CSCI 334:
Principles of Programming Languages

Lecture 8: Computability

Instructor: Dan Barowy

Williams

Announcements

- Lab 5 is a solo lab.
- Scheduled power outage: this Sunday at 10pm until Monday at 9am
  - **All CS lab machines**
  - All CS servers
- Wash hands, cough into sleeve, don't touch your face. **Stay home if you are not feeling well.**

Announcements

- No colloquium this week :(

Midterm Exam

- Friday, March 20, in class.

Outline

1. Quiz
2. Total and partial functions
3. Halting problem
4. Reduction-style proofs

Quiz

Computability

Computability

i.e., what can and cannot
be done with a computer

A function *f* is **computable** if there
is a program *P* that computes *f*.

In other words, for **any** (valid) input *x*, the
computation *P(x)* **halts** with output *f(x)*.

Computability

example

valid inputs are **integers**

P(x) is:
$f(x) = x + 5$

computable?
yes.

Computability

example

valid inputs are **integers**

P(x) is:
$f(x) = 5/x$

computable?
yes, *partially*.

Total Function

$f: A \to B$ is a subset $f \subseteq A \times B$ subject to

1. for every $a \in A$, there is a $b \in B$ with $\langle a,b \rangle \in f$ **totality**

2. if $\langle a,b \rangle \in f$ and $\langle a,c \rangle \in f$ then $b = c$ **single valued**

e.g,
$f(x) = x + 5$

Partial Function

$f: A \to B$ is a subset $f \subseteq A \times B$ subject to

1. for every $a \in A$, there is a $b \in B$ with $\langle a,b \rangle \in f$ **totality**

2. if $\langle a,b \rangle \in f$ and $\langle a,c \rangle \in f$ then $b = c$ **single valued**

e.g,
$f(x) = 5/x$

The “graph” of a function

$f(x) = x + 5$

$\{\langle x, x+5\rangle \mid x \in \mathbb{Z}\}$

{<x, x+5> | x is an integer}

The “graph” of a function

$f(x) = 5/x$

$\{\langle x, 5/x\rangle \mid x \in \mathbb{Z} \wedge x \neq 0\}$

Undefinedness

$x/0$

Activity

The Halting Problem

Decide whether program P halts on input x.

Given program P and input x,

$$\texttt{Halt(P,x)} = \begin{cases} \text{returns } \texttt{true} \text{ if } \texttt{P(x)} \text{ halts} \\ \text{returns } \texttt{false} \text{ otherwise} \end{cases}$$

How might this work?

Clarifications:

`P(x)` is the output of program `P` run on input `x`.

The type of `x` does not matter; assume `string`.

The Halting Problem

Decide whether program P halts on input x.

Given program P and input x,

$$\texttt{Halt(P,x)} = \begin{cases} \text{returns } \texttt{true} \text{ if } \texttt{P(x)} \text{ halts} \\ \text{returns } \texttt{false} \text{ otherwise} \end{cases}$$

How might this work?

Fact: it is provably impossible to write `Halt`

Notes on the proof

We utilize two key ideas:

- Function evaluation by substitution
- Reductio ad absurdum (proof form)

Notes on the proof

The *form* of the proof is ***reductio ad absurdum***.

Literally: "reduction to absurdity".

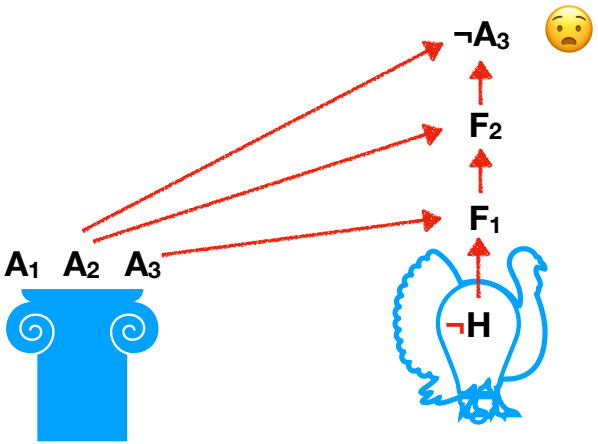
Start with **axioms** and **presuppose the outcome** we want to show.

Then, following strict rules of logic, **derive new facts**.

Finally, derive a fact that **contradicts** another fact.

Conclusion: the **presupposition must be false**.

Reductio ad Absurdum



Function Evaluation by Substitution

```
def addone(x):
    return x + 1
```

`addone(1)`	`λx.(+ x 1)1`
`[1/x]x + 1`	`[1/x](+ x 1)`
`1 + 1`	`(+ 1 1)`
`2`	`2`

The Halting Problem

Notes on the proof:

The proof relies on the kind of **substitution** that we've been using to "compute" functions in the lambda calculus.

Remember: **we are looking to produce a contradiction**.

The proof is hard to "understand" because the facts it derives **don't actually make sense**. Don't read too deeply.

The Halting Problem: Proof

Suppose:

`Halt(P,x)` = { returns `true` if `P(x)` halts; returns `false` otherwise } — `Halt` always halts!

Construct:

`DNH` does not always halt!

`DNH(P)` = { if `Halt(P,P)` is `true`, `while(1){}`; returns `false` otherwise }

The Halting Problem: Proof

Observations so far:

`DNH(P)` will run forever if `Halt(P,P)` is `true`.
`DNH(P)` will halt if `Halt(P,P)` is `false`.

Rewrite:

`DNH(P)` = { if `Halt(P,P)` is `true`, `while(1){}`
returns `false` otherwise

The Halting Problem: Proof

Observations so far:

`DNH(P)` will run forever if `Halt(P,P)` is `true`.
`DNH(P)` will halt if `Halt(P,P)` is `false`.

Rewrite:

`DNH(P)` = { if `P(P)` halts, run forever
returns `false` otherwise

The Halting Problem: Proof

Observations so far:

`DNH(P)` will run forever if `Halt(P,P)` is `true`.
`DNH(P)` will halt if `Halt(P,P)` is `false`.

Rewrite:

`DNH(P)` = { if `P(P)` halts, run forever
halt

The Halting Problem: Proof

Observations so far:

`DNH(P)` will run forever if `P(P)` halts.
`DNH(P)` will halt if `P(P)` runs forever.

Rewrite:

`DNH(P)` = { if `P(P)` halts, run forever
halt

The Halting Problem

Isn't `DNH` itself a program?

What happens if we call `DNH(DNH)`?

`P = DNH`

`DNH( P )` will run forever if `P( P )` halts.
`DNH( P )` will halt if `P( P )` runs forever.

The Halting Problem

Isn't `DNH` itself a program?

What happens if we call `DNH(DNH)`?

`P = DNH`

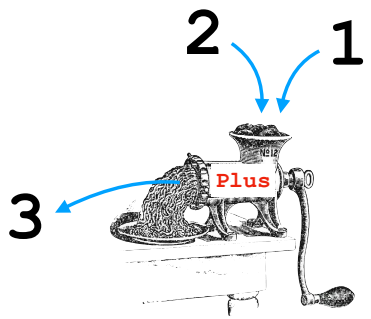
`DNH(DNH)` will run forever if `DNH(DNH)` halts.
`DNH(DNH)` will halt if `DNH(DNH)` runs forever.

This literally makes no sense. **Contradiction**!

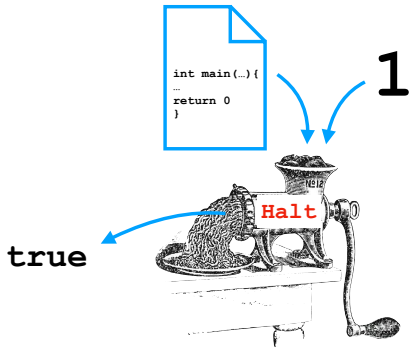
What was our one assumption? `Halt` **exists**.

Therefore, the `Halt` function **cannot exist**.

Reductions

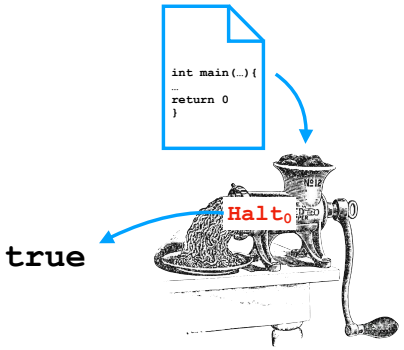


Reductions



We know that `Halt` is impossible to compute.

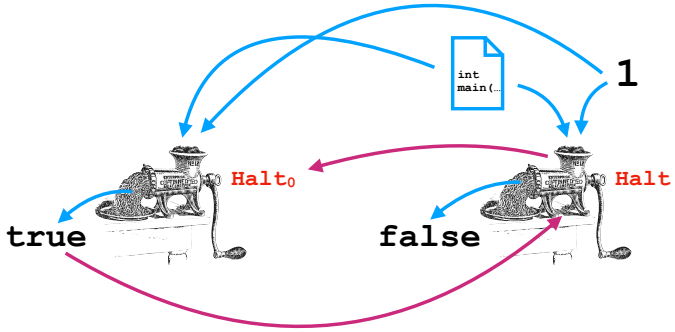
Reductions



A function `f(i)` **halts not** if and only if `f` **does not halt** on input `i`.

Is $\texttt{Halt}_0$ computable?

Reductions

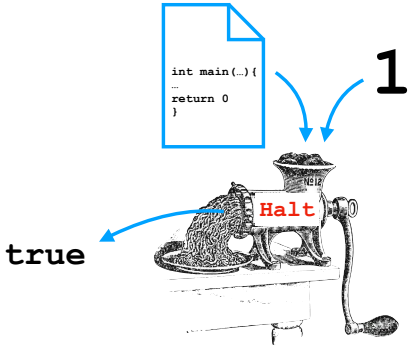


Assume that $\texttt{Halt}_0$ is computable.

(e.g., it's in your standard library)

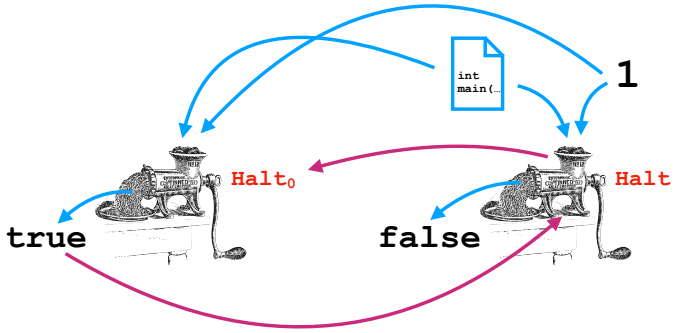
Construct `Halt` using $\texttt{Halt}_0$.

Reductions



We know that `Halt` is impossible to compute.

Reductions



If we can do it, what does this mean for $\texttt{Halt}_0$?

$\texttt{Halt}_0$ is **not computable**.

Reductions

(on board)

Reductions

We can use the Halting Problem to show that other problems cannot be solved by "reduction" to the Halting Problem.

We cannot tell, in general…

… if a program will **run forever**.

… if a program will **eventually produce an error**.

… if a program **is done using a variable**.

Generality

```
def myprog(x):
  return 0

def Halt(P,x):
  if(P = "def myprog(x):\n\treturn 0"):
    return true
  else
    return false
```

Recap & Next Class

Today we covered:

Total and partial functions

Halting problem

Reductions

Next class:

More computability