# Semantics, WS 2005 – Assignment 7

Prof. Dr. Gert Smolka, Dipl.-Inform. Andreas Rossberg
http://www.ps.uni-sb.de/courses/sem-ws05/

Recommended reading: Types and Programming Languages, Chapter 20

With this assignment you will explore recursive types. There are several exercises that explore the expressivity of types whose recursion goes through procedural types using SML.

For now, we only consider iso-recursive types, equi-recursive types will be treated later.

**Exercise 7.1: Streams** A stream represents an infinite sequence $v_1, v_2, v_3, \ldots$ of values. In SML, we can implement streams with the recursive type

```
datatype α stream = S of unit → α × α stream
```

(a) Write a procedure *stream* : $\alpha \to (\alpha \to \alpha) \to \alpha$ *stream* that yields for $x$ and $f$ the stream $x,\ fx,\ f(fx),\ \ldots$.

(b) Declare the stream $0, 1, 2, \ldots$ of natural numbers.

(c) Write a procedure *head* : $\alpha$ *stream* $\to \alpha$ that yields the head of a stream.

(d) Write a procedure *tail* : $\alpha$ *stream* $\to \alpha$ *stream* that yields the tail of a stream.

(e) Write a procedure *nth* : *int* $\to \alpha$ *stream* $\to \alpha$ that yields the $n$th element of a stream (*nth 1 s* should yield the head of a stream).

(f) Write a procedure *cons* : $\alpha \to \alpha$ *stream* $\to$ $\alpha$ *stream* that puts a value at the beginning of a stream.

(g) Write a procedure *merge* : $\alpha$ *stream* $\to \alpha$ *stream* $\to \alpha$ *stream* that merges two streams $v_1, v_2, v_3, \ldots$ and $w_1, w_2, w_3, \ldots$ into a stream $v_1, w_1, v_2, w_2, v_3, w_3, \ldots$.

**Exercise 7.2: Fix** Declare in SML a recursion operator

$$\mathit{fix} : ((\alpha \to \beta) \to \alpha \to \beta) \to \alpha \to \beta$$

without using procedural recursion. Use the recursive type

```
datatype (α,β) fix = F of (α,β) fix → α → β
```

which makes it possible to apply a procedure $f : (\alpha, \beta)\, \mathit{fix} \to \alpha \to \beta$ to itself $(f\ (F\ f))$ by lowering its type with the constructor $F$. Base your procedure *fix* on the untyped recursion operator

$$S = \lambda g f x.g(f\,f)x$$
$$\mathit{fix} = \lambda g.(Sg)(Sg)$$

and give the variable $f$ the type $(\alpha, \beta)\, \mathit{fix}$.

**Exercise 7.3: SR** We consider a simply typed language SR with recursive types and deterministic, left-to-right, call-by-value reduction:

$$\begin{aligned}
&X \in \mathit{TVar}\\
&T \in \mathit{Ty} \;=\; 1 \mid T \to T \mid T \times T \mid T + T \mid X \mid \mu X.T\\
&x \in \mathit{Var}\\
&i \in \{1, 2\}\\
&t \in \mathit{Ter} \;=\; () \mid x \mid \lambda x{:}T.t \mid t\ t \mid (t, t) \mid t.i \mid (i, t)\text{ as }T \mid \text{case } t\ t\ t\\
&\qquad\qquad \mid \text{fold } t \text{ as } T \mid \text{unfold } t\\
&v \in \mathit{Val} \subseteq \mathit{Ter} \;=\; () \mid \lambda x{:}T.t \mid (v, v) \mid (i, v)\text{ as }T \mid \text{fold } v \text{ as } T\\
&\Gamma \in \mathit{TE} \;=\; \mathit{Var} \rightharpoonup \mathit{Ty}
\end{aligned}$$

The typing rules for fold and unfold are as follows:

$$\frac{\Gamma \vdash t : T[X := \mu X.T]}{\Gamma \vdash \text{fold } t \text{ as } \mu X.T : \mu X.T} \qquad \frac{\Gamma \vdash t : \mu X.T}{\Gamma \vdash \text{unfold } t : T[X := \mu X.T]}$$

(a) State the reduction contexts for SR.

(b) State the proper reduction rule(s) needed for fold and unfold.

(c) State the evaluation rules in big-step semantics for fold and unfold.

(d) Given types $T_1$ and $T_2$, give a recursion operator $\mathit{fix} : ((T_1 \to T_2) \to (T_1 \to T_2)) \to (T_1 \to T_2)$.

(e) State a type *Nat* in SR that represents the natural numbers.

(f) State a value $\mathit{zero} : \mathit{Nat}$ that represents the number 0.

(g) State a procedure $\mathit{succ} : \mathit{Nat} \to \mathit{Nat}$ that increments a number by 1.

(h) Given a type $T$, state a procedure $\mathit{ncase}_T : \mathit{Nat} \to (1 \to T) \to (\mathit{Nat} \to T) \to T$.

(i) State a procedure $\mathit{add} : \mathit{Nat} \to \mathit{Nat} \to \mathit{Nat}$ that adds two numbers. Use *fix*.

(j) State a type *List* in SR that represents lists of natural numbers.

(k) State a value $nil : List$ that represents the empty list.

(l) State a procedure $cons : Nat \to List \to List$ that puts a number in front of a list.

(m) Given a type $T$, state a procedure $lcase_T : List \to (1 \to T) \to (Nat \to List \to T) \to T$.

(n) State a procedure $length : List \to Nat$ that yields the length of a list. Use *fix*.

**Exercise 7.4: UN** We consider an untyped language UN whose values are procedures or integers:

$$\begin{aligned} & x \in Var \\ & n \in \mathbb{N} \\ & t \in Ter \;=\; x \mid t\,t \mid \lambda x.t \mid n \mid n+1 \\ & v \in Val \subseteq Ter \;=\; \lambda x.t \mid n \end{aligned}$$

UN's evaluation is deterministic, left-to-right and call by value. A term is called *pure* if it doesn't contain arithmetic primitives. We say that a value $v$ *represents a natural number* $n$ if $v$ is closed and pure and $v(\lambda x.x+1)0$ evaluates to $n$.

We implement UN in SML using a technique that is quite different from the one we used in Assignment 1. We start with the closed values of UN:

```
datatype cval = P of cval -> cval | N of int
```

The idendity procedure and the Church numeral $c_0$ can be represented as follows:

```
val id = P (fn x => x)
val zero = P (fn f => id)
```

Next we implement an SML procedure *apply* that takes two closed values $v_1$ and $v_2$ of UN and yields the closed value obtained by evaluating the application $v_1 v_2$ in UN. If $v_1 v_2$ diverges in UN, *apply* $v_1$ $v_2$ will diverge in SML.

```
exception Error
fun apply (P f) x = f x | apply _ _ = raise Error
val apply : cval → cval → cval
```

UN's operation $t + 1$ can be implemented as follows:

```
fun plus1 (N n) = N(n+1) | plus1 _ = raise Error
val plus1 : cval → cval
```

Now it's your job to declare the following:

(a) A UN procedure *succ* : *cval* that yields a UN procedure representing $n + 1$ when applied to a value representing $n$.

(b) An SML procedure *put* : *int* → *cval* that yields a UN procedure representing $n$ when applied to a natural number $n$.

(c) A UN procedure *derep* : *cval* that yields the UN value $n$ when applied to a value representing $n$.

(d) An SML procedure *get* : *cval* → *int* that yields $n$ when applied to a value representing $n$.

(e) A UN procedure *add* : *cval* that yields a UN procedure representing $m + n$ when applied to two procedures representing $m$ and $n$.

(f) A UN procedure *mul* : *cval* that yields a UN procedure representing $m \cdot n$ when applied to two procedures representing $m$ and $n$.

**Exercise 7.5: Translation** Find a translation from SR to U (UN without numbers) such that for every closed and well-typed term $t$ of SR the following holds: $t$ terminates in SR if and only if its translation terminates in U.

$$
\begin{array}{rcl}
() & \rightsquigarrow & \lambda x.x \\
\lambda x{:}T.t & \rightsquigarrow & \lambda x.t \\
t_1\, t_2 & \rightsquigarrow & t_1\, t_2 \\
(t_1, t_2) & \rightsquigarrow & \\
t.i & \rightsquigarrow & \\
(i, t) \text{ as } T & \rightsquigarrow & \\
\text{case } t_1\ t_2\ t_3 & \rightsquigarrow & \\
\text{fold } t \text{ as } T & \rightsquigarrow & \\
\text{unfold } t & \rightsquigarrow &
\end{array}
$$