}/\vec{x}]$. By induction hypothesis $([\![\Gamma \vdash M : \sigma \multimap \tau]\!] \circ \vec{d})\, \mathcal{R}_{\sigma\multimap\tau}\, M[\vec{C}/\vec{x}]$ and $([\![\Gamma \vdash N : \sigma]\!] \circ \vec{d})\, \mathcal{R}_\sigma\, N[\vec{C}/\vec{x}]$. The result follows by the definition of $\mathcal{R}_{\sigma\multimap\tau}$.

(Y) We show $(\bigvee_{n<\omega} f(n) \circ \vec{d})\, \mathcal{R}_\sigma\, YM[\vec{C}/\vec{x}]$. We prove by induction that, for each n, $(f(n) \circ \vec{d})\, \mathcal{R}_\sigma\, YM[\vec{C}/\vec{x}]$. The case $n = 0$ follows by (C.3). For the induction step we observe:

$$(pev \circ \langle g, \underline{id} \circ f(n) \rangle)\, \mathcal{R}_\sigma\, M(\lambda x : 1.YM[\vec{C}/\vec{x}]) \ .$$

by induction hypothesis on $\Gamma \vdash M : (1 \multimap \sigma) \multimap \sigma$. Now we use (C.2) to conclude $pev \circ \langle g, \underline{id} \circ f(n) \rangle\, \mathcal{R}_\sigma\, YM[\vec{C}/\vec{x}]$. Hence by (C.4) we have the thesis. □

Corollary 8.2.7 *(1) If* $\vdash M : \sigma$ *then* $M \Downarrow$ *implies* $M \downarrow$.
(2) If $\Gamma \vdash M : \sigma$, $\Gamma \vdash N : \sigma$, *and* $[\![\Gamma \vdash M : \sigma]\!] \leq [\![\Gamma \vdash N : \sigma]\!]$ *then in any context* C *such that* $\vdash C[M] : \tau$ *and* $\vdash C[N] : \tau$ *we have* $C[M] \downarrow$ *implies* $C[N] \downarrow$.

Proof. (1) We apply the theorem 8.2.6 in the case the context is empty.

(2) We prove by induction on the structure of a context C that for any M, N such that $\vdash C[M] : \tau$ and $\vdash C[N] : \tau$,

$$[\![\Gamma \vdash M : \sigma]\!] \leq [\![\Gamma \vdash N : \sigma]\!] \quad \Rightarrow \quad [\![\vdash C[M] : \tau]\!] \leq [\![\vdash C[N] : \tau]\!] \ .$$

Next apply the adequacy theorem to show $C[M] \downarrow \ \Rightarrow \ C[M] \Downarrow \ \Rightarrow \ C[N] \Downarrow \ \Rightarrow \ C[N] \downarrow$. □

8.3 Environment Machines

The efficient reduction of λ-terms is an important research topic (see, e.g., [PJ87]). A central problem is the implementation of the substitution operation. In λ-calculus theory substitution is considered as a meta-operation whose definition involves renaming of bound variables and a complete visit of the term in which the substitution is carried on. In implementations, it is tempting to distribute the price of substitution along the computation. The idea is to record the substitution in a suitable data structure, the *environment*, which is kept on the side during the evaluation. The environment is accessed whenever the actual "value" of a variable is needed.

The weak λ-calculus. Based on this idea we present a class of machines known as *environment machines* which are related to the Categorical Abstract Machine mentioned in section 4.3 (see [Cur91] for the exact connection). We concentrate on the implementation of the *weak* λ-calculus, a λ-calculus in which reduction cannot occur under λ's. Terms are defined as usual, we omit types since they are

$$(\beta)\quad \frac{}{(\lambda x.M)N \to M[N/x]}$$

$$(\mu)\quad \frac{M \to M'}{MN \to M'N} \qquad (\nu)\quad \frac{N \to N'}{MN \to MN'}$$

Figure 8.4: Reduction rules for the weak λ-calculus

$$\frac{}{(\lambda x.M)V \to_v M[V/x]} \qquad \frac{M \to_v M'}{MN \to_v M'N} \qquad \frac{N \to_v N'}{VN \to_v VN'}$$

Figure 8.5: Call-by-value reduction strategy

not relevant to our discussion. The rules for weak reduction are shown in figure 8.4.

Note that the reduction relation $\to^*$ generated by these rules is *not* confluent. For instance consider $(\lambda y.\lambda x.y)(II)$, where I is $\lambda z.z$. This term can be reduced to two distinct normal forms: $\lambda x.II$ and $\lambda x.I$. Call-by-name and call-by-value are two popular reduction strategies for the weak reduction.

- In the call-by-name strategy rule (ν) is omitted. We denote the resulting reduction relation with $\to_n$.
- By definition, a value V is a term which begins with a λ-abstraction. The call-by-value reduction strategy is presented in figure 8.5.

Exercise 8.3.1 *Formalize a call-by-name version of the typed λ-calculus defined in section 8.2. Define a translation of call-by-name in call-by-value according to the type translation $\underline{\sigma \to \tau} = (1 \rightharpoonup \sigma) \rightharpoonup \tau$, where $\to$ is the exponentiation operator for the call-by-name calculus.*

In the study of abstract machines implementing a given strategy, one is often interested in the *evaluation relation* that we conventionally denote with $\mapsto$, in order to distinguish it from the reduction relation (an example of evaluation relation was given in figure 8.2). The evaluation relation relates terms to values (or canonical forms). The evaluation relations $\mapsto_n$ and $\mapsto_v$ for call-by-name and call-by-value, respectively, are shown in figure 8.6.

$$\frac{}{V \mapsto_n V} \qquad \frac{M \mapsto_n \lambda x.M' \quad M'[N/x] \mapsto_n V}{MN \mapsto_n V}$$

$$\frac{}{V \mapsto_v V} \qquad \frac{M \mapsto_v \lambda x.M' \quad N \mapsto_v V' \quad M'[V'/x] \mapsto_v V}{MN \mapsto_v V}$$

Figure 8.6: Evaluation relation for call-by-name and call-by-value

$$\frac{}{x[e] \to e(x)} \qquad \frac{M[e] \to \cdots \to (\lambda x.P)[e']}{MN[e] \to P[e'[N[e]/x]]} \qquad \frac{e(x) \to c}{M[e] \to M[e[c/x]]}$$

Figure 8.7: Weak reduction for the calculus of closures

Exercise 8.3.2 *Let s stand for n or v. Show that: (i) $\mapsto_s \subset \to_s^*$, and (ii) the relations $\mapsto_s$ and $\to_s$ are incomparable with respect to the inclusion relation.*

A weak calculus of closures. Next we formalize the idea of environment. To this end we define a calculus of *closures* which are pairs of λ-terms and environments. Environments and closures are mutually defined as follows:

- An *environment* is a partial morphism $e : \mathit{Var} \rightharpoonup \mathit{Closures}$ where $\mathit{Dom}(e)$ is finite (in particular the always undefined morphism is an environment), and *Closures* is the set of closures.
- A *closure* c is a term $M[e]$ where M is a term and e is an environment.

In general we evaluate closures $M[e]$ such that $FV(M) \subseteq \mathit{Dom}(e)$. The evaluation rules for weak reduction are displayed in figure 8.7. In the second rule, $M[e]$ can be already of the form $(\lambda x.P)[e']$. Observe that the schematic formulation of this rule is needed in order to keep environments at top level.

Environments can be regarded as a technical device to fix the non-confluence of the weak λ-calculus. Indeed it is shown in [Cur91] that the relation $\to^*$ is confluent on closures. Next we formalize the evaluation relations for the call-by-name and call-by-value strategies. By definition, a value v is a closure $(\lambda x.M)[e]$. The rules are shown in figure 8.8.

$$\frac{e(x) \mapsto_n v}{x[e] \mapsto_n v} \qquad \frac{M[e] \mapsto_n \lambda x.M'[e'] \quad M'[e'[N/x]] \mapsto_n v}{MN[e] \mapsto_n v}$$

$$\frac{e(x) \mapsto_v v}{x[e] \mapsto_v v} \qquad \frac{M[e] \mapsto_v \lambda x.M'[e'] \quad N[e] \mapsto_v v' \quad M'[e'[v'/x]] \mapsto_n v}{MN[e] \mapsto_n v}$$

Figure 8.8: Evaluation rules for call-by-name and call-by-value

$$\begin{array}{ll} (x[e], s) & \to (e(x), s) \\ (MN[e], s) & \to (M[e], N[e] : s) \\ (\lambda x.M[e], c : s) & \to (M[e[c/x]], s) \end{array}$$

Figure 8.9: Environment machine for call-by-name

Abstract machines. The evaluation rules described in figure 8.8 are pretty close to the definition of an interpreter. What is still needed is a data structure which keeps track of the terms to be evaluated or waiting for their arguments to be evaluated. Not surprisingly, a *stack* suffices to this end. In the call-by-name strategy, we visit the term in a leftmost outermost order looking for a redex. During this visit the terms that appear as arguments in an application are piled up with their environment in the stack. Therefore the stack s can be regarded as a possibly empty list of closures that we denote with $c_1 : \ldots : c_n$. The related environment machine is described in figure 8.9 as a rewriting system on pairs $(M[e], s)$ of closures and stacks (this formulation is due to Krivine). At the beginning of the evaluation the stack is empty.

In the call-by-value strategy, we need to know if what is on the top of the stack is a function or an argument. For this reason, we insert in the stack markers l for left and r for right that specify if the next closure on the stack is the left or right argument of the evaluation function. Therefore a stack is defined as a possibly empty list of markers $m \in \{l, r\}$ and closures: $m_1 : c_1 : \ldots m_n : c_n$. The related environment machine is described in figure 8.10.

$$\begin{array}{ll} (x[e], s) & \to (e(x), s) \\ (MN[e].s) & \to (M[e], r : N[e] : s) \\ (v, r : c : s) & \to (c, l : v : s) \\ (v, l : \lambda x.M[e] : s) & \to (M[e[v/x]], s) \end{array}$$

Figure 8.10: Environment machine for call-by-value

8.4 A FA Model for a Parallel λ-calculus

We build a filter model for an untyped, call-by-value λ-calculus adapting the techniques already introduced in chapter 3. Following [Bou94] we show that this model is fully abstract when the calculus is enriched with a join operator ($\sqcup$) allowing for the *parallel* evaluation of λ-terms. Evaluation converges as soon as *one* of the terms converges. The join operator fails to be sequential in the sense described in section 2.4 and so one can show that it cannot be defined in the pure λ-calculus. Indeed it can be shown that in the calculus extended with the join operator every compact element of a canonical model based on Scott continuity is definable (i.e. it is the interpretation of a closed term of the $\lambda_\sqcup$-calculus, a similar result was stated in chapter 6 for PCF enriched with a parallel or). This result entails the full abstraction of the model.

The $\lambda_\sqcup$-calculus. We introduce a call-by-value, untyped λ-calculus enriched with a join operator $\sqcup$ and construct a model for it as the collection of filters over a specifically tailored *eats* (cf. definition 3.3.1). The language of terms is defined as follows:

$$\begin{array}{lll} v & ::= & x \parallel y \parallel \ldots \\ M & ::= & v \mid \lambda v.M \mid MM \mid M \sqcup M \ . \end{array}$$

Canonical forms are the closed terms generated by the following grammar:

$$C ::= \lambda v.M \mid C \sqcup M \mid M \sqcup C \ .$$

Finally, the *evaluation relation* is defined inductively on closed terms as shown in figure 8.11. As usual we write $M \downarrow$ if $\exists C\,(M \mapsto C)$.

Exercise 8.4.1 *Observe that a term may reduce to more than one canonical form. Consider the reduction relation naturally associated to the evaluation relation defined in figure 8.11. Observe that this relation is not confluent, e.g.* $(\lambda x.\lambda y.x)(II \sqcup II) \mapsto \lambda y.(I \sqcup I)$ *and* $(\lambda x.\lambda y.x)(II \sqcup II) \mapsto \lambda y.(II \sqcup I)$. *This is a typical problem of* weak λ*-calculi (cf. section 8.3). Define a suitable calculus of* closures *(where environments are evaluated) and show its confluence (a solution is described in [Bou94]).*

$$\frac{M \mapsto \lambda x.M' \quad N \mapsto C' \quad M'[C'/x] \mapsto C}{MN \mapsto C}$$

$$\frac{M \mapsto M_1 \sqcup M_2 \quad M_1N \sqcup M_2N \mapsto C}{MN \mapsto C}$$

$$\frac{}{C \mapsto C} \qquad \frac{M \mapsto C}{M \sqcup N \mapsto C \sqcup N}$$

$$\frac{N \mapsto C}{M \sqcup N \mapsto M \sqcup C} \qquad \frac{M \mapsto C \quad N \mapsto C'}{M \sqcup N \mapsto C \sqcup C'}$$

Figure 8.11: Evaluation relation for the $\lambda_{\sqcup}$-calculus

We have already proved in section 8.2 the adequacy of a model for a call-by-value λ-calculus in which the function space is composed of the *partial* continuous functions. In the following, we build a filter model over an eats for call-by-value, which is a solution of the equation $D = D \rightharpoonup D$. [3] More precisely we work with total morphisms and build the initial solution of the equation $D = D \to (D)_\perp$ in the category of algebraic complete lattices and injection-projection pairs (this solution exists by the techniques presented in chapter 3 and generalized in chapter 7). [4] In a lattice the $\sqcup$ operator can be simply interpreted as the lub. In the definition of eats for call-by-value we have to axiomatize the strict behaviour of the $\rightharpoonup$ operator.

Definition 8.4.2 (v-eats) *An eats for call-by-value (v-eats) is a preorder having all finite glb's and enriched with a binary operation $\rightharpoonup$ which satisfies the following properties (as usual ω denotes a top element):*

$$(1)\quad \frac{\sigma' \leq \sigma \quad \tau \leq \tau'}{\sigma \rightharpoonup \tau \leq \sigma' \rightharpoonup \tau'} \qquad (2)\quad \sigma \rightharpoonup (\tau \wedge \tau') \leq (\sigma \rightharpoonup \tau) \wedge (\sigma \rightharpoonup \tau')$$

$$(3)\quad \sigma \rightharpoonup \omega \leq \omega \rightharpoonup \omega \qquad (4)\quad (\sigma \wedge (\omega \rightharpoonup \omega)) \rightharpoonup \tau \leq \sigma \rightharpoonup \tau\ .$$

Rule (1) and Inequality (2) are inherited from the eats axiomatization. Inequality (3) says that $\omega \rightharpoonup \omega$ is the largest defined element. The inequality $\sigma \leq \omega \rightharpoonup \omega$

[3] In op. cit. similar results are obtained for call-by-name, in this case one works with the equation $D = (D \to D)_\perp$.

[4] Following Boudol, an equivalent presentation of the domain D is as the initial solution of the system of domain equations $D = (V)_\perp$, and $V = V \to D$.

states that σ is a value, this is used in the $\rightharpoonup$-elimination rule in figure 8.12. Inequality (4) states that functions are strict, in other terms the behaviour on undefined elements is irrelevant, for instance we can derive $\omega \rightharpoonup \tau = (\omega \rightharpoonup \omega) \rightharpoonup \tau$. Given a v-eats S, consider the collection of filters $\mathcal{F}(S)$ ordered by inclusion. We write $x \Downarrow$ if $\omega \rightharpoonup \omega \in x$ and $x \Uparrow$ otherwise. We define a strict application operation as follows.

Definition 8.4.3 (strict application) *Given a v-eats S and $x, y \in \mathcal{F}(S)$ define*

$$x \bullet_v y = \begin{cases} \{\tau \mid \sigma \rightharpoonup \tau \in x, \sigma \in y\} & \text{if } x \Downarrow \text{ and } y \Downarrow \\ \uparrow \omega & \text{otherwise .} \end{cases}$$

Definition 8.4.4 (representable function) *Let S be a v-eats. A strict function f over $\mathcal{F}(S)$ is* representable *if $\exists x \, \forall y \, (f(y) = x \bullet_v y)$.*

Proposition 8.4.5 *Let S be a v-eats. Then: (1) $\mathcal{F}(S)$ is an algebraic complete lattice. (2) The strict application operation is well-defined and continuous in both arguments. In particular every representable function is strict continuous.*

PROOF HINT. (1) Follow the corresponding proof in proposition 3.3.10. (2) Simple verification. □

Proposition 8.4.6 *Let T be the smallest theory including an element ω and satisfying the conditions in definition 8.4.2 (cf. definition 3.3.4). Then every strict continuous function over $\mathcal{F}(T)$ is representable.*

PROOF HINT. First show that in the initial v-eats $\wedge_{i \in I} \sigma_i \rightharpoonup \tau_i \leq \sigma \rightharpoonup \tau$ implies $\wedge_{\sigma \leq \sigma_i} \tau_i \leq \tau$, where $\sigma, \sigma_i \leq \omega \rightharpoonup \omega$. Then the proof follows the schema presented in proposition 3.3.18. □

Exercise 8.4.7 *Show that if T is defined as in proposition 8.4.6 then $\mathcal{F}(T)$ is isomorphic to the initial solution of the equation $D = D \to (D)_\perp$ in the category of algebraic complete lattices and embedding projections pairs.*

Definition 8.4.8 (interpretation) *Let S be a v-eats. We define an interpretation function $[\![\_]\!] : \lambda_\sqcup$-term $\to (Env \to \mathcal{F}(S))$, where $Env = Var \to \mathcal{F}(S)$ with generic element ρ. When interpreting a closed term we omit writing the environment as it is irrelevant. As usual if $x \subseteq S$ then $\overline{x}$ denotes the least filter containing x.*

$$[\![x]\!]\rho = \rho(x) \qquad [\![MN]\!]\rho = [\![M]\!]\rho \bullet_v [\![N]\!]\rho$$

$$[\![\lambda x.M]\!]\rho = \overline{\{\sigma \rightharpoonup \tau \mid \tau \in [\![M]\!]\rho[\uparrow \sigma / x]\}} \qquad [\![M \sqcup N]\!]\rho = \overline{[\![M]\!]\rho \cup [\![N]\!]\rho} \, .$$

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{}{\Gamma \vdash M : \omega}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \rightharpoonup \tau} \qquad \frac{\Gamma \vdash M : \sigma \rightharpoonup \tau \quad \Gamma \vdash N : \sigma \quad \sigma \leq \omega \rightharpoonup \omega}{\Gamma \vdash MN : \tau}$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash M \sqcup N : \sigma \wedge \tau}$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \wedge \tau} \qquad \frac{\Gamma \vdash M : \sigma \quad \sigma \leq \tau}{\Gamma \vdash M : \tau}$$

Figure 8.12: Typing rules for the $\lambda_\sqcup$-calculus

Next we define a typing system that allows to compute the interpretation, in the sense that the interpretation of a term is the collection of types that we can assign to it. Types $\sigma, \tau, \ldots$ are elements of a v-eats. Contexts Γ are defined as usual. The typing system is displayed in figure 8.12.

An environment ρ is *compatible* with a context Γ if $x : \sigma \in \Gamma$ implies $\sigma \in \rho(x)$. In this case we write $\Gamma \uparrow \rho$.

Proposition 8.4.9 *For any term of the $\lambda_\sqcup$-calculus the following holds:*

$$[\![M]\!]\rho = \{\sigma \mid \Gamma \vdash M : \sigma, \Gamma \uparrow \rho\} \ .$$

PROOF. $\supseteq$: By induction on the length of the derivation we prove that:

$$\forall \rho \forall \Gamma \uparrow \rho \, (\Gamma \vdash M : \sigma \ \Rightarrow \ \sigma \in [\![M]\!]\rho) \ .$$

$\subseteq$: First we observe a weakening property of the typing system (cf. lemma 3.5.5):

$$\text{if } \Gamma, x : \sigma \vdash M : \tau \text{ then } \Gamma, x : \sigma \wedge \sigma' \vdash M : \tau \tag{8.4}$$

Second, we note that for any environment ρ, the following set is a filter:

$$\{\sigma \mid \exists \Gamma \, (\Gamma \uparrow \rho \text{ and } \Gamma \vdash M : \sigma)\} \tag{8.5}$$

By induction on the structure of M we show that for all ρ, $n \geq 0$:

$$\tau \in [\![M]\!]\rho[\uparrow \sigma_1/x_1, \ldots, \uparrow \sigma_n/x_n] \ \Rightarrow \ \exists \Gamma \, (\Gamma \uparrow \rho \text{ and } \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \tau) \ .$$

Let us consider the case $\lambda x.M$. Fix an environment ρ. By 8.5 it is enough to show that $\tau \in [\![M]\!]\rho[\uparrow \sigma/x]$ implies $\Gamma \vdash \lambda x.M : \sigma \rightharpoonup \tau$, for some Γ. By induction hypothesis, $\Gamma, x : \sigma \vdash M : \tau$, and we conclude by $\rightharpoonup$-introduction. $\Box$

Full abstraction. We outline the proofs of adequacy and full abstraction of the $\lambda_\sqcup$-calculus with respect to the filter model built on the initial v-eats. The adequacy proof follows a familiar technique already discussed in section 8.2. We start by specifying when a closed term *realizes* a type.

Definition 8.4.10 *We define a family of relations $\mathcal{R}_\sigma$ over closed terms as follows:*

$$\begin{array}{ll} \mathcal{R}_\omega & = \Lambda^o_\sqcup \ \ \textit{(all closed terms)} \\ \mathcal{R}_{\sigma\wedge\tau} & = \mathcal{R}_\sigma \cap \mathcal{R}_\tau \\ \mathcal{R}_{\sigma\to\tau} & = \{M \mid M\downarrow \ \textit{and}\ \forall N \in \mathcal{R}_\sigma\,(N\downarrow \Rightarrow\ MN \in \mathcal{R}_\tau)\}\ . \end{array}$$

We write $\models M:\sigma$ if $M \in \mathcal{R}_\sigma$ and $x_1:\sigma_1,\ldots,x_n:\sigma_n \models M:\sigma$ if for all N_i such that $N_i\downarrow$ and $\models N_i:\sigma_i$ $(i=1,\ldots,n)$ we have $\models M[N_1/x_1,\ldots,N_n/x_n]:\sigma$.

Proposition 8.4.11 *If $\Gamma \vdash M:\sigma$ then $\Gamma \models M:\sigma$.*

PROOF HINT. We prove by induction on σ that: (1) $(\lambda x.M)N\vec{Q} \in \mathcal{R}_\sigma$ iff $M[N/x]\vec{Q} \in \mathcal{R}_\sigma$ whenever $N,\vec{Q}\downarrow$, and (2) $(M \sqcup N)\vec{P} \in \mathcal{R}_\sigma$ iff $(M\vec{P} \sqcup N\vec{P}) \in \mathcal{R}_\sigma$. Moreover, we verify that $\sigma \le \tau$ implies $\mathcal{R}_\sigma \subseteq \mathcal{R}_\tau$ by induction on the derivation of $\sigma \le \tau$. Finally, we prove the statement by induction on the length of the typing proof. □

Corollary 8.4.12 *For any closed $\lambda_\sqcup$-term M, $M\downarrow$ iff $[\![M]\!]\Downarrow$ iff $\vdash M:\omega\to\omega$.*

PROOF. We prove that $M \mapsto C$ implies $[\![M]\!] = [\![C]\!]$ by induction on the length of the proof of the evaluation judgment. We observe that for any canonical form C, $\vdash C:\omega\to\omega$, and that $\mathcal{R}_{\omega\to\omega}$ is the collection of convergent (closed) terms. □

This concludes the kernel of the adequacy proof. The full abstraction proof relies on the definability of the compact elements of the model. To this end, we inductively define closed terms M_σ of type σ, and auxiliary terms T_τ, for $\tau \le \omega\to\omega$, in figure 8.13.

Exercise 8.4.13 *The definitions in figure 8.13 are modulo equality, where $\sigma=\tau$ if $\sigma\le\tau$ and $\tau\le\sigma$. Check that we associate a term to every type, and that equal types are mapped to the same term.*

Theorem 8.4.14 *For all types σ,τ such that $\tau \le \omega\to\omega$ the following holds:*

$$[\![M_\sigma]\!] = \uparrow\sigma \qquad [\![T_\tau]\!]\bullet_v x = \begin{cases} [\![I]\!] & \textit{if } \tau \in x \\ \uparrow\omega & \textit{otherwise}\ . \end{cases}$$

$$\begin{array}{lll} M_\omega & = \Omega & \\ M_{\sigma\wedge\tau} & = M_\sigma \sqcup M_\tau & \\ M_{\sigma\rightharpoonup\tau} & = \lambda x.(T_\sigma x)M_\tau & (\sigma \leq \omega \rightharpoonup \omega) \\ T_{\omega\rightharpoonup\omega} & = \lambda f.I & \\ T_{\sigma\wedge\tau} & = \lambda f.(T_\sigma f)(T_\tau f) & (\sigma,\tau \leq \omega \rightharpoonup \omega) \\ T_{\sigma\rightharpoonup\tau} & = \lambda f.T_\tau(f\,M_\sigma) & (\tau \leq \omega \rightharpoonup \omega) \end{array}$$

Figure 8.13: Defining compact elements

PROOF. By induction on σ. We just consider two cases. Case $M_{\sigma\rightharpoonup\tau}$. We check:

$$\tau' \in (\llbracket T_\sigma \rrbracket \uparrow \sigma') \uparrow \tau \quad \Rightarrow \quad \sigma \rightharpoonup \tau \leq \sigma' \rightharpoonup \tau' \ .$$

Case $M_{\sigma\wedge\tau}$. We use $\overline{\uparrow \sigma \cup \uparrow \tau} = \uparrow (\sigma \wedge \tau)$. □

We have derived the adequacy of the interpretation from the soundness of the typing system with respect to the realizability interpretation (proposition 8.4.11). Symmetrically, the full abstraction result will be obtained from a completeness property of the typing system (which follows from the definability theorem 8.4.14).

Definition 8.4.15 *Let M, N be closed terms. A* logical *preorder $M \leq_L N$ is defined as: $\forall \sigma\, (\models M : \sigma \quad \Rightarrow \quad \models N : \sigma)$.*

Corollary 8.4.16 *Let M, N be closed terms. If $M \leq_L N$ then $\llbracket M \rrbracket \leq \llbracket N \rrbracket$.*

PROOF. It is enough to show $\models M : \sigma'$ implies $\vdash M : \sigma'$ by induction on σ'. Let us consider the case for $\sigma' = \sigma \rightharpoonup \tau$. From $\vdash M_\sigma : \sigma$ we derive $\models M_\sigma : \sigma$, by proposition 8.4.11. Without loss of generality we assume $\sigma \leq \omega \rightharpoonup \omega$. Then $M_\sigma \downarrow$. It follows $\models MM_\sigma : \tau$. By induction hypothesis $\vdash MM_\sigma : \tau$. We conclude by the following chain of implications:

$$\begin{array}{lll} \vdash MM_\sigma : \tau & \Rightarrow \ \tau \in \llbracket M \rrbracket \uparrow \sigma & \Rightarrow \ \sigma' \rightharpoonup \tau \in \llbracket M \rrbracket,\ \sigma \leq \sigma' \\ & \Rightarrow \vdash M : \sigma' \rightharpoonup \tau,\ \sigma \leq \sigma' & \Rightarrow \vdash M : \sigma \rightharpoonup \tau \ . \end{array}$$

□

This result virtually concludes the full abstraction proof. It just remains to formally define an operational preorder and to verify that it coincides with the preorder induced by the model.

Definition 8.4.17 *An* applicative simulation *S is a binary relation on closed terms such that whenever MSN: (1) $M\downarrow$ implies $N\downarrow$, and (2) for all P, $P\downarrow$ implies $(MP)\,S\,(NP)$. Let $\leq_{sim}$ be the largest applicative simulation.*

Proposition 8.4.18 *Let M, N be closed terms. If $[\![M]\!] \leq [\![N]\!]$ then $M \leq_{sim} N$.*

PROOF. It follows from the observation that $\{(M,N) \mid [\![M]\!] \leq [\![N]\!]\}$ is a simulation. □

Proposition 8.4.19 *If $M \leq_{sim} N$ then $M \leq_L N$.*

PROOF. We suppose $M \leq_{sim} N$. We prove by induction on σ that $\models M : \sigma$ implies $\models N : \sigma$. □

Corollary 8.4.20 *Let M, N be closed terms. Then $[\![M]\!] \leq [\![N]\!]$ iff $M \leq_{sim} N$ iff $M \leq_L N$.*

PROOF. By corollary 8.4.16 and propositions 8.4.18, 8.4.19. □

Exercise 8.4.21 *(1) Let M, N be closed $\lambda_\sqcup$-terms. Define:*

$$M \leq_{apl} N \quad \textit{iff} \quad \forall P_1, \ldots, P_n (MP_1 \ldots P_n \downarrow \Rightarrow \; NP_1 \ldots P_n \downarrow) \; .$$

Show that $M \leq_{apl} N$ iff $M \leq_{sim} N$. (2) Let M, N be arbitrary terms. Define:

$$M \leq_{op} N \quad \textit{iff} \quad \forall C \textit{ such that } C[M], C[N] \textit{ are closed } (C[M]\downarrow \Rightarrow \; C[N]\downarrow) \; .$$

Show that for M, N closed, $M \leq_{op} N$ iff $M \leq_{apl} N$ (this is called context lemma *in the context of the full abstraction problem for* PCF*, cf. chapter 6).*

8.5 Control Operators and CPS Translation

Most programming languages whose basic kernel is based on typed λ-calculus, also include *control* operators such as *exceptions* or *call-with-current-continuation* (see for instance *Scheme* or *ML*). In the following we show how to type certain control operators and how to give them an adequate functional interpretation. As already hinted in example 8.1.1, the monad of continuations is a useful technical device to approach these problems.

A λ-calculus with control operators. As in section 8.2, we consider a simply typed call-by-value λ-calculus. This language is enriched with a ground type num, numerals $0, 1, \ldots$, and two unary combinators: $\mathcal{C}$ for *control* and $\mathcal{A}$ for *abort*. Formally we have:

$$\begin{array}{lll} \text{Types:} & \sigma & ::= num \parallel (\sigma \rightharpoonup \sigma) \\ \text{Terms:} & v & ::= x \parallel y \parallel \ldots \\ & M & ::= n \parallel x \parallel \lambda v : \sigma.M \parallel MM \parallel \mathcal{C}M \parallel \mathcal{A}M \ . \end{array}$$

We briefly refer to the related calculus as the $\lambda_{\mathcal{C}}$-calculus. In order to formalize the behaviour of the control operators $\mathcal{C}$ and $\mathcal{A}$ it is useful to introduce the notion of (call-by-value) *evaluation context* E (cf. [FFKD87]):

$$E ::= [\ ] \parallel EM \parallel (\lambda x : \sigma.M)E \ .$$

Note that an evaluation context is a context with exactly one hole which is not in the scope of a lambda abstraction. Using evaluation contexts one can provide yet another presentation of the reduction relation. First, we define a collection V of *values* as follows:

$$V ::= n \parallel \lambda v : \sigma.M \ .$$

If we forget about type labels the *one step* reduction relation on terms is defined as follows:

$$\begin{array}{lll} (\beta_v) & E[(\lambda x.M)V] & \rightarrow E[M[V/x]] \\ (\mathcal{C}) & E[\mathcal{C}M] & \rightarrow M(\lambda x.\mathcal{A}E[x]) \quad x \notin FV(E) \\ (\mathcal{A}) & E[\mathcal{A}M] & \rightarrow M \ . \end{array}$$

We can now provide a syntactic intuition for what a continuation for a given term is, and for what is special about a control operator. A redex Δ is defined as follows:

$$\Delta ::= (\lambda v.M)V \parallel \mathcal{C}M \parallel \mathcal{A}M \ .$$

Given a term $M \equiv E[\Delta]$, the *current continuation* is the abstraction of the evaluation context, that is $\lambda x.E[x]$. We will see later that there is at most one decomposition of a term into an evaluation context E and a redex Δ. A control operator is a combinator which can manipulate directly the current continuation. In particular the operator $\mathcal{A}$ disregards the current continuation and starts the execution of its argument, while the operator $\mathcal{C}$ applies the argument to $\lambda x.\mathcal{A}E[x]$, when $\lambda x.E[x]$ is the current continuation.

We illustrate by an example the role of control operators in functional programming. We want to write a function $F : Tree(num) \rightharpoonup num$ where $Tree(num)$ is a given type of binary trees whose nodes are labelled by natural numbers. The function F has to return the product of the labels of the tree nodes, but if it finds that a node has label 0, in this case it has to return zero in a constant number of steps of reduction. Intuitively the termination time has to be independent from

$$(\mathcal{C})\ \frac{\Gamma \vdash M : \neg\neg\sigma}{\Gamma \vdash \mathcal{C}M : \sigma}\quad (\mathcal{A})\ \frac{\Gamma \vdash M : num}{\Gamma \vdash \mathcal{A}M : num}\quad \text{where } \neg\sigma \equiv \sigma \rightharpoonup num\ .$$

$$\begin{array}{lll} (\beta_v) & E[(\lambda x:\sigma.M)V] & \to E[M[V/x]] \\ (\mathcal{C}) & E[\mathcal{C}M] & \to M(\lambda x:\sigma.\mathcal{A}E[x]) \quad \text{where } \mathcal{C}M:\sigma \\ (\mathcal{A}) & E[\mathcal{A}M] & \to M \end{array}$$

Figure 8.14: Typing control operators and reduction rules

the size of the current stack of recursive calls. There is a simple realization of this specification that just relies on the abort operator $\mathcal{A}$ (more involved examples can be found in [HF87]):

$$\begin{array}{lll} let & F(t) = F'(\lambda x.\mathcal{A}x)t & \\ where & F' = \lambda k.Y(\lambda f.\lambda t'. & \text{if } empty(t') \text{ then } 0 \\ & & \text{else if } val(t') = 0 \text{ then } k0 \\ & & \text{else } val(t') * f(left(t')) * f(right(t'))) \ . \end{array}$$

At the beginning of the computation we have $F(t) \to F'[\lambda x.\mathcal{A}x/k]$. If at some point the exceptional branch "if $val(t') = 0 \ldots$" is selected then the following computation is derived, in some evaluation context E:

$$E[(\lambda x.\mathcal{A}x)0] \to E[\mathcal{A}0] \to 0\ .$$

By applying a CPS translation (to be defined next) it is possible to obtain a purely functional program with a similar behaviour. This is an interesting result which finds applications in compilers' design [App92]. On the other hand, one should not conclude that we can forget about control operators. CPS translations tend to be unreadable, and programming directly in CPS style is a tricky business. In practice, control operators are directly available as primitives in functional languages such as ML and Scheme. We refer to [FFKD87] for a syntactic analysis of a λ-calculus with control operators.

Typing control operators. It is possible to type the operators $\mathcal{C}$ and $\mathcal{A}$ coherently with the reduction rules as shown in figure 8.14 (this typing naturally arises in proving subject reduction, cf. proposition 8.5.2). A *program* is a closed term of type num. The reduction rules (β_v), $(\mathcal{C})$, $(\mathcal{A})$ define a deterministic procedure to reduce programs. In the following a subscript $\mathcal{C}$ indicates that we refer to the full $\lambda_{\mathcal{C}}$-calculus.

Proposition 8.5.1 (unique decomposition) *Suppose* $\vdash_{\mathcal{C}} M : \sigma$. *Then either* M *is a value or there is a unique evaluation context* E *and redex* Δ *such that* $M \equiv E[\Delta]$.

PROOF. By induction on the structure of M. The only interesting case is when $M \equiv M'M''$. Then M is not a value, $\vdash_{\mathcal{C}} M' : \tau \rightharpoonup \sigma$, and $\vdash_{\mathcal{C}} M'' : \tau$, for some τ (note that we cannot type $\mathcal{A}$, and $\mathcal{C}$ alone).

- M' is a value. Then $M' \equiv \lambda x : \sigma.M_1$. If M'' is a value take $E \equiv [\ ]$ and $\Delta \equiv (\lambda x : \sigma.M_1)M''$. Otherwise, if M'' is not a value then, by inductive hypothesis, there are E_1, Δ_1 such that $M'' \equiv E_1[\Delta_1]$. Then take $E \equiv M'E_1$ and $\Delta \equiv \Delta_1$.
- M' is not a value. Then, by inductive hypothesis, there are E_1, Δ_1 such that $M' \equiv E_1[\Delta_1]$. Then take $E \equiv E_1M''$ and $\Delta \equiv \Delta_1$. □

Proposition 8.5.2 (subject reduction) *If* $\vdash_{\mathcal{C}} M : num$ *and* $M \rightarrow_{\mathcal{C}} N$ *then* $\vdash_{\mathcal{C}} N : num$.

PROOF. Suppose there are E, Δ such that $M \equiv E[\Delta]$. There are three cases to consider according to the shape of the redex.

- $\Delta \equiv (\lambda x : \sigma.M)V$. This requires a simple form of the substitution lemma. We observe that $x : \sigma \vdash_{\mathcal{C}} M : \tau$ and $\vdash_{\mathcal{C}} V : \sigma$ implies $\vdash_{\mathcal{C}} M[V/x] : \tau$.
- $\Delta \equiv \mathcal{C}M$. Suppose $\vdash_{\mathcal{C}} \mathcal{C}M : \sigma$. Then $\vdash_{\mathcal{C}} M : \neg\neg\sigma$ and $x : \sigma \vdash_{\mathcal{C}} E[x] : num$. Hence $x : \sigma \vdash_{\mathcal{C}} \mathcal{A}E[x] : num$, which implies $\vdash_{\mathcal{C}} \lambda x : \sigma.\mathcal{A}E[x] : \neg\sigma$, and finally $\vdash_{\mathcal{C}} M(\lambda x : \sigma.\mathcal{A}E[x]) : num$.
- $\Delta \equiv \mathcal{A}M$. $\vdash_{\mathcal{C}} \mathcal{A}M : num$ forces $\vdash_{\mathcal{C}} M : num$. Also by definition of program $\vdash_{\mathcal{C}} E[\mathcal{A}M] : num$. □

The previous propositions show that the rules (β_V), $(\mathcal{C})$, $(\mathcal{A})$ when applied to a program define a deterministic evaluation strategy which preserves the well-typing.

Remark 8.5.3 *One may consider other control operators. A popular one is the call-with-current continuation operator (callcc). The typing and reduction rule for the callcc operators can be formalized as follows:*

$$\frac{\Gamma, k : \neg\sigma \vdash M : \sigma}{\Gamma \vdash callcc(\lambda k.M) : \sigma} \quad E[callcc(\lambda k.M)] \rightarrow (\lambda k.kM)(\lambda x.\mathcal{A}E[x]) \ .$$

Exercise 8.5.4 *(1) Find a simulation of the callcc operator using the* $\mathcal{C}$ *operator. (2) Evaluate the expression* $\mathcal{C}(\lambda k.(\lambda x.n)(k\,m))$ *following a call-by-value and a call-by-name order.*

$$\begin{array}{lll} \underline{x} & = \lambda k.kx & \underline{MN} = \lambda k.\underline{M}(\lambda m.\underline{N}(\lambda n.mnk)) \\ \underline{n} & = \lambda k.kn & \underline{\mathcal{C}M} = \lambda k.\underline{M}(\lambda m.m(\lambda z.\lambda d.kz)\lambda x.x) \\ \underline{\lambda x.M} & = \lambda k.k(\lambda x.\underline{M}) & \underline{\mathcal{A}M} = \lambda k.\underline{M}(\lambda x.x) \end{array}$$

Figure 8.15: CPS translation

CPS translation. Next we describe an interpretation of the $\lambda_{\mathcal{C}}$-calculus into the $\lambda_{\mathcal{C}}$-calculus without control operators. We begin with a translation of types.

$$\underline{num} = num \quad \underline{\sigma \rightharpoonup \tau} = \underline{\sigma} \rightharpoonup \neg\neg\underline{\tau} \ .$$

The interpretation of the arrow follows the monadic view where we take num as the *type of results*. From another point of view observe that replacing num with $\bot$ one obtains a fragment of the double-negation translation from intuitionistic to classical logic. The rule for typing the $\mathcal{C}$ operator can then be seen as stating the involutive behaviour of classical negation.

Note that the translation involves both types/formulas and terms/proofs. Indeed a variant of the translation considered here was used by Friedman to extract algorithmic content from a certain class of proofs in (classical) Peano arithmetic (see [Fri78, Gri90, Mur91] for elaborations over this point). We associate to a term M a term $\underline{M}$ without control operators so that:

$$x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash_{\mathcal{C}} M : \sigma \text{ implies } x_1 : \underline{\sigma_1}, \ldots, x_n : \underline{\sigma_n} \vdash \underline{M} : \neg\neg\underline{\sigma} \ .$$

The definition is presented in figure 8.15 (we omit types). This is known as *Continuation Passing Style* translation.

Before giving the explicit typing of the translation we recall three basic combinators of the continuation monad.

$$\begin{array}{ll} M : \sigma & \eta(M) = \lambda k : \neg\sigma \ .kM : \neg\neg\sigma \\ M : \sigma \rightharpoonup \tau & \neg\neg M = \lambda k : \neg\neg\sigma \ .\lambda h : \neg\tau \ .k(\lambda x : num \ .h(Mx)) \\ M : \neg\neg\neg\neg\sigma & \mu(M) = \lambda k : \neg\sigma \ .M(\lambda h : \neg\neg\sigma \ .hk) : \neg\neg\sigma \ . \end{array}$$

The explicitly typed CPS translation is given in figure 8.16.

It is now a matter of verification to prove the following, where conventionally $\underline{\Gamma, x : \sigma} \equiv \underline{\Gamma}, x : \underline{\sigma}$.

Proposition 8.5.5 (typing CPS translation) *With reference to the translation in figure 8.15, if* $\Gamma \vdash_{\mathcal{C}} M : \sigma$ *then* $\underline{\Gamma} \vdash_{\mathcal{C}} \underline{M} : \neg\neg\underline{\sigma}$.

$$
\begin{array}{ll}
\underline{x} & = \lambda k : \neg\sigma\ .kx : \neg\neg\underline{\sigma} \ \text{ if } x : \sigma \\
\underline{n} & = \lambda k : \neg num\ .kn : \neg\neg num \\
\underline{\lambda x : \sigma.M} & = \lambda k : \neg\underline{\sigma \rightharpoonup \tau}\ .k(\lambda x : \underline{\sigma}\ .\underline{M}) : \neg\neg\underline{\sigma \rightharpoonup \tau} \\
\underline{MN} & = \lambda k : \neg\underline{\tau}\ .\underline{M}(\lambda m : \underline{\sigma \rightharpoonup \tau}\ .\underline{N}(\lambda n : \underline{\sigma}\ .mnk)) : \neg\neg\underline{\tau} \\
\underline{\mathcal{C}M} & = \lambda k : \neg\underline{\sigma}\ .\underline{M}(\lambda m : \underline{\neg\neg\sigma}\ .m(\lambda z : \underline{\sigma}\ .\lambda d : \neg num\ .kz)\lambda x : num\ .x) : \neg\neg\underline{\sigma} \\
\underline{\mathcal{A}M} & = \lambda k : \neg num\ .\underline{M}(\lambda x : num\ .x) : \neg\neg num
\end{array}
$$

Figure 8.16: Typing the CPS translation

Exercise 8.5.6 *There are many possible* CPS *translations which from a logical view point correspond to different ways to map a proof in classical logic into a proof in constructive logic. In particular verify that, consistently with the proposed typing, one can give the following translation of application:*

$$\underline{MN} = \lambda k : \neg\underline{\tau}.\underline{N}(\lambda n : \underline{\sigma}.\underline{M}(\lambda m : \underline{\sigma \rightharpoonup \tau}.mnk))\ .$$

The main problem is to show that the CPS translation adequately represents the intended behavior of the control operators. Suppose $\vdash_{\mathcal{C}} M : num$, the desired result reads as follows:

$$M \rightarrow_{\mathcal{C}}^{*} n \ \text{ iff } \ \underline{M}id \rightarrow^{*} n.$$

The difficulty in proving this result consists in relating reductions of M and Mid.

Example 8.5.7 *It is not the case that for a provable judgment* $\vdash M : num$, $M \rightarrow_{\mathcal{C}} N$ *implies* $\underline{M}id \rightarrow^{*} \underline{N}id$. *Consider for instance* $(\lambda x.x)(\mathcal{A}n) \rightarrow_{\mathcal{C}} n$. *Note* $\underline{(\lambda x.x)(\mathcal{A}n)} \rightarrow^{*} \lambda k.n$, *whereas* $\underline{n} = \lambda k.kn$.

An optimized translation. Given a term M, a new translation $\langle M\rangle \equiv \lambda k.M{:}k$ is defined with the following relevant properties:

(1) $\underline{M} \rightarrow^{*} \langle M\rangle$

(2) if $M : num$ and $M \rightarrow N$ then $M{:}id \rightarrow^{*} N{:}id$.

This optimized translation is instrumental to the proof of the adequacy of the CPS translation (cf. following theorem 8.5.14). We limit our attention to the fragment of the calculus without control operators. An extension of the results to the full calculus is possible but it would require a rather long detour (see [DF92]). The translation considered here, also known as colon translation (cf. [Plo75]) performs a more careful analysis of the term, the result is that a certain number of redexes can be statically reduced. By this, we can keep term and CPS-translation in lockstep, hence avoiding the problem presented in example 8.5.7.

Definition 8.5.8 *We define a* ψ-translation *on values. Expected typing: if* $V : \sigma$ *then* $\psi(V) : \underline{\sigma}$.

$$\psi(n) = n \quad \psi(\lambda x : \sigma.M) = \lambda x : \underline{\sigma}.\underline{M} \ .$$

Lemma 8.5.9 *For any* V, $\underline{M}[\psi(V)/x] \equiv \underline{M[V/x]}$.

We associate to every evaluation context E a well-typed closed term $\kappa(E)$ as follows:

$$\begin{array}{ll} \kappa([\ ]) & = \lambda x.x \\ \kappa(E[[\ ]N]) & = \lambda m.\underline{N}(\lambda n.mn\kappa(E)) \\ \kappa(E[V[\ ]]) & = \lambda n.\psi(V)n\kappa(E) \ . \end{array}$$

Let $\mathcal{K}$ be the image of the function κ with generic element K.

Definition 8.5.10 *We define a* semi-colon translation *on pairs* $M{:}K$, *where* M *is closed and* $K \in \mathcal{K}$. *Expected typing: if* $M : \sigma$ *and* $K : \neg\underline{\sigma}$ *then* $M{:}K : num$ *(note the double use of ":").*

$$\begin{array}{ll} V{:}K & = K\psi(V) \\ V_1V_2{:}K & = \psi(V_1)\psi(V_2)K \\ V_1N{:}K & = N{:}\lambda n.\psi(V_1)nK \\ MN{:}K & = M{:}\lambda m.\underline{N}(\lambda n.mnK) \ . \end{array}$$

We observe that if $\Gamma \vdash M : \sigma$ then $\underline{\Gamma} \vdash \langle M \rangle : \neg\neg\underline{\sigma}$. Next we prove three technical lemmas that relate the standard and optimized CPS translations.

Lemma 8.5.11 *If* $\vdash M : \sigma$, $K \in \mathcal{K}$, *and* $\vdash K : \neg\underline{\sigma}$ *then* $\underline{M}K \to^+ M{:}K$.

PROOF. By induction on M and case analysis of the semi-colon translation. For instance let us consider:

$$\underline{MN}K = (\lambda k.\underline{M}(\lambda m.\underline{N}(\lambda n.mnk)))K \ .$$

By induction hypothesis on M, $\underline{MN}K \to^+ M{:}\lambda m.\underline{N}(\lambda n.mnK) = MN{:}K$. □

Lemma 8.5.12 *If* $\vdash M : \sigma$ *and* M *is not a value then*

$$E[M]{:}\kappa(E') \equiv M{:}\kappa(E'[E]) \ .$$

PROOF. By induction on E. For instance let us consider $E \equiv E_1[[\ ]N]$. By induction hypothesis on E_1:

$$\begin{array}{ll} E_1[[M]N]{:}\kappa(E') & \equiv [M]N{:}\kappa(E'[E_1[\ ]]) \\ & \equiv M{:}\lambda m.\underline{N}(\lambda n.mn\kappa(E'[E_1[\ ]])) \\ & \equiv M{:}\kappa(E'[E_1[[\ ]N]]) \ . \end{array}$$

□

Lemma 8.5.13 *If* $\vdash V : \sigma$ *and* $\vdash \kappa(E) : \neg\underline{\sigma}$ *then* $V{:}\kappa(E) \to^* E[V]{:}id$.

PROOF. By induction on the structure of E.

- If $E[V]$ is a value then $E \equiv [\,]$.
- Otherwise we distinguish three cases: (1) $E \equiv E_1[[\,]N]$, N not a value, (2) $E \equiv E_1[[\,]V_1]$, and (3) $E \equiv E_1[V_1[\,]]$. For instance let us consider case (1):

$$\begin{aligned} V{:}\kappa(E_1[[\,]N]) &\equiv V{:}\lambda m.\underline{N}(\lambda n.mn\kappa(E_1[\,])) \\ &\equiv (\lambda m.\underline{N}(\lambda n.mn\kappa(E_1[\,])))\psi(V) \\ &\to \underline{N}(\lambda n.\psi(V)n\kappa(E_1[\,])) \\ &\to^+ N{:}\lambda n.\psi(V)n\kappa(E_1[\,])) \quad \text{(by lemma 8.5.11)} \\ &\equiv VN{:}\kappa(E_1[\,]) \\ &\equiv E_1[VN]{:}id \quad \text{(by lemma 8.5.12)} \\ &\equiv E[V]{:}id\ . \end{aligned}$$

□

Theorem 8.5.14 (adequacy CPS-translation) *Suppose* $\vdash_{\mathcal{C}} M : num$, *where* M *does not contain control operators. Then* $M \to^* n$ *iff* $\underline{M}id \to^* n$.

PROOF.

($\Rightarrow$) Suppose $M \to^* n$. By lemma 8.5.11: $\underline{M}id \to^* M{:}id$. We show $M \to M'$ implies $M{:}id \to^+ M'{:}id$.

$$\begin{aligned} E[(\lambda x.M)V]{:}id &\equiv (\lambda x.M)V{:}\kappa(E) \quad \text{(by lemma 8.5.12)} \\ &\equiv (\lambda x.\underline{M})\psi(V)\kappa(E) \\ &\to \underline{M[V/x]}\kappa(E) \quad \text{(by lemma 8.5.9)} \\ &\to^+ M[V/x]{:}\kappa(E) \quad \text{(by lemma 8.5.11)} \\ &\begin{cases} \to^* E[[V/x]M]{:}id & \text{if } [V/x]M \text{ is a value (by lemma 8.5.13)} \\ \equiv E[[V/x]M]{:}id & \text{otherwise (by lemma 8.5.12)}\ . \end{cases} \end{aligned}$$

($\Leftarrow$) By strong normalization of β-reduction, $M \to^* m$ for some numeral m, hence by ($\Rightarrow$) $\underline{M}id \to^* m$. On the other hand, by hypothesis $\underline{M}id \to^* n$, and by confluence $n = m$. □

Exercise 8.5.15 *Given a program* M *show that when following a call-by-name evaluation of* $M{:}id$ *all redexes are actually call-by-value redexes, that is the rhs of the redex is always a value. This fact is used in [Plo75] to simulate call-by-value reduction in a call-by-name* λ*-calculus.*

$$
\begin{array}{ll}
(x[e], s) & \to (e(x), s) \\
(MN[e], s) & \to (M[e], N[e] : s) \\
(\lambda x.M[e], c : s) & \to (M[e[c/x]], s) \\
(\mathcal{C}M[e], s) & \to (M[e], ret(s)) \\
(\mathcal{A}M[e], s) & \to (M[e], \_) \\
(ret(s), c : s') & \to (c, s)
\end{array}
$$

Figure 8.17: Call-by-name environment machine handling control operators

Environment machines and control operators. Environment machines provide a simple implementation of control operators. The stack of environment machines corresponds to the current evaluation context. The implementation of control operators then amounts to the introduction of operations that allow to manipulate the stack as a whole. To this end we introduce an operator ret that *retracts* a stack into a closure. Roughly, if the stack s corresponds to the evaluation context E then the closure $ret(s)$ corresponds to the term $\lambda x.\mathcal{A}E[x]$. We consider first the situation for call-by-name. The syntactic entities are defined as follows (note that the collection of closures is enlarged to include terms of the shape $ret(s)$).

$$
\begin{array}{ll}
\text{Terms} & M ::= v \mid \lambda v.M \mid MM \mid \mathcal{C}M \mid \mathcal{A}M \\
\text{Environments} & e : Var \rightharpoonup Closures \\
\text{Closures} & c ::= M[e] \mid ret(s) \\
\text{Stack} & s \equiv c_1 : \ldots : c_n \ .
\end{array}
$$

The corresponding machine is described in figure 8.17. The formalization for call-by-value is slightly more complicated. Value closures and stack are redefined as follows (as usual m stands for a marker):

$$
\begin{array}{ll}
\text{Value Closures} & vc ::= (\lambda v.M)[e] \mid ret(s) \\
\text{Stack} & s \equiv m_1 : c_1 \ldots m_n : c_n \ .
\end{array}
$$

The corresponding machine is described in figure 8.18. The last rule deserves some explanation: if $ret(s)$ corresponds to $\lambda x.\mathcal{A}E[x]$, vc corresponds to V, and s' corresponds to E' then the rule implements the reduction:

$$E'[(\lambda x.\mathcal{A}E[x])V] \to E'[\mathcal{A}E[V]] \to E[V] \ .$$

It is possible to relate environment machines and CPS interpretations [LRS94]. We give a hint of the connection. Consider the following system of domain

$$
\begin{array}{ll}
(x[e], s) & \to (e(x), s) \\
(MN[e], s) & \to (M[e], r : N[e] : s) \\
(vc, r : c : s) & \to (c, l : vc : s) \\
(vc, l : \lambda x.M[e] : s) & \to (M[e[vc/x]], s) \\
(\mathcal{C}M[e], s) & \to (M[e], r : ret(s)) \\
(\mathcal{A}M[e], s) & \to (M[e], \_) \\
(vc, l : ret(s) : s') & \to (vc, s)
\end{array}
$$

Figure 8.18: Call-by-value environment machine handling control operators

equations where D is the domain of interpretation of closures, that is terms with an environment $e \in Env$, C is the domain of continuations with generic element k, and R represents a domain of results:

$$
\left\{
\begin{array}{lll}
D & = C \to R & \\
C & = D \times C & k \in C \\
Env & = Var \to D & e \in Env \ .
\end{array}
\right.
$$

We interpret the terms as follows, where *stop* is an arbitrary but fixed element in C, and $ret(k) = \lambda(c, k').ck$.

$$
\begin{array}{ll}
[\![x]\!] e\, k & = e(x)k \\
[\![MN]\!] e\, k & = [\![M]\!] e\, \langle [\![N]\!] e, k \rangle \\
[\![\lambda x.M]\!] e\, \langle d, k \rangle & = [\![M]\!] e[d/x]\, k \\
[\![\mathcal{C}M]\!] e\, k & = [\![M]\!] e\, \langle ret(k), stop \rangle \\
[\![\mathcal{A}M]\!] e\, k & = [\![M]\!] e\, stop \ .
\end{array}
$$

Note that in the interpretation we work up to isomorphism. If we regard the continuation k as representing the stack s and $\langle ret(k), stop \rangle$ as representing $ret(s)$ then this interpretation follows exactly the pattern of the call-by-name machine described in figure 8.17.

Exercise 8.5.16 ** Define a* CPS *interpretation for call-by-value which corresponds to the machine described in figure 8.18.*

Chapter 9

Powerdomains

In example 8.1.1 we have presented a monad of non-deterministic computations which is based on the finite powerset. We seek an analogous of this construction in the framework of domain theory. To this end, we develop in section 9.1 the *convex, lower, and upper powerdomains* in categories of algebraic cpo's [Plo76, Smy78]. In order to relate these constructions to the semantics of *non-deterministic* and *concurrent* computation we introduce in section 9.2 Milner's CCS [Mil89], a simple calculus of processes interacting by *rendez-vous* synchronization on communication channels. We present an operational semantics for CCS based on the notion of *bisimulation*. Finally, in section 9.3 we give a fully abstract interpretation of CCS in a domain obtained from the solution of an equation involving the convex powerdomain [Abr91a].

9.1 Monads of Powerdomains

We look for a construction in domain theory which can play the role of the finite (or finitary) subsets in the category of sets. The need for this development clearly arises when combining recursion with non-determinism. One complication is that, in the context of domain theory, there are several possible constructions which address this problem. Their relevance might depend on the specific application one is considering. In the following we concentrate on three *powerdomains* which rely on the notion of *semi-lattice*.

Definition 9.1.1 *A* semi-lattice *is a set with a binary operation, say* $*$*, that is associative, commutative, and absorptive, that is:*

$$(x * y) * z = x * (y * z), \quad x * y = y * x, \quad x * x = x \ .$$

From our perspective we regard the binary operation of a semi-lattice as a loose generalization of the union operation on powersets. We seek a method for generating freely this algebraic structure from a domain. We illustrate the

construction for preorders and then extend it to algebraic cpo's. Let us consider semi-lattices whose carrier is a preorder.

Definition 9.1.2 *A preordered semi-lattice is a structure $(P, \leq, *)$ where $(P, \leq)$ is a preorder, $(P, *)$ is a semi-lattice, and the semi-lattice operation is monotonic, that is $x \leq x'$ and $y \leq y'$ implies $x*y \leq x'*y'$. Moreover, we say that a preordered semi-lattice $(P, \leq, *)$ is a* join preordered semi-lattice *if it satisfies $x \leq x * y$, and a* meet preordered semi-lattice *if it satisfies $x * y \leq x$.*

Incidentally, we note in the following exercise that every semi-lattice gives rise to a poset with specific properties.

Exercise 9.1.3 *Given a semi-lattice $(P, *)$ define $x \leq_* y$ iff $x * y = y$. Show that $(P, \leq_*)$ is a poset with lub's of pairs. Exhibit a bijective correspondence between semi-lattices and posets with lub's of pairs.*

However, we are looking in the other direction: we want to build a semi-lattice out of a poset. We define the category in which we can perform this construction.

Definition 9.1.4 *We denote with* **SP** *the category of preordered semi-lattices where a morphism $f : (P, \leq, *) \to (P', \leq', *')$ is a monotonic function $f : (P, \leq) \to (P', \leq')$ such that $f(x * y) = f(x) *' f(y)$. Let* **JSP** (**MSP**) *be the full subcategory of* **SP** *composed of join (meet) preordered semi-lattices.*

The category **SP** has a subcategory of semi-lattices whose carriers are algebraic cpo's with a continuous operation $*$, and whose morphisms are continuous.

Definition 9.1.5 *We denote with* **SAcpo** *the category of preordered semi-lattices $(P, \leq, *)$ such that $(P, \leq)$ is an algebraic cpo, the operation $*$ is continuous, and a morphism $f : (P, \leq, *) \to (P', \leq', *')$ is a continuous function $f : (P, \leq) \to (P', \leq')$ such that $f(x * y) = f(x) *' f(y)$. Let* **JSAcpo** (**MSAcpo**) *be the full subcategory of* **SAcpo** *composed of join (meet) preordered semi-lattices.*

We show that given an algebraic cpo there is a freely generated semi-lattice in the category **SAcpo**. In view of the technique of ideal completion (cf. proposition 1.1.21) this problem can be actually decomposed in the problem of freely generating a semi-lattice in the category **SP**, and then completing it to a semi-lattice in the category **SAcpo**. So, let us consider the situation for preorders first. We fix some notation. Let $\mathcal{P}^+_{fin}(X)$ denote the *non-empty* finite subsets of X.

- **P** is the category of preorders and monotonic maps.
- *Forget* : **SP** $\to$ **P** is the functor that forgets the semi-lattice structure.

Theorem 9.1.6 *The functor Forget* : **SP** $\to$ **P** *has a left adjoint Free* : **P** $\to$ **SP** *that is defined as:*

$$Free(P) = (\mathcal{P}^+_{fin}(P), \leq_c, \cup), \quad Free(f)(X) = f(X)$$

where the semi-lattice operation is the set-theoretical union, and the so-called convex *preorder is defined as:*

$$X \leq_c Y \quad \textit{iff} \quad \forall x \in X\, \exists y \in Y (x \leq y) \textit{ and } \forall y \in Y\, \exists x \in X (x \leq y)\ .$$

PROOF. The natural transformation $\tau_{P,S} : \mathbf{P}[P, Forget(S)] \to \mathbf{SP}[Free(P), S]$ is defined as:

$$\tau_{P,S}(f)(X) = f(x_1) * \cdots * f(x_n)$$

where $X = \{x_1, \ldots, x_n\} \in \mathcal{P}^+_{fin}(P)$ and $*$ is the binary operation in S. The inverse is defined as $\tau^{-1}_{P,S}(h)(p) = h(\{p\})$. We have to verify that these morphisms live in the respective categories.

- $\tau_{P,S}(f)$ is monotonic. Suppose $\{x_1, \ldots, x_n\} = X \leq_c Y = \{y_1, \ldots, y_m\}$. By the definition of the convex preorder we can find two multisets $X' = \{\!|w_1, \ldots, w_l|\!\}$ and $Y' = \{\!|z_1, \ldots, z_l|\!\}$ in which the same elements occur, respectively, as in X and Y and such that $w_i \leq z_i$, $i = 1, \ldots, l$. By monotonicity of f, and of the binary operation in S, we have:

$$f(w_1) * \cdots * f(w_l) \leq_S f(z_1) * \cdots * f(z_l)$$

and by absorption:

$$\tau_{P,S}(f)(X) = f(w_1) * \ldots * f(w_l), \quad \tau_{P,S}(f)(Y) = f(z_1) * \ldots * f(z_l)\ .$$

- $\tau_{P,S}(f)$ is a morphism in **SP**. Immediate by associativity and absorption. We leave to the reader the verification that $\tau^{-1}_{P,S}$ is well defined as well as the check of the naturality of τ. □

Remark 9.1.7 *The adjunction described in theorem 9.1.6 canonically induces (cf. theorem B.8.7) a* convex monad $(P_c, \{\_\}, \bigcup)$, *where:*

$$\begin{array}{ll} P_c(D) = (\mathcal{P}^+_{fin}(D), \leq_c) & \\ \{\_\} : D \to P_c(D), & \{\_\}(d) = \{d\} \\ \bigcup : P_c(P_c(D)) \to P_c(D), & \bigcup\{x_1, \ldots, x_m\} = x_1 \cup \cdots \cup x_m\ . \end{array}$$

Theorem 9.1.6 can be adapted to join and meet preordered semi-lattices by following the same proof schema.

Theorem 9.1.8 *The forgetful functors* $Forget : \mathbf{JSP} \to \mathbf{P}$ *and* $Forget : \mathbf{MSP} \to \mathbf{P}$ *have left adjoints* $Free_{\mathbf{JSP}} : \mathbf{P} \to \mathbf{JSP}$ *and* $Free_{\mathbf{MSP}} : \mathbf{P} \to \mathbf{MSP}$*, respectively, defined as:*

$$\begin{array}{ll} Free_{\mathbf{JSP}}(P) & = (\mathcal{P}^+_{fin}(P), \leq_l, \cup) \\ Free_{\mathbf{MSP}}(P) & = (\mathcal{P}^+_{fin}(P), \leq_u, \cup) \\ Free_{\mathbf{JSP}}(f)(X) & = Free_{\mathbf{MSP}}(f)(X) = f(X) \end{array}$$

where the semi-lattice operation is the set-theoretical union, and the so-called lower *and* upper *preorders are defined as:* [1]

$$\begin{array}{l} X \leq_l Y \quad \textit{iff} \quad \forall x \in X\, \exists y \in Y (x \leq y) \\ X \leq_u Y \quad \textit{iff} \quad \forall y \in Y\, \exists x \in X (x \leq y)\ . \end{array}$$

Example 9.1.9 *We consider the poset* $\mathbf{O} = \{\bot, \top\}$ *where as usual* $\bot < \top$*. We suppose that the semantics of a non-deterministic program is an element of* $\mathcal{P}^+_{fin}(\mathbf{O}) = \{\{\bot\}, \{\top\}, \{\bot, \top\}\}$*,* $\bot$ *expressing divergence and* $\top$ *convergence. The convex, lower, and upper preorders induce three distinct preorders on* $\mathcal{P}^+_{fin}(\mathbf{O})$*. In the convex preorder* $\{\bot\} <_c \{\bot, \top\} <_c \{\top\}$*, in the lower preorder* $\{\bot, \top\} = \{\top\}$*, and in the upper preorder* $\{\bot\} = \{\bot, \top\}$*. In this context, the lower preorder can be associated to partial correctness assertions, as it compares the outcomes of a program neglecting divergence, whereas the upper preorder can be associated to total correctness assertions, as it collapses programs that may diverge. The convex preorder is the most discriminating, as it compares computations with respect to both partial and total correctness assertions.*

Let us see how theorem 9.1.6 can be extended to the category **Acpo** of algebraic cpo's and continuous functions via the ideal completion.

- There is a functor $Forget : \mathbf{SAcpo} \to \mathbf{Acpo}$.
- Let $Ide : \mathbf{P} \to \mathbf{Acpo}$ be the ideal completion from preorders to algebraic cpo's which is left adjoint to the relative forgetful functor. Similarly, one can define a functor $SIde : \mathbf{SP} \to \mathbf{SAcpo}$, which makes the ideal completion of the semi-lattice and extends the monotonic binary operation to a continuous one.

Definition 9.1.10 *Let* D *be an algebraic cpo and let* x *stand for* c*,* l*, or* u*. Then we define a function* $P_x[\_] : \mathbf{Acpo} \to \mathbf{Acpo}$ *as follows:* [2]

$$P_x[D] \ = Ide(\mathcal{P}^+_{fin}(\mathcal{K}(D)), \leq_x)\ .$$

Proposition 9.1.11 *(1) If* D *is finite then* $P_c[D]$ *can be characterized as the collection of* convex *subsets with the convex partial order. Namely, we have* $(\{Con(u) \mid u \in \mathcal{P}^+_{fin}(D)\}, \leq_c)$*, where* $Con(u) = \{d \mid \exists d', d'' \in u\, (d' \leq d \leq d')\}$*.*

[1] Observe the combination of the terminologies for semi-lattices and preorders: the lower preorder occurs with join preordered semi-lattices, and the upper preorder occurs with meet preordered semi-lattices. Note that $X \leq_c Y$ iff $X \leq_l Y$ and $X \leq_u Y$.

[2] Note the difference between, say, $P_c(\_)$ as defined in remark 9.1.7, and $P_c[\_]$ as defined here.

(2) For the flat domain $(\omega)_\perp$ the order $P_c[(\omega)_\perp]$ is isomorphic to the following set with the convex preorder, $\{u \mid u \in \mathcal{P}^+_{fin}(\omega)\} \cup \{u \cup \{\perp\} \mid u \subseteq \omega\}$.

PROOF HINT. (1) This follows from the observation that $\mathcal{K}(D) = D$ and the fact that the ideal completion of a finite set does not add any limit point. (2) Left as an exercise. Note that every computation with a countable collection of results may also diverge. □

Exercise 9.1.12 *Characterize $P_x[D]$ when x equals u or l and D is finite or $(\omega)_\perp$.*

The function $P_c[\_]$ can be extended to a functor which is left adjoint to the forgetful functor.

Proposition 9.1.13 *There is a left adjoint Free to the forgetful functor Forget* : **SAcpo** $\to$ **Acpo**.

PROOF HINT. We define $Free(D) = P_c[D]$. Given $f : D \to E$, we define $Free(f)$ on the principal ideals by:

$$Free(f)(\downarrow \{d_1, \ldots, d_m\}) = \{u \in \mathcal{P}^+_{fin}(\mathcal{K}(E)) \mid u \leq_c \{fd_1, \ldots, fd_m\}\} \ .$$

Note that this is an ideal, and that $Free(f)$ can be extended canonically to $P_c[D]$. □

Exercise 9.1.14 *Prove the analogous of proposition 9.1.13 for the categories* **JSAcpo** *and* **MSAcpo**.

Exercise 9.1.15 *Show that the category of Scott domains is closed under the lower and upper powerdomains constructions. Hint: it is enough to prove that every pair of compact elements which is bounded has a lub. On the other hand, show that the category of Scott domains is not closed under the convex powerdomain construction. Hint: consider the domain $\mathbf{T}^2$ where $\mathbf{T} = \{\perp, tt, ff\}$.*

Exercise 9.1.16 *Let D be a bifinite domain and let $\{p_i\}_{i \in I}$ be the associated directed set of image finite projections such that $\bigvee_{i \in I} p_i = id_D$. Show that $\bigvee_{i \in I} P_c[p_i] = P_c[id]$. Conclude that bifinite domains are closed under the convex powerdomain. Extend this result to the lower and upper powerdomains.*

9.2 CCS

The semantics of the programming languages considered so far associates to every input a set of output values. For instance, a finite set if the computation is non-deterministic but finitely branching (cf. example 8.1.1(2)). On the other hand, system applications often require the design of programs which have to interact repeatedly with their environment (e.g. other programs, physical devices,...). In this case the specification of a program as an input-output relation is not adequate. In order to specify the ability of a program to perform a certain action it is useful to introduce the simple notion of labelled transition system.

Definition 9.2.1 *A* labelled transition system *(lts) is triple of sets* $(Pr, Act, \rightarrow)$ *where* $\rightarrow \subseteq Pr \times Act \times Pr$.

We have adapted our notation to a process calculus to be introduced next, Pr stands for the collection of *processes* and Act for the collection of *actions*. We write $p \stackrel{\alpha}{\rightarrow} q$ for $(p, \alpha, q) \in \rightarrow$, to be read as p *makes an action* α *and becomes* q.

Definition 9.2.2 *A lts is said to be* image finite *if, for all* $p \in Pr$, $\alpha \in Act$, *the set* $\{p' \mid p \stackrel{\alpha}{\rightarrow} p'\}$ *is finite. An image finite lts can be represented as a function* $\rightarrow : Pr \times Act \rightarrow \mathcal{P}_{fin}(Pr)$.

Next we present (a fragment of) Milner's Calculus of Communicating Systems (CCS) [Mil89]. CCS is a model of computation in which a set of *agents* interact by *rendez-vous* synchronization on communication channels (syntactically one can think of an agent as a sequential unit of computation, that is as a process that cannot be decomposed in the parallel composition of two or more processes).

In general several agents can compete for the reception or the transmission on a certain channel, however each accomplished communication involves just one sending and one receiving agent. Moreover any agent may attempt at the same time a communication on several channels (a non-deterministic sum is used for this purpose).

In CCS communication is pure synchronization, no data are exchanged between the sender and the receiver. Therefore, it is not actually necessary to distinguish between input and output. All we need to know is when two interactions are one dual of the other. This idea can be formalized as follows. Let L be a *finite* collection of labels (we make this hypothesis to simplify the interpretation described in section 9.3). Each label $l \in L$ has a complement $\overline{l}$ which belongs to $\overline{L} = \{\overline{l} \mid l \in L\}$. The overline symbol can be understood as a special marker that one adds to an element of L. The marker is chosen so that L and $\overline{L}$ are disjoint.

We denote with $a, b, \ldots$ generic elements in $L \cup \overline{L}$. The complement operation is extended to $\overline{L}$ by making it *involutive*, that is $\overline{\overline{a}} = a$. Finally we define the collection of actions $Act = L \cup \overline{L} \cup \{\tau\}$, where $\tau \notin L \cup \overline{L}$. We denote with $\alpha, \beta, \ldots$ generic elements in Act.

The actions $a, \overline{a}$ may be understood as complementary input/output synchronization operations on a channel. The action τ is an *internal action* in the sense that a process may perform it without the cooperation of the environment.

In figure 9.1, we define a calculus of processes which includes basic combinators for termination, sequentialization, non-deterministic sum, parallel composition, and restriction.

A process is well-formed if it is: (i) closed, that is all process variables are in the scope of a *fix* operator, and (ii) guarded, that is all (bound) process variables are preceded by a prefix, for instance $fix\,X.a.X$ is guarded whereas $fix\,Y.(fix\,X.a.X) + Y$ is not. In the following we always assume that processes are

Process variables.	$V ::= X, Y, Z, \ldots$
Processes.	$P ::= 0 \mid V \mid \alpha.P \mid P + P \mid P \mid P \mid P\backslash a \mid \mathit{fix}V.P$

Figure 9.1: Syntax Ccs

well-formed, these are the objects for which an operational semantics is defined. The intuitive operational behaviour of the process operators is as follows. 0 is the terminated process which can perform no action. $a.P$ is the *prefixing* of a to P, that is $a.P$ performs the action a and becomes P. $P + P'$ is the non-deterministic *choice* (sum) between the execution of P and that of P'. The choice operator presented here is very convenient in the development of an *algebra of processes*. On the other hand its implementation on a distributed architecture requires sophisticated and expensive protocols. For this reason most parallel languages adopt a restricted form of non-deterministic choice. $P \mid P'$ is the *parallel composition* of P and P'. $P\backslash a$ is the process P where the channel a has become private to P. This operation is called *restriction*. Finally, *fix* is the least fix-point operator with the usual unfolding computation rule. We define next a lts on processes. The intuitive interpretation of the judgment $P \stackrel{\alpha}{\rightarrow} P'$ is the following:

- If $\alpha \equiv \tau$ then P may reduce to P' by means of an internal autonomous communication.
- If $\alpha \equiv a$ then P may reduce to P' provided the environment supplies a dual action $\overline{a}$.

The definition of the lts proceeds non-deterministically by analysis of the process expression structure. The rules are displayed in figure 9.2. The rules (*sum*) and (*comp*) have a symmetric version which is omitted. Given a process P one may repeatedly apply the derivation rules above and build a possibly infinite tree whose edges are labelled by actions.

Exercise 9.2.3 *(1) Show that any process without a fix operator generates a finite tree. (2) Verify that any* Ccs *process generates an image finite lts. (3) Consider the non-guarded process* $P \equiv \mathit{fix}X.((X.0 + a.0) \mid b.0)$. *Verify* $P \stackrel{a}{\rightarrow} 0 \mid b.0 \mid \cdots \mid b.0$, *for an arbitrary number of* $b.0$*'s. Conclude that* Ccs *with unguarded recursive definitions is* not *image finite.*

The tree representation is still too concrete to provide a reasonable semantics even for finite Ccs processes built out of prefixing and sum. In the first place the sum should be commutative and associative, and in the second place two identical subtrees with the same root should collapse into one. In other words

$$(prefix)\ \frac{}{\alpha.P \stackrel{\alpha}{\to} P} \qquad (sum)\ \frac{P_1 \stackrel{\alpha}{\to} P_1'}{P_1 + P_2 \stackrel{\alpha}{\to} P_1'}$$

$$(comp)\ \frac{P_1 \stackrel{\alpha}{\to} P_1'}{P_1 \mid P_2 \stackrel{\alpha}{\to} P_1' \mid P_2} \qquad (sync)\ \frac{P_1 \stackrel{a}{\to} P_1' \quad P_2 \stackrel{\overline{a}}{\to} P_2'}{P_1 \mid P_2 \stackrel{\tau}{\to} P_1' \mid P_2'}$$

$$(res)\ \frac{P \stackrel{\alpha}{\to} P' \quad \alpha \notin \{a, \overline{a}\}}{P\backslash a \stackrel{\alpha}{\to} P'\backslash a} \qquad (fix)\ \frac{P[fix\, X.P/X] \stackrel{\alpha}{\to} P'}{fix\, X.P \stackrel{\alpha}{\to} P'}$$

Figure 9.2: Labelled transition system for CCS

the sum operator of CCS should form a semi-lattice with 0 as identity. For processes generating a finite tree, it is possible to build a canonical set-theoretic representation. We define inductively:

$$ST_0 = \emptyset \quad ST_{n+1} = \mathcal{P}_{fin}(Act \times ST_n) \quad ST_\omega = \bigcup\{ST_n \mid n < \omega\}\ .$$

If P generates a finite tree then let $[\![P]\!] = \{(\alpha, [\![P']\!]) \mid P \stackrel{\alpha}{\to} P'\}$. For instance one can compute:

$$[\![a.0 \mid \overline{a}.0]\!] = \{(a, \{(\overline{a}, \emptyset)\}), (\tau, \emptyset), (\overline{a}, \{(a, \emptyset)\})\}\ .$$

Exercise 9.2.4 *Verify that the previous interpretation is well-defined for processes generating a finite tree and that it satisfies the semi-lattice equations.*

There are serious difficulties in extending this naive set-theoretic interpretation to infinite processes. For instance one should have:

$$[\![fix\, X.a.X]\!] = \{(a, \{(a, \{(a, \ldots$$

This seems to ask for the construction of a set A such that $A = \{(a, A)\}$. Assuming the standard representation of an ordered pair (x, y) as $\{x, \{x, y\}\}$ we note that this set is *not well-founded* with respect to the *belongs to* relation as $A \in \{a, A\} \in A$. This contradicts the foundation axiom which is often added to, say, Zermelo-Fraenkel set-theory (see e.g. [Jec78]).

On the other hand it is possible to remove the foundation axiom and develop a non-standard set-theory with an *anti-foundation axiom* which assumes the existence of sets like A (see in particular [Acz88] for the development of the connections with process calculi). In section 9.3, we will take a different approach which pursues the construction of a set-theoretical structure in domain theory relying on the *convex powerdomain*. The initial idea, is to associate to $[\![fix\, X.a.X]\!]$

the lub of elements of the shape $\{(a, \{(a, \ldots, \{(a, \bot)\} \ldots)\})\}$, modulo a suitable interpretation of the set-theoretical notation.

In the following we develop the operational semantics of CCS. To this end we introduce the notion of *bisimulation* [Par81] which is a popular notion of equivalence on lts's. Let $(Pr, Act, \rightarrow)$ be a lts. We define an equivalence relation over Pr that can be characterized as the greatest element of a collection of relations known as *bisimulations* or, equivalently, as the greatest fix-point of a certain monotonic operator defined on the powerset of binary relations on Pr.

Definition 9.2.5 (operator $\mathcal{F}$) *Let $(Pr, Act, \rightarrow)$ be a given lts. We define $\mathcal{F} : \mathcal{P}(Pr \times Pr) \rightarrow \mathcal{P}(Pr \times Pr)$ as:*

$$\begin{aligned}\mathcal{F}(X) = \{(p,q) \mid\ & \forall p', \alpha\,(p \xrightarrow{\alpha} p' \Rightarrow \exists q'\,(q \xrightarrow{\alpha} q' \text{ and } (p', q') \in X)) \text{ and} \\ & \forall q', \alpha\,(q \xrightarrow{\alpha} q' \Rightarrow \exists p'\,(p \xrightarrow{\alpha} p' \text{ and } (p', q') \in X))\}\ .\end{aligned}$$

Definition 9.2.6 *The operator $\mathcal{F}$ is iterated as follows:*

$$\begin{aligned}&\mathcal{F}^0 = Pr \times Pr \quad \mathcal{F}^{\kappa+1} = \mathcal{F}(\mathcal{F}^\kappa) \\ &\mathcal{F}^\lambda = \textstyle\bigcap_{\kappa<\lambda} \mathcal{F}^\kappa \quad \text{for } \lambda \text{ limit ordinal}\ .\end{aligned}$$

Proposition 9.2.7 *The operator $\mathcal{F}$ is a monotonic operator over $\mathcal{P}(Pr \times Pr)$.*

PROOF HINT. In the definition 9.2.5, the relation X occurs in positive position. □

It follows from exercise 1.1.9 that the operator $\mathcal{F}$ has a greatest fix-point (gfp), where $gfp(\mathcal{F}) = \bigcap_{\kappa<\mu} \mathcal{F}^\kappa$, for some ordinal μ.

Proposition 9.2.8 *If the lts is image finite then the operator $\mathcal{F}$ preserves codirected sets, in particular $gfp(\mathcal{F}) = \bigcap_{k<\omega} \mathcal{F}^k$.*

PROOF. Suppose $\{S_i\}_{i\in I}$ is a codirected set of relations over Pr. The interesting point is to show:

$$\bigcap_{i\in I} \mathcal{F}(S_i) \subseteq \mathcal{F}(\bigcap_{i\in I} S_i)\ .$$

Suppose $\forall i \in I\,(p\,S_i\,q)$ and $p \xrightarrow{\alpha} p'$. By hypothesis, $\forall i \in I\,\exists q'\,(q \xrightarrow{\alpha} q'$ and $p'\,S_i\,q')$. Moreover, the set $Q = \{q' \mid q \xrightarrow{\alpha} q'$ and $\exists i \in I\,(p'\,S_i\,q')\}$ is finite, and the set $\{S_i\}_{i\in I}$ is codirected. It follows that there has to be a $q' \in Q$ such that $\forall i \in I\,(p'\,S_i\,q')$. To see this, suppose $q', q'' \in Q$, $p'\,S_i\,q'$, and $p'\,S_j\,q''$, for $i, j \in I$. Then $\exists k \in I\,(S_i, S_j \supseteq S_k)$. Moreover, $\exists q''' \in Q\,(p\,S_k\,q''')$, from which $p\,S_i\,q'''$ and $p\,S_j\,q'''$ follows. By applying a symmetric argument on Q we can conclude $p\,\mathcal{F}(\bigcap_{i\in I} S_i)\,q$. □

Definition 9.2.9 (bisimulation) *Let $(Pr, Act, \rightarrow)$ be a given lts. A binary relation $S \subseteq Pr \times Pr$ is a bisimulation if $S \subseteq \mathcal{F}(S)$.*

Exercise 9.2.10 *Show that: (i) the empty and identity relations are bisimulations, (ii) bisimulations are closed under inverse, composition, and arbitrary unions, and (iii) there is a greatest bisimulation. Verify that bisimulations are not closed under (finite) intersection.*

Definition 9.2.11 *Let Pr be the collection of* Ccs *processes. Let $\mathcal{F}$ be the operator related to* Ccs *bisimulation. We denote with $\sim$ the largest* Ccs *bisimulation and we set $\sim^\kappa = \mathcal{F}^\kappa$.*

Exercise 9.2.12 *Show for* Ccs *that $\sim = \sim^\omega$. Hint: apply exercise 9.2.3 and proposition 9.2.8.*

Exercise 9.2.13 *Prove that $\sim$ is a congruence for prefixing, sum, parallel composition, and restriction. Hint: to prove that $P \sim Q$ it is enough to find a bisimulation S such that $P\,S\,Q$. Let $[Pr]$ be the collection of equivalence classes generated by the greatest bisimulation $\sim$ on the set of* Ccs *processes. Extend the operations $+$ and $|$ to $[P]$ and prove that $([Pr], +, [0])$ is a semi-lattice, and that $([Pr], |, [0])$ is a commutative monoid.*

The previous exercise suggests that bisimulation equivalence captures many reasonable process equivalences. However, as stated it is still unsatisfactory as the internal action τ and an input-output action on a channel are treated in the same way. This implies that for instance, the process $\tau.\tau.a.0$ is not bisimilar to the process $\tau.a.0$. One needs to abstract to some extent from the internal actions. A standard approach to this problem, is to consider a *weak* labelled transition system in which any action (in the sense of the lts defined in figure 9.2) can be preceded and followed by an arbitrary number of internal τ-actions.

Definition 9.2.14 *Labelled weak reduction, say $\stackrel{\alpha}{\Rightarrow}$, is a relation over* Ccs *processes which is defined as follows:*

$$P \stackrel{a}{\Rightarrow} P' \quad \textit{iff} \quad P(\stackrel{\tau}{\rightarrow})^* \cdot \stackrel{a}{\rightarrow} \cdot (\stackrel{\tau}{\rightarrow})^* P'$$
$$P \stackrel{\tau}{\Rightarrow} P' \quad \textit{iff} \quad P(\stackrel{\tau}{\rightarrow})^* P' \ .$$

Weak bisimulation *is the greatest bisimulation relation built on top of the weak lts (which is uniquely determined according to definition 9.2.9).*

The properties of weak bisimulation with respect to Ccs operators are described in [Mil89]. We will meet again this equivalence in chapter 16, for the time being we just observe some basic properties.

Exercise 9.2.15 *Verify that the following equations hold for weak-bisimulation:*

$$\alpha.\tau.P = \alpha.P \quad P + \tau.P = \tau.P \quad \alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q) \ .$$

9.3 Interpretation of CCS

We define an interpretation of CCS in the bifinite domain D which is the initial solution of the domain equation:

$$D = P_c[(Act \times D)_\perp] \oplus (1)_\perp \tag{9.1}$$

where $\oplus$ is the coalesced sum (cf. definition 1.4.22), $(\ )_\perp$ is the lifting (cf. definition 1.4.16), and 1 is the one point cpo. The role of the adjoined element $(1)_\perp$ is to represent the terminated process 0 (cf. [Abr91b]). We denote with F the functor associated to $P_c[(Act \times \_)_\perp] \oplus (1)_\perp$.

We will show that the related interpretation captures bisimulation (a full abstraction result). To this end, we will introduce a notion of *syntactic approximation* (definition 9.3.5). A syntactic approximation plays a role similar to that of finite Böhm trees in the λ-calculus (cf. definition 2.3.1): it provides an approximate description of the operational behaviour of a process. It turns out that syntactic approximations, are interpreted in the domain D by compact elements. The key lemma 9.3.12 relates syntactic and semantic approximations. Full abstraction, is then obtained by going to the limit.

The existence of an initial solution to equation 9.1 can be proven by the technique already presented in section 3.1 and generalized in section 7.1. However, in order to relate denotational and operational semantics of CCS it is useful to take a closer look at the structure of D. The domain D is the ω-colimit in $\mathbf{Bif}^{ip}$ of the ω-chain $\{F^n(1), F^n(f_0)\}_{n\in\omega}$ where the morphism $f_0 : 1 \to F(1)$ is uniquely determined in $\mathbf{Bif}^{ip}$ (cf. theorem 7.1.15). We note that, for each $n \in \omega$, the domain $F^n(1)$ is finite. Therefore the ideal completion has no effect, and we have that $F^{n+1}(1) \cong (\mathcal{P}^+_{fin}((Act \times F^n(1))_\perp) \oplus (1)_\perp, \leq_c)$. Every compact element in D can be regarded as an element in $F^n(1)$, for some $n \in \omega$, and, vice versa, every element in $F^n(1)$ can be regarded as a compact element in D. It is actually convenient to build inductively the collection of compact elements.

Definition 9.3.1 (compacts) *The sets K_n, for $n \in \omega$, are the least sets such that:*

$$\frac{}{\{\perp\} \in K_0} \qquad \frac{}{\emptyset \in K_{n+1}}$$

$$\frac{\alpha_i \in Act, d_i \in K_n, m \geq 1}{\{(\alpha_1, d_1), \ldots, (\alpha_m, d_m)\} \in K_{n+1}} \qquad \frac{\alpha_i \in Act, d_i \in K_n, m \geq 0}{\{\perp\} \cup \{(\alpha_1, d_1), \ldots, (\alpha_m, d_m)\} \in K_{n+1}}\ .$$

Proposition 9.3.2 *For any $n \in \omega$, (1) $K_n \subseteq K_{n+1}$, and (2) if $d, d' \in K_n$ then $d \cup d' \in K_n$.*

PROOF HINT. By induction on n. □

It is easy to verify that elements in K_n are in bijective correspondence with elements in $F^n(1)$. The bijection becomes an order isomorphism when the elements in $K = \bigcup_{n\in\omega} K_n$ are ordered as follows.

Definition 9.3.3 (order) *Let $\leq$ be the least relation on K such that (I, J can be empty):*

$$\frac{\forall i \in I\, \exists j \in J\, (\alpha_i = \alpha'_j \text{ and } d_i \leq d'_j) \quad \forall j \in J\, \exists i \in I\, (\alpha_i = \alpha'_j \text{ and } d_i \leq d'_j)}{\{(\alpha_i, d_i) \mid i \in I\} \leq \{(\alpha'_j, d'_j) \mid j \in J\}}$$

$$\frac{\forall i \in I\, \exists j \in J\, (\alpha_i = \alpha'_j \text{ and } d_i \leq d'_j)}{\{\bot\} \cup \{(\alpha_i, d_i) \mid i \in I\} \leq \{(\alpha'_j, d'_j) \mid j \in J\}}$$

$$\frac{\forall i \in I\, \exists j \in J\, (\alpha_i = \alpha'_j \text{ and } d_i \leq d'_j)}{\{\bot\} \cup \{(\alpha_i, d_i) \mid i \in I\} \leq \{\bot\} \cup \{(\alpha'_j, d'_j) \mid j \in J\}}\ .$$

This provides an explicit description of the compact elements. We can assume $D = Ide(K, \leq)$ with the inclusion order, and $\mathcal{K}(D) = \{\downarrow d \mid d \in K\}$. We denote with I, J elements in D. We can explicitly define a chain $\{p_n\}_{n\in\omega}$ of image finite projections on D by:

$$p_n(I) = I \cap K_n \tag{9.2}$$

In figure 9.3 we define inductively on K monotonic functions corresponding to the CCS operators. Of course, these functions can be canonically extended to continuous functions on D. In general, given $f : K^n \to K$ we define $\hat{f} : D^n \to D$ as:

$$\hat{f}(I_1, \ldots, I_n) = \bigcup\{\downarrow f(d_1, \ldots, d_n) \mid d_j \in I_j, j = 1, \ldots, n\} \tag{9.3}$$

Next we define a notion of *syntactic approximation* of a process. Syntactic approximations can be analysed by finite means both at the syntactic level, as it is enough to look at a finite approximation of the bisimulation relation (proposition 9.3.7), and at the semantic level, as they are interpreted as compact elements (definition 9.3.8). To this end we suppose that the language of processes is extended with a constant $\bot$. The notion of labelled transition system and bisimulation for this extended calculus are left unchanged (so $\bot$ behaves as 0, operationally).

Definition 9.3.4 *We define inductively a collection of normal forms $\mathcal{N}_k$:*

$$\frac{}{\bot \in \mathcal{N}_0} \qquad \frac{\forall i \in I\, (N_i \in \mathcal{N}_k) \quad \sharp I < \omega}{\Sigma_{i\in I}\alpha_i.N_i \in \mathcal{N}_{k+1}}\ .$$

If $I = \{1, \ldots, n\}$ then $\Sigma_{i\in I}\alpha_i.N_i$ is a shorthand for $\alpha_1.N_1 + \cdots + \alpha_n.N_n$. Conventionally 0 stands for the empty sum. We consider terms up to associativity and commutativity of the sum, hence the order of the summands is immaterial.

$$
\begin{array}{lll}
Nil \in K & Nil & = \emptyset \\
Pre_\alpha : K \to K & Pre_\alpha(d) & = \{(\alpha, d)\} \\
Sum : K^2 \to K & Sum(d, d') & = d \cup d'
\end{array}
$$

$$
\begin{array}{ll}
Res_a : K \to K & \\
Res_a(\{\bot\}) & = \{\bot\} \\
Res_a(\emptyset) & = \emptyset \\
Res_a(\{(\alpha_i, d_i) \mid i \in I\}) & = \{(\alpha_i, Res_a(d_i)) \mid i \in I, \alpha_i \notin \{a, \overline{a}\}\} \quad (I \neq \emptyset) \\
Res_a(\{\bot\} \cup \{(\alpha_i, d_i) \mid i \in I\}) & = \{\bot\} \cup \{(\alpha_i, Res_a(d_i)) \mid i \in I, \alpha_i \notin \{a, \overline{a}\}\}
\end{array}
$$

$$
\begin{array}{ll}
Par : K^2 \to K & \\
Par(\{\bot\}, d) = Par(d, \{\bot\}) & = \{\bot\} \\
Par(\emptyset, d) = Par(d, \emptyset) & = d \\
Par(\{(\alpha, d)\}, \{(\alpha', d')\}) & = \begin{array}{l} \{(\alpha, Par(d, \{(\alpha', d')\}))\} \cup \\ \{(\alpha', Par(\{(\alpha, d)\}, d'))\} \cup \\ \{(\tau, Par(d, d')) \mid \alpha = \overline{\alpha'} \in L \cup \overline{L}\} \end{array} \\
Par(\{d_i \mid i \in I\}, \{d_j \mid j \in J\}) & = \bigcup_{i \in I, j \in J} Par(d_i, d_j) \quad (\sharp I + \sharp J \geq 3,\ \sharp I, \sharp J \geq 1)
\end{array}
$$

Figure 9.3: Interpretation of CCS operators on compact elements

Definition 9.3.5 (syntactic approximation) *For any process P, we define a k-th approximation $(P)_k \in \mathcal{N}_k$ as follows:*

$$\begin{array}{llll} (P)_0 & = \bot & (\alpha.P)_{k+1} & = \alpha.(P)_k \\ (P+P')_{k+1} & = (P)_{k+1} + (P')_{k+1} & (\mathit{fix}\, X.P)_{k+1} & = ([\mathit{fix}\, X.P/X]P)_{k+1} \end{array}$$

$$\frac{(P)_{k+1} = \Sigma_{i\in I}\alpha_i.P_i}{(P\backslash a)_{k+1} = \Sigma\{\alpha_i.(P_i\backslash a)_k \mid i \in I, \alpha_i \notin \{a,\overline{a}\}\}}$$

$$\frac{(P)_{k+1} = \Sigma_{i\in I}\alpha_i.N_i \quad (P')_{k+1} = \Sigma_{j\in J}\beta_j.N'_j}{(P \mid P')_{k+1} = \left\{ \begin{array}{l} \Sigma_{i\in I}\alpha_i.(N_i \mid \Sigma_{j\in J}\beta_j.N'_j)_k + \\ \Sigma_{j\in J}\beta_j.(\Sigma_{i\in I}\alpha_i.N_i \mid N'_j)_k + \\ \Sigma\{\tau.(N_i \mid N'_j)_k \mid i \in I,\ j \in J, \alpha_i = \overline{\beta_j}\}\ . \end{array} \right.}$$

To show that the definition is well-founded we define a measure *nfix* on processes that counts the number of recursive definitions at top level:

$$\begin{array}{l} \mathit{nfix}(0) = \mathit{nfix}(X) = \mathit{nfix}(\alpha.P) = 0 \\ \mathit{nfix}(P\backslash a) = \mathit{nfix}(P) \\ \mathit{nfix}(P \mid P') = \mathit{nfix}(P + P') = max\{\mathit{nfix}(P), \mathit{nfix}(P')\} \\ \mathit{nfix}(\mathit{fix}\, X.P) = 1 + \mathit{nfix}(P)\ . \end{array}$$

By the hypothesis that recursive definitions are guarded we have $\mathit{nfix}(\mathit{fix}\, X.P) = 1 + \mathit{nfix}(P[\mathit{fix}\, X.P/X])$.

Exercise 9.3.6 *Prove that the definition of k-th approximation is well-founded. by induction on $(n, \mathit{nfix}(P), P)$.*

Proposition 9.3.7 *Let P, Q be processes. Then:*

$$P \sim Q \quad \textit{iff} \quad \forall k \in \omega\, ((P)_k \sim^k (Q)_k)\ .$$

Proof. First, we observe that for any process P:

$$\begin{array}{ll} (P)_0 & = \bot \in \mathcal{N}_0 \\ (P)_{k+1} & = \Sigma\{\alpha.(P')_k \mid P \xrightarrow{\alpha} P'\} \in \mathcal{N}_{k+1}\ . \end{array}$$

It follows that for any process P, and for any $k \in \omega$, $(P)_k \sim^k P$. Combining with proposition 9.2.8 and using the transitivity of the relation $\sim^k$ we can conclude that:

$$P \sim Q \ \text{ iff } \ \forall k \in \omega\, (P \sim^k Q) \ \text{ iff } \ \forall k \in \omega\, ((P)_k \sim^k (Q)_k)\ .$$

□

Definition 9.3.8 *We define an interpretation in K of the normal forms (with reference to the operators in figure 9.3):*

$$\begin{array}{llll} [\![\bot]\!]^K & = \{\bot\} & [\![0]\!]^K & = \emptyset \\ [\![\alpha.N]\!]^K & = Pre_\alpha([\![N]\!]^K) & [\![N+N']\!]^K & = Sum([\![N]\!]^K, [\![N']\!]^K) \ . \end{array}$$

Proposition 9.3.9 *Let $N, N' \in \mathcal{N}_n$ be normal forms, for $n \in \omega$. Then (1) $[\![N]\!]^K \in K_n$, and (2) $N \sim^n N'$ iff $[\![N]\!]^K = [\![N']\!]^K$.*

PROOF HINT. By induction on n. □

The interpretation of normal forms is canonically lifted to all processes, by taking the continuous extensions (cf. equation 9.3) of the functions defined in figure 9.3, and interpreting *fix* as the least fix-point. This is spelled out in the following definition.

Definition 9.3.10 (interpretation) *Let V be the collection of process variables, and let $\rho : V \to D$ be an environment. We interpret a process P in the environment ρ as follows:*

$$\begin{array}{ll} [\![0]\!]\rho & = \downarrow(\emptyset) \\ [\![\alpha.P]\!]\rho & = \bigcup\{\downarrow(\{(\alpha,d)\}) \mid d \in [\![P]\!]\rho\} \\ [\![P+P']\!]\rho & = \bigcup\{\downarrow(d \cup d') \mid d \in [\![P]\!]\rho, d' \in [\![P']\!]\rho\} \\ [\![P\backslash a]\!]\rho & = \bigcup\{\downarrow(Res_a(d)) \mid d \in [\![P]\!]\rho\} \\ [\![P \mid P']\!]\rho & = \bigcup\{\downarrow(Par(d,d')) \mid d \in [\![P]\!]\rho, d' \in [\![P']\!]\rho\} \\ [\![fix X.P]\!]\rho & = \bigcup_{n\in\omega} I_n \text{ with } I_0 = \downarrow(\{\bot\}),\ I_{n+1} = [\![P]\!]\rho[I_n/X] \\ [\![X]\!]\rho & = \rho(X) \ . \end{array}$$

Exercise 9.3.11 *Prove that the function $\lambda d.[\![P]\!]\rho[d/X]$ is continuous.*

Lemma 9.3.12 (approximation) *For any process P, $n \in \omega$,*

$$p_n([\![P]\!]) = [\![P]\!] \cap K_n = [\![(P)_n]\!] \ .$$

PROOF HINT. By induction on $(n, nfix(P), P)$. We consider a few significative cases.

$\alpha.P$ We compute:

$$\begin{array}{ll} [\![\alpha.P]\!] \cap K_{n+1} & = \bigcup\{\downarrow(\{(\alpha,d)\}) \mid d \in [\![P]\!] \cap K_n\} \\ & = \bigcup\{\downarrow(\{(\alpha,d)\}) \mid d \leq [\![(P)_n]\!]^K\} \\ & = \downarrow(\{(\alpha, [\![(P)_n]\!]^K)\}) = \downarrow([\![(\alpha.P)_{n+1}]\!]^K) \ . \end{array}$$

$fix X.P$ A direct application of the induction hypothesis:

$$\begin{array}{ll} [\![fix X.P]\!] \cap K_{n+1} & = [\![P[fix X.P/X]]\!] \cap K_{n+1} \\ & = \downarrow([\![(P[fix X.P/X])_{n+1}]\!]^K) \\ & = \downarrow([\![(fix X.P)_{n+1}]\!]^K) \ . \end{array}$$

$P \mid P'$ We have:

$$\begin{aligned}
[\![P \mid P']\!] \cap K_{n+1} &= \bigcup\{\downarrow (Par(d,d')) \mid d \in [\![P]\!], d' \in [\![P']\!]\} \cap K_{n+1} \\
&= \bigcup\{\downarrow (Par(d,d')) \mid d \in [\![P]\!] \cap K_{n+1}, d' \in [\![P']\!] \cap K_{n+1}\} \cap K_{n+1} \\
&= \bigcup\{\downarrow (Par(d,d')) \mid d \leq [\![(P)_{n+1}]\!]^K, d' \in [\![(P')_{n+1}]\!]^K\} \cap K_{n+1} \\
&= \downarrow (Par([\![(P)_{n+1}]\!]^K, [\![(P')_{n+1}]\!]^K)) \cap K_{n+1} = \downarrow ([\![(P \mid P')_{n+1}]\!]^K) \ .
\end{aligned}$$

□

Theorem 9.3.13 (full abstraction) *Let P, Q be* Ccs *processes. Then:*

$$P \sim Q \quad \textit{iff} \quad [\![P]\!] = [\![Q]\!] \ .$$

Proof. By proposition 9.3.7, $P \sim Q$ iff $\forall k \in \omega\,((P)_k \sim^k (Q)_k)$. By proposition 9.3.9, $\forall k \in \omega\,((P)_k \sim^k (Q)_k)$ iff $\forall k \in \omega\,[\![(P)_k]\!] = [\![(Q)_k]\!]$. By lemma 9.3.12, $[\![(P)_k]\!] = p_k([\![P]\!])$, and since $\bigvee_{k\in\omega} p_k = id$, we have $[\![P]\!] = [\![Q]\!]$ iff $\forall k \in \omega\,(p_k([\![P]\!]) = p_k([\![Q]\!]))$. □

Remark 9.3.14 *(1) Not all compact elements in D are definable by a* Ccs *process. Indeed, if this was the case then full abstraction would fail. For instance we would have $P + \bot \sim P$, whereas in general $[\![P + \bot]\!] \neq [\![P]\!]$. It is possible to refine the bisimulation relation by taking diverging elements into account (cf. [Abr91b]). (2) At the time of writing, the denotational framework described here has not been adapted in a satisfying way to capture weak bisimulation.*

Chapter 10

Stone Duality

We introduce a fundamental duality that arises in topology from the consideration of points versus opens. A lot of work in topology can be done by working at the level of opens only. This subject is called pointless topology, and can be studied in [Joh82]. It leads generally to formulations and proofs of a more constructive nature than the ones "with points". For the purpose of computer science, this duality is extremely suggestive: points are programs, opens are program properties. The investigation of Stone duality for domains has been pioneered by Martin-Löf [ML83] and by Smyth [Smy83b]. The work on intersection types, particularly in relation with the D^∞ models, as exposed in chapter 3, appears as an even earlier precursor. We also recommend [Vic89], which offers a computer science oriented introduction to Stone duality.

In sections 10.1 and 10.1 we introduce locales and Stone duality in its most abstract form. In sections 10.2 and 10.4 we specialise the construction to Scott domains, and to bifinite domains. On the way, in section 10.3, we prove Stone's theorem: every Boolean algebra is order-isomorphic to an algebra of subsets of some set X, closed under set-theoretic intersection, union, and complementation. The proof of Stone's theorem involves a form of the axiom of choice (Zorn's lemma), used in the proof of an important technical lemma, known as Scott open filter theorem. In contrast, the dualities for domains can be proved more directly, as specialisations of a simple duality, which we call the basic domain duality. (We have not seen this observation in print before.) Once the dualities are laid down, we can present the domain constructions "logically", by means of formulas representing the compact opens. This programme, which has been carried out quite thoroughly by Abramsky [Abr91b], is the subject of sections 10.5 and 10.6.

10.1 Topological Spaces and Locales

If we abstract away from the order-theoretic properties of the opens of a topology, we arrive at the following definition.

Definition 10.1.1 (locale) *A locale, or a frame, is an ordered set $(A, \leq)$ satisfying the following properties:*

1. *every finite subset of A has a greatest lower bound,*
2. *every subset of A has a least upper bound,*
3. *the following distributivity property holds, for any $x \in A$ and $Y \subseteq A$:*

$$x \wedge (\bigvee Y) = \bigvee \{x \wedge y \mid y \in Y\}.$$

In particular, there is a minimum (the empty lub) and a maximum (the empty glb). For any topological space $(X, \Omega X)$, the collection ΩX, ordered by inclusion, is a locale. The elements of a locale will be often called opens, even if the locale does not arise as a topology. We make some remarks about this definition:

- Condition (1) is implied by condition (2), which in fact implies that all glb's exist. But the maps we consider being those which preserve finite glb's and arbitrary lub's, it is natural to put condition (1) explicitly in the definition of a locale.
- Locales are equivalently defined as complete Heyting algebras, where a complete Heyting algebra is a complete lattice which viewed as a partial order is cartesian closed (cf. definition 4.2.3 and example B.5.3).

Definition 10.1.2 (frames/locales) *The category* **Frm** *of frames is the category whose objects are locales, and whose morphisms are the functions preserving finite glb's and all lub's. The category* **Loc** *of locales is defined as* $\mathbf{Frm}^{op}$*. Locales and frames are named such according to which category is meant.*

Since we develop the theory of locales as an abstraction of the situation with topological spaces, it is natural to focus on **Loc**: for any continuous function $f : (X, \Omega X) \to (Y, \Omega Y)$, the function f^{-1} is a locale morphism from ΩX to ΩY.

Definition 10.1.3 *The functor $\Omega : \mathbf{Top} \to \mathbf{Loc}$, called localisation functor, is defined by*

$$\Omega(X, \Omega X) = \Omega X \quad \Omega(f) = f^{-1}.$$

The two-points flat domain $\mathbf{O} = \{\bot, \top\}$ (cf. example 1.1.6) lives both in **Top** (endowed with its Scott topology $\{\emptyset, \{\top\}, \{\bot, \top\}\}$) and in **Loc**, and plays a remarkable role in each of these categories:

Top: For any topological space $(X, \Omega X)$, the opens in ΩX are in one-to-one correspondence with the continuous functions from $(X, \Omega X)$ to $\mathbf{O}$.

Loc: $\mathbf{O}$ considered as a locale is terminal in **Loc**: let $(A, \leq)$ be a locale, and $f : A \to \{\bot, \top\}$ be a locale morphism. Then $f(\bot) = \bot$ and $f(\top) = \top$, since the minimum and maximum elements must be preserved.

The fact that $\mathbf{O}$ is terminal in **Loc** suggests a way to recover a topological space out of a locale. The standard categorical way of defining a point in an object A is to take a morphism from the terminal object. We shall thus define points of a locale A to be locale morphisms $g : \{\bot, \top\} \to A$. One may approach the reconstruction of points from opens in a perhaps more informative way by analyzing the situation of the locale ΩX of some topological space X. If we try to recover a point x out of the locale ΩX, the simplest idea is to collect all the opens that contain it. The fact that the mapping $x \mapsto \{U \mid x \in U\}$ is injective is exactly the property of the topology to be T_0. Any set $F = \{U \mid x \in U\}$ (x fixed) has the following properties:

1. It is closed under finite intersections.
2. It is upward closed.
3. If $P \subseteq \Omega X$ and $\bigcup P \in F$, then $U \in F$ for some U in P.

The first two conditions are those defining filters (cf. chapter 3). We abstract the three properties together in the following definition, which generalises and extends definition 3.3.8.

Definition 10.1.4 ((completely coprime) filter) *Let A be a partial order. A filter over A is an ideal over A^{op}, that is, a non-empty subset F such that:*

1. If $x \in F$ and $x \leq y$, then $y \in F$.

2. $\forall x, y \in F \; \exists z \in F \; z \leq x, y$.

A filter F in (a complete lattice) A is called completely coprime if:

3. $\forall Y \subseteq A \;\; (\bigvee Y \in F \;\Rightarrow\; \exists y \in Y \;\; y \in F)$

We consider two restrictions of condition (3) (in a lattice, in a dcpo, respectively):

3′. $\forall Y \subseteq_{fin} A \;\; (\bigvee Y \in F \;\Rightarrow\; \exists y \in Y \;\; y \in F)$

3″. $\forall Y \subseteq_{dir} A \;\; \bigvee Y \in F \;\Rightarrow\; \exists y \in Y \;\; y \in F)$

A filter satisfying (3′) is called coprime, and a filter satisfying (3″) is called a Scott-open filter *(indeed, (3″) is the familiar condition defining Scott opens, cf. definition 1.2.1). We write:*

$\mathcal{F}(A)$ for the set of filters of A,
$Spec(A)$ for the set of coprime filters of A,
$Pt(A)$ for the set of completely coprime filters of A.

All these sets are ordered by inclusion.

Remark that if $\bot = \bigvee \emptyset \in F$, then F is not coprime. In particular, coprime filters are proper subsets.

Here is a third presentation of the same notion. The complement G of a completely coprime filter F is clearly closed downwards, and is closed under arbitrary lub's. In particular $G = \downarrow (\bigvee G)$, and we have, by conditions (1) and (2):

$$\bigwedge P \leq \bigvee G \ (P \textit{ finite}) \ \Rightarrow \ \exists p \in P \ p \leq \bigvee G.$$

Definition 10.1.5 ((co)prime) *Let $(X, \leq)$ be a partial order. An element x of X is called prime if*

$$\forall P \subseteq_{fin} X \ (\bigwedge P \textit{ exists and } \bigwedge P \leq x) \Rightarrow \exists p \in P \ p \leq x.$$

Dually, a coprime element is an element y such that for any finite $Q \subseteq X$, if $\bigvee P$ exists and $x \leq \bigvee Q$, then $x \leq q$ for some $q \in Q$.

Notice that a prime element cannot be $\top$ $(= \bigwedge \emptyset)$. Dually, the minimum, if it exists, is not coprime.

Exercise 10.1.6 *Show that if $(X, \leq)$ is a distributive lattice, then $z \in X$ is coprime iff it is* irreducible, *i.e., $z = x \vee y$ always implies $x = z$ or $y = z$.*

Thus the complements of completely coprime filters are exactly the sets of the form $\downarrow q$, where q is prime, and there is a one-to-one correspondence between prime opens and completely coprime filters. The following proposition summarises the discussion.

Proposition 10.1.7 (points) *The following are three equivalent definitions of the set $Pt(A)$ of points a locale A:*

locale morphisms from **O** *to A,*
completely coprime filters of A,
prime elements of A.

We write $x \models p$ to mean $x(p) = \top$, $p \in x$, or $p \not\leq x$, depending on how points are defined. The most standard view is $p \in x$ (completely coprime filters).

We have further to endow the set of points of a locale A with a topology.

Proposition 10.1.8 *For any locale A, the following collection $\{U_p\}_{p \in A}$ indexed over A is a topology over $Pt(A)$: $U_p = \{x \mid x \models p\}$. This topology, being the image of $p \mapsto \{x \mid x \models p\}$, is called the image topology.*

PROOF. We have $U_p \cap U_q = U_{p \wedge q}$, and $\bigcup \{U_p \mid p \in B\} = U_{\bigvee B}$ for any $B \subseteq A$. □such thattionThe Basic Duality The following result states that we did the right construction to get a topological space out of a locale. We call spatialisation, or Pt, the operation which takes a locale to its set of points with the image topology.

Proposition 10.1.9 ($\Omega \dashv Pt$) *The spatialisation $A \mapsto Pt(A)$ provides a right adjoint to the localisation functor Ω (cf. definition 10.1.3). The counity at A is the map $p \mapsto \{x \mid x \models p\}$ (in* **Loc***), and the unity is the map $x \mapsto \{U \mid x \in U\}$ (in* **Top***).*

PROOF HINT. Take as inverses:

$$\begin{array}{ll} f \mapsto (p \mapsto f^{-1}(\{y \mid y \models p\})) & (f : X \to Pt(B) \text{ (in } \mathbf{Top}), p \in B) \\ g \mapsto (x \mapsto \{p \mid x \in g(p)\}) & (g : \Omega X \to B \text{ (in } \mathbf{Loc}), x \in X)\ . \end{array}$$

□

Theorem 10.1.10 (basic duality) *The adjunction $\Omega \dashv Pt$ cuts down to an equivalence, called the basic duality, between the categories of* spatial *locales and of* sober *spaces, which are the locales at which the counity is iso and the topological spaces at which the unity is iso, respectively.*

PROOF. Cf. exercise B.6.4. □

We shall restrict the basic duality to some full subcategories of topological spaces and locales.

The following is an intrinsic description of sober spaces. We recall that the closure $\overline{A}$ of a subset A is the smallest closed subset containing it.

Proposition 10.1.11 (sober-irreducible) *Let $(X, \Omega X)$ be a T_0-space. The following are equivalent:*

1. *$(X, \Omega X)$ is sober,*
2. *each irreducible (cf. exercise 10.1.6) closed set is of the form $\overline{\{x\}}$ for some x.*
3. *each prime open is of the form $X \backslash \overline{\{x\}}$ for some x.*

PROOF. Looking at the unity of the adjunction, sober means: "all the completely coprime filters are of the form $\{U \mid x \in U\}$". Unravelling the equivalence of definitions of points, this gets reformulated as: "all the prime opens are of the form $\bigcup\{U \mid x \notin U\}$", which is the complement of $\overline{\{x\}}$. □

Remark 10.1.12 *Any set of the form $\overline{\{x\}}$ is irreducible, so that in sober spaces, the irreducible closed sets are exactly those of the form $\overline{\{x\}}$ for some x.*

By definition of spatiality, a locale A is spatial if and only if, for all $a, b \in A$:

$$(\forall\, x \in Pt(A)\ \ x \models a \Rightarrow x \models b) \ \Rightarrow\ a \leq b$$

or equivalently: $a \not\leq b \Rightarrow \exists\, x \in Pt(A)\ x \models a$ and $x \not\models b$. Actually, it is enough to find a Scott-open filter F such that $a \in F$ and $b \notin F$. But a form of the axiom of choice is needed to prove this.

Theorem 10.1.13 (Scott-open filter) *Let A be a locale. The following properties hold:*

1. For every Scott-open filter F, we have $\bigcap\{x \in Pt(A) \mid F \subseteq x\} = F$.

2. A is spatial iff for all $a, b \in A$ such that $a \not\leq b$ there exists a Scott-open filter F such that $a \in F$ and $b \notin F$.

PROOF. (1) $\supseteq$ is obvious. We prove $\subseteq$ by contraposition. Suppose $a \notin F$. We want to find an x such that $F \subseteq x$ and $a \notin x$. We claim that there exists a prime open p such that $p \notin F$ and $a \leq p$. Then we can take $x = \{c \mid c \not\leq p\}$. Consider the set P of opens b such that $b \notin F$ and $a \leq b$. It contains a, and every chain of P has an upper bound in P (actually the lub of any directed subset of P is in P, because F is Scott open). By Zorn's lemma P contains a maximal element q. We show that q is prime. Suppose that S is a finite set of opens, and that $b \not\leq q$ for each b in S. Then $b \vee q$ is larger than q, and thus belongs to F, by maximality of q. Since F is a filter, it also contains $\bigwedge\{b \vee q \mid b \in S\} = (\bigwedge S) \vee q$, which is therefore larger than q, by maximality of q. A fortiori $\bigwedge S \not\leq q$. Hence q is prime and the claim follows.

(2) One direction follows obviously from the fact that a point is a fortiori a Scott open filter. Conversely, if $a \in F$ and $b \notin F$, by (1) there exists a point x such that $F \subseteq x$ and $b \notin x$. Then x fits since $a \in F$ and $F \subseteq x$ imply $a \in x$. □

We shall not use theorem 10.1.13 for the Stone dualities of domains. But it is important for Stone's theorem (section 10.3). Another characterisation of sobriety and spatiality is obtained by exploiting the fact that in the adjunction $\Omega \dashv Pt$ the counity is mono (by definition of the topology on $Pt(A)$, the map $p \mapsto \{x \mid x \models p\}$ is surjective, hence, as a locale morphism, is a mono).

Proposition 10.1.14 *Spatial locales and sober spaces are those topological spaces which are isomorphic to ΩX for some topological space X, and to $Pt(A)$ for some locale A, respectively.*

PROOF. By application of lemma B.6.6 to the adjunction $\Omega \dashv Pt$. □

We now exhibit examples of sober spaces. If a topological space is already T_0, one is left to check that the mapping $x \mapsto \{U \mid x \in U\}$ reaches all the completely coprime filters.

Proposition 10.1.15 *T_2-spaces are sober.*

PROOF. Let W be a prime open; in particular its complement is non-empty. Suppose that two distinct elements x, y are in the complement, and take disjoint U, V containing x, y respectively. Then $U \cap V$, being empty, is a fortiori contained

in W, but neither U nor V are, contradicting primeness of W. Thus a prime open W is necessarily the complement of a singleton $\{x\}$. We conclude by proposition 10.1.11 (in a T_1-space, $\{x\} = \overline{\{x\}}$). □

In exercise 1.2.6 we have anticipated that algebraic dcpo's are sober. This provides an example of a non-T_2 (even non-T_1, cf. chapter 1) sober space. Actually, more generally, continuous dcpo's (cf. definition 5.1.1) are sober. Before proving this, we exhibit a friendlier presentation of $\overline{\{x\}}$ in suitable topologies on partial orders.

Proposition 10.1.16 *Given a poset X, and a topology Ω over X, the following are equivalent:*

1. $\forall\, x \in X\ \overline{\{x\}} = {\downarrow} x$,
2. *weak* $\subseteq \Omega \subseteq$ *Alexandrov*,
3. $\leq_\Omega\, = \leq$,

where the weak topology is given by the basis $\{X \setminus {\downarrow} x \mid x \in X\}$, and where $\leq_\Omega$ is the specialisation ordering defined by Ω[1].

PROOF. (2) $\Rightarrow$ (1) If $x \in A$ (A closed), then ${\downarrow} x \subseteq A$, since $\Omega \subseteq$ *Alexandrov*. Moreover ${\downarrow} x$ is closed, since *weak* $\subseteq \Omega$. Hence $\overline{\{x\}} = {\downarrow} x$.

(1) $\Rightarrow$ (2) If $\overline{\{x\}} = {\downarrow} x$, then a fortiori ${\downarrow} x$ is closed, hence *weak* $\subseteq \Omega$. If A is closed and $x \in A$, then $\overline{\{x\}} \subseteq A$, hence $\Omega \subseteq$ *Alexandrov*.

(2) $\Leftrightarrow$ (3) We have: $\leq \subseteq \leq_\Omega \Leftrightarrow (x \in U, x \leq y \Rightarrow y \in U) \Leftrightarrow \Omega \subseteq$ *Alexandrov*.

- (*weak* $\subseteq \Omega$) $\Rightarrow$ ($\leq_\Omega \subseteq \leq$): Suppose $x \not\leq y$. Then $X \setminus ({\downarrow} y)$ is an open containing x but not y, hence $x \not\leq_\Omega y$.
- ($\leq_\Omega \subseteq \leq$ and $\Omega \subseteq$ *Alexandrov*) $\Rightarrow$ (*weak* $\subseteq \Omega$): We have to prove that any $X \setminus ({\downarrow} x)$ is in Ω. Pick $y \in X \setminus ({\downarrow} x)$, i.e., $y \not\leq x$. Then $\leq_\Omega \subseteq \leq$ implies that there exists an open U such that $y \in U$ and $x \notin U$. Since $\Omega \subseteq$ *Alexandrov* implies $z \notin U$ for any $z \leq x$, we have $U \subseteq X \setminus ({\downarrow} x)$. □

Proposition 10.1.16 applies in particular to the Scott topology τ_S, since *weak* $\subseteq \tau_S$ (cf. exercise 1.2.2) and since $\tau_S \subseteq$ *Alexandrov* by definition.

Proposition 10.1.17 *The Scott topology for a continuous dcpo is sober.*

PROOF. Let A be closed irreducible, and consider $B = \bigcup_{a \in A} \Downarrow a$, (cf. definition 5.1.1). We first prove that B is directed. Suppose not: let $d, d' \in B$ such that there exists no $a \in A$ and $d'' \ll a$ such that $d, d' \leq d''$. We claim:

$$\Uparrow d \cap \Uparrow d' \cap A = \emptyset.$$

[1] We refer to section 1.2 for the definition of Alexandrov topology and specialisation ordering.

Indeed, suppose $d, d' \ll a$ for some $a \in A$. Then by directedness of $\Downarrow a$ there would exist $d'' \ll a$ such that $d, d' \leq d''$, contradicting our assumption about d, d'. This proves the claim, which we rephrase as

$$A \subseteq (D \setminus \Uparrow d) \cup (D \setminus \Uparrow d').$$

But this contradicts the irreducibility of A, since $\Uparrow d$ and $\Uparrow d'$ are open (see exercise 5.1.12), and since $d, d' \in B$ can be rephrased as

$$A \cap \Uparrow d \neq \emptyset \text{ and } A \cap \Uparrow d' \neq \emptyset.$$

Hence B is directed. Since closed sets are closed downwards, we have $B \subseteq A$. Hence $\bigvee B \in A$ since A is closed. We show that $\bigvee B$ is an upper bound of A: this follows immediately from the definition of continuous dcpo: if $a \in A$, then $a = \bigvee \Downarrow a \leq \bigvee B$. Therefore $A = \downarrow (\bigvee B) = \overline{\bigvee B}$. □

Scott topologies are not always sober.

Proposition 10.1.18 (Johnstone) *Consider $\omega \cup \{\infty\}$, ordered by: $n \leq n'$ iff $n \leq n'$ in ω or $n' = \infty$. Consider the following partial order on the set $D = \omega \times (\omega \cup \{\infty\})$:*

$$(m, n) \leq (m', n') \text{ iff } (m = m' \text{ and } n \leq n') \text{ or } (n' = \infty \text{ and } n \leq m').$$

This forms a dcpo. Its Scott topology is not sober.

PROOF. We first check that we have a dcpo. We claim that any element (m, ∞) is maximal. Let $(m, \infty) \leq (m', n')$: if $m = m'$ and $\infty \leq n'$, then $\infty = n'$, while the other alternative ($n' = \infty$ and $\infty \leq m'$) cannot arise because m' ranges over ω. In particular, there is no maximum element, since the elements (m, ∞) are comparable only when they are equal.

Let Δ be directed. If it contains some (m, ∞), then it has a maximum. Otherwise let (m', n'), (m'', n'') be two elements of Δ: a common upper bound in Δ can only be of the form (m''', n'''), with $m''' = m' = m''$. Hence $\Delta = \{m\} \times \Delta'$, for some m and some $\Delta' \subseteq_{dir} \omega$. It is then obvious that Δ has a lub.

Next we observe that a non-empty Scott open contains all elements (p, ∞), for p sufficiently large. Indeed, if $(m, n) \in U$, then $p \geq n \Rightarrow (m, n) \leq (p, \infty)$. In particular, any finite intersection of non-empty open sets is non empty. In other words $\emptyset$ is a prime open, or, equivalently, the whole space $\omega \times (\omega \cup \{\infty\})$ is irreducible. By lemma 10.1.16, we should have $D = \downarrow x$ for some x, but we have seen that D has no maximum. □

Nevertheless, sober spaces have something to do with Scott topology.

Proposition 10.1.19 *The specialisation order of any sober space (X, Ω) forms a dcpo, whose Scott topology contains Ω.*

PROOF. Let S be $\leq_\Omega$-directed. We show that its closure $\overline{S}$ is irreducible. Let $\overline{S} = F_1 \cup F_2$, and suppose $S \not\subseteq F_1$ and $S \not\subseteq F_1$. Let $x \in S \backslash F_1$ and $y \in S \backslash F_2$, and let $z \geq_\Omega x, y$ in S. Since $S \subseteq F_1 \cup F_2$, we have, say, $z \in F_1$. Then $x \in F_1$ by definition of $\leq_\Omega$: contradiction. Therefore $\overline{S} = \overline{\{y\}}$ for some y.

- y is an upper bound: Pick $s \in S$ and suppose $y \notin U$. Then $\overline{S} = \overline{\{y\}} \subseteq X \backslash U$, and a fortiori $s \notin U$. Hence $s \leq_\Omega y$. We claim:

$$\text{If } S \cap U = \emptyset, \text{ then } y \notin U.$$

Indeed, $S \cap U = \emptyset$ implies $\overline{S} \cap U = \emptyset$, and a fortiori $y \notin U$. The rest of the statement follows from the claim:

- y is the least upper bound: Let z be an upper bound of S, and suppose $z \notin U$. Then $S \cap U = \emptyset$ by definition of $\leq_\Omega$, and $y \notin U$ follows by the claim.
- Any open is Scott open: By the claim, since we now know that $y = \bigvee S$. □

Exercise 10.1.20 *Show that the statement of proposition 10.1.18 can actually be strengthened by replacing "Its Scott topology is not sober" by: "There is no sober topology whose specialisation order is the order of D". Hint: use proposition 10.1.19.*

10.2 The Duality for Algebraic Dcpo's

We recall that in algebraic dcpo's the basic Scott-open sets have the form $\uparrow a$ (a compact), and have the remarkable property that if $\uparrow a \subseteq \bigcup_i U_i$, then $\uparrow a \subseteq U_i$ for some i. This motivates the following definition.

Definition 10.2.1 (coprime algebraic) *Let $(A, \leq)$ be a partial order. A* compact coprime *is an element a such that, for all $B \subseteq A$, if $\bigvee B$ exists and $a \leq \bigvee B$, then $a \leq b$ for some $b \in B$. A poset $(D, \leq)$ is called coprime algebraic if each element of D is the lub of the compact coprimes it dominates. We write $\mathcal{C}(D)$ for the set of compact coprime elements of D.*

Remark 10.2.2 *In definition 10.2.1, we do not specify under which lub's we assume A to be closed. In this chapter we are concerned with complete lattices, and in chapter* **??**, *we shall have to do with bounded complete coprime algebraic cpo's.*

Lemma 10.2.3 (lower-set completion) *A complete lattice is coprime algebraic iff it is isomorphic to the lower set completion of some partial order $(X, \leq)$, defined as $Dcl(X) = \{Y \subseteq X \mid Y \text{ is a lower subset}\}$. In particular, a coprime algebraic partial order is a (spatial) locale, since $Dcl(X)$ is Alexandrov's topology over $(X \geq)$.*

PROOF. Like the proof of proposition 1.1.21. □

Exercise 10.2.4 *Show that "compact coprime" defined as above is the same as "compact and coprime". Show that a partial order A is coprime algebraic iff it is an algebraic dcpo such that all finite lub's of compact coprime elements exist, and such that every compact is a finite lub of compact coprimes.*

Lemma 10.2.5 *Let A be a coprime algebraic locale. The points of A are in one-to-one correspondence with the filters over the set $\mathcal{C}(A)$ of compact coprimes of A.*

PROOF. The inverses are $G \mapsto \uparrow G$ and $F \mapsto \{x \in F \mid x \text{ compact coprime}\}$. □

Proposition 10.2.6 (duality – algebraic dcpo's) *The basic duality cuts down to an equivalence between the category* **Adcpo** *of algebraic dcpo's and continuous functions, and the category of locales arising as lower set completions of some partial order.*

PROOF. By lemma 10.2.3, any coprime algebraic locale is spatial. By proposition 1.1.21, any algebraic dcpo is isomorphic to $Ide(X)$ for some partial order $(X, \leq)$. By lemma 10.2.5 we have

$$Ide(X) = \mathcal{F}(X^{op}) = Pt(Dcl(X^{op})).$$

(We omit the proof that the topology induced by Pt on $Ide(X)$ is Scott topology.) Therefore, up to isomorphism, the class of algebraic dcpo's is the image under Pt of the class of coprime algebraic locales. The statement then follows (cf. exercise B.6.5). □

We call the duality algebraic dcpo's / coprime algebraic locales the *basic domain duality*. The key to this duality is that both terms of the duality have a common reference, namely the set $\mathcal{C}(A)$ of compact coprimes on the localic side, the set $\mathcal{K}(D)$ of compacts on the spatial side, with opposite orders:

$$(\mathcal{K}(D), \leq) \cong (\mathcal{C}(A), \geq).$$

We shall obtain other dualities for various kinds of domains as restriction of the basic domain duality, through the following metalemma.

Lemma 10.2.7 *If (S) is a property of algebraic dcpo's and (L) is a property of locales such that any algebraic dcpo satisfies (S) iff its Scott topology satisfies (L), then the basic domain duality cuts down to a duality between the category of algebraic dcpo's satisfying (S) and the category of coprime algebraic locales satisfying (L).*

PROOF. Cf. exercise B.6.5. □

Here are two examples of the use of lemma 10.2.7.

Proposition 10.2.8 (duality – algebraic cpo's) *The basic domain duality restricts to an equivalence between the category* **Acpo** *of algebraic cpo's and continuous functions on one side, and the category of locales arising as lower set completions of a partial order having a largest element.*

PROOF. By lemma 10.2.7, with "has a minimum element" for (S), and "has a maximum compact coprime" for (L). If D satisfies (S), then $\uparrow \perp$ fits. If $\uparrow x$ is the maximum compact coprime of ΩD, then $\uparrow y \subseteq \uparrow x$ for any other compact coprime, i.e., x is minimum. □

Proposition 10.2.9 (duality – Scott domains) *The basic domain duality cuts down to an equivalence between the category of Scott domains (cf. definition 1.4.9) and continuous functions on one side, and the category of locales arising as lower set completions of a conditional lower semi-lattice (i.e., a poset for which every finite lower bounded subset has a glb).*

PROOF. Take "has a minimum element, and binary compatible lub's" as (S), and "compact coprimes form a conditional lower semi-lattice" as (L), and notice that $x \vee y$ exists iff $\uparrow x, \uparrow y$ have a glb. □

We can use exercise 10.2.4 to get an alternative description of the posets arising as lower set completions of conditional lower semi-lattices:

Proposition 10.2.10 *The following conditions are equivalent for a coprime algebraic partial order A:*

1. *A is isomorphic to the lower set completion of a conditional lower semi-lattice.*
2. *Finite glb's of compacts are compact and any finite glb of compact coprimes is coprime or $\perp$.*

PROOF. By proposition 10.2.3, (1) can be replaced by:

1′. *The compact coprimes of A form a conditional lower semi-lattice.*

Also, since all lub's exist in A, glb's also exist and are defined by

$$\bigvee\{x \mid \forall p \in P \;\; x \leq p\} = \bigvee\{q \mid q \text{ compact coprime and } \forall p \in P \;\; q \leq p\}.$$

Now, consider a finite set P of compact coprimes. There are two cases:

P has no compact coprime lower bound: then $\bigwedge P = \bigvee \emptyset = \perp$.
P has a compact coprime lower bound: then $\bigwedge P \neq \perp$.

(1′) ⇒ (2) We already know that A is a locale; a fortiori it is distributive. Let $d = a_1 \vee \cdots \vee a_m$ and $e = b_1 \vee \cdots \vee b_n$ be two compacts. Then $d \wedge e = \bigvee_{i,j}(a_i \wedge b_j)$.

It is enough to check that each $a_i \wedge b_j$ is compact. If $\{a_i, b_j\}$ has no compact coprime lower bound, then $a_1 \wedge b_j = \bot$, which is compact. Otherwise, $a_i \wedge b_j$ is compact by assumption.

(2) $\Rightarrow$ (1′) We have to prove that, in case (b), $\bigwedge P$ is compact and coprime. It is compact by the first part of the assumption. By the second part of the assumption, $\bigwedge P$ is either coprime or $\bot$. Since (b) implies $\bigwedge P \neq \bot$, $\bigwedge P$ is coprime. □

Information Systems An alternative description of Scott domains is obtained by starting, not from a conditional upper semi-lattice, but from a partial order equipped with a weaker structure, which we first motivate. Let A be a conditional upper semi-lattice. Let $I_1, I_2, I \in Ide(A)$ be such that $I_1, I_2 \subseteq I$. Then the lub of I_1, I_2 is given by the following formula:

$$I_1 \vee I_2 = \{a \mid a \leq \bigvee X \text{ for some } X \subseteq_{fin} I_1 \cup I_2\}.$$

This suggests us to consider the collection of the finite bounded subsets X of A, and the pairs (X, a) with the property $a \leq \bigvee X$. It turns out that we actually do not need to be so specific about this structure. It is enough to have a distinguished collection *Con* of finite subsets over which X ranges, and an "entailment" relation $\vdash$ consisting of pairs (X, a). This view leads us to Scott's notion of information system [Sco82], whose axioms we shall discover progressively.

Given a partial order A of tokens, a subset *Con* of finite subsets of A, and a relation $\vdash \subseteq Con \times A$, we construct a "completion" whose elements are the non-empty subsets $x \in A$ which satisfy:

1. $X \subseteq_{fin} x \Rightarrow X \in Con$,
2. $(X \subseteq_{fin} x \text{ and } X \vdash a) \Rightarrow a \in x$.

If A is a conditional upper semi-lattice, if *Con* is the boundedness predicate and $X \vdash a$ is defined by $a \leq \bigvee X$, then it is easily checked that conditions (1) and (2) together characterise the ideals of A (notice that (1) is weaker than directedness, and (2) is stronger than downward closedness). A directed union Δ of elements is an element: if $X \subseteq_{fin} \bigcup \Delta$, then by directedness $X \subseteq_{fin} x$ for some $x \in \Delta$, and (1) and (2) for $\bigcup \Delta$ follow from (1) and (2) applied to x.

Candidates for the compact elements are the elements of the form $\overline{X} = \{a \mid X \vdash a\}$. The sets $\overline{X}$ are not necessarily finite, but can be considered finitely generated from X. We expect that$X \subseteq \overline{X}$ and that $\overline{X}$ is an element (which by construction is the smallest containing X). This is easily proved thanks to the following axioms:

(A) $(X \in Con \text{ and } a \in X) \Rightarrow X \vdash a$,
(B) $X \subseteq Y \in Con \Rightarrow X \in Con$,
(C) $X \vdash a \Rightarrow X \cup \{a\} \in Con$,
(D) $(\{a_1, \ldots, a_n\} \vdash a \text{ and } X \vdash a_1, \ldots, X \vdash a_n) \Rightarrow X \vdash a$.

Axiom (D) is exactly condition (2) for $\overline{X}$. As for (1), we check, say, that if $a_1, a_2 \in \overline{X}$, then $\{a_1, a_2\} \in Con$. First, an easy consequence of (A) and (D) is:

$$(X \subseteq Y \in Con \text{ and } X \vdash a) \Rightarrow Y \vdash a.$$

Applying this to X, $X \cup \{a_1\}$ (which is in Con by (C)) and a_2, we obtain $X \cup \{a_1\} \vdash a_2$, and deduce $\{a_1, a_2\} \in Con$ by (A) and (B).

The elements $\overline{X}$ form a basis: Consider an element x and $\{\overline{X} \mid \overline{X} \subseteq x\} = \{\overline{X} \mid X \subseteq x\}$. This set is directed, since if $X_1, X_2 \subseteq x$, then $X_1 \cup X_2 \subseteq x$ and $\overline{X_1}, \overline{X_2} \subseteq \overline{X_1 \cup X_2}$. Its union is x thanks to the following axiom:

$$\text{(E)} \quad \forall a \in A \ \ \{a\} \in Con.$$

We are left to show that $\overline{X}$ is compact. This follows easily from: $\overline{X} \subseteq x$ iff $X \subseteq x$. Finally, we address bounded completeness. If $x_1, x_2 \subseteq x$, then

$$x_1 \vee x_2 = \{a \mid \exists\, X \subseteq x_1 \cup x_2 \ \ X \vdash a\}.$$

Definition 10.2.11 *We call information system a structure $(A, Con, \vdash)$ satisfying the above axioms (A)-(E), and we write $D(A, Con, \vdash)$ for the set of its elements ordered by inclusion.*

Theorem 10.2.12 *The class of all bounded complete algebraic dcpo's is the class of partial orders which are isomorphic to $D(A, Con, \vdash)$, for some information system.*

PROOF. We have done most of the work to establish that $D(A, Con, \vdash)$ is algebraic and bounded complete. Conversely, given D, we take the "intended" interpretation discussed above: $A = \mathcal{K}(D)$, $X \in Con$ iff X has an upper bound, and $X \vdash d$ iff $d \leq \bigvee X$. □

Theorem 10.2.12 is an example of a representation theorem, relating abstract order-theoretic structures (Scott domains) with more concrete ones (information systems). Event structures, concrete data structures, considered in sections 12.3, 14.2, respectively, will provide other examples of representation theorems.

Exercise 10.2.13 *In our treatment, we have not included the axiomatisation of the minimum element. Show that this can be done by means a special token (which Scott has called Δ).*

Information systems allow for an attractive characterisation of injection-projection pairs, in the line of proposition 3.1.4 and remark 3.1.5.

Exercise 10.2.14 *Show that, for any two bounded complete algebraic dcpo's D, D', there exists an injection-projection pair between D and D' iff there exist two information systems $(A, Con, \vdash)$ and $(A', Con', \vdash')$ representing D and D' (i.e., such that D, D' are isomorphic to $D(A, Con, \vdash)$, $D(A', Con', \vdash')$, respectively), and such that*

$$A \subseteq A' \quad Con = Con' \cap A \quad \vdash = \vdash' \cap (Con \times A).$$

10.3 Stone Spaces *

In this section we focus on algebraicity on the localic side, and prove Stone's theorem. The "algebraic cpo line" (section 10.2) and the "algebraic locale line" will be related when addressing Stone duality for bifinite domains (section 10.4).

Proposition 10.3.1 *Algebraic locales, i.e., locales which viewed as cpo's are algebraic, are spatial.*

PROOF. Let $a \not\leq b$. By theorem 10.1.13, it is enough to find a Scott-open filter F such that $a \in F$ and $b \notin F$. By algebraicity, we can find a compact d such that $d \leq a$ and $d \not\leq b$. Then the filter $F = \uparrow d$ is Scott-open and fits. □

Proposition 10.3.2 *1. The ideal completions of sup-semi-lattices are the algebraic complete lattices (i.e., the algebraic cpo's with all lub's).*

2. The ideal completions of lattices are the algebraic complete lattices whose compact elements are closed under finite glb's.

3. The ideal completions of distributive lattices are the algebraic locales whose compact elements are closed under finite glb's.

PROOF. (1) Let A be an algebraic complete lattice. Then $\mathcal{K}(A)$ is a sup- semi-lattice, since the lub of a finite set of compacts is always compact. Conversely, it is enough to define binary lub's, since the existence of directed and binary lub's implies the existence of all lub's. Define $a \vee b = \bigvee\{d \vee e \mid d, e \in \mathcal{K}(A), d \leq a \text{ and } e \leq b\}$.

(2) Obvious.

(3) Let X be a distributive lattice. We show that $A = Ide(X)$ is distributive. We have, for ideals:

$$\begin{aligned} I \wedge J &= \{z \mid \exists\, a \in I, b \in J \;\; z \leq a \wedge b\} \\ \bigvee_{i \in I} I_i &= \{z \mid \exists\, i_1, \ldots, i_n, a_1 \in I_1, \ldots, a_n \in I_n \;\; z \leq a_1 \vee \cdots \vee a_n\}\ . \end{aligned}$$

Hence, if $z \in J \wedge (\bigvee_{i \in I} I_i)$, then $z \leq a \wedge (a_1 \vee \cdots \vee a_n)$ for some $a \in J$ and $a_1 \in I_{i_1}, \ldots, a_n \in I_{i_n}$, hence $z \leq (a \wedge a_1) \vee \cdots \vee (a \wedge a_n)$ by distributivity of X, and $z \in \bigvee_{i \in I}(J \wedge I_i)$. □

Definition 10.3.3 (coherent locale) *Locales arising as ideal completions of distributive lattices are called coherent (or spectral). A topological space is called coherent if its topology is coherent.*

In particular, coherent topological spaces are compact (the total space is the empty glb).

Proposition 10.3.4 (duality – coherent) *The basic duality cuts down to an equivalence between the category of coherent topological spaces and the category of coherent locales.*

PROOF. Coherent locales are a fortiori algebraic locales, hence they are spatial by proposition 10.3.1. The statement follows (cf. exercise B.6.5). □

It is then possible to combine the correspondences:

$$\text{coherent spaces} \leftrightarrow \text{coherent locales} \leftrightarrow \text{distributive lattices.}$$

However, these correspondences do not extend to dualities of the respective "natural" categories of continous functions, locale morphisms, and $\mathbf{DLat}^{op}$ morphisms, where **DLat** is the category of distributive lattices and lattice morphisms. The reason is that a locale morphism does not map compact elements to compact elements in general. However, this will be true of Stone spaces.

As for coprime algebraic locales, the points of a coherent locale enjoy a simple characterisation.

Lemma 10.3.5 *Let A be a coherent locale. Then the points of A are in one-to-one correspondence with the coprime filters over $\mathcal{K}(A)$:*

$$Spec(\mathcal{K}(A)) = Pt(A).$$

PROOF. The inverse mappings are: $G \mapsto \uparrow G$ and $F \mapsto \{x \in F \mid x \text{ compact}\}$. We check only that $\uparrow G$ is coprime. Let $x \vee y \in \uparrow G$. Let $g \in G$ be such that $g \leq x \vee y$. Since G is completely coprime, we may assume that g is compact. Since A is an algebraic lattice, we can write

$$x \vee y = (\bigvee\{d \mid d \leq x\}) \vee (\bigvee\{e \mid e \leq x\}) = \bigvee\{d \vee e \mid d \leq x, e \leq x\}.$$

By compactness, $g \leq d \vee e$, for some $d, e \in \mathcal{K}(A)$. Hence $d \vee e \in G$, and we have $d \in G$ or $e \in G$, since G is prime, implying $x \in \uparrow G$ or $y \in \uparrow G$. □

Remark 10.3.6 *The following table should help to compare lemmas 10.2.5 and 10.3.5:*

$\mathcal{F}(\mathcal{C}(A)) \cong Pt(A)$	*filter*	*lower set completion*
$Spec(\mathcal{K}(A)) \cong Pt(A)$	*prime filter*	*ideal completion* .

We move on to Stone spaces. There are several equivalent definitions of Stone spaces [Joh82]. We choose the one which serves to prove the duality.

Definition 10.3.7 (Stone space) *A Stone space is a T_2-space whose topology is coherent.*

Proposition 10.3.8 (duality – Stone) *The Stone spaces are the topological spaces whose topology is isomorphic to the ideal completion of a Boolean algebra. The three following categories are equivalent:*

1. *Stone spaces and continuous functions,*
2. *The category of locales arising as ideal completions of Boolean algebras,*

3. **Bool**^{op}, *where* **Bool** *is the category of Boolean algebras and lattice morphisms, that is, the functions preserving finite glb's and lub's.*

PROOF. Let $(X, \Omega X)$ be a Stone space. We show that $\mathcal{K}(\Omega X)$ is Boolean. In compact T_2-spaces, compact subsets are closed, and hence the compact subsets are the closed subsets. Hence the compact open subsets are the closed open subsets, which are closed under set-theoretic complementation.

Conversely, let B be a Boolean algebra, and consider two distinct coprime filters G_1, G_2 of B. Combining proposition 10.1.8 and lemma 10.3.5, the opens of $Pt(Ide(B))$ are the sets $U_b = \{G \mid b \in \uparrow G\}$. We look for b_1, b_2 in B such that $G_1 \in U_{b_1}$, $G_2 \in U_{b_2}$ and $U_{b_1} \cap U_{b_2} = \emptyset$.

• $G_1 \in U_{b_1}$, $G_2 \in U_{b_2}$: Since $G_1 \neq G_2$, we can pick, say, $b_1 \in G_1 \backslash G_2$. We have, setting $b_2 = \neg b_1$:

$$\begin{array}{rcll} b_1 \vee b_2 = 1 & \Rightarrow & b_1 \vee b_2 \in G_2 & (G_2 \text{ filter}) \\ & \Rightarrow & b_1 \in G_2 \text{ or } b_2 \in G_2 & (\text{coprimeness}) \\ & \Rightarrow & b_2 \in G_2 & (b_1 \notin G_2)\ . \end{array}$$

A fortiori, $b_1 \in G_1, b_2 \in G_2$ imply $G_1 \in U_{b_1}$, $G_2 \in U_{b_2}$.

• $U_{b_1} \cap U_{b_2} = \emptyset$: Suppose not, and let G be such that $b_1 \in \uparrow G$ and $b_2 \in \uparrow G$. We have:

$$\begin{array}{rcll} b_1 \wedge b_2 = 0 & \Rightarrow & 0 \in \uparrow G & (\uparrow G \text{ is a filter}) \\ & \Rightarrow & 0 \in G & (\text{definition of } \uparrow G) \\ & \Rightarrow & G = B & (G \text{ filter})\ . \end{array}$$

But $G = B$ contradicts the primeness of G.

The categories (1) and (2) are equivalent by restriction of the basic duality. The equivalence between categories (2) and (3) follows from the following claim: the morphisms of category (2) map compact elements to compact elements. The claim is proved easily by taking advantage of spatiality and of the duality (1)/(2). We have seen that the compact opens are the closed opens. The claim then follows from the observation that for any continuous function f in **Top**, f^{-1} maps closed subsets to closed subsets. □

10.4 Stone Duality for Bifinite Domains *

In order to relate propositions 10.2.6 and 10.3.4, we have to understand under which conditions the Scott topology of an algebraic dcpo is coherent.

Proposition 10.4.1 *An algebraic dcpo D is coherent as a topological space iff it has a minimum element and its basis satisfies property M (cf. definition 5.2.6).*

PROOF. Recall from the proof of proposition 10.2.6 that the compact coprimes of the Scott topology Ω of D are the $\uparrow d$'s ($d \in \mathcal{K}(D)$). Therefore, the compacts of Ω are the finite unions $\uparrow d_1 \cup \cdots \cup \uparrow d_m$.

• $M \Rightarrow$ coherent: We have to check that the compacts of Ω are closed under finite intersections. For the empty intersection, take $\uparrow \bot$. For the binary intersection, observe

that $(\uparrow d_1 \cup \cdots \cup \uparrow d_m) \cap (\uparrow e_1 \cup \cdots \cup \uparrow e_n)$ is a union of sets $\uparrow d_i \cap \uparrow e_j$, which can be written as

$$\begin{aligned} \uparrow d_i \cap \uparrow e_j &= UB(d_i, e_j) \\ &= \uparrow x_1 \cup \cdots \cup \uparrow x_p \quad (\text{where } MUB(d_i, e_j) = \{x_1, \ldots, x_p\}) \ . \end{aligned}$$

- coherent $\Rightarrow M$: Let $Y = \{y_1, \ldots, y_q\}$ be a finite subset of $\mathcal{K}(D)$. Then

$$\begin{aligned} UB(Y) &= \uparrow y_1 \cap \cdots \cap \uparrow y_q \\ &= \uparrow x_1 \cup \cdots \cup \uparrow x_p \quad (\text{for some } x_1, \ldots, x_p, \text{ by coherence}) \ . \end{aligned}$$

□

Definition 10.4.2 (coherent algebraising) *A coherent locale is called coherent algebraising* [2] *if it coprime algebraic.*

Thus "coherent algebraising" means "coherent + coprime algebraic". The following proposition provides an alternative definition of coherent algebraising locale.

Proposition 10.4.3 *A bounded complete algebraic cpo D is coprime algebraic iff it satisfies the following decomposition property:*

every compact $d \neq \bot$ is a finite lub of compact coprimes.

PROOF. ($\Rightarrow$) Let d be compact, and let $X = \{e \leq d \mid e \text{ compact coprime}\}$. We have:

$$\begin{array}{ll} d = \bigvee X & (\text{by coprime algebraicity}) \\ d \leq e_1 \vee \cdots \vee e_n \text{ for some } e_1, \ldots, e_n \in X & (\text{by bounded completeness and algebraicity}) \ . \end{array}$$

Hence $d = e_1 \vee \cdots \vee e_n$.

($\Leftarrow$) Putting together the algebraicity and the decomposition property, we have for any $x \in D$:

$$\begin{aligned} x &= \bigvee\{d \leq x \mid d \text{ compact}\} \\ &= \bigvee\{e_1 \vee \cdots \vee e_n \mid e_1, \ldots, e_n \text{ compact coprime and } e_1 \vee \cdots \vee e_n \leq x\} \\ &= \bigvee\{e \leq x \mid e \text{ compact coprime}\} \ . \end{aligned}$$

□

Proposition 10.4.4 (duality – algebraic + M) *The basic domain duality cuts down to an equivalence between the category of algebraic cpo's whose basis satisfies M and the category of coherent algebraising locales.*

PROOF. The statement follows from proposition 10.4.1: we apply lemma 10.2.7 with "has a minimum element and the basis satisfies M" for (S), and "coherent" for (L).□

The following lemma indicates the way to axiomatise bifinite domains logically.

[2] A more standard terminology for this is "coherent algebraic". We prefer to use "algebraising", to stress that one refers to the algebraicity of the Stone dual cpo, rather than to the algebraicity of the locale (which is also relevant and is part of the definition of "coherent").

Lemma 10.4.5 *Let D be an algebraic cpo. The following properties hold:*

1. $\mathcal{K}(D)$ satisfies M iff $(\forall X \subseteq_{fin} \mathcal{K}(D)\ \exists Y \subseteq_{fin} \mathcal{K}(D)\ \bigcap_{x\in X}(\uparrow x) = \bigcup_{y\in Y}(\uparrow y))$.

2. D is bifinite iff

$$\begin{array}{l}\forall X \subseteq_{fin} \mathcal{K}(D)\ \exists Y \subseteq_{fin} \mathcal{K}(D)\\ X \subseteq Y \text{ and } (\forall Z_1 \subseteq Y\ \exists Z_2 \subseteq Y\ \bigcap_{x\in Z_1}(\uparrow x) = \bigcup_{y\in Z_2}(\uparrow y))\ .\end{array}$$

Proof. (1) follows from the following claim, for $X \subseteq_{fin} \mathcal{K}(D)$:

$$\exists Y \subseteq_{fin} \mathcal{K}(D)\ \bigcap_{x\in X}(\uparrow x) = \bigcup_{y\in Y}(\uparrow y) \Leftrightarrow MUB(X) \subseteq Y \text{ and } MUB(X) \text{ is complete.}$$

($\Rightarrow$) It is easy to check that $MUB(X)$ is the set of minimal elements of Y.

($\Leftarrow$) Take $Y = MUB(X)$.

(2) By the claim, the equivalence can be rephrased as: D is bifinite iff

$$\begin{array}{l}\forall X \subseteq_{fin} \mathcal{K}(D)\ \exists Y \subseteq_{fin} \mathcal{K}(D)\\ X \subseteq Y \text{ and } (\forall Z \subseteq Y\ MUB(Z) \subseteq Y \text{ and } MUB(Z) \text{ is complete})\ .\end{array}$$

and by definition, D bifinite means: $\forall X \subseteq_{fin} \mathcal{K}(D)\ U^\infty(X)$ is finite.

($\Rightarrow$) Take $Y = U^\infty(X)$.

($\Leftarrow$) By induction on n, we obtain $U^n(X) \subseteq Y$ for all n, and since Y is finite the sequence $\{U^n(X)\}_{n\geq 0}$ becomes stationary. □

Proposition 10.4.6 (duality – bifinite) *The basic domain duality cuts down to an equivalence between the category of bifinite cpo's and the category of coherent algebraising locales A satisfying the following property:*

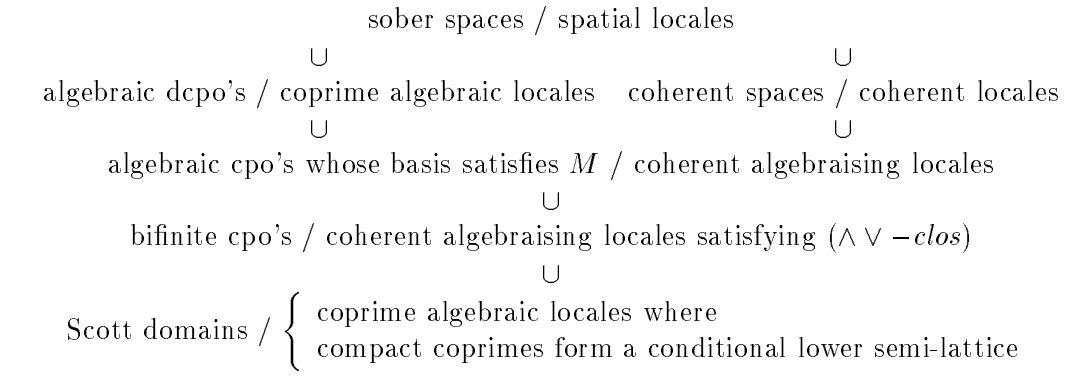
$$(\wedge\vee - clos)\quad \left\{\begin{array}{l}\forall X \subseteq_{fin} \mathcal{C}(A)\ \exists Y \subseteq_{fin} \mathcal{C}(A)\\ X \subseteq Y \text{ and } (\forall Z \subseteq Y\ \exists Z_1 \subseteq Y\ \bigwedge Z = \bigvee Z_1)\ .\end{array}\right.$$

Proof. The statement follows from lemma 10.4.5: we apply lemma 10.2.7 with "bifinite" for (S), and the property of the statement as (L). □

In figure 10.1, we summarise the dualities for domains that we have proved in this chapter.

10.5 Scott Domains in Logical Form *

We present domains via their compact open sets, constructed as (equivalence classes of) formulas. Abramsky has called this "domains in logical form". As a first step in this direction, we show how to present the compact opens of $D \to E$ in terms of the compact opens of D and E. When we write ΩD, we mean the Scott topology on D.



with:

$$\text{coprime algebraic} \quad = \quad \text{lower set completion of a partial order}$$

$$\begin{aligned} \text{coherent} \quad &= \quad \text{ideal completion of a distributive lattice} \\ &= \quad \left\{ \begin{array}{l} \text{algebraic} + \\ \text{closure of compacts under finite glb's} \end{array} \right. \end{aligned}$$

$$\begin{aligned} \text{coherent algebraising} \quad &= \quad \left\{ \begin{array}{l} \text{ideal completion of a distributive lattice} + \\ \text{every compact is a finite lub of compact coprimes} \end{array} \right. \\ &= \quad \left\{ \begin{array}{l} \text{coprime algebraic} + \\ \text{closure of compacts under finite glb's} \end{array} \right. \end{aligned}$$

$$\left. \begin{array}{r} \text{coprime algebraic} \; + \; \text{compact coprimes} \\ \text{form a conditional lower semi-lattice} \end{array} \right\} \quad = \quad \left\{ \begin{array}{l} \text{coherent algebraising} \; + \; \text{glb's of} \\ \text{compact coprimes are coprime or } \bot \end{array} \right.$$

Figure 10.1: Summary of dualities

Proposition 10.5.1 *If D and E are algebraic cpo's such that $D \to E$ is algebraic and its basis satisfies M, then:*

1. *For any $U \in \mathcal{K}(\Omega D)$ and $V \in \mathcal{K}(\Omega E)$, the following set is compact:*

$$U \to V = \{f : D \to E \mid f(U) \subseteq V\}.$$

2. *Any compact open of $D \to E$ is a finite union of finite intersections of such sets $U \to V$, where U, V are moreover coprime.*

PROOF. (1) Let $U = \uparrow d_1 \cup \cdots \cup \uparrow d_m$ and $V = \uparrow e_1 \cup \cdots \cup e_n$. The definition of $U \to V$ can be reformulated as

$$\begin{aligned} U \to V &= \{f : D \to E \mid \forall\, i\ \exists\, j\ \ f(d_i) \geq e_j\} \\ &= \bigcap_i (\bigcup_j (\uparrow (d_i \to e_j))) \ . \end{aligned}$$

Each $\uparrow (d_i \to e_j)$ is compact, therefore each $\bigcup_j (\uparrow (d_i \to e_j))$ is compact; the conclusion follows by propositions 10.4.1 and 10.3.2.

(2) The compact opens of $D \to E$ are the subsets of the form $\uparrow f^1 \cup \cdots \cup \uparrow f^p$ where each f^i is compact, hence is of the form $(d^1 \to e^1) \vee \cdots \vee (d^q \to e^q)$, that is:

$$\uparrow f^i = \uparrow (d^1 \to e^1) \cap \cdots \cap (d^q \to e^q).$$

Then the conclusion follows from the observation that $\uparrow d \to \uparrow e = \uparrow (d \to e)$, for any compact d, e. □

A second step consists in constructing a logical theory based on these sets $U \to V$, now considered as (atomic) formulas. We seek a complete theory in the sense that if two different formulas u, v present two opens $[\![u]\!], [\![v]\!]$ such that $[\![u]\!] \subseteq [\![v]\!]$, then $u \leq v$ is provable.

Proposition 10.5.2 *Let D, E be Scott domains. Then the set of compact opens of $D \to E$ is order-isomorphic to the partial order associated to a preorder Φ defined as follows. The elements of Φ are formulas defined by:*

$$\frac{U \in \mathcal{K}(\Omega D)\ V \in \mathcal{K}(\Omega E)}{U \to V \in \Phi} \qquad \frac{\forall\, i \in I\ \ u_i \in \Phi\ (I \text{ finite})}{\bigwedge_{i \in I} u_i \in \Phi} \qquad \frac{\forall\, i \in I\ \ u_i \in \Phi\ (I \text{ finite})}{\bigvee_{i \in I} u_i \in \Phi}$$

and the preorder is the least preorder closed under the following rules (where $=$ stands for the equivalence associated with the preorder):

$$u \le u \qquad \frac{u \le v \quad v \le w}{u \le w} \qquad u \wedge (v \vee w) \le (u \wedge v) \vee (u \wedge w)$$

$$\frac{\forall\, i \in I \;\; u_i \le v \quad (I \textit{ finite})}{\bigvee_{i\in I} u_i \le v} \qquad u_i \le \bigvee_{i\in I} u_i$$

$$\frac{\forall\, i \in I \;\; u \le v_i \quad (I \textit{ finite})}{u \le \bigwedge_{i\in I} v_i} \qquad \bigwedge_{i\in I} v_i \le v_i$$

$$\frac{U' \subseteq U \quad V \subseteq V'}{U' \to V \le U \to V} \qquad U \to (\textstyle\bigcap_{i\in I} V_i) = \bigwedge_{i\in I}(U \to V_i)$$

$$(\textstyle\bigcup_{i\in I} U_i) \to V = \bigwedge_{i\in I}(U_i \to V) \qquad \frac{U \textit{ coprime}}{U \to (\bigcup_{i\in I} V_i) = \bigvee_{i\in I}(U \to V_i)}$$

Proof. Proposition 10.5.1 gives us a surjection $[\![\_]\!]$ from Φ to $\Omega D \to E$ (interpreting $\wedge, \vee$ as $\cap, \cup$). The soundness of the rules defining the preorder is easy to check, and implies that the surjection is monotonic. All what we have to do is to prove completeness: if $[\![u]\!] \le [\![v]\!]$, then $u \le v$ is provable. By proposition 10.5.1, each formula u is provably equal to a finite disjunction of formulas of the form $\bigwedge_{i\in I}(U_i \to V_i)$, with the U_i's and the V_i's coprime. We know from proposition 10.2.10 that $[\![\bigwedge_{i\in I}(U_i \to V_i)]\!] = \bigcap_{i\in I}(U_i \to V_i)$ is either coprime or $\emptyset$. The proof goes through two claims.

Claim 1. If the U_i's and V_i's are coprime and $\bigcap_{i\in I}(U_i \to V_i) = \emptyset$, then $\bigwedge_{i\in I}(U_i \to V_i) = 0\ (= \bigvee \emptyset)$ is provable.

Since U_i, V_i are coprime, we can write $U_i = {\uparrow}\, d_i$, $V_i = {\uparrow}\, e_i$, and $U_i \to V_i = {\uparrow}\,(d_i \to e_i)$. Therefore, $\bigwedge_{i\in I}(U_i \to V_i) \neq \emptyset$ iff $\{d_i \to e_i \mid i \in I\}$ has an upper bound iff $\{d_i \to e_i \mid i \in I\}$ has a lub iff

$$\forall\, J \subseteq I \;\; \{d_j \mid j \in J\} \text{ has an upper bound} \Rightarrow \{e_j \mid j \in J\} \text{ has an upper bound.}$$

Hence $\bigcap_{i\in I}(U_i \to V_i) = \emptyset$ iff there exists $J \subseteq I$ such that $\bigcap_{j\in J} U_j$ (hence is coprime, by proposition 10.2.10) and $\bigcap_{j\in J} V_j = \emptyset$. Now the subclaim is proved as follows:

$$\bigwedge_{i\in I}(U_i \to V_i) \le \bigwedge_{j\in J}(U_j \to V_j) \le \bigwedge_{j\in J}((\bigcap_{j\in J} U_j) \to V_j) = (\bigcap_{j\in J} U_j) \to (\bigcap_{j\in J} V_j).$$

The last formula can be written $(\bigcap_{j\in J} U_j) \to (\bigcup \emptyset)$, and since $\bigcap_{j\in J} U_j$ is coprime, we have

$$(\bigcap_{j\in J} U_j) \to (\bigcap_{j\in J} V_j) = \bigvee \emptyset.$$

By the subclaim, we can eliminate the conjunctions $\bigwedge_{i\in I}(U_i \to V_i)$ such that $\bigcap_{i\in I}(U_i \to V_i) = \emptyset$. Call u', v' the resulting $u' = u'_1 \vee \cdots \vee u'_m = u$ and $v' = v'_1 \vee \cdots \vee v'_n = v$. Then we can write

$$[\![u'_1]\!] = {\uparrow}\, f_1, \ldots, [\![u'_m]\!] = {\uparrow}\, f_m$$
$$[\![v'_1]\!] = {\uparrow}\, g_1, \ldots, [\![v'_n]\!] = {\uparrow}\, g_n$$

and

$$\begin{array}{lllll} [\![u]\!] \leq [\![v]\!] & \Leftrightarrow & [\![u']\!] \leq [\![v']\!] & \Leftrightarrow & \forall p \ \uparrow f_p \subseteq [\![v']\!] \\ & \Leftrightarrow & \forall p \ f_p \in [\![v']\!] & \Leftrightarrow & \forall p \ \exists q \ f_p \geq g_q \quad \Leftrightarrow \quad \forall p \ \exists q \ [\![u'_p]\!] \leq [\![v'_q]\!] \end{array}$$

which brings us to the following second claim.

Claim 2 (coprime completeness). If u, v both have the form $\bigwedge_{i\in I}(U_i \to V_i)$, with the U_i's and V_i's coprime, if $[\![u]\!] \leq [\![v]\!]$, and if $[\![u']\!], [\![v']\!]$ are coprime, then $u \leq v$ is provable.

By the definition of $\bigwedge$, we can assume that v is reduced to one conjunct: $v = U \to V = \uparrow (d \to e)$. Then, setting $U_i = \uparrow d_i$ and $V_i = \uparrow e_i$ for all i, the assumption $[\![u]\!] \leq [\![v]\!]$ reads as $d \to e \leq \bigvee_{i\in I}(d_i \to e_i)$, or, equivalently:

$$e \leq (\bigvee_{i\in I}(d_i \to e_i))(d) = \bigvee\{e_j \mid d_j \leq d\}.$$

Setting $J = \{j \mid d_j \leq d\}$, we have: $U \subseteq \bigcap_{j\in J} U_j$ and $\bigcap_{j\in J} V_j \subseteq V$. Then

$$\bigwedge_{i\in I}(U_i \to V_i) \leq (\bigcap_{j\in J} U_j) \to (\bigcap_{j\in J} V_j) \leq U \to V.$$

We now complete the proof of the completeness claim:

$$\begin{array}{lllll} [\![u]\!] \leq [\![v]\!] & \Leftrightarrow & \forall p \ \exists q \ [\![u'_p]\!] \leq [\![v'_q]\!] & \Leftrightarrow & \forall p \ \exists q \ u'_p \leq v'_q \\ & \Leftrightarrow & u' = \bigvee_{k=1,\ldots,m} u'_k \leq \bigvee_{l=1,\ldots,n} u'_l = v' & \Rightarrow & u = u' \leq v' = v \ . \end{array}$$

□

The last step consists in further "syntaxizing" domains, by defining a language of formulas, not only for $\mathcal{K}(\Omega(D \to E))$, but also for $\mathcal{K}(\Omega D)$, $\mathcal{K}(\Omega E)$, and more generally for all types. Since the axioms used to describe $\mathcal{K}(\Omega(D \to E))$ involve coprimeness at the lower types, the coprimeness predicate has to be axiomatised as well.

Definition 10.5.3 *Let $\{\kappa_1, \ldots, \kappa_n\}$ be a fixed collection of basic types, and let $D_1, \ldots, D_n$ be fixed Scott domains associated with $\kappa_1, \ldots, \kappa_n$. Consider:*

- *The following collection of types:*

$$\sigma ::= \kappa_i \ (i = 1, \ldots, n) \mid \sigma \to \sigma.$$

- *The formal system for deriving typed formulas given in figure 10.2. We write $\bigwedge \emptyset = 1$ and $\bigvee \emptyset = 0$.*

- *The formal system for deriving two kinds of judgements*

$$\begin{array}{ll} u \leq v & (\textit{with } u = v \textit{ standing for}: \ (u \leq v \textit{ and } v \leq u)) \\ C(u) & (\textit{"}u \textit{ is coprime"}) \end{array}$$

given in figure 10.3.

$$\frac{U \in \mathcal{K}(\Omega D_i)}{U \in \Phi(\kappa_i)} \qquad \frac{u \in \Phi(\sigma) \quad v \in \Phi(\tau)}{u \to v \in \Phi(\sigma \to \tau)}$$

$$\frac{\forall\, i \in I \;\; u_i \in \Phi(\sigma) \quad (I \textit{ finite})}{\bigwedge_{i\in I} u_i \in \Phi(\sigma)} \qquad \frac{\forall\, i \in I \;\; u_i \in \Phi(\sigma) \quad (I \textit{ finite})}{\bigvee_{i\in I} u_i \in \Phi(\sigma)}$$

Figure 10.2: Domain logic: formulas

$$u \leq u \qquad \frac{u \leq v \quad v \leq w}{u \leq w} \qquad u \wedge (v \vee w) \leq (u \wedge v) \vee (u \wedge w)$$

$$\frac{\forall\, i \in I \;\; u_i \leq v \quad (I \textit{ finite})}{\bigvee_{i\in I} u_i \leq v} \qquad u_i \leq \bigvee_{i\in I} u_i$$

$$\frac{\forall\, i \in I \;\; u \leq v_i \quad (I \textit{ finite})}{u \leq \bigwedge_{i\in I} v_i} \qquad \bigwedge_{i\in I} v_i \leq v_i$$

$$\frac{U, V \in \mathcal{K}(\Omega D_i) \quad U \subseteq V}{U \leq V} \qquad \frac{u' \leq u \quad v \leq v'}{u' \to v' \leq u \to v}$$

$$u \to (\bigwedge_{i\in I} v_i) = \bigwedge_{i\in I}(u \to v_i) \qquad (\bigvee_{i\in I} u_i) \to v = \bigwedge_{i\in I}(u_i \to v)$$

$$C(1) \qquad \frac{C(u)}{u \to (\bigvee_{i\in I} v_i) = \bigvee_{i\in I}(u \to v_i)}$$

$$\frac{U \in \mathcal{K}(\Omega D_i) \quad U \textit{ coprime}}{C(U)} \qquad \frac{C(u) \quad u = v}{C(v)}$$

$$(C \to \textit{Scott}) \quad \frac{\forall\, i \in I \;\; C(u_i) \textit{ and } C(v_i)\; \forall\, J \subseteq I \;\; (\bigwedge_{j\in J} v_j = 0 \Rightarrow \bigwedge_{j\in J} u_j = 0)}{C(\bigwedge_{i\in I}(u_i \to v_i))}$$

Figure 10.3: Domain Logic: entailment and coprimeness judgments

$$\frac{x : u \in \Gamma}{\Gamma \vdash x : u} \qquad \frac{\Gamma \vdash M : u \quad \Delta \leq \Gamma \quad u \leq v}{\Delta \vdash M : v}$$

$$\frac{\Gamma \vdash M : u \to v \quad \Gamma \vdash N : u}{\Gamma \vdash MN : v} \qquad \frac{\Gamma \cup \{x : u\} \vdash M : v}{\Gamma \vdash \lambda x.M : u \to v}$$

$$\frac{\forall\, i \in I \;\; \Gamma \cup \{x : u_i\} \vdash M : v \quad (I \textit{ finite})}{\Gamma \cup \{x : \bigvee_{i\in I} u_i\} \vdash M : v} \qquad \frac{\forall\, i \in I \;\; \Gamma \vdash M : u_i \quad (I \textit{ finite})}{\Gamma \vdash M : \bigwedge_{i\in I} u_i}$$

Figure 10.4: Domain logic: typing judgments

$$\begin{array}{lcllcl} [\![U]\!] & = & U & [\![u \to v]\!] & = & [\![u]\!] \to [\![v]\!] \\ [\![\bigwedge_{i\in I} u_i]\!] & = & \bigcap_{i\in I} [\![u_i]\!] & [\![\bigvee_{i\in I} u_i]\!] & = & \bigcup_{i\in I} [\![u_i]\!] \end{array}$$

Figure 10.5: Semantics of formulas

- *The "type" system whose judgements have the form* $\Gamma \vdash M : u$*, where* M *is a* λ*-term and* Γ *is a set consisting of distinct pairs* $x : v$*, given in figure 10.4. All the free variables of* M *are declared in* Γ*, and* $\Delta \leq \Gamma$ *means:* $\Delta = \{x_1 : u_1, \ldots, x_n : u_n\}$*,* $\Gamma = \{x_1 : v_1, \ldots, x_n : v_n\}$*, and* $u_i \leq v_i$ *for all* i*.*

The denotational semantics of types and of simply typed λ*-terms are defined as in chapter 4:* $[\![\sigma \to \tau]\!] = [\![\sigma]\!] \to [\![\tau]\!]$*, etc... The meaning of the formulas of* $\Phi(\sigma)$*, for all* σ*, is given in figure 10.5. Validity of the three kinds of judgements is defined in figure 10.6.*

Theorem 10.5.4 *The following properties hold:*

$$\begin{array}{rll} \models u \leq v & \textit{iff} & [\![u]\!] \leq [\![v]\!] \\ \models C(u) & \textit{iff} & [\![u]\!] \textit{ is coprime} \\ x_1 : u_1, \ldots, x_n : u_n \models M : u & \textit{iff} & \forall\, \rho \;\; ((\forall\, i \;\; \rho(x_i) \in [\![u_i]\!]) \Rightarrow [\![M]\!]\rho \in [\![u]\!]) \end{array}$$

Figure 10.6: Semantics of judgments

1. $u \leq v$ *is provable iff* $\models u \leq v$,
2. $C(u)$ *is provable iff* $\models C(u)$,
3. $\Gamma \vdash M : u$ *iff* $\Gamma \models M : u$

PROOF HINT. (1) and (2) have been already proved in substance in proposition 10.5.2. (3) is proved via a coprime completeness claim (cf. proposition 10.5.2). For an u such that $[\![u]\!]$ is coprime, i.e., $[\![u]\!] = \uparrow d$ for some compact d, $[\![M]\!]\rho \in [\![u]\!]$ reads $d \leq [\![M]\!]\rho$. Then the coprime completeness claim follows from the following almost obvious equivalences:

$$\begin{array}{ll} d \leq [\![MN]\!]\rho \text{ iff } \exists\, e\ (d \to e) \leq [\![M]\!]\rho \text{ and } e \leq [\![N]\!]\rho & \text{by continuity} \\ (d \to e) \leq [\![\lambda x.M]\!]\rho \text{ iff } e \leq [\![M]\!]\rho[d/x] & \text{by definition of } d \to e\ . \end{array}$$

□

10.6 Bifinite Domains in Logical Form *

We sketch how the logical treatment just given for Scott domains can be adapted to bifinite cpo's.

Definition 10.6.1 (Gunter joinable) *A finite subset* $\gamma \subseteq \mathcal{K}(D) \times \mathcal{K}(E)$ *is called Gunter joinable if*

$$\forall\, d_0 \in \mathcal{K}(D)\ \ \{(d,e) \in \gamma \mid d \leq d_0\} \text{ has a maximum in } \gamma.$$

Any Gunter joinable set γ *induces a function* $\mathcal{G}(\gamma)$ *defined by*

$$\mathcal{G}(\gamma)(x) = \max\{e \mid \exists\, d\ \ (d,e) \in \gamma \text{ and } d \leq x\}.$$

Lemma 10.6.2 *If* γ, γ' *are Gunter joinable, then:*

1. $\mathcal{G}(\gamma) = \bigvee\{d \to e \mid (d,e) \in \gamma\}$,
2. $d' \to e' \leq \mathcal{G}(\gamma) \Leftrightarrow \exists\, d, e\ \ (d \leq d', e' \leq e \text{ and } (d,e) \in \gamma)$,
3. $\mathcal{G}(\gamma) \leq \mathcal{G}(\gamma') \Leftrightarrow \forall\, (d',e') \in \gamma'\ \exists\, d, e\ \ d \leq d', e' \leq e \text{ and } (d,e) \in \gamma$,

PROOF. (1) follows from the following remark: by definition of $\mathcal{G}(\gamma)$, if $\mathcal{G}(\gamma) \neq \bot$, then $\mathcal{G}(\gamma)(x) = (d \to e)(x)$ for some $(d,e) \in \gamma$.

(2) First recall that $d' \to e' \leq \mathcal{G}(\gamma)$ can be reformulated as $e' \leq \mathcal{G}(\gamma)(d')$.

($\Leftarrow$) Then $d' \to e' \leq d \to e$, and a fortiori $d' \to e' \leq \mathcal{G}(\gamma)$.

($\Rightarrow$) By definition of $\mathcal{G}(\gamma)$, $\mathcal{G}(\gamma)(d') = e$ for some $(d,e) \in \gamma$ with $d \leq d'$.

(3) is an obvious consequence of (2). □

Proposition 10.6.3 *If* D, E *are bifinite, then* $\mathcal{K}(D \to E) = \{\mathcal{G}(\gamma) \mid \gamma \text{ is Gunter joinable}\}$.

PROOF. Clearly, each $\mathcal{G}(\gamma)$, as a finite lub of step functions, is compact. Conversely, we know from proposition 5.2.4 that the compact elements of $D \to E$ have the form $r(f)$, where $f : D \to E$ is a continuous function, and r is a finite projection defined from two finite projections $p : D \to D$ and $q : E \to E$ by $r(f)(x) = q(f(p(x)))$. We have to find a γ such that $r(f) = \mathcal{G}(\gamma)$. We claim that the following does the job:

$$\gamma = \{(p(y), q(f(p(y)))) \mid y \in D\}.$$

- γ is finite: by the finiteness of the ranges of p, q.
- γ is Gunter joinable: Let $x \in D$. We claim:

$$p(x) = \max\{p(y) \mid y \in D \text{ and } p(y) \leq x\}.$$

Then obviously $(p(x), q(f(p(x))))$ is the maximum of $\{(d, e) \in \gamma \mid d \leq x\}$.

- $r(f) = \mathcal{G}(\gamma)$: We have:

$$\begin{array}{lcll} r(f(x)) & = & q(f(p(x))) & \text{by definition of } r \\ \mathcal{G}(\gamma)(x) & = & q(f(p(x))) & \text{by definition of } \mathcal{G}(\gamma)\ . \end{array}$$

□

The following equivalent formulation of Gunter joinable subsets is due to Abramsky, and is more easy to capture in logical form.

Proposition 10.6.4 *Let $\{(d_i, e_i) \mid i \in I\} \subseteq \mathcal{K}(D) \times \mathcal{K}(E)$ be finite. Then $\{(d_i, e_i) \mid i \in I\}$ is Gunter joinable iff*

$$\forall J \subseteq I\ \exists K \subseteq I\ \ MUB(\{d_j \mid j \in J\}) = \{d_k \mid k \in K\} \text{ and } \forall j \in J, k \in K\ \ e_j \leq e_k.$$

PROOF. ($\Rightarrow$) Let $m \in MUB(\{d_j \mid j \in J\})$, and let $(d_k, e_k) = \max\{(d_i, e_i) \mid d_i \leq m\}$. We claim: $m = d_k$. Since $d_k \leq m$ by definition, it is enough to show that $d_k \in UB(\{d_j \mid j \in J\})$, which follows from the obvious inclusion $\{d_j \mid j \in J\} \subseteq \{(d_i, e_i) \mid d_i \leq m\}$. This inclusion also implies $\forall j \in J\ \ e_j \leq e_k$.

($\Leftarrow$) Let $d \in \mathcal{K}(D)$, $J = \{j \mid d_j \leq d\}$, and let K be as in the statement. By property M there exists $k \in K$ such that $d \geq d_k$. But then $k \in J$ by definition of J, and since d_k is both an upper bound and an element of $\{d_j \mid j \in J\}$, it is a maximum of this set. Moreover, since $e_j \leq e_k$, for all $j \in J$, we have that (d_k, e_k) is the desired maximum. □

Exercise 10.6.5 (*) *Show that the statement of theorem 10.5.4 remains true after the following two changes in definition 10.5.3: (1) $D_1, \ldots, D_n$ are now fixed bifinite cpo's. (2) Axiom ($C \to$ Scott) of definition 10.5.3 is replaced by the following axiom:*

$$(C \to \textit{bifinite}) \quad \frac{\forall i \in I\ \ C(u_i) \text{ and } C(v_i) \qquad \forall J \subseteq I\ \exists K \subseteq I\ \ \bigwedge_{j \in J} u_j = \bigvee_{k \in K} u_k \text{ and } \forall j \in J, k \in K\ \ v_j \leq v_k}{C(\bigwedge_{i \in I}(u_i \to v_i))}$$

Hints: The principal difficulty is to make sure that any u can be written as a disjunction of formulas of the form $\bigwedge_{i \in I}(u_i \to v_i)$ where the u_i's and the v_i's satisfy the conditions of rule ($C \to$ bifinite). Remove faulty disjunctions and replace them by disjunctions of conjunctions. Design a terminating strategy for this.

Chapter 11

Dependent and Second Order Types

The main goal of this chapter is to introduce λ-calculi with *dependent* and *second order* types, to discuss their interpretation in the framework of traditional domain theory (chapter 15 will mention another approach based on realizability), and to present some of their relevant syntactic properties.

Calculi with dependent and second order types are rather complex syntactic objects. In order to master some of their complexity let us start with a discussion from a semantic viewpoint. Let $\mathbf{T}$ be a category whose objects are regarded as types. The category $\mathbf{T}$ contains atomic types like the singleton type 1, the type *nat* representing natural numbers, and the type *bool* representing boolean values. The collection $\mathbf{T}$ is also closed with respect to certain data type constructions. For example, if A and B are types then we can form new types such as a *product type* $A \times B$, a *sum type* $A + B$, and an *exponent type* $A \to B$.

In first approximation, a *dependent type* is a family of types indexed over another type A. We represent such family as a transformation F from A into the collection of types $\mathbf{T}$, say $F : A \to \mathbf{T}$. As an example of dependent type we can think of a family $Prod.bool : nat \to \mathbf{T}$ that given a number n returns the type $bool \times \cdots \times bool$ (n times).

If the family F is indexed over the collection of all types $\mathbf{T}$, say $F : \mathbf{T} \to \mathbf{T}$, then we are in the realm of *second order types*. As an example of a second order type we can think of a family $Fun : \mathbf{T} \to \mathbf{T}$ that given a type A returns the type $A \to A$ of functions over A.

If types, and the collection of types $\mathbf{T}$, can be seen as categories then we can think of dependent and second order types as functors. Let us warn the reader that in this preliminary discussion we are considering a simplified situation. In general we want to combine dependent and second order types. For example, we may consider the family $Poly.Prod : \mathbf{T} \times nat \to \mathbf{T}$ that takes a type A, a number n, and returns the type $A \times \cdots \times A$ (n times).

Probably the most familiar examples of dependent and second-order types

arise in logic. If $\phi(x)$ is a formula depending on the variable x then we can think of $\phi(x)$ as a family of propositions indexed over the universe of terms U, say $\phi : U \to Prop$. This is a dependent type. On the other hand, if we consider a formula $\phi(X)$, parametric in a formula variable X then we can think of $\phi(X)$ as a family of propositions indexed over the universe of propositions, say $\phi : Prop \to Prop$. This is a a second order type. If we allow quantifications over variables we can form the formulas $\forall x.\phi$, and $\exists x.\phi$. This is the realm of first-order logic. If moreover we allow quantifications over formula variables we can form the formulas $\forall X.\phi$, and $\exists X.\phi$, and we are in the realm of second order logic.

Dependent types also appear in several type systems (or generalized logics) such as DeBruijn's Automath [dB80], Martin-Löf's Type Theory [ML84], and Edinburgh LF [HHP93]. Second order types appear in a rather pure form in Girard's system F [Gir72] (which is equivalent to a system of natural deduction for minimal, implicative, propositional second order logic), they also appear, for instance, in the Calculus of Constructions [CH88] but there they are combined with dependent types and more.

Let us now look at the interpretation. Given a family $A : U \to Prop$ we can obtain two new propositions $\forall_U A$, and $\exists_U A$ where we understand $\forall_U$ as a meet or a product, and $\exists_U$ as a join or a sum. In general, given a family of types $F : \mathbf{I} \to \mathbf{T}$ indexed over a category $\mathbf{I}$ we are interested in building two new types that we may denote, respectively, with $\Pi_{\mathbf{I}} F$ and $\Sigma_{\mathbf{I}} F$, and that correspond, respectively, to the product and the sum of the family F.

Relying on this informal discussion, we can summarize the contents of this chapter as follows. The main problem considered in section 11.1 is to provide a concrete domain-theoretical interpretation of the constructions sketched above. In particular, we build a category of domains that is "closed" under (certain) indexed products, and (certain) indexed sums. The first simple idea is to interpret types as domains of a given category $\mathbf{C}$, and the collection of types as the related category $\mathbf{C}^{ip}$ of injection-projection pairs. What is then a dependent type F indexed over some domain D? Since every preorder can be seen as a category, it is natural to ask that F is a functor from D to $\mathbf{C}^{ip}$. Analogously a second order type will be seen as an endo-functor over $\mathbf{C}^{ip}$. However this will not suffice, for instance we will need that the family F preserves directed colimits, namely it is *cocontinuous*.

In section 11.2 we provide a syntactic formalization of the semantic ideas sketched above. To this end we introduce a calculus of dependent and second order types and discuss some of its basic properties. We call this calculus $\lambda P2$-calculus, following a classification proposed in [Bar91a] (the "P" stands for positive logic and the "2" for second order). We also briefly discuss an interpretation of the $\lambda P2$-calculus which relies on the domain-theoretical constructions introduced in section 11.1. The interpretation is presented in a set-theoretical

notation, a general categorical treatment would require an amount of category-theoretical background that goes beyond our goals. In this respect let us mention [AL91] which contains a rather complete analysis of the categorical structure needed to interpret second order types from the viewpoint of *indexed category theory* and *internal category theory*. Several approaches to the categorical semantics of dependent types have been considered, we refer to [JMS91] for an account based on fibrations.

In section 11.3 we describe another interpretation of type theories based on the idea that *types denote retractions*. In this respect we take two different but related approaches. First, we further develop the properties of the domain of finitary projections studied in section 7.4. In particular we show how to represent dependent and second order types in this structure. It turns out that certain "size problems" encountered in the domain constructions described in section 11.1 can be avoided in this context. Second, we present an extension of the $\lambda\beta$-calculus called $\lambda\beta p$-calculus in which "p" is a constant that denotes the *retraction of all retractions*. We define a simple, adequate translation of the $\lambda P2$-calculus in the $\lambda\beta p$-calculus.

The $\lambda P2$-calculus can be seen as the combination of two systems of independent interest: the system LF of dependent types and the system F of second order types. We reserve the sections 11.4 and 11.5 to a careful presentation of the syntactic properties of these two systems the main result being that both systems enjoy the strong normalization property (this property is enjoyed by the $\lambda P2$-calculus as well and can be proved by combining the techniques for system F and system LF). We also discuss two interesting applications that illustrate the expressive power of these systems: (1) The system LF has been proposed as a tool for the encoding of certain recurring aspects of logical systems such as α-conversion and substitution. We illustrate this principle by presenting an adequate and faithful representation of first-order classical logic in LF. (2) The system F can represent a large variety of inductively defined structures and functions defined on them by *iteration*.

11.1 Domain-Theoretical Constructions

In set theory we may represent a family of sets as a function $F : X \to \mathbf{Set}$. More precisely, we consider a graph given as $\{(x, Fx)\}_{x \in X}$. In this way we do not have to speak about the *class* of sets. We formulate some basic constructions that will be suitably abstracted in the sequel. In the first place we can build the (disjoint) sum of the sets in the family as:

$$\Sigma_X F = \{(x, y) \mid x \in X \text{ and } y \in Fx\} \ .$$

Observe that there is a projection morphism $p : \Sigma_X F \to X$ that is defined as $p(x, y) = x$. On the other hand we can build a product of the sets in the family

as:

$$\Pi_X F = \{f : X \to \bigcup_{x \in X} Fx \mid \forall x \in X\,(fx \in Fx)\}\ .$$

There is another way to write $\Pi_X F$ using the notion of *section* of the projection morphism $p : \Sigma_X F \to X$ (the weakness of this method is that it requires the existence of $\Sigma_X F$). A section is a morphism $s : X \to \Sigma_X F$ such that $p \circ s = id_X$, in other words for any $x \in X$ the section s picks up an element in Fx. It is then clear that the collection of sections of p is in bijective correspondence with $\Pi_X F$.

Exercise 11.1.1 *Verify that the definitions of $\Sigma_X F$ and $\Pi_X F$ can be completed so to obtain sum and product of the objects in the family in the category of sets.*

Exercise 11.1.2 *Suppose that the family $F : X \to \mathbf{Set}$ is constant, say $F(x) = Y$ for each x in X. Then verify that $\Sigma_X F \cong X \times Y$, and $\Pi_X F \cong X \to Y$.*

Exercise 11.1.3 *Show that every small category with arbitrary products is a poset (this is an observation of Freyd). Hint: We recall that a category is small if the collection of its morphisms is a set. Given two distinct morphisms $f, g : a \to b$ in the small complete category $\mathbf{C}$ consider $\Pi_I b$. The cardinality of $\mathbf{C}[a, \Pi_I b]$ exceeds that of $Mor_{\mathbf{C}}$ when I is big enough.*

Remark 11.1.4 *Observe that in the definition of $\Sigma_X F$ and $\Pi_X F$ it is important that X is a set, so that the graph of F is again a set, and so are $\Sigma_X F$ and $\Pi_X F$. This observation preludes to the problem we will find when dealing with second order types. In the interpretation suggested above neither the graph of a family $F : \mathbf{Set} \to \mathbf{Set}$ nor $\Sigma_{\mathbf{Set}} F$ and $\Pi_{\mathbf{Set}} F$ turn out to be sets!*

In the following we generalize the ideas sketched above to a categorical setting. Given a family F as a functor $F : \mathbf{X} \to \mathbf{Cat}$, the category $\Sigma_{\mathbf{X}} F$ provides the interpretation of the sum. On the other hand, the product is represented by the category of *sections*, say $\Pi_X F$, of the *fibration* $p : \Sigma_{\mathbf{X}} F \to \mathbf{X}$ that projects $\Sigma_{\mathbf{X}} F$ onto $\mathbf{X}$. A section s of p is a functor $s : \mathbf{X} \to \Sigma_{\mathbf{X}} F$ such that $p \circ s = id_{\mathbf{X}}$.

Dependent types in Cat. Let $F : \mathbf{X} \to \mathbf{Cat}$ be a functor where $\mathbf{X}$ is a small category, we define the categories $\Sigma_{\mathbf{X}} F, \Pi_{\mathbf{X}} F$, and the functor $p : \Sigma_{\mathbf{X}} F \to \mathbf{X}$ as follows:

$$\begin{array}{ll} \Sigma_{\mathbf{X}} F & = \{(x, y) \mid x \in \mathbf{X}, y \in Fx\} \\ \Sigma_{\mathbf{X}} F[(x, y), (x', y')] & = \{(f, \alpha) \mid f : x \to x', \alpha : F(f)(y) \to y'\} \\ id_{(x,y)} & = (id_x, id_y) \\ (g, \beta) \circ (f, \alpha) & = (g \circ f, \beta \circ (Fg)(\alpha)) \end{array}$$

The category $\Sigma_{\mathbf{X}} F$ is often called the *Grothendieck category*. The functor $p : \Sigma_{\mathbf{X}} F \to \mathbf{X}$ is defined as:

$$p(x, y) = x \quad p(f, \alpha) = f\ .$$

The category $\Pi_{\mathbf{X}}F$ is defined as:

$$\begin{array}{ll} \Pi_{\mathbf{X}}F & = \{s : X \to \Sigma_{\mathbf{X}}F \mid p \circ s = id_{\mathbf{X}}\} \\ \Pi_{\mathbf{X}}F[s,s'] & = \{\nu : s \to s' \mid \nu \text{ is a } \textit{cartesian} \text{ natural transformation}\} \end{array}$$

where a *cartesian* natural transformation $\nu : s \to s'$ is a natural transformation determined by a family $\{(id_x, \gamma_x)\}_{x \in X}$ with $s(x) = (x, y)$, $s'(x) = (x, z)$, and $\gamma_x : y \to z$ (so the first component of the transformation is constrained to be the identity). Observe that for a section s we have $s(x) = (x, y)$, for all $x \in \mathbf{X}$, and $s(f) = (f, \alpha)$, for all $f \in Mor_{\mathbf{X}}$.

The next issue concerns the specialization of these definitions to the categories of cpo's and Scott domains. The problem is to determine suitable continuity conditions so that the constructions of sum and product return a "domain", say an algebraic cpo's. It turns out that everything works smoothly for dependent types. On the other hand second order types give some problems.

(1) The sum of a second order type is not in general a domain.

(2) The product of a second order type is only equivalent, as a category, to a domain.

(3) Bifinite domains are not closed under the product construction (this motivates our shift towards Scott domains).

Dependent types in Cpo. We refine the construction above to the case where $F : D \to \mathbf{Cpo}^{ip}$ is a functor, D is a cpo, and $\mathbf{Cpo}^{ip}$ is the category of cpo's and injection-projection pairs. In other terms $\mathbf{X}$ becomes a poset category D and the codomain of the functor is $\mathbf{Cpo}^{ip}$. By convention, if $d \leq d'$ in D then we also denote with $d \leq d'$ the unique morphism from d to d' in the poset category D. If $f : D \to E$ is a morphism in $\mathbf{Cpo}^{ip}$ then we denote with f^+ the injection and with f^- the projection.

Proposition 11.1.5 (dependent sum in $\mathbf{Cpo}^{ip}$) *Let D be a cpo and $F : D \to \mathbf{Cpo}^{ip}$ be a functor, then the following is a cpo:*

$$\begin{array}{l} \Sigma_D F = \{(d, e) \mid d \in D, e \in Fd\}, \textit{ ordered by} \\ (d, e) \leq_\Sigma (d', e') \textit{ iff } d \leq_D d' \textit{ and } F(d \leq d')^+(e) \leq_{F(d')} e' \ . \end{array}$$

PROOF. By proposition 3.1.3, the category $\mathbf{Cpo}^{ip}$ is the same as the category where a morphism is the injection component of an injection-projection pair. The latter is a subcategory of **Cat**. It is immediate to verify that $(\Sigma_D F, \leq_\Sigma)$ is a poset with least element $(\bot_D, \bot_{F(\bot_D)})$.

Next let $X = \{(d_i, e_i)\}_{i \in I}$ be directed in $\Sigma_D F$. Set for $d = \bigvee_{i \in I} d_i$:

$$\bigvee X = (d, \bigvee_{i \in I} F(d_i \leq d)^+(e_i)) \ .$$

We claim that this is well defined and the lub of X in $\Sigma_D F$.

- $\{F(d_i \leq d)^+(e_i)\}_{i\in I}$ is directed. Since X is directed:

$$\forall i,j,\exists k(d_i \leq d_k,\ d_j \leq d_k,\ F(d_i \leq d_k)^+(e_i) \leq e_k,\ F(d_j \leq d_k)^+(e_j) \leq e_k)\ .$$

Hence $F(d_i \leq d)^+(e_i) = F(d_k \leq d)^+ \circ F(d_i \leq d_k)^+(e_i) \leq F(d_k \leq d)^+(e_k)$, and similarly for j.

- $\bigvee X$ is an upper bound for X: immediate, by definition.

- $\bigvee X$ is the lub. If (d', e') is an upper bound for X then it is clear that $d \leq d'$. Next we observe:

$$\begin{aligned} F(d \leq d')^+(\textstyle\bigvee_{i\in I} F(d_i \leq d)^+(e_i)) &= \textstyle\bigvee_{i\in I} F(d \leq d')^+(F(d_i \leq d)^+(e_i)) \\ &= \textstyle\bigvee_{i\in I}(F(d_i \leq d')^+(e_i)) \leq e'\ . \end{aligned}$$

□

Exercise 11.1.6 *Verify that the definition of $\Sigma_D F$ is an instance of the definition in* **Cat**.

Proposition 11.1.7 (dependent product in $\mathbf{Cpo}^{ip}$) *Let D be a cpo and $F : D \to \mathbf{Cpo}^{ip}$ be a functor, then the following is a cpo with the pointwise order induced by the space $D \to \Sigma_D F$:*

$$[\Pi_D F] = \{s : D \to \Sigma_D F \mid s \text{ continuous},\ p \circ s = id_D\}\ .$$

PROOF. First we observe that $p : \Sigma_D F \to D$ is continuous as for any $\{(d_i, e_i)\}_{i\in I}$ directed set in $\Sigma_D F$ we have, taking $d = \bigvee_{i\in I} d_i$:

$$\begin{aligned} p(\textstyle\bigvee_{i\in I}(d_i, e_i)) &= p(\textstyle\bigvee_{i\in I} d_i, \bigvee_{i\in I} F(d_i \leq d)^+(e_i)) \\ &= \textstyle\bigvee_{i\in I} d_i = \bigvee_{i\in I} p(d_i, e_i)\ . \end{aligned}$$

We can also define a least section as $s(d) = (d, \bot_{F(d)})$. Next we remark that for any directed set $\{s_i\}_{i\in I}$ in $[\Pi_D F]$ we have, for any $d \in D$:

$$p \circ (\bigvee_{i\in I} s_i)(d) = p(\bigvee_{i\in I} s_i(d)) = \bigvee_{i\in I} p(s_i(d)) = d\ .$$

Hence the lub of a directed set of sections exists and it is the same as the lub in $D \to \Sigma_D F$. □

We given an equivalent definition of continuous section.

Definition 11.1.8 *Let D be a cpo and $F : D \to \mathbf{Cpo}^{ip}$ be a functor. Consider $f : D \to \bigcup_{d\in D} Fd$ such that $fd \in Fd$, for each $d \in D$. We say that f is* cocontinuous *if $F(d \leq d')^+(fd) \leq fd'$, and for any $\{d_i\}_{i\in I}$ directed in D, such that $\bigvee_{i\in I} d_i = d$,*

$$f(d) = \bigvee_{i\in I} F(d_i \leq d)^+(f(d_i))\ .$$

Clearly $[\Pi_D F]$ is isomorphic to:

$\{f : D \to \bigcup_{d\in D} Fd \mid \forall d\,(fd \in Fd)$ and f is cocontinuous$\}$, ordered by
$f \leq g$ iff $\forall d \in D\,(fd \leq_{Fd} gd)$.

Exercise 11.1.9 *Verify that the definition of* $[\Pi_D F]$ *in* $\mathbf{Cpo}^{ip}$ *corresponds to select a full subcategory of cocontinuous sections out of the general construction described for* **Cat**.

Dependent types in Scott domains. We denote with **S** (S for Scott) the category of algebraic, bounded complete, cpo's (Scott domains for short, cf. chapter 1). The following hypotheses suffice to guarantee that the constructions defined above return Scott domains:

- The domain of the family is a Scott domain.
- The codomain of the family is the category $\mathbf{S}^{ip}$ of Scott domains and injection-projection pairs.
- Less obviously, the functor F is *cocontinuous* in a sense which we define next.

Definition 11.1.10 (directed colimits) *A* directed diagram *is a diagram indexed over a directed set. We say that a category has* directed colimits *if it has colimits of directed diagrams. We say that a functor is* cocontinuous *if it preserves colimits of directed diagrams.*

Applying the theory developed in section 7.1 it is easy to derive the following properties.

Proposition 11.1.11 *(1) The category* $\mathbf{S}^{ip}$ *has directed colimits.*
(2) *Given a Scott domain* D *and a functor* $F : D \to \mathbf{S}^{ip}$, F *is cocontinuous iff for any* $\{d_i\}_{i\in I}$ *directed in* D *such that* $\bigvee_{i\in I} d_i = d$,

$$\bigvee_{i\in I} F(d_i \leq d)^+ \circ F(d_i \leq d)^- = id_{F(D)} \ .$$

(3) *A functor* $F : \mathbf{S}^{ip} \to \mathbf{S}^{ip}$ *is cocontinuous iff for any Scott domain* D *and any directed set* $\{p_i\}_{i\in I}$ *of projections over* D,

$$\bigvee_{i\in I} p_i = id_D \quad \Rightarrow \quad \bigvee_{i\in I} F(p_i) = id_{F(d)} \ .$$

Proposition 11.1.12 (dependent sum and product in Scott domains) *Let* D *be a Scott domain and* $F : D \to \mathbf{S}^{ip}$ *be a cocontinuous functor, then the cpo's* $\Sigma_D F$ *and* $[\Pi_D F]$ *are Scott domains.*

PROOF. • $\Sigma_D F$ is bounded complete. Let $X = \{(d_i, e_i)\}_{i\in I}$ be bounded in $\Sigma_D F$ by (d', e'). Then: (i) $\{d_i\}_{i\in I}$ is bounded in D by d' and therefore $\exists \bigvee_{i\in I} d_i = d$. (ii) Moreover $\{F(d_i \leq d)^+(e_i)\}_{i\in I}$ is bounded by $F(d \leq d')^-(e')$ as:

$$F(d_i \leq d')^+(e_i) = F(d \leq d')^+ F(d_i \leq d)^+(e_i) \leq e' \text{ implies}$$
$$F(d_i \leq d)^+(e_i) \leq F(d \leq d')^-(e') \ .$$

Hence we set $e = \bigvee_{i\in I} F(d_i \leq d)^+(e_i)$. It is immediate to check that (d, e) is the lub.

• $\Sigma_D F$ is algebraic. We claim:

(1) $\mathcal{K}(\Sigma_D F) \supseteq \{(d, e) \mid d \in \mathcal{K}(D) \text{ and } e \in \mathcal{K}(F(d))\} = K$.

(2) For any $(d, e) \in \Sigma_D F$, $\downarrow (d, e) \cap K$ is directed with lub (d, e).

Proof of (1). Let $d' \in \mathcal{K}(D)$, $e' \in \mathcal{K}(F(d'))$, and $X = \{(d_i, e_i)\}_{i\in I}$ be directed in $\Sigma_D F$ with $d' \leq \bigvee_{i\in I} d_i = d$, and $F(d' \leq d)^+(e') \leq \bigvee_{i\in I} F(d_i \leq d)^+(e_i)$. By hypothesis, d' and e' are compact. $F(d' \leq d)^+(e')$ is also compact, hence we can find j such that $d' \leq d_j$, $F(d' \leq d)^+(e') \leq F(d_j \leq d)^+(e_j)$, that implies $F(d' \leq d_j)^+(e') \leq e_j$. That is $(d', e') \leq (d_j, e_j)$. Hence $(d', e') \in \mathcal{K}(\Sigma_D F)$.

Proof of (2). The set is directed because $\Sigma_D F$ is bounded complete. Given (d, e) we consider: (i) $\{d_i\}_{i\in I} \subseteq \mathcal{K}(D)$ directed such that $\bigvee_{i\in I} d_i = d$, and (ii) $\forall i \in I$ $\{e_{i,j}\}_{j\in J_i} \subseteq \mathcal{K}(F(d_i))$ directed such that $\bigvee_{j\in J_i} e_{i,j} = F(d_i \leq d)^-(e)$.

Then the following equations hold (the last one by cocontinuity of F):

$$\begin{aligned}\bigvee_{i\in I, j\in J_i}(d_i, e_{ij}) &= (d, \bigvee_{i\in I, j\in J_i} F(d_i \leq d)^+(e_{i,j})) \\ &= (d, \bigvee_{i\in I} F(d_i \leq d)^+(\bigvee_{j\in J_i} e_{i,j})) \\ &= (d, \bigvee_{i\in I} F(d_i \leq d)^+ F(d_i \leq d)^-(e)) = (d, e) \ .\end{aligned}$$

• $[\Pi_D F]$ is bounded complete. Suppose $\{s_i\}_{i\in I}$ is a bounded set in $[\Pi_D F]$. Since bounded completeness is preserved by exponentiation we can compute $\bigvee_{i\in I} s_i$ in $D \to \Sigma_D F$. It remains to show that $p \circ (\bigvee_{i\in I} s_i) = id_D$. We observe that for any $d \in D$:

$$p((\bigvee_{i\in I} s_i)(d)) = p(\bigvee_{i\in I} s_i(d)) = \bigvee_{i\in I} p(s_i(d)) = d \ .$$

• $[\Pi_D F]$ is algebraic. We consider the step sections (cf. lemma 1.4.8) $[d, e]$ for $d \in \mathcal{K}(D)$, $e \in \mathcal{K}(F(d))$, defined as:

$$[d, e](x) = \begin{cases} (x, F(d \leq x)^+(e)) & \text{if } d \leq x \\ (x, \bot_{F(x)}) & \text{otherwise} \end{cases} \ .$$

One can verify that $[d, e]$ is compact in $[\Pi_D F]$. It remains to observe that for any $s \in [\Pi_D F]$, $\{[d, e] \mid [d, e] \leq s\}$ determines s. □

Second order types in Scott domains. We look for an interpretation of second order types as domains. Suppose that $F : \mathbf{S}^{ip} \to \mathbf{S}^{ip}$ is a cocontinuous functor. Then, as an instance of the general categorical construction, we can form the category $\Sigma_{\mathbf{S}^{ip}} F$. It is easily verified that $\Sigma_{\mathbf{S}^{ip}} F$ does not need to be a preorder as there can be several injection-projection pairs between two domains. We therefore concentrate our efforts on products. To this end we spell out the notion of cocontinuous section.

Definition 11.1.13 *Let $F : \mathbf{S}^{ip} \to \mathbf{S}^{ip}$ be a cocontinuous functor. A* cocontinuous *section s is a family $\{s(D)\}_{D \in \mathbf{S}^{ip}}$ such that:*

$$f : D \to E \text{ in } \mathbf{S}^{ip} \quad \Rightarrow \quad F(f)^+(s(D)) \leq s(E) \tag{11.1}$$

and for any $D \in \mathbf{S}^{ip}$ for any $\{f_i : D_i \to D\}_{i \in I}$ such that $\{f_i^+ \circ f_i^-\}_{i \in I}$ is directed we have:

$$\bigvee_{i \in I} (f_i^+ \circ f_i^-) = id_D \quad \Rightarrow \quad s(D) = \bigvee_{i \in I} (Ff_i)^+(s(D_i)) \tag{11.2}$$

Let $[\Pi^{ip}_{\mathbf{S}} F]$ be the collection of cocontinuous sections with the pointwise partial order:

$$s \leq s' \text{ iff } \forall D \in \mathbf{S}^{ip}\, (s(D) \leq s'(D)) \ .$$

The problem with this partial order is that the cocontinuous sections are not sets, hence a fortiori $[\Pi_{\mathbf{S}^{ip}} F]$ cannot be a Scott domain. However there is a way out of this foundational problem, namely it is possible to build a Scott domain which is order isomorphic to $[\Pi_{\mathbf{S}^{ip}} F]$. To this end we observe that the compact objects (cf. definition 7.3.3) in $\mathbf{S}^{ip}$ are the finite bounded complete cpo's, and that there is an enumeration $S_o = \{C_i\}_{i \in \omega}$ up to order-isomorphism of the compact objects. We define $[\Pi_{\mathbf{S}^{ip}_o} F]$ as the collection of sections $\{s(D)\}_{D \in \mathbf{S}^{ip}_o}$ such that:

$$s : D \to E \text{ in } \mathbf{S}^{ip}_o \quad \Rightarrow \quad F(f)^+(s(D)) \leq s(E) \tag{11.3}$$

This is the monotonicity condition 11.1 in definition 11.1.13 restricted to the subcategory $\mathbf{S}^{ip}_o$ (there is no limit condition, as $\mathbf{S}^{ip}_o$ is made up of compact objects). We observe that $[\Pi_{\mathbf{S}^{ip}_o} F]$ with the pointwise order is a poset. The following theorem is due to [Coq89], after [Gir86]. The basic remark is that a cocontinuous section is determined by its behaviour on $\mathbf{S}^{ip}_o$.

Theorem 11.1.14 (second order product) *Let $F : \mathbf{S}^{ip} \to \mathbf{S}^{ip}$ be a cocontinuous functor then: (1) $[\Pi_{\mathbf{S}^{ip}} F]$ is order isomorphic to $[\Pi_{\mathbf{S}^{ip}_o} F]$, and (2) the poset $[\Pi_{\mathbf{S}^{ip}_o} F]$ is a Scott-domain.*

PROOF HINT. (1) Any cocontinuous section $s \in [\Pi_{\mathbf{S}^{ip}} F]$ determines by restriction a section $res(s) \in [\Pi_{\mathbf{S}^{ip}_o} F]$. Vice versa given a section $s \in [\Pi_{\mathbf{S}^{ip}_o} F]$ we define its extension $ext(s)$, as follows:

$$ext(s)(E) = \bigvee \{(Ff)^+(s(D)) \mid D \in \mathbf{S}^{ip}_o \text{ and } f : D \to E \text{ in } \mathbf{S}^{ip}\} \tag{11.4}$$

The set $\{(Ff)^+(s(D)) \mid D \in \mathbf{S}^{ip}_o \text{ and } f : D \to E \text{ in } \mathbf{S}^{ip}\}$ is directed. Given $f_0 : D_0 \to E$, $f_1 : D_1 \to E$ we can find $D' \in \mathbf{S}^{ip}_o$ and $g_0 : D_0 \to D'$, $g_1 : D_1 \to D'$, $g : D' \to E$ such that $g \circ g_0 = f_0$ and $g \circ g_1 = f_1$.

The section $ext(s)$ satisfies condition 11.1 because given $g : E \to E'$ we compute:

$$\begin{aligned}(Fg)^+(ext(s)(E)) &= (Fg)^+(\bigvee\{(Ff)^+(s(D)) \mid D \in \mathbf{S}^{ip}_o \text{ and } f : D \to E\}) \\ &= \bigvee\{F(g \circ f)^+(s(D)) \mid D \in \mathbf{S}^{ip}_o \text{ and } f : D \to E\} \\ &\leq \bigvee\{F(h)^+(s(D)) \mid D \in \mathbf{S}^{ip}_o \text{ and } h : D \to E'\} = (ext(s))(E') \ .\end{aligned}$$

With reference to condition 11.2 we need to check that:

$$ext(s)(D) \leq \bigvee_{i \in I} (Ff_i)^+(ext(s)(D_i))$$

(the other inequality follows by condition 11.1). According to the definition of ext consider $D' \in \mathbf{S}^{ip}_o$ and $f : D' \to D$. We can find $j \in I$ and $h : D' \to D_j$ such that $f_j \circ h = f$. Then:

$$\begin{aligned}F(f)^+(s(D')) &= F(f_j \circ h)^+(s(D')) \\ &= F(f_j)^+((Fh)^+(s(D'))) \quad \leq F(f_j)^+(ext(s(D_j))) \ .\end{aligned}$$

It is easily checked that res and ext are monotonic. We observe that $s(D) = ext(s)(D)$ if $D \in \mathbf{S}^{ip}_o$. To show $\leq$ consider the identity on D, and to prove $\geq$ use condition 11.1. It follows $res(ext(s)) = s$.

To prove $ext(res(s)) = s$ we compute applying condition 11.2:

$$\begin{aligned}ext(res(s))(D) &= \bigvee\{(Ff)^+((res(s))(D')) \mid D' \in \mathbf{S}^{ip}_o \text{ and } f : D' \to D\} \\ &= \bigvee\{(Ff)^+(s(D')) \mid D' \in \mathbf{S}^{ip}_o \text{ and } f : D' \to D\} = s(D) \ .\end{aligned}$$

(2) The least element is the section $\{\bot_D\}_{D \in \mathbf{S}^{ip}_o}$. The lub s of a directed set $\{s_i\}_{i \in I}$ is defined as $s(D) = \bigvee_{i \in I} s_i(D)$. Bounded completeness is left to the reader. To show algebraicity, we define for $D \in \mathbf{S}^{ip}_o$ and $e \in \mathcal{K}(FD)$ the section:

$$[D, e](D') = \bigvee\{(Ff)^+(e) \mid f : D \to D' \text{ in } \mathbf{S}^{ip}\} \tag{11.5}$$

Compact elements are the existing finite lub's of sections with the shape 11.5. □

Hence, although $[\Pi_{\mathbf{S}^{ip}} F]$ is not a poset because its elements are classes, it is nevertheless order-isomorphic to a Scott domain $[\Pi_{\mathbf{S}^{ip}_o} F]$. Figure 11.1 summarizes our results on the closure properties of the Σ and Π constructions in the categories $\mathbf{Cpo}^{ip}$ and $\mathbf{S}^{ip}$.

Exercise 11.1.15 *Consider the identity functor $Id : \mathbf{S}^{ip} \to \mathbf{S}^{ip}$. Prove that $[\Pi_{\mathbf{S}^{ip}} Id]$ is the cpo with one element. Hint: let s be a cocontinuous section and D a Scott domain. Then there are two standard embeddings, in_l and in_r, of D in $D + D$, where $+$ is the coalesced sum. The condition on sections requires that $s(D + D) = in_l(s(D)) = in_r(s(D))$, but this forces $s(D) = \bot_D$.*

$F : D \to \mathbf{Cpo}^{ip}$, F functor, D cpo	$\Rightarrow$	$\Sigma_D F$, $[\Pi_D F]$ cpo's
$F : D \to \mathbf{S}^{ip}$, F cocont., D Scott domain	$\Rightarrow$	$\Sigma_D F$, $[\Pi_D F]$ Scott domains
$F : \mathbf{S}^{ip} \to \mathbf{S}^{ip}$, F cocont.	$\Rightarrow$	$[\Pi_{\mathbf{S}^{ip}} F] \cong [\Pi_{\mathbf{S}^{ip}_o} F]$ Scott domain

Figure 11.1: Dependent and second order types in $\mathbf{Cpo}^{ip}$ and $\mathbf{S}^{ip}$

Remark 11.1.16 *(1) Exercise 11.1.15 hints at the fact that cocontinuous sections satisfy certain* uniformity conditions*, namely the choice of the elements has to be invariant with respect to certain embeddings. In practice syntactically definable functors are "very" uniform so we can look for even stronger uniformity conditions in the model. Here is one that arises in the* stable case *(see chapter 12 and [Gir86]) and that leads to a "smaller" interpretation of certain types. Every section s satisfies the following uniformity condition:*

$$h : D \to E \text{ in } \mathbf{S}^{ip} \text{ implies } s(D) = (F(h))^{-}(s(E)) \tag{11.6}$$

This condition implies the standard condition in the continuous case. In the stable case one considers stable *injection projection pairs (cf. section 12.4) and the sections s are such that for all D, $s(D)$ is stable. (2) It can be proved that bifinite domains are not closed with respect to the $[\Pi_{\mathbf{Bif}} F]$ construction (see [Jun90]). The basic problem arises from the observation that $\mathbf{S}^{ip}_o$ does not need to satisfy property M (cf. definition 5.2.6).*

The following two exercises require the knowledge of stability theory and of coherence spaces (chapters 12 and 13). The first exercise witnesses the difference between the stable and the continuous interpretation. The second presents the uniformity condition as a requirement of stability.

Exercise 11.1.17 *(1) Show that, in the stable setting just described, the interpretation of $\forall t.t \to t$ is (isomorphic to) $\mathbf{O}$. (2) In contrast, show that in the continuous setting the interpretation of $\forall t.t \to t$ is infinite. Hints: For (1), consider a section s. Show that if $xe \in trace(s(E, \frown))$, then $x \subseteq \{e\}$; make use of two injections from E into $E \cup e'$, where e' is coherent with all the events of x. Show that $x \neq \perp$ with a similar method (e' being now incoherent with e). Show that if s is not $\perp$ for all D, then $s(\{e\}, \frown) = id$, and hence $s(D)$ is the identity everywhere. For (2), consider the (non-stable) functions defined by $s(D)(x) = x$ if x bounds at least n compact elements of D, and $s(D)(x) = \perp$ otherwise.*

Exercise 11.1.18 (Moggi) *Let s be a section satisfying the condition in the proof of theorem 11.1.14 and consisting of stable functions. Show that s satisfies the uniformity condition 11.6 iff s, viewed as a functor in the Grothendieck category, preserves pullbacks. Hints: (1) Show that $f : (D, x) \to (D', x')$ and $f' : (D, x) \to (D', x'')$ form the limit cone of a pullback diagram in the Grothendieck category iff $x = F(f)^-(x') \wedge F(f')^-(x'')$. (2) Show that for any stable injection-projection pair $f : D \to D'$, the pair of f and f forms the limit cone of a pullback diagram.*

11.2 Dependent and Second Order Types

We introduce the typing rules of the $\lambda P2$-calculus, a λ-calculus with dependent and second order types. We restrict our attention to the introduction and elimination rules for products. The syntactic categories of the $\lambda P2$-calculus are presented as follows.

Variables	$v ::= x \mid y \mid \ldots$
Contexts	$\Gamma ::= \phi \mid \Gamma, v : \sigma \mid \Gamma, v : K$
Kinds	$K ::= tp \mid \Pi v : \sigma.K$
Type Families	$\sigma ::= v \mid \Pi v : \sigma.\sigma \mid \Pi v : tp.\sigma \mid \lambda v : \sigma.\sigma \mid \sigma M$
Objects	$M ::= v \mid \lambda v : \sigma.M \mid \lambda v : tp.M \mid MM \mid M\sigma$.

Contexts, type families, and objects generalize the syntactic categories we have already defined in the simply typed case (cf. chapter 4). *Kinds* form a new syntactic category, which is used to classify type families, so, intuitively, kinds are the "types of types". The basic kind is tp which represents the collection of all types. More complex kinds are built using the Π construction and are employed to classify functions from types to the collection of types (type families). The formal system is based on the following *judgments*.

Well formed kind	$\Gamma \vdash K : kd$
Well formed type family	$\Gamma \vdash \sigma : K$
Well formed object	$\Gamma \vdash M : \sigma$.

The formal rules are displayed in figure 11.2. In the following we will use $A, B, \ldots$ as meta-symbols ranging over objects, type families, kinds, and a special constant kd which is introduced here just to have a uniform notation.

A well-formed context has always the shape $x_1 : A_1, \ldots, x_n : A_n$ where A_i is either a kind or a type (that is a type family of kind tp, but not a function over types). Note that A_i might actually depend on the previous variables. Syntactically this entails that the rule of exchange of premises is not derivable in the system; the order of hypotheses is important. Semantically we remark that a context cannot be simply interpreted as a product. We will see next that the product is replaced by the Grothendieck category (cf. exercise 11.1.2)

The formation rules for kinds are directly related to those for contexts, indeed we use $\Gamma \vdash tp : kd$ to state that the context Γ is well-formed. One can consider a slightly less synthetic presentation in which one adds a fourth judgment, say $\Gamma \vdash ok$, which asserts the well-formation of contexts.

We remark that not all premises in the context can be λ-abstracted. In particular, type families cannot be abstracted with respect to kinds, and objects can be abstracted only with respect to types and the kind tp. By convention we abbreviate $\Pi x : A.B$ with $A \to B$, whenever $x \notin FV(B)$.

In the $\lambda P2$-calculus it is not possible to type a *closed* type family $\lambda x : \sigma.\tau : \Pi x : \sigma.tp$ in such a way that τ actually depends on x. In the applications (e.g., see section 11.4) we enrich the calculus with constants such as $Prod.bool : nat \to tp$.

Finally, we note that the rules Π_I and Π_E for type families and objects follow the same pattern.

Kinds and types are assigned to type families and objects, respectively, modulo β-conversion (rules $(tp.Eq)$ and (Eq)). Formally, we define the relation $=$ as the symmetric and transitive closure of a relation of *parallel* β-reduction which is specified in figure 11.3. This is a suitable variation over the notion of parallel β-reduction that we have defined in figure 2.5 to prove the confluence of the untyped $\lambda\beta$-calculus. Note that the definition of the reduction relation does not rely on the typability of the terms. Indeed this is not necessary to obtain confluence as stated in the following.

Proposition 11.2.1 (confluence) *If $A \Rightarrow A'$ and $A \Rightarrow A''$ then there is B such that $A' \Rightarrow B$ and $A'' \Rightarrow B$.*

PROOF HINT. Show that if $A \Rightarrow A'$ and $B \Rightarrow B'$ then $A[B/x] \Rightarrow A'[B'/x]$. □

We state three useful properties of the $\lambda P2$-calculus. We omit the proofs which go by simple inductions on the length of the proof and the structure of the terms.

Proposition 11.2.2 *Type uniqueness: If $\Gamma \vdash A : B$ and $\Gamma \vdash A : B'$ then $B = B'$.*

Abstraction typing: If $\Gamma \vdash \lambda x : A.A' : \Pi x : B.C$ then $\Gamma, x : A \vdash A' : C$ and $A = B$.

Subject reduction: If $\Gamma \vdash A : B$ and $A \Rightarrow A'$ then $\Gamma \vdash A' : B$.

Let us briefly discuss two relevant extensions of the $\lambda P2$-calculus:

• When embedding logics or data structures in the $\lambda P2$-calculus it is often useful to include η-conversion as well (cf. sections 11.4 and 11.5):

$$(\eta) \quad \lambda x : A.(Bx) = B \quad x \notin FV(B) \ .$$

$$(K.\phi)\ \frac{}{\phi \vdash tp : kd} \qquad (K.kd)\ \frac{\Gamma \vdash K : kd \quad x \notin dom(\Gamma)}{\Gamma, x : K \vdash tp : kd}$$

$$(K.tp)\ \frac{\Gamma \vdash \sigma : tp \quad x \notin dom(\Gamma)}{\Gamma, x : \sigma \vdash tp : kd} \qquad (K.\Pi)\ \frac{\Gamma, x : \sigma \vdash K : kd \quad \Gamma \vdash \sigma : tp}{\Gamma \vdash \Pi x : \sigma.K : kd}$$

Well formed kind

$$(tp.Asmp)\ \frac{x : K \in \Gamma \quad \Gamma \vdash tp : kd}{\Gamma \vdash x : K} \qquad (tp.Eq)\ \frac{\Gamma \vdash \sigma : K \quad \Gamma \vdash K' : kd \quad K = K'}{\Gamma \vdash \sigma : K'}$$

$$(tp.\Pi)\ \frac{\Gamma, x : \sigma \vdash \tau : tp \quad \Gamma \vdash \sigma : tp}{\Gamma \vdash \Pi x : \sigma.\tau : tp} \qquad (tp.\Pi^2)\ \frac{\Gamma, x : tp \vdash \tau : tp}{\Gamma \vdash \Pi x : tp.\tau : tp}$$

$$(tp.\Pi_I)\ \frac{\Gamma, x : \sigma \vdash \tau : K \quad \Gamma \vdash \sigma : tp}{\Gamma \vdash \lambda x : \sigma.\tau : \Pi x : \sigma.K} \qquad (tp.\Pi_E)\ \frac{\Gamma \vdash \tau : \Pi x : \sigma.K \quad \Gamma \vdash M : \sigma}{\Gamma \vdash \tau M : K[M/x]}$$

Well formed type family

$$(Asmp)\ \frac{x : \sigma \in \Gamma \quad \Gamma \vdash tp : kd}{\Gamma \vdash x : \sigma} \qquad (Eq)\ \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash \tau : tp \quad \sigma = \tau}{\Gamma \vdash M : \tau}$$

$$(\Pi_I)\ \frac{\Gamma, x : \sigma \vdash M : \tau \quad \Gamma \vdash \sigma : tp}{\Gamma \vdash \lambda x : \sigma.M : \Pi x : \sigma.\tau} \qquad (\Pi_E)\ \frac{\Gamma \vdash M : \Pi x : \sigma.\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau[N/x]}$$

$$(\Pi^2_I)\ \frac{\Gamma, x : tp \vdash M : \tau}{\Gamma \vdash \lambda x : tp.M : \Pi x : tp.\tau} \qquad (\Pi^2_E)\ \frac{\Gamma \vdash M : \Pi x : tp.\tau \quad \Gamma \vdash \sigma : tp}{\Gamma \vdash M\sigma : \tau[\sigma/x]}$$

Well formed object

Figure 11.2: Typing rules for the $\lambda P2$-calculus

$$\frac{A \Rightarrow A' \quad B \Rightarrow B'}{(\lambda x : C.A)B \Rightarrow A'[B'/x]} \qquad \frac{A \Rightarrow A' \quad B \Rightarrow B'}{AB \Rightarrow A'B'}$$

$$\frac{A \Rightarrow A' \quad B \Rightarrow B'}{\lambda x : A.B \Rightarrow \lambda x : A'.B'} \qquad \frac{A \Rightarrow A' \quad B \Rightarrow B'}{\Pi x : A.B \Rightarrow \Pi x : A'.B'}$$

$$\frac{}{A \Rightarrow A} \qquad \frac{A \Rightarrow C \quad B \Rightarrow C}{A = B}$$

Figure 11.3: Parallel β-reduction and equality for the $\lambda P2$-calculus

The system with $\beta\eta$-conversion is still confluent and strongly normalizing but the proof of this fact is considerably harder than the one for β-conversion. A basic problem is that *confluence cannot be proved without appealing to typing.* Consider:

$$\begin{aligned} &N \equiv \lambda x : \sigma.(\lambda y : \tau.M)x \quad x \notin FV(M) \\ &N \to_\beta \lambda x : \sigma.M[x/y] \\ &N \to_\eta \lambda y : \tau.M \ . \end{aligned}$$

It is not possible to close the diagram unless σ and τ are convertible. This is proven by appealing to judgments of the shape $\Gamma \vdash \sigma = \tau : K$.

• The following rules can be used to formalize the Σ-construction on dependent types. Observe the introduction of the constructor $\langle \_, \_ \rangle$ and destructors fst, snd which generalize the familiar operators associated to the cartesian product.

$$(tp.\Sigma) \ \frac{\Gamma, x : \sigma \vdash \tau : tp}{\Gamma \vdash \Sigma x : \sigma.\tau : tp} \qquad (\Sigma_I) \ \frac{\Gamma \vdash M : \sigma \quad \Gamma, x : \sigma \vdash N : \tau}{\Gamma \vdash \langle M, N[M/x] \rangle : \Sigma x : \sigma.\tau}$$

$$(\Sigma_{E_1}) \ \frac{\Gamma \vdash M : \Sigma x : \sigma.\tau}{\Gamma \vdash fst\ M : \sigma} \qquad (\Sigma_{E_2}) \ \frac{\Gamma \vdash M : \Sigma x : \sigma.\tau}{\Gamma \vdash snd\ M : \tau[fstM/x]} \ .$$

Interpretation in Scott domains. We interpret the $\lambda P2$-calculus in the category of Scott domains and injection-projection pairs by appealing to the constructions introduced in section 11.1. The interpretation is given in a naive set-theoretical style, our goal being to suggest how the sum and product constructions can be used in an interpretation.

In first approximation the interpretation of a context Γ such that $\Gamma \vdash tp : kd$, is a category, say $[\![\Gamma]\!]$, the interpretation of tp is the category $\mathbf{S}^{ip}$ of Scott domains and injection-projection pairs, the interpretation of a type, $\Gamma \vdash \sigma : tp$, is a

functor $F = [\![\Gamma \vdash \sigma : tp]\!]$ from $[\![\Gamma]\!]$ to $\mathbf{S}^{ip}$, and the interpretation of a term, $\Gamma \vdash M : \sigma$, is a section of the Grothendieck fibration $p : \Sigma_{[\![\Gamma]\!]} F \to [\![\Gamma]\!]$. Note that the interpretations are inter-dependent, and they are defined in figure 11.4 by induction on the derivation of the judgment. We use a set-theoretical style, in a rigorous approach we should make sure that the defined objects exist in the domain-theoretical model. Another aspect which we ignore is the soundness of the equality rules. Indeed, one should verify that β-reduction is adequately modelled ([CGW88] carries on this verification for second order types).

We start with the trivial category 1, and we use the Grothendieck category to extend the context. The interpretation of a kind judgment is a functor from the context interpretation to **Cat**. We define the interpretation parametrically on $y \in [\![\Gamma]\!]$. Given a variable, say x, occurring in the well formed context Γ we write y_x for the projection of the x-th component of the vector $y \in [\![\Gamma]\!]$.

Exercise 11.2.3 *Extend the interpretation to handle the rules for dependent sum stated above.*

11.3 Types as Retractions

We present two approaches which are based on the interpretation of types as (particular) retractions over a domain. In the first approach, we develop the properties of finitary projections (cf. chapter 7) towards the interpretation of dependent, and second order types. In the second approach, we present a purely syntactic interpretation of the $\lambda P2$-calculus into the $\lambda\beta p$-calculus, which is a λ-calculus enriched with a constant p that plays the role of a retraction of all retractions.

In section 7.4, we have discussed how to represent countably based Scott domains as finitary projections over a universal domain U. In the following we briefly describe the construction of the operators Σ and Π in this framework (see [ABL86]). Suppose that U is a Scott domain such that:

$$\begin{array}{ll} U \times U \trianglelefteq U & \text{via } (\lambda(u,u').\langle u,u'\rangle, \lambda u.((fst\, u),(snd\, u))) : U \times U \to U \\ (U \to U) \trianglelefteq U & \text{via } (i,j) : (U \to U) \to U \end{array}$$

We also know that (see exercise 7.4.8):

$$FP(U) \trianglelefteq (U \to U) \quad \text{via } (id_{FP(U)}, \pi)\ .$$

We set $\underline{\pi} = i \circ \pi \circ j \in FP(U)$. We suppose the following correspondences:

- A projection $p \in FP(U)$ represents the domain $im(p)$.
- A function $f : U \to U$ such that $f = \underline{\pi} \circ f \circ p$ represents a cocontinuous functor from the domain $im(p)$ to the category $\mathbf{S}^{ip}$, where $f(d) = im(fd)$ and $f(d \leq d') = (id, f(d))$.

$(K.\phi)$	$[\![\phi]\!]$	$= 1$
$(K.kd)$	$[\![\Gamma, x : K]\!]$	$= \Sigma_{[\![\Gamma]\!]}[\![\Gamma \vdash K : kd]\!]$
$(K.tp)$	$[\![\Gamma, x : \sigma]\!]$	$= \Sigma_{[\![\Gamma]\!]}[\![\Gamma \vdash \sigma : tp]\!]$

Context interpretation

$(K.\phi, kd, tp)$	$[\![\Gamma \vdash tp : kd]\!](y)$	$= \mathbf{S}^{ip}$
$(K.\Pi)$	$[\![\Gamma \vdash \Pi x : \sigma.K : kd]\!](y)$	$= [\Pi_{Gy} \lambda y'.F(y, y')]$
		where: $Gy = [\![\Gamma \vdash \sigma : tp]\!](y)$
		and $F(y, y') = [\![\Gamma, x : \sigma \vdash K : kd]\!](y, y')$

Kind interpretation

$(tp.Asmp)$	$[\![\Gamma \vdash x : tp]\!](y)$	$= y_x$
$(tp.\Pi)$	$[\![\Gamma \vdash \Pi x : \sigma.\tau : tp]\!](y)$	$= [\Pi_{Gy} \lambda y'.F(y, y')]$
		where: $Gy = [\![\Gamma \vdash \sigma : tp]\!](y)$
		and $F(y, y') = [\![\Gamma, x : \sigma \vdash \tau : tp]\!](y, y')$
$(tp.\Pi^2)$	$[\![\Gamma \vdash \Pi x : tp.\tau : tp]\!](y)$	$= [\Pi_{\mathbf{S}^{ip}} \lambda y'.F(y, y')]$
		where: $F(y, y') = [\![\Gamma, x : tp \vdash \tau : tp]\!](y, y')$
$(tp.\Pi_I)$	$[\![\Gamma \vdash \lambda x : \sigma.\tau : \Pi x : \sigma.K]\!](y)$	$= \lambda y' \in Gy.([\![\Gamma, x : \sigma \vdash \tau : K]\!])(y, y')$
		where: $Gy = [\![\Gamma \vdash \sigma : tp]\!](y)$
$(tp.\Pi_E)$	$[\![\Gamma \vdash \tau M : K[M/x]]\!](y)$	$= ([\![\Gamma \vdash \tau : \Pi x : \sigma.K]\!](y))([\![\Gamma \vdash M : \sigma]\!](y))$

Type family interpretation

$(Asmp)$	$[\![\Gamma \vdash x : \sigma]\!](y)$	$= y_x$
(Π_I)	$[\![\Gamma \vdash \lambda x : \sigma.M : \Pi x : \sigma.\tau]\!](y)$	$= \lambda y' \in Gy.([\![\Gamma, x : \sigma \vdash M : \tau]\!])(y, y')$
		where: $Gy = [\![\Gamma \vdash \sigma : tp]\!](y)$
(Π_E)	$[\![\Gamma \vdash MN : \tau[N/x]]\!](y)$	$= ([\![\Gamma \vdash M : \Pi x : \sigma.\tau]\!](y))([\![\Gamma \vdash N : \sigma]\!](y))$
(Π_I^2)	$[\![\Gamma \vdash \lambda x : tp.M : \Pi x : tp.\tau]\!](y)$	$= \lambda y' \in \mathbf{S}^{ip}.([\![\Gamma, x : tp \vdash M : \tau]\!])(y, y')$
$(\Pi_E^2$	$[\![\Gamma \vdash M\sigma : \tau[\sigma/x]]\!](y)$	$= ([\![\Gamma \vdash M : \Pi x : tp.\tau)]\!](y))([\![\Gamma \vdash \sigma : tp]\!](y))$

Object interpretation

Figure 11.4: Interpretation of the $\lambda P2$-calculus in $\mathbf{S}^{ip}$

It has already been remarked in 7.4.10 that $FP(U)$ and $\mathbf{S}^{ip}$ (more precisely, the subcategory of countably based domains) are not equivalent categories as $FP(U)$ is just a poset. As a matter of fact we get a different model of the $\lambda P2$-calculus where, in particular, one can interpret the second order Σ-construction as a domain.

Definition 11.3.1 (Σ **and** Π **constructions in** $FP(U)$) *Let $p \in FP(U)$, and $f : U \to U$ be such that $f = \underline{\pi} \circ f \circ p$. We define:*

$$\begin{aligned} \Sigma_p f &= \lambda u.\langle p(fst\, u), (f(fst\, u))(snd\, u)\rangle : U \to U \\ \Pi_p f &= \lambda u.i(\lambda x.j(fx)((ju)(px)) : U \to U \ . \end{aligned}$$

Exercise 11.3.2 *Show that under the hypotheses of definition 11.3.1, $\Sigma_p f, \Pi_p f \in FP(U)$.*

When $f : im(p) \to \mathbf{S}^{ip}$ is regarded as a functor, the sum and product constructions defined in propositions 11.1.5 and 11.1.7, respectively, apply. In particular we have:

$$\begin{aligned} \Sigma_{im(p)} f &= \{(d, e) \mid pd = d \text{ and } (fd)e = e\} \\ [\Pi_{im(p)} f] &= \{\phi : U \to U \mid \phi \circ p = \phi \text{ and } \forall d\,((fd)(\phi d) = \phi d)\} \ . \end{aligned}$$

We can then show that $\Sigma_p f$ and $\Pi_p f$ are finitary projections representing the "right" domains.

Exercise 11.3.3 *Show that under the hypotheses of definition 11.3.1 the following isomorphims hold:*

$$im(\Sigma_p f) \cong \Sigma_{im(p)} f \quad \textit{and} \quad im(\Pi_p f) \cong [\Pi_{im(p)} f] \ .$$

Exercise 11.3.4 *Compute $\Pi_\pi Id$. Compare the corresponding domain with the one obtained in exercise 11.1.15.*

Exercise 11.3.5 *Consider the formal system for the $\lambda P2$-calculus with the identification $tp \equiv kd$. This system has been shown to be logically inconsistent (all types are inhabited) by Girard. However, not all terms are equated. To prove this fact propose an interpretation of the calculus in the domain of finitary projection. Hint: the finitary projection π represents the type of all types (see [ABL86]).*

We now turn to the syntactic approach. We present an extension of the untyped $\lambda\beta$-calculus with a constant p whose properties are displayed in figure 11.5 (by convention, let $P \circ Q$ stand for $\lambda x.P(Qx)$, with x fresh). The intention is to let p denote the retraction of all retractions. On this basis, (p_1) states that elements in the image of p are retractions, (p_2) entails that p is a retraction as $p \circ p = pp \circ pp = pp = p$, and (p_3) states that all retractions are in the image of p.

$$(p_1)\ \frac{}{(px)\circ(px)=px} \qquad (p_2)\ \frac{}{pp=p} \qquad (p_3)\ \frac{M\circ M=M}{pM=M}$$

Figure 11.5: Additional rules for the $\lambda\beta p$-calculus

$$\begin{array}{ll}
\langle kd\rangle & = p \\
\langle tp\rangle & = p \\
\langle x\rangle & = x \\
\langle \Pi x:A.B\rangle & = \lambda z.\lambda t.(\lambda x.\langle B\rangle)(\langle A\rangle t)(z(\langle A\rangle t)) \quad z,t\notin FV(A)\cup FV(B) \\
\langle \lambda x:A.B\rangle & = (\lambda x.\langle B\rangle)\circ\langle A\rangle \\
\langle AB\rangle & = \langle A\rangle\langle B\rangle
\end{array}$$

$$\begin{array}{ll}
\text{Suppose:} & \Gamma_i \equiv x_1:A_1,\ldots,x_i:A_i, i=1,\ldots,n. \\
\langle A\rangle^{\Gamma} & = \langle A\rangle[P_1/x_1,\ldots,P_n/x_n]. \\
P_{i+1} & = \langle A_{i+1}\rangle^{\Gamma_i} x_i \\
P_1 & = \langle A_1\rangle x_1
\end{array}$$

Figure 11.6: Translation of the $\lambda P2$-calculus into the $\lambda\beta p$-calculus

We want to show that: (i) every model of the $\lambda\beta p$-calculus is also a model of the $\lambda P2$-calculus, and (ii) there are models of the $\lambda\beta p$-calculus. Point (ii) is a a corollary of theorem 12.4.18. In particular, we will we will see that every reflexive object in the category of bifinite (stable) domains and stable morphisms (there are plenty of them) can be canonically extended to a model of the $\lambda\beta p$-calculus.

We remark that the finitary projection model presented above, although based on similar ideas, does not provide a model of the $\lambda\beta p$-calculus if we interpret (as it is natural) p as the projection π. The problem is that the rule (p_3) requires that *every* retraction is in π image (a similar problem would arise in models based on finitary retractions).

In order to address point (i), we exhibit a syntactic translation of the $\lambda P2$-calculus into the $\lambda\beta p$-calculus which preserves equality. By combining (i) and (ii) we can conclude that every model of the $\lambda\beta$-calculus based on bifinite stable domains, canonically provides a (non-trivial) interpretation of the $\lambda P2$-calculus.

Let us give some intuition for the interpretation. A type or a kind is rep-

resented as a retraction, say r. An object d has type r if $d = r(d)$. When interpreting the λ-abstraction $\lambda x : A.B$ the retraction $\langle A \rangle$ is used to coerce the argument to the right type. A similar game is played in the interpretation of $\Pi x : A.B$ which resembles $\Pi_p f$ in definition 11.3.1. Note that if $x \notin FV(B)$ then $\langle \Pi x : A.B \rangle = \lambda z.\langle B \rangle \circ z \circ \langle A \rangle$, which is the way to build a functional space in Karoubi envelope (cf. definition 4.6.8). Another special case is when $A \equiv tp$, then we obtain $\lambda t.\lambda z.\langle B \rangle[pt/x](z(pt))$. Here the type of the result "$\langle B \rangle[pt/x]$" depends on the input type "pt". The translation is defined in figure 11.6, it respects typing and reduction as stated in the following.

Proposition 11.3.6 *(1) If* $\Gamma \vdash A : B$ *then* $\langle A \rangle^\Gamma =_{\beta p} \langle B \rangle^\Gamma \langle A \rangle^\Gamma$.

(2) *If* $\Gamma \vdash A : B$ *and* $A \Rightarrow B$ *then* $\langle A \rangle^\Gamma =_{\beta p} \langle B \rangle^\Gamma$.

PROOF. In the first place we observe that $\langle A[B/x] \rangle^\Gamma =_{\beta p} \langle A \rangle^\Gamma[\langle B \rangle^\Gamma/x]$. Next we prove the two statements simultaneously by induction on the length of the typing proof. We consider some significative cases.

$(K.\phi)$ We apply axiom (p_2).

$(K.\Pi)$ Let $Q \equiv \langle \Pi x : \sigma.K \rangle^\Gamma$. We prove $pQ =_{\beta p} Q$ by showing $Q \circ Q =_{\beta p} Q$. To this end we expand the left hand side of the equation and apply the inductive hypotheses: $p\langle K \rangle^{\Gamma, x:\sigma} =_{\beta p} \langle K \rangle^{\Gamma, x:\sigma}$ and $p\langle \sigma \rangle^\Gamma =_{\beta p} \langle \sigma \rangle^\Gamma$.

$(tp.Asmp)$ There is a shorter proof of $\Gamma \vdash K : kd$. Then by induction hypothesis we know $p\langle K \rangle^\Gamma =_{\beta p} \langle K \rangle^\Gamma$. We conclude observing that $\langle x \rangle^\Gamma =_{\beta p} \langle K \rangle^\Gamma x$.

$(tp.Eq)$ There are shorter proofs of $\Gamma \vdash K : kd$ and $\Gamma \vdash K' : kd$. By confluence we know that K and K' have a common reduct. By applying the second part of the statement above we can conclude that $\langle K \rangle^\Gamma =_{\beta p} \langle K' \rangle^\Gamma$. By inductive hypothesis we know $\langle K \rangle^\Gamma \langle \sigma \rangle^\Gamma =_{\beta p} \langle \sigma \rangle^\Gamma$. Combining with the previous equation we get the desired result.

$(tp.\Pi_I)$ By expanding definitions as in the $(K.\Pi)$ case.

(Π_E) We observe:

$$\langle MN \rangle^\Gamma =_{\beta p} (\langle \Pi x : \sigma.\tau \rangle^\Gamma \langle M \rangle^\Gamma)(\langle N \rangle^\Gamma) =_{\beta p} \langle \tau[N/x] \rangle^\Gamma (\langle M \rangle^\Gamma \langle N \rangle^\Gamma) \ .$$

For the second part of the statement we proceed by induction on the typing and the derivation of a β-reduction. For instance consider the case $(\lambda x : A.B)C \to B[C/x]$. If $(\lambda x : A.B)C$ is typable in a context Γ then we can extract a proof that $\Gamma \vdash C : A$. By (1) we know $\langle A \rangle^\Gamma \langle C \rangle^\Gamma =_{\beta p} \langle C \rangle^\Gamma$. Hence we can compute $\langle (\lambda x : A.B)C \rangle^\Gamma =_{\beta p} (\lambda x.\langle B \rangle^\Gamma)(\langle A \rangle^\Gamma \langle C \rangle^\Gamma)$. Which is convertible to $\langle B \rangle^\Gamma[\langle C \rangle^\Gamma/x]$. □

11.4 System LF

The system LF corresponds to the fragment of the $\lambda P2$-calculus in which we drop second order types. Formally one has to remove the following rules: $(tp.\Pi^2)$, (Π_I^2), and (Π_E^2).

It has been shown that the system can faithfully encode a large variety of logical systems [AHMP95]. We will highlight some features of this approach by studying the encoding of a Hilbert style presentation of classical first-order logic with equality and arithmetic operators. Dependent products play a central role in this encoding. From this one may conclude that dependent products are more "expressive" than simple types. [1]

On the other hand from the view point of the length of the normalization procedure dependent types do not add any complexity. As a matter of fact we show that the strong normalization of system LF can be deduced from the strong normalization of the simply typed λ-calculus via a simple translation. [2]

Remark 11.4.1 *Kinds, type families, and objects in β-normal form have the following shapes where recursively the subterms are in β-normal form:*

$$\begin{array}{ll} \textit{Kind:} & \Pi x_1 : \sigma_1 \ldots \Pi x_n : \sigma_n . tp \\ \textit{Type family:} & \lambda x_1 : \sigma_1 \ldots \lambda x_n : \sigma_n . \Pi y_1 : \tau_1 \ldots y_m : \tau_m . x M_1 \ldots M_k \\ \textit{Object:} & \lambda x_1 : \sigma_1 \ldots \lambda x_n : \sigma_n . x M_1 \ldots M_k \ . \end{array}$$

In order to define precise encodings of logics in LF it is useful to introduce the notion of *canonical* form. Roughly a term is in canonical form if it is in β normal form and η-expansion is performed as much as possible. Canonical forms can be regarded as a way to avoid the problematic introduction of full $\beta\eta$-conversion.

Definition 11.4.2 *The* arity *of a type or kind is the number of Π's in the prefix of its β-normal form (which is to say the number of arguments). Let $\Gamma \vdash A : B$ be a derivable judgment. The arity of a variable occurring in A or B is the arity of its type or kind.*

Definition 11.4.3 *Let $\Gamma \vdash A : B$ be a derivable judgment. The term A is in* canonical form *if it is in β-normal form and all variable occurrences in A are fully applied, where we say that a variable occurrence is* fully applied *if it is applied to a number of arguments equal to the variable's arity.*

[1] It is known that the validity of a sentence is a decidable problem for propositional logic and an undecidable one for first-order logic. Dependent types can be connected to predicate logic in the same way simple types were connected to propositional logic in section 4.1.

[2] From a logical view point this relates to the well-known fact that the cut-elimination procedures in propositional and first-order logic have the same complexity.

$$\begin{array}{ll} \text{Individuals} & t ::= x \parallel 0 \parallel s(t) \\ \text{Formulas} & \phi ::= t = t \parallel \neg\phi \parallel \phi \supset \phi \parallel \forall x.\phi \end{array}$$

$$(eq_1)\ \frac{}{t = t} \qquad (eq_2)\ \frac{t = t'}{t' = t}$$

$$(eq_3)\ \frac{t = t' \quad t' = t''}{t = t''} \qquad (eq_4)\ \frac{t = t'}{\phi[t/x] = \phi[t'/x]}$$

$$(pp_1)\ \frac{}{\phi \supset (\psi \supset \phi)} \qquad (pp_2)\ \frac{}{(\phi \supset (\psi \supset \chi)) \supset ((\phi \supset \psi) \supset (\phi \supset \chi))}$$

$$(pp_3)\ \frac{}{(\neg\phi \supset \neg\psi) \supset (\psi \supset \phi)} \qquad (mp)\ \frac{\phi \supset \psi \quad \phi}{\psi}$$

$$(pc_1)\ \frac{\phi}{\forall x.\phi} \qquad (pc_2)\ \frac{\forall x.\phi}{\phi[t/x]}$$

Figure 11.7: First-order logic with equality

In figure 11.7 we give a presentation of classical first-order logic (FOL) with equality and arithmetic operators. In figure 11.8 we encode the language in the system LF. To this end we build a context Γ_{FOL} composed of:

- The declaration of two new types ι, o corresponding to the collection of individuals and formulas, respectively.
- The declaration of objects $\hat{0}, \hat{s}$ corresponding to the arithmetic operators and objects $\hat{=}, \hat{\supset}, \hat{\neg}, \hat{\forall}$ corresponding to the logical operators.

Next we define a function $\lceil \_ \rceil$ that translates terms into objects of type ι and formulas into objects of type o. Note in particular that:

- Variables are identified with the variables of system LF.
- λ-abstraction is used to encode the quantifier $\forall$.

These features are essential to inherit the definitions of α-renaming and substitution available in the meta-theory, i.e. in LF. The correspondence between the language of FOL and its encoding in LF is quite good.

Proposition 11.4.4 *There is a bijective correspondence between terms (formulas) having free variables in $x_1, \ldots, x_n$ and terms M in canonical form such that $\Gamma^{syn}_{FOL}, x_1 : \iota, \ldots, x_n : \iota \vdash M : \iota$ ($\Gamma^{syn}_{FOL}, x_1 : \iota, \ldots, x_n : \iota \vdash M : o$).*

A second task concerns the encoding of the proof rules. The complete definition is displayed in figure 11.9. The basic judgment in FOL is that a formula

$$\Gamma^{syn}_{FOL} \left\{ \begin{array}{lll} \text{Constant types} & \iota, o : tp & \\ \text{Constant terms} & \hat{0} : \iota & \hat{s} : \iota \to \iota \\ & \hat{=} : \iota \to \iota \to o & \hat{\supset} : o \to o \to o \\ & \hat{\neg} : o \to o & \hat{\forall} : (\iota \to o) \to o \end{array} \right.$$

$$\begin{array}{llll} \lceil x \rceil & = x & \lceil 0 \rceil & = \hat{0} \\ \lceil s(t) \rceil & = \hat{s}\lceil t \rceil & & \\ \lceil t = t' \rceil & = \hat{=}\lceil t \rceil \lceil t' \rceil & \lceil \neg t \rceil & = \hat{\neg}\lceil t \rceil \\ \lceil \phi \supset \phi' \rceil & = \hat{\supset}\lceil \phi \rceil \lceil \phi' \rceil & \lceil \forall x.\phi \rceil & = \hat{\forall}(\lambda x : \iota.\lceil \phi \rceil) \end{array}$$

Figure 11.8: Coding FOL language in LF

holds, say $\vdash \phi$. Correspondingly we introduce a *dependent type* $T : o \to tp$. This is the point where dependent types do play a role! We also note that the rule $(tp.\Pi_E)$ is used to type the proof encodings. The basic idea is to introduce a series of constants which correspond to the proof rules in such a way that objects of type $T(\lceil \phi \rceil)$ relate to proofs of the formula ϕ. The property of the proof encoding can be stated as follows. [3]

Proposition 11.4.5 *There is a bijective correspondence between proofs of a formula ϕ from the assumptions $\phi_1, \ldots \phi_m$ and with free variables $x_1, \ldots, x_n$, and terms M in canonical form such that:*

$$\Gamma^{syn}_{FOL}, \Gamma^{rl}_{FOL}, x_1 : \iota, \ldots, x_n : \iota, y_1 : T(\lceil \phi_1 \rceil), \ldots, y_m : T(\lceil \phi_m \rceil) \vdash M : T(\lceil \phi \rceil) \ .$$

For instance, to the proof $\frac{x=x}{\forall x.(x=x)}$ we associate the term $pc_1(\lambda x : \iota.\hat{=}xx)(eq_1)$.

Next we turn to the strong normalization problem for the system LF. This is proven via a translation in the simply typed λ-calculus which is specified in figure 11.10. The function t applies to kinds and type families whereas the function $|\_|$ applies to type families and objects. The function t forgets the type dependency by replacing every variable by the ground type o and ignoring the argument of a type family. The function $|\_|$ reflects all possible reductions of the LF term. In order to translate terms of the shape $\Pi x : A.B$ we suppose that the simply typed λ-calculus is enriched with a family of constants π having type $o \to (t(A) \to o) \to o$. In the first place, we observe some syntactic properties of these translations.

Lemma 11.4.6 *If M is an object then: (1) $t(A[M/x]) \equiv t(A)$, and (2) $|A[M/x]| \equiv |A|[|M|/x]$.*

[3] Detailed proofs for propositions 11.4.4 and 11.4.5 can be found in [HHP93].

Judgment		$T : o \to tp$
Rules	Γ^{rl}_{FOL}	$\left\{ \begin{array}{l} eq_1 : \Pi x : i.T(\hat{=}xx) \\ eq_2 : \Pi x, y : i.(T(\hat{=}xy) \to T(\hat{=}yx)) \\ eq_3 : \Pi x, y, z : i.(T(\hat{=}xy) \to T(\hat{=}yz) \to T(\hat{=}xz)) \\ eq_4 : \Pi f : \iota \to o.\Pi x, y : i.(T(\hat{=}xy) \to T(\hat{=}(fx)(fy))) \\ pp_1 : \Pi f, g : o.T(f\hat{\supset}g\hat{\supset}h) \\ pp_2 : \Pi f, g, h : o.T((f\hat{\supset}(g\hat{\supset}h))\hat{\supset}((f\hat{\supset}g)\hat{\supset}(f \supset h))) \\ pp_3 : \Pi f, g : o.T((\hat{\neg}f\hat{\supset}\hat{\neg}g)\hat{\supset}(g\hat{\supset}f)) \\ mp : \Pi f, g : o.T(f\hat{\supset}g) \to T(f) \to T(g) \\ pc_1 : \Pi F : \iota \to o.(\Pi x : \iota.T(Fx)) \to T(\hat{\forall}F) \\ pc_2 : \Pi F : \iota \to o.\Pi x : \iota.T(\hat{\forall}F) \to T(Fx) \end{array} \right.$

Figure 11.9: Coding FOL proof rules in LF

$$
\begin{array}{ll}
t(tp) & = o \\
t(x) & = o \\
t(\Pi x : A.B) & = t(A) \to t(B) \\
t(\lambda x : A.B) & = t(B) \\
t(AB) & = t(A)
\end{array}
$$

$$
\begin{array}{ll}
|x| & = x \\
|AB| & = |A||B| \\
|\Pi x : A.B| & = \pi|A|(\lambda x : t(A).|B|) \\
|\lambda x : A.B| & = (\lambda y : o.\lambda x : t(A).|B|)|A| \quad (y \text{ fresh})
\end{array}
$$

Figure 11.10: Translation of LF in the simply typed λ-calculus

PROOF HINT. By induction on the structure of A. □

Lemma 11.4.7 *If $\Gamma \vdash A : B$ and $A \Rightarrow A'$, where A is a kind or a type family, then $t(A) \equiv t(A')$.*

PROOF. By induction on the proof of the reduction. In the basic case $(\lambda x : A.B)C \to B[C/x]$ we use the fact that, by the typability hypothesis, C is an object. □

The translations t and $|\_|$ preserve typing.

Proposition 11.4.8 *If $\Gamma \vdash A : B$ and $B \neq kd$ then $t(\Gamma) \vdash |A| : t(B)$, where $t(x_1 : A_1, \ldots, x_n : A_n) \equiv x_1 : t(A_1), \ldots, x_n : t(A_n)$.*

PROOF HINT. By induction on the length of the proof. □

Finally we can show that the translation reflects reductions, which, by the strong normalization of the simply typed λ-calculus, implies immediately the strong normalization of system LF.

Theorem 11.4.9 *If $\Gamma \vdash A : B$, $B \neq kd$, $A \Rightarrow A'$, and in the reduction $A \Rightarrow A'$ we find at least one β-reduction, then $|A| \to_\beta^+ |A'|$.*

PROOF. By induction on the derivation of $A \Rightarrow A'$. For instance, suppose we derive $(\lambda x : A.B)C \Rightarrow A'[C'/x]$ from $B \Rightarrow B'$ and $C \Rightarrow C'$. Then:

$$\begin{aligned} |(\lambda x : A.B)C| &= ((\lambda y : o.\lambda x : t(A).|B|)|A|)|C| \\ &\to (\lambda x : t(A).|B|)|C| \\ &\to^+ (\lambda x : t(A).|B'|)|C'| \quad \text{by induction hypothesis} \\ &\to |B'|[|C'|/x] = |B'[C'/x]| \quad \text{by lemma 11.4.6 .} \end{aligned}$$

□

Remark 11.4.10 *By combining the results on confluence and strong normalization it is possible to prove that it is decidable if a judgment is derivable in the system LF.*

11.5 System F

System F is the fragment of the $\lambda P2$-calculus where dependent types and type families are removed. Formally we eliminate the rules: $(K.\Pi)$, $(tp.\Pi_I)$, and $(tp.\Pi_E)$. With these restrictions, types cannot depend on objects and the equality rules $(tp.Eq)$ and (Eq) can be dispensed with, as type equality becomes α-conversion. Note that in the type $\Pi x : \sigma.\tau$, the type τ never depends on x and therefore we can simply write $\sigma \to \tau$. Finally we remark that the rules for the

$$(Asmp)\quad \frac{x:\sigma\in\Gamma}{\Gamma\vdash x:\sigma}$$

$$(\to_I)\quad \frac{\Gamma,x:\sigma\vdash M:\tau}{\Gamma\vdash \lambda x:\sigma.M:\sigma\to\tau} \qquad (\to_E)\quad \frac{\Gamma\vdash M:\sigma\to\tau \quad \Gamma\vdash N:\sigma}{\Gamma\vdash MN:\tau}$$

$$(\forall_I)\quad \frac{\Gamma\vdash M:\sigma \quad t\notin FV_t(\Gamma)}{\Gamma\vdash \lambda t.M:\forall t.\sigma} \qquad (\forall_E)\quad \frac{\Gamma\vdash M:\forall t.\sigma}{\Gamma\vdash M\tau:\sigma[\tau/t]}$$

Figure 11.11: Typing rules for system F

kind (and context) formation are redundant. Namely we can represent a context as a list $x_1:\sigma_1,\ldots,x_n:\sigma_n$ (as in the simply typed λ-calculus), where the types σ_i may depend on type variables. According to these remarks we give a more compact presentation of system F. Since we have eliminated the kinds, we need some notation to distinguish between type variables (i.e. variables of type tp) and term variables (i.e. variables of type σ, where σ has kind tp): we denote the former with $t,s,\ldots$ and the latter with $x,y,\ldots$ Terms and types are defined as follows:

$$\begin{array}{lll} \text{Types} & tv & ::= t \parallel s \parallel \ldots \\ & \sigma & ::= tv \parallel \sigma\to\sigma \parallel \forall tv.\sigma \\ \text{Terms} & v & ::= x \parallel y \parallel \ldots \\ & M & ::= v \parallel \lambda v:\sigma.M \parallel MM \mid \lambda tv.M \parallel M\sigma\ . \end{array}$$

Note that the type of all types is never explicitly mentioned. $\forall t.\ldots$ is an abbreviation for $\Pi t:tp.\ldots$ and $\lambda t.\ldots$ is an abbreviation for $\lambda t:tp.\ldots$.

A context Γ is a list $x_1:\sigma_1,\ldots,x_n:\sigma_n$, so the type variables declarations are left implicit. We denote with $FV_t(\Gamma)$ the collection of type variables that occur free in types occurring in Γ. Derivable typing judgments are specified in figure 11.11. Mutatis mutandis, the system is equivalent to the one presented in section 11.2.

Exercise 11.5.1 *Show that in system F $\beta\eta$-reduction is locally confluent on well-typed terms.*

The system F was introduced by Girard [Gir72] as a tool for the study of the cut-elimination procedure in second order arithmetic (PA_2), more precisely the normalization of system F implies the termination of the cut-elimination procedure in PA_2. By relying on this strong connection between system F and PA_2 it was proven that all functions that can be shown to be total in PA_2 are *representable* in system F. This is a huge collection of total recursive functions that

goes well beyond the primitive recursive functions. System F was later rediscovered by Reynolds [Rey74] as a concise calculus of *type parametric* functions. In this section we illustrate the rich type structure of system F by presenting a systematic method to code finite *free algebras* and *iterative functions* defined on them.

In the following an algebra $\mathcal{S}$ is a sort S equipped with a t-uple of constructors:

$$f_i^{n_i} : \underbrace{S \times \cdots \times S}_{n_i \ times} \to S \text{ for } i = 1, \ldots, k, k \geq 0, n_i \geq 0 \ .$$

We inductively define a collection of total computable functions over the *ground terms* of the algebra as follows.

Definition 11.5.2 *The collection of* iterative *functions $f : S^n \to S$ over an algebra $\mathcal{S}$ is the smallest set such that:*

- *The basic functions $f_i^{n_i}$, constant functions, and projection functions are iterative functions.*
- *The set is closed under composition. If $f_1 : S^m \to S, \ldots, f_n : S^m \to S$, and $g : S^n \to S$ are iterative then $\lambda \vec{x}.g(f_1(\vec{x}), \ldots, f_n(\vec{x}))$ is iterative.*
- *The set is closed under iteration. If $h_i : S^{n_i+m} \to S$ are iterative functions for $i = 1, \ldots, k$ then the function $f : S^{m+1} \to S$ defined by the following equations is iterative.*

$$f(\vec{x}, f_i(\vec{y})) = h_i(\vec{x}, f(\vec{x}, y_1), \ldots, f(\vec{x}, y_{n_i})) \quad i = 1, \ldots, k \ .$$

Iterative definitions, generalize to arbitrary algebras primitive recursive definitions (cf. appendix A). The basic idea is to define a function by induction on the structure of a closed term, hence we have an equation for every function of the algebra.

Exercise 11.5.3 *Consider the algebra of natural numbers $(\omega, s^1, 0^0)$. Show that the iterative functions coincide with the primitive recursive ones. Hint: the definitions by primitive recursion are apparently more general but they can be simulated using pairing and projections.*

Definition 11.5.4 (coding) *In figure 11.12 we associate to an algebra $\mathcal{S}$ a type σ of system F, and to a ground term a of the algebra a closed term $\underline{a}$ of type σ.*

Example 11.5.5 *If we apply the coding method to the algebra of natural numbers defined in exercise 11.5.3 we obtain the type $\forall t.(t \to t) \to (t \to t)$. The term $s(\cdots(s0)\cdots)$ can be represented by the term $\lambda t.\lambda f : t \to t.\lambda x : t.f(\cdots(fx)\cdots)$, which is a polymorphic version of Church numerals.*

Given: $S, f_i^{n_i} : \underbrace{S \times \cdots \times S}_{n_i \ times} \to S, i = 1, \ldots, k$

Let: $\sigma \equiv \forall t.\tau_1 \to \cdots \to \tau_k \to t$

where: $\tau_i \equiv \underbrace{t \to \cdots \to t}_{n_i \ times} \to t$

$$\underline{f_i^n(a_1, \ldots, a_n)} \quad = \lambda t.\lambda x_1 : \tau_1 \ldots \lambda x_k : \tau_k.x_i(\underline{a_1} t\vec{x}) \cdots (\underline{a_n} t\vec{x})$$

Figure 11.12: Coding algebras in system F

Exercise 11.5.6 *Explicit the coding of the following algebras: the algebra with no operation, the algebra with two* 0*-ary operations, the algebra of binary trees* $(T, nil^0, couple^2)$.

Proposition 11.5.7 *There is a bijective correspondence between the ground terms of the algebra* $\mathcal{S}$ *and the closed terms of type* σ *modulo* $\beta\eta$*-conversion.*

PROOF. Let M be a closed term in β-normal form of type σ. Then M has to have the shape:

$$M \equiv \lambda t.\lambda x_1 : \tau_1 \ldots \lambda x_i : \tau_i.M' \quad i \leq k \ .$$

If $i < k$ and M' is not a λ-abstraction then M' has the shape $(\cdots (x_j M_1) \cdots M_h)$ and so we can η-expand M' without introducing a β-redex. By iterated η-expansions we arrive at a term in β normal form of the shape

$$\lambda t.\lambda x_1 : \tau_1 \ldots \lambda x_k : \tau_k.M'' \ .$$

where M'' has type t, it is in β normal form, and may include free variables $x_1, \ldots, x_k$. We claim that M'' cannot contain a λ-abstraction:

- A λ-abstraction on the left of an application would contradict the hypothesis that M is in β normal form.
- A λ-abstraction on the right of an application is incompatible with the "first-order" types of the variables τ_i.

We have shown that a closed term of type σ is determined up to $\beta\eta$ conversion by a term M'' which is a well-typed combination of the variables x_i, for $i = 1, \ldots, k$. Since each variable corresponds to a constructor of the algebra we can conclude that there is a unique ground term of the algebra which corresponds to M''. □

Having fixed the representation of ground terms let us turn to the representation of functions.

Definition 11.5.8 *A function $f : S^n \to S$ is representable (with respect to the coding defined in figure 11.12) if there is a closed term $M : \sigma^n \to \sigma$, such that for any ground term $\vec{a}$, $M\underline{\vec{a}} =_{\beta\eta} \underline{f(\vec{a})}$.*

Proposition 11.5.9 *The iterative functions over an algebra $\mathcal{S}$ are representable.*

PROOF. We proceed by induction on the definition of iterative function. The only non-trivial case is iteration. Let $h_i : S^{n_i+m} \to S$ be iterative functions for $i = 1, \ldots, k$, and the function $f : S^{m+1} \to S$ be defined by:

$$f(\vec{x}, f_i(\vec{y})) = h_i(\vec{x}, f(\vec{x}, y_1), \ldots, f(\vec{x}, y_{n_i})) \quad i = 1, \ldots, k \tag{11.7}$$

where $\vec{x} \equiv x_1, \ldots, x_m$. We represent f with the function:

$$\underline{f} \equiv \lambda x_1 : \sigma. \ldots \lambda x_m : \sigma. \lambda x : \sigma. x\sigma(\underline{h_1}\vec{x}) \cdots (\underline{h_k}\vec{x})$$

where we know inductively that $\underline{h_i}$ represents h_i. Note that iteration is already built into the representation of the data. We prove by induction on the structure of a ground term a that for any vector of ground terms $\vec{b}$, $\underline{f}\underline{\vec{b}}\underline{a} =_{\beta\eta} \underline{f(\vec{b}, a)}$.

- If $a \equiv f_i^0$ then $\underline{f}\underline{\vec{b}}\underline{f_i^0} \to^* \underline{f_i^0}\sigma(\underline{h_1}\underline{\vec{b}}) \cdots (\underline{h_k}\underline{\vec{b}}) \to^* \underline{h_i}\underline{\vec{b}} = \underline{h_i(\vec{b})}$, the last step holds by induction hypothesis on h_i.
- If $a \equiv f_i^n(a_1, \ldots, a_n)$ then $f(\vec{b}, f_i(a_1, \ldots, a_n)) = h_i(\vec{b}, f(\vec{b}, a_1), \ldots, f(\vec{b}, a_n))$, by equation 11.7. Then by induction hypothesis on h_i:

$$\underline{f(\vec{b}, f_i(a_1, \ldots, a_n))} = \underline{h_i(\vec{b}, f(\vec{b}, a_1), \ldots, f(\vec{b}, a_n))} = \underline{h_i}\underline{\vec{b}}\underline{f(\vec{b}, a_1)} \ldots \underline{f(\vec{b}, a_n)} \ .$$

On the other hand we compute:

$$\begin{aligned} &\underline{f}\underline{\vec{b}}\underline{f_i^n(a_1, \ldots, a_n)} \\ &\to \underline{f_i^n(a_1, \ldots, a_n)}\sigma(\underline{h_1}\underline{\vec{b}}) \cdots (\underline{h_k}\underline{\vec{b}}) \\ &\to (\underline{h_i}\underline{\vec{b}})(\underline{a_1}\sigma(\underline{h_1}\underline{\vec{b}}) \cdots (\underline{h_k}\underline{\vec{b}})) \cdots (\underline{a_n}\sigma(\underline{h_1}\underline{\vec{b}}) \cdots (\underline{h_k}\underline{\vec{b}})) \ . \end{aligned}$$

and we observe that by induction hypothesis on a:

$$\underline{f(\vec{b}, a_i)} = \underline{f}\underline{\vec{b}}\underline{a_i} = \underline{a_i}\sigma(\underline{h_1}\underline{\vec{b}}) \cdots (\underline{h_k}\underline{\vec{b}})) \ .$$

□

Exercise 11.5.10 *Consider the case of algebras which are defined parametrically with respect to a collection of data. For instance $List(D)$ is the algebra of lists whose elements belong to the set D. This algebra is equipped with the constructors $nil : List(D)$ and $cons : D \times List(D) \to D$. Define iterative functions over $List(D)$ and show that these functions can be represented in system F for a suitable embedding of the ground terms in system F. Hint: The sort $List(D)$ is coded by the type $\forall t.t \to (t \to r \to t) \to t$, where r is a type variable, and generic elements in $List(D)$ are represented by (free) variables of type r.*

In system F it is also possible to give *weak* representations of common type constructors. We explain the weakness of the representation in the following example concerning products.

Example 11.5.11 *For σ, τ types of system F define:*

$$\sigma \underline{\times} \tau \equiv \forall t.(\sigma \to \tau \to t) \to t \ .$$

Pairing and projections terms can be defined as follows:

$$\begin{array}{ll} \langle M, N \rangle & = \lambda t.\lambda f : \sigma \to \tau \to t.fMN \\ \pi_1 M & = M\sigma(\lambda x : \sigma.\lambda y : \tau.x) \\ \pi_2 M & = M\tau(\lambda x : \sigma.\lambda y : \tau.y) \ . \end{array}$$

Note that $\pi_i\langle M_1, M_2\rangle =_{\beta\eta} M_i$ but pairing is not surjective, i.e. $\langle \pi_1 M, \pi_2 M\rangle \neq_{\beta\eta} M$.

Exercise 11.5.12 *Study the properties of the following codings of sum and existential:*

$$\begin{array}{ll} \sigma \underline{+} \tau & = \forall t.(\sigma \to t) \to (\tau \to t) \to t \\ \underline{\exists} t.\sigma & = \forall s.(\forall t.\sigma \to s) \to s \ . \end{array}$$

We conclude by proving the core of Girard's celebrated result: all terms typable in system F strongly normalize. The proof is based on the notion of *reducibility candidate* already considered in definition 3.5.13 and in the adequacy proof of section 8.2. In order to make notation lighter we will work with untyped terms obtained from the *erasure* of well-typed terms.

Definition 11.5.13 *The* erasure *function er takes a typed term and returns an untyped λ-term. It is defined by induction on the structure of the term as follows:*

$$\begin{array}{lll} er(x) = x & er(\lambda x : \sigma.M) = \lambda x.er(M) & er(MN) = er(M)er(N) \\ er(\lambda t.M) = er(M) & er(M\tau) = er(M) \ . & \end{array}$$

In system F we distinguish two flavours of β-reduction: the one involving a redex $(\lambda x : \sigma.M)N$ which we call simply β and the one involving a redex $(\lambda t.M)\sigma$ which we call β_t. Erasing type information may eliminate some reductions of the shape $(\lambda t.M)\sigma \to M[\sigma/t]$, however this does not affect the strong normalization property as shown in the following.

Proposition 11.5.14 *Let M be a well-typed term in system F. Then:*

(1) *If $M \to_\beta N$ then $er(M) \to_\beta er(N)$.*

(2) *If $M \to_{\beta_t} N$ then $er(M) \equiv er(N)$.*

(3) *If M diverges then $er(M)$ diverges.*

PROOF. We leave (1-2) to the reader. For (3), we observe that sequences of β_t-reductions always terminate. Hence we can extract an infinite reduction of $er(M)$ from an infinite reduction of M. □

Definition 11.5.15 (reducibility candidate) *Let SN be the collection of untyped $\lambda\beta$-strongly normalizable terms. A set $X \subseteq SN$ is a reducibility candidate if:*

(1) *$Q_i \in SN$, $i = 1, \ldots, n$, $n \geq 0$ implies $xQ_1, \ldots, Q_n \in X$.*

(2) *$P[Q/x]Q_1, \ldots, Q_n \in X$ and $Q \in SN$ implies $(\lambda x.P)QQ_1, \ldots, Q_n \in X$.*

We denote with RC the collection of reducibility candidates and we abbreviate $Q_1, \ldots, Q_n$ with $\vec{Q}$.

Proposition 11.5.16 *(1) The set SN is a reducibility candidate.*

(2) *If $X \in RC$ then $X \neq \emptyset$.*

(3) *The collection RC is closed under arbitrary intersections.*

(4) *If $X, Y \in RC$ then the following set is a reducibility candidate:*

$$X \Rightarrow Y = \{M \mid \forall N \in X\,(MN \in Y)\}\ .$$

PROOF. (1) We observe that $P[Q/x]\vec{Q} \in SN$ and $Q \in SN$ implies $(\lambda x.P)Q\vec{Q} \in SN$. Proceed by induction on $ln(P)+ln(Q)+ln(Q_1)+\cdots+ln(Q_n)$, where $ln(P)$ is the length of the longest reduction.

(2) By definition $x \in X$.

(3) Immediate.

(4) Here we see the use of the vector $\vec{Q}$. For instance let us consider the second condition. To show $(\lambda x.P)Q\vec{Q} \in X \Rightarrow Y$ observe $\forall Q' \in X\,(P[Q/x]\vec{Q}Q' \in Y)$ since by hypothesis $P[Q/x]\vec{Q} \in X \Rightarrow Y$. □

Definition 11.5.17 *Given a type environment $\eta : Tvar \to RC$ we interpret types as follows:*

$$\begin{array}{ll} [\![t]\!]\eta & = \eta(t) \\ [\![\sigma \to \tau]\!]\eta & = [\![\sigma]\!]\eta \Rightarrow [\![\tau]\!]\eta \\ [\![\forall t.\sigma]\!]\eta & = \bigcap_{X \in RC} [\![\sigma]\!]\eta[X/t]\ . \end{array}$$

Theorem 11.5.18 (strong normalization of system F) *Given an arbitrary type environment η, and a derivable judgment $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \tau$, if $P_i \in [\![\sigma_i]\!]\eta$, for $i = 1, \ldots, n$ then $er(M)[P_1/x_1, \ldots, P_n/x_n] \in [\![\tau]\!]\eta$.*

PROOF. We abbreviate $[P_1/x_1, \ldots, P_n/x_n]$ with $[\vec{P}/\vec{x}]$. We proceed by induction on the length of the typing proof. The case $(Asmp)$ follows by definition.

($\to_I$) We have to show $\lambda x.er(M)[\vec{P}/\vec{x}] \in [\![\sigma \to \tau]\!]\eta$. By inductive hypothesis we know $er(M)[\vec{P}/\vec{x}][P/x] \in [\![\tau]\!]\eta$, for all $P \in [\![\sigma]\!]\eta$. We conclude using property (2) of reducibility candidates.

($\to_E$) By the definition of $\Rightarrow$.

($\forall_I$) We have to show $er(M)[\vec{P}/\vec{x}] \in \bigcap_{X \in RC}[\![\tau]\!]\eta[X/t]$. By the side condition on the typing rule we know $[\![\sigma_i]\!]\eta = [\![\sigma_i]\!]\eta[X/t]$, for an arbitrary $X \in RC$. By inductive hypothesis $er(M)[\vec{P}/\vec{x}] \in [\![\tau]\!]\eta[X/t]$, for an arbitrary $X \in RC$.

($\forall_E$) We have to show $er(M)[\vec{P}/\vec{x}] \in [\![\tau]\!]\eta[[\![\sigma]\!]\eta/t]$. By inductive hypothesis $er(M)[\vec{P}/\vec{x}] \in \bigcap_{X \in RC}[\![\tau]\!]\eta[X/t]$. Pick up $X = [\![\sigma]\!]\eta$. □

The formal statement of theorem 11.5.18 can be regarded as a syntactic version of the fundamental lemma of (unary) logical relations (cf. 4.5.3). The following exercises present two variations over this result.

Exercise 11.5.19 *We say that a set X of untyped λ-terms is saturated (or closed by head expansion) if $P[Q/x]Q_1, \ldots, Q_n \in X$ implies $(\lambda x.P)QQ_1, \ldots, Q_n \in X$. Following definition 11.5.17 associate a saturated set to every type and prove the analogous of theorem 11.5.18.*

Exercise 11.5.20 *We say that a term is* neutral *if it does not start with a λ-abstraction. The collection RC' (cf. [GLT89]) is given by the sets X of strongly normalizing terms satisfying the following conditions:*

(1) *$M \in X$ and $M \to_\beta M'$ implies $M' \in X$.*

(2) *M neutral and $\forall M'(M \to_\beta M' \Rightarrow M' \in X)$ implies $M \in X$.*

Carry on the strong normalization proof using the collection RC'.

Exercise 11.5.21 *Extend the strong normalization results for system F to $\beta\eta$-reduction, where the η rule for type abstraction is: $\lambda t.Mt \to M \quad t \notin FV(M)$.*

Remark 11.5.22 *Note that η-expansion in system F does not normalize, as: $\lambda x : \forall t.t.x \to \lambda x : \forall t.t.\lambda t.xt \to \cdots$*

Remark 11.5.23 *It is possible to reduce the strong normalization of the $\lambda P2$-calculus to the strong normalization of system F by a translation technique that generalizes the one employed in section 11.4 for the system LF [GN91].*

Chapter 12

Stability

The theory of stable functions is originally due to Berry [Ber78]. It has been rediscovered by Girard [Gir86] as a semantic counterpart of his theory of dilators. Similar ideas were also developed independently and with purely mathematical motivations by Diers (see [Tay90a] for references).

Berry discovered stability in his study of sequential computation (cf. theorem 2.4) and of the full abstraction problem for PCF (cf. section 6.4). His intuitions are drawn from an operational perspective, where one is concerned, not only with the input-output behaviour of procedures, but also with questions such as: "which amount of the input is actually explored by the procedure before it produces an output". In Girard's work, stable functions arose in a construction of a model of system F (see chapter 11); soon after, his work on stability paved the way to linear logic, which is the subject of chapter 13.

In section 12.1 we introduce the conditionally multiplicative functions, which are the continuous functions preserving binary compatible glb's. In section 12.2 we introduce the stable functions and the stable ordering, focusing on minimal points and traces. Stability and conditional multiplicativity are different in general, but are equivalent under a well-foundedness assumption. They both lead to cartesian closed categories. In section 12.5 we build another cartesian closed category of stable functions, based on a characterisation of stable functions by the preservation of connected glb's. This category involves certain L-domains satisfying a strong distributivity axiom, which are investigated in section 12.6.

In the rest of the chapter, we impose algebraicity, as in chapter 5. In Section 12.3 we introduce event domains and their representations by event structures, and we show that they form a cartesian closed category. Berry's dI-domains are examples of event domains, and Girard's coherence spaces (which give rise to a model of linear logic) are examples of dI-domains. In section 12.4 we discuss the stable version of bifiniteness. Within this framework a remarkably simple theory of retractions can be developed. Figure 12.4 summarises the cartesian closed categories described in this chapter.

The present chapter is based on [Ber78] (sections 12.1, 12.2, 12.3), [Win80]

(section 12.3), [Tay90a] (sections 12.5 and 12.6), and [Ama91a] (section 12.4).

12.1 Conditionally Multiplicative Functions

In this section we focus on functions preserving the compatible binary glb's. We therefore work with cpo's which have such glb's. Moreover, this partial glb operation is required to be continuous. This condition ensures that function spaces ordered by the stable ordering are cpo's.

Definition 12.1.1 (meet cpo) *A cpo $(D, \leq)$ is called a meet cpo if*

1. $\forall x, y \ (x \uparrow y \Rightarrow x \wedge y \ exists)$,
2. $\forall x \ \forall \Delta \subseteq_{dir} D \ (x \uparrow (\bigvee \Delta) \Rightarrow x \wedge (\bigvee \Delta) = \bigvee \{x \wedge \delta \mid \delta \in \Delta\})$.

The condition (2) of definition 12.1.1, which expresses the continuity property of binary glb's, can be relaxed. Morevover, it comes for free in an algebraic cpo.

Lemma 12.1.2 *1. In a meet cpo, as soon as $x \wedge (\bigvee \Delta)$ exists, then the distributivity equality $x \wedge (\bigvee \Delta) = \bigvee \{x \wedge \delta \mid \delta \in X\}$ holds.*

2. An algebraic cpo is a meet cpo iff condition (1) of definition 12.1.1 holds.

PROOF. (1) We apply condition (2) with $x \wedge (\bigvee \Delta)$ in place of x:

$$(x \wedge (\bigvee \Delta)) \wedge (\bigvee \Delta) = \bigvee \{(x \wedge (\bigvee \Delta)) \wedge \delta \mid \delta \in X\} = \bigvee \{x \wedge \delta \mid \delta \in X\}.$$

(2) To check $x \wedge (\bigvee \Delta) \leq \bigvee \{x \wedge \delta \mid \delta \in X\}$, it is enough to check that every compact e such that $e \leq x \wedge (\bigvee \Delta)$ is also such that $e \leq \bigvee \{x \wedge \delta \mid \delta \in X\}$, which is clear since, by the definition of compact elements, $e \leq \bigvee \Delta$ implies $e \leq \delta$ for some $\delta \in \Delta$. □

In particular, in bounded complete cpo's the glb function is defined everywhere and is continuous. Some (counter-)examples are given in figure 12.1.

Definition 12.1.3 (conditionally multiplicative) *Let D and D' be meet cpo's. A function $f : D \to D'$ is called conditionally multiplicative, or cm for short if*

$$\forall x, y \in D \ \ x \uparrow y \Rightarrow f(x \wedge y) = f(x) \wedge f(y).$$

We write $D \to_{cm} D'$ for the set of cm functions from D to D'.

The fonction *por* considered in section 6.4 is an example of a continuous functions which is not cm:

$$por(\bot, tt) \wedge por(tt, \bot) = tt \neq \bot = por(\bot, \bot).$$