

# Formal Methods for Software Development

---

Till Mossakowski, Lutz Schröder

17.11.2004

# Denotational Semantics

- developed by Christopher Strachey and Dana Scott
- approach to formalize the semantics of computer programs
- while operational semantics uses a reduction relation, denotational semantics is **compositional**, i.e. the meaning of an expression (or indeed a whole program) is composed of the meaning of its constituent parts
- types are interpreted as domains (e.g. complete partial orders)
- programs are interpreted as continuous functions

## Partial Orders

A partial order  $(D, \leq)$  is a set  $D$  equipped with a binary relation  $\leq$  (i.e. a subset of  $D \times D$ ) that satisfies the following laws:

- **reflexivity**:  $x \leq x$
- **transitivity**:  $x \leq y \wedge y \leq z \Rightarrow x \leq z$
- **antisymmetry**:  $x \leq y \wedge y \leq x \Rightarrow x = y$

In domain theory,  $\leq$  generally is written as  $\sqsubseteq$ .

---

## Examples of Partial Orders

- natural numbers with the standard ordering
- words over an alphabet with lexicographic ordering
- divisibility relation on integers
- ancestor relation in trees

## $\omega$ -Chains and Suprema

An  $\omega$ -chain in a partial order  $(D, \sqsubseteq)$  is a family  $(x_i)_{i \in \mathbb{N}}$  with  $x_i \sqsubseteq x_{i+1}$  for all  $i \in \mathbb{N}$ .

An upper bound for an  $\omega$ -chain  $(x_i)_{i \in \mathbb{N}}$  is an element  $y \in D$  with  $x_i \sqsubseteq y$  for all  $i \in \mathbb{N}$ .

A supremum of an  $\omega$ -chain  $(x_i)_{i \in \mathbb{N}}$  is a least upper bound, i.e. an upper bound less or equal than every upper bound.

# Pointed $\omega$ -Complete Partial Orders ( $\omega$ -PCPOs)

A partial order  $(D, \sqsubseteq)$  is a **pointed  $\omega$ -complete partial order ( $\omega$ -PCPO)**, if

- if contains a least element  $\perp$ , and
- each chain  $(x_i)_{i \in \mathbb{N}}$  in  $D$  has a supremum  $\bigsqcup(x_i)$ , also written  $\bigsqcup_{i \in \mathbb{N}} x_i$ .

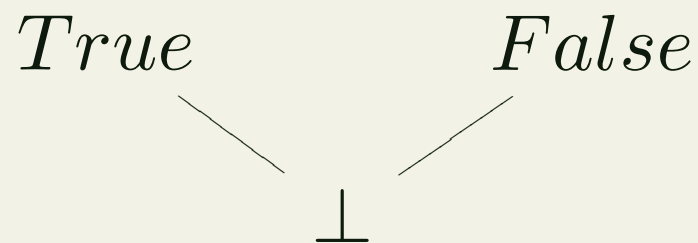
In the sequel, we just write CPO for  $\omega$ -PCPO.

## Example: Flat CPOs

Every set  $X$  can be turned into a CPO  $(D, \sqsubseteq)$  by putting

- $D = X \uplus \{\perp\}$
- $x \sqsubseteq y$  iff  $x = \perp$

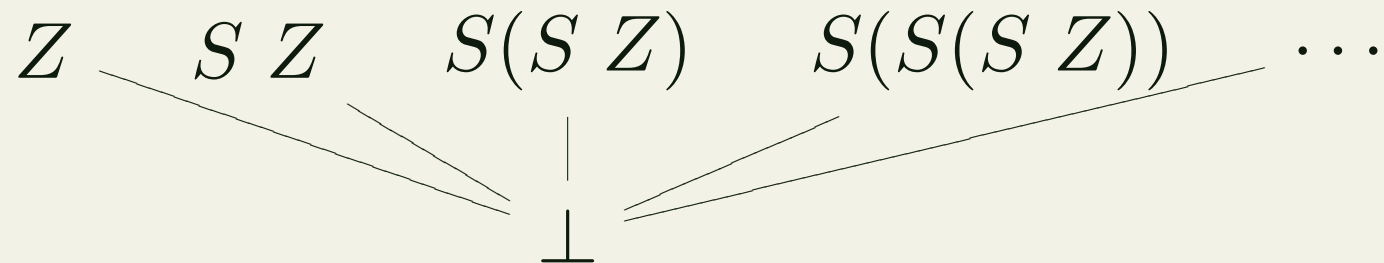
Example in Haskell semantics: Booleans



## Example: Strict Natural Numbers

```
data Nat = Z | S !Nat
```

$D = \{S^n Z \mid n \in \mathbb{N}\} \cup \{\perp\}$ , ordered as a flat CPO



The semantics of

```
infinity = S infinity
```

is  $\perp$ .



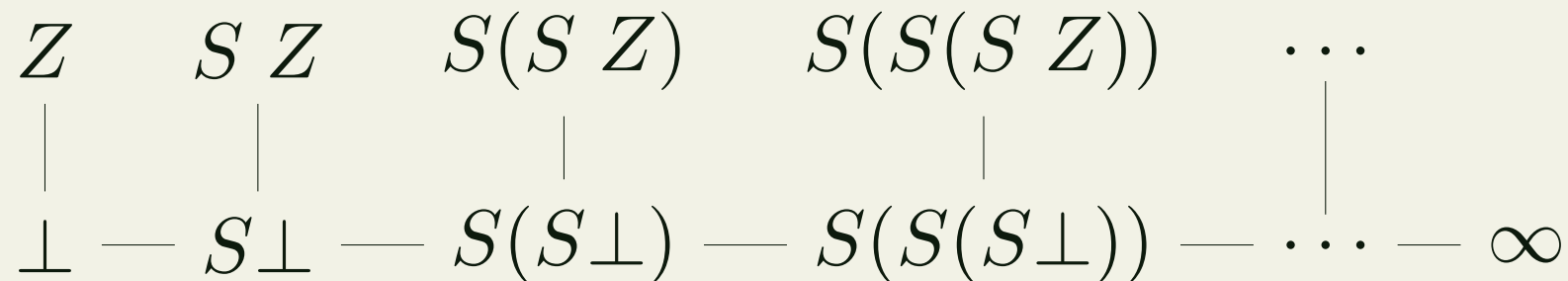
## Example: Lazy Natural Numbers

data Nat = Z | S Nat

$D = \{S^n \perp \mid n \in \mathbb{N}\} \cup \{S^n Z \mid n \in \mathbb{N}\} \cup \{\infty\}$

$\sqsubseteq$  is the least partially ordered relation with

- $S^m \perp \sqsubseteq S^n \perp$  for  $m \leq n$ ;
- $S^m \perp \sqsubseteq S^n Z$  for  $m \leq n$
- $S^n \perp \sqsubseteq \infty$  for any  $n$

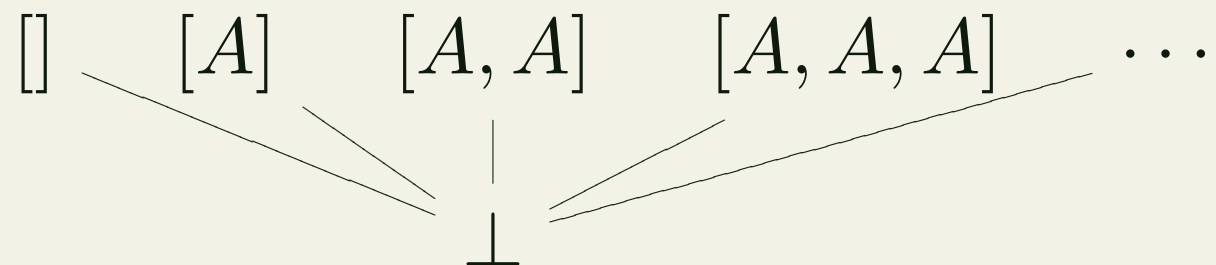


The semantics of `infinity = S infinity` is  $\infty$ .

## Example: Strict Lists

```
data List a = [] | (:) !a !(List a)
```

$D_{List\ a} = (D_a \setminus \{\perp\})^* \cup \{\perp\}$ , ordered as a flat CPO



Note that  $[x_1, \dots, x_n]$  is shorthand for  $x_1 : \dots : x_n : []$ .

## Example: Lazy Lists

```
data List a = [] | (:) a (List a)
```

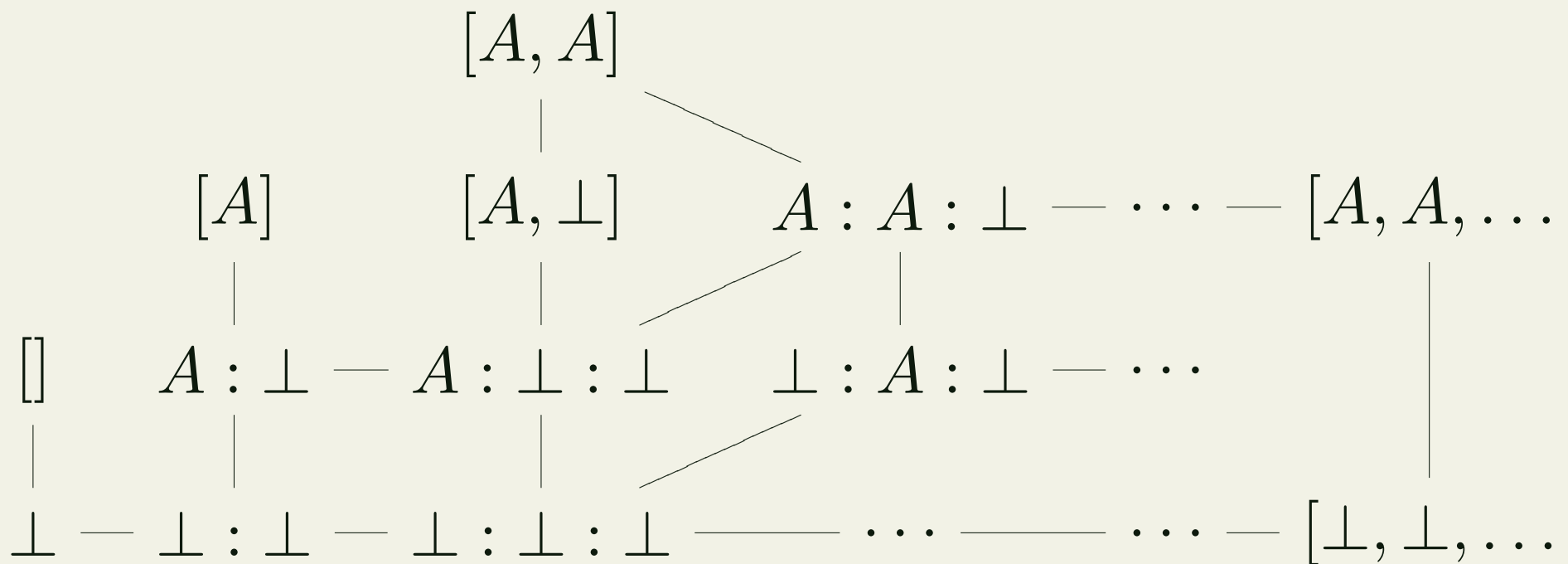
$$D_{List\ a} = D_a^* \times \{\perp, []\} \cup D_a^\omega$$

$D_a^\omega$  consists of sequences (indexed by  $\mathbb{N}$ ) of elements from  $D_a$ . Note that  $D_a$  contains an element  $\perp$ .

Informally,  $x \sqsubseteq y$  if  $y$  may be obtained from  $x$  by replacing some  $\perp$ 's with elements from  $D_a$  or  $D_{List\ a}$

# Lazy Lists Over One-Element Datatype

data unitA = A



# Semantics of Haskell

- Datatypes are interpreted as CPOs. Let  $\llbracket \tau \rrbracket$  be the interpretation of type  $\tau$
- Type constructors are interpreted as functions mapping CPOs to CPOs
- Functions are interpreted as continuous functions between two CPOs
- Polymorphic functions are interpreted as families of functions between CPOs (indexed by types)

## Function Spaces as CPOs

Let  $(D_1, \sqsubseteq_1)$  and  $(D_2, \sqsubseteq_2)$  be CPOs.

A function  $f : D_1 \rightarrow D_2$  is **continuous**, if it preserves  $\sqsubseteq$  and  $\sqcup$ . I.e.

$$x \sqsubseteq_1 y \text{ implies } f(x) \sqsubseteq_2 f(y)$$

$$f(\sqcup_{i \in \mathbb{N}} x_i) = \sqcup_{i \in \mathbb{N}} f(x_i)$$

**Proposition.** If  $(D_1, \sqsubseteq_1)$  and  $(D_2, \sqsubseteq_2)$  are CPOs, then the **space of continuous functions**  $[D_1 \rightarrow D_2]$  is a CPO with the pointwise ordering.

## Kleene's Theorem

**Theorem.** A continuous function  $F : D \rightarrow D$  on a CPO  $(D, \sqsubseteq)$  has a least fixed point, given by

$$\bigsqcup_{n \in \mathbb{N}} F^n(\perp)$$

where  $F^0(\perp) = \perp$

$$F^{n+1}(\perp) = F(F^n(\perp))$$

That is,  $F(\bigsqcup_{n \in \mathbb{N}} F^n(\perp)) = \bigsqcup_{n \in \mathbb{N}} F^n(\perp)$ , and for any  $x$  with  $F(x) = x$ , we have  $\bigsqcup_{n \in \mathbb{N}} F^n(\perp) \leq x$ .

# Semantics of Recursive Functions

A recursive definition

$$f :: t \rightarrow u$$
$$f\ x = \dots f \dots$$

leads to a functional  $F : [[t] \rightarrow [u]] \rightarrow [[t] \rightarrow [u]]$ , given by

$$F(f)(x) = \dots f \dots$$

The semantics of the recursive definition is given by the least fixed point of the functional  $F$ .



## Example: Factorial Function

A recursive definition

```
fac :: Int -> Int
```

```
fac x = if x=0 then 1 else x*fac(x-1)
```

The corresponding functional:

```
F :: (Int -> Int) -> (Int -> Int)
```

```
F f x = if x=0 then 1 else x*f(x-1)
```

$F \perp = \lambda x . \text{if } x = 0 \text{ then } 1 \text{ else } \perp$

$F(F \perp) = \lambda x . \text{if } x = 0 \text{ then } 1 \text{ else}$   
 $x * (\text{if } x - 1 = 0 \text{ then } 1 \text{ else } \perp)$

$\bigsqcup_{n \in \mathbb{N}} F^n(\perp)$  is the factorial function

## Admissible Predicates

- Given a CPO  $(D, \sqsubseteq)$ , a predicate  $P \subseteq D$  is called **admissible**, if it is closed under suprema of  $\omega$ -chains. This means: for any  $\omega$ -chain  $(x_i)_{i \in \mathbb{N}}$ , if  $x_i \in P$  for all  $i \in \mathbb{N}$ , then also  $\bigsqcup_{i \in \mathbb{N}} x_i \in P$
- Predicates in P-logic are interpreted as admissible predicates.

## Obtaining Admissible Predicates

- Admissible predicates are closed under intersections, but not under unions.
- If  $P$  is an arbitrary predicate, the intersection of all admissible predicates containing  $P$  is denoted by  $\langle P \rangle$  (the **admissible predicate generated by  $P$** ):

$$\langle P \rangle = \bigcap \{ Q \mid P \subseteq Q, Q \text{ admissible} \}$$

## Constructive Description of $\langle P \rangle$

$$P_0 = P$$

$$P_{n+1} = \{\bigsqcup_{i \in \mathbb{N}} x_i \mid x_i \in P_n \text{ for } i \in \mathbb{N}\}$$

$$P_\alpha = \bigcup_{i < \alpha} P_i \quad (\alpha \text{ a limit ordinal})$$

$$\langle P \rangle = \bigcup_i P_i$$

That is,  $\langle P \rangle$  adds to  $P$  all suprema of chains in  $P$ .

# Semantics of Fixed-Point Formulas

$$[Lfp \xi \bullet H]^{\tau} = \langle \bigcup_i H_i \rangle$$

$$[Gfp \xi \bullet H]^{\tau} = \bigcap_i H'_i$$

where

$$H_0 = \{\perp\}$$

$$H'_0 = \lceil \tau \rceil$$

$$H_{n+1} = H[H_n/\xi]$$

$$H'_{n+1} = H[H'_n/\xi]$$

$$H_{\alpha} = \bigcup_{i < \alpha} H_i$$

$$H'_{\alpha} = \bigcap_{i < \alpha} H'_i$$

## Example: Finite Lists

$$Lfp \xi \bullet [] \vee (Univ : \$\xi)$$

$H_i$  = set of all lists of length at most  $i$ , closing with  $[]$

$\bigcup_i H_i = \langle \bigcup_i H_i \rangle$  = set of all finite lists

## Example: Head-strict Lists

$$Lfp \xi \bullet [] \vee (\$Univ : \xi)$$

$H_i$  = set of all lists of length at most  $i$ , with defined elements (but possibly closing with  $\perp$ )

$\bigcup_i H_i$  = set of all head-strict finite lists (i.e. with defined elements, but possibly closing with  $\perp$ )

$\langle \bigcup_i H_i \rangle$  = set of all head-strict finite and infinite lists

## Example: Infinite Lists

$$Gfp \xi \bullet (Univ : \$\xi)$$

$H'_i$  = set of all lists of length at least  $i$

$\bigcap_i H'_i$  = set of all infinite lists