CIS 500

Software Foundations

Fall 2003

29 September

Administrivia

- Reading for this Wednesday: Chapter 8
- First midterm is next Wednesday
  - covering OCaml programming, TAPL chapters 3–8, and all material in lectures and homeworks
  - Monday's class will be a review session: come with questions!

Equivalence of Lambda Terms

Representing Numbers

We have seen how certain terms in the lambda-calculus can be used to represent natural numbers.

c_0 = λs. λz. z

c_1 = λs. λz. s z

c_2 = λs. λz. s (s z)

c_3 = λs. λz. s (s (s z))

Other lambda-terms represent common operations on numbers:

scc = λn. λs. λz. s (n s z)

Representing Numbers

We have seen how certain terms in the lambda-calculus can be used to represent natural numbers.

$c_0 = \lambda s.\ \lambda z.\ z$

$c_1 = \lambda s.\ \lambda z.\ s\ z$

$c_2 = \lambda s.\ \lambda z.\ s\ (s\ z)$

$c_3 = \lambda s.\ \lambda z.\ s\ (s\ (s\ z))$

Other lambda-terms represent common operations on numbers:

$scc = \lambda n.\ \lambda s.\ \lambda z.\ s\ (n\ s\ z)$

In what sense can we say this representation is "correct"?

In particular, on what basis can we argue that scc on church numerals corresponds to ordinary successor on numbers?

The naive approach

One possibility:

For each n, the term $\mathtt{scc}\ c_n$ evaluates to c_{n+1}.

The naive approach... doesn't work

One possibility:

For each n, the term $\texttt{scc}\ c_n$ evaluates to c_{n+1}.

Unfortunately, this is false.

E.g.:

$$
\begin{array}{rcl}
\texttt{scc}\ c_2 & = & (\lambda\texttt{n.}\ \lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s (n s z)})\ (\lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s (s z)}) \\
 & \longrightarrow & \lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s ((}\lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s (s z)) s z)} \\
 & \neq & \lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s (s (s z))} \\
 & = & c_3
\end{array}
$$

A better approach

Recall the intuition behind the church numeral representation:

- a number n is represented as a term that "does something n times to something else"
- $\mathtt{scc}$ takes a term that "does something n times to something else" and returns a term that "does something $n+1$ times to something else"

I.e., what we really care about is that $\mathtt{scc}\ c_2$ behaves the same as c_3 when applied to two arguments.

$$
\begin{array}{lcl}
\texttt{scc}\ c_2\ \texttt{v w} & = & (\lambda\texttt{n. }\lambda\texttt{s. }\lambda\texttt{z. s (n s z)) (}\lambda\texttt{s. }\lambda\texttt{z. s (s z)) v w} \\
 & \longrightarrow & (\lambda\texttt{s. }\lambda\texttt{z. s ((}\lambda\texttt{s. }\lambda\texttt{z. s (s z)) s z)) v w} \\
 & \longrightarrow & (\lambda\texttt{z. v ((}\lambda\texttt{s. }\lambda\texttt{z. s (s z)) v z)) w} \\
 & \longrightarrow & \texttt{v ((}\lambda\texttt{s. }\lambda\texttt{z. s (s z)) v w)} \\
 & \longrightarrow & \texttt{v ((}\lambda\texttt{z. v (v z)) w)} \\
 & \longrightarrow & \texttt{v (v (v w))}
\end{array}
$$

$$
\begin{array}{lcl}
c_3\ \texttt{v w} & = & (\lambda\texttt{s. }\lambda\texttt{z. s (s (s z))) v w} \\
 & \longrightarrow & (\lambda\texttt{z. v (v (v z))) w} \\
 & \longrightarrow & \texttt{v (v (v w)))}
\end{array}
$$

A More General Question

We have argued that, although scc c_2 and c_3 do not evaluate to the same thing, they are nevertheless "behaviorally equivalent."

What, precisely, does behavioral equivalence mean?

Intuition

Roughly,

> terms `s` and `t` are behaviorally equivalent

should mean:

> there is no "test" that distinguishes `s` and `t` — i.e., no way to use them in the same context and obtain different results.

Some test cases

```
tru = λt. λf. t
tru' = λt. λf. (λx.x) t
fls = λt. λf. f
omega = (λx. x x) (λx. x x)
poisonpill = λx. omega
placebo = λx. tru
```

Y_f `= (λx. f (x x)) (λx. f (x x))`

Which of these are behaviorally equivalent?

Observational equivalence

As a first step toward defining behavioral equivalence, we can use the notion of normalizability to define a simple way of testing terms.

> Two terms `s` and `t` are said to be observationally equivalent if either both are normalizable (i.e., they reach a normal form after a finite number of evaluation steps) or both are divergent.

I.e., our primitive notion of "observing" a term's behavior is simply running it on our abstract machine.

Observational equivalence

As a first step toward defining behavioral equivalence, we can use the notion of normalizability to define a simple way of testing terms.

> Two terms `s` and `t` are said to be observationally equivalent if either both are normalizable (i.e., they reach a normal form after a finite number of evaluation steps) or both are divergent.

I.e., our primitive notion of "observing" a term's behavior is simply running it on our abstract machine.

Aside:

- Is observational equivalence a decidable property?

Observational equivalence

As a first step toward defining behavioral equivalence, we can use the notion of normalizability to define a simple way of testing terms.

> Two terms `s` and `t` are said to be observationally equivalent if either both are normalizable (i.e., they reach a normal form after a finite number of evaluation steps) or both are divergent.

I.e., our primitive notion of "observing" a term's behavior is simply running it on our abstract machine.

Aside:

- Is observational equivalence a decidable property?
- Does this mean the definition is ill-formed?

Examples

- `omega` and `tru` are *not* observationally equivalent

Examples

- `omega` and `tru` are *not* observationally equivalent
- `tru` and `fls` *are* observationally equivalent

Behavioral Equivalence

This primitive notion of observation now gives us a way of "testing" terms for behavioral equivalence

Terms `s` and `t` are said to be behaviorally equivalent if, for every finite sequence of values $v_1, v_2, \ldots, v_n$, the applications

$$s\ v_1\ v_2\ \ldots\ v_n$$

and

$$t\ v_1\ v_2\ \ldots\ v_n$$

are observationally equivalent.

Examples

These terms are behaviorally equivalent:

```
tru = λt. λf. t
tru' = λt. λf. (λx.x) t
```

So are these:

```
omega = (λx. x x) (λx. x x)
```

Y_f = (λx. f (x x)) (λx. f (x x))

These are *not* behaviorally equivalent (to each other, or to any of the terms above):

```
fls = λt. λf. f
poisonpill = λx. omega
placebo = λx. tru
```

Formalizing the Lambda-Calculus

(From TAPL chapter 6, on the board...)