Revision notes - CS3203

Ma Hongqiang

August 19, 2019

Contents

1 SIMPLE

1.1 Language Syntax

SIMPLE is the target language of CS3203.
We use suffix + to denote 1 to many. We use suffix $*$ to denote 0 to many.
We further define the following lexical tokens:

- LETTER: A-Z | a-z
- DIGIT: 0-9
- NAME: LETTER(LETTER — DIGIT)$*$
- INTEGER: DIGIT+

Its **grammar** is defined as below:

- **program**: procedure+
- **procedure**: `procedure procedureName` { stmtLst }
- **stmtLst**: statement+
- **statement**: read | print| call | while | if | assign
- **read, print**(I/O): `read` *variableName*`;`, `print` *variableName*`;`
- **call**(function call): `call` *procedureName*`;`
- **while**(loop): `while (` cond_expr `)` { stmtLst }
- **cond_expr**: Can be one of the following forms. Note that cond_expr requires bracketing *always*.
  - rel_expr
  - `!(` cond_expr `)`
  - `(` cond_expr `)`
  - `(` cond_expr `) && (` cond_expr `)`
  - `(` cond_expr `) || (` cond_expr `)`
- **rel_expr**: gt | gte | lt | lte| eq | neq
- **gt, gte, lt, lte, eq, neq**(=op): rel_factor op rel_factor
- **rel_factor**: *variableName* | *constant* | expr
- **expr**: plus | minus | times | div | mod | ref. Specifically, it is of the following forms:
  - expr and term

  - expr minus term
  - term
- **term**: One of the following forms:
  - term times factor
  - term div factor
  - term mod factor
  - factor
- **factor**: *variableName* | *constant* | `(` expr `)`
- **plus, minus, times, div, mod, ref**(=op): expr op expr
- **ref**: *variableName* | *constant*
- **if**: `if (` cond_expr `) then {` stmtLst `} else {` stmtLst `}`
- **assign**: *variableName* `=` expr`;`
- **variableName**, **procedureName**: NAME
- **constant**: INTEGER

Note, we represent operators as such:

- **and, or, not**: `&&`, `||`, `!`
- **gt, gte, lt, lte, eq, neq**: `>`, `>=`, `<`, `<=`, `==`, `!=`
- **plus, minus, times, div, mod**: `+`, `-`, `*`, `/`, `%`

1.2 Programme Knowledge Base

Definition 1.1 (Program Design Entities)**.**
The following list contains all **programme design entities**:

- procedure
- stmtLst
- stmt
- read, print, assign, call, while, if
- variable
- constant

Definition 1.2 (Programme Design Abstraction)**.**
Program design abstractions are *relationships* betwwen programme design entities.

Following are 4 basic design abstractions:

Definition 1.3 (Follow)**.**
For any statement s_1, s_2,

- **Follows**(s_1, s_2) holds if they are at the same nesting level, in the same statement list, and s_2 appears *immediately* after s_1.
- **Follows***(s_1, s_2) holds *if* **Follows**(s_1, s_2) *or* **Follows**(s_1, s) and **Follows***(s, s_2) for some statement s.

Definition 1.4 (Parent)**.**
Forany statement s_1, s_2,

- **Parent**(s_1, s_2) holds if s_2 is directly nested in s_1.
- **Parent***(s_1, s_2) holds *if* **Parent**(s_1, s_2) *or* **Parent**(s_1, s) and **Parent***(s, s_2) for some statement s.