Bureau d'études
Automates et Théorie des Langages
Documents autorisés – 1h00

1 Prélude

Attention, le non respect des noms imposés pour les répertoires dans ce qui suit pourra donner lieu à une pénalisation au niveau de la note.

- Télécharger depuis moodle l'archive `be2024-source.tgz`
- Désarchiver son contenu avec la commande : `tar xzvf be2024-source.tgz`
- Vous obtenez un répertoire nommé `be2024-source`
- Renommer ce répertoire sous la forme `be2024-source-Nom1-Nom2` (en remplaçant `Nom1` et `Nom2` par vos noms dans l'ordre alphabétique. Par exemple, si vous êtes le binôme Jacques Requin et Pénélope Pieuvre, vous utiliserez la commande :
`mv be2024-source be2024-source-Pieuvre-Requin`

2 Postlude

Lorsque la séance se termine au bout d'1h00 (1h20 pour les étudiants bénéficiant d'un tiers-temps), vous devrez :

- Vérifier que les résultats de vos travaux sont bien compilables (gardez toujours une archive compilable pour ne pas rendre une version qui ne compile pas)
- Créer une archive avec la commande :
`tar czvf be2024-source-Nom1-Nom2.tgz be2024-source-Nom1-Nom2`
- Déposer cette archive sur moodle

3 Un langage dérivé de Scheme

L'objectif du bureau d'étude est de construire deux analyseurs pour une version simplifiée du langage Scheme un descendant du langage Lisp. Ceux-ci seront composés d'un analyseur lexical construit avec l'outil `ocamlex` (sujet de TP 1) et d'un analyseur syntaxique construit d'une part, en exploitant l'outil `menhir` pour générer l'analyseur syntaxique (sujet de TP 2), et d'autre part la technique d'analyse descendante récursive programmée en `ocaml` en utilisant la structure de monade (sujet de TP 3).

Voici un exemple de système :

```
(AbCd (E . (1 FG)) ())
```

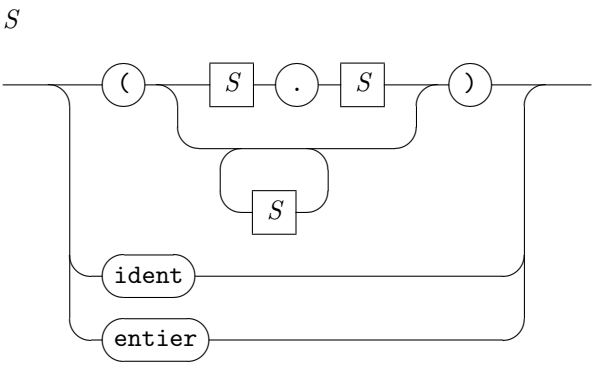
Cette syntaxe respecte les contraintes suivantes :

- les terminaux sont les identificateurs, les entiers, les parenthèses ouvrante `(` et fermante `)`, et le point `.` ;
- la définition d'un programme est appelée une S-expression ;
- une S-expression est soit une expression parenthésée, soit un identificateur, soit un entier ;
- une expression parenthésée est soit une S-expression suivie d'un `.` puis d'une seconde S-expression, soit une répétition de S-expression éventuellement vide, l'une ou l'autre comprise entre parenthèse ouvrante `(` et fermante `)`

Voici les expressions régulières pour les terminaux complexes :

- identificateur (noté `Ident`) : $([A-Z][a-zA-Z]^\star)^+$
- valeur entière positive (notée `entier`) : $0 \mid ([1-9][0-9]^\star)$

Voici la grammaire au format graphique de Conway :



Voici une grammaire LL(1) sous la forme de règles de production et les symboles directeurs de chaque règle de production :

1.	$S \to$ (X)	(
2.	$S \to$ ***Ident***	***Ident***
3.	$S \to$ ***entier***	***entier***
4.	$X \to \Lambda$	)
5.	$X \to S\ Y$	(***Ident entier***
6.	$Y \to$. S	.
7.	$Y \to L$	( ) ***Ident entier***)
8.	$L \to S\ L$	(***Ident entier***
9.	$L \to \Lambda$	)

4 Analyseur syntaxique ascendant

Vous devez travailler dans le répertoire `ascendant`.

Vous compilerez régulièrement les modifications réalisées pour détecter les erreurs au plus tôt.

Vous testerez régulièrement votre travail en ajoutant des tests de difficulté croissante dans le répertoire `tests` à la racine de l'archive. Les noms des tests doivent être préfixés par `ok_` pour les tests qui doivent réussir et par `ok_` pour les tests qui doivent échouer. Les tests seront pris en compte dans la notation.

La sémantique de l'analyseur syntaxique consiste à afficher les règles appliquées pour l'analyse.

Complétez les fichiers `Lexer.mll` (analyseur lexical) puis `Parser.mly` (analyseur syntaxique). Le programme principal est contenu dans le fichier `MainScheme.ml`. La commande `dune build MainScheme.exe` produit l'exécutable `_build/default/MainScheme.exe` qui prend comme paramètre le fichier à analyser. L'exemple de ce sujet est disponible dans le répertoire `tests`.

5 Analyseur syntaxique par descente récursive

Vous devez travailler dans le répertoire `descendant`.

Vous compilerez régulièrement les modifications réalisées pour détecter les erreurs au plus tôt.

Vous testerez régulièrement votre travail en ajoutant des tests de difficulté croissante dans le répertoire `tests` à la racine de l'archive. Les noms des tests doivent être préfixés par `ok_` pour les tests qui doivent réussir et par `ok_` pour les tests qui doivent échouer. Les tests seront pris en compte dans la notation.

L'analyseur syntaxique devra afficher les règles appliquées au fur et à mesure de l'analyse. Les éléments nécessaires sont disponibles en commentaires dans le fichier.

Complétez les fichiers `Scanner.mll` (analyseur lexical) puis `Parser.ml` (analyseur syntaxique). Attention, le nom du fichier contenant l'analyseur lexical est différent de celui du premier exercice car les actions lexicales effectuées sont différentes (l'analyseur lexical du premier exercice renvoie l'unité lexicale reconnue; l'analyseur lexical du second exercice construit la liste de toutes les unités lexicales et renvoie cette liste). Le programme principal est contenu dans le fichier `MainScheme.ml`. La commande `dune build MainScheme.exe` produit l'exécutable `_build/default/MainScheme.exe` qui prend comme paramètre le fichier à analyser. L'exemple de ce sujet est disponible dans le répertoire `tests`.