Compilation - corrigé du TD2

Analyse lexicale, fin
Analyse syntaxique, début

1 un AFD directement à partir d'une expression régulière

Question 1-1

Je n'ai pas le temps de faire les dessins, ceux faits en TD étaient justes. Les désespérés peuvent aller emprunter un Dragon, certains dessins y sont, pour le même exemple (car la première année, le TDman n'est pas téméraire).

Question 1-2

Pour obtenir un automate déterministe, il suffit de trouver un algo de construction des nouveaux états qui énumère les lettres possibles en partant d'un état, et ne construit qu'une transition par lettre. En voici un[1].

Les états seront des ensembles de *positions suivantes possibles* (PSP). On part d'un état S_0 qui contient l'ensemble des PSP de la racine[2]. Cet état est marqué "non traité".

Ensuite on a une boucle : tant qu'il reste des états non traités, en choisir un (appelons le S), et lui faire subir la manip suivante.

- Le marquer "traité" pour ne pas le faire plusieurs fois.
- Construire l'ensemble des symboles de l'alphabet correspondant aux positions de S (fastoche).
- Pour chacun de ces symboles (par exemple a), on veut construire l'arc de l'automate étiqueté par ce symbole. Dans quel état arrive cet arc ? Dans un état étiqueté par l'ensemble des *positions suivantes possibles* une fois qu'on a mangé ce symbole.
- Pour construire cet ensemble, on regarde chacune des positions de S qui correspond à ce symbole a, et on fait l'union de leurs ensembles de PSP.
- Si cette union est déja un de nos états, on met juste la transition dessus. Sinon, on crée un nouvel état, on le marque "non traité", et on l'ajoute à notre ensemble d'états de l'automate.

Question 1-3

On construit facilement ANN (en premier), puis PP et DP par induction sur l'arbre A :

A	ANN(A)	PP(A)	DP(A)
ϵ	Vrai	$\emptyset$	$\emptyset$
$c \neq \epsilon$, position i	Faux	{i}	{i}
$A_1 \mid A_2$	ANN(A_1) ou ANN(A_2)	PP(A_1) $\cup$ PP(A_2)	DP(A_1) $\cup$ DP(A_2)
A_1*	Vrai	PP(A_1)	DP(A_1)
$A_1.A_2$	ANN(A_1) et ANN(A_2)	Si ANN(A_1) alors PP(A_1) $\cup$ PP(A_2) sinon PP(A_1)	Si ANN(A_2) alors DP(A_1) $\cup$ DP(A_2) sinon DP(A_2)

[1] La bonne question à se poser est la suivante : n'est-ce pas la compression en une seule passe de l'algo intuitif qui construit d'abord un automate non déterministe pour le déterminiser ensuite.

[2] Au fait, l'ensemble des PSP de la racine, c'est son ensemble de premières positions qu'on calcule un peu plus bas...

Pour la construction de l'ensemble POSITION_SUIVANTE de chaque feuille, il doit suffire d'appliquer les deux règles suivantes.

- Pour un nœud $A_1.A_2$, pour chaque position $i \in \mathrm{DP}(A_1)$, toutes les positions de $\mathrm{PP}(A_2)$ doivent faire partie de POSITION_SUIVANTE(i).
- Pour un nœud A_1*, pour chaque position $i \in \mathrm{DP}(A_1)$, toutes les positions de $\mathrm{PP}(A_1)$ sont dans POSITION_SUIVANTE(i).

Si on a étiqueté son arbre auparavant avec PP, DP et ANN, on voit qu'on peut faire un parcours de l'arbre en partant de la racine et en accumulant les POSITION_SUIVANTE(i) dans un tableau.

2 Grammaires

Question 2-1 Les expressions régulières ne sont pas récursives.

Question 2-3 J'espère que cela ne vous mélange pas trop les pinceaux que nous ayons pris la grammaire des expressions régulières comme exemple de grammaire...

$$\begin{array}{lcl} R & \to & R \vee T \mid T \\ T & \to & TD \mid D \\ D & \to & L* \mid L \\ L & \to & a \mid b \mid (R) \end{array}$$

Question 2-5

La difficulté est de permettre plus de parenthèses ouvrantes que fermantes. La première idée est d'écrire la grammaire suivante, mais celle-ci est ambigüe[3] :

$$\begin{array}{lcl} E & \to & [T] \mid T \\ T & \to & (E) \mid (E \end{array}$$

La version suivante est non ambigüe (il y en a d'autres) :

$$\begin{array}{lcl} E & \to & [N] \mid [E] \mid (E) \mid \epsilon \\ N & \to & (N \mid E \end{array}$$

De nos jours, ce genre de "convention" est considérée comme une Très Mauvaise Chose dans un langage de programmation...

Question 2-6 On est parti sur une mauvaise piste en croyant que le problème était la boucle infinie et improductive dans $A \to A\,B$.

L'ambigüité d'une grammaire étant indécidable, il faut renoncer à un algorithme qui détecterait et supprimerait les cycles infinis ambigus, dont l'archétype est $A \to A \mid \epsilon$.

Par contre il y a un non-terminal *inutile* : A. Formellement un non-terminal A est inutile s'il n'apparaît dans aucune dérivation donnant une séquence terminale, de type

$$L \to^* xAy \to^* xwy$$

Ceci est plus facile à détecter.

[3] Au fait, j'ai demandé à Marianne, eh bien l'ambigüité d'une grammaire est indécidable.