Section 7

Patrick Larson

CSE341 – Spring 2013

Abstract Syntax Trees

```
(add 2 2)

(add (const 2) (const 2))

(add (const 2) (bool #t))
```

Abstract Syntax Trees

(add 2 2)

Incorrect Syntax

(add (const 2) (const 2))

Correct Syntax, Correct Semantics

(add (const 2) (bool #t))

Correct Syntax, Incorrect Semantics

Abstract Syntax Trees

When evaluating an AST, we are not required to check for bad syntax

But we ARE required to check for bad semantics

Abstract Syntax Trees

Evaluation can assume a legal syntax tree

If the input is illegal the evaluation may crash

(But this is okay!)

We DO need to check that the return types of sub-expressions are correct

Abstract Syntax Trees

Interpreter should return expression, but only expressions that evaluate to themselves.

Otherwise we haven’t interpreted far enough

MUPL “macros”

We are interpreting MUPL with Racket

And MUPL is just Racket data

So why not write a Racket function that returns a MUPL AST?

Quote

Racket statements can be thought of as lists of tokens

We can use the built in quote operation to turn a racket program into a list of tokens

We can use an apostrophe as syntactic sugar.

Quasiquote

Useful for inserting expressions into a quote

Use unquote to escape a quote and evaluate it

Quasiquote and quote are the same unless we have an unquote expression

Can use the back tick for quasiquote and , for unquote

Eval

Treat the input data as a program and run it!

This means we need a language implementation at runtime

(This is useful, but there is typically a better way to do things)

Eval

```
(define (make-some-code y) ; just returns a list
  (if y
      (list 'begin (list 'print "hi") (list '+ 4 2))
      (list '+ 5 3)))

(eval (make-some-code #t)) ; prints "hi", result 6
```