Syntax and Semantics

CS4700

Kenneth Sundberg

Language

- Given an alphabet (a set of characters)
- A language is a set of character strings
- All computable questions can be framed in terms of languages

Recognizers

- A recognizer for a language is a machine that returns true or false given a string of characters
- The recognizer only returns true for elements of the language
- Some recognizers may never return at all on some inputs, those inputs are not in the language

Generators

- A generator is a machine (sometimes non-deterministic) that can produce every element of a language

Hierarchy of computable languages

- There are three types of computable languages
    - Regular
    - Context Free
    - Context Sensitive
- Each langauge class is a proper subset of the next

Lexemes

- A Lexeme is a word in a programming language
- Forms the most basic structure
- The language of legal lexemes for a programming language is regular
- Examples:
  - Identifiers
  - Keywords

Lexical Analysis

Regular Expressions

- Concatenation
- Alternation
- Kleene Closure

Finite Automata

FA = RE

NFA = FA

Context Free Grammars

- Alphabet of terminal symbols (same alphabet as language)
    - Lexemes
- Alphabet of non-terminal symbols (Not in the language)
- Set of re-writing rules
    - Left side is a single non-terminal
    - Right side is a string of symbols (terminal, non-terminal, or mixed) which may be empty

Parsing

- CFGs are convenient for people and are good generators
- A parser is a mechanical acceptor of a context free grammar
- Also known as a push down automata
  - CFG = PDA
  - NPDA = PDA
- Parsing produces a parse tree (may be implicit)

Ambiguity

- A sentence in a language for which there is more than one possible parse tree is ambiguous
- Programming languages do not tolerate ambiguity well

Operator precedence

- A common source of ambiguity are math operators
- Unambiguous grammars exist but require more states
- The issue can be solved more efficiently with precedence rules

Recursive Descent

- Recursive descent parsing (LL parsing) is simple to write
- Write a function for each non-terminal
- Following the right hand side if there is a terminal consume it if there is a non-terminal invoke the appropriate function
- Unable to cope with left recursion - making it generally unsuitable for programming languages

LR Parser

- LR parsing is more general than LL
- Produces very fast , table based parsers
- The algorithm is beyond the scope of the course and is very prone to human error
    - A very important use case for goto
    - Generated by programs called compiler compilers (javacc, bison, etc)

Arithmetic Parsing

Static Semantics

- Some rules of languages are difficult (or impossible) to specify only with a CFG
- Any rule that depends on context
    - Type checking
    - Undefined identifiers

Attribute Grammars

Arithmatic Grammar

Dynamic Semantics

- What does a program mean?
- What are the defined run time behaviors

Operational Semantics

- Define meaning of program in terms of state changes on a low level machine
- Changes are too small and numerous to be of much use
- Natural Operational Semantics - Define the effect of entire program
- Structural Operational Semantics - Define low level sequence of state changes

Denotational Semantics

- Map language constructs onto math objects
- Typically functions, but also possibly ranges, categories, groups and others
- Complex denotational semantics generally indicate a difficult to implement language

Axiomatic Semantics

- The program means what it can be proven to mean
  - No model of a state machine
  - Only the fixed relations between language entities
- Useful in program verification
  - Preconditions
  - Postconditions

Textbook sections covered:

- Section 03-02 (frame 6)
- Section 03-03 (frame 12)
- Section 03-04 (frame 19)
- Section 03-05 (frame 22)
- Section 04-02 (frame 7)
- Section 04-03 (frame 13)
- Section 04-04 (frame 16)
- Section 04-05 (frame 17)