Course: Operational Semantics
Second Assessed Exercise
Lecturer: Steffen van Bakel
Hand in date: March 26, 2002

1) The following is the abstract syntax of a Snail control language:

$$\begin{array}{lcl} p & \in & \mathrm{Program} \\ n & \in & \mathrm{Numeral} \\ a & \in & \mathrm{Arithmetic-expression} \\ p & ::= & \mathbf{up} \mid \mathbf{down} \mid \mathbf{move}(a_1, a_2) \mid p_1;p_2 \\ a & ::= & n \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \end{array}$$

The Snail navigates its way around a two dimensional space. It has a pen which may be up or down; in the latter case it leaves an ink trace of its movements. It can move from its current position to the relative co-ordinates indicated by the vector (a_1, a_2).

a) Define the Denotational Semantics of arithmetic expressions.

Answer:

$$\begin{array}{rcl} \mathcal{A}[\![n]\!] & = & \mathcal{N}[\![n]\!], \\ \mathcal{A}[\![a_1 + a_2]\!] & = & \mathcal{A}[\![a_1]\!] + \mathcal{A}[\![a_2]\!], \\ \mathcal{A}[\![a_1 - a_2]\!] & = & \mathcal{A}[\![a_1]\!] - \mathcal{A}[\![a_2]\!], \\ \mathcal{A}[\![a_1 \times a_2]\!] & = & \mathcal{A}[\![a_1]\!] \times \mathcal{A}[\![a_2]\!] \end{array}$$

---

b) The state of the Snail is represented by a triple: *(x,y,pen)*. The first two elements give the Snail's current position (cartesian coordinates); the third element is a boolean indicating whether the pen is up or down. Use your answer to (a) to define the Natural Semantics of programs.

Answer:

$$(\mathrm{COMP}_{ns})\quad \frac{\langle S_1, s_1\rangle \rightarrow s_2 \quad \langle S_2, s_2\rangle \rightarrow s_3}{\langle S_1\,;\,S_2, s_1\rangle \rightarrow s_3}$$

$$(\mathrm{UP}_{ns})\quad \frac{}{\langle \mathbf{up}, \langle x, y, pen\rangle\rangle \rightarrow \langle x, y, \mathbf{up}\rangle}$$

$$(\mathrm{DOWN}_{ns})\quad \frac{}{\langle \mathbf{down}, \langle x, y, pen\rangle\rangle \rightarrow \langle x, y, \mathbf{down}\rangle}$$

$$(\mathrm{MOVE}_{ns})\quad \frac{}{\langle \mathbf{move}(a_1, a_2), (x, y, pen)\rangle \rightarrow (x + \mathcal{A}[\![a_1]\!], y + \mathcal{A}[\![a_2]\!], pen)}$$

---

c) The Snail has configurations $\langle c, e, s\rangle \in$ *Code* $\times$ *Stack* $\times$ *State*, where:

$$\begin{array}{lcl} c & \in & \mathit{Code} \\ i & \in & \mathit{Instruction} \\ c & ::= & \varepsilon \mid i : c \\ i & ::= & \mathbf{PUSH}{-}n \mid \mathbf{ADD} \mid \mathbf{SUB} \mid \mathbf{MULT} \mid \mathbf{UP} \mid \mathbf{DOWN} \mid \mathbf{MOVE} \end{array}$$

Define an operational semantics for the Snail.

Answer:

$$
\begin{array}{rcl}
\langle \texttt{PUSH}{-}n, e, s\rangle & \rhd & \langle \varepsilon, \mathcal{A}[\![n]\!] : e, s\rangle \\
\langle \texttt{ADD}, n_1 : n_2 : e, s\rangle & \rhd & \langle \varepsilon, n_1 + n_2 : e, s\rangle \\
\langle \texttt{SUB}, n_1 : n_2 : e, s\rangle & \rhd & \langle \varepsilon, n_1 - n_2 : e, s\rangle \\
\langle \texttt{MULT}, n_1 : n_2 : e, s\rangle & \rhd & \langle \varepsilon, n_1 \times n_2 : e, s\rangle \\
\langle \texttt{UP}, e, (x, y, pen)\rangle & \rhd & \langle \varepsilon, e, (x, y, \texttt{up})\rangle \\
\langle \texttt{DOWN}, e, (x, y, pen)\rangle & \rhd & \langle \varepsilon, e, (x, y, \texttt{down})\rangle \\
\langle \texttt{MOVE}, n_1 : n_2 : e, (x, y, pen)\rangle & \rhd & \langle \varepsilon, e, (x + n_1, y + n_2, pen)\rangle
\end{array}
$$

---

d) Define suitable translation functions to translate control programs into Snail code.

Answer:

$$
\begin{array}{rcl}
\mathcal{CA}[\![n]\!] & = & \texttt{PUSH}{-}n \\
\mathcal{CA}[\![a_1 + a_2]\!] & = & \mathcal{CA}[\![a_2]\!] : \mathcal{CA}[\![a_1]\!] : \texttt{ADD} \\
\mathcal{CA}[\![a_1 - a_2]\!] & = & \mathcal{CA}[\![a_2]\!] : \mathcal{CA}[\![a_1]\!] : \texttt{SUB} \\
\mathcal{CA}[\![a_1 \times a_2]\!] & = & \mathcal{CA}[\![a_2]\!] : \mathcal{CA}[\![a_1]\!] : \texttt{MULT} \\
\mathcal{CS}[\![\texttt{up}]\!] & = & \texttt{UP} \\
\mathcal{CS}[\![\texttt{down}]\!] & = & \texttt{DOWN} \\
\mathcal{CS}[\![\texttt{move}(a_1, a_2)]\!] & = & \mathcal{CA}[\![a_2]\!] : \mathcal{CA}[\![a_1]\!] : \texttt{MOVE} \\
\mathcal{CS}[\![S_1; S_2]\!] & = & \mathcal{CA}[\![S_1]\!] : \mathcal{CA}[\![S_2]\!]
\end{array}
$$

---

e) Assuming that:

$$\text{if } \langle c_1, e_1, s\rangle \rhd^k \langle c', e', s'\rangle \text{ then } \langle c_1 : c_2, e_1 : e_2, s\rangle \rhd^k \langle c' : c_2, e' : e_2, s'\rangle,$$

and that the translation function for arithmetic expressions is correct, show that the translation function for programs is correct.

Answer: To prove: if $\langle S, s\rangle \to s'$, then $\langle \mathcal{CS}[\![S]\!], \varepsilon, s\rangle \rhd^* \langle \varepsilon, \varepsilon, s'\rangle$. By induction on the structure of derivations.

1) $\langle S_1; S_2, s_1\rangle \to s_3$ because $\langle S_1, s_1\rangle \to s_2$ and $\langle S_2, s_2\rangle \to s_3$. Then by induction, $\langle \mathcal{CS}[\![S_1]\!], \varepsilon, s_1\rangle \rhd^* \langle \varepsilon, \varepsilon, s_2\rangle$ and $\langle \mathcal{CS}[\![S_2]\!], \varepsilon, s_2\rangle \rhd^* \langle \varepsilon, \varepsilon, s_3\rangle$. Using the first assumption, we get $\langle \mathcal{CS}[\![S_1]\!] : \mathcal{CS}[\![S_2]\!], \varepsilon, s_1\rangle \rhd^* \langle S_2, \varepsilon, s_2\rangle \rhd^* \langle \varepsilon, \varepsilon, s_3\rangle$. Since $\mathcal{CS}[\![S_1; S_2]\!] = \mathcal{CA}[\![S_1]\!] : \mathcal{CA}[\![S_2]\!]$, we obtain $\langle \mathcal{CS}[\![S_1; S_2]\!], \varepsilon, s_1\rangle \rhd^* \langle \varepsilon, \varepsilon, s_3\rangle$.

2) The cases for **up** and **down** are easy.

3) $\langle \texttt{move}(a_1, a_2), (x, y, pen)\rangle \to (x + \mathcal{A}[\![a_1]\!], y + \mathcal{A}[\![a_2]\!], pen)$.

$$
\begin{array}{rcl}
\langle \mathcal{CS}[\![\texttt{move}(a_1, a_2)]\!], \varepsilon, (x, y, pen)\rangle & = & \langle \mathcal{CA}[\![a_2]\!] : \mathcal{CA}[\![a_1]\!] : \texttt{MOVE}, \varepsilon, (x, y, pen)\rangle \\
& \rhd^* & \langle \mathcal{CA}[\![a_1]\!] : \texttt{MOVE}, \mathcal{A}[\![a_2]\!], (x, y, pen)\rangle \\
& \rhd^* & \langle \texttt{MOVE}, \mathcal{A}[\![a_1]\!], \mathcal{A}[\![a_2]\!], (x, y, pen)\rangle \\
& \rhd & \langle \varepsilon, \varepsilon, (x + \mathcal{A}[\![a_1]\!], y + \mathcal{A}[\![a_2]\!], pen)\rangle
\end{array}
$$

---

2) The abstract syntax for the language `While` is given by:

$$
\begin{array}{lcl}
x & \in & \text{Variable} \\
a & \in & \text{Arithmetic expression} \\
b & \in & \text{Boolean expression} \\
S & \in & \text{Statement} \\
S & ::= & x := a \mid S_1;S_2 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \mid \textbf{skip} \mid \textbf{while } b \textbf{ do } S
\end{array}
$$

The syntax of expressions (both arithmetic and boolean) is unspecified.

a) Extend the syntax of `While` with a **for** statement:

$$\textbf{for } x := a_1 \textbf{ to } a_2 \textbf{ do } S$$

The intended meaning of this construct is x gets assigned a_1, and that S is repeatedly excecuted, incrementing x, until x reaches a_2.

Write down both a Natural Semantics and a SOS-style semantics for the new construct; if necessary, you can assume the existence of an 'inverse' $\mathcal{N}^{-1}()$ of the function $\mathcal{S}_{ns}[\![]\!]$, such that $\mathcal{N}^{-1}(\mathcal{S}_{ns}[\![n]\!]) = n$. (The semantics of the **for** -loop should not depend on the semantics of the **while** -loop.)

Answer:

$$(\text{FOR-TO}^{\text{T}}_{ns}) \quad \frac{\langle S, s_1\rangle \to s_2 \quad \langle \textbf{for } x := n_1 \textbf{ to } n_2 \textbf{ do } S, s_2\rangle \to s_3}{\langle \textbf{for } x := a_1 \textbf{ to } a_2 \textbf{ do } S, s_1\rangle \to s_3} \; \begin{array}{l} \mathcal{A}[\![a_1]\!]s_1 \leq \mathcal{A}[\![a_2]\!]s_1, \\ (n_1 = \mathcal{N}^{-1}(\mathcal{A}[\![a_1]\!]s_1 + 1), \;) \\ n_2 = \mathcal{N}^{-1}(\mathcal{A}[\![a_2]\!]s_1) \end{array}$$

$$(\text{FOR-TO}^{\text{F}}_{ns}) \quad \frac{}{\langle \textbf{for } x := a_1 \textbf{ to } a_2 \textbf{ do } S, s\rangle \to s} \; (\mathcal{A}[\![a_1]\!]s > \mathcal{A}[\![a_2]\!]s)$$

$$(\text{FOR-TO}^{\text{F}}_{ns}) \quad \frac{}{\langle \textbf{for } x := a_1 \textbf{ to } a_2 \textbf{ do } S, s\rangle \Rightarrow \langle S\,\textbf{;}\,\textbf{for } x := n_1 \textbf{ to } n_2 \textbf{ do } S, s\rangle} \; \begin{array}{l} \mathcal{A}[\![a_1 \leq a_2]\!]s_1, \\ (n_1 = \mathcal{N}^{-1}(\mathcal{A}[\![a_1]\!]s_1 + 1), \qquad) \\ n_2 = \mathcal{N}^{-1}(\mathcal{A}[\![a_2]\!]s_1) \end{array}$$

$$(\text{FOR-TO}^{\text{T}}_{ns}) \quad \frac{}{\langle \textbf{for } x := a_1 \textbf{ to } a_2 \textbf{ do } S, s\rangle \Rightarrow s} \; (\mathcal{A}[\![a_1 > a_2]\!]s)$$

---

b) Extend the syntax of `While` with a **contif** statement:

$$\textbf{loop } S_1 \textbf{ contif } b\,\textbf{;}\,S_2 \textbf{ endl}$$

The intended meaning of this construct is that S_1 and S_2 are executed repeatedly, until, after execution of S_1, b has become false; in that case the loop is exited. Extend the Natural Semantics of `While` to cover this extension.

Answer:

$$(\text{CONTIF}^{\text{F}}_{ns}) \quad \frac{\langle S_1, s_1\rangle \to s_2}{\langle \textbf{loop } S_1 \textbf{ contif } b\,\textbf{;}\,S_2 \textbf{ endl}, s_1\rangle \to s_2} \; (\mathcal{B}[\![b]\!]s_2 = \textbf{ff})$$

$$(\text{CONTIF}^{\text{T}}_{ns}) \quad \frac{\langle S_1, s_1\rangle \to s_2 \quad \langle S_2, s_2\rangle \to s_3 \quad \langle \textbf{loop } S_1 \textbf{ contif } b\,\textbf{;}\,S_2 \textbf{ endl}, s_3\rangle \to s_4}{\langle \textbf{loop } S_1 \textbf{ contif } b\,\textbf{;}\,S_2 \textbf{ endl}, s_1\rangle \to s_4} \; (\mathcal{B}[\![b]\!]s_2 = \textbf{tt})$$

---

c) Show that ‘**while** b **do** S’ is semantically equivalent to ‘**loop skip contif** b **;** S **endl**’.

Answer: By induction to the structure of derivations. Base case:

$$\frac{\dfrac{}{\langle \textbf{skip}, s\rangle \to s}}{\langle \textbf{loop skip contif}\ b\,\textbf{;}\ S\ \textbf{endl}, s\rangle \to s}\ (\mathcal{B}\,[\![b]\!]\,s = \mathbf{ff})$$

$$\frac{}{\langle \textbf{while}\ b\ \textbf{do}\ S, s\rangle \to s}\ (\mathcal{B}\,[\![b]\!]\,s = \mathbf{ff})$$

Inductive case: Assume

$$\frac{\dfrac{}{\langle \textbf{skip}, s_1\rangle \to s_1} \quad \begin{array}{c} D_1 \\ \langle S, s_1\rangle \to s_2 \end{array} \quad \begin{array}{c} D_2 \\ \langle \textbf{loop skip ; contif}\ b\,\textbf{;}\ S_2\ \textbf{endl}, s_2\rangle \to s_3 \end{array}}{\langle \textbf{loop skip ; contif}\ b\,\textbf{;}\ S\ \textbf{endl}, s_1\rangle \to s_3}\ (\mathcal{B}\,[\![b]\!]\,s_2 = \mathbf{tt})$$

Then, by induction, there exists a derivation

$$\begin{array}{c} D_2' \\ \langle \textbf{while}\ b\ \textbf{do}\ S, s_2\rangle \to s_3 \end{array}$$

Then also

$$\frac{\begin{array}{c} D_1 \\ \langle S, s_1\rangle \to s_2 \end{array} \quad \begin{array}{c} D_2' \\ \langle \textbf{while}\ b\ \textbf{do}\ S, s_2\rangle \to s_3 \end{array}}{\langle \textbf{while}\ b\ \textbf{do}\ S, s_1\rangle \to s_3}\ (\mathcal{B}\,[\![b]\!]\,s = \mathbf{tt})$$

The proof in the opposite direction is similar.

---

d) Extend the syntax of Exif-Loop with a **repeat** statement:

$$\textbf{repeat}\ S\ a\ \textbf{times}$$

The intended meaning is that the statement S is executed the number of times specified by the arithmetic expression a. Write down both a Natural Semantics and a SOS-style semantics for the new construct.

Answer:

$$(\textsc{rep-times}^{\mathrm{T}}_{ns})\quad \frac{\langle S, s_1\rangle \to s_2 \quad \langle \textbf{repeat}\ S (a-1) \textbf{times}, s_2\rangle \to s_3}{\langle \textbf{repeat}\ S\ a\ \textbf{times}, s_1\rangle \to s_3}\ (\mathcal{B}\,[\![a > 0]\!]\,s_1 = \mathbf{tt})$$

$$(\textsc{rep-times}^{\mathrm{F}}_{ns})\quad \frac{}{\langle \textbf{repeat}\ S\ a\ \textbf{times}, s\rangle \to s}\ (\mathcal{B}\,[\![a > 0]\!]\,s = \mathbf{ff})$$

$$(\textsc{rep-times}_{sos})\quad \frac{}{\langle \textbf{repeat}\ S\ a\ \textbf{times}, s\rangle \Rightarrow \langle \textbf{if}\ a > 0\ \textbf{then}\ (S\,\textbf{;}\,\textbf{repeat}\ S\ (a-1)\ \textbf{times})\ \textbf{else skip}, s\rangle}$$
