université de BORDEAUX

Année Universitaire 2022/2023
Session de Printemps 2023

Parcours : Licence 3 Informatique
Code UE : 4TIN603U
Épreuve : Compilation
Date : Lundi 24 avril 2023 **Heure :** 11h30
Durée : 1 heure 30
Documents : non autorisés
Épreuve de M. Vincent Penelle

Collège Sciences et Technologies

Indiquez votre code d'anonymat :

Avertissement

- Les trois exercices sont indépendants, vous pouvez les traiter dans l'ordre de votre choix.
- Vos réponses doivent être justifiées et aussi concises que possible.

Question	Points	Score
Sémantique	7	
Parsing	9	
Compréhension	4	
Total:	20	

Rappel:

Un AST est un arbre de syntaxe abstraite. C'est une représentation sous forme d'arbre d'un programme.

Menhir est un générateur de parseur d'OCaml.

Un token est un élément syntaxique de base (mot) d'un langage de programmation.

Une grammaire ambiguë est une grammaire qui admet des arbres de dérivation différents pour au moins une séquence de symboles terminaux.

Une grammaire est LR0 si son automate LR0 ne contient aucun conflit.

Une grammaire est SLR si les seuls conflits de son automate LR0 sont des conflits shift-reduce tels que pour chaque conflit, aucune lettre pouvant être shiftée n'appartient à l'ensemble Follow du non-terminal de gauche de la règle pouvant être réduite : si dans un état q la règle à réduire est $X \to \alpha$, alors il n'y a un conflit SLR que si on peut shifter un élément de Follow(X) depuis q.

Étant donné un non-terminal X, on rappelle que Follow(X) est l'ensemble des symboles terminaux pouvant apparaître à droite de X dans une dérivation de la grammaire. On ne vous demandera pas de détailler le calcul de Follow, mais d'être capable de donner cet ensemble.

Les annotations de priorité pour un token sont `%left`, `%right` et `%nonassoc`, et une règle peut être annotée avec `%prec`.

Exercice 1 : Sémantique (7 points)

On considère un mini-langage d'expressions qui manipule des listes de lettres minuscules, dont la syntaxe et la sémantique suivent. Les environnements sont des fonctions des variables (identifiants) vers des listes de chaînes de lettres minuscules. Un environnement est par défaut noté ρ. L'environnement qui associe la liste vide à la variable x et rien aux autres variables peut être noté $< x \to [] >$. Étant donné un environnement ρ, l'environnement égal à ρ mais dans lequel la valeur de x est remplacée par $['a']$ est noté $\rho[x \leftarrow ['a']]$.

Dans la sémantique, pour une lettre c, $c.+1$ qui est la lettre suivante ($a.+1 = b$, $o.+1 = p$ et $z.+1 = a$).

Étant donnée une expression e et un environnement ρ, on note $[e]_\rho$ la valeur de e dans ρ.

Syntaxe	AST	$[e]_\rho$
`[]`	[]	la liste vide []
`[c1]`	[c1]	la liste constante $[c1]$
`name`	name	$\rho(name)$
`e1 :> e2`	:> (e1, e2)	$[a1;\cdots;ak;b1;\cdots;bl;a1;\cdots;ak]$ si $[e1]_\rho = [a1;\cdots;ak]$ et $[e2]_\rho = [b1;\cdots;bl]$ (c'est-à-dire $[e1]_\rho@[e2]_\rho@[e1]_\rho$, @ étant la concaténation)
`P e`	P (e)	$[a1;\cdots;ak;ak;\cdots;a1]$ si $[e]_\rho = [a1;\cdots;ak]$ (c'est-à-dire $[e]_\rho$ suivi de son miroir).
`N e`	N (e)	$[a_0.+1;\cdots;a_k.+1]$ si $[e]_\rho = [a_0;\cdots;a_k]$ (c'est-à-dire chaque lettre est remplacée par la suivante)
`C e`	C (e)	$[a_0;\cdots;a_{k-1}]$ si $[e]_\rho = [a_0;\cdots;a_k]$ (c'est-à-dire on retire la dernière lettre) non-défini sur la liste vide

(a) Donnez l'AST correspondant à l'expression `C(N(['s'] :> ['n'])) :> []`, puis calculez sa valeur (sur un environnement quelconque ρ). Nommez chaque sous arbre et donner successivement la valeur correspondante à chaque sous-arbre (en commençant aux feuilles). Si un sous-arbre revient plusieurs fois, ne le calculez qu'une fois.

(b) On ajoute maintenant les instructions suivantes et on donne leur sémantique ($[\![i]\!]$ désigne la sémantique de l'instruction i, qui est une fonction prenant un environnement et renvoyant un environnement) :

Syntaxe	$[\![i]\!](\rho)$
name = e	$\rho[name \leftarrow [e]_\rho]$
i1;i2	$[\![i2]\!]([\![i1]\!](\rho))$
until(name,e) i	$\begin{cases}\rho \text{ si } \rho(name) = [e]_\rho \\ [\![until(name, e)\ i]\!]([\![i]\!](\rho)) \text{ sinon}\end{cases}$

Soit ρ l'environnement $< l \rightarrow [\,'n'\,]; a \rightarrow [\,'a'\,] >$.

On considère l'instruction I : `until(a,['b']) (l = l :> (N l); a = N a)`

Que vaut $[\![I]\!](\rho)$? On ne vous demande pas de détailler les calculs à fond, juste de donner le nouvel environnement à chaque instruction (les expressions étant simple, vous pourrez donner le résultat de chaque expression sans détail).

(c) On considère l'instruction I2 : `until(l,C(P m)) l = P l`. Que peut-on dire de l'exécution de cette boucle ? Vous distinguerez les cas possibles en fonction des valeurs de $\rho(l)$ et $\rho(m)$.

On attend juste un argument, vous n'avez pas forcément à faire de calcul explicite ici.

Exercice 2 : Parsing (9 points)

(a) Rappelez le rôle d'un analyseur lexical (lexeur) dans un compilateur.

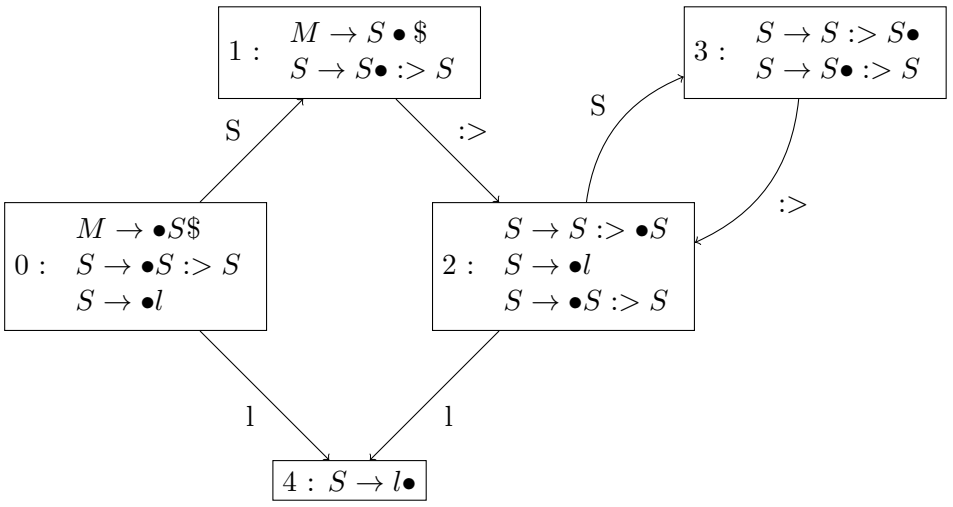
(b) Rappelez le rôle d'un parseur dans un compilateur.

(c) Pourquoi n'utilise t'on pas uniquement un lexeur pour ces deux tâches ?
Pourquoi n'utilise t'on pas uniquement un parseur pour ces deux tâches ?

On considère la grammaire G qui suit, dans laquelle les non-terminaux sont S et M, et les terminaux sont `$` `:>` et l. `$` est le symbole spécial de fin d'entrée. M est le non-terminal initial. L'état acceptant est celui qui pourrait shifter `$`.

```
M -> S $
S -> S :> S
S -> l
```

Son automate LR0 est le suivant :



(d) Quels est l'état acceptant ? Quels états ont des actions reduce et quelles sont-elles ?

(e) G est-elle LR0 ? Est-elle SLR ? Justifiez.

(f) G est-elle ambiguë ? Si oui, donnez une séquence de token qui peut être parsée de deux manière différentes, en donnant les deux arbres la générant et les deux calculs de l'automate correspondant.

Si votre exemple a plus de deux arbres le reconnaissant, n'en donnez que deux (mais cherchez surtout un exemple plus simple).

(g) Donner des annotations de priorités qui permettraient à menhir d'accepter cette grammaire. Comment les conflits sont-ils réglés avec ces annotations ? Dans la question précédente, quel calcul est celui accepté avec ces annotations ?

Exercice 3 : Compréhension (4 points)

(a) Que fait la phase d'analyse de types et quelles sont ses utilités ?

(b) Quelles sont les différences entre un argument passé par valeur et un argument passé par référence lors d'un appel de fonction ?

(c) Quelles sont les différences principales entre un interpréteur et un compilateur ?