Programme de colle - Semaine 22 (semaine du 16 mars au 20 mars)

La démonstration des énoncés marqués d'une étoile est exigible

1 Grammaires et langages non-contextuels

- Définition d'une grammaire non-contextuelle (= hors-contexte = algébrique). Symboles non-terminaux (notés en majuscule) et terminaux (notés en minuscules). Règles de production de la forme $X \to u$.
- Dérivation immédiate $u \Rightarrow v$, dérivation $u \Rightarrow^* v$, dérivation gauche, dérivation droite.
- Langage engendré par une grammaire. Langages non-contextuels (= langages algébriques).
- **Les langages réguliers sont non-contextuels (*)** (mais l'inverse n'est pas toujours vrai.) : preuve par induction sur les langages réguliers.
- Arbres de dérivation (aussi appelés arbres d'analyse). $X \Rightarrow^* u$ si et seulement s'il existe un arbre de dérivation de racine X et dont la concaténation de feuilles forme le mot u.
- Savoir faire en pratique : passer d'une suite de dérivations à un arbre de dérivation; passer d'un arbre de dérivation à une suite de dérivations immédiates (par parcours en profondeur de l'arbre de dérivation).
- Équivalence de $X \Rightarrow^* u$, $X \Rightarrow_g^* u$ et $X \Rightarrow_d^* u$.
- Grammaire ambiguë : grammaire pour laquelle il existe un mot $u \in \Sigma^*$ admettant deux arbres de dérivation distincts. Ou de manière équivalente : admettant deux suite de dérivations gauches $S \Rightarrow_g^* u$ distinctes.
- Analyse syntaxique : aucune connaissance sur la théorie des analyseurs syntaxiques n'est exigible; mais les élèves doivent pouvoir écrire à la main un petit analyseur syntaxique pour une grammaire très simple (par exemple un langage balisé).
- Exemples traités en cours :
  - **$S \to aSb \mid \varepsilon$ qui engendre le langage $\{a^n b^n, n \in \mathbb{N}\}$ (*)** (démonstration par double inclusion : par récurrence sur la longueur de la dérivation dans un sens, par récurrence sur la longueur du mot dans l'autre sens)
  - $S \to aSbS \mid \varepsilon$ qui engendre le langage des mots bien parenthésés (langage de Dyck).
  - Exemples pratiques : formules propositionnelles, expressions arithmétiques
  - Grammaire pour le langage $L_1 \cup L_2 = \{a^n b^n c^m, (n,m) \in \mathbb{N}^2\} \cup \{a^n b^m c^m, (n,m) \in \mathbb{N}^2\}$ qui est ambiguë.
  - Le problème du *sinon pendant* : langage avec des `if` admettant un `else` optionnel. Cela conduit à une grammaire ambiguë.
- **À savoir faire :**
  - Reconnaître le langage engendré par une grammaire.
  - Écrire une grammaire pour engendrer un langage.
  - Démontrer que le langage engendré par une grammaire G est le langage L (preuve par double inclusion).

  - Raisonner par récurrence sur la longueur d'une dérivation.
  - Raisonner par induction sur les arbres de dérivation (priviligier dans ce cas des grammaires simples ayant un seul symbole non terminal)
  - Justifier qu'une grammaire est ambiguë en trouvant un exemple de mot admettant deux arbres d'analyse distincts.
- **Hors programme :**
  - Formes normales pour les grammaires (mais un exercice guidé peut être donné)
  - Analyseurs syntaxiques ($LL(k)$, $LR(k)$, $LALR$, ...)
  - Lemme d'Ogden
  - Grammaires contextuelles
  - Automates à piles

2 Composantes fortement connexes

- Rappels : définition de composante connexe dans un graphe non orienté.
- Algorithme de parcours pour calculer les composantes connexes.
- Composantes fortement connexes pour un graphe orienté : définies comme les classes d'équivalence de la relation $x \mathcal{R} y$ s'il existe un chemin de x à y et de y à x. **Savoir démontrer que $\mathcal{R}$ est une relation d'équivalence (*).**
- Algorithme du tri topologique : on classe les sommets par inverse du post-ordre d'un parcours en profondeur. On note $\leq_T$ l'ordre total obtenu sur les sommets.
- La succession d'événements de début/fin d'exploration forme un mot bien parenthésé.
- **Dans un DAG (directed acyclic graph) : $\leq_T$ donne un *ordre topologique* (c'est-à-dire que (x, y) est un arc implique $x \leq_T y$). (*)**
- **Algorithme de Kosaraju** pour le calcul des composantes fortement connexes. Complexité linéaire.
- **À savoir faire :**
  - Calculer les composantes connexes d'un graphe non orienté avec un algorithme de parcours.
  - Appliquer l'algorithme du tri topologique sur un graphe orienté quelconque.
  - Calculer les composantes fortement connexes d'un graphe orienté avec l'algorithme de Kosaraju.

3 Programmation concurrente

le programme suppose que l'on travaille sur une machine mono-coeur exécutant des programmes multi-threadés qui s'entrelacent. On fait l'hypothèse de la cohérence séquentielle et que les lectures et écritures en mémoire sont atomiques. Les exemples de cours ont été traités en langage C.

Pas de démonstration à apprendre pour cette semaine, mais il faut maîtriser les concepts suivants :

- Principes de la programmation concurrente : fils d'exécution (threads), entrelacement des processus, opérations `create` et `join` pour un thread.
- Dangers de la programmation concurrente : accès concurrent (race condition), section critique de code, interblocage, famine.
- Protection des sections critiques à l'aide de zones d'exclusion mutuelle implémentées à l'aide de **verrous** (**mutex**), opérations `lock` et `unlock`
- **Sémaphores** : définition, opération V (`sem_post` en C) et P (`sem_wait` en C), applications (verrous, jauges, communication par signaux, ...)
- **Algorithme de Peterson** : pour une exclusion mutuelle algorithmique de deux **threads** par une attente active. L'algorithme de Peterson vérifie les propriétés d'exclusion mutuelle et d'absence de famine.
- **À savoir faire :**
  - Ecrire un programme utilisant plusieurs threads.
  - Reconnaître une situtaion d'accès concurrent et protéger les sections critiques avec des mutex.
  - Utiliser les sémaphores. Applications vues en cours : utilisation comme mutex, comme jauge ou pour envoyer des signaux de synchronisation.
  - Si l'exercice le demande : écrire une zone d'exclusion mutuelles avec l'algorithme de Peterson.

Aucune connaissance précise des bibliothèques `C` et `OCaml` n'est exigible, mais vous devez connaître les opérations existantes et leur but.