Lecture
Models of Computation
(DIT310, TDA184)

Nils Anders Danielsson

2017-11-13

Today

X, 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

X	Haskell
$\lambda x.\, e$	`\x -> e`
True()	`True`
Suc(n)	`Suc 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

X:

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

Haskell:

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

And two more

$$\textbf{rec}\ add = \lambda m.\,\lambda n.\,\textbf{case}\ n\ \textbf{of}$$
$$\{\,\mathsf{Zero}()\ \to m$$
$$;\,\mathsf{Suc}(n) \to \mathsf{Suc}(add\ m\ n)$$
$$\}$$

$$\lambda m.\,\textbf{rec}\ add = \lambda n.\,\textbf{case}\ n\ \textbf{of}$$
$$\{\,\mathsf{Zero}()\ \to m$$
$$;\,\mathsf{Suc}(n) \to \mathsf{Suc}(add\ n)$$
$$\}$$

What is the following expression's value?

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

- Zero()
- Suc(Zero())
- Suc(Suc(Zero()))
- Suc(Suc(Suc(Zero())))

Abstract syntax

Abstract syntax

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

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

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

- $e \Downarrow v$: e terminates with the value v.
- The expression e terminates if $\exists v.\, e \Downarrow v$.
- Note that a “crash” does not count as termination.
- The binary relation $\Downarrow$ relates *closed* expressions.
- An expression is closed if it has no free variables.

Quiz

Which of the following expressions are closed?

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

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

$$(\mathbf{rec}\ y = \mathbf{case}\ x\ \mathbf{of}\ \{\,\mathsf{C}() \to x; \mathsf{D}(x) \to x\,\})\ [x \leftarrow \lambda\, z.\, z]?$$

$$\begin{array}{llllll}
\mathbf{rec}\ y = \mathbf{case}\ x & \mathbf{of}\ \{\,\mathsf{C}() \to x; & \mathsf{D}(x) & \to x & \} \\
\mathbf{rec}\ y = \mathbf{case}\ x & \mathbf{of}\ \{\,\mathsf{C}() \to \lambda\, z.\, z; & \mathsf{D}(x) & \to x & \} \\
\mathbf{rec}\ y = \mathbf{case}\ \lambda\, z.\, z & \mathbf{of}\ \{\,\mathsf{C}() \to x; & \mathsf{D}(x) & \to x & \} \\
\mathbf{rec}\ y = \mathbf{case}\ \lambda\, z.\, z & \mathbf{of}\ \{\,\mathsf{C}() \to \lambda\, z.\, z; & \mathsf{D}(x) & \to x & \} \\
\mathbf{rec}\ y = \mathbf{case}\ \lambda\, z.\, z & \mathbf{of}\ \{\,\mathsf{C}() \to \lambda\, z.\, z; & \mathsf{D}(x) & \to \lambda\, z.\, z & \} \\
\mathbf{rec}\ y = \mathbf{case}\ \lambda\, z.\, z & \mathbf{of}\ \{\,\mathsf{C}() \to \lambda\, z.\, z; & \mathsf{D}(\lambda\, z.\, z) & \to \lambda\, z.\, z & \}
\end{array}$$

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

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

Operational semantics (3/3)

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

Operational semantics (3/3)

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

The first matching branch, if any:

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

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

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?

$$\textbf{case}\ \mathsf{C}()\ \textbf{of}\ \{\, \mathsf{C}() \ \ \to \mathsf{D}(); \mathsf{C}() \to \mathsf{C}() \,\} \Downarrow \mathsf{C}()$$

$$\textbf{case}\ \mathsf{C}()\ \textbf{of}\ \{\, \mathsf{C}() \ \ \to \mathsf{D}(); \mathsf{C}() \to \mathsf{C}() \,\} \Downarrow \mathsf{D}()$$

$$\textbf{case}\ \mathsf{C}()\ \textbf{of}\ \{\, \mathsf{C}(x) \to \mathsf{D}(); \mathsf{C}() \to \mathsf{D}() \,\} \Downarrow \mathsf{D}()$$

$$\textbf{case}\ \mathsf{C}(\mathsf{C}(), \mathsf{D}())\ \textbf{of}\ \{\, \mathsf{C}(x, x) \to x \,\} \Downarrow \mathsf{C}()$$

$$\textbf{case}\ \mathsf{Suc}(\mathsf{False}())\ \textbf{of}$$
$$\quad \{\, \mathsf{Zero}() \to \mathsf{True}(); \mathsf{Suc}(n) \to n \,\} \Downarrow \mathsf{False}()$$

$$\textbf{case}\ \mathsf{Suc}(\mathsf{False}())\ \textbf{of}$$
$$\quad \{\, \mathsf{Zero}() \to \mathsf{True}(); \mathsf{Suc}() \to \mathsf{False}() \,\} \Downarrow \mathsf{False}()$$

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:

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

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

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

There is a non-terminating expression

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

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

The extensional halting problem

Note the abuse of notation:

- The variables $halts$ and p are not χ variables.
- *Meta-variables* standing for χ expressions.
- An alternative is to use abstract syntax:

  $$\text{apply } halts\ (\text{lambda } \underline{x}\ p) \Downarrow \text{const } \underline{True}\ \text{ nil}$$
  $$\text{apply } halts\ (\text{lambda } \underline{x}\ p) \Downarrow \text{const } \underline{False}\ \text{ nil}$$

  (For *distinct* $\underline{True}$, $\underline{False} \in Const$.)
- More verbose.

The extensional halting problem

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

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

- 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:

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

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

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

Natural numbers

One method:

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

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

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

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

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

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 $code\ x \in \mathbb{N}$.
- Define $\ulcorner x \urcorner = \ulcorner code\ x \urcorner$.
- Similarly for constants.

Variables, constants

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

Source code

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

Example

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

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

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

  $$\begin{aligned}&\mathsf{Lambda}(\ulcorner \underline{x} \urcorner,\\ &\qquad\qquad \mathsf{Const}(\ulcorner \underline{Suc} \urcorner, \mathsf{Cons}(\mathsf{Var}(\ulcorner \underline{x} \urcorner), \mathsf{Nil}()))))\end{aligned}$$

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

  $$\begin{aligned}&\mathsf{Lambda}(\mathsf{Zero}(),\\ &\qquad\qquad \mathsf{Const}(\mathsf{Zero}(),\\ &\qquad\qquad\qquad\qquad \mathsf{Cons}(\mathsf{Var}(\mathsf{Zero}()), \mathsf{Nil}()))))\end{aligned}$$

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.

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

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,

- $halts \ulcorner p \urcorner \Downarrow$ True(), if $p \ulcorner p \urcorner$ terminates, and
- $halts \ulcorner p \urcorner \Downarrow$ 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,

- $halts \ulcorner p \urcorner \Downarrow$ True(), if p terminates, and
- $halts \ulcorner p \urcorner \Downarrow$ 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

$$code \ulcorner p \urcorner \Downarrow \ulcorner \ulcorner p \urcorner \urcorner$$

for any closed expression p.

The intensional halting problem

Exercise: Define a closed expression $code$ satisfying

$$code \ulcorner p \urcorner \Downarrow \ulcorner \ulcorner p \urcorner \urcorner$$

for any closed expression p.

Example:

$$\ulcorner \ulcorner \lambda x. x \urcorner \urcorner$$

The intensional halting problem

Exercise: Define a closed expression $code$ satisfying

$$code \ulcorner p \urcorner \Downarrow \ulcorner \ulcorner p \urcorner \urcorner$$

for any closed expression p.

Example:

$$\ulcorner \ulcorner \text{lambda } \underline{x} \, (\text{var } \underline{x}) \urcorner \urcorner$$

The intensional halting problem

Exercise: Define a closed expression $code$ satisfying

$$code \ulcorner p \urcorner \Downarrow \ulcorner \ulcorner p \urcorner \urcorner$$

for any closed expression p.

Example:

$$\ulcorner \mathsf{Lambda}(\ulcorner \underline{x} \urcorner, \mathsf{Var}(\ulcorner \underline{x} \urcorner)) \urcorner$$

The intensional halting problem

Exercise: Define a closed expression $code$ satisfying

$$code \ulcorner p \urcorner \Downarrow \ulcorner \ulcorner p \urcorner \urcorner$$

for any closed expression p.

Example:

$$\ulcorner \mathsf{Lambda}(\mathsf{Zero}(), \mathsf{Var}(\mathsf{Zero}())) \urcorner$$

The intensional halting problem

Exercise: Define a closed expression $code$ satisfying

$$code \ulcorner p \urcorner \Downarrow \ulcorner \ulcorner p \urcorner \urcorner$$

for any closed expression p.

Example:

$\ulcorner$ const $\underline{Lambda}$ (
cons $\ulcorner$ Zero() $\urcorner$ (
cons $\ulcorner$ Var(Zero()) $\urcorner$
nil)) $\urcorner$

The intensional halting problem

Exercise: Define a closed expression $code$ satisfying

$$code \ulcorner p \urcorner \Downarrow \ulcorner\ulcorner p \urcorner\urcorner$$

for any closed expression p.

Example:

$\ulcorner$ const $\underline{Lambda}$ (
cons (const $\underline{Zero}$ nil) (
cons $\ulcorner$ Var(Zero()) $\urcorner$
nil)) $\urcorner$

The intensional halting problem

Exercise: Define a closed expression $code$ satisfying

$$code \ulcorner p \urcorner \Downarrow \ulcorner \ulcorner p \urcorner \urcorner$$

for any closed expression p.

Example:

$\ulcorner$ const $\underline{Lambda}$ (
cons (const $\underline{Zero}$ nil) (
cons (const $\underline{Var}$ (cons (const $\underline{Zero}$ nil) nil))
nil)) $\urcorner$

The intensional halting problem

Exercise: Define a closed expression $code$ satisfying

$$code \ulcorner p \urcorner \Downarrow \ulcorner \ulcorner p \urcorner \urcorner$$

for any closed expression p.

Example:

$$\begin{aligned}
&\mathsf{Const}(\ulcorner \underline{Lambda} \urcorner, \\
&\quad \mathsf{Cons}(\mathsf{Const}(\ulcorner \underline{Zero} \urcorner, \mathsf{Nil}()), \\
&\quad \mathsf{Cons}(\mathsf{Const}(\ulcorner \underline{Var} \urcorner, \\
&\qquad\qquad \mathsf{Cons}(\mathsf{Const}(\ulcorner \underline{Zero} \urcorner, \mathsf{Nil}()), \\
&\qquad\qquad \mathsf{Nil}()))), \\
&\quad \mathsf{Nil}())))
\end{aligned}$$

The intensional halting problem

Exercise: Define a closed expression $code$ satisfying

$$code \ulcorner p \urcorner \Downarrow \ulcorner \ulcorner p \urcorner \urcorner$$

for any closed expression p.

Example:

```
Const(Suc(Zero()),
   Cons(Const(Suc(Suc(Zero())), Nil()),
   Cons(Const(Suc(Suc(Suc(Zero())))),
            Cons(Const(Suc(Suc(Zero())), Nil()),
            Nil())),
   Nil())))
```

The intensional halting problem

Define the closed expression $halts'$ by

$$\lambda\, p.\, halts\ \mathsf{Apply}(p, code\ p).$$

For any closed expression p:

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

The intensional halting problem

Define the closed expression $halts'$ by

$$\lambda\, p.\, halts\ \mathsf{Apply}(p, code\ p).$$

For any closed expression p:

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

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.