Rapport Projet Programmation Fonctionnelle
Inférence de type pour MINIML

Jérémie CARREZ, Léo LAZARE, Julie PAUL et Héloïse LAFARGUE

Département Sciences du Numérique - 2A Systèmes Logiciels
Programmation Fonctionnelle
2022-2023

Table des matières

Table des figures

1 Objectif du projet

Lors de ce projet nous allons reproduire l'intéfence de types de OCAML pour le langage MINIML. L'inférence de type permet de déterminer les types des expressions manipulées ou de détecter une potentielle mauvaise utilisation des données qui provoquerait une erreur à l'exécution.

2 Réalisation des tâches

2.1 Reconnaissance du langage MINIML

Dans cette première étape, il faut réaliser un parser à partir de la grammaire simplifiée de MINIML et du lexer fourni.

Cette section est très fortement inspiré du TP10 sur les parsers, il faut juste adapter au projet les variables et les types de flux.Par exemple, contrairement au TP10, comme le lexer est déjà définie, le seul type de flux utilisé est le flux des solutions et on utilise lazyflux pour gagner en efficacité. De la même manière qu'au TP10, on définit la structure monadique additive du parser (lignes 100 - 160).

Maintenant (ligne 160 - 208), on définit les fonctions de parsing de token (lignes 160 - 214). On commence par définir des fonctions générales qui vont nous faciliter la tâche par la suite. Une fonction "drop" qui va permettre de retourner le résultat d'un parser. Et une fonction "p_TOKEN" qui utilise drop et permet de retourner le token s'il existe bien. Finalement on créé les parsers pour chaque Token simple et complexe existant en utilisant "p_TOKEN".

Enfin, on définit récursivement les fonctions qui permettent le parsing des différentes structures (Expr, Liaison ...) en respectant la grammaire MINIML fournit dans le sujet. Par exemple, pour la fonction "p_Expr" qui parse la structure Expression, le premier type d'expression est "let" : on parse le token LET puis on parse une liaison avec la fonction "$p\_Liaison$" définit après qui renvoit un identifiant (id) et une autre expression (e1), ensuite on parse le token IN, puis on parse une autre expression avec "p_Expr" qui renvoit une deuxième expression (e2). On retourne le constructeur ELet, avec id, e1 et e2, qu'on peut trouver dans le fichier "$miniml\_printer.ml$".

Pour finir, on a rajouté une petite fonction de test qui se lance en faisant "dune utop" et une douzaine de fichiers tests. Ces tests permettent de vérifier que le parsing fonctionne bien pour des fichiers respectant la grammaire MINIML. Voici le résultat du test (figure 1) :

```
KO_fun_sans_().miniml

** parsing failed ! **

OK_ident.miniml
bonjourjesuIsunidEnt
** parsing succesfull ! **

OK_fun.miniml
(fun x -> ((( / ) x) 3))
** parsing succesfull ! **

KO_Condition_sansExpr.miniml

** parsing failed ! **

OK_letrec.miniml
let rec x = 1 in ((( * ) x) x)
** parsing succesfull ! **

OK_Condition.miniml
if false then true else false
** parsing succesfull ! **

OK_EBinopE.miniml
((( + ) 2) 2)
** parsing succesfull ! **

OK_const.miniml
1024
** parsing succesfull ! **

OK_EE.miniml
(x x)
** parsing succesfull ! **

OK_TestCompliqué.miniml
(fun x -> let rec x = 0 in ((( * ) x) x))
** parsing succesfull ! **

OK_E.miniml
true
** parsing succesfull ! **

OK_let.miniml
let x = 3 in (((fun x -> (fun y -> (x::y))) x) k)
** parsing succesfull ! **
```

FIGURE 1 – Résultat des Tests de parsing

2.2 Règles de typage du langage MINIML

Cette deuxième étape consiste à définir un algorithme qui :
— donne le type d'une expression,
— accumule les équations de types si ces équations ont une solution.

L'algorithme qui réalise cette tâche est la fonction *type_expr* présente dans le fichier *typer.ml*. Pour définir cet algorithme, nous avons utilisé les règles de typage présentées dans la Figure 3 du sujet.
Cette fonction parcourt récursivement l'expression entrée en argument et retourne le type de l'expression et les equations accumulées. Elle est composée d'une fonction auxiliaire qui permet d'accumuler les équations et mettre à jour l'environnement.
L'environnement initial contenir un certain nombre d'opérateurs qui peuvent être utiliser tels que les opérateurs binaires, @, : :...
Le type de chaque sous-expression est calculé de manière récursive dans l'environnent à jour.
Une expression peut prendre plusieurs formes :
— Constante : le type de la constante est retourné.
— Variable : 2 cas possibles : 1 La variable est présente dans l'environnement alors on retourne le type. 2 La variable n'est pas présente dans l'environnement alors on crée une variable de type fraiche qu'on ajoute dans l'environnement qu'on retourne.
— Paire : les 2 sous expressions sont typées puis le type des 2 expressions est retourné.

— Cons : les 2 sous expressions sont typées puis le type de la 2ème expression est retourné. La liste des expressions est mise à jour.
— Fonction : Une variable de type fraiche alpha est ajoutée à l'environnement puis le type de la fonction est retourné.
— If Then Else : elle composée d'un booleen et de 2 expressions, seul le type de l'expression 1 est retourné. La liste des équations est mise à jour.
— Application : La 1ère sous expression est typé : c'est un opérateur binaire ou alors une application. La 2ème sous expression est typé puis une variable de type fraiche alpha est créée et retournée, c'est le type de l'application. La liste des équations est mise à jour.
— Let : La 1ère expression est typée, puis l'environnement est mis à jour à l'aide du type de la 1ère expression. La 2ème expression est typé, le type obtenu est le type de l'expression.
— LetRec : Une variable de type fraiche alpha est créée et ajoutée à l'environnement. Les 2 sous-expression sont ensuite typées. Le type de la 2ème expression est retourné. La liste des équation est mise à jour.

2.3 Résolution du système d'équations

Dans cette partie, l'objectif est de résoudre le système d'équations obtenu précédemment grâce aux règles de la figure 4 du sujet. Ainsi, nous obtiendrons une forme normale du type calculé.

Pour cela, nous créeons une fonction de résolution des équations de type *resolution* qui prend en paramètre la liste des équations à résoudre. Elle renvoit vrai à la fin de l'éxecution si il existe bien une solution au système d'équation créé à la partie précédente, faux si il n'en existe pas.

Une partie des différentes équations peut etre testé en exécutant dune runtest (tests unitaires). Il faut pour cela des fois commenter la derniere fonction présente dans miniml_parser.ml, servant aux tests de la partie 1, et la derniere fonction du fichier miniml_typer.ml, qui sert également aux tests d'intégration (c.f. paragraphe suivant).

Pour les tests d'intégration, on a créé une méthode similaire à celle utilisée dans la partie 1, et on applique aux mêmes fichers de test les fonctions de typage et de vérification d'existence de solution. Les solutions sont affichées directement à l'esécution de dune utop également, à la suite de celles du parsing.

2.4 Programme principal

On a crée un programme principal, qui prend en argument le chemin (absolu) vers le fichier .miniml à analyser. Le fichier retourne le type de l'expression évaluée et affiche si elle est résolvable ou pas. Si le parsing échoue ou en cas d'autres erreurs, ces dernières sont affichées également.

2.5 Généralisation des types

Cette partie consiste à rendre plus généraux les types calculés. Pour cela il faut remplacer les types ajoutés dans l'environnement par des schémas de types. Il faut alors modifier la partie Ident, Let et LetRec dans la fonction *type_expr* :

— Pour Ident modifier le type retourné par un type instancié frais
— Pour Let et LetRec modifier le type ajouter dans l'environnement par un type général composé de l'environnement et du schéma de type. Il a été difficile de comprendre comment implanter ces 2 nouveaux types, qui n'ont pas été implantés.

3 Conclusion

En conclusion, ce projet visait à reproduire le comportement de l'algorithme d'inférence de types de OCAML pour le langage MINIML, en utilisant une grammaire simplifiée et un lexer fourni. Les tâches à réaliser ont été la reconnaissance du langage MINIML en utilisant un parser, la définition d'un algorithme pour donner le type d'une expression et accumuler les équations entre types, la résolution du système d'équations pour obtenir une forme normale du type calculé ou détecter l'absence de solutions, et la définition d'un programme principal qui lit un programme MINIML, applique les règles de typage, détermine la forme la plus générale des solutions ou l'absence de solution et affiche le type obtenu ou une erreur de typage. Nous n'avons cependant pas réussi à généraliser les types. L'analyse de tout fichier .miniml est accessible en exécutant le fichier miniml_principal.exe avec en argument le chemin vers le fichier à analyser.