TP n° 4

Paradigmes et interprétation
Licence Informatique
Université Côte d'Azur

Dans ce TP, on modifie l'interpréteur avec l'ordre supérieur : repartez du fichier `store-with.rkt`.

Limiter la tailler de la représentation de la mémoire

Dans l'interpréteur `store-with.rkt`, quand on modifie le contenu d'une boîte à l'aide de `set-box!`, on augmente la taille de la liste représentant la mémoire alors que le nombre de cellules utilisées est inchangé.

```
(test (interp (parse `{set-box! {box 2} 3}) mt-env mt-store)
      (v*s (numV 3) (list (cell 1 (numV 3)) (cell 1 (numV 2)))))
```

Modifiez la fonction `override-store` de sorte que la taille de la liste représentant la mémoire reste inchangée dans ces cas. Elle n'augmentera que lors de l'allocation d'un nouvel espace mémoire. Le test précédent devient alors :

```
(test (interp (parse `{set-box! {box 2} 3}) mt-env mt-store)
      (v*s (numV 3) (list (cell 1 (numV 3)))))
```

Après cette modification, la taille de la liste représentant la mémoire est égale au nombre d'allocations qui ont été nécessaires dans le programme et toutes les cellules ont des adresses distinctes.

Bloc d'instructions

L'instruction `begin` permet d'enchaîner l'évaluation de deux expressions et renvoie la valeur de la deuxième. Étendez l'instruction `begin` pour qu'elle permette d'enchaîner un nombre quelconque d'instructions (au moins une) et dont la valeur est celle de la dernière expression évaluée.

```
<Exp> = ...
      | {begin <Exp> <Exp>*}
```

Exemple :

```
(test (interp-expr `{let {[b {box 0}]}
                      {begin
                        {set-box! b {+ 1 {unbox b}}}
                        {set-box! b {* 2 {unbox b}}}
                        {set-box! b {+ 3 {unbox b}}}}})
      (numV 5))
```

Les enregistrements

Étendez l'interpréteur pour qu'il supporte les enregistrements et l'accès aux champs.

```
<Exp> ::= ...
        | {record [<Symbol> <Exp>]*}
        | {get <Exp> <Symbol>}
```

Vous devez définir une nouvelle valeur pour les enregistrements. Plusieurs représentations sont possibles, vous êtes libres de sélectionner celle qui vous convient le mieux. Il vous est cependant conseillé de lire l'énoncé de l'exercice suivant : celui-ci peut jouer sur votre choix.

Différentes erreurs peuvent survenir à l'exécution après cet ajout. Un enregistrement peut être utilisé en lieu et place d'un nombre ou d'une fonction, ou, inversement, le premier argument d'un `get` pourrait ne pas s'évaluer à un enregistrement. De même, le deuxième argument d'un `get` pourrait ne pas être un nom de champ valide. Vous devrez prendre en compte ces cas et générer des erreurs de manière adéquate. Dans le dernier cas, vous lancerez une erreur avec le message `"no such field"`. Les autres erreurs porteront les messages classiques `"not a number"`, `"not a function"` ou `"not a record"`.

Ces modifications correspondent à ce qui a été fait pour l'interpréteur `record.rkt`. La principale différence est l'existence d'une mémoire explicite qu'il faut gérer correctement. Par exemple, à l'initialisation d'un enregistrement, les différents arguments sont évalués de manière séquentielle de gauche à droite. Il faut passer la mémoire entre ces différentes évaluations, ce qui exclut l'utilisation de `map`.

Exemple :

```
(test (interp-expr `{let {[a {box 1}]}
                      {let {[r {record
                                [a {set-box! a {* 2 {unbox a}}}]
                                [b {set-box! a {* 2 {unbox a}}}]}]}
                        {+ {unbox a} {+ {get r a} {get r b}}}}})
      (numV 10))
```

Enregistrements mutables

On souhaite rendre les enregistrements mutables en ajoutant l'instruction `set!` au langage. Elle réalise une mise à jour impérative des champs des enregistrements.

```
<Exp> ::= ...
        | {set! <Exp> <Symbol> <Exp>}
```

Pour que les champs soient mutables, il faut que leurs valeurs soient stockées dans des emplacements mémoires. À la création d'un enregistrement, il faut donc allouer et initialiser ces emplacements mémoires.

Deux implémentations sont possibles. Dans la première, les adresses mémoires des champs sont directement stockées dans la représentation des enregistrements. L'accès et la modification des valeurs passent tous les deux par une indirection en mémoire. Une autre manière de faire est de lier

chaque champ à une boîte de notre langage. La valeur du champ est alors le contenu de la boîte. Celle-ci étant mutable, le champ peut l'être aussi.

Les deux approches sont viables et sont en fait très proches dans les mécanismes internes. Cependant, elles n'impactent pas l'interpréteur de la même façon. Vous êtes libre de choisir l'implémentation qui vous convient le mieux.

Exemples :

```
(test (interp-expr `{let {[r {record [a 1]}]}
                      {begin {set! r a 2} {get r a}}})
      (numV 2))

(test (interp-expr `{let {[r {record [a 1] [b 2]}]}
                      {begin
                        {set! r a {+ {get r b} 3}}
                        {set! r b {* {get r a} 4}}
                        {+ {get r a} {get r b}}}})
      (numV 25))
```