CS2PLD: Programming Language Design Paradigms
Maribel Fernández

More Half-Term Revision — Exam May 2013

Answers (summary)

Question 1:

1. What are the main features of variables in imperative programming languages?

   Your answer should mention that variables in imperative languages are an abstraction of the memory and as such have an address, a value, possibly a type, lifetime and scope. Check the slides and textbook for more details.

2. The abstract syntax of SIMP is defined by:

$$\begin{array}{lll} C & ::= & skip \mid l := E \mid C; C \mid if\ B\ then\ C\ else\ C \mid while\ B\ do\ C \\ E & ::= & !l \mid n \mid E\ op\ E \\ op & ::= & + \mid - \mid * \mid / \\ B & ::= & True \mid False \mid E\ bop\ E \mid \neg B \mid B \wedge B \\ bop & ::= & > \mid < \mid = \end{array}$$

   We add to the language SIMP a multiple selector:

   `switch` E

   `case positive:` C_1

   `case zero:` C_2

   `case negative:` C_3

   where the integer expression E will be evaluated and if it is positive, then C_1, C_2 and C_3 will be executed (in that order); if E is zero then C_2 and C_3 will be executed (in that order); and if E is negative then only C_3 will be executed.

   Give a formal definition of the semantics of this command (you can give transition rules for an abstract machine, or rules to specify the small-step semantics or the big-step semantics of this command).

   We give big-step semantic rules below. There are three cases, depending on the value of the expression E, which must be evaluated first. Therefore we need three rules.

$$\frac{\langle E, s\rangle \Downarrow \langle n, s'\rangle \quad \langle C_1; C_2; C_3, s'\rangle \Downarrow \langle skip, s''\rangle}{\langle switch\ E\ case\ positive\ C_1\ case\ zero\ C_2\ case\ negative\ C_3, s\rangle \Downarrow \langle skip, s''\rangle}\ where\ n > 0$$

$$\frac{\langle E, s\rangle \Downarrow \langle n, s'\rangle \quad \langle C_2; C_3, s'\rangle \Downarrow \langle skip, s''\rangle}{\langle switch\ E\ case\ positive\ C_1\ case\ zero\ C_2\ case\ negative\ C_3, s\rangle \Downarrow \langle skip, s''\rangle}\ where\ n = 0$$

$$\frac{\langle E, s\rangle \Downarrow \langle n, s'\rangle \quad \langle C_3, s'\rangle \Downarrow \langle skip, s''\rangle}{\langle switch\ E\ case\ positive\ C_1\ case\ zero\ C_2\ case\ negative\ C_3, s\rangle \Downarrow \langle skip, s''\rangle}\ where\ n < 0$$

3. Show that the extension proposed in Question 1 part 2 does not change the computational power of the language by giving a translation of

   `switch E case positive:` C_1 `case zero:` C_2 `case negative:` C_3

   in terms of the constructs that already exist in SIMP.

   We can encode it using two if-then-else with branches corresponding to each case:

   `if E > 0 then` $C_1; C2; C_3$ `else (if E = 0 then` $C_2; C_3$ `else` C_3`)`

   It remains to prove that this command has the same semantics (i.e., it is an equivalent program). This can be done by comparing the results obtained for each program, using the semantic rules.

Question 2:
We add to the language SIMP, studied in the course, a new class of expressions that will be used to represent *strings*.
The *concrete syntax* for strings is simply a sequence of letters (possibly empty) starting with the symbol " and ending with the symbol ", as in the string "pld". The empty string is written "".
The *abstract syntax* of string expressions is defined by the grammar:

$$S ::= null \mid l \cdot S \mid S@S$$

where l is a letter. Here, *null* represents the empty string, whereas $\cdot$ and @ are binary operators. The abstract syntax tree $l \cdot S$ represents a string where the first element is the letter l and the rest of the string is defined by S. For example, the string "abc" is represented by an abstract syntax tree which we can write as $a \cdot (b \cdot (c \cdot null))$.

1. Draw the abstract syntax tree of the expression:
   ""@"pld"

   The abstract syntax tree has a root labelled by @, with two subtrees. The left one is null (empty string) and the right subtree is the tree $(p \cdot (l \cdot (d \cdot null)))$.

   Note that the grammar describes the abstract syntax of SIMP; the expression ""@"pld" is concrete syntax. The difference was discussed in the lectures and tutorials.

2. The semantics of string expressions is defined by the following axioms and rules:

$$\frac{}{\langle null, s\rangle \Downarrow \langle null, s\rangle}$$

$$\frac{\langle S, s\rangle \Downarrow \langle v, s'\rangle}{\langle l \cdot S, s\rangle \Downarrow \langle l \cdot v, s'\rangle}$$

$$\frac{\langle S_1, s\rangle \Downarrow \langle null, s'\rangle \quad \langle S_2, s'\rangle \Downarrow \langle v, s''\rangle}{\langle S_1@S_2, s\rangle \Downarrow \langle v, s''\rangle}$$

$$\frac{\langle S_1, s\rangle \Downarrow \langle l \cdot v_1, s'\rangle \quad \langle v_1@S_2, s'\rangle \Downarrow \langle v_2, s''\rangle}{\langle S_1@S_2, s\rangle \Downarrow \langle l \cdot v_2, s''\rangle}$$

Explain, in plain English, the role of the operator @.

The rules indicate that the empty string is a value, since it is not evaluated any further, and if a string is not empty, that is, it is represented by an expression $l \cdot S$, then the resulting string has the letter l as first element, followed by the value of S.

The evaluation of $S_1@S_2$ starts from S_1, and produces a string that is the concatenation of the values of S_1 and S_2.

What is the value of the expression $(a \cdot null)@(b \cdot null)$? Use the axioms and rules given above to justify your answer.

The value is "ab". It is obtained by applying the second rule for @ first, which requires to evaluate $a \cdot null$ and $null@(b \cdot null)$. The latter evaluates to $(b \cdot null)$ so the final result is $(a \cdot (b \cdot null))$.

3. Give a principle of induction that can be used to prove that a property $P(S)$ holds for all string expressions S in SIMP.

Principles of induction for integer expressions and for Boolean expressions in SIMP were discussed in lectures and tutorials. Structural induction on lists was also discussed.

The principle of induction for the new string expressions in SIMP has a base case

$P(null)$

and induction steps:

$P(S) \Rightarrow P(l \cdot S)$ for any letter l and string expression S

$P(S_1)$ and $P(S_2) \Rightarrow P(S_1@S_2)$, for any string expressions S_1, S_2.

4. Show by induction that each string expression S has a unique value according to the axioms and rules given in Question 2 part 2.

The base case is $\langle null, s\rangle \Downarrow \langle null, s\rangle$ (unique value) for any store s, which can be proved using the axiom.

The inductive step is proved using the rules for $\cdot$ and @, and the induction hypothesis (give full details to get full marks).