Compilation — Partiel — Automne 2023

Durée 2h. Documents de cours et notes personnelles autorisés. Les exercices sont indépendants.

Exercice 1 (Commentaires) Dans les langages C et Java, un commentaire commence par la séquence `/*` et termine par la séquence `*/`. L'intérieur du commentaire peut contenir n'importe quel caractère, Notez que les commentaires ne peuvent pas être emboîtés : une séquence `/*` à l'intérieur d'un commentaire ne joue aucun rôle, et la première séquence `*/` rencontrée clôt le commentaire courant. Exemples :

1. Cette séquence est un commentaire.

```
/* La guerre des Gaules */
```

2. Cette séquence n'est pas un commentaire

```
/* Un commentaire commence par /*, termine par */ et peut contenir
   n'importe quel caractère. */
```

   car tout ce qui vient après le premier `*/` est en dehors.

3. Cette séquence est un commentaire

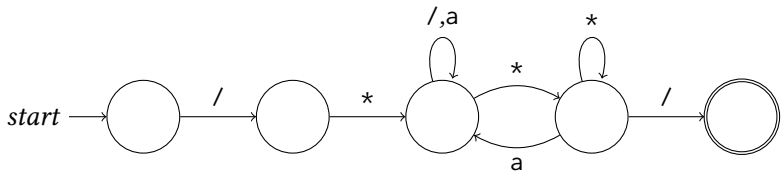
```
/* Un commentaire commence par /*, termine par * suivi de /, et
   peut contenir n'importe quel caractère. */
```

Questions.

1. Donner un automate acceptant exactement les mots sur l'alphabet $A = \{ /, *, a \}$ qui sont des commentaires bien formés.
2. Donner une expression régulière reconnaissant ce même langage.
3. Supposons maintenant que les commentaires peuvent être emboîtés. Alors les exemples 1 et 2 ci-dessus sont des commentaires bien formés, mais pas l'exemple 3. Montrer qu'il ne peut exister d'automate fini acceptant exactement les mots sur l'alphabet A qui sont des commentaires bien formés selon ce nouveau critère. *Indication :* supposer qu'un tel automate *fini* existe, et montrer qu'il accepte nécessairement aussi des mots invalides.

Correction :

1. *Point délicat : gérer ce qui vient après un symbole $*$ à l'intérieur d'un commentaire.*



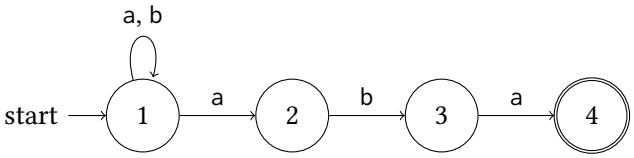
2. *Solution compacte (mais non unique) :*

   $/*(/ \mid *^{\star}a)^{\star}*/$

3. *Supposons qu'il existe un automate fini déterministe reconnaissant le langage des commentaires emboîtés. Notons N le nombre d'états de cet automate. Considérons le mot $m = (/*)^N(*/)^N$. Ce mot étant un commentaire valide, il est accepté par l'automate : il étiquette un chemin allant de l'état initial à un état acceptant. Le préfixe $m_1 = (/*)^N$ en particulier, de longueur $> N$, étiquette un chemin comportant une boucle. On décompose $m_1 = m_2m_3m_4$ où m_3 correspond à la boucle. Alors le mot $m_2m_4(*/)^N$ donne un chemin acceptant, et est donc un commentaire bien formé. Or, $m_2m_4 < 2N$, et ce mot ne peut donc contenir N occurrences de `/*`. Ainsi $m_2m_4(*/)^N$ ne peut pas être un commentaire bien formé. Contradiction.*

□

Exercice 2 (Automates) Voici un automate non déterministe sur l'alphabet $A = \{ a, b \}$.



Questions.

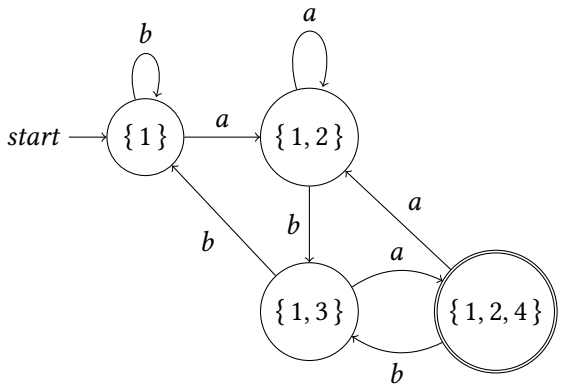
1. Décrire *en français* l'ensemble des mots acceptés par cet automate.
2. Donner une expression régulière reconnaissant l'ensemble des mots acceptés par cet automate.
3. Déterminiser l'automate.

Correction :

1. *Cet automate reconnaît l'ensemble des mots qui terminent par la séquence* aba.
2. *Expression régulière :*

   $$(a \mid b)^{\star} aba$$

3. *Automate déterminisé, où on étiquette chaque état par l'ensemble des états de l'automate d'origine qu'il représente.*



□

Exercice 3 (Expression conditionnelle) Le langage Python propose une expression conditionnelle de la forme

```
<expr1> if <condition> else <expr2>
```

Si la condition est vraie, on calcule la valeur de l'expression 1, et sinon on calcule la valeur de l'expression 2. Ainsi, on peut définir le minimum entre deux valeurs `x` et `y` avec l'expression suivante.

```
x if x<=y else y
```

Voici une grammaire pour un fragment des expressions Python faisant intervenir cette construction.

E	::=	n	constante numérique
	|	E `<=` E	opération binaire
	|	`(` E `)`	expression entre parenthèses
	|	E `if` E `else` E	expression conditionnelle

Questions.

1. Détailler les étapes de l'analyse ascendante de la phrase suivante.

   ```
   (1 if 2 <= 3 else 4) <= 2
   ```

2. Montrer qu'il existe deux arbres de dérivation pour la phrase suivante.

   ```
   2 <= 1 if 4 <= 3 else 5
   ```

3. En Python, l'évaluation de `0 if True else 1 <= 2` donne `0`.
   (a) En déduire quelle est la dérivation retenue parmi les deux précédentes.
   (b) Dire ce qu'aurait été le résultat si l'autre dérivation avait été choisie.
4. Montrer qu'il existe deux arbres de dérivation pour la phrase suivante.

   ```
   1 if E else 2 if E else 3
   ```

5. En Python, l'évaluation de `1 if False else 2 if True else 3` donne `2`.
   (a) Expliquer pourquoi ce test ne permet pas de distinguer entre les deux dérivations précédentes.
   (b) Proposer une autre expression dont le résultat serait plus révélateur.

(c) Pour l'expression précédente, détailler les différentes issues possibles et ce que chacune nous enseigne sur la dérivation choisie.

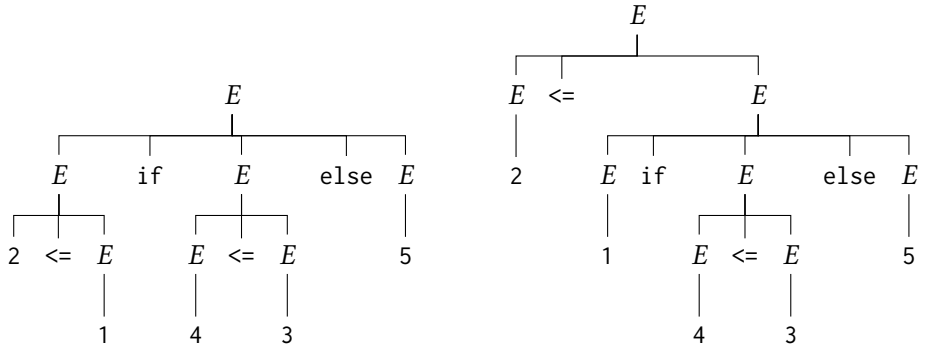
6. Construire une partie de l'automate LR(0) déterministe de cette grammaire montrant :
   — l'état de départ,
   — un état avec un conflit correspondant à l'ambiguïté aperçue à la question 2,
   — les états intermédiaires et les transitions permettant d'aller de l'état de départ à l'état de conflit susmentionné.

Correction :

1. *Tableau étape par étape. Progression : S. Réduction : R.*

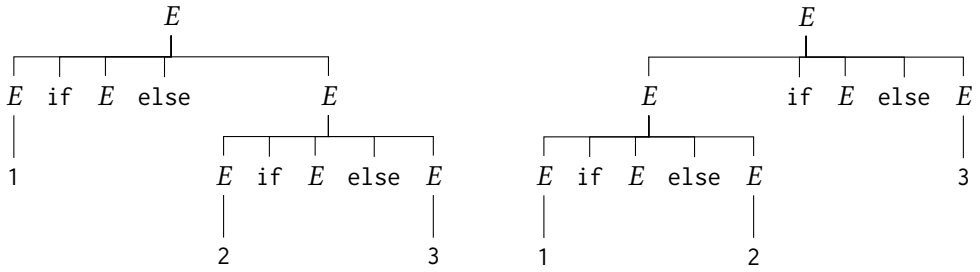
Pile	*Entrée*	*Action*
∅	`(1 if 2 <= 3 else 4) <= 2`	*S*
`(`	`1 if 2 <= 3 else 4) <= 2`	*S*
`(1`	`if 2 <= 3 else 4) <= 2`	*R n*
`(`*E*	`if 2 <= 3 else 4) <= 2`	*S*
`(`*E* `if`	`2 <= 3 else 4) <= 2`	*S*
`(`*E* `if 2`	`<= 3 else 4) <= 2`	*R n*
`(`*E* `if` *E*	`<= 3 else 4) <= 2`	*S*
`(`*E* `if` *E* `<=`	`3 else 4) <= 2`	*S*
`(`*E* `if` *E* `<= 3`	`else 4) <= 2`	*R n*
`(`*E* `if` *E* `<=` *E*	`else 4) <= 2`	*R E* `<=` *E*
`(`*E* `if` *E*	`else 4) <= 2`	*S*
`(`*E* `if` *E* `else`	`4) <= 2`	*S*
`(`*E* `if` *E* `else 4`	`) <= 2`	*R n*
`(`*E* `if` *E* `else` *E*	`) <= 2`	*R E* `if` *E* `else` *E*
`(`*E*	`) <= 2`	*S*
`(`*E*`)`	`<= 2`	*R* `(` *E* `)`
E	`<= 2`	*S*
E `<=`	`2`	*S*
E `<= 2`	∅	*R n*
E `<=` *E*	∅	*R E* `<=` *E*
E	∅	*acceptation*

2. *La partie* `2 <=` *peut être ou non incluse dans l'expression conditionnelle.*

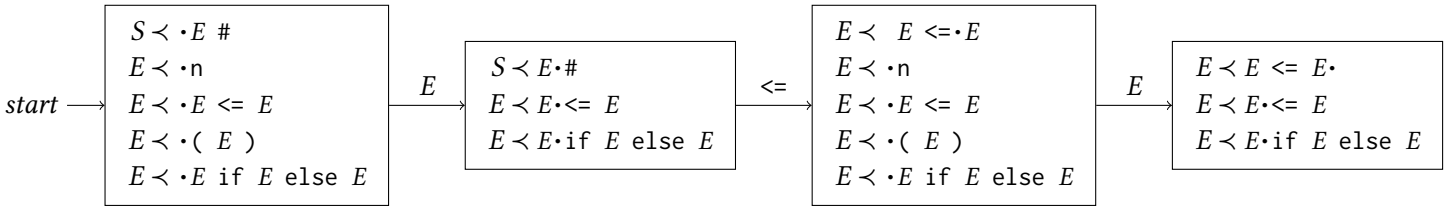


3. *Le résultat indique que l'expression* `0 if True else 1 <= 2` *est interprétée comme* `0 if True else (1 <= 2)`. *Cela correspondrait à la première des deux dérivations précédentes, où la comparaison est intégrée à l'expression conditionnelle. Sur cette expression, l'autre interprétation* `(0 if True else 1) <= 2` *aurait renvoyé* `True`, *car* `0 <= 2`.

4. *La conditionnelle principale peut être celle de gauche ou de droite.*



5. *Les deux interprétations* `1 if False else (2 if True else 3)` *et* `(1 if False else 2) if True else 3` *donnent la valeur* 2. *Exemple plus discriminant :* `1 if True else 2 if False else 3`. *On a alors l'interprétation* `1 if True else (2 if False else 3)` *qui s'évalue en* 1 *et l'interprétation* `(1 if True else 2) if False else 3` *qui s'évalue en* 3.
6. *On se concentre sur le chemin qui mène au choix entre réduire* `E <= E` *et progresser avec* `if`.



□

Exercice 4 (Valeurs maximales) On se donne la syntaxe abstraite minimale suivante pour des expressions arithmétiques.

e	$::=$	1	la constante 1
	$\mid$	x	variable
	$\mid$	$\mathsf{add}(e,e)$	addition
	$\mid$	$\mathsf{mul}(e,e)$	multiplication

Un environnement ρ est une fonction des variables vers les valeurs entières positives. On définit la taille $|e|$ d'une expression e et la valeur $\mathsf{eval}(e,\rho)$ d'une expression e dans un environnement ρ par les équations suivantes.

$$\begin{array}{rcl@{\qquad}rcl} |1| & = & 0 & \mathsf{eval}(1,\rho) & = & 1 \\ |x| & = & 1 & \mathsf{eval}(x,\rho) & = & \rho(x) \\ |\mathsf{add}(e_1,e_2)| & = & 1+|e_1|+|e_2| & \mathsf{eval}(\mathsf{add}(e_1,e_2),\rho) & = & \mathsf{eval}(e_1,\rho)+\mathsf{eval}(e_2,\rho) \\ |\mathsf{mul}(e_1,e_2)| & = & 1+|e_1|+|e_2| & \mathsf{eval}(\mathsf{mul}(e_1,e_2),\rho) & = & \mathsf{eval}(e_1,\rho)\times\mathsf{eval}(e_2,\rho) \end{array}$$

On s'intéresse aux valeurs maximales que peut prendre une expression e, en fonction de sa taille.

Questions.

1. Soit $M \geq 2$ un nombre, et ρ un environnement dont toutes les valeurs sont inférieures ou égales à M. Montrer qu'alors, pour toute expression e on a $\mathsf{eval}(e,\rho) \leq M^{|e|}$. *Indication :* par induction sur e.
2. On s'autorise maintenant une nouvelle forme d'expressions en plus des précédentes

e	$::=$	...	
	$\mid$	$\mathsf{let}\ x = e\ \mathsf{in}\ e$	définition locale

   avec les définitions étendues suivantes de la taille et de l'évaluation.

   $$|\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2| = 1+|e_1|+|e_2|$$
   $$\mathsf{eval}(\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2,\rho) = \mathsf{eval}(e_2,\rho') \quad \text{avec } \rho' \text{ défini par } \begin{cases} \rho'(x) = \mathsf{eval}(e_1,\rho) \\ \rho'(y) = \rho(y) & \text{si } y \neq x \end{cases}$$

   (a) Montrer que la borne établie à la question précédente n'est plus valable avec ces expressions.
   (b) Monter que, maintenant, sous les mêmes conditions pour ρ et M, on a $\mathsf{eval}(e,\rho) \leq M^{2^{|e|}}$.

Correction :

1. *Par induction structurelle sur l'expression e.*
   - *Cas de la constante* 1 *:*

     $$eval(1, \rho) = 1 \leq M^0 = M^{|1|}$$

   - *Cas d'une variable x :*

     $$eval(x, \rho) = \rho(x) \leq M = M^1 = M^{|x|}$$

   - *Cas d'une addition* $add(e_1, e_2)$ *:*

     $$eval(add(e_1, e_2), \rho) = eval(e_1, \rho) + eval(e_2, \rho) \leq M^{|e_1|} + M^{|e_2|} \leq M^{1+|e_1|+|e_2|} = M^{|add(e_1,e_2)|}$$

   - *Cas d'une multiplication* $mul(e_1, e_2)$ *:*

     $$eval(add(e_1, e_2), \rho) = eval(e_1, \rho) \times eval(e_2, \rho) \leq M^{|e_1|} \times M^{|e_2|} \leq M^{1+|e_1|+|e_2|} = M^{|add(e_1,e_2)|}$$

2. *On peut trouver un contre-exemple en enchaînant des définitions locales. L'expression e suivante*

   $e = let\ x_1 = add(1, 1)\ in\ let\ x_2 = mul(x_1, x_1)\ in\ let\ x_3 = mul(x_2, x_2)\ in \quad \ldots \quad let\ x_n = mul(x_{n-1}, x_{n-1})\ in\ x_n$

   vérifie $eval(e, \emptyset) = 2^{2^n}$ *pour tout n, alors que sa taille n'est que linéaire en n.*
   Preuve de la nouvelle borne, par induction structurelle sur e.
   - *Les quatre premiers cas sont similaires à la question précédente.*
   - *Cas d'une définition de variable locale* $let\ x = e_1\ in\ e_2$ *:*

     $$eval(let\ x = e_1\ in\ e_2, \rho) \quad = \quad eval(e_2, \rho')$$

     avec $\rho'(x) = eval(e_1, \rho)$ *et* $\rho'(y) = \rho(y)$ *pour tout* $y \neq x$. *Par hypothèse de récurrence sur* e_1, $eval(e_1, \rho) \leq M^{2^{|e_1|}}$. *Donc les valeurs de l'environnement mis à jour* ρ' *sont bornées par la valeur* $M' = M^{2^{|e_1|}}$ *(on a bien* $M' \geq M$*). Donc par hypothèse de récurrence sur* e_2 *on a*

     $$eval(e_2, \rho') \leq (M')^{2^{|e_2|}} = (2^{2^{|e_1|}})^{2^{|e_2|}} = 2^{2^{|e_1|} \times 2^{|e_2|}} = 2^{2^{|e_1|+|e_2|}} \leq 2^{2^{|let\ x=e_1\ in\ e_2|}}$$