Préparation aux oraux CCINP n°10

Grammaires

Rappelons qu'une grammaire est un quadruplet $\mathcal{G} = (V, \Gamma, \Pi, S)$, où V désigne l'alphabet des symboles non terminaux (ou variables), Γ l'alphabet des symboles terminaux, $S \in V$ le symbole initial et Π l'ensemble des règles de production, de la forme $X \to \sigma$ avec $X \in V$ et $\sigma \in (V \cup \Gamma)^\star$.

Représentons une grammaire en OCaml ! On supposera pouvoir numéroter les éléments de V, ils seront donc représentés par un entier. Les symboles terminaux seront représentés par des chaînes de caractères. Un ensemble sera représenté par une liste sans répétition d'éléments. On définit donc le type `grammar` comme dans le code donné.

Q1. Représenter, en OCaml, une grammaire reconnaissant les listes OCaml de booléens comme

`[]` , `[true]` et `[true; false; true]`

mais pas

`[not false]` et `true :: [false]`.

Q2. Expliquer le comportement de la fonction `next` définie dans le code donné. On pourra executer cette fonction plusieurs fois.

Q3. Écrire une fonction OCaml `cfg_of_reg : regexp -> grammar` [1] qui transforme une expression régulière $e \in \mathrm{Reg}(\Gamma)$ en une grammaire $\mathcal{G}$ telle que $\mathscr{L}(e) = \mathscr{L}(\mathcal{G})$.

Définition. Une grammaire $\mathcal{G} = (V, \Gamma, \Pi, S)$ est sous *forme normale de Chomsky* dès lors que les règles de production de cette grammaire sont de la forme :
- $A \to a$;
- $A \to BC$ avec $B \neq S \neq C$;
- $S \to \varepsilon$,

où A, B et C sont des variables, et a est un terminal.

On admet que toute grammaire est *faiblement équivalente* (*i.e.* a le même langage) à une grammaire sous forme normale de Chomsky. Cette conversion se fait en $\mathrm{O}(n^2)$, où n est la « taille » de la grammaire, c'est-à-dire, la longueur de sa sérialisation.

On représente en OCaml une formule sous forme normale de Chomsky par le type `grammar_chomsky`.

Q4. Trouver une grammaire sous forme normale de Chomsky répondant à la question Q1. La définir en OCaml à l'aide de ce nouveau type.

Q5. En utilisant l'algorithme de la Fig. 1 (page 2), corriger la fonction OCaml ayant pour signature `membership : grammar_chomsky -> string -> bool` qui, pour une grammaire $\mathcal{G}$ et un mot w donnés, vérifie si $w \in \mathscr{L}(\mathcal{G})$.

Q6. L'algorithme de la Fig. 1 (page 2) ressemble à un algorithme de graphe, lequel ?

Étant donné un langage $L \subseteq \Sigma^\star$, on définit le problème $\textsc{Appartient}_L$ comme :

$\textsc{Appartient}_L$:	**Entrée.** Un mot $w \in \Sigma^\star$. **Sortie.** Est-ce que $w \in L$?

[1] CFG signifie *Context Free Grammar*, c'est-à-dire grammaire non contextuelle.

Q7. Quelle est la complexité de l'algorithme de la Fig. 1 ? En déduire que $\textsc{Appartient}_L \in \mathbf{P}$, où le langage L est non contextuel (c'est-à-dire, le langage d'une grammaire non contextuelle).

Entrée. Une grammaire $\mathcal{G} = (V, \Gamma, \Pi, S)$ sous forme normale de Chomsky et $w \in \Gamma^\star$.
Sortie. Le mot w est-il reconnu par $\mathcal{G}$?

Si $w = \varepsilon$ et $(S \to \varepsilon) \in \Pi$ **alors Renvoyer** VRAI ▷ *On s'occupe du cas $w = \varepsilon$.*

On note $w = w_1 \dots w_n$, où n est la longueur de w.
On initialise une matrice tab de taille $n \times n$, initialisé à $\emptyset$.

Pour $i = 1$ **à** n **faire** ▷ *On regarde les lettres dans w,* i.e. *les sous-mots de longueur* 1.
  Pour $A \in V$ **faire**
    Si $(A \to w_i) \in \Pi$ **alors**
      $\mathrm{tab}[i, i] \leftarrow \mathrm{tab}[i, i] \cup \{A\}$
    Fin si
  Fin pour
Fin pour

Pour $\ell = 2$ **à** n **faire** ▷ *On regarde les sous-mots de longueur ℓ.*
  Pour $i = 1$ **à** $n - \ell - 1$ **faire** ▷ *Indice de début*
    $j \leftarrow i + \ell - 1$ ▷ *Indice de fin*
    Pour $k = i$ **à** $j - 1$ **faire** ▷ *Indice de découpe*
      Pour $(A \to BC) \in \Pi$ **faire**
        Si $B \in \mathrm{tab}[i, k]$ et $C \in \mathrm{tab}[k + 1, j]$ **alors**
          $\mathrm{tab}[i, j] \leftarrow \mathrm{tab}[i, j] \cup \{A\}$
        Fin si
      Fin pour
    Fin pour
  Fin pour
Fin pour

Renvoyer $S \overset{?}{\in} \mathrm{tab}[1, n]$

Fig. 1. Algorithme CYK (Cocke, Younger et Kasami)