Lecture
Models of Computation
(DIT310, TDA184)

Nils Anders Danielsson

2017-11-20

Today

- X-computability.
- A self-interpreter for χ.
- Reductions.
- More problems that are or are not computable.
- More about coding.

X-computability

X-computable functions

Assume that we have methods for representing members of the sets A and B as closed χ expressions.

A partial function $f \in A \rightharpoonup B$ is χ-*computable* (with respect to these methods) if there is a closed expression e such that:

- $\forall a \in A$.
  if $f\ a$ is defined then $e \ulcorner a \urcorner \Downarrow \ulcorner f\ a \urcorner$.
- $\forall a \in A$, $v \in Exp$.
  if $e \ulcorner a \urcorner \Downarrow v$ then $f\ a$ is defined and $v = \ulcorner f\ a \urcorner$.

χ-computable functions

A special case:

A (total) function $f \in A \to B$ is χ-computable if there is a closed expression e such that:

- $\forall a \in A.\ e \ulcorner a \urcorner \Downarrow \ulcorner f\ a \urcorner$.

An alternative characterisation

- Define $CExp = \{ p \in Exp \mid p \text{ is closed} \}$.
- The semantics as a partial function:

$$\begin{aligned} \llbracket \_ \rrbracket &\in CExp \rightharpoonup CExp \\ \llbracket p \rrbracket &= v \quad \text{if } p \Downarrow v \end{aligned}$$

- $f \in A \rightharpoonup B$ is χ-computable iff

$$\exists\, e \in CExp.\ \forall\, a \in A.\, \llbracket e \ulcorner a \urcorner \rrbracket = \ulcorner f\ a \urcorner.$$

Quiz

What would go "wrong" if we decided to represent closed χ expressions in the following way?

A closed χ expression is represented by $\mathrm{True}()$ if it terminates, and by $\mathrm{False}()$ otherwise.

Representation

- The choice of representation is important.
- In this course (unless otherwise noted or inapplicable): The "standard" representation.
- It might make sense to require that the representation function $\ulcorner \_ \urcorner$ is "computable".
  - However, how should this be defined?

Examples

- Addition of natural numbers is χ-computable:

$$\begin{aligned} &add \in \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\ &add\ (m, n) = m + n \end{aligned}$$

- The intensional halting problem is not χ-computable:

$$\begin{aligned} &halts \in CExp \to Bool \\ &halts\ p = \mathbf{if}\ p\ \text{terminates}\ \mathbf{then}\ \text{true}\ \mathbf{else}\ \text{false} \end{aligned}$$

- The semantics $[\![\_]\!]$ is computable.

Self-interpreter

Self-interpreter

Goal: Define $\mathit{eval} \in \mathit{CExp}$ satisfying:

- $\forall e, v \in \mathit{CExp}$,
  if $e \Downarrow v$ then $\mathit{eval} \ulcorner e \urcorner \Downarrow \ulcorner v \urcorner$.
- $\forall e, v' \in \mathit{CExp}$,
  if $\mathit{eval} \ulcorner e \urcorner \Downarrow v'$ then there is some v such that $e \Downarrow v$ and $v' = \ulcorner v \urcorner$.

Or: $\forall e \in \mathit{CExp}.\ [\![\mathit{eval} \ulcorner e \urcorner]\!] = \ulcorner [\![e]\!] \urcorner$.

Self-interpreter

$$\textbf{rec}\ eval = \lambda\, e.\, \textbf{case}\ e\ \textbf{of}$$
$$\quad \{\ldots$$
$$\quad \}$$

Self-interpreter

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

$$\textsf{Lambda}(x, e) \rightarrow \textsf{Lambda}(x, e)$$

Self-interpreter

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

$$\begin{array}{l}\textsf{Apply}(e_1, e_2) \rightarrow \textbf{case}\ eval\ \ e_1\ \textbf{of} \\ \quad \{\,\textsf{Lambda}(x, e) \rightarrow eval\ (subst\ x\ (eval\ \ e_2)\ e) \\ \quad \}\end{array}$$

Exercise: Define $subst$.

Self-interpreter

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

$$\textsf{Rec}(x, e) \to eval\ (subst\ x\ \textsf{Rec}(x, e)\ \ e)$$

Self-interpreter

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

$\mathsf{Const}(c, es) \rightarrow \mathsf{Const}(c, map\ \ eval\ \ es)$

Exercise: Define map.

Self-interpreter

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

$$
\begin{array}{l}
\mathsf{Case}(e, bs) \rightarrow \textbf{case}\ \mathit{eval}\ e\ \textbf{of} \\
\quad \{ \mathsf{Const}(c, vs) \rightarrow \textbf{case}\ \mathit{lookup}\ c\ bs\ \textbf{of} \\
\qquad \{ \mathsf{Pair}(xs, e') \rightarrow \mathit{eval}\ (\mathit{substs}\ xs\ vs\ e') \\
\qquad \} \\
\quad \}
\end{array}
$$

Exercise: Define *lookup* and *substs*.

Self-interpreter

$$
\begin{array}{l}
\textbf{rec}\ \mathit{eval} = \lambda\, e.\ \textbf{case}\ e\ \textbf{of} \\
\quad \{\, \mathsf{Lambda}(x, e) \rightarrow \mathsf{Lambda}(x, e) \\
\quad ;\ \mathsf{Apply}(e_1, e_2)\ \ \rightarrow \textbf{case}\ \mathit{eval}\ e_1\ \textbf{of} \\
\qquad \{\, \mathsf{Lambda}(x, e) \rightarrow \mathit{eval}\ (\mathit{subst}\ x\ (\mathit{eval}\ e_2)\ e) \,\} \\
\quad ;\ \mathsf{Rec}(x, e) \qquad \rightarrow \mathit{eval}\ (\mathit{subst}\ x\ \mathsf{Rec}(x, e)\ e) \\
\quad ;\ \mathsf{Const}(c, es) \rightarrow \mathsf{Const}(c, \mathit{map}\ \mathit{eval}\ es) \\
\quad ;\ \mathsf{Case}(e, bs)\ \ \rightarrow \textbf{case}\ \mathit{eval}\ e\ \textbf{of} \\
\qquad \{\, \mathsf{Const}(c, vs) \rightarrow \textbf{case}\ \mathit{lookup}\ c\ bs\ \textbf{of} \\
\qquad\quad \{\, \mathsf{Pair}(xs, e') \rightarrow \mathit{eval}\ (\mathit{substs}\ xs\ vs\ e') \,\} \\
\qquad \} \\
\quad \}
\end{array}
$$

Note: subst, map, lookup and substs are meta-variables that stand for (closed) expressions.

Quiz

Is the following partial function χ-computable?

$$
\begin{aligned}
&halts \in CExp \rightharpoonup Bool \\
&halts\ p = \\
&\quad \textbf{if}\ p\ \text{terminates}\ \textbf{then}\ \text{true}\ \textbf{else}\ \text{undefined}
\end{aligned}
$$

$\mathcal{X}$-decidable

A function $f \in A \to Bool$ is χ-*decidable* if it is χ-computable. If not, then it is χ-*undecidable*.

X-semi-decidable

A function $f \in A \rightarrow Bool$ is χ*-semi-decidable* if there is a closed expression e such that, for all $a \in A$:

- If $f\ a =$ true then $e \ulcorner a \urcorner \Downarrow \ulcorner \text{true} \urcorner$.
- If $f\ a =$ false then $e \ulcorner a \urcorner$ does not terminate.

The halting problem is semi-decidable

The halting problem:

$$halts \in CExp \to Bool$$
$$halts\ p = \textbf{if}\ p \text{ terminates } \textbf{then} \text{ true } \textbf{else} \text{ false}$$

A program witnessing the semi-decidability:

$$\lambda\, p.\, (\lambda\, \_.\, \mathsf{True}())\ (eval\ p)$$

Reductions

Reductions (one variant)

A χ-reduction of $f \in A \rightharpoonup B$ to $g \in C \rightharpoonup D$ consists of a proof showing that, if g is χ-computable, then f is χ-computable.

Reductions (one variant)

A χ-reduction of $f \in A \rightharpoonup B$ to $g \in C \rightharpoonup D$ consists of a proof showing that, if g is χ-computable, then f is χ-computable.

- If f is reducible to g, and f is not computable, then g is not computable.
- Last week we proved that the halting problem is undecidable by reducing another problem to it.

More (un)decidable problems

Semantic equality

- Are two closed χ expressions semantically equal?

$$\begin{aligned}
&\mathit{equal} \in \mathit{CExp} \times \mathit{CExp} \to \mathit{Bool} \\
&\mathit{equal}\ (e_1, e_2) = \\
&\quad \mathbf{if}\ [\![e_1]\!] = [\![e_2]\!]\ \mathbf{then}\ \mathsf{true}\ \mathbf{else}\ \mathsf{false}
\end{aligned}$$

- The halting problem reduces to this one:

$$\mathit{halts} = \lambda\, p.\ \mathit{not}\ (\mathit{equal}\ \mathsf{Pair}(p, \ulcorner\, \mathbf{rec}\ x = x\, \urcorner))$$

Pointwise equality

- Pointwise equality:

$$
\begin{array}{l}
\mathit{pointwise\text{-}equal} \in \mathit{CExp} \times \mathit{CExp} \to \mathit{Bool} \\
\mathit{pointwise\text{-}equal}\ (e_1, e_2) = \\
\quad \textbf{if}\ \forall\ e \in \mathit{CExp}.\, [\![e_1\ e]\!] = [\![e_2\ e]\!] \\
\quad \textbf{then}\ \mathsf{true}\ \textbf{else}\ \mathsf{false}
\end{array}
$$

- The previous problem reduces to this one:

$$
\begin{array}{l}
\mathit{equal} = \lambda\, p.\, \textbf{case}\ p\ \textbf{of} \\
\quad \{\, \mathsf{Pair}(e_1, e_2) \to \\
\qquad\quad \mathit{pointwise\text{-}equal} \\
\qquad\qquad \mathsf{Pair}(\mathsf{Lambda}(\mathsf{Zero}(), e_1), \\
\qquad\qquad\qquad \mathsf{Lambda}(\mathsf{Zero}(), e_2)) \\
\quad \}
\end{array}
$$

Termination in n steps

- Termination in n steps:

$$\begin{aligned}
&\mathit{terminates\text{-}in} \in \mathit{CExp} \times \mathbb{N} \to \mathit{Bool} \\
&\mathit{terminates\text{-}in}\ (e, n) = \\
&\quad \mathbf{if}\ \exists\, v.\ \exists\, p \in e \Downarrow v.\, |\, p\, | \leq n \\
&\quad \mathbf{then}\ \textsf{true}\ \mathbf{else}\ \textsf{false}
\end{aligned}$$

  $|p|$: The number of rules in the derivation tree.
- Decidable: We can define a variant of the self-interpreter that tries to evaluate e but stops if more than n rules are needed.

Representation

- How do we represent a χ-computable function?
- Example:

$$\{ f \in \mathbb{N} \to \mathbb{N} \mid f \text{ is } \chi\text{-computable} \}$$

- By the representation of one of the closed expressions witnessing the computability of the function.

Quiz

Is the following problem χ-decidable for $A = Bool$? What if $A = \mathbb{N}$?

Let $Fun = \{ f \in A \rightarrow Bool \mid f \text{ is } \chi\text{-computable} \}$.

$$
\begin{aligned}
& pointwise\text{-}equal' \in Fun \times Fun \rightarrow Bool \\
& pointwise\text{-}equal' \ (f, g) = \\
& \quad \mathbf{if} \ \forall \ a \in A . \ f \ a = g \ a \ \mathbf{then} \ \mathsf{true} \ \mathbf{else} \ \mathsf{false}
\end{aligned}
$$

Hint: Use $eval$ or $terminates\text{-}in$.

Pointwise equality of computable functions in $Bool \to Bool$

- The function $pointwise\text{-}equal'$ is decidable.
- Implementation:

$$
\begin{aligned}
& pointwise\text{-}equal' = \lambda\, p.\, \mathbf{case}\ p\ \mathbf{of} \\
& \quad \{\mathsf{Pair}(f, g) \to \\
& \qquad and\ (equal_{Bool}\ (eval\ \mathsf{Apply}(f, \ulcorner \mathsf{True}() \urcorner)) \\
& \qquad\qquad\qquad\qquad\ (eval\ \mathsf{Apply}(g, \ulcorner \mathsf{True}() \urcorner))) \\
& \qquad\qquad\ (equal_{Bool}\ (eval\ \mathsf{Apply}(f, \ulcorner \mathsf{False}() \urcorner)) \\
& \qquad\qquad\qquad\qquad\ (eval\ \mathsf{Apply}(g, \ulcorner \mathsf{False}() \urcorner))) \\
& \quad \}
\end{aligned}
$$

Pointwise equality of computable functions in $Bool \to Bool$

- The function $\mathit{pointwise\text{-}equal}'$ is decidable.
- Implementation:

$$
\begin{array}{l}
\mathit{pointwise\text{-}equal}' = \lambda\, p.\, \mathbf{case}\ p\ \mathbf{of} \\
\quad \{\, \mathsf{Pair}(f, g) \to \\
\qquad \mathit{and}\ (\mathit{equal}_{Bool}\ (\mathit{eval}\ \ulcorner \llcorner f \lrcorner\ \mathsf{True}()\, \urcorner) \\
\qquad\qquad\qquad\qquad\quad (\mathit{eval}\ \ulcorner \llcorner g \lrcorner\ \mathsf{True}()\, \urcorner)) \\
\qquad\qquad\quad (\mathit{equal}_{Bool}\ (\mathit{eval}\ \ulcorner \llcorner f \lrcorner\ \mathsf{False}()\, \urcorner) \\
\qquad\qquad\qquad\qquad\quad (\mathit{eval}\ \ulcorner \llcorner g \lrcorner\ \mathsf{False}()\, \urcorner)) \\
\quad \}
\end{array}
$$

Pointwise equality of computable functions in $\mathbb{N} \rightarrow Bool$

- The function $pointwise\text{-}equal'$ is undecidable.
- The halting problem reduces to it:

$$\begin{aligned}
&halts = \lambda\, p.\, not\ (pointwise\text{-}equal' \\
&\quad \mathsf{Pair}(\ulcorner \lambda\, n.\, terminates\text{-}in\ \mathsf{Pair}(\llcorner\ code\ p\ \lrcorner, n)\urcorner, \\
&\qquad \ulcorner \lambda\, _.\, \mathsf{False}()\urcorner))
\end{aligned}$$

Coding

$\llcorner \overline{} \lrcorner$

One way to give a semantics to $\llcorner \_ \lrcorner$:

- $\llcorner \_ \lrcorner$ is a constructor of a variant of Exp:

$$\frac{e \in \mathit{Exp}}{\llcorner e \lrcorner \in \overline{\mathit{Exp}}} \qquad \frac{e_1 \in \overline{\mathit{Exp}} \qquad e_2 \in \overline{\mathit{Exp}}}{\mathsf{apply}\ e_1\ e_2 \in \overline{\mathit{Exp}}} \qquad \cdots$$

- This variant is the domain of $\ulcorner \_ \urcorner$:

$$\begin{aligned}
\ulcorner _ \urcorner &\in \overline{\mathit{Exp}} \to \mathit{Exp} \\
\ulcorner \llcorner e \lrcorner \urcorner &= e \\
\ulcorner \mathsf{apply}\ e_1\ e_2 \urcorner &= \mathsf{Apply}(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner) \\
&\vdots
\end{aligned}$$

$\llcorner \_ \lrcorner$

- Examples:

$$\begin{aligned} \ulcorner \llcorner f \lrcorner \, \mathsf{True}() \, \urcorner &= \mathsf{Apply}(f, \ulcorner \mathsf{True}() \urcorner) \\ \ulcorner \, eval \llcorner code \;\; e \lrcorner \urcorner &= \mathsf{Apply}(\ulcorner \, eval \, \urcorner, code \;\; e) \end{aligned}$$

- Note that you do not have to use $\llcorner \_ \lrcorner$.

$\llcorner \ \overline{} \ \lrcorner$

The reduction used above:

$$
\begin{aligned}
&\mathit{halts} = \lambda p.\, \mathit{not}\ (\mathit{pointwise\text{-}equal}' \\
&\qquad \mathsf{Pair}(\ulcorner \lambda n.\, \mathit{terminates\text{-}in}\ \mathsf{Pair}(\llcorner\, \mathit{code}\ p\, \lrcorner, n)\, \urcorner, \\
&\qquad\qquad \ulcorner \lambda \_.\, \mathsf{False}()\, \urcorner))
\end{aligned}
$$

Expanded:

$$
\begin{aligned}
&\lambda p.\, \mathit{not}\ (\mathit{pointwise\text{-}equal}' \\
&\qquad \mathsf{Pair}(\mathsf{Lambda}(\ulcorner n \urcorner, \\
&\qquad\qquad\qquad \mathsf{Apply}(\ulcorner \mathit{terminates\text{-}in} \urcorner, \\
&\qquad\qquad\qquad\qquad \mathsf{Const}(\ulcorner \mathsf{Pair} \urcorner, \\
&\qquad\qquad\qquad\qquad\quad \mathsf{Cons}(\mathit{code}\ p, \\
&\qquad\qquad\qquad\qquad\qquad \mathsf{Cons}(\mathsf{Var}(\ulcorner n \urcorner), \mathsf{Nil}())))))), \\
&\qquad\qquad \ulcorner \lambda \_.\, \mathsf{False}()\, \urcorner))
\end{aligned}
$$

Coding

Probably not what you want:

$$\lambda p.\ulcorner eval\ p \urcorner = \lambda p.\,\mathsf{Apply}(\ulcorner eval \urcorner, \mathsf{Var}(\ulcorner p \urcorner))$$

If p corresponds to 0:

$$\lambda p.\,\mathsf{Apply}(\ulcorner eval \urcorner, \mathsf{Var}(\mathsf{Zero}()))$$

A constant function.

Coding

Perhaps more useful:

$$\lambda p. \ulcorner eval \llcorner code\ p \lrcorner \urcorner = \lambda p. \mathsf{Apply}(\ulcorner eval \urcorner, code\ p)$$

For any expression e:

$$(\lambda p. \ulcorner eval \llcorner code\ p \lrcorner \urcorner) \ulcorner e \urcorner \Downarrow \ulcorner eval \ulcorner e \urcorner \urcorner$$

Quiz

What is the result of evaluating $(\lambda\, p.\, eval \ulcorner eval \llcorner code\ p \lrcorner \urcorner) \ulcorner \text{Zero}() \urcorner$?

- Nothing
- $\text{Zero}()$
- $\ulcorner \text{Zero}() \urcorner$
- $\ulcorner \ulcorner \text{Zero}() \urcorner \urcorner$
- $\ulcorner \ulcorner \ulcorner \text{Zero}() \urcorner \urcorner \urcorner$
- $\ulcorner \ulcorner \ulcorner \ulcorner \text{Zero}() \urcorner \urcorner \urcorner \urcorner$

Types

- The language χ is untyped.
- However, it may be instructive to see certain programs as typed.

Types

- $Rep\ A$: Representations of programs of type A.
- Some examples:

$$
\begin{array}{ll}
\mathsf{True}() & : Bool \\
\ulcorner \mathsf{True}() \urcorner & : Rep\ Bool \\
\ulcorner \mathsf{true} \urcorner & : Bool \\
\lambda f\,.\,\lambda x\,.\,f\ x & : (A \to B) \to A \to B \\
\lambda f\,.\,\lambda x\,.\,\mathsf{Apply}(f, x) & : Rep\ (A \to B) \to \\
 & \ Rep\ A \to Rep\ B \\
eval & : Rep\ A \to Rep\ A \\
code & : Rep\ A \to Rep\ (Rep\ A) \\
terminates\text{-}in & : Rep\ A \times \mathbb{N} \to Bool \\
\ulcorner terminates\text{-}in \urcorner & : Rep\ (Rep\ A \times \mathbb{N} \to Bool)
\end{array}
$$

Types

The reduction used above:

$$
\begin{aligned}
& halts = \lambda\, p.\, not\ (pointwise\text{-}equal' \\
& \quad \mathsf{Pair}(\ulcorner \lambda\, n.\, terminates\text{-}in\ \mathsf{Pair}(\llcorner\, code\ p\, \lrcorner, n) \urcorner, \\
& \qquad \ulcorner \lambda\, \_.\, \mathsf{False}() \urcorner))
\end{aligned}
$$

If

$$
\begin{aligned}
& pointwise\text{-}equal' : \\
& \quad Rep\ (\mathbb{N} \to Bool) \times Rep\ (\mathbb{N} \to Bool) \to Bool
\end{aligned}
$$

then

$$halts : Rep\ A \to Bool.$$

More undecidable problems

Quiz

Is the following function χ-computable?

$$
\begin{array}{l}
optimise \in CExp \to CExp \\
optimise\ e = \\
\quad \text{some optimally small expression with} \\
\quad \text{the same semantics as } e
\end{array}
$$

Size: The number of constructors in the abstract syntax (Exp, Br, $List$, not Var or $Const$).

Full employment theorem for compiler writers

- An optimally small non-terminating expression is equal to $\mathbf{rec}\ x = x$ (for some x).
- The halting problem reduces to this one:

$$
\begin{array}{l}
\mathit{halts} = \lambda\, p.\, \mathbf{case}\ \mathit{optimise}\ p\ \mathbf{of} \\
\quad \{\, \mathsf{Rec}(x, e) \to \mathbf{case}\ e\ \mathbf{of} \\
\qquad \{\, \mathsf{Var}(y) \quad\ \to \mathsf{False}() \\
\qquad ;\ \mathsf{Rec}(x, e) \to \mathsf{True}() \\
\qquad ;\ \ldots \\
\qquad \} \\
\quad ;\ \ldots \\
\quad \}
\end{array}
$$

Computable real numbers

- Computable reals can be defined in many ways.
- One example, using signed digits:

$$\begin{aligned}
&\mathit{Interval} = \\
&\quad \{f \in \mathbb{N} \to \{-1, 0, 1\} \mid f \text{ is } \chi\text{-computable}\}
\end{aligned}$$

$$\begin{aligned}
&[\![_]\!] \in \mathit{Interval} \to [-1, 1] \\
&[\![f]\!] = \textstyle\sum_{i=0}^{\infty} f\ i \cdot 2^{-i-1}
\end{aligned}$$

- Why signed digits? Try computing the first digit of $0.00000... + 0.11111...$ (in binary notation).

Is a computable real number equal to zero?

- Is a computable real number equal to zero?

$$\begin{array}{l} \textit{is-zero} \in \textit{Interval} \rightarrow \textit{Bool} \\ \textit{is-zero}\ x = \textbf{if}\ [\![x]\!] = 0\ \textbf{then}\ \mathsf{true}\ \textbf{else}\ \mathsf{false} \end{array}$$

- The halting problem reduces to this one:

$$\begin{array}{l} \textit{halts} = \lambda\, p.\ \textit{not}\ (\textit{is-zero}\ \ulcorner\, \lambda\, n. \\ \quad \textbf{case}\ \textit{terminates-in}\ \mathsf{Pair}(\llcorner\ \textit{code}\ p\ \lrcorner, n)\ \textbf{of} \\ \quad\quad \{\, \mathsf{True}()\ \rightarrow \mathsf{One}() \\ \quad\quad ;\ \mathsf{False}() \rightarrow \mathsf{Zero}() \\ \quad\quad \}\urcorner) \end{array}$$

Undecidable problems

- A list on Wikipedia.
- A list on MathOverflow.

Summary

- X-computability.
- A self-interpreter for χ.
- Reductions.
- More problems that are or are not computable.
- More about coding.