COP4020 Spring 2008 Homework-1 Solutions

1. Exercise 1.1 (Scott):

Errors in a computer program can be classified according to when they are detected and, if they are detected at compile time, what part of the compiler detects them. Using your favorite imperative language, give an example or each of the following.

(a) A lexical error, detected by the scanner
(b) A syntax error, detected by the parser
(c) A static semantic error, detected by semantic analysis
(d) A dynamic semantic error, detected by code generated by the compiler
(e) An error that the compiler can neither catch nor easily generate code to catch(this should be a violation of the language definition, not just a program bug)

Solution:
There are many possible answers to this question.

(1)Pascal:
Lexical error: '@' sign outside of a string or comment
Syntax error: mismatched parentheses in an arithmetic expression
Static semantic error: use of an identifier that was never declared
Dynamic semantic error: attempt to access an element beyond the bounds of an array
Error that can't reasonably be caught: modification of the index of a for loop by a subroutine called from within the loop

(2) Java:
Lexical error: '0a', 0 cannot be at the beginning of a string.
Syntax error: missing ';' at the end of a statement
Static semantic error: Access a method that is not declared in the class or the superclasses.
Dynamic semantic error: attempt to access an element beyond the bounds of an array
Error that can't reasonably by caught: use method name as a variable

2. Please describe the characteristics of the following languages:

$\sum = \{0, 1\}$

(1) $L_1 = \{ 0^n1^n \mid n \geq 1\}$
(2) $L_2 = \{ 0^n1^m0^n \mid n, m \geq 1\}$
(3) $L_3 = \{ awa \mid a \in \sum^*, w \in \sum^+\}$

Solution:
(1) L_1 is a language with first more than one 0s and then the same number of 1s.
(2) L_2 includes sentences with same number of 0s in the front and back, while in the middle, there are more than one 1s.
(3) L_3 begins and ends with the same 0 or more 0|1 strings, and more than one 0|1 strings in the middle.

3. Give a derivation:
Given a grammar G = (Φ, Σ, P, S), where
Φ = {S, A, B, C, D}
Σ = {a, b, c, d, #}
P = { S→ABCD | abc#
A→aaA
AB→aabbB
BC→bbccC
cC→cccC
CD→ccd#
CD→d#

CD→#d }

Please give the derivation of string: aaaaaabbbbcccc#d.

Solution:

S => ABCD => aaABCD => aaaaABCD => aaaaaabbBCD => aaaaaabbbbccCD
=> aaaaaabbbbccccCD => aaaaaabbbbcccc#d

4. Exercise 1.2 (Scott):

Algol family languages are typically compiled, while Lisp family languages, in which many issues cannot be settled until run time, are typically interpreted. Is interpretation simply what one “has to do” when compilation is infeasible, or are there actually some advantages to interpreting a language, even when a compiler is available? (What's the advantage and disadvantage of both compiler and interpreter?)

Solution:

Compiled code usually runs significantly faster than interpreted code, so programmers interested in performance tend to demand compilers, even for languages like Lisp. Compilation also catches some errors earlier, when they are easier to fix, rather than waiting to detect them until the program actually runs. Interpretation definitely has its advantages, however; it is not just a last resort. It is very handy during development because it allows a newly modified program to be executed without waiting for potentially time-consuming compilation and linking steps. It may be desirable on small machines because it takes less space, both in memory when running and on disk (because there are no executables). Interpretation facilitates higher quality error messages, because the interpreter has access to the full program source. It is much easier to bootstrap or port an interpreter than a compiler, so experimental language implementations are very often based on interpreters.