CSCI 334:
Principles of Programming Languages

Lecture 8: PL Fundamentals IV

Instructor: Dan Barowy

Williams

Announcements

No class on Tuesday (reading period)

Midterm exam on Thursday

Midterm review Q&A:
Here, Tuesday, normal time
(optional!)

Will post study guide shortly after class

Another way to find redexes

`(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`

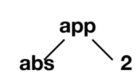
Normal order: `(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`

Applicative order: `(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`

Neither: `(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`

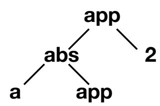
Another way to find redexes

`(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`



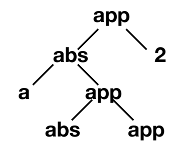
Another way to find redexes

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



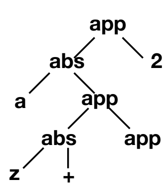
Another way to find redexes

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



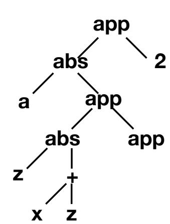
Another way to find redexes

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



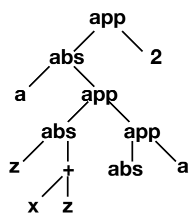
Another way to find redexes

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



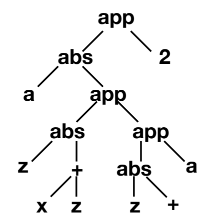
Another way to find redexes

`(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`



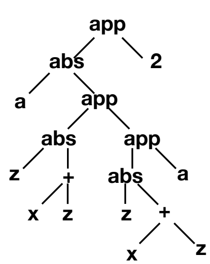
Another way to find redexes

`(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`



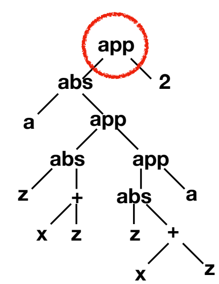
Another way to find redexes

`(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`



Another way to find redexes

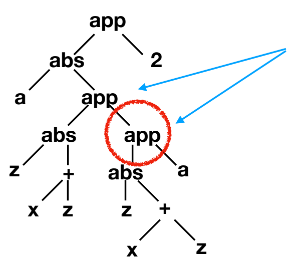
`(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`



Normal order: “leftmost outermost” application

Another way to find redexes

`(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`



Q: Wait!
Isn't there a "more left" app?

A: Yes, but it is not the *innermost*. "Leftmost" is used to break ties among multiple innermost apps.

Applicative order: "leftmost innermost" application

The Halting Problem

The Halting Problem

Decide whether program P halts on input x.

Given program P and input x,

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

How might this work?

<u>Clarifications:</u>

`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,

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

How might this work?

Fact: it is provably impossible to write `Halt`

Notes on the proof

We utilize two key ideas:

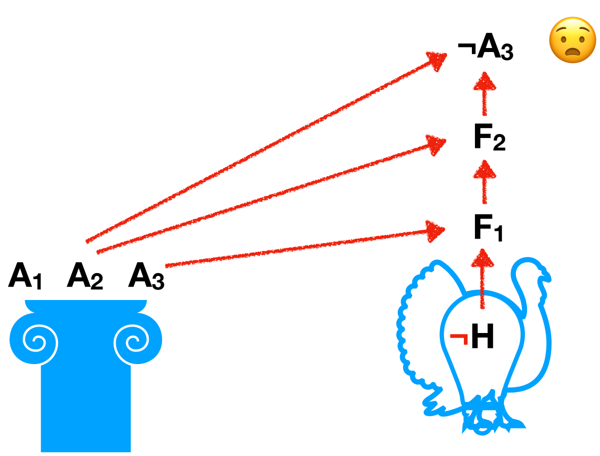
- Function evaluation by substitution
- Reductio ad absurdum (proof form)

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

Reductio ad Absurdum



The Halting Problem: Proof

Suppose:

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

Construct:

`DNH(P)` = { if `Halt(P,P)` is `true`, `while(1){}`; returns `false` otherwise } — `DNH` does not always halt!

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:

$$\texttt{DNH(P)} = \begin{cases} \text{if } \texttt{Halt(P,P)} \text{ is } \texttt{true}, \texttt{while(1)\{\}} \\ \text{returns } \texttt{false} \text{ otherwise} \end{cases}$$

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:

$$\texttt{DNH(P)} = \begin{cases} \text{if } \texttt{P(P)} \text{ halts, run forever} \\ \text{returns } \texttt{false} \text{ otherwise} \end{cases}$$

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:

$$\texttt{DNH(P)} = \begin{cases} \text{if } \texttt{P(P)} \text{ halts, run forever} \\ \text{halt} \end{cases}$$

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:

$$\texttt{DNH(P)} = \begin{cases} \text{if } \texttt{P(P)} \text{ halts, run forever} \\ \text{halt} \end{cases}$$

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.

Generality

```
def myprog(x):
  return 0
```

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

Reductions

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

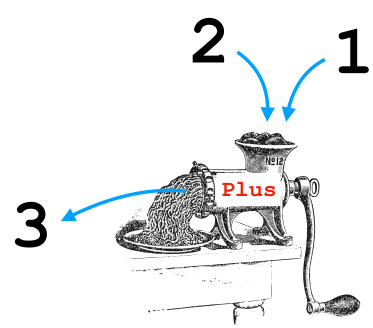
We cannot tell, in general…

… if a program will run forever.

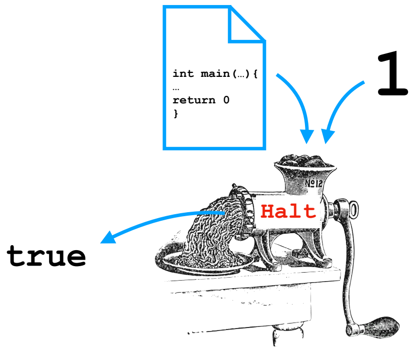
… if a program will eventually produce an error.

… if a program is done using a variable.

Reductions

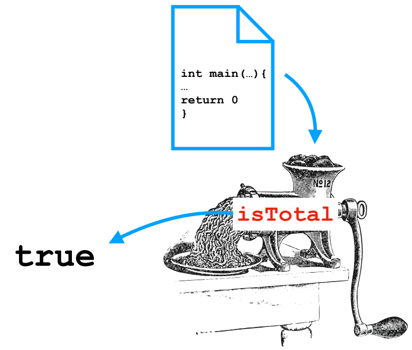


Reductions



We know that `Halt` is impossible to compute.

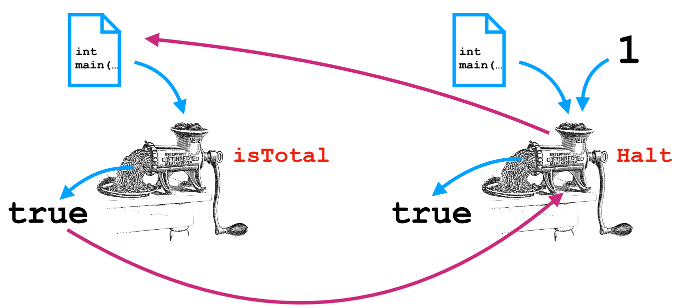
Reductions



A function `f(i)` is total if and only if `f` is defined for every input `i`.

Is `isTotal` computable?

Reductions

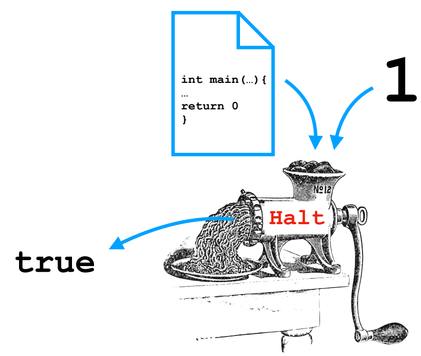


Assume that `isTotal` is computable.

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

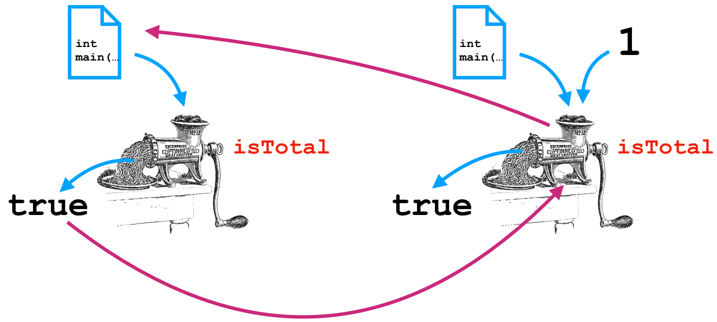
Construct `Halt` using `isTotal`.

Reductions



We know that `Halt` is impossible to compute.

Reductions



If we can do it, what does this mean for `isTotal`?

`isTotal` is not computable.

Reductions

```
def Halt(p,i):


    return Total(       p        )
```

Why doesn't this version do what we want?

Because `Total` tells us whether `p` halts on all inputs.

`Halt` asks specifically about input `i`.

Reductions

```
def Halt(p,i):
    def Runs(j):
        p(i)
        return "done!"
    return Total( getsource(Runs) )
```

This is a valid program.

So we can solve the Halting Problem!

Therefore, `Total` is not computable. 😞

Mini-Midterm Review

What have we learned?

The C programming language

- Basic syntax
- Program evaluation using a call stack
- Memory as an explicitly-managed resource

Grammars and parsing

- Backus-Naur form
- Prec. and assoc. rules for ambiguous grammars
- Derivation and abstract syntax trees

The lambda calculus

- Syntax
- Program evaluating using reduction rules

Mini-Midterm Review

Not on the exam:

- Halting problem
- Reductions