Signature

Definition

A (first order) signature Σ is a tuple $\langle \mathcal{F}, \mathrm{arity} \rangle$, where

- $\mathcal{F}$ is a set of function symbols
- $\mathrm{arity} : \mathcal{F} \rightarrow \mathbb{N}$ assign an arity to every function symbol.

Terms

Definition

Given

- A signature $\Sigma \equiv \langle \mathcal{F}, \mathrm{arity} \rangle$,
- A set of variables $\mathcal{V}$, such that $\mathcal{V} \cap \mathcal{F} = \emptyset$.

the set of terms over σ and $\mathcal{V}$ the smallest set $\mathcal{T}(\Sigma, \mathcal{V})$, such that

- if $x \in \mathcal{V}$ then $x \in \mathcal{T}(\Sigma, \mathcal{V})$;
- if $f \in \mathcal{F}$, $n = \mathrm{arity(f)}$ and $t_1, \cdots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$ then $f(t_1, \cdots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$.

Conventions

- From now we assume the existence of a set $\mathcal{V}$ of variables.
- If $\text{arity}(c) = 0$ then we abbreviate $c()$ by c.

OCaml as terms

```
type 'a mylist = Nil | Cons of 'a * 'a mylist;;
let lst=Cons(1,Cons(3,Cons(5,Nil)));;
let rec length = function
  | Nil -> 0
  | Cons(_,xs) -> 1+(length xs)
;;
```

arity	type expressions	normal expressions
0	int string ...	Nil lst length () ...
1	mylist	Cons
2	*	(.,.) + appl [(f a) stands for appl(f,a)]
3		(.,.,.)

Note that let (rec), match and fun are not first order constructs.

Variables

The set of variables occurring in a term t is

$$\mathrm{Var}(t) = \begin{cases} \{x\} & \text{, if } t \equiv x \in \mathcal{V} \\ \mathrm{Var}(t_1) \cup \cdots \cup \mathrm{Var}(t_n)), & \text{if } t \equiv f(t_1, \cdots, t_n) \end{cases}$$

Substitution

Definition

Given a signature Σ, and a function $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$,
we define $\sigma : \mathcal{T}(\Sigma, \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$, as

$$\sigma(t) = \begin{cases} \sigma(x) & \text{, if } t \equiv x \in \mathcal{V} \\ f(\sigma(t_1), \cdots, \sigma(t_n)) & \text{, if } t \equiv f(t_1, \cdots, t_n) \end{cases}$$

$$[x_1 := t_1, \cdots, x_n := t_n] \text{ denotes } x \mapsto \begin{cases} t_i & \text{, if } x \equiv x_i \\ x & \text{, otherwise} \end{cases}$$

The set of lambda terms Λ is the smallest set such that

- if $x \in \mathcal{V}$ then $x \in \Lambda$ [variable];
- if $M, N \in \Lambda$ then $(M\,N) \in \Lambda$ [application];
- if $x \in \mathcal{V}, M \in \Lambda$ then $\lambda x.M \in \Lambda$ [abstraction].

- $s\,t\,u$ means $((s\,t)\,u)$
- $\lambda x.s\,t$ means $\lambda x.(s\,t)$
- $\lambda x\,y\,z.M$ means $\lambda x.\lambda y.\lambda z.M$

Variables

The set of variables occurring free in a lambda term t is

$$\mathrm{FV}(t) = \begin{cases} \{x\} & \text{, if } t \equiv x \in \mathcal{V} \\ \mathrm{FV}(M) \setminus \{x\} & \text{, if } t \equiv \lambda x.M \\ \mathrm{FV}(M) \cup \mathrm{FV}(N) & \text{, if } t \equiv M\,N \end{cases}$$

The set of variables bound in a lambda term t is

$$\mathrm{BV}(t) = \begin{cases} \emptyset & \text{, if } t \equiv x \in \mathcal{V} \\ \mathrm{BV}(M) \cup \{x\} & \text{, if } t \equiv \lambda x.M \\ \mathrm{BV}(M) \cup \mathrm{BV}(N) & \text{, if } t \equiv M\,N \end{cases}$$

Finite Substitutions

Let $[x_i := t_i]$ denote $[x_1 := t_1, \cdots, x_n := t_n]$ then

$$t[x_i := t_i] = \begin{cases} t_i & \text{, if } t \equiv x_i \\ x & \text{, if } x \in \mathcal{V} \text{ and } x \notin \{x_1, \cdots, x_n\} \\ M[x_i := t_i]\, N[x_i := t_i] & \text{, if } t \equiv M\, N \\ \lambda x.M[x_i := t_i] & \text{, if } t \equiv \lambda x.M \\ & \quad \text{and } x \text{ does not occur in } [x_i := t_i] \\ \lambda z.M[x := z][x_i := t_i] & \text{, if } t \equiv \lambda x.M \\ & \quad \text{and } z \text{ does not occur in } [x_i := t_i] \text{ or } M \end{cases}$$

α-equivalence

- If z does not occur in M then $\lambda x.M$ and $\lambda z.M[x := z]$ are α-equivalent.
- We think of α-equivalent terms as the same term. E.g.
  - The terms $\lambda x.x\,y$ and $\lambda z.z\,y$ are the same term.
  - The terms $\lambda x.x\,y$ and $\lambda y.y\,y$ are different terms.

β-reduction

$$\frac{}{(\lambda x.M)\,N \xrightarrow[\beta]{} M[x := N]}$$

$$\frac{M \xrightarrow[\beta]{} M'}{\lambda x.M \xrightarrow[\beta]{} \lambda x.M'}$$

$$\frac{M \xrightarrow[\beta]{} M'}{M\,N \xrightarrow[\beta]{} M'\,N} \qquad \frac{N \xrightarrow[\beta]{} N'}{M\,N \xrightarrow[\beta]{} M\,N'}$$

The module system of OCaml

- Modules can contain both type declarations and function declarations.
- Functors are module definitions that take other modules are parameters.
  Functors are 'evaluated' at compile time.
- The type system has module types, which can be used for hiding implementation details.