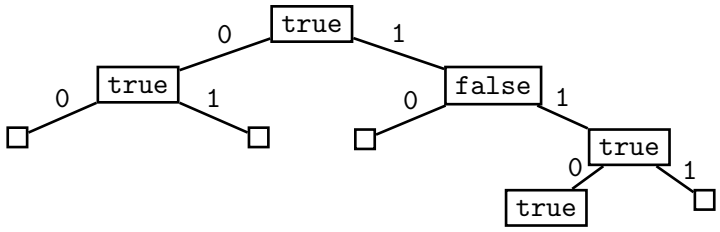
Langages, interprétation, compilation – Partiel

Durée 2 heures. Tous documents autorisés.

Exercice 1. Induction structurelle

La structure de *trie* est un arbre binaire permettant de représenter un ensemble de mots binaires. Pour chaque nœud interne, le fils gauche correspond à la lecture d'un `0` et le fils droit à la lecture d'un `1`. Chaque nœud est associé à un mot : la séquence de `0` et de `1` lue pour atteindre ce nœud en partant de la racine. Chaque nœud interne est également étiqueté par un booléen, indiquant si le mot correspondant est ou non dans l'ensemble représenté. Ainsi, l'arbre ci-dessous représente l'ensemble $\{\varepsilon, \mathtt{0}, \mathtt{11}, \mathtt{110}\}$ (on a omis quelques-uns des sous-arbres vides).



Le type caml suivant permet de représenter un *trie* à l'aide de deux constructeurs : `E` pour un arbre vide et `N` pour un nœud interne, étiqueté par un booléen, avec deux fils.

```
type trie =
  | E
  | N of bool * trie * trie
```

On se donne les équations suivantes pour une fonction in qui indique si une séquence binaire ℓ appartient ou non à un *trie* t :

$$
\begin{array}{rcl}
\mathsf{in}(\ell, \mathtt{E}) & = & \mathtt{false} \\
\mathsf{in}(\varepsilon, \mathtt{N}(b, t_0, t_1)) & = & b \\
\mathsf{in}(\mathtt{0}\ell, \mathtt{N}(b, t_0, t_1)) & = & \mathsf{in}(\ell, t_0) \\
\mathsf{in}(\mathtt{1}\ell, \mathtt{N}(b, t_0, t_1)) & = & \mathsf{in}(\ell, t_1)
\end{array}
$$

Questions.

1. Écrire le terme représentant le *trie* dessiné en introduction de l'exercice.
2. Définir une fonction size déterminant le cardinal de l'ensemble représenté par un *trie* t.
   Vous pouvez répondre au choix par des équations ou par du code caml.
3. Définir une fonction union qui calcule l'union de deux *tries* t_1 et t_2 donnés en paramètres.
   Vous pouvez répondre au choix par des équations ou par du code caml.
4. Démontrer que pour tous *tries* t_1 et t_2 on a $\mathsf{size}(t_1) \leq \mathsf{size}(\mathsf{union}(t_1, t_2)) \leq \mathsf{size}(t_1) + \mathsf{size}(t_2)$.
5. Démontrer que pour tous *tries* t_1 et t_2 et toute séquence ℓ, si $\mathsf{in}(\ell, t_1)$ alors $\mathsf{in}(\ell, \mathsf{union}(t_1, t_2))$.

Correction.

1.
```
N(true, N(true, E, E),
        N(false, E,
                 N(true, N(true, E, E), E)))
```

2.

$$
\begin{array}{rcl}
\mathsf{size}(\mathtt{E}) & = & 0 \\
\mathsf{size}(\mathtt{N}(\mathtt{true}, t_1, t_2)) & = & 1 + \mathsf{size}(t_1) + \mathsf{size}(t_2) \\
\mathsf{size}(\mathtt{N}(\mathtt{false}, t_1, t_2)) & = & \mathsf{size}(t_1) + \mathsf{size}(t_2)
\end{array}
$$

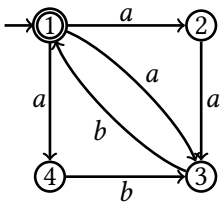
3.

$$\begin{array}{rcl} \mathsf{union}(\mathtt{E}, t_2) & = & t_2 \\ \mathsf{union}(t_1, \mathtt{E}) & = & t_1 \\ \mathsf{union}(\mathtt{N}(b_1, t_1', t_1''), \mathtt{N}(b_2, t_2', t_2'')) & = & \mathtt{N}(b_1 \vee b_2, \mathsf{union}(t_1', t_2'), \mathsf{union}(t_1'', t_2'')) \end{array}$$

4. Par induction structurelle sur t_1.
   - Cas $t_1 = \mathtt{E}$. Alors $\mathsf{size}(\mathsf{union}(\mathtt{E}, t_2)) = \mathsf{size}(t_2) = \mathsf{size}(\mathtt{E}) + \mathsf{size}(t_2)$.
   - Cas $t_1 = \mathtt{N}(b_1, t_1', t_1'')$. Hypothèses de récurrence :
     $\mathsf{size}(t_1') \leq \mathsf{size}(\mathsf{union}(t_1', t_2')) \leq \mathsf{size}(t_1') + \mathsf{size}(t_2')$ (pour tout t_2') et
     $\mathsf{size}(t_1'') \leq \mathsf{size}(\mathsf{union}(t_1'', t_2'')) \leq \mathsf{size}(t_1'') + \mathsf{size}(t_2'')$ (pour tout t_2'').
     On raisonne par cas selon la forme de t_2.
     - Si $t_2 = \mathtt{E}$ alors $\mathsf{size}(\mathsf{union}(t_1, t_2)) = \mathsf{size}(t_1)$ et la conclusion est immédiate.
     - Si $t_2 = \mathtt{N}(b_2, t_2', t_2'')$, alors par définition on a
       $\mathsf{size}(\mathsf{union}(t_1, t_2)) = \mathtt{N}(b_1 \vee b_2, \mathsf{union}(t_1', t_2'), \mathsf{union}(t_1'', t_2'')) = x + \mathsf{size}(\mathsf{union}(t_1', t_2')) + \mathsf{size}(\mathsf{union}(t_1'', t_2''))$,
       avec $x = 1$ si b_1 ou b_2 est vrai, et $x = 0$ sinon. Les hypothèses de récurrence donnent donc
       $x + \mathsf{size}(t_1') + \mathsf{size}(t_1'') \leq \mathsf{size}(\mathsf{union}(t_1, t_2)) \leq x + \mathsf{size}(t_1') + \mathsf{size}(t_2') + \mathsf{size}(t_1'') + \mathsf{size}(t_2'')$
       et on peut conclure car, par définition, $\mathsf{size}(\mathtt{N}(b_1, t_1', t_1'')) \leq x + \mathsf{size}(t_1') + \mathsf{size}(t_1'')$ et
       $x + \mathsf{size}(t_1') + \mathsf{size}(t_2') + \mathsf{size}(t_1'') + \mathsf{size}(t_2'') \leq \mathsf{size}(\mathtt{N}(b_1, t_1', t_1'')) + \mathsf{size}(\mathtt{N}(b_2, t_2', t_2''))$.
5. Par induction structurelle sur t_1.
   - Cas $t_1 = \mathtt{E}$. Il n'existe pas de ℓ tel que $\mathsf{in}(\ell, \mathtt{E})$: propriété vraie par vacuité.
   - Cas $t_1 = \mathtt{N}(b_1, t_1', t_1'')$. Hypothèses de récurrence : si $\mathsf{in}(\ell, t_1')$ alors $\mathsf{in}(\ell, \mathsf{union}(t_1', t_2'))$ (pour tout t_2') et si $\mathsf{in}(\ell, t_1'')$ alors $\mathsf{in}(\ell, \mathsf{union}(t_1'', t_2''))$ (pour tout t_2''). Par cas sur la forme de t_2.
     - Si $t_2 = \mathtt{E}$, alors $\mathsf{union}(t_1, t_2) = t_1$ et la propriété est immédiatement vraie.
     - Si $t_2 = \mathtt{N}(b_2, t_2', t_2'')$. Alors $\mathsf{union}(t_1, t_2) = \mathtt{N}(b_1 \vee b_2, \mathsf{union}(t_1', t_2'), \mathsf{union}(t_1'', t_2''))$. Soit ℓ tel que $\mathsf{in}(\ell, t_1)$. Par cas sur ℓ.
       - Si $\ell = \varepsilon$. Comme on suppose que $\mathsf{in}(\ell, t_1)$, on a nécessairement $b_1 = \mathtt{true}$. Donc $b_1 \vee b_2 = \mathtt{true}$ et on a bien $\mathsf{in}(\ell, \mathsf{union}(t_1, t_2))$.
       - Si $\ell = \mathtt{0}\ell'$. Comme on suppose qu $\mathsf{in}(\mathtt{0}\ell', t_1)$, on a nécessairement $\mathsf{in}(\ell', t_1')$. Par hypothèse de récurrence on a alors $\mathsf{in}(\ell', \mathsf{union}(t_1', t_2'))$, et donc $\mathsf{in}(\mathtt{0}\ell', \mathsf{union}(t_1, t_2))$.
       - Si $\ell = \mathtt{1}\ell'$: cas identique au précédent, mais sur le sous-arbre de droite.

Exercice 2. Automates

On considère l'automate suivant sur l'alphabet $\{a, b\}$.

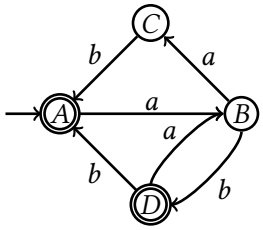


Questions.

1. Déterminiser l'automate.
2. Donner une expression régulière décrivant le langage reconnu par l'automate.

Correction.

1. À gauche, l'automate déterminisé, à droite la correspondance entre les états de l'automate déterminisé et des ensembles d'états de l'automate de départ.



A	$\{1\}$
B	$\{2,3,4\}$
C	$\{3\}$
D	$\{1,3\}$

2. L'automate reconnaît n'importe quelle répétition des motifs ab, aab et abb. D'où l'expression $(ab|aab|abb)^*$.

Exercice 3. Langages réguliers

Les langages suivants sur l'alphabet $\{a, b\}$ sont-ils reconnaissables ? Si oui, donner une expression régulière ou un automate les reconnaissant. Si non, démontrer que ce n'est pas le cas.

L_1 : l'ensemble des mots contenant au plus 4 occurrences de a.

L_2 : l'ensemble des mots contenant un nombre impair d'occurrences disjointes du motif bb.

L_3 : l'ensemble des mots dont la longueur est un carré.

L_4 : l'ensemble des mots dont le nombre de a est égal à k modulo n, pour deux constantes $0 \leq k < n$ arbitraires fixées.

L_5 : l'ensemble des mots « redoublés », c'est-à-dire qui peuvent s'écrire comme la concaténation mm d'un mot m avec lui-même.

Correction.

1. Reconnaissable. Justification : expression $b^*(\varepsilon|a)b^*(\varepsilon|a)b^*(\varepsilon|a)b^*(\varepsilon|a)b^*$, ou automate formé par une ligne de 6 états où chaque a fait progresser d'un état au suivant, chaque b fait rester sur place, et seuls les 5 premiers états sont acceptants.
2. Reconnaissable. Justification : expression $(a|ba)^*bb(a|ba)^*\,[bb(a|ba)^*bb(a|ba)^*]^*$, ou automate formé par quatre états 1, 2, 3, 4 où 1 est initial, 3 acceptant, et où on a les transitions

$$\begin{array}{ll} 1 \xrightarrow{b} 2 & 1 \xrightarrow{a} 1 \\ 2 \xrightarrow{b} 3 & 2 \xrightarrow{a} 1 \\ 3 \xrightarrow{b} 4 & 3 \xrightarrow{a} 3 \\ 4 \xrightarrow{b} 1 & 4 \xrightarrow{a} 3 \end{array}$$

   Note : d'autres interprétations de l'énoncé seront acceptées.
3. Non reconnaissable. Justification en raisonnant par l'absurde : supposons que L_3 soit reconnaissable. Alors par le lemme de l'étoile il existe un N tel que... On considère le mot a^{N^2}. Comme $N^2 \geq N$, il existe une décomposition $a^{N^2} = a^{k_1}a^{k_2}a^{k_3}$ avec $k_2 \neq 0$ telle que pour tout n on a $a^{k_1}a^{nk_2}a^{k_3} \in L_3$. Or il existe un n tel que $k_1 + nk_2 + k_3$ n'est pas un carré. Argument de cardinalité pour l'existence de n : pour K grand, l'intervalle $[0, K[$ contient de l'ordre de $\sqrt{K}$ nombres qui sont des carrés, et de l'ordre de $\frac{K}{k_2}$ de nombres de la forme $k_1 + nk_2 + k_3$, avec $\sqrt{K} = o(\frac{K}{k_2})$. Contradiction. Donc le langage n'est pas reconnaissable.
4. Reconnaissable. Justification : expression $b^*(ab^*ab^*\ldots)(ab^*ab^*\ldots)^*$ où la première séquence avec ... contient k occurrences de ab^* et la deuxième en contient n, ou automate en forme de cycle de longueur n, où chaque a fait progresser à l'état suivant et chaque b fait rester sur place, et où seul l'état numéro $k+1$ est acceptant. Note : tout ceci fonctionne uniquement car k et n sont *fixés*.

5. Non reconnaissable. Justification en raisonnant par l'absurde : supposons que L_5 soit reconnaissable. Alors par le lemme de l'étoile *forte* il existe un N tel que... On considère le mot $a^N b a^N b$. Son premier préfixe a^N a une longueur $\geq N$. On a donc une décomposition $a^N = a^{k_1} a^{k_2} a^{k_3}$ avec $k_2 \neq 0$ telle que pour tout n on a $a^{k_1} a^{nk_2} a^{k_3} b a^N b \in L_5$. Or $a^{k_1} a^{0k_2} a^{k_3} b a^N b \notin L_5$. Contradiction. Le langage n'est pas reconnaissable.

Exercice 4. Grammaires

On s'intéresse à des grammaires pour des expressions contenant des conditionnelles avec ou sans `else`. On considère les symboles terminaux `x` (pour une variable) et `if`, `then`, `else`. E et E' sont les symboles non terminaux, et E est le symbole de départ.

G_1

$$\begin{array}{lcl} E & \to & \texttt{x} \\ & \mid & \texttt{if } E \texttt{ then } E \\ & \mid & \texttt{if } E \texttt{ then } E \texttt{ else } E \end{array}$$

G_2

$$\begin{array}{lcl} E & \to & \texttt{x} \\ & \mid & \texttt{if } E \texttt{ then } E\ E' \\ E' & \to & \varepsilon \\ & \mid & \texttt{else } E \end{array}$$

G_3

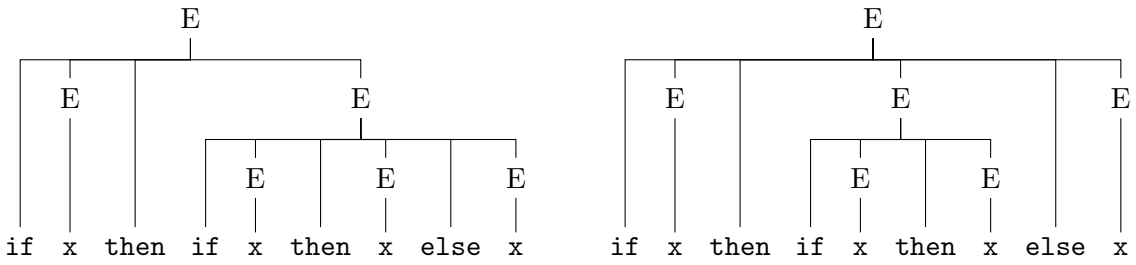
$$\begin{array}{lcl} E & \to & \texttt{if } E \texttt{ then } E \\ & \mid & E' \\ E' & \to & \texttt{x} \\ & \mid & \texttt{if } E \texttt{ then } E' \texttt{ else } E' \end{array}$$

Questions

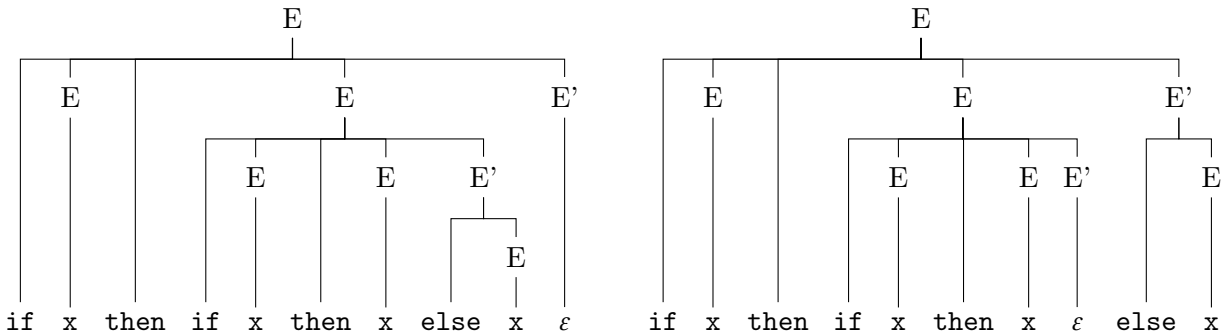
1. Donner deux dérivations distinctes pour la phrase `if x then if x then x else x` dans la grammaire G_1, de manière à déduire que cette grammaire est ambiguë.
2. Montrer que la grammaire G_2 ne corrige pas le problème.
3. Montrer que la grammaire G_3 est incomplète, c'est-à-dire qu'elle ne permet pas de dériver certaines phrases qui sont dérivables par G_1.
4. Proposer une manière de corriger G_3 de sorte à ce qu'elle permette de dériver toute phrase dérivable avec G_1, sans être ambiguë.
5. Ce problème d'ambiguïté des conditionnelles a été écrit en utilisant une syntaxe « à la caml ». Existe-t-il aussi en Python ? En C ou en Java ?

Correction.

1. Deux arbres de dérivation distincts, qui se distinguent par le `if` auquel se rapporte l'unique `else`.



2. On a les deux mêmes interprétations possibles, avec le `else` se rapportant à l'un ou l'autre des `if`.



3. La phrase `if x then x else if x then x` n'est pas dérivable dans G_3. En effet, la seule règle de G_3 permettant de dériver un `else` impose que celui-ci soit suivi par une phrase E', et la phrase `if x then x` n'est pas dérivable à partir de E'. Note : cette solution associe nécessairement chaque nouveau `else` au `if` encore ouvert le plus proche.
4. Il suffit d'ajouter la troisième règle $E \rightarrow$ `if` E `then` E' `else` E pour le symbole non terminal E. Interprétation de ces règles : E désigne une phrase conditionnelle quelconque, et E' une phrase conditionnelle qui ne peut plus être complétée par un `else`.
5. Ce problème n'existe pas lorsque les blocs d'instructions sont délimités de manière explicite. C'est nécessairement le cas en python puisque les blocs sont définis par l'indentation. En C ou en java, on ne voit pas le problème si l'on utilise systématiquement les accolades pour délimiter les blocs, mais il existe néanmoins car ces langages permettent d'omettre les accolades lorsqu'un bloc ne contient qu'une seule instruction.

   ```
   if (x) if (x) { ... } else { ... }
   ```

   Note : en ocaml, on peut délimiter explicitement des blocs avec les parenthèses ou avec les mots-clés `begin` et `end`.

Exercice 5. Interprétation

Vous trouverez en dernière page quatre programmes Java et des indications de l'effet de chacun.

1. Qu'est-ce que ces exemples nous montrent des règles régissant les variables et les attributs en Java ? Comparer avec d'autres langages que vous connaissez, comme Caml ou Python (la fin de la page 3 donne deux exemples de programmes que vous pouvez considérer).
2. Si l'on souhaitait réaliser un interprète pour des programmes Java :
   (a) quelle(s) structure(s) de données pourrait-on utiliser pour réaliser l'environnement ?
   (b) quelle(s) opérations réaliserait-on lors de l'introduction d'une nouvelle variable ? lors d'une affecation ?
   (c) quelle(s) opérations réaliserait-on lors de l'entrée dans un bloc ? lors de la sortie ?

   Votre réponse peut montrer du code mais ce n'est pas indispensable. Elle doit surtout montrer en quoi la stratégie retenue réalise la bonne sémantique.

Correction.

1. `Test1` montre que l'effet d'une affectation à une variable locale réalisée à l'intérieur d'un bloc est observable depuis l'extérieur de ce bloc. C'est identique à ce qui se passe en python (ou en caml avec des références).

   `Test2` montre que deux variables locales d'une même méthode ne peuvent pas avoir le même nom. C'est contraire à ce qui se passe en caml, où on peut introduire une nouvelle variable de même nom qu'une variable déjà présente (la nouvelle masquant l'ancienne dans sa portée). À noter dans l'exemple caml : la valeur `x=2` n'est pas visible au moment de l'appel `print_int x`, cette variable locale n'existant que dans la ligne `in 1`.

   `Test3` et `Test4` montrent qu'un attribut d'une classe et une variable locale d'une méthode de cette classe peuvent en revanche avoir le même nom. C'est aussi le cas en python (où les accès aux attributs se font toujours explicitement sous la forme `self.x`). `Test3` et `Test4` montrent également que la portée d'une variable locale est le bloc dans lequel elle est définie (cohérent avec caml mais pas complètement avec python), et que dans cette portée la variable locale masque un éventuel élément extérieur de même nom (cohérent avec caml et avec python) (dans `Test4` l'élément extérieur est un attribut de la classe).
2. Une proposition de stratégie de réalisation d'un environnement.
   (a) Une table de hachage pour chaque portée : une pour les attributs, une pour chaque méthode, une pour chaque bloc à l'intérieur d'une méthode, etc. On inclut un chaînage de chaque table vers la table de la portée englobante. Lorsque l'on cherche une variable, on commence par la table la plus locale, puis on remonte la chaîne tant que l'on n'a pas trouvé.

(b) Lors de l'introduction d'une nouvelle variable : vérifier qu'elle n'existe pas déjà dans la chaîne (à part éventuellement dans le dernier niveau, contenant les attributs), puis l'inclure dans la table du bloc en cours. Lors d'une affectation : modifier l'association pour la variable dans la première table où elle est trouvée.

(c) Lors de l'entrée dans un bloc : on crée une nouvelle table avec un chaînage vers la table courante. Lors de la sortie d'un bloc : on défausse la table courante (en laissant faire le GC le cas échéant) et on reprend la table englobante comme nouvelle table courante.

À noter : avec cette stratégie, le coût de l'accès à une variable dépend de la profondeur d'emboîtement des blocs. On peut faire plus efficace avec une unique table de hachage si l'on dispose comme en caml de trois fonctions `add` (qui ajoute une nouvelle association qui masque une éventuelle association précédente pour la même clé sans la détruire), `update` (qui modifie une association précédente plutôt que de la masquer) et `remove` (qui efface la dernière association pour une clé donnée et rend à nouveau visible l'éventuelle association précédente). À défaut, pour s'en tirer avec une tableau de hachage unique il faut ajouter à l'entrée (resp. à la sortie) de chaque nouvelle portée des sauvegardes (resp. des restaurations) de toutes les variables masquées par cette portée locale.

Annexe. Programmes Java pour l'exercice 5

Rappels de syntaxe Java : les accolades délimitent un bloc à l'intérieur d'un fragment de code. À l'intérieur d'une fonction, une déclaration `int x = 1;` déclare une variable locale (et l'initialise). Une déclaration `public static int x = 1;` (dans une classe, mais à l'extérieur de toute fonction) déclare un « attribut de classe », assimilable à une variable globale.

```
class Test1 {
  public static void main(String[] args) {
    int x = 1;
    { x = 2; }
    System.out.println(x);
  }
} // Affiche 2
```

```
class Test2 {
  public static void main(String[] args) {
    int x = 1;
    { int x = 2; }
    System.out.println(x);
  }
}
/* Rejeté par le compilateur avec le message suivant.
test.java:5: error: variable x is already defined in method main(String[])
       { int x = 2; }
             ^
*/
```

```
class Test3 {
  public static int x = 1;
  public static void main(String[] args) {
    { int x = 2; }
    System.out.println(x);
  }
} // Affiche 1
```

```
class Test4 {
  public static int x = 1;
  public static void main(String[] args) {
    int x = 2;
    System.out.println(x);
  }
} // Affiche 2
```

```
let exempleCaml () =
  let x = let x = 2
          in 1
  in print_int x
```

```
def exemplePython():
  x = 1
  for i in range(3): x = i
  print(x)
```