Declarative Computation Model
Defining practical programming languages (VRH2.1)

Carlos Varela
RPI
September 18, 2006
Adapted with permission from:
Seif Haridi
KTH
Peter Van Roy
UCL

Declarative Programming Model

• Guarantees that the computations are evaluating functions on (partial) data structures
• The core of functional programming (LISP, Scheme, ML, Haskell)
• The core of logic programming (Prolog, Mercury)
• Stateless programming vs. stateful (imperative) programming
• We will see how declarative programming underlies concurrent and object-oriented programming (Erlang, Java, SALSA)

Defining a programming language

• Syntax (grammar)
• Semantics (meaning)

Language syntax

• Defines what are the legal programs, i.e. programs that can be executed by a machine (interpreter)
• Syntax is defined by grammar rules
• A grammar defines how to make ‘sentences’ out of ‘words’
• For programming languages: sentences are called statements (commands, expressions)
• For programming languages: words are called tokens
• Grammar rules are used to describe both tokens and statements

Language syntax (2)

• A statement is a sequence of tokens
• A token is a sequence of characters
• A program that recognizes a sequence of characters and produces a sequence of tokens is called a lexical analyzer
• A program that recognizes a sequence of tokens and produces a sequence of statement representation is called a parser
• Normally statements are represented as (parse) trees
### Extended Backus-Naur Form

- EBNF (Extended Backus-Naur Form) is a common notation to define grammars for programming languages.
- Terminal symbols and non-terminal symbols
- Terminal symbol is a token
- Nonterminal symbol is a sequence of tokens, and is represented by a grammar rule (nonterminal) ::= (rule body)

### Grammar rules

- (digit) ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- (digit) is defined to represent one of the ten tokens 0, 1, …, 9
- The symbol ‘|’ is read as ‘or’
- Another reading is that (digit) describes the set of tokens {0,1,…,9}
- Grammar rules may refer to other nonterminals
- (integer) ::= (digit) { (digit) }
- (integer) is defined as the sequence of a (digit) followed by a zero or more (digit)’s

### How to read grammar rules

- (a) is a nonterminal
- (a) ::= Body : (a) is defined by Body
- (a) | (b) : (a) is a choice
- (a) (b) : the sequence (a) followed by (b)
- { (a) } : a sequence of zero or more occurrences of (a)
- [ (a) ] : a sequence of one or more occurrences of (a)
- [0] : zero or one occurrences of (a)
- Read the grammar rule from left to right to give the following sequence:
  - Each terminal symbol is added to the sequence
  - Each nonterminal is replaced by its definition
  - For each (a) | (b) pick any of the alternatives
  - For each (a) (b) (c) pick any of the alternatives

### Context-free and context-sensitive grammars

- Grammar rules can be used to either
  - verify that a statement is legal, or
  - to generate all possible statements
- The set of all possible statements generated from a grammar and one nonterminal symbol is called a (formal) language
- EBNF notation defines a class of grammars called context-free grammars
- Expansion of a nonterminal is always the same regardless of where it is used
- For practical languages context-free grammar is not enough, usually a condition on the context is sometimes added

### Context-free grammar (e.g. with EBNF)

- It is easy to read and understand
- Defines a superset of the language
- Expresses restrictions imposed by the language (e.g. variable must be declared before use)
- Makes the grammar rules context sensitive

### Examples

- (statement) ::= if (expression) then (statement) { else (statement) } end | ...
- (expression) ::= (variable) | (integer) | ...

### Set of extra conditions

- (statement) ::= if (expression) then (statement) { else (statement) } end | ...
Example: (Parse Trees)

- if (expression) then (statement)₁ else (statement)₂ end

Language Semantics

- Semantics defines what a program does when it executes
- Semantics should be simple and yet allows a programmer to reason about programs (correctness, execution time, and memory use)
- How can this be achieved for a practical language that is used to build complex systems (millions lines of code)?
- The kernel language approach

Kernel Language Approach

- Define a very simple language (kernel language)
- Define the computation model of the kernel language
- By defining how the constructs (statements) of the language manipulate (create and transform) the data structures (the entities) of the language
- Define a mapping scheme (translation) of full programming language into the kernel language
- Two kinds of translations: linguistic abstractions and syntactic sugar

Linguistic abstractions vs. syntactic sugar

- Linguistic abstractions, provide higher level concepts that the programmer can use to model and reason about programs (systems)
- Examples: functions (fun), iterations (for), classes and objects (class), mailboxes (receive)
- The functions (calls) are translated to procedures (calls)
- The translation answers questions about the functions: [F1 (F2 X) [F3 X]]
Approaches to semantics

- Programming Language
  - Operational model
  - Formal calculus
  - Abstract machine

- Kernel Language
  - Aid the programmer in reasoning and understanding
  - Mathematical study of programming languages
    - λ-calculus, π-calculus

- Aid to the implementer
  - Efficient execution on a real machine

Exercises

20. Write a valid EBNF grammar for lists of non-negative integers in Oz.

21. Write a valid EBNF grammar for the λ-calculus.
   - Which are terminal and which are non-terminal symbols?
   - Draw the parse tree for the expression: \((\lambda x. x)\) \((\lambda y. y)\) \((\lambda z. z)\)

22. The grammar
   \[
   \langle \text{exp} \rangle ::= \langle \text{int} \rangle \ | \ \langle \text{exp} \rangle \ \langle \text{op} \rangle \ \langle \text{exp} \rangle
   
   \langle \text{op} \rangle ::= + \ | \ *
   \]

   is ambiguous (e.g., it can produce two parse trees for the expression \(2*3+4\)). Rewrite the grammar so that it accepts the same language unambiguously.