TP Noté 22/23

L'objectif du TP est de réaliser un interprète pouvant exécuter des programmes ayant la forme suivante :

```
from (X = 123, Y = 456, D = North)

repeat 10 times (
  step,
  step 10,
  turn left,
  step 20,
  turn right,
  with sub =
    step 2,
    turn right
  do
    when (X = 10 and Y = 20) or not (D = North) do
      sub
    done
  done
done)
```

Ce langage permet de spécifier des trajectoires de déplacement dans une grille à deux dimensions. L'état courant est donné par un triplet `(X,Y,D)` où `X` et `Y` sont les coordonnées de la position courante et `D` est la direction vers laquelle on regarde. Les déplacements de base sont simplement d'avancer (`step`), de pivoter à gauche (`turn left`) et de pivoter à droite (`turn right`).

Le langage permet également de séquencer plusieurs déplacements (opérateur virgule), de répéter des déplacements (`repeat … times … done`), de nommer des déplacements (`with … = … do … done`), et de conditionner des déplacements (`when … do … done`).

Les sections suivantes décrivent l'élaboration de l'interprète par étape. Le barème est donné à titre indicatif.

Séquence de déplacements de base (7/20)

Le premier objectif est de permettre l'interprétation d'une suite de déplacements de base. Par exemple,

```
from (X = 0, Y = 0, D = East)
step,
turn left,
step,
turn right,
step,
turn left,
step
```

Ce code permet d'atteindre l'état `(2, 2, North)`.

La grammaire est la suivante :

$$
\begin{array}{lcl}
P & ::= & \underline{\texttt{from}}\ \underline{(}\ \underline{\texttt{X}}\ \underline{=}\ \underline{n}\ \underline{,}\ \underline{\texttt{Y}}\ \underline{=}\ \underline{n}\ \underline{,}\ \underline{\texttt{D}}\ \underline{=}\ D\ \underline{)}\ I \\
D & ::= & \underline{\texttt{North}} \mid \underline{\texttt{South}} \mid \underline{\texttt{East}} \mid \underline{\texttt{West}} \\
I & ::= & \underline{\texttt{step}} \\
 & \mid & \underline{\texttt{turn}}\ \underline{\texttt{left}} \\
 & \mid & \underline{\texttt{turn}}\ \underline{\texttt{right}} \\
 & \mid & I\ \underline{,}\ I \\
 & \mid & \underline{(}\ I\ \underline{)}
\end{array}
$$

Dans cette grammaire, $\underline{n}$ représente une unité lexicale correspondant à un nombre entier positif écrit en décimal. L'opérateur de séquencement (la virgule) est associatif.

L'exécution d'un programme sera formalisée à travers deux jugements :

- $P \Rightarrow (x, y, d)$ qui se lit : *L'exécution du programme P amène à la position (x, y, d)*
- $(x, y, d) \vdash I \rightarrow (x', y', d')$ qui se lit : *A partir de la position (x, y, d), l'exécution de l'instruction I amène à la position (x', y', d')*

La définition de ces jugements est donnée par les règles d'inférence suivantes :

$$\frac{(x, y, d) \vdash I \rightarrow (x', y', d')}{\texttt{from (X = } x\texttt{, Y = } y\texttt{, D = } d\texttt{) } I \Rightarrow (x', y', d')} \qquad \frac{}{(x, y, d) \vdash \texttt{step} \rightarrow \texttt{move } (x, y, d)\ \texttt{Step}}$$

$$(x, y, d) \vdash \texttt{turn left} \rightarrow \texttt{move}\ (x, y, d)\ \texttt{TurnLeft} \qquad (x, y, d) \vdash \texttt{turn right} \rightarrow \texttt{move}\ (x, y, d)\ \texttt{TurnRight}$$

$$\frac{(x, y, d) \vdash I_1 \rightarrow (x', y', d') \qquad (x', y', d') \vdash I_2 \rightarrow (x'', y'', d'')}{(x, y, d) \vdash I_1\ ,\ I_2 \rightarrow (x'', y'', d'')}$$

La fonction `move` est directement fournie dans le fichier `ast.ml` (cf. la section suivante).

Réalisations

Vous trouverez dans l'archive tpnote.zip le squelette de l'interprète que vous devez compléter. Ce code est déjà très complet et seuls quelques points de suspension **…** accompagnés d'un commentaire `TODO: A compléter` sont à remplir. Seuls les fichiers `ast.ml`, `parser.mly` et `lexer.mll` sont à compléter.

Une fois compléter le code exemple ci-dessus devrait produire la sortie suivante :

```
from (0, 0, East)
step,
turn left,
step,
turn right,
step,
turn left,
step
    => (2, 2, North)
```

Répétition de déplacements (10/20)

La prochaine étape consiste à ajouter la construction permettant de répéter un certain nombre de fois des déplacements :

```
from (X = 0, Y = 0, D = East)
repeat 4 times
  step,
  turn left,
  repeat 5 times
    step
  done,
  turn right,
  step,
  turn left,
  step
done
```

Le non-terminal I de la grammaire est étendu avec la construction suivante :

$$\begin{array}{rcl} I & ::= & \ldots \\ & | & \underline{\texttt{repeat}}\ \underline{n}\ \underline{\texttt{times}}\ I\ \underline{\texttt{done}} \end{array}$$

Les règles sémantiques à implémenter sont les suivantes :

$$\frac{}{(x, y, d) \vdash \texttt{repeat}\ 0\ \texttt{times}\ I\ \texttt{done} \rightarrow (x, y, d)}$$

$$\frac{(x, y, d) \vdash I \rightarrow (x', y', d') \qquad (x', y', d') \vdash \texttt{repeat}\ (n-1)\ \texttt{times}\ I\ \texttt{done} \rightarrow (x'', y'', d'')}{(x, y, d) \vdash \texttt{repeat}\ n\ \texttt{times}\ I\ \texttt{done} \rightarrow (x'', y'', d'')}$$

Réalisations

1. Augmentez la syntaxe de l'interprète. Le code exemple ci-dessus devrait produire la sortie suivante :

```
from (0, 0, East)
repeat 4 times
step,
turn left,
repeat 5 times
step
done,
turn right,
step,
turn left,
```

```
step
done
    => (0, 0, East)
```

2. Rajouter la construction syntaxique

$$I ::= \ldots \mid \texttt{step}\ n$$

comme du sucre syntaxique équivalent à `repeat` n `times step done`. Le code exemple est alors équivalent à :

```
from (X = 0, Y = 0, D = East)
repeat 4 times
  step,
  turn left,
  step 5,
  turn right,
  step,
  turn left,
  step
done
```

Déplacements conditionnés (15/20)

La prochaine étape consiste à ajouter la construction permettant de mettre une condition pour certains déplacements ; par exemple :

```
from (X = 0, Y = 0, D = East)
repeat 4 times
  step,
  turn left,
  repeat 5 times
    step
  done,
  turn right,
  step,
  when D = North or D = South do
    turn left,
    step
  done
done
```

Le non-terminal I de la grammaire est étendu avec la construction suivante :

$$\begin{array}{rcl} I & ::= & \ldots \\ & \mid & \underline{\texttt{when}}\ C\ \underline{\texttt{do}}\ I\ \underline{\texttt{done}} \\ C & ::= & C\ \underline{\texttt{and}}\ C \\ & \mid & C\ \underline{\texttt{or}}\ C \\ & \mid & \underline{\texttt{not}}\ C \\ & \mid & \underline{(}\ C\ \underline{)} \\ & \mid & \underline{\texttt{X}} = \underline{n} \\ & \mid & \underline{\texttt{Y}} = \underline{n} \\ & \mid & \underline{\texttt{D}} = D \\ & \mid & \underline{\texttt{X}} <= \underline{n} \\ & \mid & \underline{\texttt{Y}} <= \underline{n} \end{array}$$

La négation (`not`) est plus prioritaire que la conjonction (`and`), elle même plus prioritaire que la disjonction (`or`). Les opérateurs de conjonction et de disjonction sont associatifs.

Afin de réaliser l'évaluation des conditions, on définit le jugement suivant :

- $(x, y, d) \vdash C \rightsquigarrow b$ qui se lit : *Dans l'état* (x, y, d) *la condition* C *s'évalue en le booléen* b

Les règles sémantiques à implémenter (évidentes) sont les suivantes :

$$\frac{(x, y, d) \vdash C \rightsquigarrow \top \qquad (x, y, d) \vdash I \rightarrow (x', y', d')}{(x, y, d) \vdash \texttt{when}\ C\ \texttt{do}\ I\ \texttt{done} \rightarrow (x', y', d')}$$

$$\frac{(x,y,d) \vdash C \rightsquigarrow \bot}{(x,y,d) \vdash \texttt{when}\ C\ \texttt{do}\ I\ \texttt{done} \rightarrow (x,y,d)}$$

$$\frac{(x,y,d) \vdash C_1 \rightsquigarrow b_1 \qquad (x,y,d) \vdash C_2 \rightsquigarrow b_2}{(x,y,d) \vdash C_1\ \texttt{and}\ C_2 \rightsquigarrow b_1 \wedge b_2}$$

$$\frac{(x,y,d) \vdash C_1 \rightsquigarrow b_1 \qquad (x,y,d) \vdash C_2 \rightsquigarrow b_2}{(x,y,d) \vdash C_1\ \texttt{or}\ C_2 \rightsquigarrow b_1 \vee b_2}$$

$$\frac{(x,y,d) \vdash C \rightsquigarrow b}{(x,y,d) \vdash \texttt{not}\ C \rightsquigarrow \neg b}$$

$$\frac{}{(x,y,d) \vdash \texttt{X = } n \rightsquigarrow x = n}$$

$$\frac{}{(x,y,d) \vdash \texttt{Y = } n \rightsquigarrow y = n}$$

$$\frac{}{(x,y,d) \vdash \texttt{X <= } n \rightsquigarrow x \leq n}$$

$$\frac{}{(x,y,d) \vdash \texttt{Y <= } n \rightsquigarrow y \leq n}$$

$$\frac{}{(x,y,d) \vdash \texttt{D = } d' \rightsquigarrow d = d'}$$

Réalisations

1. Augmentez la syntaxe de l'interprète. Le code exemple ci-dessus devrait produire la sortie suivante :

```
from (0, 0, East)
repeat 4 times
step,
turn left,
repeat 5 times
step
done,
turn right,
step,
when (D = North) or (D = South) do
turn left,
step
done
done
    => (8, 20, East)
```

2. Rajouter toutes les comparaison de base en utilisant du sucre syntaxique :
   - `D != d` est équivalent à `not(D = d)`
   - `X != n` est équivalent à `not(X = n)`
   - `Y != n` est équivalent à `not(Y = n)`
   - `X < n` est équivalent à `X <= n - 1`
   - `Y < n` est équivalent à `Y <= n - 1`
   - `X > n` est équivalent à `not(X <= n)`
   - `Y > n` est équivalent à `not(Y <= n)`

- `X >=` n est équivalent à `not(X <=` $n-1$`)`
- `Y >=` n est équivalent à `not(Y <=` $n-1$`)`

Nommage de déplacement (20/20)

La prochaine étape consiste à pouvoir nommer une suite de déplacements pour pouvoir la réutiliser :

```
from (X = 0, Y = 0, D = East)
with diagstep = step, turn left, step, turn right do
  repeat 4 times
    diagstep,
    turn left,
    repeat 5 times
      diagstep
    done,
    turn right,
    diagstep,
    when D = North or D = South do
      turn left,
      diagstep
    done
  done
done
```

Le non-terminal I de la grammaire est étendu avec la construction suivante :

$$\begin{array}{rcl} I & ::= & \ldots \\ & | & \underline{\texttt{with}}\ \underline{id}\ \underline{=}\ I\ \underline{\texttt{do}}\ I\ \underline{\texttt{done}} \\ & | & \underline{id} \end{array}$$

La première production permet de créer une nouvelle association entre un identifiant et une instruction de déplacements. La seconde permet de faire appel à la suite d'instructions enregistrées.

Afin de pouvoir gérer les noms, nous utiliserons un environnement comme nous l'avons déjà vu en TP. Les jugements sont étendu pour prendre en compte cet environnement :

- $P \Rightarrow (x,y,d)$ devient $\Gamma \vdash P \Rightarrow (x,y,d)$
- $(x,y,d) \vdash I \rightarrow (x',y',d')$ devient $\Gamma, (x,y,d) \vdash I \rightarrow (x',y',d')$
- $(x,y,d) \vdash C \rightsquigarrow b$ devient $\Gamma, (x,y,d) \vdash C \rightsquigarrow b$

Ces nouveaux jugements se lisent de la même façon qu'avant en les faisant précéder par : *Dans l'environnement* Γ ...

Les règles sémantiques précédentes sont toutes étendues avec l'environnement (rien n'est à faire dans le code car l'environnement était déjà prévu du départ à travers le paramètre `gamma` de la fonction `eval`).

Afin de pouvoir exécuter une suite de déplacements enregistrée par son nom, il est nécessaire de sauver l'environnement courant au moment de la création de la nouvelle de l'enregistrement. Nous retrouvons ici le concept de clôture vu dans le TP1.

Les règles sémantiques sont les suivantes :

$$\frac{\Gamma[id \mapsto \langle I_1, \Gamma\rangle], (x,y,d) \vdash I_2 \rightarrow (x',y',d')}{\Gamma, (x,y,d) \vdash \texttt{with}\ id = I_1\ \texttt{do}\ I_2\ \texttt{done} \rightarrow (x',y',d')}$$

$$\frac{\Gamma(id) = \langle I, \Gamma'\rangle \qquad \Gamma', (x,y,d) \vdash I \rightarrow (x',y',d')}{\Gamma, (x,y,d) \vdash id \rightarrow (x',y',d')}$$

Le sens des notations est le suivant :

- $\Gamma[id \mapsto \langle I_1, \Gamma\rangle]$ est l'environnement Γ auquel on a ajouté l'association entre l'identifiant id la clôture $\langle I_1, \Gamma\rangle$; cela correspond à la fonction `add_env` dans `ast.ml`
- $\Gamma(id) = \langle I, \Gamma'\rangle$ signifie que dans l'envrionnement est Γ, l'identifiant id est associé à la clôture $\langle I, \Gamma'\rangle$; cela correspond à la fonction `find` dans `ast.ml`

Réalisations

1. Augmentez la syntaxe de l'interprète. Le code exemple ci-dessus devrait produire la sortie suivante :

```
from (0, 0, East)
with diagstep = step,
turn left,
```

```
step,
turn right
do
repeat 4 times
diagstep,
turn left,
repeat 5 times
diagstep
done,
turn right,
diagstep,
when (D = North) or (D = South) do
turn left,
diagstep
done
done done
    => (-12, 28, East)
```