Bureau d'études
Automates et Théorie des Langages
Documents autorisés
1h30

1 Prélude

- Télécharger depuis moodle l'archive `source.tgz`
- Désarchiver son contenu avec la commande : `tar xzvf source.tgz`
- Vous obtenez un répertoire nommé `source`
- Renommer ce répertoire sous la forme `source_Nom1_Nom2` (en remplaçant Nom1 et Nom2 par le nom des deux membres du binôme). Par exemple, si les membres sont Xavier Crégut et Marc Pantel, vous utiliserez la commande : `mv source source_Cregut_Pantel`

2 Postlude

Lorsque la séance se termine à 15h30 (16h pour les étudiants bénéficiant d'un tiers-temps), vous devrez :

- Vérifier que les résultats de vos travaux sont bien compilables
- Créer une archive avec la commande : `tar czvf source_Xxx_Yyy.tgz source_Xxx_Yyy`
- Déposer cette archive sur moodle

3 Le langage Prolog version Edimbourg

L'objectif du bureau d'étude est de construire deux analyseurs pour une version simplifiée du langage Prolog selon la syntaxe Edimbourg. Ceux-ci seront composés d'un analyseur lexical construit avec l'outil `ocamlex` et d'un analyseur syntaxique construit respectivement, en exploitant l'outil `menhir` pour générer l'analyseur syntaxique, et la technique d'analyse descendante récursive programmée en `ocaml` en utilisant la structure de monade.

Voici un exemple de programme Prolog selon la syntaxe Edimbourg :

```
requin(jacques).

carnivore(requin(X)).

lapin(bugs).

vegetarien(lapin(X)).

omnivore(X) :-
   carnivore(X),
   vegetarien(X).
```

Cette syntaxe respecte les contraintes suivantes :

- les terminaux sont les variables `variable`, les symboles `symbole`, l'échec `fail`, les parenthèses ouvrante `(` et fermante `)`, la coupure `!`, la négation `-`, le point `.`, la virgule `,` et le symbole de déduction `:-` ;
- un programme est une suite non vide d'axiomes et de règles de déduction ;

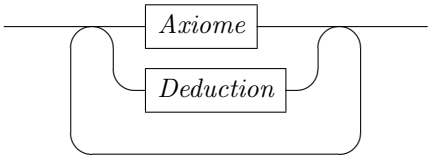
- un axiome est un prédicat suivi d'un point ;
- une règle de déduction est un prédicat suivi du symbole de déduction, puis d'une séquence non vide d'éléments séparés par des virgules. La règle de déduction se termine par un point ;
- un élément est soit un prédicat, soir la coupure, soit l'échec ;
- un prédicat est composé d'un symbole suivi de paramètres ;
- les paramètres sont représentés par une liste non vide de termes séparés par des virgules. Cette liste est précédée d'une parenthèse ouvrante et suivie d'une parenthèse fermante ;
- un terme est soit une variable, soit un symbole éventuellement suivi de paramètres.

Voici les expressions régulières pour les terminaux complexes :

- `symbole` : "$[a-z][a-zA-Z]^\star$"
- `variable` : "$[A-Z][a-zA-Z]^\star$"

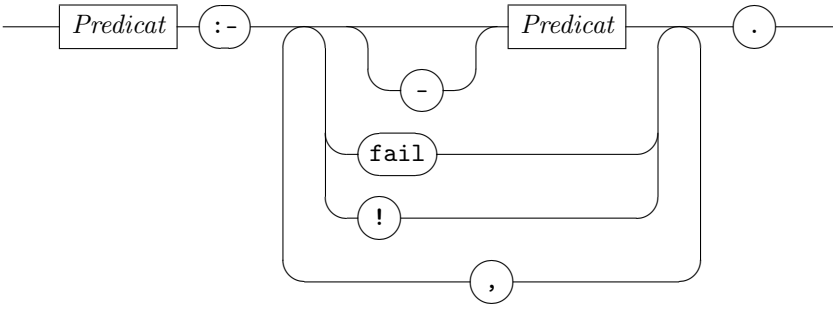
Voici la grammaire au format graphique de Conway :

Programme

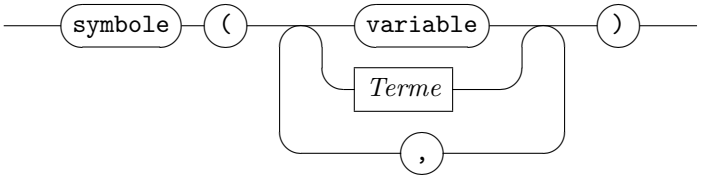


Axiome

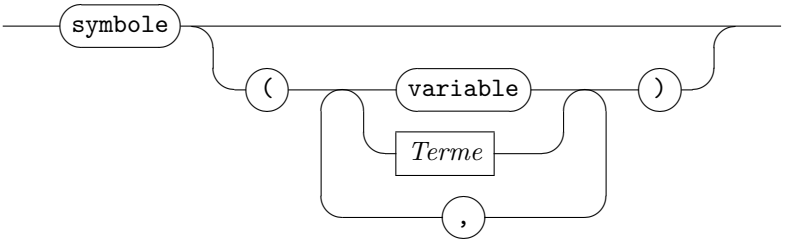
Deduction



Predicat



Terme



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

1.	$P_R \rightarrow R\ S_R$	***symbole***
2.	$S_R \rightarrow \Lambda$	**$**
3.	$S_R \rightarrow R\ S_R$	***symbole***
4.	$R \rightarrow P\ S_P$.	***symbole***
5.	$S_P \rightarrow \Lambda$	.
6.	$S_P \rightarrow$ **:-** $E\ S_E$	**:-**
7.	$E \rightarrow$ ***fail***	***fail***
8.	$E \rightarrow$ **!**	**!**
9.	$E \rightarrow$ -P	-
10.	$E \rightarrow P$	***symbole***
11.	$S_E \rightarrow \Lambda$	.
12.	$S_E \rightarrow$, $E\ S_E$	,
13.	$P \rightarrow$ ***symbole*** ($T\ S_T$)	***symbole***
14.	$T \rightarrow$ ***variable***	***variable***
15.	$T \rightarrow$ ***symbole*** O	***symbole***
16.	$S_T \rightarrow \Lambda$	)
17.	$S_T \rightarrow$, $T\ S_T$	,
18.	$O \rightarrow \Lambda$	, .
19.	$O \rightarrow$ ($T\ S_T$)	(

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.

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 `MainProlog.ml`. La commande `dune build MainProlog.exe` produit l'exécutable `_build/default/MainProlog.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.

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 `MainProlog.ml`. La commande `dune build MainProlog.exe` produit l'exécutable `_build/default/MainProlog.exe` qui prend comme paramètre le fichier à analyser. L'exemple de ce sujet est disponible dans le répertoire `tests`.