CMSC 430, Feb 13th 2020

Dupe

First things first

First things first

- Consider the following Racket code

First things first

- Consider the following Racket code

```
x
```

First things first

- Consider the following Racket code

x

- Is **x** `free'?

First things first

- Consider the following Racket code

```
(cons x y)
```

First things first

- Consider the following Racket code

```
(cons x y)
```

- Is **x** `free'?

First things first

- Consider the following Racket code

```
(cons x y)
```

- Is **x** `free'?
- Is **y** `free'?

First things first

- Consider the following Racket code

```
(lambda (x) (cons x y))
```

First things first

- Consider the following Racket code

```
(lambda (x) (cons x y))
```

- Is **x** `free'?

First things first

- Consider the following Racket code

```
(lambda (x) (cons x y))
```

- Is **x** `free'?
- Is **y** `free'?

First things first

- Consider the following Racket code

```
(let ((y 5))
     (lambda (x) (cons x y)))
```

First things first

- Consider the following Racket code

```
(let ((y 5))
     (lambda (x) (cons x y)))
```

- Is **x** `free'?

First things first

- Consider the following Racket code

```
(let ((y 5))
     (lambda (x) (cons x y)))
```

- Is **x** `free'?
- Is **y** `free'?

Our languages so far:

Our languages so far:

- We can branch based on computed values, but it's a bit clunky
  - We'd like to a) understand the clunkiness, and b) fix it

Language du jour

Language du jour

- Today, we will double the number of types we can deal with!

Language du jour

- Today, we will double the number of types we can deal with!
  - Right now, we've only got integers

Language du jour

- Today, we will double the number of types we can deal with!
  - Right now, we've only got integers
  - By the end of today we'll have integers *and* booleans

Dupe

Dupe

- Our language **`Dupe`** is going to modify *and* extend **`con`**

Con’s AST

Con's AST

- Let's review

Con’s AST

- Let’s review
  - **`e ::= i | add1 e | sub1 e | if (zero? e) e e`**

Con's AST

- Let's review
  - **`e ::= i | add1 e | sub1 e | if (zero? e) e e`**
- This is clunky

Con's AST

- Let's review
  - **`e ::= i | add1 e | sub1 e | if (zero? e) e e`**
- This is clunky
  - **`if`** is `hard coded' to dispatch based on **`zero?`** and can do nothing else

Con's AST

- Let's review
  - **`e ::= i | add1 e | sub1 e | if (zero? e) e e`**
- This is clunky
  - **`if`** is `hard coded' to dispatch based on **`zero?`** and can do nothing else
- Let's make **`if`** be more like what we experience in other languages

Con's AST

- Let's review
  - **`e ::= i | add1 e | sub1 e | if (zero? e) e e`**
- This is clunky
  - **`if`** is `hard coded' to dispatch based on **`zero?`** and can do nothing else
- Let's make **`if`** be more like what we experience in other languages
  - It should dispatch on arbitrary boolean expressions!

Dupe's AST

Dupe's AST

- Some changes:

Dupe's AST

- Some changes:
  - **`e ::= ... | if e e e | zero? e`**

Dupe's AST

- Some changes:
  - **`e ::= ... | if e e e | zero? e`**
- This is less clunky

Dupe's AST

- Some changes:
  - **`e ::= ... | if e e e | zero? e`**
- This is less clunky
  - **`if`** is no longer `hard coded' to dispatch based on **`zero?`**

Dupe's AST

- Some changes:
  - **`e ::= ... | if e e e | zero? e`**
- This is less clunky
  - **`if`** is no longer `hard coded' to dispatch based on **`zero?`**
- **`if`** is now more like what we experience in other languages

Dupe's AST

- Some changes:
  - **`e ::= ... | if e e e | zero? e`**
- This is less clunky
  - **`if`** is no longer `hard coded' to dispatch based on **`zero?`**
- **`if`** is now more like what we experience in other languages
  - Thing to think about:

Dupe's AST

- Some changes:
  - **`e ::= ... | if e e e | zero? e`**
- This is less clunky
  - **`if`** is no longer `hard coded' to dispatch based on **`zero?`**
- **`if`** is now more like what we experience in other languages
  - Thing to think about:
  - Why do we still need **`zero?`** (if at all)

Valley Date

- Syntax validation for **`Dupe`** is just what you might expect

Valley Date

- Syntax validation for **Dupe** is just what you might expect

```
(define (expr? x)
  (match x
    [(? integer?) #t]
    [(? boolean?) #t]
    [`(add1 ,x) (expr? x)]
    [`(sub1 ,x) (expr? x)]
    [`(zero? ,x) (expr? x)]
    [`(if ,x ,y ,z)
     (and (expr? x)
          (expr? y)
          (expr? z))]
    [_ #f]))
```

Some Ant, ick!

Some Ant, ick!

- The meaning of integers is subsumed by a meaning for *values*

Some Ant, ick!

- The meaning of integers is subsumed by a meaning for *values*

$$\overline{\boldsymbol{D}[\![v,\ v]\!]}$$

Some Ant, ick!

Some Ant, ick!

- The meaning of **`add1/sub1`** is unchanged since **`blackmail`**

Some Ant, ick!

- The meaning of **`add1/sub1`** is unchanged since **`blackmail`**

$$\frac{\boldsymbol{D}[\![e_0, i_0]\!] \quad i_1 = i_0 + 1}{\boldsymbol{D}[\![(\text{add1 } e_0), i_1]\!]}$$

Some Ant, ick!

- The meaning of **`add1/sub1`** is unchanged since **`blackmail`**

$$\frac{\boldsymbol{D}[\![e_0, i_0]\!] \quad i_1 = i_0 + 1}{\boldsymbol{D}[\![(\text{add1 } e_0), i_1]\!]}$$

$$\frac{\boldsymbol{D}[\![e_0, i_0]\!] \quad i_1 = i_0 - 1}{\boldsymbol{D}[\![(\text{sub1 } e_0), i_1]\!]}$$

Some Ant, ick!

Some Ant, ick!

- The meaning of **`if`** has changed a bit

Some Ant, ick!

- The meaning of **`if`** has changed a bit

$$\frac{\boldsymbol{D}[\![e_0,\ v_0]\!] \quad \text{is-true}[\![v_0]\!] \quad \boldsymbol{D}[\![e_1,\ v_1]\!]}{\boldsymbol{D}[\![(\text{if } e_0\ e_1\ e_2),\ v_1]\!]}$$

Some Ant, ick!

- The meaning of **`if`** has changed a bit

$$\frac{\boldsymbol{D}[\![e_0,\ v_0]\!] \quad \text{is-true}[\![v_0]\!] \quad \boldsymbol{D}[\![e_1,\ v_1]\!]}{\boldsymbol{D}[\![(\text{if } e_0\ e_1\ e_2),\ v_1]\!]}$$

$$\frac{\boldsymbol{D}[\![e_0,\ v_0]\!] \quad \text{is-false}[\![v_0]\!] \quad \boldsymbol{D}[\![e_2,\ v_2]\!]}{\boldsymbol{D}[\![(\text{if } e_0\ e_1\ e_2),\ v_2]\!]}$$

Some Ant, ick!

Some Ant, ick!

- Now we need a separate meaning for **`zero?`**

Some Ant, ick!

- Now we need a separate meaning for **`zero?`**

$$\frac{\boldsymbol{D}[\![e_0,\, i]\!] \quad i = 0}{\boldsymbol{D}[\![(\text{zero?}\ e_0),\, \#\text{t}]\!]}$$

Some Ant, ick!

- Now we need a separate meaning for **`zero?`**

$$\frac{\boldsymbol{D}[\![e_0,\, i]\!] \quad i = 0}{\boldsymbol{D}[\![(\text{zero?}\ e_0),\, \#\text{t}]\!]}$$

$$\frac{\boldsymbol{D}[\![e_0,\, i]\!] \quad i \neq 0}{\boldsymbol{D}[\![(\text{zero?}\ e_0),\, \#\text{f}]\!]}$$

Some Ant, ick!

Some Ant, ick!

- Let's take a look at **`if`** again, with some helper rules

Some Ant, ick!

- Let's take a look at **`if`** again, with some helper rules

$$\frac{\boldsymbol{D}[\![e_0, i]\!] \quad i = 0}{\boldsymbol{D}[\![(\text{zero? } e_0), \#\text{t}]\!]} \qquad \frac{\boldsymbol{D}[\![e_0, i]\!] \quad i \neq 0}{\boldsymbol{D}[\![(\text{zero? } e_0), \#\text{f}]\!]}$$

$$\frac{}{\text{is-true}[\![\#\text{t}]\!]} \qquad \frac{}{\text{is-false}[\![\#\text{f}]\!]} \qquad \frac{}{\text{is-true}[\![i]\!]}$$

Things to consider

Things to consider

- All of the following are *syntactically valid* programs

Things to consider

- All of the following are *syntactically valid* programs
- What do you expect them to do (i.e. what do the semantics say about them)?

Things to consider

- All of the following are *syntactically valid* programs
- What do you expect them to do (i.e. what do the semantics say about them)?

```
(if 0 1 2)
```

Things to consider

- All of the following are *syntactically valid* programs
- What do you expect them to do (i.e. what do the semantics say about them)?

```
(if 0 1 2)
```

```
(if (zero? 1) 1 2)
```

```
(if #t 1 2)
```

Things to consider

- All of the following are *syntactically valid* programs
- What do you expect them to do (i.e. what do the semantics say about them)?

```
(if 0 1 2)
```

```
(if (zero? 1) 1 2)
```

```
(if #t 1 2)
```

```
(if #t (add1 #f) 2)
```

Let’s look at the interpreter

We'll do that in the terminal, as it's starting to get a bit too cumbersome

Let's experiment

```
dupe> (require "dupe_interp.rkt")
```

Things are bit tricky

Things are bit tricky

- Now let’s think about generating x86 code

Things are bit tricky

- Now let's think about generating x86 code
- Clearly, **`#f`** is not the same as **`0`**

Things are bit tricky

- Now let's think about generating x86 code
- Clearly, **`#f`** is not the same as **`0`**
  - How do we make sure that the values from the different types don't get mixed up?

** several people are typing...

** several people are typing...

- This is the crux of a *type system*

** several people are typing...

- This is the crux of a *type system*
- Different type systems have different tradeoffs

** several people are typing...

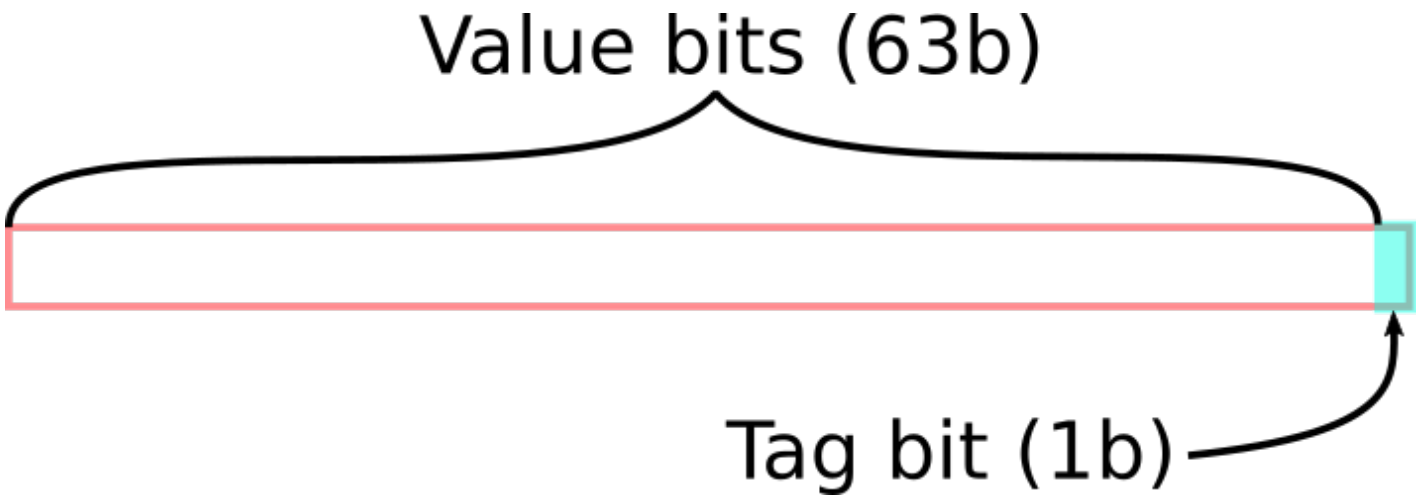
- This is the crux of a *type system*
- Different type systems have different tradeoffs
- We are going to implement a *dynamic* type system

** several people are typing...

- This is the crux of a *type system*
- Different type systems have different tradeoffs
- We are going to implement a *dynamic* type system
  - What does this imply about how our implementation doesn't get values from different types mixed up?

Tag your int

Tag your int



Tag your int

Tag your int

- We have to choose: which type gets **1**?

Tag your int

- We have to choose: which type gets **`1`**?
- Either can work, but we'll argue that **`booleans`** should get the **`1`**

Tag your int

Tag your int

- What does this imply about our

Tag your int

- What does this imply about our
  - Runtime system?

Tag your int

- What does this imply about our
  - Runtime system?
  - Compiler?

Let’s take a look at the RTS and compiler

Assignment 3

- Will go live tomorrow
  - Please tell your fellow students to check the webpage periodically
  - If there are any issues that might make you unable to do the assignment on time, *talk to me*