CMSC 430, Feb 25th 2020

Grift

Update

Update

- I have credentials!

Update

- I have credentials!
  - I got UMD credentials near the end of last week, and am now jumping through all the hoops to get your grades on ELMS/Canvas

Update

- I have credentials!
  - I got UMD credentials near the end of last week, and am now jumping through all the hoops to get your grades on ELMS/Canvas
  - Hoping to get grading done by the end of the week.

Update

- I have credentials!
  - I got UMD credentials near the end of last week, and am now jumping through all the hoops to get your grades on ELMS/Canvas
  - Hoping to get grading done by the end of the week.
  - I’m scraping the plan of having the TA disambiguate and am going to try and do it through ELMS. You should already see a quiz on ELMS?

Word to the wise

Word to the wise

- I’ve been seein some of the assignments that have been submitted

Word to the wise

- I’ve been seein some of the assignments that have been submitted
  - I am consistently seeing a very serious mistake!

Word to the wise

- I’ve been seein some of the assignments that have been submitted
  - I am consistently seeing a very serious mistake!
  - Unless you defined **`interp`** using macros, *you must quote your input expression!*

Word to the wise

- I’ve been seein some of the assignments that have been submitted
  - I am consistently seeing a very serious mistake!
  - Unless you defined **`interp`** using macros, *you must quote your input expression!*
  - Why is the following wrong?

Word to the wise

- I've been seein some of the assignments that have been submitted
  - I am consistently seeing a very serious mistake!
  - Unless you defined **`interp`** using macros, *you must quote your input expression!*
  - Why is the following wrong?

```
(check-equal? (interp (add1 1)) 2)
```

Word to the wise

Word to the wise

- This is partly my fault (and is why I stopped using macros in class)

Word to the wise

- This is partly my fault (and is why I stopped using macros in class)
  - Some of you wrote tests (yay!)

Word to the wise

- This is partly my fault (and is why I stopped using macros in class)
  - Some of you wrote tests (yay!)
  - But those tests are just testing racket, not your interpreter.

Word to the wise

- This is partly my fault (and is why I stopped using macros in class)
  - Some of you wrote tests (yay!)
  - But those tests are just testing racket, not your interpreter.
  - This is not a rare mistake. You should _all_ double-check your code.

Appreciating what we have:

Appreciating what we have:

- To recap, we've got:

Appreciating what we have:

- To recap, we've got:
  - *unary* arithmetic primitives

Appreciating what we have:

- To recap, we've got:
  - *unary* arithmetic primitives
  - Conditionals, for branching

Appreciating what we have:

- To recap, we've got:
  - *unary* arithmetic primitives
  - Conditionals, for branching
  - Errors that halt our programs

Appreciating what we have:

- To recap, we've got:
  - *unary* arithmetic primitives
  - Conditionals, for branching
  - Errors that halt our programs
  - let-bound variables

Grift

Grift

- What would be useful to add?

Fraud's AST

- **e = i | b | if e e e | let ((id e)) e | id | p e**

Fraud's AST

- **e = i | b | if e e e | let ((id e)) e | id | p e**
  - **p = add1 | sub1 | zero?**

Fraud's AST

- **e = i | b | if e e e | let ((id e)) e | id | p e**
  - **p = add1 | sub1 | zero?**
  - **id = variable**

Grift's AST

- We go
- from:

Grift's AST

- We go
- from:
  - `e = ... | p e`

Grift's AST

- We go
- to:

Grift's AST

- We go
- to:
  - **`e = ... | p1 e | p2 e e`**

Grift's AST

- We go
- to:
  - **`e = ... | p1 e | p2 e e`**
  - **`p1 = add1 | sub1 | zero?`**

Grift's AST

- We go
- to:
  - **`e = ... | p1 e | p2 e e`**
  - **`p1 = add1 | sub1 | zero?`**
  - **`p2 = + | -`**

Binary Operators!

Binary Operators!

- Interpretation is easy (as we'll see)

Binary Operators!

- Interpretation is easy (as we'll see)
- Compilation is not hard, but requires a non-trivial insight (as we'll see)

Binary Operators!

- Interpretation is easy (as we'll see)
- Compilation is not hard, but requires a non-trivial insight (as we'll see)
- Can anyone think of why interpretation might be much easier?

Meanings

Meanings

- Grift doesn’t add much:

Meanings

- Grift doesn't add much:
- First we factor out a rule for primitives

$$\frac{\textbf{\textit{G-env}}[\![e_0,\ r,\ a_0]\!] \quad \ldots}{\textbf{\textit{G-env}}[\![(p\ e_0\ \ldots),\ r,\ \textbf{\textit{G-prim}}[\![(p\ a_0\ \ldots)]\!]]\!]}$$

Meanings

- Grift doesn't add much:
- Then we use that rule

$$
\begin{aligned}
&\textbf{\textit{G-prim}} : (p\ a\ \ldots) \to a \\
&\textbf{\textit{G-prim}}[\![(p\ v\ \ldots\ \text{err}\ \_\ \ldots)]\!] &&= \text{err} \\
&\textbf{\textit{G-prim}}[\![(\text{add1}\ i_0)]\!] &&= (+\ i_0\ \ 1) \\
&\textbf{\textit{G-prim}}[\![(\text{sub1}\ i_0)]\!] &&= (-\ i_0\ \ 1) \\
&\textbf{\textit{G-prim}}[\![(\text{zero?}\ 0)]\!] &&= \#\text{t} \\
&\textbf{\textit{G-prim}}[\![(\text{zero?}\ i)]\!] &&= \#\text{f} \\
&\textbf{\textit{G-prim}}[\![(+\ i_0\ i_1)]\!] &&= (+\ i_0\ \ i_1) \\
&\textbf{\textit{G-prim}}[\![(-\ i_0\ i_1)]\!] &&= (-\ i_0\ \ i_1) \\
&\textbf{\textit{G-prim}}[\![\_]\!] &&= \text{err}
\end{aligned}
$$

Interpreter

Interpreter

- Switch to the terminal...

The Compiler

The Compiler

- We can’t do it naively, consider:

The Compiler

- We can't do it naively, consider:

```
(define (compile-+ e0 e1 c)
  (let ((c0 (compile-e e0 c))
        (c1 (compile-e e1 c)))
    `(,@c0
      ,@c1
      (add rax ???))))
```

The Compiler

The Compiler

- What are some alternatives?

The Compiler

- What are some alternatives?
- With those alternatives in mind, consider:

The Compiler

- What are some alternatives?
- With those alternatives in mind, consider:

```
(+ (add1 2) (add1 3))
```

The Compiler

- What are some alternatives?
- With those alternatives in mind, consider:

```
(+ (add1 2) (add1 3))
```

```
(+ (add1 2) 3)
```

The Compiler

- What are some alternatives?
- With those alternatives in mind, consider:

```
(+ (add1 2) (add1 3))

(+ (add1 2) 3)

(+ (add1 2) x)
```

The Compiler

The Compiler

- Before we dive in, let's review compiling `let` and add comments

The Compiler

- Before we dive in, let's review compiling `let` and add comments
  - Reminder to José: in assembly they're called `remarks'