SNU 4541.574
Programming Language Theory

Ack: BCP's slides

Equivalence of Lambda Terms

Representing Numbers

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

$$
\begin{aligned}
&\texttt{c}_0 = \lambda\texttt{s}.\ \lambda\texttt{z}.\ \texttt{z}\\
&\texttt{c}_1 = \lambda\texttt{s}.\ \lambda\texttt{z}.\ \texttt{s z}\\
&\texttt{c}_2 = \lambda\texttt{s}.\ \lambda\texttt{z}.\ \texttt{s (s z)}\\
&\texttt{c}_3 = \lambda\texttt{s}.\ \lambda\texttt{z}.\ \texttt{s (s (s z))}
\end{aligned}
$$

Other lambda-terms represent common operations on numbers:

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

Representing Numbers

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

$$
\begin{aligned}
c_0 &= \lambda s.\ \lambda z.\ z \\
c_1 &= \lambda s.\ \lambda z.\ s\ z \\
c_2 &= \lambda s.\ \lambda z.\ s\ (s\ z) \\
c_3 &= \lambda s.\ \lambda z.\ s\ (s\ (s\ z))
\end{aligned}
$$

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 $\texttt{scc}\ \texttt{c}_n$ 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”
- $\texttt{scc}$ 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.

$$
\begin{aligned}
\texttt{scc}\ \texttt{c}_2\ \texttt{v}\ \texttt{w} &= (\lambda\texttt{n.}\ \lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s}\ (\texttt{n}\ \texttt{s}\ \texttt{z}))\ (\lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s}\ (\texttt{s}\ \texttt{z}))\ \texttt{v}\ \texttt{w} \\
&\longrightarrow (\lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s}\ ((\lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s}\ (\texttt{s}\ \texttt{z}))\ \texttt{s}\ \texttt{z}))\ \texttt{v}\ \texttt{w} \\
&\longrightarrow (\lambda\texttt{z.}\ \texttt{v}\ ((\lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s}\ (\texttt{s}\ \texttt{z}))\ \texttt{v}\ \texttt{z}))\ \texttt{w} \\
&\longrightarrow \texttt{v}\ ((\lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s}\ (\texttt{s}\ \texttt{z}))\ \texttt{v}\ \texttt{w}) \\
&\longrightarrow \texttt{v}\ ((\lambda\texttt{z.}\ \texttt{v}\ (\texttt{v}\ \texttt{z}))\ \texttt{w}) \\
&\longrightarrow \texttt{v}\ (\texttt{v}\ (\texttt{v}\ \texttt{w}))
\end{aligned}
$$

$$
\begin{aligned}
\texttt{c}_3\ \texttt{v}\ \texttt{w} &= (\lambda\texttt{s.}\ \lambda\texttt{z.}\ \texttt{s}\ (\texttt{s}\ (\texttt{s}\ \texttt{z})))\ \texttt{v}\ \texttt{w} \\
&\longrightarrow (\lambda\texttt{z.}\ \texttt{v}\ (\texttt{v}\ (\texttt{v}\ \texttt{z})))\ \texttt{w} \\
&\longrightarrow \texttt{v}\ (\texttt{v}\ (\texttt{v}\ \texttt{w}))
\end{aligned}
$$

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 $\mathtt{v}_1$, $\mathtt{v}_2$, ..., $\mathtt{v}_n$, the applications
>
> $$\mathtt{s}\ \mathtt{v}_1\ \mathtt{v}_2\ \ldots\ \mathtt{v}_n$$
>
> and
>
> $$\mathtt{t}\ \mathtt{v}_1\ \mathtt{v}_2\ \ldots\ \mathtt{v}_n$$
>
> 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 $\texttt{s}$ and $\texttt{t}$ are *not* behaviorally equivalent, it suffices to find a sequence of values $\texttt{v}_1 \ldots \texttt{v}_n$ such that one of

$$\texttt{s}\ \texttt{v}_1\ \texttt{v}_2\ \ldots\ \texttt{v}_n$$

and

$$\texttt{t}\ \texttt{v}_1\ \texttt{v}_2\ \ldots\ \texttt{v}_n$$

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 $(\lambda\texttt{x. x})$ `poisonpill` $(\lambda\texttt{x. x})$ demonstrate that `tru` is not behaviorally equivalent to `fls`:

$$\begin{aligned}&\texttt{tru } (\lambda\texttt{x. x}) \texttt{ poisonpill } (\lambda\texttt{x. x})\\ &\longrightarrow^* (\lambda\texttt{x. x})(\lambda\texttt{x. x})\\ &\longrightarrow^* \lambda\texttt{x. x}\end{aligned}$$

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

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 $\mathtt{v}_1 \ldots \mathtt{v}_n$, either both

$$\mathtt{s}\ \mathtt{v}_1\ \mathtt{v}_2\ \ldots\ \mathtt{v}_n$$

and

$$\mathtt{t}\ \mathtt{v}_1\ \mathtt{v}_2\ \ldots\ \mathtt{v}_n$$

diverge, or else both reach a normal form.

How can we do this?

Proving behavioral equivalence

A general proof technique (so-called *bisimulation*) is beyond the scope of this course. 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 just one element (i.e., $n = 1$), note that both $\mathtt{tru}\ \mathtt{v}_1$ and $\mathtt{tru}'\ \mathtt{v}_1$ reach normal forms after one reduction step.
- 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 (in two steps) 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 $\texttt{v}_1 \ldots \texttt{v}_n$.

Inductive Proofs about the Lambda Calculus

Two induction principles

Like before, we have two ways to prove that properties are true of the untyped lambda calculus.

- Structural induction on terms
- Induction on a derivation of $\mathtt{t} \longrightarrow \mathtt{t}'$.

Let's look at an example of each.

Structural induction on terms

To show that a property $\mathcal{P}$ holds for all lambda-terms $\mathtt{t}$, it suffices to show that

- $\mathcal{P}$ holds when $\mathtt{t}$ is a variable;
- $\mathcal{P}$ holds when $\mathtt{t}$ is a lambda-abstraction $\lambda\mathtt{x}.\ \mathtt{t}_1$, assuming that $\mathcal{P}$ holds for the immediate subterm $\mathtt{t}_1$; and
- $\mathcal{P}$ holds when $\mathtt{t}$ is an application $\mathtt{t}_1\ \mathtt{t}_2$, assuming that $\mathcal{P}$ holds for the immediate subterms $\mathtt{t}_1$ and $\mathtt{t}_2$.

Structural induction on terms

To show that a property $\mathcal{P}$ holds for all lambda-terms $\mathtt{t}$, it suffices to show that

- $\mathcal{P}$ holds when $\mathtt{t}$ is a variable;
- $\mathcal{P}$ holds when $\mathtt{t}$ is a lambda-abstraction $\lambda\mathtt{x}.\ \mathtt{t}_1$, assuming that $\mathcal{P}$ holds for the immediate subterm $\mathtt{t}_1$; and
- $\mathcal{P}$ holds when $\mathtt{t}$ is an application $\mathtt{t}_1\ \mathtt{t}_2$, assuming that $\mathcal{P}$ holds for the immediate subterms $\mathtt{t}_1$ and $\mathtt{t}_2$.

N.b.: The variant of this principle where "immediate subterm" is replaced by "arbitrary subterm" is also valid. (Cf. *ordinary induction* vs. *complete induction* on the natural numbers.)

An example of structural induction on terms

Define the set of *free variables* in a lambda-term as follows:

$$\begin{aligned} FV(\mathtt{x}) &= \{\mathtt{x}\} \\ FV(\lambda\mathtt{x}.\mathtt{t}_1) &= FV(\mathtt{t}_1) \setminus \{\mathtt{x}\} \\ FV(\mathtt{t}_1\ \mathtt{t}_2) &= FV(\mathtt{t}_1) \cup FV(\mathtt{t}_2) \end{aligned}$$

Define the *size* of a lambda-term as follows:

$$\begin{aligned} size(\mathtt{x}) &= 1 \\ size(\lambda\mathtt{x}.\mathtt{t}_1) &= size(\mathtt{t}_1) + 1 \\ size(\mathtt{t}_1\ \mathtt{t}_2) &= size(\mathtt{t}_1) + size(\mathtt{t}_2) + 1 \end{aligned}$$

Theorem: $|FV(\mathtt{t})| \leq size(\mathtt{t})$.

An example of structural induction on terms

Theorem: $|FV(\mathtt{t})| \leq size(\mathtt{t})$.

Proof: By induction on the structure of $\mathtt{t}$.

- If $\mathtt{t}$ is a variable, then $|FV(\mathtt{t})| = 1 = size(\mathtt{t})$.
- If $\mathtt{t}$ is an abstraction $\lambda\mathtt{x.\ t}_1$, then

$$
\begin{array}{cll}
 & |FV(\mathtt{t})| & \\
= & |FV(\mathtt{t}_1) \setminus \{\mathtt{x}\}| & \text{by defn} \\
\leq & |FV(\mathtt{t}_1)| & \text{by arithmetic} \\
\leq & size(\mathtt{t}_1) & \text{by induction hypothesis} \\
\leq & size(\mathtt{t}_1) + 1 & \text{by arithmetic} \\
= & size(\mathtt{t}) & \text{by defn.}
\end{array}
$$

An example of structural induction on terms

Theorem: $|FV(\texttt{t})| \leq size(\texttt{t})$.

Proof: By induction on the structure of $\texttt{t}$.

- If $\texttt{t}$ is an application $\texttt{t}_1\ \texttt{t}_2$, then

$$
\begin{array}{lll}
 & |FV(\texttt{t})| & \\
= & |FV(\texttt{t}_1) \cup FV(\texttt{t}_2)| & \text{by defn} \\
\leq & max(|FV(\texttt{t}_1)|, |FV(\texttt{t}_2)|) & \text{by arithmetic} \\
\leq & max(size(\texttt{t}_1), size(\texttt{t}_2)) & \text{by IH and arithmetic} \\
\leq & |size(\texttt{t}_1)| + |size(\texttt{t}_2)| & \text{by arithmetic} \\
\leq & |size(\texttt{t}_1)| + |size(\texttt{t}_2)| + 1 & \text{by arithmetic} \\
= & size(\texttt{t}) & \text{by defn.}
\end{array}
$$

Induction on derivations

Recall that the reduction relation is defined as the smallest binary relation on terms satisfying the following rules:

$$(\lambda \mathtt{x}.\mathtt{t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \qquad \text{(E-AppAbs)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \qquad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \qquad \text{(E-App2)}$$

Induction on derivations

Induction principle for the small-step evaluation relation.

To show that a property $\mathcal{P}$ holds for all derivations of $\texttt{t} \longrightarrow \texttt{t}'$, it suffices to show that

- $\mathcal{P}$ holds for all derivations that use the rule E-AppAbs;
- $\mathcal{P}$ holds for all derivations that end with a use of E-App1 assuming that $\mathcal{P}$ holds for all subderivations; and
- $\mathcal{P}$ holds for all derivations that end with a use of E-App2 assuming that $\mathcal{P}$ holds for all subderivations.

Example

Theorem: if $\mathtt{t} \longrightarrow \mathtt{t}'$ then $FV(\mathtt{t}) \supseteq FV(\mathtt{t}')$.

Induction on derivations

We must prove, for all derivations of $\mathtt{t} \longrightarrow \mathtt{t}'$, that $FV(\mathtt{t}) \supseteq FV(\mathtt{t}')$.

There are three cases.

Induction on derivations

We must prove, for all derivations of $\mathtt{t} \longrightarrow \mathtt{t}'$, that $FV(\mathtt{t}) \supseteq FV(\mathtt{t}')$.

There are three cases.

- If the derivation of $\mathtt{t} \longrightarrow \mathtt{t}'$ is just a use of E-AppAbs, then $\mathtt{t}$ is $(\lambda\mathtt{x}.\mathtt{t}_1)\mathtt{v}$ and $\mathtt{t}'$ is $[\mathtt{x} \mapsto \mathtt{v}]\mathtt{t}_1$. Reason as follows:

$$\begin{aligned} FV(\mathtt{t}) &= FV((\lambda\mathtt{x}.\mathtt{t}_1)\mathtt{v}) \\ &= FV(\mathtt{t}_1)/\{\mathtt{x}\} \cup FV(\mathtt{v}) \\ &\supseteq FV([\mathtt{x} \mapsto \mathtt{v}]\mathtt{t}_1) \\ &= FV(\mathtt{t}') \end{aligned}$$

- If the derivation ends with a use of E-App1, then $\mathtt{t}$ has the form $\mathtt{t}_1\ \mathtt{t}_2$ and $\mathtt{t}'$ has the form $\mathtt{t}_1'\ \mathtt{t}_2$, and we have a subderivation of $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$

  By the induction hypothesis, $FV(\mathtt{t}_1) \supseteq FV(\mathtt{t}_1')$. Now calculate:

$$\begin{aligned} FV(t) &= FV(\mathtt{t}_1\ \mathtt{t}_2) \\ &= FV(\mathtt{t}_1) \cup FV(\mathtt{t}_2) \\ &\supseteq FV(\mathtt{t}_1') \cup FV(\mathtt{t}_2) \\ &= FV(\mathtt{t}_1'\ \mathtt{t}_2) \\ &= FV(t') \end{aligned}$$

- If the derivation ends with a use of E-App1, then $\mathtt{t}$ has the form $\mathtt{t}_1\ \mathtt{t}_2$ and $\mathtt{t}'$ has the form $\mathtt{t}_1'\ \mathtt{t}_2$, and we have a subderivation of $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$

  By the induction hypothesis, $FV(\mathtt{t}_1) \supseteq FV(\mathtt{t}_1')$. Now calculate:

$$\begin{aligned} FV(t) &= FV(\mathtt{t}_1\ \mathtt{t}_2) \\ &= FV(\mathtt{t}_1) \cup FV(\mathtt{t}_2) \\ &\supseteq FV(\mathtt{t}_1') \cup FV(\mathtt{t}_2) \\ &= FV(\mathtt{t}_1'\ \mathtt{t}_2) \\ &= FV(t') \end{aligned}$$

- If the derivation ends with a use of E-App2, the argument is similar to the previous case.