Lecture
Models of Computation
(DIT310, TDA184)

Nils Anders Danielsson

2016-11-14

Today

χ, a small functional language:

- Concrete and abstract syntax.
- Operational semantics.
- Several variants of the halting problem.
- Representing inductively defined sets.

Concrete syntax

Concrete syntax

$$
\begin{aligned}
e ::= {} & x \\
\mid {} & (e_1 \; e_2) \\
\mid {} & \lambda x.\; e \\
\mid {} & \mathsf{c}(e_1, \ldots, e_n) \\
\mid {} & \mathbf{case}\; e \;\mathbf{of}\; \{\mathsf{c}_1(x_1, \ldots, x_n) \to e_1; \ldots\} \\
\mid {} & \mathbf{rec}\; x = e
\end{aligned}
$$

Variables (x) and constructors (c) are assumed to come from two disjoint, countably infinite sets.

Sometimes extra parentheses are used, and sometimes parentheses are omitted around applications: $e_1\ e_2\ e_3$ means $((e_1\ e_2)\ e_3)$.

Examples

χ	Haskell
$\lambda x.\ e$	`\x -> e`
True()	`True`
Succ(n)	`Succ n`
Cons(x, xs)	`x : xs`
$\mathbf{rec}\ x = e$	`let x = e in x`

Note: Haskell is typed and non-strict, χ is untyped and strict.

Another example

χ:

$$\textbf{case}\ e\ \textbf{of}\ \{\,\mathsf{Zero}() \to x; \mathsf{Succ}(n) \to y\,\}$$

Haskell:

```
case e of
  Zero   -> x
  Succ n -> y
```

And two more

$$
\begin{array}{l}
\mathbf{rec}\ add = \lambda m.\ \lambda n.\ \mathbf{case}\ n\ \mathbf{of} \\
\quad \{\, \mathsf{Zero}() \quad \to m \\
\quad ;\, \mathsf{Succ}(n) \to \mathsf{Succ}(add\ m\ n) \\
\quad \}
\end{array}
$$

$$
\begin{array}{l}
\lambda m.\ \mathbf{rec}\ add = \lambda n.\ \mathbf{case}\ n\ \mathbf{of} \\
\quad \{\, \mathsf{Zero}() \quad \to m \\
\quad ;\, \mathsf{Succ}(n) \to \mathsf{Succ}(add\ n) \\
\quad \}
\end{array}
$$

What is the value of the following expression?

$$
\begin{array}{l}
(\textbf{rec}\ \mathit{foo} = \lambda\, m.\ \lambda\, n.\ \textbf{case}\ n\ \textbf{of}\ \{ \\
\quad \mathsf{Zero}() \quad \to m; \\
\quad \mathsf{Succ}(n) \to \textbf{case}\ m\ \textbf{of}\ \{ \\
\qquad \mathsf{Zero}() \quad \to \mathsf{Zero}(); \\
\qquad \mathsf{Succ}(m) \to \mathit{foo}\ m\ n\}\}) \\
\mathsf{Succ}(\mathsf{Succ}(\mathsf{Zero}()))\ \mathsf{Succ}(\mathsf{Zero}())
\end{array}
$$

- Zero()
- Succ(Zero())
- Succ(Succ(Zero()))
- Succ(Succ(Succ(Zero())))

Abstract syntax

Abstract syntax

$$\frac{x \in Var}{\mathsf{var}\ x \in Exp} \qquad \frac{e_1 \in Exp \qquad e_2 \in Exp}{\mathsf{apply}\ e_1\ e_2 \in Exp}$$

$$\frac{x \in Var \qquad e \in Exp}{\mathsf{lambda}\ x\ e \in Exp} \qquad \frac{x \in Var \qquad e \in Exp}{\mathsf{rec}\ x\ e \in Exp}$$

Var: Assumed to be countably infinite.

Abstract syntax

$$\frac{c \in \mathit{Const} \qquad es \in \mathit{List}\ \mathit{Exp}}{\mathsf{const}\ c\ es \in \mathit{Exp}}$$

$$\frac{e \in \mathit{Exp} \qquad bs \in \mathit{List}\ \mathit{Br}}{\mathsf{case}\ e\ bs \in \mathit{Exp}}$$

$$\frac{c \in \mathit{Const} \qquad xs \in \mathit{List}\ \mathit{Var} \qquad e \in \mathit{Exp}}{\mathsf{branch}\ c\ xs\ e \in \mathit{Br}}$$

Const: Assumed to be countably infinite.

Operational semantics

Operational semantics

- The binary relation $\Downarrow$ relates *closed* expressions.
- An expression is closed if it has
  no free variables.
- $e \Downarrow v$: e terminates with the value v.

Quiz

Which of the following expressions are closed?

- y
- $\lambda x.\ \lambda y.\ x$
- $\textbf{case}\ x\ \textbf{of}\ \{\mathsf{Cons}(x, xs) \to x\}$
- $\textbf{case}\ \mathsf{Succ}(\mathsf{Zero}())\ \textbf{of}\ \{\mathsf{Succ}(x) \to x\}$
- $\textbf{rec}\ f = \lambda x.\ f$

Operational semantics (1/3)

$$\frac{}{\textsf{lambda}\ x\ \ e \Downarrow \textsf{lambda}\ x\ \ e}$$

$$\frac{e_1 \Downarrow \textsf{lambda}\ x\ \ e \qquad e_2 \Downarrow v_2 \qquad e\ [x \leftarrow v_2] \Downarrow v}{\textsf{apply}\ \ e_1\ e_2 \Downarrow v}$$

$$\frac{e\ [x \leftarrow \textsf{rec}\ x\ \ e] \Downarrow v}{\textsf{rec}\ x\ \ e \Downarrow v}$$

Substitution

- $e\,[x \leftarrow e']$: Substitute e' for every *free* occurrence of x in e.
- To keep things simple: e' must be closed.
- If e' is not closed, then this definition is prone to *variable capture*.

Substitution

$$\text{var } x \ [x \leftarrow e'] = e'$$
$$\text{var } y \ [x \leftarrow e'] = \text{var } y \qquad \text{if } x \neq y$$

$$\text{apply } e_1 \ e_2 \ [x \leftarrow e'] =$$
$$\quad \text{apply } (e_1 \ [x \leftarrow e']) \ (e_2 \ [x \leftarrow e'])$$

$$\text{lambda } x \ e \ [x \leftarrow e'] = \text{lambda } x \ e$$
$$\text{lambda } y \ e \ [x \leftarrow e'] =$$
$$\quad \text{lambda } y \ (e \ [x \leftarrow e']) \qquad \text{if } x \neq y$$

And so on...

Quiz

What is the result of

$$(\mathbf{rec}\ y = \mathbf{case}\ x\ \mathbf{of}\ \{\mathsf{c}() \to x; \mathsf{d}(x) \to x\})\ [x \leftarrow \boldsymbol{\lambda} z.\ z]?$$

- $\mathbf{rec}\ y = \mathbf{case}\ x\ \mathbf{of}\ \{\mathsf{c}() \to x; \mathsf{d}(x) \to x\}$
- $\mathbf{rec}\ y = \mathbf{case}\ x\ \mathbf{of}\ \{\mathsf{c}() \to x; \mathsf{d}(x) \to \boldsymbol{\lambda} z.\ z\}$
- $\mathbf{rec}\ y = \mathbf{case}\ \boldsymbol{\lambda} z.\ z\ \mathbf{of}\ \{\mathsf{c}() \to \boldsymbol{\lambda} z.\ z; \mathsf{d}(x) \to x\}$
- $\mathbf{rec}\ y = \mathbf{case}\ \boldsymbol{\lambda} z.\ z\ \mathbf{of}\ \{\mathsf{c}() \to \boldsymbol{\lambda} z.\ z; \mathsf{d}(\boldsymbol{\lambda} z.\ z) \to x\}$
- $\mathbf{rec}\ y = \mathbf{case}\ \boldsymbol{\lambda} z.\ z\ \mathbf{of}\ \{\mathsf{c}() \to \boldsymbol{\lambda} z.\ z; \mathsf{d}(x) \to \boldsymbol{\lambda} z.\ z\}$

Operational semantics (2/3)

$$\frac{es \Downarrow^\star vs}{\mathsf{const}\ c\ \ es \Downarrow \mathsf{const}\ c\ \ vs}$$

$$\frac{}{\mathsf{nil} \Downarrow^\star \mathsf{nil}} \qquad \frac{e \Downarrow v \qquad es \Downarrow^\star vs}{\mathsf{cons}\ e\ \ es \Downarrow^\star \mathsf{cons}\ v\ \ vs}$$

An example

$$\dfrac{\dfrac{}{\mathsf{lambda}\ x\ (\mathsf{var}\ x) \Downarrow \mathsf{lambda}\ x\ (\mathsf{var}\ x)} \qquad \dfrac{\dfrac{}{\mathsf{nil} \Downarrow^\star \mathsf{nil}}}{\mathsf{const}\ c\ \mathsf{nil} \Downarrow \mathsf{const}\ c\ \mathsf{nil}} \qquad \dfrac{\dfrac{}{\mathsf{nil} \Downarrow^\star \mathsf{nil}}}{\mathsf{var}\ x\ [x \leftarrow \mathsf{const}\ c\ \mathsf{nil}] \Downarrow \mathsf{const}\ c\ \mathsf{nil}}}{\mathsf{apply}\ (\mathsf{lambda}\ x\ (\mathsf{var}\ x))\ (\mathsf{const}\ c\ \mathsf{nil}) \Downarrow \mathsf{const}\ c\ \mathsf{nil}}$$

Operational semantics (3/3)

$$\frac{e \Downarrow \mathsf{const}\ c\ vs \qquad \mathit{Lookup}\ c\ bs\ xs\ e' \qquad e'\ [xs \leftarrow vs] \mapsto e'' \qquad e'' \Downarrow v}{\mathsf{case}\ e\ bs \Downarrow v}$$

Operational semantics (3/3)

$$\frac{e \Downarrow \mathsf{const}\ c\ vs \qquad \mathit{Lookup}\ c\ bs\ xs\ e' \qquad e'\,[\,xs \leftarrow vs\,] \mapsto e'' \qquad e'' \Downarrow v}{\mathsf{case}\ e\ bs \Downarrow v}$$

The first matching branch, if any:

$$\frac{}{\mathit{Lookup}\ c\ (\mathsf{cons}\ (\mathsf{branch}\ c\ xs\ e)\ bs)\ xs\ e}$$

$$\frac{c \neq c' \qquad \mathit{Lookup}\ c\ bs\ xs\ e}{\mathit{Lookup}\ c\ (\mathsf{cons}\ (\mathsf{branch}\ c'\ xs'\ e')\ bs)\ xs\ e}$$

Operational semantics (3/3)

$$\frac{e \Downarrow \mathsf{const}\ c\ vs \qquad \mathit{Lookup}\ c\ bs\ xs\ e' \qquad e'\,[xs \leftarrow vs] \mapsto e'' \qquad e'' \Downarrow v}{\mathsf{case}\ e\ bs \Downarrow v}$$

$e\,[xs \leftarrow vs] \mapsto e'$ holds iff

- there is some n such that
  $xs = \mathsf{cons}\ x_1\ (\ldots(\mathsf{cons}\ x_n\ \mathsf{nil}))$ and
  $vs = \mathsf{cons}\ v_1\ (\ldots(\mathsf{cons}\ v_n\ \mathsf{nil}))$, and
- $e' = ((e\,[x_n \leftarrow v_n])\ldots)\,[x_1 \leftarrow v_1]$.

Operational semantics (3/3)

$$\frac{e \Downarrow \mathsf{const}\ c\ vs \qquad \mathit{Lookup}\ c\ bs\ xs\ e' \qquad e'\ [xs \leftarrow vs] \mapsto e'' \qquad e'' \Downarrow v}{\mathsf{case}\ e\ bs \Downarrow v}$$

$$\frac{}{e\ [\mathsf{nil} \leftarrow \mathsf{nil}] \mapsto e}$$

$$\frac{e\ [xs \leftarrow vs] \mapsto e'}{e\ [\mathsf{cons}\ x\ xs \leftarrow \mathsf{cons}\ v\ vs] \mapsto e'\ [x \leftarrow v]}$$

Quiz

Which of the following sets are inhabited?

- $\mathbf{case}\ \mathsf{c}()\ \mathbf{of}\ \{\,\mathsf{c}() \to \mathsf{d}(); \mathsf{c}() \to \mathsf{c}()\,\} \Downarrow \mathsf{c}()$
- $\mathbf{case}\ \mathsf{c}()\ \mathbf{of}\ \{\,\mathsf{c}() \to \mathsf{d}(); \mathsf{c}() \to \mathsf{c}()\,\} \Downarrow \mathsf{d}()$
- $\mathbf{case}\ \mathsf{c}()\ \mathbf{of}\ \{\,\mathsf{c}(x) \to \mathsf{d}(); \mathsf{c}() \to \mathsf{d}()\,\} \Downarrow \mathsf{d}()$
- $\mathbf{case}\ \mathsf{Succ}(\mathsf{False}())\ \mathbf{of}$
  $\{\,\mathsf{Zero}() \to \mathsf{True}(); \mathsf{Succ}(n) \to n\,\} \Downarrow \mathsf{False}()$
- $\mathbf{case}\ \mathsf{Succ}(\mathsf{False}())\ \mathbf{of}$
  $\{\,\mathsf{Zero}() \to \mathsf{True}(); \mathsf{Succ}() \to \mathsf{False}()\,\}$
  $\Downarrow \mathsf{False}()$

Some properties

Deterministic

The semantics is deterministic:
$e \Downarrow v_1$ and $e \Downarrow v_2$ imply $v_1 = v_2$.

Values

- An expression e is called a value if $e \Downarrow e$.
- Values can be characterised inductively:

$$\frac{}{\mathit{Value}\ (\mathsf{lambda}\ x\ e)} \qquad \frac{\mathit{Values}\ es}{\mathit{Value}\ (\mathsf{const}\ c\ es)}$$

$$\frac{}{\mathit{Values}\ \mathsf{nil}} \qquad \frac{\mathit{Value}\ e \qquad \mathit{Values}\ es}{\mathit{Value}\ (\mathsf{cons}\ e\ es)}$$

- $\mathit{Value}\ e$ holds iff $e \Downarrow e$.
- If $e \Downarrow v$, then $\mathit{Value}\ v$.

There is a non-terminating expression

- The following program does not terminate: $\text{rec } x \; (\text{var } x)$.
- Recall the rule for rec: $\dfrac{e\;[x \leftarrow \text{rec } x \;\; e] \Downarrow v}{\text{rec } x \;\; e \Downarrow v}$.
- Note that $\text{var } x \; [x \leftarrow \text{rec } x \; (\text{var } x)] = \text{rec } x \; (\text{var } x)$.
- Idea:

$$\begin{array}{ll} \text{rec } x \; (\text{var } x) & \rightarrow \\ \text{var } x \; [x \leftarrow \text{rec } x \; (\text{var } x)] & = \\ \text{rec } x \; (\text{var } x) & \rightarrow \\ \vdots & \end{array}$$

There is a non-terminating expression

- If the program did terminate, then there would be a *finite* derivation of the following form:

$$\dfrac{\dfrac{\dfrac{\vdots}{\textsf{rec}\ x\ (\textsf{var}\ x) \Downarrow v}}{\textsf{rec}\ x\ (\textsf{var}\ x) \Downarrow v}}{\textsf{rec}\ x\ (\textsf{var}\ x) \Downarrow v}$$

- Exercise: Prove more formally that this is impossible, using induction on the structure of the semantics.

The halting problem

The extensional halting problem

There is no closed expression $halts$ such that, for every closed expression p,

- $halts\ (\lambda x.\ p) \Downarrow$ True(), if p terminates, and
- $halts\ (\lambda x.\ p) \Downarrow$ False(), otherwise.

The extensional halting problem

- Assume that $halts$ can be defined.
- Define $terminv \in Exp \to Exp$:

$$\begin{aligned} terminv\ p = \mathbf{case}\ & halts\ (\lambda x.\ p)\ \mathbf{of} \\ & \{\, \mathsf{True}() \ \to \mathsf{rec}\ x = x \\ & ;\, \mathsf{False}() \to \mathsf{Zero}() \\ & \} \end{aligned}$$

- For any closed expression p:
  $terminv\ p$ terminates iff p does not terminate.

The extensional halting problem

- Now consider the closed expression $\mathit{strange}$ defined by $\mathbf{rec}\ p = \mathit{terminv}\ p$.
- We get a contradiction:

$$
\begin{array}{lll}
(\exists v.\ \mathit{strange} & \Downarrow v) & \Leftrightarrow \\
(\exists v.\ \mathbf{rec}\ p = \mathit{terminv}\ p & \Downarrow v) & \Leftrightarrow \\
(\exists v.\ \mathit{terminv}\ p\,[p \leftarrow \mathit{strange}] & \Downarrow v) & \Leftrightarrow \\
(\exists v.\ \mathit{terminv}\ \mathit{strange} & \Downarrow v) & \Leftrightarrow \\
\neg\,(\exists v.\ \mathit{strange} & \Downarrow v) &
\end{array}
$$

The extensional halting problem

- Note that we apply $halts$ to a program, not to the source code of a program.
- How can source code be represented?

Representing inductively defined sets

Natural numbers

One method:

- Notation: $\ulcorner n \urcorner \in Exp$ represents $n \in \mathbb{N}$.
- Representation:

$$\begin{aligned} \ulcorner \mathsf{zero} \urcorner &= \mathsf{Zero}() \\ \ulcorner \mathsf{suc}\ n \urcorner &= \mathsf{Succ}(\ulcorner n \urcorner) \end{aligned}$$

Natural numbers

One method:

- Notation: $\ulcorner n \urcorner \in \mathit{Exp}$ represents $n \in \mathbb{N}$.
- Representation:

$$\begin{aligned}
\ulcorner \textsf{zero} \urcorner &= \textsf{Zero}() \\
\ulcorner \textsf{suc}\ n \urcorner &= \textsf{Succ}(\ulcorner n \urcorner)
\end{aligned}$$

- Note that the concrete syntax should be interpreted as abstract syntax:

$$\begin{aligned}
\ulcorner \textsf{zero} \urcorner &= \textsf{const}\ \underline{\mathit{Zero}}\ \textsf{nil} \\
\ulcorner \textsf{suc}\ n \urcorner &= \textsf{const}\ \underline{\mathit{Succ}}\ (\textsf{cons}\ \ulcorner n \urcorner\ \textsf{nil})
\end{aligned}$$

(For some distinct $\underline{\mathit{Zero}}, \underline{\mathit{Succ}} \in \mathit{Const}$.)

Lists

If elements in A can be represented, then elements in $List\ A$ can also be represented:

$$\begin{aligned}
\ulcorner \mathsf{nil} \urcorner &= \mathsf{Nil}() \\
\ulcorner \mathsf{cons}\ x\ xs \urcorner &= \mathsf{Cons}(\ulcorner x \urcorner, \ulcorner xs \urcorner)
\end{aligned}$$

Many inductively defined sets can be represented using constructor trees in analogous ways.

Variables, constants

- Var: Countably infinite.
- Thus each variable $x \in Var$ can be assigned a unique natural number $n \in \mathbb{N}$.
- Define $\ulcorner x \urcorner = \ulcorner n \urcorner$.
- Similarly for constants.

Source code

$$
\begin{aligned}
\ulcorner \textsf{var}\ x \urcorner &= \textsf{Var}(\ulcorner x \urcorner) \\
\ulcorner \textsf{apply}\ e_1\ e_2 \urcorner &= \textsf{Apply}(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner) \\
\ulcorner \textsf{lambda}\ x\ e \urcorner &= \textsf{Lambda}(\ulcorner x \urcorner, \ulcorner e \urcorner) \\
\ulcorner \textsf{rec}\ x\ e \urcorner &= \textsf{Rec}(\ulcorner x \urcorner, \ulcorner e \urcorner) \\
\ulcorner \textsf{const}\ c\ es \urcorner &= \textsf{Const}(\ulcorner c \urcorner, \ulcorner es \urcorner) \\
\ulcorner \textsf{case}\ e\ bs \urcorner &= \textsf{Case}(\ulcorner e \urcorner, \ulcorner bs \urcorner) \\
\ulcorner \textsf{branch}\ c\ xs\ e \urcorner &= \textsf{Branch}(\ulcorner c \urcorner, \ulcorner xs \urcorner, \ulcorner e \urcorner)
\end{aligned}
$$

Example

- Concrete syntax: $\lambda x.\ \mathsf{Succ}(x)$.
- Abstract syntax:

  $$\mathsf{lambda}\ \underline{x}\ (\mathsf{const}\ \underline{Succ}\ (\mathsf{cons}\ (\mathsf{var}\ \underline{x})\ \mathsf{nil}))$$

  (for some $\underline{x} \in Var$ and $\underline{Succ} \in Const$).
- Representation (concrete syntax):

  $$\mathsf{Lambda}(\ulcorner \underline{x} \urcorner,\ \mathsf{Const}(\ulcorner \underline{Succ} \urcorner, \mathsf{Cons}(\mathsf{Var}(\ulcorner \underline{x} \urcorner), \mathsf{Nil}())))$$

- If $\underline{x}$ and $\underline{Succ}$ both correspond to zero:

  $$\mathsf{Lambda}(\mathsf{Zero}(),\ \mathsf{Const}(\mathsf{Zero}(),\ \mathsf{Cons}(\mathsf{Var}(\mathsf{Zero}()), \mathsf{Nil}())))$$

Example

Representation (abstract syntax):

```
const Lambda (
  cons (const Zero nil) (
  cons (const Const (
    cons (const Zero nil) (
    cons (const Cons (
      cons (const Var (cons (const Zero nil) nil)) (
      cons (const Nil nil)
      nil)))
    nil)))
  nil))
```

Quiz

How is $\mathbf{rec}\ x = x$ represented?
Assume that x corresponds to 1.

- $\mathsf{Rec}(\mathsf{X}(), \mathsf{X}())$
- $\mathsf{Rec}(\mathsf{X}(), \mathsf{Var}(\mathsf{X}()))$
- $\mathsf{Equals}(\mathsf{Rec}(\mathsf{X}()), \mathsf{X}())$
- $\mathsf{Rec}(\mathsf{Succ}(\mathsf{Zero}()), \mathsf{Succ}(\mathsf{Zero}()))$
- $\mathsf{Rec}(\mathsf{Succ}(\mathsf{Zero}()), \mathsf{Var}(\mathsf{Succ}(\mathsf{Zero}())))$
- $\mathsf{Equals}(\mathsf{Rec}(\mathsf{Succ}(\mathsf{Zero}())), \mathsf{Succ}(\mathsf{Zero}()))$

The halting problem, take two

The intensional halting problem (with self-application)

There is no closed expression $halts$ such that,
for every closed expression p,

- $halts \ulcorner p \urcorner \Downarrow \mathsf{True}()$, if $p \ulcorner p \urcorner$ terminates, and
- $halts \ulcorner p \urcorner \Downarrow \mathsf{False}()$, otherwise.

With self-application

- Assume that $halts$ can be defined.
- Define the closed expression $terminv$:

$$
\begin{aligned}
terminv = \lambda p.\ \mathbf{case}\ halts\ p\ \mathbf{of}\ & \\
& \{\,\mathsf{True}()\ \rightarrow \mathsf{rec}\ x = x \\
& ;\ \mathsf{False}() \rightarrow \mathsf{Zero}() \\
& \}
\end{aligned}
$$

- For any closed expression p:
  $terminv \ulcorner p \urcorner$ terminates iff
  $p \ulcorner p \urcorner$ does not terminate.
- Thus $terminv \ulcorner terminv \urcorner$ terminates iff
  $terminv \ulcorner terminv \urcorner$ does not terminate.

The intensional halting problem

There is no closed expression $halts$ such that,
for every closed expression p,

- $halts \ulcorner p \urcorner \Downarrow \mathsf{True}()$, if p terminates, and
- $halts \ulcorner p \urcorner \Downarrow \mathsf{False}()$, otherwise.

The intensional halting problem

- Assume that $halts$ can be defined.
- If we can use $halts$ to solve the previous variant of the halting problem, then we have found a contradiction.

The intensional halting problem

- Exercise:
  Define a closed expression $code$ satisfying:
  - For any closed expression p,
    $code \ulcorner p \urcorner \Downarrow \ulcorner \ulcorner p \urcorner \urcorner$.
- Define the closed expression $halts'$ by
  $\lambda\, p.\ halts\ \mathsf{Apply}(p, code\ p)$.

The intensional halting problem

For any closed expression p:

$$
\begin{array}{lll}
p \ulcorner p \urcorner \text{ terminates} & & \Rightarrow \\
\mathit{halts} \ulcorner p \ulcorner p \urcorner \urcorner & \Downarrow \mathsf{True}() & \Rightarrow \\
\mathit{halts}\ \mathsf{Apply}(\ulcorner p \urcorner, \ulcorner \ulcorner p \urcorner \urcorner) & \Downarrow \mathsf{True}() & \Rightarrow \\
\mathit{halts}\ \mathsf{Apply}(\ulcorner p \urcorner, \mathit{code} \ulcorner p \urcorner) & \Downarrow \mathsf{True}() & \Rightarrow \\
\mathit{halts'} \ulcorner p \urcorner & \Downarrow \mathsf{True}() &
\end{array}
$$

The intensional halting problem

For any closed expression p:

$$\begin{array}{llll}
p \ulcorner p \urcorner \text{ does not terminate} & & & \Rightarrow \\
\mathit{halts} \ulcorner p \ulcorner p \urcorner \urcorner & \Downarrow & \mathsf{False}() & \Rightarrow \\
\mathit{halts}\ \mathsf{Apply}(\ulcorner p \urcorner, \ulcorner \ulcorner p \urcorner \urcorner) & \Downarrow & \mathsf{False}() & \Rightarrow \\
\mathit{halts}\ \mathsf{Apply}(\ulcorner p \urcorner, \mathit{code} \ulcorner p \urcorner) & \Downarrow & \mathsf{False}() & \Rightarrow \\
\mathit{halts}' \ulcorner p \urcorner & \Downarrow & \mathsf{False}() &
\end{array}$$

Thus halts' solves the previous variant of the halting problem, and we have found a contradiction.

Summary

- Concrete and abstract syntax.
- Operational semantics.
- Several variants of the halting problem.
- Representing inductively defined sets.