Concrete and Abstract Syntax

CS 131: Programming Languages
September 18, 2001

Syntax

• The legal “form” or “structure” of programs
  – How sub-constructs are put together to get larger constructs
  – Correct syntax is a precondition for being a valid program.

• Syntax is frequently distinguished from semantics, which relates to the meaning of programs.

Describing Syntax

• Computer languages need precisely-defined syntax
  – Otherwise, no way to make a program portable between implementations.
  – Can use this to automate program-analysis tools

• Can use results from formal language theory
  – Regular expressions
  – Context-free languages

Formal Languages

• A formal language is a set of finite strings of symbols
• Examples:
  – the set of all English words starting with “q”
  – the set of all natural numbers written in base 10
  – the set of all valid C programs
  – the set {"", "a", "b", "ab"}
• Given a finite string, we can ask whether or not it is in a given language.
Regular Expressions

- A regular expression is a way of denoting certain languages (called regular languages)
- Notation
  - The symbol $\emptyset$ is a regular expression denoting the empty language.
  - The symbol $\epsilon$ is a regular expression denoting the language containing only the empty string $"\"$.
  - Any other symbol $a$ is a regular expression denoting the language containing the single string $"a"$.

Regular Expression Examples

- Set of all binary numbers

Abbreviations

- It is often convenient to make some abbreviations:

  - $[\text{abcde}]$ The set \{"a","b","c","d","e"\}
  - $[\text{a-z}]$ The set \{"a","b","\ldots","z"\}
  - $[\text{AC-ES}]$ The set \{"A","C","D","E","\#\"\}

  - $r+$
  - $r(\epsilon^*)$
  - $r?$
  - $r^+$
More R. E. Examples

- SML (non-symbolic) identifiers, which must begin with a letter, and then may have any string of letters, digits, underscores, and primes.

- Ada identifiers, which must begin with a letter and then may have any string of letters, digits and underscores, with the proviso that underscores may only occur one at a time and cannot be the last character.

Limitations of R. E.'s

- Regular expressions are very useful for describing "tokens" of a language:
  - keywords
  - valid variable names
  - valid constants
  - pieces of punctuation

- Consider language of balanced parentheses
  \{"", ",", "((()", "(()(", "(()))", ...\}

- Regular expressions correspond to finite automata, which have finite memories.
  - These cannot count arbitrarily high
  - Hence this language cannot be described by a regular expression.

- We need to be able to require correct "bracketing" in syntax
  - e.g., parentheses, or let ... in ... end

Limitations of RE's

- Consider language of balanced parentheses
  \{"", ",", "((()", "(()(", "(()))", ...\}

- Regular expressions correspond to finite automata, which have finite memories.
  - These cannot count arbitrarily high
  - Hence this language cannot be described by a regular expression.

- We need to be able to require correct "bracketing" in syntax
  - e.g., parentheses, or let ... in ... end

BNF Grammars

- The most common way to specify a language grammar is using Backus-Naur form, or BNF.
  - This corresponds to the formal-language definition of "context-free languages".
BNF Example: Simple Arithmetic

```plaintext
<digit> ::= 0 | 1 | 2 | 3 | 4
        | 5 | 6 | 7 | 8 | 9
<number> ::= <number><digit> | <digit>
<exp> ::= <exp> + <exp> | <exp> - <exp>
         | ( <exp> ) | <number>
```

::= specifies an "is-a" relationship
Alternatives are separated by vertical bars.
<digit> and <number> and <exp> are called nonterminals
actual digits, +, -, (, and ) are called terminal symbols.

Example Production Sequence

```plaintext
<digit> ::= 0 | 1 | 2 | 3 | 4
        | 5 | 6 | 7 | 8 | 9
<number> ::= <number><digit> | <digit>
<exp> ::= <exp> + <exp> | <exp> - <exp>
         | ( <exp> ) | <number>

<number>  → <number><digit>
         → <number><digit><digit>
         → <digit><digit><digit>
         → 3<digit><digit>
         → 34<digit>
         → 345
```

Production Sequences

• A production sequence is
  – a sequence of strings (which may contain both terminals and nonterminals)
  – where each string is obtained from the previous one by expanding out a single nonterminal

BNF Example: Nested Parens

```plaintext
<P> ::= ε
     | ( <P> )
     | <P><P>
<P>  → ( <P> )
     → ( <P><P> )
     → ( ( <P> ) )
     → ( ( )<P> )
     → ( ( ) )<P>
     → ( ( ) )
```

Example Production Sequence

```plaintext
<number>  → <number><digit>
         → <number><digit><digit>
         → <digit><digit><digit>
         → 3<digit><digit>
         → 34<digit>
         → 345
```
Representing Programs

- The programmer treats code as simply a long sequence of characters.
  - This is not an efficient representation for compilers or interpreters
  - Does not directly represent the structure of the program.
- We can retain more information by remembering why we believe the program is syntactically valid.
  - We could remember a production sequence for the program, but this is even bigger and includes information we don’t care about.
  - But, we can summarize the production sequence by using parse trees.

Parse Trees

- The parse tree for a program is a representation of a production sequence (actually, many equivalent sequences)
  - Leaves are terminals
  - Internal nodes are nonterminals
  - The children of each node are the items that replaced that nonterminal

Parse Tree for 2-3+5

```
<exp> ::= <exp> + <exp> | <exp> - <exp>
 | ( <exp> ) | <digit>
```

```
<exp>
  └── └──<exp>
      └──<exp>
          └──<exp>
              └──<exp>
                  └──<exp>
                      └──<exp>
                          └──<exp>
                              └──<exp>
                                  └──<exp>
                                      └──<exp>
                                          └──<exp>
                                              └──<exp>
                                                  └──<exp>
                                                      └──<exp>
                                                          └──<exp>
                                                              └──<exp>
                                                                  └──<exp>
                                                                      └──<exp>
                                                                          └──<exp>
                                                                              └──<exp>
                                                                                  └──<exp>
                                                                                      └──<exp>
                                                                                          └──<exp>
                                                                                              └──<exp>
                                                                                                  └──<exp>
                                                                                                      └──<exp>
                                                                                                          └──<exp>
                                                                                                              └──<exp>
                                                                                                                  └──<exp>
                                                                                                                      └──<exp>
                                                                                                                          └──<exp>
                                                                                                                              └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                                └──<exp>
                                                                                                                                                    └──<exp>
                                                                                                                                                        └──<exp>
                                                                                                                                                            └──<exp>
                                                                                                                                                                └──<exp>
                                                                                                                                                                    └──<exp>
                                                                                                                                                                        └──<exp>
                                                                                                                                                                                └──<exp>
                                                                                                                                                                                        └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
                                                                                                                                  └──<exp>
An Unambiguous Grammar

Claim: Every arithmetic expression has a unique parse tree according to the following grammar.

\[
\begin{align*}
\text{<exp>} & \ ::= \text{<exp>} + \text{<term>} \\
& \quad \mid \text{<exp>} - \text{<term>} \\
& \quad \mid \text{<term>} \\
\text{<term>} & \ ::= \text{<term>} \ast \text{<factor>} \\
& \quad \mid \text{<factor>} \\
\text{<factor>} & \ ::= \left\{ \text{<exp>} \right\} \\
& \quad \mid \text{<digit>} \\
\end{align*}
\]

2–3+5 Unambiguously

Claim: Every arithmetic expression has a unique parse tree according to the following grammar.

\[
\begin{align*}
\text{<exp>} & \ ::= \text{<exp>} + \text{<term>} \\
& \quad \mid \text{<exp>} - \text{<term>} \\
& \quad \mid \text{<term>} \\
\text{<term>} & \ ::= \text{<term>} \ast \text{<factor>} \\
& \quad \mid \text{<factor>} \\
\text{<factor>} & \ ::= \left\{ \text{<exp>} \right\} \\
& \quad \mid \text{<digit>} \\
\end{align*}
\]

2–(3+5) Unambiguously

Parse Tree Critique

- The parse tree is a better representation of the program than character strings
  - Shows separates subexpressions
  - Shows grouping
- But it contains a lot of junk
  - Who cares whether 3 is a num or a term or a factor or an exp?
  - Do we really have to remember the parentheses?
Abstract Syntax

• Idea: remember the bare essentials

Writing Abstract Syntax

• We will frequently want to write down a piece of abstract syntax, but trees are tedious.
• Therefore we will write abstract syntax as an ordinary expression, and the underlying tree is implicit
  – We throw in parentheses as needed
  – Use conventions like * having higher precedence than +, and - being left-associative
  – But we’re always referring to an specific tree

Lexing and Parsing

• Modern compilers usually start with a lexer and a parser
  – Lexer: breaks input into tokens
  – Parser: turns tokens into trees
• Tools exist for automatically generating these from a language description.
  – Lexer needs RE’s describing tokens
  – Parser needs BNF describing grammar
Abstract vs. Concrete Syntax

- **Concrete Syntax**
  - What the user sees
  - Concerned with programs as strings of characters
  - How to resolve ambiguities (e.g., precedence and associativity of operators)
  - Spelling of keywords, punctuation, formatting, etc.

- **Abstract Syntax**
  - What the compiler needs to remember
  - Concerned with programs as structured data
  - No ambiguities remaining
  - Parsing details abstracted away

Concrete vs. Abstract Syntax

- Concrete syntax is an API for the language
- Can choose very different concrete syntaxes which map to the same abstract syntax

```plaintext
fun fact(x) = if (x = 0) then 1 else x*fact(x-1)
```

```plaintext
(define (fact x)
  (if (eq x 0) 1 (* x (fact (- x 1))))
)
```

Binding and Scope

- Most language have a notions of
  - Variable binding (declaration of new variable)
  - Scope of variables (where variables can be referenced)

```plaintext
let val x = 3 in x + x end
```

- Here x is a bound variable
- The scope of x is the expression x + x

Bound Variables

- Every use of a bound variable refers to a binding

```plaintext
let val x = 3 in x + x end
```

```plaintext
let val x = 10 in (let val x = x+1 in x + x end) + x end
```

- Nested bindings of same variable called “shadowing”
  - Usual rule: use of variable refers to nearest enclosing binder.
Renaming Bound Variables

• In sane languages, choices of bound variables don’t matter:

\[
\begin{align*}
\text{fn}(x : \text{int}) &= x + 1 \\
\text{fn}(y : \text{int}) &= y + 1 \\
\text{fn}(### : \text{int}) &= ### + 1 \\
\end{align*}
\]

\[
\begin{align*}
\text{let val } x = 3 \text{ in } x + x \text{ end} \\
\text{let val } y = 3 \text{ in } y + y \text{ end} \\
\text{let val } ### = 3 \text{ in } ### + ### \text{ end}
\end{align*}
\]

α-conversion

• Systematic renaming of bound variables is called α-conversion
• Shadowing can then always be avoided

\[
\begin{align*}
\text{let val } x = 10 \text{ in } \\
\text{ in (let val } x = x + 1 \text{ in } x + x \text{ end) + x end}
\end{align*}
\]

\[
\begin{align*}
\text{let val } x = 10 \text{ in } \\
\text{ in (let val } y = x + 1 \text{ in } y + y \text{ end) + x end}
\end{align*}
\]

α-equivalence

• Expressions that differ only in the names of bound variables said to be α-equivalent:
  - If α-conversion does not change meaning, then it is often convenient to ignore names of bound variables.
  - Formally: α-equivalent expressions are considered equal/equivalent/the same/indistinguishable.
  - More formally: abstract syntax is equivalence classes of expressions under α-equivalence

Free Variables

• Variables used but not bound are said to be “free”

\[
\begin{align*}
\text{let val } x = 3 \text{ in } x + y \text{ end}
\end{align*}
\]

(Here \(x\) is bound and \(y\) is free)

\[
\begin{align*}
x + \text{let val } x = 3 \text{ in } x + x \text{ end}
\end{align*}
\]

(Here \(x\) occurs both bound and free)
Free Variables

• Free-ness of variables is relative

\texttt{let \textbf{val} \ x = 3 \ \textbf{in} \ x + y \ \textbf{end}}

Here \(x\) and \(y\) are free in the “body” of the \texttt{let},
but \(x\) is \textit{not} free in the entire expression.

Substitution

• Replacing \textit{free} variables with terms

\begin{align*}
(x + \texttt{let} \ x = 3 \ \textbf{in} \ x + y \ \textbf{end})[y \leftarrow z+1] \\
= (x + \texttt{let} \ x = 3 \ \textbf{in} \ x + (z+1) \ \textbf{end})
\end{align*}

\begin{align*}
(x + \texttt{let} \ x = 3 \ \textbf{in} \ x + y \ \textbf{end})[x \leftarrow z+1] \\
= ((z+1) + \texttt{let} \ x = 3 \ \textbf{in} \ x + y \ \textbf{end})
\end{align*}

Substitution

• Unless otherwise specified, \textit{capture-avoiding} substitution.
  
  - Particularly if we when identifying terms up to \(\alpha\)-equivalence
  
  - Free vars in substituted expression must stay free.

Then,

\begin{align*}
(y + \texttt{let} \ x = 3 \ \textbf{in} \ x + y \ \textbf{end})[y \leftarrow x+1] \\
\text{is not} \\
(x+1) + \texttt{let} \ x = 3 \ \textbf{in} \ x + (x+1) \ \textbf{end}
\end{align*}