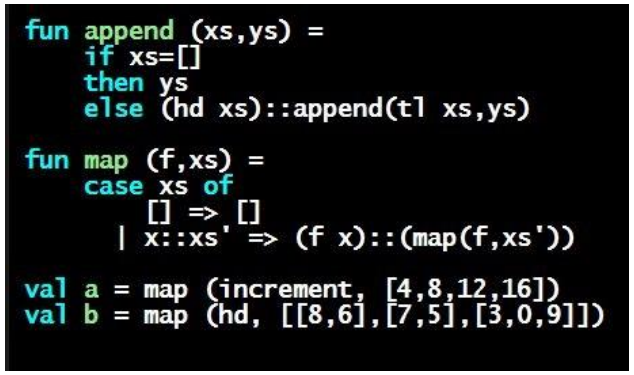

Programming Languages

Dan Grossman

What Your Interpreter Can and Cannot Assume

What we know

- Define (abstract) syntax of language *B* with Racket structs
  - *B* called MUPL in homework
- Write *B* programs directly in Racket via constructors
- Implement interpreter for *B* as a (recursive) Racket function

Now, a subtle-but-important distinction:

- Interpreter can *assume* input is a "legal AST for B"
  - Okay to give wrong answer or inscrutable error otherwise
- Interpreter *must check* that recursive results are the right kind of *value*
  - Give a good error message otherwise

Legal ASTs

- "Trees the interpreter must handle" are a subset of all the trees Racket allows as a dynamically typed language

```
(struct const (int) #:transparent)
(struct negate (e) #:transparent)
(struct add (e1 e2) #:transparent)
(struct multiply (e1 e2) #:transparent)
```

- Can assume "right types" for struct fields
  - **`const`** holds a number
  - **`negate`** holds a legal AST
  - **`add`** and **`multiply`** hold 2 legal ASTs

- Illegal ASTs can "crash the interpreter" – *this is fine*

```
(multiply (add (const 3) "uh-oh") (const 4))
(negate -7)
```

Interpreter results

- Our interpreters return expressions, but not any expressions
  - Result should always be a *value*, a kind of expression that evaluates to itself
  - If not, the interpreter has a bug
- So far, only values are from **`const`**, e.g., **`(const 17)`**
- But a larger language has more values than just numbers
  - Booleans, strings, etc.
  - Pairs of values (definition of value recursive)
  - Closures
  - …

Example

See code for language that adds booleans, number-comparison, and conditionals:

```
(struct bool (b) #:transparent)
(struct eq-num (e1 e2) #:transparent)
(struct if-then-else (e1 e2 e3) #:transparent)
```

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

- For example, “add a boolean”
- You should detect this and give an error message not in terms of the interpreter implementation
- Means checking a recursive result whenever a particular kind of value is needed
  - No need to check if any kind of value is okay