UPMC
Master informatique 2 – STL
NI503 – Conception de langages
Examen 1

Janvier 2012

Exercice I.

Question (I.1) Modifiez la sémantique donnée en annexe pour obtenir la trace du déplacement de la tortue.
Réponse

La fonction **P** d'interprétation des programmes doit produire une suite de couples de coordonnées (on peut les prendre entières ou réelle, ici ce n'est pas l'important). La fonction d'interprétation des commades dot également produire une suite de couples.

$$\begin{array}{lcl} \mathbf{P} & : & \text{Prog} \to (I\!R^2)^*_\perp \\ \mathbf{C} & : & \text{cmd} \to Env \to Mem \to Cont \to (I\!R^2)^*_\perp \end{array}$$

Comme le résultat d'un programme est celui de l'application d'une continuation (voir les commandes `MOVE` et `TURN`), les continuations doivent produire le résultat escompté : une suite de couples.

$$Cont \quad = \quad Mem \to (I\!R^2)^*$$

La continuation initiale est la fonction qui donne la liste vide $[(0,0)]$, position initiale de la tortue

$$K_0 \quad = \quad \lambda m.[(0,0)]$$

La seule équation à modifier est alors celle de `MOVE` :

$$\begin{array}{lcl} \mathbf{C}[[\texttt{MOVE } e]]\rho\ \mu\ \kappa & = & ((getM\ \mu'\ a_2), (getM\ \mu'\ a_3)) :: (\kappa\ \mu') \\ & & \text{avec } \mu' = (setM\ (setM\ \mu\ a_2\ (x + k\sin(\alpha)))\ a_3\ (y + k\cos(\alpha)))) \\ & & \text{et } \alpha = (getM\ \mu\ a_1), x = (getM\ \mu\ a_2), y = (getM\ \mu\ a_3) \\ & & \text{et } k = \mathbf{E}[[e]]\rho\ \mu \end{array}$$

Question (I.2) Ajoutez un crayon de couleur à la tortue qui pourra être levé ou baissé et dont la couleur pourra changer. La trace obtenue par l'exécution d'un programme devra aussi mentionner la couleur.
Réponse

On ajoute à la syntaxe de quoi manier le crayon : une commande `SETCOLOR` peut suffire ici (une pseudo-couleur vaudra pour «lever le crayon», toute autre, pour «baisser le crayon»)

$$\begin{array}{lcl} \text{cmd} & ::= & \ldots \\ & | & \texttt{SETCOLOR Num} \end{array}$$

Pour la sémantique, on peut se donner un domaine des couleurs, par exemple, un intervale d'entiers :

$$Coul \quad = \quad [0 \ldots 255]$$

On ajoute une nouvelle variable d'état, c'est-à-dire une case mémoire pour l'état du crayon. La pseudo-couleur sera la valeur -1. La mémoire initiales est définie de la manière suivante :

$$
\begin{array}{lcl}
\mu_0 & = & \emptyset \\
\mu_1 & = & (setM\ \mu'_1\ a_1\ \frac{\pi}{2}) \\
 & & \text{avec } a_1, \mu'_1 = (newM\ \mu_0) \\
\mu_2 & = & (setM\ \mu'_2\ a_2\ 0) \\
 & & \text{avec } a_2, \mu'_2 = (newM\ \mu_1) \quad \text{On pose : } M_0 = \mu_4. \\
\mu_3 & = & (setM\ \mu'_3\ a_3\ 0) \\
 & & \text{avec } a_3, \mu'_3 = (newM\ \mu_2) \\
\mu_4 & = & (setM\ \mu'_4\ a_4\ -1) \\
 & & \text{avec } a_4, \mu'_4 = (newM\ \mu_2)
\end{array}
$$

La valeur de la trace change. Elle devient une suite de triplets, d'où

$$
\begin{array}{lcl}
\mathbf{P} & : & \textsc{Prog} \to (I\!R^3)^*_\perp \\
\mathbf{C} & : & \textsc{cmd} \to Env \to Mem \to Cont \to (I\!R^3)^*_\perp
\end{array}
$$

et

$$
\begin{array}{lcl}
Cont & = & Mem \to (I\!R^3)^* \\
Cont & = & Mem \to (I\!R^2)^*
\end{array}
$$

La continuation initiale est la fonction qui donne la liste vide $[(0,0,-1)]$, position initiale de la tortue

$$K_0 \quad = \quad \lambda m.[(0,0,-1)]$$

L'évaluation de la commande `SETCOLOR` est

$$\mathbf{C}[[\texttt{SETCOLOR}\ c]]\rho\ \mu\ \kappa \quad = \quad (\kappa\ (setM\ \mu\ a_4\ c))$$

Enfin, l'évaluation de la commande `MOVE` doit tenir compte de l'état du crayon.

$$
\begin{array}{lcl}
\mathbf{C}[[\texttt{MOVE}\ e]]\rho\ \mu\ \kappa & = & \mathsf{case}\ (getM\ \mu\ a_4) : \\
 & & \quad -1 \to (\kappa\ \mu') \\
 & & \mid 0..255 \to ((getM\ \mu'\ a_2), (getM\ \mu'\ a_3), (getM\ \mu'\ a_4)) :: (\kappa\ \mu') \\
 & & \mid \_ \to \perp \\
 & & \text{avec } \mu' = (setM\ (setM\ \mu\ a_2\ (x + k\sin(\alpha)))\ a_3\ (y + k\cos(\alpha)))) \\
 & & \text{ et } \alpha = (getM\ \mu\ a_1), x = (getM\ \mu\ a_2), y = (getM\ \mu\ a_3) \\
 & & \text{ et } k = \mathbf{E}[[e]]\rho\ \mu
\end{array}
$$

Exercice II.

Question (II.1) Modifiez la syntaxe et la sémantique données en annexe pour que la commande `TRY CATCH` puisse capturer plusieurs exceptions.

Réponse

Il s'agit d'associer au `CATCH` une suite de couples (identificateur, traitement).

On se donne un nouveau mot clé `OR` pour séparer les éléments d'un `CATCH`. On reformule la règle syntaxique du `TRY CATCH` de la façon suivante :

```
CMD      ::=  ...
           |  TRY PROG CATCH CATCHES
CATCHES  ::=  Ident PROG
           |  Ident PROG OR CATCHES
```

Pour la sémantique, il faut étendre l'environnement avec autant de captures de continuation qu'il y a d'exceptions à traiter. Informellement, on peut écrire :

$$
\begin{array}{lcl}
\mathbf{C}[[\texttt{TRY}\ ss\ \texttt{CATCH}\ x_1\ ss_1\ \ldots x_n\ ss_n]]\rho\ \mu\ \kappa & = & \mathbf{Ss}[[ss]]\rho'\ \mu\ \kappa \\
 & & \text{avec } \rho' = \rho[x_1 := inC(\kappa'_1); \ldots; x_n := inC(\kappa'_n)] \\
 & & \text{ et } \kappa'_i = \lambda m.(\mathbf{Ss}[[ss_i]]\rho\ m\ \kappa) \\
 & & \text{ pour chaque } i \in [1 \ldots n]
\end{array}
$$

Plus formellement, on peut définir deux fonctions pour étendre l'environnement

$$\begin{array}{lcl} \mathbf{X} & = & (\mathsf{Ident} \times \text{PROG}) \to Env \to Cont \to Env \\ \mathbf{Xs} & = & \text{CATCHES} \to Env \to Cont \to Env \end{array}$$

avec les équations

$$\begin{array}{lcl} \mathbf{X}[[x\ ss]]\rho\ \kappa & = & \rho[x := inC(\kappa')] \\ & & \text{avec } \kappa' = \lambda m.(\mathbf{Ss}[[ss]]\rho\ m\ \kappa) \\ \mathbf{Xs}[[x\ ss\ \texttt{OR}\ xss]]\rho\ \kappa & = & \mathbf{Xs}[[xss]](\mathbf{X}[[x\ ss]]\rho\ \kappa)\ \kappa \\ \mathbf{C}[[\texttt{TRY}\ ss\ \texttt{CATCH}\ xss]]\rho\ \mu\ \kappa & = & \mathbf{Ss}[[ss]](\mathbf{Xs}[[xss]]\rho\ \kappa)\ \mu\ \kappa \end{array}$$

Exercice III.

On veut ajouter au langage une boucle générique `LOOP` dont on sort à l'aide d'une instructiom `EXIT WHEN` qui donne la condition de sortie. Cette instruction de sortie peut être utilisée à n'importe quel endroit de la séquence exécutée par la boucle. Par exemple :

```
[
  VAR x;
  VAR y;
  x := 10;
  y := 0;
  LOOP
    [
      y := (add y (mul 2 x));
      EXIT WHEN (lt 100 y);
      x := (sub x 1);
    ]
]
```

Question (III.1) Donnez la sémantique de ces deux nouvelles commandes.

Réponse

La boucle `LOOP` est une boucle sans fin (un ≪goto≫ inconditionnel). Pour en sortir avec l'instruction `EXIT WHEN` (≪goto≫ conditionnel) on capture la continuation courrante en entrée de boucle et on la récupère sur le `EXIT` lorsque la condition est vérifiée. Les équations correspondantes sont :

$$\begin{array}{lcl} \mathbf{C}[[\texttt{LOOP}\ ss]]\rho\ \mu\ \kappa & = & ((!k.\lambda m.(\mathbf{Ss}[[ss]]\rho'\ m\ k))\ \mu) \\ & & \text{avec } \rho' = \rho[*exit* := inC(\kappa)] \\ \mathbf{C}[[\texttt{EXIT WHEN}\ e]]\rho\ \mu\ \kappa & = & \mathsf{case}\ (\mathbf{E}[[e]]\rho\ \mu) : \\ & & \quad 0 \to (\kappa\ \mu) \\ & & \quad \mid \_ \to (\mathsf{case}\ (\rho\ *exit*) : \\ & & \qquad\qquad inC(\kappa') \to (\kappa'\ \mu) \\ & & \qquad\qquad \mid \_ \to \bot) \end{array}$$
