Semantics of abrupt completion: a case study in natural semantics

lecture notes for *Semantiek en Correctheid*
Radboud University Nijmegen

Erik Barendsen, version February 2010

1. Introduction

In these notes we will extend the language **While** from Nielson and Nielson (1992) with three examples of 'abrupt completion' inspired by constructions in Java (see Gosling et al. (2005), sections 14.15, 14.16, 14.18, 14.20). We will specify the semantics in ns style. This will involve extending the resulting state with termination information.

2. Break

A `break` statement causes the execution of an immediately enclosing loop to be terminated. If a `break` occurs during the execution of the body S in the loop `while` b `do` S, the rest of the body is skipped and the loop terminates. Thus a `break` can be used to 'jump out of a loop'.

The extended syntax is

$$S ::= \texttt{skip} \mid x := a \mid S_1; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \mid \texttt{while } b \texttt{ do } S \mid \texttt{break}.$$

Expressing the semantics in ns style is not completely straightforward. Indeed, the semantics of `while` b `do` S needs to be expressed in terms of the semantics of S (as a whole). The result state of the execution of S is not sufficient to determine the semantics of `while` b `do` S, since the result of the latter statement depends on the fact whether there has been a `break` during execution of S.

Our modified semantics expressions will be of the form

$$\langle S, s\rangle \to (s', B)$$

where S is a statement, s, s' are states and B is a *break status* which can be either $\circ$ ('no break has occurred') or $\bullet$ ('a break has occurred').

We first treat the most interesting statements.

$$\langle \texttt{break}, s\rangle \to (s, \bullet) \qquad \text{[break]}$$

As to the composition, we have to express that statements following a `break` are skipped.

$$\frac{\langle S_1, s\rangle \to (s', \bullet)}{\langle S_1; S_2, s\rangle \to (s', \bullet)} \qquad [\text{comp}^\bullet]$$

$$\frac{\langle S_1, s\rangle \to (s', \circ) \qquad \langle S_2, s'\rangle \to (s'', B)}{\langle S_1; S_2, s\rangle \to (s'', B)} \qquad [\text{comp}^\circ]$$

In the latter rule we use the generic metavariable B instead of formulating two separate rules for $\bullet$ and $\circ$. Moreover this formulation makes the break status dependencies more explicit.

The loop semantics will express the termination behaviour depending on the occurrence of a `break`.

$$\frac{\langle S, s\rangle \to (s', \bullet)}{\langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \to (s', \circ)} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \qquad [\text{while}^{\mathbf{tt}\bullet}]$$

$$\frac{\langle S, s\rangle \to (s', \circ) \qquad \langle \texttt{while}\ b\ \texttt{do}\ S, s'\rangle \to (s'', B)}{\langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \to (s'', \circ)} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \qquad [\text{while}^{\mathbf{tt}\circ}]$$

$$\langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \to (s, \circ) \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \qquad [\text{while}^{\mathbf{ff}}]$$

The 'break status memory' should be cleared upon completion of the loop. This is expressed in the derivation rules: the break status in the semantics of a `while`-statement is always $\circ$.

Observe that B in $[\text{while}^{\mathbf{tt}\circ}]$ will be $\circ$ in any concrete application of the rule. We use B to make it obvious that the set of rules is complete, i.e., covers all cases.

$$\langle \texttt{skip}, s\rangle \to (s, \circ) \qquad [\text{skip}]$$

$$\langle x := a, s\rangle \to (s[x \mapsto \mathcal{A}[\![a]\!]s], \circ) \qquad [\text{ass}]$$

$$\frac{\langle S_1, s\rangle \to (s', B)}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, s\rangle \to (s', B)} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \qquad [\text{if}^{\mathbf{tt}}]$$

$$\frac{\langle S_2, s\rangle \to (s', B)}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, s\rangle \to (s', B)} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \qquad [\text{if}^{\mathbf{ff}}]$$

Note that the specifications of syntax and semantics do not require that a `break` occurs only inside a loop. We have generalized the 'skipping behaviour' of `break` to general composite statements to provide an elegant description. Typically, the occurrences of `break` are restricted at a higher level description of the language.

3. Continue

When a `continue` occurs inside a loop `while` b `do` S, the body S also terminates, but instead of terminating the loop as a whole, the loop condition b is evaluated again and the loop is continued depending on the value of b. Thus, a `continue` can be used to 'jump to the beginning of a loop'.

The characterization of the semantics is left as an exercise.

4. Try-catch and throw

The exception mechanism involves terminating execution of a statement and proceeding ('jumping') to another statement depending on a specified (*thrown*) label.

The syntax is specified by

$$\begin{array}{rcl} S & ::= & \texttt{skip} \mid x := a \mid S_1; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \mid \texttt{while } b \texttt{ do } S \mid \\ & & \texttt{try } S \texttt{ catch } \ell : S' \mid \texttt{throw } \ell \end{array}$$

where ℓ ranges over a set of *labels*. We assume that we have an infinite set of labels, just as we did in the case of variables.

Multiple exceptions can be expressed by nested `try-catch` statements:

$$\texttt{try } (\texttt{try } S \texttt{ catch } \ell_1 : S_1) \texttt{ catch } \ell_2 : S_2.$$

The specification if the semantics is again left as an exercise.

References

Gosling, J., B. Joy, G. Steele and G. Bracha (2005). *The Java Language Specification, Third Edition*, Addison-Wesley, Wokingham. Available at `java.sun.com/docs/books/jls/`.

Nielson, H.R. and F. Nielson (1992). *Semantics with Applications*, Wiley, Chichester. Revised edition (1999) available at `www.daimi.au.dk/~hrn`.