APS petit guide

désolé pour les fautes de français

Conseils de ouf :

1. Lire le pdf du prof, beaucoup plus clair. : https://www-apr.lip6.fr/~manoury/Enseignement/2020-21/APS/aps0.pdf
2. Tuto interpréteur en ocaml : https://github.com/thizanne/imp/blob/master/tutoriel/article.org *(avec ça vous allez beaucoup mieux comprendre ce qu'on attend de vous sur le projet)*

Partie pratique (projet)

Objectif : écrire un langage de programmation interprété, et un typechecker en prolog

APS 0 : Langage fonctionnel (+ un print)

Par ou commencer ? Quoi faire ?

Créer un langage de programmation se fait en minimum **deux étapes** : L'analyse syntaxique, et la sémantique.

En APS, on a **trois étape** : l'analyse syntaxique, le typage, la sémantique.

Donc le premier truc a faire, c'est **l'analyse syntaxique**, et l'analyse syntaxique c'est quoi ? C'est definir les mots clés, et la grammaire de notre langage. Les mots clés sont les mots qu'on a envie d'accepter comme faisant partie de notre langage. (par exemple, on veut accepter le mot "true", mais pas le mot "manoury")

Et la grammaire, ce sont des regles qui disent quelles associations de mots clés sont valide ou pas, par exemple :

"**if** true **then** 1 **else** 2" C'est une association valide de trois mots clés accepté par le langage OCaml : **if**, **then**, **else**.

"Ok mais comment je définis les mots clés que j'accèpte et comment je définis ma grammaire ?"

C'est simple, les mots clés sont definis dans le "lexer", facilement grace a ocamllex que vous avez déja vu. Et la grammaire est définis dans un "parser" sous la forme d'une grammaire BNF (https://fr.wikipedia.org/wiki/Forme_de_Backus-Naur pour ceux qui ont pas fait compilation en L3 ou ceux qui ont fait semblant d'écouter).

Donc si vous savez pas par quoi commencer, fixez vous des objectifs clair : **première étape, faire l'analyse syntaxique de APS0**

ETAPE 1

Analyse syntaxique : Lexing -> Parsing -> AST

Partez du principe qu'un lexer, c'est une fonction du type : **string -> lexemes**

Partez du principe qu'un parser c'est une fonction du type : **lexemes -> AST**

Prérequis : écrire le lexer et le parser (le prof vous a donné un squelette, et le deuxieme lien que je donne en haut va vous aider a les ecrire si vous galérez), ensuite, dans un autre fichier :

1. Lire un fichier ".aps" en parametre
2. Le contenu de ce fichier est une string, donc on applique le lexer a cette string, pour récuperer les lexemes
3. Ensuite on applique le parser a ces lexemes, pour récuperer notre programme sous la forme d'un AST
4. Eventuellement on créer aussi une fonction de pretty_printer sur l'AST, pour pouvoir afficher notre programme sous la forme d'un AST et verifier que tout fonctionne bien.

Ok, **vous avez fini votre premier objectif** : maintenant vous avez un programme qui reconnais la syntaxe de votre langage et qui l'affiche proprement en AST sur la sortie standard, maintenant, votre deuxieme objectif : écrire un verificateur de type pour vos programmes.

ETAPE 2

Analyse de type : AST -> terme prolog -> typechecker.pl

Prérequis : Si vous connaissez pas prolog, allez voir un tuto vite fait quand meme, pour la syntaxe etc

1. Faite un fichier "apsToProlog.ml"
2. Comme pour le pretty_printer, ecrivez des fonctions ou vous allez "pattern-matcher" votre AST pour créer une string qui seras votre terme prolog représentant votre programme.
3. Ensuite, prenez le pdf du prof, lisez la partie "typage" jusqu'a ce que vous l'ayez compris, et codez les regles de typage du pdf, en prolog *(c'est difficile, cette partie la vous devez emmerder le prof pour qu'il vous aide si vous n'y arrivez pas seul)*
4. En executant votre programme avec un terme, il vous diras si c'est bon ou non, allez voir comment compiler/executer du code prolog et comment le tester.

Exemple :

$$\text{(IF)}\ \text{si}\ \Gamma \vdash_{\text{EXPR}} e_1 : \texttt{bool}, \text{si}\ \Gamma \vdash_{\text{EXPR}} e_2 : t\ \text{et si}\ \Gamma \vdash_{\text{EXPR}} e_3 : t \\ \text{alors}\ \Gamma \vdash_{\text{EXPR}} (\texttt{if}\ e_1\ e_2\ e_3) : t$$

ça, c'est la regle de typage pour le if, je vais la traduire en français

Si, dans le contexte de typage Γ, l'expression **e1** (la condition) a le type **bool**, et si dans ce meme contexte, l'expression **e2** (la conséquence) a le type **t**, et si l'expression e3 (l'alternative) a le type **t**, alors avec notre meme contexte de typage, l'expression (if e1 e2 e3) a le type **t**

Remarques : Vous voyez que la regle de typage nous dit que l'expression e2 et e3 doivent avoir le meme type "t", comme en OCaml, vous pouvez pas ecrire : `if true then 6 else false`, les deux branches du if doivent avoir le meme type.

Alors comment on traduit cette expression, et cette regle de typage, en terme prolog ? Voici ce que **moi** j'ai fait (vous pouvez donc faire totalement différement, attention)

```
typeExpr(if(Cond,Cons,Alter),Env,T2) :-

typeExpr(Cond,Env,bool) ,

typeExpr(Cons,Env,T2),

typeExpr(Alter,Env,T2).
```

En vrai, je trouve ça assez clair et explicite si on a lu la regle de typage du if avant, c'est exactement la meme chose, ça se lis d'abord a droite du symbole ":-" et ensuite a gauche du symbole.

Si notre condition a le type bool, si notre conséquence a le type **T2**, et si notre alternative a le meme type **T2**, alors **if cond cons alter**, a le type **T2**

la morale de cette histoire : **LISEZ LE PDF**, et faite exactement la meme traduction regle formelle -> terme prolog que je vient de faire.

Ok, deuxieme objectif finalisé, vous avez votre typechecker, il type check bien vos programmes, c'est quoi la suite ? **L'évaluation de nos programme** c'est la partie la plus facile, pour aps0 du moins.

ETAPE 3

Evaluation : AST -> évaluer le programme -> afficher le résultat (ce seras toujours un entier)

1. Dans un fichier "evaluation.ml", faite la meme chose que pour l'étape 1, on récupère un fichier, on lexer, on parser, on a notre programme en AST
2. Ecrire des fonctions d'évaluation sur l'AST *(en gros : un pattern matching sur l'AST, et chaque cas correspond a la ligne correspondate dans le PDF, genre si vous voulez écrire le cas de l'addition, regardez la regle de sémantique de l'addition dans le pdf)*

Regardez le deuxieme lien que je donne tout en haut dans "conseil de ouf", il y'a de l'évaluation, c'est exactement ce que vous devez faire, mais pour **votre langage.**

On résume :

La **syntaxe** : lexer, parser, ast

Le **typage** : On a un terme prolog, et on va coder les regles de typage écrite dans le pdf, dans un programme prolog, qui va nous dire si oui ou non notre terme est bien typé. (**la partie la plus difficile au début**)

La **sémantique** correspond a comment ont évalue un programme dans notre langage, donc chaque regle de sémantique se traduit en une ligne de notre évaluateur de programme en OCaml.

Partie théorique (partiel/examen)

Objectif : comprendre les notations (le pdf aide) et etre capable, a partir d'un programme, de l'évaluer grace aux regles de sémantique, a la main. Et de le "type checker" grace aux regle de typage, a la main.

Pas de secret, faut le faire vous meme a la main et **demandez au prof une demo et des annales, forcez si besoin.**

Voila.