CS 6352 Notes transcription 9/2/2008 Nathan Leach

Language History (cont)

PL/I – Means "Programming Language One"

(Non-qualified-yet-adequate reference: http://en.wikipedia.org/wiki/PL/I)

Combines features of Fortran, Algol 60, and Cobol

Concepts:

- external procedures – procedures in a separate file for purposes of program organization
- Multitasking
- Pointers and List Processing
- Static, automatic, and controlled storage

Side note: Lisp had feature of] added to close all open left parens because lists in Lisp are defined in parentheses , EX:

(() () (()))

and it was easy to miss a paren. Emacs is an editor that assists in language syntax.

Simula 67 – intended to be used for simulations

wasn't very popular until the 1990's

Contributions:

- Introduced class concept of OO
- Assignment of classes to class-valued variables (identifiers)
- Subclass concept

Algol 68

First language designed by a committee, inspired by Algol 60. Intent was to start with Algol 60 and do a systematic generalization.

- "orthogonal" language concepts – the idea that if two features don't interfere with each other then combine them. Non-interfering features should be possible to combine whenever it makes sense.

- Introduced basic features & combining forms – powerful mechanisms for combining features
- '||' parallel programming facility

Pascal – emphasized simplicity

Designed by N. Wirth

Tries to control some problems caused by pointers

- Structured programming
- emphasized readability of programs (without as much syntactic sugar as cobol)
- strong typing – programmer must declare the type of each identifier + type checking

Type checking example:
lhs = vhs
comparison where types must be compatible types

Prolog - "Programmation Logique"

became popular in late 80's, by Alain Colmerauer and Robert Kowalski

- First language based on logical model
- Based on Horn-clause logic

Ada – language requisition by DoD to be able to combine all programming tasks into a unified language (which never happened)

- Based on Pascal
- Uses class concept of Simula in its abstract data type facility (called packages)
- Exception handling features of PL/I
- Features for concurrent programming

More than 200 pages descibe the basics of Ada. It became complex because it was composed of many languages. Complex languages don't tend to become very popular.

Description of a PL

3 key terms:

<u>Syntax</u>

concerns the form or the notation – determines when a program is well-formed

Example: semi-colon is a separator or a terminator

Separator	Terminator
A;	A;
B	B;

<u>Semantics</u>

all concerns with meaning – program can be well-formed, but what does it mean?

Example:

E1 + E2 (x+1)

What is the value? We don't know until we know the value of x. The meaning is modeled as a mathematical function. We can't say this is a simple function from values->values.

Imperative languages introduced the concept of state to handle complex functions.

ε(means semantincs of expression)

ε: states->values X states

where the resulting "states" is to handle side effects. States represent all memory and I/O devices that can be accessed by the program.

<u>Pragmatics</u>

all concerns with the origins of PL, history of Pls, users, implementation techniques, programming methodology (software engineering)

3 Systematic Approaches for PL Semantics

1. Axiomatic
2. Denotional (a.k.a. Syntax directed)
3. Operational

Axiomatic

For each construct/feature of PL we have an inference rule + some axioms.

Meanings to programs are given through reasoning with the axioms and inference rules.

Example:

construct assignment statement:

P{x := E}Q

P is precondition/antecedent – predicate that holds before
Q is postcondition/consequence – predicate that holds after

on postcondition, we want value of x = value of E before execution.

Precondition is “true” because we don't have any preconditions.

P{if E then C1 else C2}Q

REFERENCE – Ravi Sethi book, look up inference rules for:

if-then-else
while E do C

Denotional semantics/syntax directed semantics

Give the meanings to basic features of a programming language directly in terms of mathematical objects such as sets, sequences, etc. For composite features, the meaning is given in terms of the immediate constituents.

Example: Composite expression E1+E2

E1 and E2 are constituents

Simple Example:

Language of binary numerals

Syntax given by inductive definition

1. '0', '1' are binary numerals
2. If 'n' is a binary numeral then so are 'n0' and 'n1'
3. Nothing else is a binary numeral unless it follows from 1&2

Note: No meaning is expressed here, only syntax

Semantics – Morse code provides the meaning by defining the application

0 = dot
1 = dash

Syntax directed semantics approach:

Numbers as intended semantics:

1. '0' represents 0, '1' represents 1
2. If numeral 'n' represents the number N then 'n0' represents 2N and 'n1' represents 2N+1.

Side note:
BNF/EBNF – Backus-Naur form – used to specify a context-free syntax

<u>Operational</u>

Popular with Prolog and functional languages

Steps:

1. Define an abstract machine (a simple model of a computer) with its own instruction set
2. Translate programs from source lang to abstract machine instruction set
3. The meaning of the program is defined as the effect it has on the abstract machine when it is executed

Not a very satisfactory approach – when the program outputs “5” what does it mean?

Prolog abstract machine known as WAM (Warren abstract machine developed by David H.D. Warren)