Foundations of Software
Spring 2025

Week 4

Equivalence of Lambda Terms

Recall: Church Numerals

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)

Recall: Church Numerals

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:

$$\mathtt{scc} = \lambda n.\ \lambda s.\ \lambda z.\ s\ (n\ s\ z)$$

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 $\texttt{scc}\ \texttt{c}_n$ evaluates to $\texttt{c}_{n+1}$.

The naive approach... doesn't work

One possibility:

> For each n, the term $\texttt{scc c}_n$ evaluates to $\texttt{c}_{n+1}$.

Unfortunately, this is false.

E.g.:

$$
\begin{array}{lcl}
\texttt{scc c}_2 & = & (\lambda \texttt{n. } \lambda \texttt{s. } \lambda \texttt{z. s (n s z)) } (\lambda \texttt{s. } \lambda \texttt{z. s (s z))} \\
 & \longrightarrow & \lambda \texttt{s. } \lambda \texttt{z. s ((} \lambda \texttt{s. } \lambda \texttt{z. s (s z)) s z)} \\
 & \neq & \lambda \texttt{s. } \lambda \texttt{z. s (s (s z))} \\
 & = & \texttt{c}_3
\end{array}
$$

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"
- `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 $\texttt{scc}\ \texttt{c}_2$ behaves the same as $\texttt{c}_3$ when applied to two arguments.

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

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

A 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 put them in the same context and observe different results."

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 put them in the same context and observe different results."

To make this precise, we need to be clear what we mean by a *testing context* and how we are going to *observe* the results of a test.

Examples

```
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 notion of *test*.

> 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 diverge.

I.e., we "observe" a term's behavior simply by running it and seeing if it halts.

Observational equivalence

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

> 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 diverge.

I.e., we "observe" a term's behavior simply by running it and seeing if it halts.

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 notion of *test*.

> 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 diverge.

I.e., we "observe" a term's behavior simply by running it and seeing if it halts.

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 $\texttt{v}_1$, $\texttt{v}_2$, ..., $\texttt{v}_n$, the applications
>
> $$\texttt{s}\ \texttt{v}_1\ \texttt{v}_2\ \ldots\ \texttt{v}_n$$
>
> and
>
> $$\texttt{t}\ \texttt{v}_1\ \texttt{v}_2\ \ldots\ \texttt{v}_n$$
>
> 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
```

Proving behavioral equivalence

Given terms `s` and `t`, how do we *prove* that they are (or are not) behaviorally equivalent?

Proving behavioral inequivalence

To prove that `s` and `t` are *not* behaviorally equivalent, it suffices to find a sequence of values $v_1 \ldots v_n$ such that one of

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

and

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

diverges, while the other reaches a normal form.

Proving behavioral inequivalence

Example:

- the single argument `unit` demonstrates that `fls` is not behaviorally equivalent to `poisonpill`:

$$\begin{aligned} &\texttt{fls unit} \\ = &\ (\lambda\texttt{t. }\lambda\texttt{f. f) unit} \\ \longrightarrow^* &\ \lambda\texttt{f. f} \end{aligned}$$

$$\texttt{poisonpill unit}$$

diverges

Proving behavioral inequivalence

Example:

- the argument sequence `(λx. x) poisonpill (λx. x)` demonstrate that `tru` is not behaviorally equivalent to `fls`:

$$\texttt{tru}\ (\lambda\texttt{x}.\ \texttt{x})\ \texttt{poisonpill}\ (\lambda\texttt{x}.\ \texttt{x})$$
$$\longrightarrow^{*}\ (\lambda\texttt{x}.\ \texttt{x})(\lambda\texttt{x}.\ \texttt{x})$$
$$\longrightarrow^{*}\ \lambda\texttt{x}.\ \texttt{x}$$

$$\texttt{fls}\ (\lambda\texttt{x}.\ \texttt{x})\ \texttt{poisonpill}\ (\lambda\texttt{x}.\ \texttt{x})$$
$$\longrightarrow^{*}\ \texttt{poisonpill}\ (\lambda\texttt{x}.\ \texttt{x}),\ \text{which diverges}$$

Proving behavioral equivalence

To prove that `s` and `t` *are* behaviorally equivalent, we have to work harder: we must show that, for *every* sequence of values $v_1 \ldots v_n$, either both

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

and

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

diverge, or else both reach a normal form.

How can we do this?

Proving behavioral equivalence

In general, such proofs require some additional machinery that we will not have time to get into in this course (so-called *applicative bisimulation*). But, in some cases, we can find simple proofs.
Theorem: These terms are behaviorally equivalent:

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

Proof: Consider an arbitrary sequence of values $\mathtt{v}_1 \ldots \mathtt{v}_n$.

- For the case where the sequence has up to one element (i.e., $n \leq 1$), note that both $\mathtt{tru}$ / $\mathtt{tru}\ \mathtt{v}_1$ and $\mathtt{tru}'$ / $\mathtt{tru}'\ \mathtt{v}_1$ reach normal forms after zero / one reduction steps.
- For the case where the sequence has more than one element (i.e., $n > 1$), note that both $\mathtt{tru}\ \mathtt{v}_1\ \mathtt{v}_2\ \mathtt{v}_3\ \ldots\ \mathtt{v}_n$ and $\mathtt{tru}'\ \mathtt{v}_1\ \mathtt{v}_2\ \mathtt{v}_3\ \ldots\ \mathtt{v}_n$ reduce to $\mathtt{v}_1\ \mathtt{v}_3\ \ldots\ \mathtt{v}_n$. So either both normalize or both diverge.

Proving behavioral equivalence

Theorem: These terms are behaviorally equivalent:

```
omega = (λx. x x) (λx. x x)
Y_f = (λx. f (x x)) (λx. f (x x))
```

Proof: Both

$$\texttt{omega}\ \texttt{v}_1 \ldots \texttt{v}_n$$

and

$$\texttt{Y}_f\ \texttt{v}_1 \ldots \texttt{v}_n$$

diverge, for every sequence of arguments $\texttt{v}_1 \ldots \texttt{v}_n$.