CS 476 – Programming Language Design

William Mansky

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Structure of a Language

- Syntax
  - Abstract – what are the pieces of a program?
  - Concrete – what do programs look like?
- Semantics
  - Static – what are acceptable programs?
  - Dynamic – what do programs do when we run them?
- Pragmatics
  - Implementation – how can we actually make the semantics happen?
  - IDE, tool support, etc.

Metalanguages

- We want to describe how languages should work, and write code that actually runs those languages
- Natural language: "x := y + z sets x to be y plus z"
- Inference rules: $$\frac{(e, \sigma) \Downarrow v}{(x := e, \sigma) \Downarrow \sigma[x \mapsto v]}$$
- OCaml:
```
match s with
      | Assign x e => update env x (eval env e)
```

- We'll learn to *translate* between these three metalanguages!

Defining Syntax for a Language

- This language has *expressions* and *commands*
- *Expressions* are things like adding or subtracting two numbers, or calling a function
- *Commands* include assigning values to variables, and if-then-else blocks

Defining Syntax for a Language

- This language has *expressions* ***E*** and *commands* ***C***
- *Expressions* are things like adding two numbers, or calling a function
- *Commands* include assigning values to variables, and if-then-else blocks

Defining Syntax for a Language

- This language has *expressions* ***E*** and *commands* ***C***

- ***E*** := <num> + <num> | <name> (***E***)

- *Commands* include assigning values to variables, and if-then-else blocks

Defining Syntax for a Language

- This language has *expressions* ***E*** and *commands* ***C***

- ***E*** := <num> + <num> | <name> (***E***)

- ***C*** := <name> = ***E*** ; | if (***E***) { ***C*** } else { ***C*** }

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- Each rule has a *nonterminal* on the left, and a sequence of *nonterminals* and *terminals* (letters, numbers, operators, etc.) on the right

E := <num> + <num> | <name> (***E***)

C := <name> = ***E*** ; | if (***E***) { ***C*** } else { ***C*** }

nonterminals: ***E*** ***C*** terminals: <num> <name> + ( ) = ; if else { }

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- Any string that can be made by applying the rules belongs to the language

E := <num> + <num> | <name> (*E*)

C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

C

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- Any string that can be made by applying the rules belongs to the language

E := <num> + <num> | <name> (*E*)

C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

C ⇒ x = *E* ;

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- Any string that can be made by applying the rules belongs to the language

E := <num> + <num> | <name> (*E*)

C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

C ⇒ x = *E* ; ⇒ x = f (*E*) ;

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- Any string that can be made by applying the rules belongs to the language

E := <num> + <num> | <name> (*E*)

C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

C ⇒ x = *E* ; ⇒ x = f (*E*) ; ⇒ x = f (2 + 4) ;

A program in the language of the grammar

- This is “just syntax”: it doesn’t do anything yet!

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- …that looks a lot like an inductive datatype!

E := <num> + <num> | <name> (*E*)

C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

```
type E = Add | Call
type C = Assign | If
```

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- …that looks a lot like an inductive datatype!
- We’re interested in the *abstract* syntax, the parts that can vary

E := <num> + <num> | <name> (*E*)
C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

```
type E = Add | Call
type C = Assign | If
```

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- …that looks a lot like an inductive datatype!
- We’re interested in the *abstract* syntax, the parts that can vary

E := <num> + <num> | <name> (*E*)
C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

```
type E = Add of int * int | Call
type C = Assign | If
```

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- …that looks a lot like an inductive datatype!
- We’re interested in the *abstract* syntax, the parts that can vary

E := <num> + <num> | <name> (*E*)
C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

```
type E = Add of int * int | Call of name * E
type C = Assign | If
```

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- …that looks a lot like an inductive datatype!
- We’re interested in the *abstract* syntax, the parts that can vary

E := <num> + <num> | <name> (*E*)
C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

```
type E = Add of int * int | Call of name * E
type C = Assign | If
```

Context-Free Grammars

- A series of rules describing a set of strings (“language”)
- …that looks a lot like an inductive datatype!
- We’re interested in the *abstract* syntax, the parts that can vary

E := <num> + <num> | <name> (*E*)
C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

```
type E = Add of int * int | Call of name * E
type C = Assign of name * E | If of E * C * C
```

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Grammars: Variations

E := <num> + <num> | <name> (*E*)

C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

```
type E = Add of int * int | Call of name * E
type C = Assign of name * E | If of E * C * C
```

- Exercise: Change one thing about the grammar of the language. How would that change the OCaml types?

Grammars: Variations

E := <num> + <num> | <name> (*E*)

C := <name> := *E* | if *E* then *C* else *C*

```
type E = Add of int * int | Call of name * E
type C = Assign of name * E | If of E * C * C
```

- Exercise: Change one thing about the grammar of the language. How would that change the OCaml types?

Grammars: Variations

E := <num> + <num> - <num> | <name> (*E*)

C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

```
type E = Add of int * int * int | Call of name
* E
type C = Assign of name * E | If of E * C * C
```

- Exercise: Change one thing about the grammar of the language. How would that change the OCaml types?

Grammars: Variations

E := <num> + <num> | <name> (*E*) | <num> * 2

C := <name> = *E* ; | if (*E*) { *C* } else { *C* } | *C*; *C*

```
type E = Add of int * int | Call of name * E |
Mul2 of int
type C = Assign of name * E | If of E * C * C |
Seq of C * C
```

- Exercise: Change one thing about the grammar of the language. How would that change the OCaml types?

Grammars: Variations

E := <num> + <num> | <name> (*E*)

C := <name> = *E* ; | if (*E*) { *C* } else { *C* }

```
type E = Add of int * int | Call of name * E
type C = Assign of name * E | If of E * C * C
```

- Exercise: Change one thing about the grammar of the language. How would that change the OCaml types?

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Defining Syntax for a Language

- Step 1: write down what's in the language in English
- Step 2: write a grammar that describes all possible programs
- Step 3: write a datatype that abstracts the grammar
- Result: a datatype of *programs in the language*

Writing Functions on Syntax

```
type E = Add of int * int | Call of name * E
type C = Assign of name * E | If of E * C * C

let my_prog : C = If (Add (3, 4), Assign (“x”, Add (3, 4)), …)

let rec print_vars (prog : C) =
  match prog with
  | Assign (x, e) -> print_string x
  | If (cond, tcase, fcase) ->
        print_vars tcase; print_vars fcase
```

Writing Functions on Syntax

- Step 1: write down what's in the language in English
- Step 2: write a grammar that describes all possible programs
- Step 3: write a datatype that abstracts the grammar
- Result: a datatype of *programs in the language*

- Now we can write programs that operate on programs!

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.