TP n° 10

Paradigmes et interprétation
Licence Informatique
Université Côte d'Azur

Le but de ce TP est d'étendre le langage de l'interpréteur `type.rkt`. Utilisez les implémentations faites dans les TP précédents pour modifier la fonction `interp`. Focalisez vos efforts sur la fonction `typecheck`. Les types `boolT` (pour les booléens) et `prodT` (pour les paires) apparaissent déjà dans le fichier `tp10-base.rkt` (une extension de `type.rkt`) mais ne sont pas encore exploités. Si une erreur de typage est détectée, utilisez l'une des trois fonctions d'erreurs `type-error`, `type-error-function` ou `type-error-product`.

Instructions booléennes

Ajoutez les expressions booléennes manquantes dans l'interpréteur.

```
<Exp> ::= ...
        | true
        | false
        | {if <Exp> <Exp> <Exp>}
        | {= <Exp> <Exp>}
```

L'égalité compare deux nombres et renvoie un booléen tandis que le branchement conditionnel prend une expression booléenne pour le test (pas un nombre !). Vous êtes libre de choisir la représentation interne des **valeurs** booléennes. Vous pouvez créer une nouvelle variante dans `Value` ou bien décider qu'une valeur booléenne est un nombre.

```
(test (typecheck-expr `{if {= 1 2} true false})
      (boolT))
(test/exn (typecheck-expr `{if 1 2 3})
          "typecheck")
```

Paires

Implémentez les expressions liées aux paires.

```
<Exp> ::= ...
        | {pair <Exp> <Exp>}
        | {fst <Exp>}
        | {snd <Exp>}
```

Retrouvez les règles de typage à partir des exemples du cours et des tests ci-dessous.

```
(test (typecheck-expr `{pair 1 true}) (prodT (numT) (boolT)))
(test (typecheck-expr `{fst {pair 1 true}}) (numT))
(test (typecheck-expr `{snd {pair 1 true}}) (boolT))
(test (typecheck-expr `{lambda {[x : (num * bool)]} {fst x}})
      (arrowT (prodT (numT) (boolT)) (numT)))
```

Récursion

Ajoutez l'expression `letrec` au langage.

```
<Exp> ::= ...
        | {letrec {[<Sym> : <Type> <Expr>]} <Expr>}
```

Le choix de l'implémentation est libre (entre méta-circulaire ou par mutation). Pour retrouver la règle de typage de `letrec`, construisez dans un premier temps celle de `let` (en se basant sur le sucre syntaxique). Exploitez ensuite vos connaissances pour déterminer la différence dans les règles de typage de `let` et de `letrec`.

```
(test (typecheck-expr
       `{letrec {[fib : (num -> num)
                      {lambda {[n : num]}
                        {if {= n 0}
                            0
                            {if {= n 1}
                                1
                                {+ {fib {+ n -2}}
                                   {fib {+ n -1}}}}}}}]}
          {fib 6}})
      (numT))
```

Inférence de type

Ajoutez l'inférence de type à l'interpréteur. Il vous faut reprendre l'algorithme d'unification de l'interpréteur `infer.rkt` ainsi que les différentes fonctions annexes associées. Modifiez la fonction `typecheck` pour que les règles que vous avez mises en place précédemment se transforme en contraintes d'unification. N'oubliez pas d'ajouter l'appel à `resolve` dans `typecheck-expr` pour que tous vos tests continuent à fonctionner.

```
(define (typecheck-expr [e : S-Exp]) : Type
  (resolve (typecheck (parse e) mt-env)))

(test (typecheck-expr `{let {[f : ? {lambda {[x : ?]} x}]}
                         {let {[x : ? {f 1}]}
                           f}})
      (arrowT (numT) (numT)))
```

```
(test (typecheck-expr
       `{letrec {[fib : ?
                      {lambda {[n : ?]}
                        {if {= n 0}
                            0
                            {if {= n 1}
                                1
                                {+ {fib {+ n -2}}
                                   {fib {+ n -1}}}}}}]}
           {fib 6}})
      (numT))
```

Indications (ou de l'utilité de lire le sujet jusqu'au bout)

Pour les implémentations dans `interp` des différentes parties, vous pouvez vous baser sur les travaux réalisés durant les séances précédentes.

- Les instructions booléennes ont déjà été implémentées dans le TP n° 3.
- Les paires ont déjà été implémentées dans le TP n° 7.
- Les interpréteurs `letrec-circ.rkt` et `letrec-mut.rkt` du cours n° 7 fournissent des implémentations alternative pour `letrec`.