THL – EPITA_ING1_2014_S1_THL
EPITA_APP1_2014_S1_THL

EPITA – Promo 2014 – **Sans document ni machine**
Avec formulaire de QCM

Juin 2012 (2h)

Répondre sur les formulaires de QCM ; aucune réponse manuscrite ne sera corrigée. Renseigner les champs d'identité. Sauf mention contraire, il y a exactement une et une seule réponse juste pour ces questions. Si plusieurs réponses sont valides, sélectionner la plus restrictive. Par exemple s'il est demandé si 0 est *nul*, *non nul*, *positif*, ou *négatif*, cocher *nul* qui est plus restrictif que *positif* et *négatif*, tous deux vrais. Une bonne réponse crédite, une réponse fausse pénalise, et pas de réponses donne 0.

1 Incontournables

Chaque erreur (ou non réponse) aux trois questions suivantes retire 1/6 de la note finale. Avoir tout faux divise donc la note par 2.

Q.1 La concaténation de deux langages rationnels est rationnelle.

a. vrai b. faux

Q.2 Une partie d'un langage rationnel est rationnelle.

a. vrai b. faux

Q.3 Toute grammaire hors-contexte *non ambiguë* engendre un langage reconnu par automate fini *non-déterministe*.

a. vrai b. faux

2 Contrôle

Q.4 L'expression rationnelle étendue $[\_]^?[a-z]^*[a-zA-Z0-9]^+[a-zA-Z0-9\_]^+$ n'engendre pas :

a. `_MSC_VER`
b. `_WIN32`
c. `__OS400__`
d. `_DARWIN_C_SOURCE`
e. `OS_2`

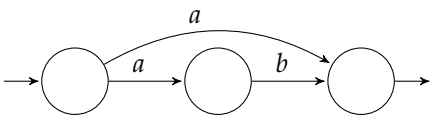
Q.5 Combien existe-t-il de sous-ensembles d'un ensemble de taille n ?

a. $n-2$ b. $n!$ c. $n(n+1)/2$ d. n^2 e. 2^n

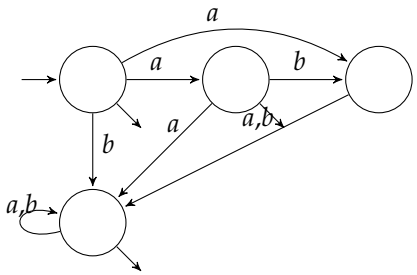
Q.6 Combien existe-t-il de mots de n lettres écrits dans un alphabet de m symboles ?

a. $n-m$ b. $n!$ c. $n(n+m)/m$ d. n^m e. m^n

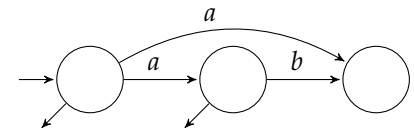
Q.7 Quel automate reconnaît le langage complémentaire du langage reconnu par l'automate suivant (sur l'alphabet $\Sigma = \{a, b\}$) :



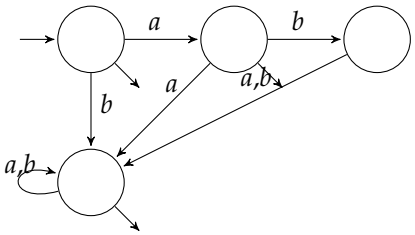
a.



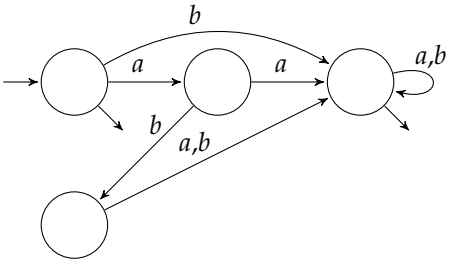
b.



c.



d.



Q.8 Quelle est la classe de la grammaire suivante ?

$$A \rightarrow aABC \qquad CB \rightarrow BC \qquad bC \rightarrow bc$$
$$A \rightarrow abC \qquad bB \rightarrow bb \qquad cC \rightarrow cc$$

a. Choix finis
b. Rationnelle
c. Hors contexte
d. Sensible au contexte
e. Monotone

Q.9 Quelle est la classe de la grammaire suivante ?

$$S \rightarrow aSb \mid c$$

a. Choix finis
b. Rationnelle
c. Hors contexte
d. Sensible au contexte
e. Monotone

Q.10 Si une grammaire hors contexte est non ambiguë

a. elle est LL(1)
b. elle est LL(k)
c. elle n'est pas nécessairement LL
d. elle produit nécessairement des conflits dans un parseur LL

Q.11 Si le parseur LALR(1) associé à une grammaire présente des conflits

a. Il n'existe pas de parseur pour cette grammaire ;
b. Il peut exister un parseur LR(0) pour cette grammaire ;
c. Il peut exister un parseur LLR(1) pour cette grammaire ;
d. Il peut exister un parseur LR(1) pour cette grammaire ;
e. Il peut exister un parseur SLR(1) pour cette grammaire.

Q.12 LL(k) signifie

a. lecture en deux passes de gauche à droite, avec k symboles de regard avant ;
b. lecture en deux passes de gauche à droite, avec une pile limitée à k symboles ;
c. lecture en une passe de gauche à droite, avec k symboles de regard avant ;
d. lecture en une passe de gauche à droite, avec une pile limitée à k symboles.

Q.13 Dans une analyse classique en utilisant Lex et Yacc :

a. on appelle la fonction `yyparse` une fois, elle appelle la fonction `yylex` plusieurs fois ;
b. on appelle la fonction `yyparse(yylex())` plusieurs fois ;
c. on appelle la fonction `yylex` plusieurs fois, puis la fonction `yyparse` une fois ;
d. on appelle la fonction `yyparse` plusieurs fois, elle appelle la fonction `yylex` chaque fois.

3 Dessine-moi un programme

Le langage Logo permet (entre autres) de dessiner sur un écran à l'aide d'une « tortue » que l'on manipule avec des instructions comme `forward 10` (avance de 10 unités en traçant un trait) ou `right 30` (tourne à droite de 30 degrés).

Considérez la grammaire suivante. Le terminal `"num"` désigne un entier littéral et le terminal `"id"` un identifiant.

```
⟨instr⟩ ::= "forward" ⟨exp⟩ | "left" ⟨exp⟩ | "right" ⟨exp⟩
          | "repeat" ⟨exp⟩ "[" ⟨instrs⟩ "]"  # Loop.
          | "to" "id" ⟨args⟩ ⟨instrs⟩ "end"  # Subprogram definition.
          | "id"                             # Subprogram call.

⟨exp⟩ ::= "num" | ":" "id"

⟨args⟩ ::= ⟨args⟩ ⟨arg⟩ |
```

Q.14 La grammaire ne définit pas ⟨instrs⟩ (une ou plusieurs ⟨instr⟩). Quelles règles ajouter pour un parseur LR(1) :

a.
```
⟨instrs⟩ ::= ⟨instr⟩ | ⟨instrs⟩ ⟨instr⟩
```

b.
```
⟨instrs⟩ ::= | ⟨instrs⟩ ⟨instr⟩
```

c.
```
⟨instrs⟩ ::= ⟨instr⟩ | ⟨instr⟩ ⟨instrs⟩
```

d.
```
⟨instrs⟩ ::= | ⟨instr⟩ ⟨instrs⟩
```

Q.15 Pour un parseur LL(1) ?

a.
```
⟨instrs⟩ ::= ⟨instr⟩ | ⟨instrs⟩ ⟨instr⟩
```

b.
```
⟨instrs⟩ ::= | ⟨instrs⟩ ⟨instr⟩
```

c.
```
⟨instrs⟩ ::= ⟨instr⟩ | ⟨instr⟩ ⟨instrs⟩
```

d.
```
⟨instrs⟩ ::= | ⟨instr⟩ ⟨instrs⟩
```

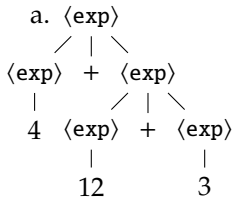
Arithmétique

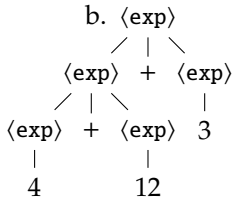
On souhaite donner la possibilité à l'utilisateur de saisir des expressions arithmétiques afin de donner plus de flexibilité. Ainsi nous modifions les règles ⟨exp⟩ :

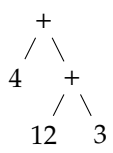
```
⟨exp⟩  ::= ⟨exp⟩ "+" ⟨exp⟩ | ⟨exp⟩ "-" ⟨exp⟩
         | ⟨exp⟩ "*" ⟨exp⟩  | ⟨exp⟩ "/" ⟨exp⟩
         | ":" "id" | "num"
```

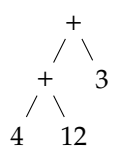
Malheureusement, ces règles introduisent des ambiguïtés. Elles ne codent pas l'associativité gauche comme on le souhaiterait ni la priorité de "*" et "/" sur "+" et "-".

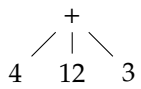
Q.16 Quel(s) est(sont) le(s) arbre(s) de dérivation de `4 + 12 + 3` ? (plusieurs réponses possibles)

a.


b.


c.


d.


e.


Q.17 Que faut-il en déduire à propos de la grammaire.

a. Linéaire
b. A des conflits
c. Réductible
d. Non-déterministe
e. Ambiguë

Q.18 Que faut-il en déduire à propos du parseur LALR(1) engendré pour cette grammaire ?

a. Linéaire
b. A des conflits
c. Réductible
d. Non-déterministe
e. Ambigu

Q.19 Proposer une grammaire non ambiguë équivalente à la grammaire de ⟨exp⟩ pour résoudre les conflits et pour respecter l'associativité gauche ainsi que la priorité des opérateurs. Profitez-en pour ajouter les parenthèses.

a.
```
⟨exp⟩ ::= ⟨exp⟩ "*" ⟨term⟩ | ⟨exp⟩ "/" ⟨term⟩ | ⟨term⟩ ;
⟨term⟩ ::= ⟨term⟩ "+" ⟨factor⟩ | ⟨term⟩ "-" ⟨factor⟩ | ⟨factor⟩ ;
⟨factor⟩ ::= ":" "id" | "num" | "(" ⟨factor⟩ ")" ;
```

b.
```
⟨exp⟩ ::= ⟨exp⟩ "+" ⟨term⟩ | ⟨exp⟩ "-" ⟨term⟩ | ⟨term⟩ ;
⟨term⟩ ::= ⟨term⟩ "*" ⟨factor⟩ | ⟨term⟩ "/" ⟨factor⟩ | ⟨factor⟩ ;
⟨factor⟩ ::= ":" "id" | "num" | "(" ⟨exp⟩ ")" ;
```

c.
```
⟨exp⟩ ::= ⟨exp⟩ "*" ⟨term⟩ | ⟨exp⟩ "/" ⟨term⟩ | ⟨term⟩ ;
⟨term⟩ ::= ⟨term⟩ "+" ⟨factor⟩ | ⟨term⟩ "-" ⟨factor⟩ | ⟨factor⟩ ;
⟨factor⟩ ::= ":" "id" | "num" | "(" ⟨exp⟩ ")" ;
```

d.
```
⟨exp⟩ ::= ⟨term⟩ "*" ⟨exp⟩ | ⟨term⟩ "/" ⟨exp⟩ | ⟨term⟩ ;
⟨term⟩ ::= ⟨factor⟩ "+" ⟨term⟩ | ⟨factor⟩ "-" ⟨term⟩ | ⟨factor⟩ ;
⟨factor⟩ ::= ":" "id" | "num" | "(" ⟨factor⟩ ")" ;
```

e.
```
⟨exp⟩ ::= ⟨term⟩ "*" ⟨exp⟩ | ⟨term⟩ "/" ⟨exp⟩ | ⟨exp⟩ ;
⟨term⟩ ::= ⟨factor⟩ "+" ⟨term⟩ | ⟨factor⟩ "-" ⟨term⟩ | ⟨term⟩ ;
⟨factor⟩ ::= ":" "id" | "num" | "(" ⟨factor⟩ ")" ;
```

Q.20 Avec Bison/Yacc, quelles directives ajouter pour respecter l'associativité et les priorités sans changer les règles de ⟨exp⟩.

a.
```
%left "+" "-"  %left "/" "*"
```

b.
```
%left "/" "*"  %left "+" "-"
```

c.
```
%left %{ "/" "*" %} %> %{ "+" "-" %}
```

d.
```
%left "+" "-" "/" "*"
```

e.
```
%left "/" "*" "+" "-"
```

Q.21 Compléter, jusqu'à l'acceptation, la séquence de décalages/réductions suivante :

```
  ⊢            left : s * 12 + : i ⊣
s ⊢ "left"           : s * 12 + : i ⊣
s ⊢ "left" ":"         s * 12 + : i ⊣
s ⊢ "left" ":" "id"      * 12 + : i ⊣
r ⊢ "left" exp           * 12 + : i ⊣
```

a.

```
  ⊢                 left : v * 12 + : i ⊣
s ⊢ "left"                : v * 12 + : i ⊣
s ⊢ "left" ":"              v * 12 + : i ⊣
s ⊢ "left" ":" "id"           * 12 + : i ⊣
r ⊢ "left" exp                * 12 + : i ⊣
r ⊢ "left" exp                * 12 + : i ⊣
s ⊢ "left" exp "*"              12 + : i ⊣
s ⊢ "left" exp "*" "num"           + : i ⊣
s ⊢ "left" exp "*" "num" "+"         : i ⊣
s ⊢ "left" exp "*" "num" "+" ":"       i ⊣
s ⊢ "left" exp "*" "num" "+" ":" "id"    ⊣
r ⊢ "left" exp "*" "num" "+" exp         ⊣
r ⊢ "left" exp "*" exp                   ⊣
r ⊢ "left" exp                           ⊣
r ⊢ instr                                ⊣
s ⊢ instr ⊣
a
```

b.

```
  ⊢               left : v * 12 + : i ⊣
s ⊢ "left"             : v * 12 + : i ⊣
s ⊢ "left" ":"           v * 12 + : i ⊣
s ⊢ "left" ":" "id"        * 12 + : i ⊣
r ⊢ "left" exp             * 12 + : i ⊣
r ⊢ "left" exp             * 12 + : i ⊣
s ⊢ "left" exp "*"           12 + : i ⊣
s ⊢ "left" exp "*" "num"        + : i ⊣
r ⊢ "left" exp "*" exp          + : i ⊣
s ⊢ "left" exp "*" exp "+"        : i ⊣
s ⊢ "left" exp "*" exp "+" ":"      i ⊣
s ⊢ "left" exp "*" exp "+" ":" "id"   ⊣
r ⊢ "left" exp "*" exp "+" exp        ⊣
r ⊢ "left" exp "*" exp                ⊣
r ⊢ "left" exp                        ⊣
r ⊢ instr                             ⊣
s ⊢ instr ⊣
a
```

c.

```
  ⊢               left : v * 12 + : i ⊣
s ⊢ "left"             : v * 12 + : i ⊣
s ⊢ "left" ":"           v * 12 + : i ⊣
s ⊢ "left" ":" "id"        * 12 + : i ⊣
r ⊢ "left" exp             * 12 + : i ⊣
r ⊢ "left" exp             * 12 + : i ⊣
s ⊢ "left" exp "*"           12 + : i ⊣
s ⊢ "left" exp "*" "num"        + : i ⊣
r ⊢ "left" exp "*" exp          + : i ⊣
r ⊢ "left" exp                  + : i ⊣
s ⊢ "left" exp "+"                : i ⊣
s ⊢ "left" exp "+" ":"              i ⊣
s ⊢ "left" exp "+" ":" "id"           ⊣
```

```
r ⊢ "left" exp "+" exp                ⊣
r ⊢ "left" exp                        ⊣
r ⊢ instr                             ⊣
s ⊢ instr ⊣
a
```

d.

```
  ⊢               left : v * 12 + : i ⊣
s ⊢ "left"             : v * 12 + : i ⊣
s ⊢ "left" ":"           v * 12 + : i ⊣
s ⊢ "left" ":" "id"        * 12 + : i ⊣
r ⊢ "left" exp             * 12 + : i ⊣
r ⊢ instr                  * 12 + : i ⊣
r ⊢ instrs                 * 12 + : i ⊣
s ⊢ instrs "*"               12 + : i ⊣
s ⊢ instrs "*" "num"            + : i ⊣
r ⊢ instrs "*" exp              + : i ⊣
r ⊢ instrs                      + : i ⊣
s ⊢ instrs "+"                    : i ⊣
s ⊢ instrs "+" ":"                  i ⊣
s ⊢ instrs "+" ":" "id"               ⊣
r ⊢ instrs "+" exp                    ⊣
r ⊢ instrs                            ⊣
s ⊢ instrs ⊣
a
```

e.

```
  ⊢               left : v * 12 + : i ⊣
s ⊢ "left"             : v * 12 + : i ⊣
s ⊢ "left" ":"           v * 12 + : i ⊣
s ⊢ "left" ":" "id"        * 12 + : i ⊣
r ⊢ "left" exp             * 12 + : i ⊣
r ⊢ instr                  * 12 + : i ⊣
s ⊢ instr "*"                12 + : i ⊣
s ⊢ instr "*" "num"             + : i ⊣
r ⊢ instr "*" exp               + : i ⊣
s ⊢ instr "*" exp "+"             : i ⊣
s ⊢ instr "*" exp "+" ":"           i ⊣
s ⊢ instr "*" exp "+" ":" "id"        ⊣
r ⊢ instr "*" exp "+" exp             ⊣
r ⊢ instr "*" exp                     ⊣
r ⊢ instr                             ⊣
r ⊢ instr                             ⊣
s ⊢ instr ⊣
a
```

4 À propos de ce cours

Nous nous engageons à ne pas tenir compte des renseignements ci-dessous pour noter votre copie. Ils ne sont pas anonymes, car nous sommes curieux de confronter vos réponses à votre note. En échange, quelques points seront attribués pour avoir répondu. Merci d'avance.

Répondez sur les formulaires de QCM qui vous sont remis. Vous pouvez cocher plusieurs réponses par question.

Q.22 Prises de notes

a. Aucune
b. Sur papier
c. Sur ordinateur à clavier
d. Sur ardoise
e. Sur le journal du jour

Q.23 Travail personnel

a. Rien
b. Bachotage récent
c. Relu les notes entre chaque cours
d. Fait les annales
e. Lu d'autres sources

Q.24 Ce cours

a. Est incompréhensible et j'ai rapidement abandonné
b. Est difficile à suivre mais j'essaie
c. Est facile à suivre une fois qu'on a compris le truc
d. Est trop élémentaire

Q.25 Ce cours

a. Ne m'a donné aucune satisfaction
b. N'a aucun intérêt dans ma formation
c. Est une agréable curiosité
d. Est nécessaire mais pas intéressant
e. Je le recommande

Q.26 L'enseignant

a. N'est pas pédagogue
b. Parle à des étudiants qui sont au dessus de mon niveau
c. Me parle
d. Se répète vraiment trop
e. Se contente de trop simple et devrait pousser le niveau vers le haut