CS6110 Lect 21 Friday March 9, 2012

A state machine version of cbn eval.

The state of this EvalMachine is

State = Stop(Term) | Eval(Term, Cont) | Apply(Term, Cont) | E

The transition function mapping State to State is

```
trans S  case(S) of
           Eval(t,k) →
              case(t) of
                 isvar(x) → E        (an error)
                 isabs(v) → Apply(t,k)
                 isap(fa) → Eval(f, C(a,k))

           Apply(t,k) →
              case(k) of
                 Halt → Stop(t)
                 C(a,k') →
                       case(t) of
                          λ(x b) → Eval(b[v/x], k')
                       else → E

           else → S
end trans
```

The EvalMachine (EvalM) is loop(Eval(t, Halt))

where loop(S) = case(S)

```
                E → Error
                Stop t → t
                else → loop(trans S)
```

Example of the EvalMachine in action

Let $t = ap(\lambda(x.xx); \lambda(y.y))$

S_0 = Eval(t, Halt)

loop (S_1 = Eval($\lambda(x.xx)$, C($\lambda(y.y)$, Halt))

↳ S_2 = Apply($\lambda(x.xx)$, C($\lambda(y.y)$, Halt))

↳ S_3 = Eval($\lambda(y.y)\ \lambda(y.y)$, Halt)

↳ S_4 = Eval($\lambda(y.y)$, C($\lambda(y.y)$, Halt))

↳ S_5 = Apply($\lambda(y.y)$, C($\lambda(y.y)$, Halt))

↳ S_6 = Eval($\lambda(y.y)$, Halt)

↳ S_7 = Apply($\lambda(y.y)$, Halt)

↳ S_8 = stop($\lambda(y.y)$)

CS6110 Lecture 21 Friday March 9, 2012
CS6116 Lecture 7

Now we look at subclasses of R that go far beyond the primitive recursive functions. They can be defined simply in some cases as in this first example related to Gödel's "second programming language," the one given in his system T, circa 1958 (ideas from 1941, elaborated further in 1972)! Fundamental work on this system was done by Tait in 1967 Intensional interp of functionals of finite type JSL.32, 198-212. We will study Tait His method led to what computer scientists call logical relations.

Here is a remarkably simple version, by Gilles Dowek, draft unknown date. He gives the reduction rules for system T combinators.

$$Kxy \to x$$
$$Sxyz \to xz(yz)$$
$$Rxf0 \to x$$
$$Rxf(Sy) \to fy(Rxfy)$$

Note K is $\lambda(x.\lambda(y.x))$ S is $\lambda(x.\lambda(y.\lambda(z.xz(yz))))$
Think of R as the computational meaning of induction.
Another form of it close to what is used in Intuitionistic Type Theory (ITT) is

$$R(0,x,f) = x$$
$$R(S(u),x,f) = f(y, R(u,x,f))$$

The ITT form is

$$ind(0;b;u,i.h) \triangleright b$$
$$ind(S(y);b;u,i.h) \triangleright h(y/u, ind(y;b;u,i.h)/i)$$

CS 6110 Lect 21
CS 6116 Lect 7

Stenlund 1972 presents T using the λ-calculus.

1. $\lambda(x.b(x))\ a \triangleright b[a/x]$
2. $R\ b\ h\ 0 \triangleright b$
3. $R\ b\ h\ S(t) \triangleright h\ t\ (R\ b\ h\ t)$

This uses an applied λ-calculus with 0, S, and R. Unlike the primitive recursive functions, this class of functions includes all λ terms, hence is very expressive, including all partial recursive functions.

Gödel needed a mechanism to specify the total functions. His key idea was to use types from Principia Mathematica in a form simplified by Church

First, notice that we can obtain all of the primitive recursive algorithms as we demonstrated in the first problem set. That is, we can include them naturally using only 0, S(t), and case as new primitives along with fix(t) as a primitive operator. But note, all of these "primitives" can be defined in the pure λ-calculus or combinatory calculus.

Types

The simple types that Gödel used are these.

N is the type of natural numbers.
If α and β are types, then $\alpha \to \beta$ is the type of total objects (functions) from α to β.