CS 476 – Programming Language Design

William Mansky
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.
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 \textit{expressions} \(E\) and \textit{commands} \(C\)

• \textit{Expressions} are things like adding two numbers, or calling a function

• \textit{Commands} include assigning values to variables, and if-then-else blocks
Defining Syntax for a Language

• This language has expressions $E$ and commands $C$

• $E := <\text{num}> + <\text{num}> \mid <\text{name}> (E)$

• *Commands* include assigning values to variables, and if-then-else blocks
Defining Syntax for a Language

• This language has expressions $E$ and commands $C$

• $E := <\text{num}> + <\text{num}> \mid <\text{name}> (E)$

• $C := <\text{name}> = E ; \mid \text{if} (E) \{C\} \text{else} \{C\}$
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

\[
E := \langle\text{num}\rangle + \langle\text{num}\rangle \mid \langle\text{name}\rangle (E)
\]
\[
C := \langle\text{name}\rangle = E; \mid \text{if } (E) \{ C \} \text{ else } \{ C \}
\]

nonterminals: \textbf{E} \textbf{C} \quad \text{terminals: } \langle\text{num}\rangle \langle\text{name}\rangle + ( ) ; \text{ if else } \{ \}
Questions?
Context-Free Grammars

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

\[ E := \langle \text{num} \rangle + \langle \text{num} \rangle \mid \langle \text{name} \rangle (E) \]
\[ C := \langle \text{name} \rangle = E ; \mid \text{if} (E) \{C\} \text{else} \{C\} \]

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

\[
E := \texttt{<num>} + \texttt{<num>} \mid \texttt{<name> ( E )}
\]

\[
C := \texttt{<name> = E ;} \mid \texttt{if ( E )\{ C \} else \{ C \}}
\]

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

\[
E := \langle \text{num} \rangle + \langle \text{num} \rangle \mid \langle \text{name} \rangle \left( E \right) \\
C := \langle \text{name} \rangle = E ; \mid \text{if} \left( E \right) \{ C \} \text{else} \{ C \}
\]

type E = Add of int * int \mid \text{Call} \\
type C = Assign \mid \text{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

\[
E ::= \text{<num>} + \text{<num>} | \text{<name>} (E)
\]
\[
C ::= \text{<name>} = E; | \text{if } (E)\{C\} \text{ else } \{C\}
\]

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

\[ E ::= \langle\text{num}\rangle + \langle\text{num}\rangle \mid \langle\text{name}\rangle(\langle E \rangle) \]
\[ C ::= \langle\text{name}\rangle = \langle E \rangle; \mid \text{if} (\langle E \rangle)\{ \langle C \rangle \}\text{else}\{ \langle C \rangle \} \]

\[
\begin{align*}
\text{type } E &= \text{Add of int} \ast \text{int} \mid \text{Call of name} \ast E \\
\text{type } C &= \text{Assign} \mid \text{If}
\end{align*}
\]
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

\[
E := <\text{num}> + <\text{num}> \mid <\text{name}> (E)
\]

\[
C := <\text{name}> = E \mid \text{if} (E) \{ C \} \text{else} \{ C \}
\]

\[
\text{type } E = \text{Add of int} \ast \text{int} \mid \text{Call of name} \ast E
\]

\[
\text{type } C = \text{Assign of name} \ast E \mid \text{If of } E \ast C \ast C
\]
Questions?
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*

• Exercise: what are some functions you might want to run on a program?
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 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?