Semantics, WS 2003:
Solutions for assigment 9

Prof. Dr. Gert Smolka, Dipl.-Inform. Guido Tack

Exercise 9.1: Recursive types and fix

```
datatype 't at = F of 't at -> 't

val fix = fn f =>    (fn (F x) => fn y => f (x (F x)) y)
                  (F (fn (F x) => fn y => f (x (F x)) y))
```

Exercise 9.2: Streams

```
datatype 'a stream = S of unit -> 'a * 'a stream

fun hd (S s) = #1 (s ())
fun tl (S s) = #2 (s ())
fun nth s 0 = hd s
  | nth s n = nth (tl s) (n-1)

val nat =
    let
      fun upfrom n = S (fn () => (n, upfrom (n+1)))
    in
      upfrom 0
    end

val fac =
    let
      fun next n acc = S (fn () => (acc, next (n+1) (acc*(n+1))))
    in
      next 0 1
    end

val fib =
    let
      fun next n1 n2 = S (fn () => (n1+n2, next n2 (n1+n2)))
    in
      S (fn () => (0,
                   S (fn () => (1,
                                next 0 1))))
    end
```

Exercise 9.3: Untyped λ-calculus in SML

```
exception Error
datatype lam = L of lam -> lam
             | I of int

val zero = L (fn s => L (fn z => z))
val succ = L (fn (L n) =>
               L (fn (L s) =>
                   L (fn z =>
                       (case n (L s) of
                         (L ns) => s (ns z)))))

fun app (L t1) t2 = t1 t2

fun num 0 = zero
  | num n = app succ (num (n-1))

fun int n =
    let
        fun next (I n) = I (n+1)
          | next _ = raise Error
    in
         case
            app (app n (L next)) (I 0)
         of (I i) => i
    end

val plus = L (fn (L m) =>
               L (fn (L n) =>
                   L (fn s =>
                       L (fn z =>
                           case n s of (L ns) =>
                           case m s of (L ms) =>
                           ms (ns z)))))

val times = L (fn (L m) =>
                L (fn n =>
                    case m (app plus n) of (L mp) => mp zero))
```

Error handling is omitted to make the code more readable. Usually, in every pattern matching an error case has to be introduced.

Exercise 9.4: Infinite trees in SML

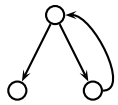
```
datatype inftree = T of int * (unit -> inftree * inftree)

val inftree =
    let
        fun next n = T (n, fn () => (next (n+1), next (n+1)))
    in
        next 1
    end
```

Exercise 9.5: μ-terms and tree domains

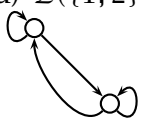
(a) $\mathcal{L}(\{1\})$

(b) $L_0 \;=\; \{[], [1], [2]\}$
$L_i \;=\; \{[2,1]@l \mid l \in L_{i-1}\}$
$T \;=\; \bigcup_{i \in \mathbb{N}} L_i$

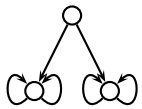


(c) $\mathcal{L}(\{1,2\})$

(d) $\mathcal{L}(\{1,2\})$



(e) $\mathcal{L}(\{1,2\})$



Exercise 9.6: Tree domains

(a) $\{[], [1], [2], [2,1], [2,1,1], [2,2]\}$

(b) $\mathcal{L}(\{1\})$

(c) $\{[n] \mid n \in \mathbb{N}\} \;\cup\; \{[]\}$

(d) $\mathcal{L}(\mathbb{N})$

Exercise 9.7: Tree domains and subtrees We use pattern matching syntax to get the head and tail of a list:

$sub(T, n) = \{l \mid (n :: l) \in T\}$

Exercise 9.8: Iso-recursive λ-calculus

(a)
$$\frac{T = \mu X.T' \qquad \Gamma \vdash v : T'[X := \mu X.T']}{\Gamma \vdash fold[T]v : T}$$

$$\frac{}{\Gamma \vdash unit : Unit}$$

$$\frac{\Gamma \vdash v : T}{\Gamma \vdash \langle 1 = v \rangle : T + T'}$$

$$\frac{\Gamma \vdash v : T'}{\Gamma \vdash \langle 2 = v \rangle : T + T'}$$

(b) $Unit + \mu X.X$ and $Unit + \mu X.(X + X)$

Exercise 9.9: Fix in iso-recursive λ-calculus Let $U = T_0 \to T_1$. Then

$$\begin{aligned} fix \quad = \quad & \lambda f : (U \to U) \to U. \\ & (\lambda x : (\mu X.X \to U).f(unfold[\mu X.X \to U]x\ x)) \\ & (fold[\mu X.X \to U](\lambda x : (\mu X.X \to U).f(unfold[\mu X.X \to U]x\ x))) \end{aligned}$$

Exercise 9.10: Fix in equi-recursive λ-calculus Let $U = T_0 \to T_1$. Then

$$\begin{aligned} fix \quad = \quad & \lambda f : (U \to U) \to U. \\ & (\lambda x : (\mu X.X \to U).f(x\ x)) \\ & (\lambda x : (\mu X.X \to U).f(x\ x)) \end{aligned}$$