Starter Questions

Feel free to discuss these with your neighbour:

- What do you call a function in $\mathbb{N} \rightarrow \mathbb{B}$?
- Are there uncomputable functions in $\mathbb{N} \rightarrow \mathbb{B}$?
- What is the difference between $\exists x \forall y : P(x, y)$ and $\forall x \exists y : P(x, y)$?
- Are there well-defined fragments of the \texttt{while} language for which we can always determine whether programs in that fragment will terminate?

Reminder: Next Monday is a Bank Holiday

Reminder: Course Unit Surveys are open
Lecture 8
Decidability and the Halting Problem
COMP11212

Giles Reger

April 2017
Using a while program

Universality

Coding

\( f : \mathbb{N} \rightarrow \mathbb{B} \)

(Un)Decidability

Existence

Halting Problem

\( f : \mathbb{N} \rightarrow \mathbb{N} \)

\( f : \mathbb{N}^m \rightarrow \mathbb{N}^n \)

computable
uncomputable
Recap

- Computable functions are those that can be computed by a program
- Uncomputable functions are the ones that are not computable
- We can count while programs
- We cannot count functions
- There are functions without programs i.e. uncomputable ones
- We can write a universal program that can simulate all programs
- All sensible models of computation are equivalent
Predicates

We consider a special kind of function: predicates in $\mathbb{N} \to \mathbb{B}$

(or more generally any boolean function i.e. in $A \to \mathbb{B}$ for any $A$)

These represent 'yes' or 'no' questions and cover a large range of interesting problems
We say that a function $P : \mathbb{N} \rightarrow \mathbb{B}$ is *computable*, if, and only if, the function $f : \mathbb{N} \rightarrow \mathbb{N}$ is computable, where

$$P(x) = \begin{cases} 
  \text{True} & \text{if } f(x) = 1 \\
  \text{False} & \text{if } f(x) = 0
\end{cases}$$

Functions which return boolean values are called *predicates*. 

Giles Reger
Lecture 8
April 2017
Decidability

Definition

The predicate $P$ is **decidable** if, and only if, there is a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that:

$$f(x) = \begin{cases} 1 & \text{if } P(x) \text{ holds} \\ 0 & \text{if } P(x) \text{ doesn’t hold} \end{cases}$$

- The total function $f$ is called the **characteristic function** of $P$
- The associated program in `while` is a **decision procedure** for $P$.
- Any predicate which is *not* decidable is **undecidable**.
Decidability

Definition

The partial function $P$ is partially decidable if, and only if, there exists a computable partial function $f : \mathbb{N} \rightarrow \mathbb{N}$ with

$$f(x) = \begin{cases} 1 & \text{if } P(x) \text{ holds} \\ \text{undefined} & \text{if } P(x) \text{ doesn’t hold} \end{cases}$$

- The partial function $f$ is called the partial characteristic function of $P$.
- The associated program in while is a partial decision procedure for $P$.
- All decidable functions are partially decidable.

Partial decidability (or semi-decidability) means that it always terminates if the answer is ‘yes’ but not necessarily if the answer is ‘no’.
There exist undecidable predicates

As before, this follows from two things:

1. There are a countable number of decision procedures
2. There are an uncountable number of functions of type $\mathbb{N} \rightarrow \mathbb{B}$

(1) is true as decision procedures form a subset of the countable programs

The proof of (2) is left as an exercise (for the Example’s class).
Combining decidable and partially decidable predicates

We can build decidable predicates from decidable predicates, in exactly the same was as for computable functions.

If $P$ and $Q$ are decidable predicates then so are $\neg P$, $P \land Q$, $P \lor Q$, $P \rightarrow Q$ etc. We can show this by combining the decision procedures for $P$ and $Q$.

However, we cannot necessarily do the same for partially decidable predicates (*Exercise*).

**Hint for the exercise:** if $P$ is decidable and $Q$ is partially decidable then let $S_P$ and $S_Q$ be the related decision and partial decision procedures. Now $P \lor Q$ is partially decidable as we can write the partial decision procedure

$$y := x; \ S_P; \text{if } x = 1 \text{ then } x := y; \ S_Q$$

which relies on the fact that $S_P$ will always terminate.
There exist uncomputable binary total functions $f : (\mathbb{N} \times \mathbb{N}) \to \mathbb{N}$. Such a function can be tabulated as

\[
\begin{array}{cccccc}
  f & 0 & 1 & 2 & 3 & \ldots \\
  \hline
  0 & f(0,0) & f(0,1) & f(0,2) & f(0,3) & \ldots \\
  1 & f(0,1) & f(1,1) & f(1,2) & f(1,3) & \ldots \\
  2 & f(2,0) & f(2,1) & f(2,2) & f(2,3) & \ldots \\
  3 & f(3,0) & f(3,1) & f(3,2) & f(3,3) & \ldots \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{array}
\]

Let $g : \mathbb{N} \to \mathbb{B}$ be $g(i) \begin{cases} True & \text{if } f(i, i) = 0 \\ False & \text{otherwise} \end{cases}$

As there are undecidable predicates in $\mathbb{N} \to \mathbb{B}$ there must be uncomputable functions in $(\mathbb{N} \times \mathbb{N}) \to \mathbb{N}$, otherwise we could use $f$ to decide $g$. 
If total function $f$ is computable then the following partial function $g$ is computable

$$g(i) \begin{cases} 
0 & \text{if } f(i, i) = 0 \\
\text{undefined} & \text{otherwise}
\end{cases}$$

Let $F$ be the program that computes $f$. Define the program $G$ as

$$y := x; F; \text{ if } x \neq 0 \text{ then while true do skip}$$

If we can show that $g$ is not computable. Then this means that $f$ is not computable.
"This sentence is false."

The paradox comes from mixing the feature being described ("this sentence") with a description of its properties ("is false").

We will show how doing something similar with functions (forcing them to say something about themselves) can also lead to a paradox, which shows that we cannot do something we would like to.
The Halting Problem

Given a program \( P \) and an input \( n \) does \( P \) halt on \( s = [x \mapsto n] \)?

As usual, we state this for unary programs but it can be lifted.

Equivalent to asking, is there a predicate \( \text{halt}(f, n) \) that returns true if \( \psi_U(f, n) \) halts and false otherwise?

Note that \( \text{halt} \) has to work for arbitrary computable functions.
The concise proof

Assume the halting program HALT for the function halt

Let $G$ be the program

$$y := x; \text{HALT}; \text{if } x \neq 0 \text{ then while true do skip}$$

for the partial function

$$g(i) \begin{cases} 
0 & \text{if } \text{halt}(i, i) = 0 \\
\text{undefined} & \text{otherwise}
\end{cases}$$

Now what is the result of $\text{halt}(\gamma(G), \gamma(G))$?

- If it is 0 then $g(\gamma(G)) = 0$ by the definition of $g$ and therefore $G$ terminates on $\gamma(G)$.
- If it is 1 then $g(\gamma(G))$ is undefined by the definition of $g$ and $G$ does not terminate on $\gamma(G)$.

Therefore, $g$ cannot be computable, hence halt is uncomputable.
The concise proof

Assume the halting program HALT for the function halt

Let $G$ be the program

\[
y := x; \text{ HALT; } \text{ if } x \neq 0 \text{ then while true do skip}
\]

for the partial function

\[
g(i) \begin{cases} 
0 & \text{if } \text{halt}(i, i) = 0 \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

Now what is the result of \(\text{halt}(\gamma(G), \gamma(G))\)?

- If it is 0 then \(g(\gamma(G)) = 0\) by the definition of \(g\) and therefore \(G\) terminates on \(\gamma(G)\) \(\text{CONTRADICTION}\)
- If it is 1 then \(g(\gamma(G))\) is undefined by the definition of \(g\) and \(G\) does not terminate on \(\gamma(G)\)

Therefore, \(g\) cannot be computable, hence \(\text{halt}\) is uncomputable.
The concise proof

Assume the halting program HALT for the function halt

Let $G$ be the program

\[
y := x; \quad \text{HALT}; \quad \text{if } x \neq 0 \text{ then while true do skip}
\]

for the partial function

\[
g(i) \begin{cases} 
0 & \text{if } \text{halt}(i, i) = 0 \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

Now what is the result of $\text{halt}(\gamma(G), \gamma(G))$?

- If it is 0 then $g(\gamma(G)) = 0$ by the definition of $g$ and therefore $G$ terminates on $\gamma(G)$ \textbf{CONTRADICTION}

- If it is 1 then $g(\gamma(G))$ is undefined by the definition of $g$ and $G$ does not terminate on $\gamma(G)$ \textbf{CONTRADICTION}

Therefore, $g$ cannot be computable, hence halt is uncomputable.
The above argument is implicitly applying diagonalisation. Let $g$ be

$$g(i) \begin{cases} 
    \text{True} & \text{if } \psi_U(i, i) \text{ halts} \\
    \text{False} & \text{otherwise}
\end{cases}$$

If $g$ is computable then it must exist in the enumeration of inputs to $\psi_U$. However, this forces us to put something in this box and by construction of $g$ we cannot, therefore $\eta_G$ is not in the enumeration, and $g$ is not computable, hence neither is halt.

$$
\begin{array}{c|cccc}
\psi_U & 0 & 1 & \ldots & \gamma(G) & \ldots \\
\hline
\eta_0 & \eta_0(0) & \eta_0(1) & \ldots & \eta_0(\gamma(G)) & \ldots \\
\eta_1 & \eta_1(0) & \eta_1(1) & \ldots & \psi_1(\gamma(G)) & \ldots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\eta_G & \eta_G(0) & \eta_G(1) & \ldots & \eta_G(\gamma(G)) & \ldots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\end{array}
$$
The Halting Problem is Partially Decidable

Let

\[ \text{halt}(f, a) = \begin{cases} 
\text{True} & \text{if } \psi_U(f, a) \text{ halts} \\
\text{False} & \text{otherwise}
\end{cases} \]

Let \( U \) be the universal program. A partial decision procedure is

\[ U; \ x := 1; \]

This will always return the correct answer if the answer is \( \text{True} \), which is all we require of a partial decision procedure.
As a direct result of the Halting Problem, the following problems are also undecidable:

- Whether a given program terminates on the input 5
- Whether a given program terminates on all inputs
- Whether a given program terminates on any inputs
- The maximum number of steps a program will take before it produces some answer
Theorem (Rice’s Theorem)

Let $P_1$ be the set of all (partial) unary computable functions, as previously defined. For every non-empty $F \subseteq P_1$, the predicate $P_f : \mathbb{N} \to \mathbb{B}$ defined as

$$P_f(i) = \begin{cases} 
\text{true} & \text{if } \eta_i \in F \\
\text{false} & \text{if } \eta_i \notin F
\end{cases}$$

is undecidable.

Proof by reduction to the Halting Problem. As $F$ is non-empty let $f \in F$, then define the function $h$ as

$$h(i, j, k) = \begin{cases} 
f(k) & \text{if } \psi_U(i, j) \text{ halts} \\
\text{undefined} & \text{otherwise}
\end{cases}$$

Now we can use $P_f(\gamma(h))$ to decide the halting problem.
Are these statements the same?

1. There exists a program $P$ that decides whether every program $Q$ halts.
2. For all programs $Q$ there exists a program $P$ that decides whether $Q$ halts.

No. The first is about whether there is a general method of showing termination. The second is about whether given a specific program there is a program that reports the correct result. Trivially this is the case, as there is always a correct result.

The existence of total correctness proofs should tell us that termination checking is not completely a dead end.
Recalling Definitions

- Computable functions
- Uncomputable functions
- Decidability
- Partial-decidability
- The Gödel number of a program
- The Universal Program
- The Halting Problem
Reflective Questions

- What do all of these things look like when defined with Turing Machines? Hint: see either of the recommended texts. In fact, almost any definitions online will use Turing Machines.

- Can a partial decision procedure ever be useful?

- What if we had non-determinism or parallelism in the language, does that change anything? What about quantum computation?

- What are primitive recursive functions, where do they fit in?

- What has any of this got to do with Gödel’s incompleteness theorem or the weird German word “Entscheidungsproblem”? 