Travail Pratique # 2 : rapport
IFT-2035

Felix Beaudoin (20244864) & Celina Zhang (20207461)

17 décembre 2023

1 Sucre syntaxique

— **1.1 Compréhension** :
Afin de pouvoir faire la conversion de Sexp à Lambda, il fallait d'abord qu'on utilise la fonction *readSexp*, qui était fournie, pour pouvoir interpréter comment les lignes de code se traduisent. À partir de cela, on a pu comprendre, avec la définition de Lexp, comment traduire nos Sexp en Lexp. Vous pouvez voir un exemple d'utilisation de *readSexp* à la section numéro 5. Et l'étape finale était de comprendre comment bien faire le pattern matching avec le résultat du readSexp qu'on reçoit. De plus, pour le *s2l Ltype*, c'était du pattern matching trivial. Nous avons du écrire un fonction que nous avons nommé *s2t*, qui prend un Sexp et retourne un type. Par la suite, nous avons implémenté une autre fonction auxilliaire que nous avons nommé *estFonction*, qui détermine si l'avant dernier symbole est une flèche (→). Si elle retourne True, nous avons une fonction et nous pouvons faire appel à *construireFonctionType*. Cette dernière fonction est implémentée afin de pouvoir créer le type de la fonction en temps que tel.

— **1.2 Problèmes rencontrées** :
La compréhension initiale de ce qui était demandé était assez simple puisque le processus de traduction était la même qu'au TP1. Cependant, le *(letrec ds e1 ... en)* ≃ *(letrec ds (begin e1 ... e))* nous a donné beaucoup de difficiulté. Nous n'étions donc pas capable de le faire même après plusieurs essais. Les 4 autres expressions Slip étaient assez simples et facilement implémentable une fois qu'on a compris ce que chaque élément représentait.

2 Vérification des types

— **2.1 Compréhension** :
Suite à la lecture de la *FIGURE 2 - Règles de typage* de l'énoncé, nous avons pu comprendre les 2 jugements qui sont la vérification et la synthèse. Notre *check* représente donc le jugement de vérification, où le type est déjà connu et il faut le vérifier. Notre *synth* représente donc le jugement de synthèse, où le type n'est pas connu et il faut l'inférer. De plus, la règle de typage numéro 4 (si nous comptons de gauche à droite) nous indique que si nous sommes capable d'inférer un type, on peut le vérifier. Ainsi, nous n'avons pas besoin d'implémenter la vérification des types ET les synthétiser.

— **2.2 Problèmes rencontrées** :
La compréhension initiale des règles de typage aide amplement dans l'implémentation de la vérification de types. Ensuite, l'implémentation du *check* était assez simple puisqu'il y en avait qu'un seul, le *Labs*. Il fallait simplement créer un environnement avec notre variable x de type t1. Par la suite, l'implémentation du *synth* sera développé dans le point numéro 3.

3 Implémentation du synth

— **3.1 Compréhension** :
La compréhension du *synth* était généralement simple puisque c'était plus ou moins le même principe que le *check*.

— **3.2 Problèmes rencontrées** :
3.2.1 Pour *Llit* et *Lid*, c'était très simple.
3.2.2 Pour *Ltype*, c'était très simple puisque c'était juste un *check* qu'il faut faire.
3.2.3 Pour *Lfuncall*, nous nous sommes largement inspirés du *eval*.
3.2.4 Pour *Ldec*, nous nous sommes inspirés de notre *check* du lambda.
3.2.5 Pour les trois fonctions de mémoire (*Lmkref*, *Lderef* et *Lassign*), c'était très simple d'implémenter.

3.2.6 Pour *Lrec*, après amplement d'essais et grâce à l'extension, nous avons réussi à l'implémenter. Nous avons testé plusieurs version du *Lrec*, mais la version que nous avons soumise est celle qui performait le mieux avec l'algorithme le plus efficace.

— **3.3 Surprises** :
Nous étions pris par surprise par le niveau de difficulté de la synthétisation de Lrec, c'était la partie du TP qui nous a prit le plus de temps.

4 Conclusion

En conclusion, ce travail pratique a beaucoup appronfi notre connaissance et notre compréhension sur le language fonctionnel. Ce travail nous a poussé à faire des recherches et d'acquérir des compétences au-delà de ce que nous avons vu en classe. Il nous a également apprit comment coder en pair puisque la séparation des tâches était compliquée. Finalement, nous avons trouvé que le niveau de difficulté était un peu plus bas comparativement au TP1.

5 Exemple : utilisation de readSexp

```
1 readSexp"(let x 1 e1 e2 e3 e4)"
2
3 Snode (Ssym "let") [Ssym "x",Snum 1,Ssym "e1",Ssym "e2",Ssym "e3",Ssym "e4"]
```