PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

CSE341: Programming Languages

Section 8
Macros and Language Interpretation

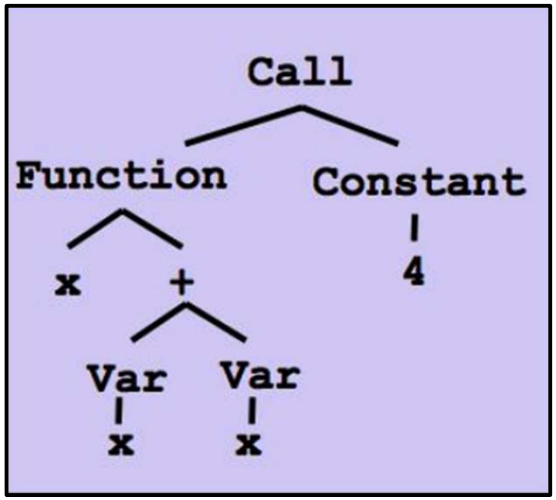
Spring 2020

Agenda

- Interpreting LBI (Language Being Implemented)
  - Assume correct syntax
  - Check for correct semantics
  - Evaluating the AST
- LBI “Macros”

Building an LBI Interpreter

- We are skipping the parsing phase
  - Can be skipped because AST (“Abstract Syntax Tree”) nodes represented as Racket structs
- LBI vs. Metalanguage
  - For HW5, MUPL is the LBI
  - Racket is the “metalanguage”



A Larger Language Example

```
(struct const (int) #:transparent)
(struct negate (e1) #:transparent)
(struct add (e1 e2) #:transparent)
(struct bool (b) #:transparent)
(struct multiply (e1 e2) #:transparent)
(struct eq-num (e1 e2) #:transparent)
(struct if-then-else (e1 e2 e3) #:transparent)
```

LBI → `(add (const 1) (const 1))`

Metalanguage → Racket structs/operations on structs/the above code

Let's try Prob 1 on the worksheet!

Correct Syntax Examples

Using these Racket structs…

```
(struct const (int) #:transparent)
(struct add (e1 e2) #:transparent)
(struct if-then-else (e1 e2 e3) #:transparent)
```

…we can interpret these LBI programs:

```
(const 34)
(add (const 34) (const 30))
(if-then-else (bool #t) (const 10) (const 20))
```

Incorrect Syntax Examples

While using these Racket structs…

```
(struct const (int) #:transparent)
(struct add (e1 e2) #:transparent)
(struct if-then-else (e1 e2 e3) #:transparent)
```

…we can assume we won't see LBI programs like:

```
(const "dan then dog")
(add 5 4)
(if-then-else (bool '(1 2)) (const 5) (bool #f))
```

Illegal input ASTs may crash the interpreter - this is OK

Racket vs. LBI

Structs in Racket, when defined to take an argument, can take any Racket value:

```
(struct const (int) #:transparent)
(struct add (e1 e2) #:transparent)
(struct if-then-else (e1 e2 e3) #:transparent)
```

But in LBI, we restrict const to take only an integer value, add to take two LBI expressions, and so on…

```
(const "dan then dog")
(add 5 4)
(if-then-else (bool '(1 2)) (const 5) (bool #f))
```

Illegal input ASTs may crash the interpreter - this is OK

LBI Semantics

- All values evaluate to themselves. This includes bool and const.
- An **`add`** evaluates its subexpressions and, assuming they both produce integers, produces the integer that is their sum.
- An **`if-then-else`** evaluates its first expression to a value v1. If it is a boolean, then if it is **`#t`**, then evaluates its second subexpression, else it evaluates its third subexpression.
- …...

Check for Correct Semantics

What if the program is a legal AST, but evaluation of it tries to use the wrong kind of value?

```
(struct const (int) #:transparent)
(struct add (e1 e2) #:transparent)
(struct if-then-else (e1 e2 e3) #:transparent)
```

This is invalid LBI syntax that we need to check for…

```
(add (const 1) (bool #t))
(if-then-else (const 5) (const 5) (bool #f))
```

You should detect this and give an error message that is not in terms of the interpreter implementation

Semantic Error or Illegal Program?

```
(const #t)
```

Illegal Program! Can assume const always contain numbers.

```
(negate (bool #t))
```

Semantic Error! Can only negate const. Must check for this!

```
(if-then-else (multiply (const 1) (const 2))
      (const 1) (const 2))
```

Semantic Error! e1 in if-then-else should evaluate to a bool. Must check for this!

```
(eq-num 5 (bool #f))
```

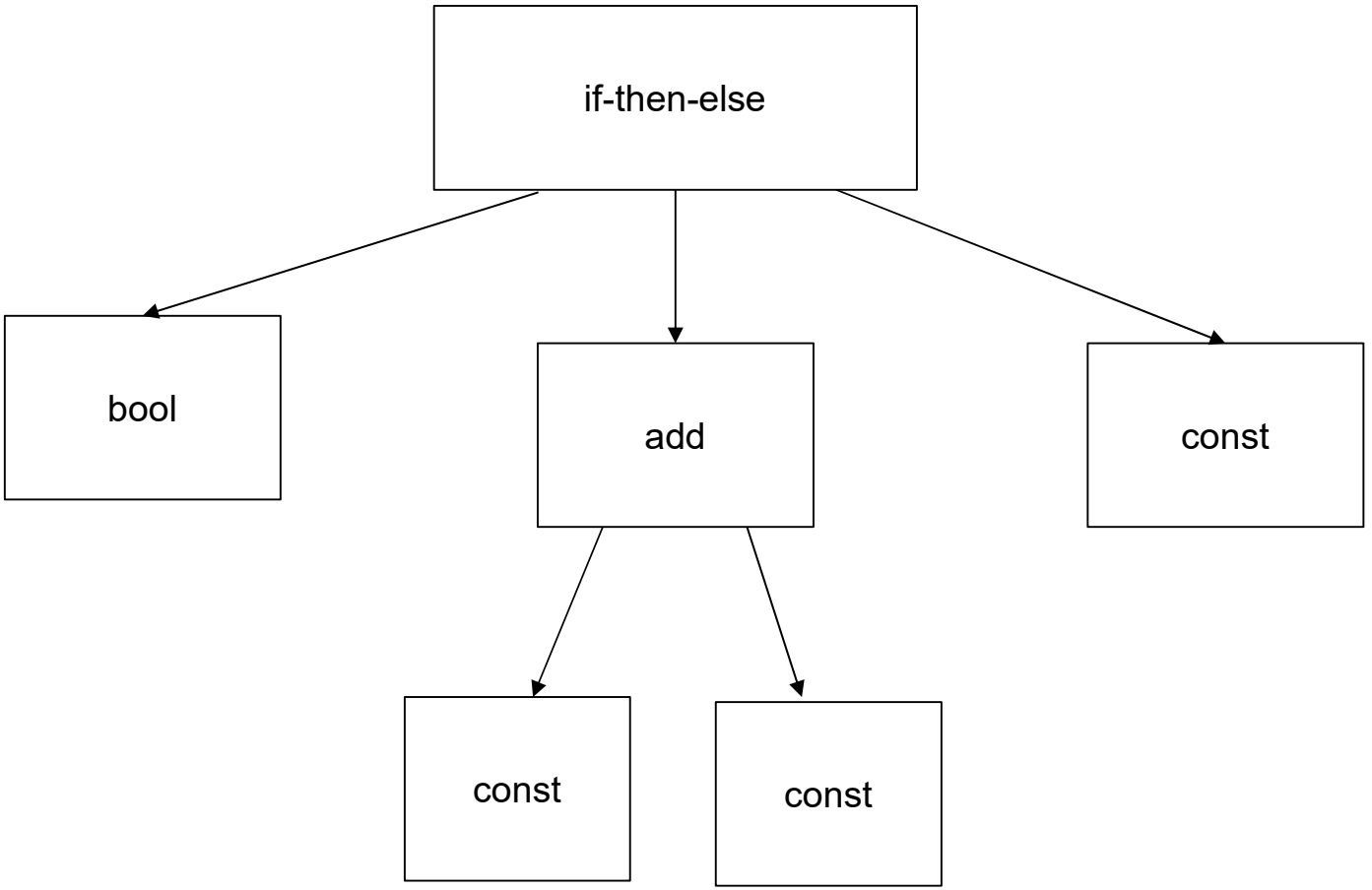
Both! 5 is not a valid expression (can assume these won't happen). However, e1/e2 in eq-num must evaluate to const, and bool is not a const, which we should check!

```
(multiply (eq-num (bool #t) (bool #f)) (const 3))
```

Semantic Error! e1 in multiply should evaluate to a const, but eq-num evaluates to a bool. Likewise, eq-num operates on consts, not bools. Should detect both of these!

What's the AST?

```
(if-then-else ; evaluates to (const 7)
    (bool #t) (add (const 3) (const 4)) (const 20))
```



Evaluating the AST

- **`eval-exp`** should return a LBI value
- LBI values all evaluate to themselves
- Otherwise, we haven't interpreted far enough

```
(const 7) ; evaluates to (const 7)
```

```
(add (const 3) (const 4)) ; evaluates to (const 7)
```

```
(if-then-else ; evaluates to (const 7)
    (bool #t) (add (const 3) (const 4)) (const 20))
```

Evaluating the AST

What's wrong with this implementation of eval? (other than it being called "**`eval-exp-wrong`**"...)

Evaluating the AST

- It doesn't recursively check for semantic correctness!!
  - Let's see a better version of this….

    …. by doing Problem #2 of the Worksheet!

Review: Macros

- Extend language syntax (allow new constructs)
- Written in terms of existing syntax
- Expanded *before* language is actually interpreted or compiled

Example Racket macro definitions

Two simple macros

```
(define-syntax my-if                    ; macro name
  (syntax-rules (then else)             ; other keywords
    [(my-if e1 then e2 else e3)  ; macro use
     (if e1 e2 e3)]))                   ; form of expansion
```

```
(define-syntax comment-out          ; macro name
  (syntax-rules ()                  ; other keywords
    [(comment-out ignore instead) ; macro use
     instead]))                     ; form of expansion
```

If the form of the use matches, do the corresponding expansion

- In these examples, list of possible use forms has length 1
- Else syntax error

Local variables in macros

In C/C++, defining local variables inside macros is unwise

- When needed done with hacks like **`__strange_name34`**

Here is why with a silly example:

- Macro:

```
(define-syntax dbl
  (syntax-rules ()
     [(dbl x) (let ([y 1])
                (* 2 x y))]))
```

- Use:

```
(let ([y 7]) (dbl y))
```

- Naïve expansion:

```
(let ([y 7]) (let ([y 1])
               (* 2 y y)))
```

- But instead Racket "gets it right," which is part of *hygiene*

How to implement “Macros” In LBI

- Interpreting LBI using Racket as the metalanguage
- LBI is made up of Racket structs
- In Racket, these are just data types
- Why not write a Racket function that returns LBI ASTs?

LBI “Macros”

If our LBI Macro is a Racket function:

```
(define (++ exp) (add (const 1) exp))
```

Then the LBI code

```
(++ (++ (const 7)))
```

Expands to:

```
(add (const 1) (add (const 1) (const 7)))
```

LBI “Macros”

If our LBI Macro is a Racket function:

```
((define (andalso e1 e2) (if-then-else e1 e2 (bool #f)))
```

Then the LBI code

```
(andalso (bool #t) (bool #t))
```

Expands to:

```
(if-then-else (bool #t) (bool #t) (bool #f))
```