Master mention informatique
Spécialité STL

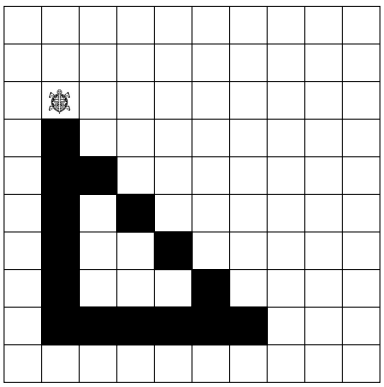
MI030 – Analyse des programmes et sémantique (APS)
Premier examen répartie
Lundi 22 mars 2010, 13h30 – 15h30

Directives

1. Le contrôle dure 2h00.
2. Le total des points des questions est de 30, mais la note obtenue sera ramenée sur 20 (soit n la note sur 30, la note de l'examen sera $e = si\ \mathrm{n} > 20\ alors\ 20\ sinon\ n$.
3. Tous les documents sont autorisés.
4. Tous les appareils électroniques sont **prohibés** (y compris les téléphones portables, les assistants numériques personnels et les agendas électroniques).

Le langage Logo et sa tortue

Le langage Logo a rendu populaire sa fameurs tortue, se déplaçant sur une grille de pixels avec son crayon levé ou baissé, de manière à tracer par ses déplacements depuis des formes géométriques simples jusqu'à des « œuvres » d'un art numérique qu'on pourrait qualifier de naïf..., par exemple :



S'agissant d'apprendre aux enfants les rudiments de la programmation impérative, les ordres que ces derniers pouvaient donner à leur tortue peut se résumer en un langage dont une syntaxe abstraite simplifiée serait :

$$
\begin{array}{lcl}
i & \in & \textit{Instructions} \\
d & \in & \textit{Directions} \\
c & \in & \textit{Conditions} \\
n & \in & \textit{Numéraux} \\
i & ::= & \texttt{lever} \mid \texttt{baisser} \mid \texttt{tourner}\ d \mid \texttt{avancer}\ n \mid i\ \texttt{;}\ i \mid \texttt{répéter}\ n\ i \mid \texttt{si}\ c\ i \\
d & ::= & \texttt{droite} \mid \texttt{gauche} \\
c & ::= & \texttt{non}\ c \mid \texttt{murDevant} \mid \texttt{murAGauche} \mid \texttt{murADroite}
\end{array}
$$

Avec cette syntaxe abstraite, on peut construire un programe dont la sémantique produirait la figure précédente, en supposant qu'initialement la tortue se trouve sur le carré en bas à gauche et qu'elle « regarde » vers le haut) :

```
avancer 1 ; tourner droite ; baisser ; avancer 6 ; tourner gauche ;
répéter  5
  (lever ; tourner gauche ; avancer 1 ; baisser ; avance 1 ; tourner droite) ;
répéter 2 (tourner gauche) ;
avancer 6 ; lever ;
répéter 2 (tourner droite) ;
avancer 7
```

Notez que lorsque le crayon de la tortue est baissé, elle noircit le carré d'arrivée lors de son déplacement, et pas celui du départ.

La sémantique informelle des instructions est :

`lever`	crayon levé ; les déplacements de la tortue ne laissent plus de trace.
`baisser`	crayon baissé ; les déplacements de la tortue laissent une trace.
`tourner` d	la tortue tourne dans la direction indiquée.
`avancer` n	la tortue avance de n pas dans la direction courante.
`répéter` n i	la tortue répète n fois i.
`si` c i	si c est vraie, la tortue exécute i.
`droite`	90° à droite.
`gauche`	90° à gauche.
`murDevant`	la tortue est face à un mur.
`murAGauche`	la tortue a un mur à sa gauche immédiate.
`murADroite`	la tortue a un mur à sa droite immédiate.
`non` c	la négation de la condition c.

Question 1. Définir les éléments de l'état du programme et la signature des relations $\rightarrow$ qu'il faut pour donner la sémantique de ce langage. Précisez bien la nature (les types) de chaque élément. (6 points)

Note : Vous pouvez supposer qu'il existe une relation *Numéraux* $\rightarrow \mathbb{N}$ donnant l'entier correspondant à n.

Question 2. Définir la sémantique opérationnelle structurelle pour ce langage. (12 points)

Question 3. Écrire un programme Prolog implantant la sémantique opérationnelle structurelle donnée à la question précédente. *Attention, il ne s'agit pas d'écrire n'importe quel programme Prolog qui fonctionne, mais bien celui qui correspond (ou correspondrait) à des règles en sémantique opérationnelle structurelle.* (12 points)

Note : Vous pouvez supposer que les numéraux sont directement représentés par des entiers Prolog. Vous pouvez également utiliser une représentation de la grille sous forme de liste telle que [(1, 1, noir), (2, 3, noir)], indiquant que les cases (1, 1) et (2, 3) sont noires. Alors, vous pouvez utiliser les deux prédicats suivants pour « noircir » des cases en avançant dans la direction des axes i et j respectivement :

```
noircirEnI(I, I, _, G, G).
noircirEnI(I, NewI, J, G, NewG) :-
  I < NewI, IP1 is I + 1,
  noircirEnI(IP1, NewI, J, [(IP1, J, noir) | G], NewG).

noircirEnJ(_, J, J, G, G).
noircirEnJ(I, J, NewJ, G, NewG) :-
  J < NewJ, JP1 is J + 1,
  noircirEnJ(I, JP1, NewJ, [(I, JP1, noir)|G], NewG).
```

Fin du contrôle.