Programme de colle - Semaine 21 (21 mars)

La démonstration des énoncés marqués d'une étoile est exigible

1 Composantes fortement connexes

- Rappels sur la connexité dans les graphes orientés ou non : relation d'accessibilité, composantes connexes, composantes fortement connexes.
- Ordre topologique sur un graphe orienté acyclique (DAG). Généralisation au cas des graphes orientés quelconques (on parle d'ordre pré-topologique) Si un sommet x est situé avant y dans un ordre pré-topologique et s'il existe un chemin de y à x alors il existe un chemin de x à y.
- Algorithme de Kosaraju (démonstration non exigible).
- Problème 2-SAT : graphe d'implications d'une instance de 2-SAT. Une instance de 2-SAT est positive si et seulement si son graphe d'implications ne contient pas de composante fortement connexe contenant x et $\neg x$ à la fois. Algorithme pour 2-SAT. 2-SAT est dans **P**.

2 Grammaires et langages non-contextuels

- Définition d'une grammaire non-contextuelle (= hors-contexte = algébrique). Symboles non-terminaux (en majuscule) et terminaux (en minuscules). Règles de production de la forme $X \to u$.
- Dérivation immédiate $u \Rightarrow v$, dérivation $u \Rightarrow^* v$, dérivation gauche, dérivation droite. Si $u \Rightarrow^* v$ alors on peut obtenir la dérivation de u en v en utilisant uniquement des dérivations gauche (resp. 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).
- Arbre de dérivation (= arbre d'analyse). Définition à connaître précisément.
- **Théorème :** $X \Rightarrow^* u$ si et seulement si il existe un arbre d'analyse de racine X dont la contaténation des feuilles (frontière) forme u.
  - **Sens direct (*)** : savoir démontrer le résultat par récurrence sur la longueur de la dérivation, savoir appliquer l'algorithme pour construire l'arbre d'analyse à partir d'une suite de dérivations.
  - **Sens réciproque (*)** : savoir démontrer le résultat par induction sur les arbres, savoir en pratique passer d'un arbre d'analyse à une suite de dérivations.
- Savoir sur des exemples simples et concrets : représenter un langage à l'aide d'une grammaire, proposer un algorithme d'analyse syntaxique pour construire l'arbre d'analyse d'un mot engendré par la grammaire (sur des cas simples seulement !).

3 Programmation concurrente

Remarque : la documentation des fonctions de synchronisation en C ou en OCaml pourra être demandée au colleur si nécessaire.

- Définition d'un fil d'exécution. Chaque file possède sa propre pile mais le tas est partagé. Création et attente (*join*) d'un fil d'exécution.
- Savoir identifier et expliquer les problèmes liés à l'accès simultané en lecture ou en écriture à une donnée partagée. Notion de zone critique.
- Mécanismes de synchronisation, il n'est pas attendu une connaissance exacte de la syntaxe en C ou en OCaml mais il faut au moins savoir écrire les algorithmes en pseudo-code.
  a. Verrous (mutex) : pour créer une zone d'exclusion mutuelle. Acquisition et libération d'un verrou.
  b. Sémaphores. Initialisation à une valeur entière arbitraire (éventuellement 0). Opérations $P(s)$ (`sem_wait`) et $V(s)$ (`sem_post`).
  c. Algorithme de Peterson uniquement pour 2 fils d'exécution. **L'algorithme de Peterson garantit l'exclusion mutuelle, l'absence d'interblocage et de famine(*)**
  d. Algorithme de la boulangerie de Lamport pour plusieurs fils d'exécution. Savoir exprimer l'algorithme en pseudo-code. L'algorithme garantit l'exclusion mutuelle, l'absence d'interblocage et de famine (preuve non exigible).
- Vu en TP : comptage parallèle des nombres premiers, modèle producteur-consommateur
- Problèmes liés à la synchronisation : interblocage, équité d'accès (famine). Il faut savoir présenter ces notions à l'aide d'exemples simples.Exemple du dîner des philosophes.

4 Jeux d'accessibilité

- Définition d'un jeu d'accessibilité à 2 joueurs. États finaux. Il n'y a que 3 types d'états finaux : gagnés pour J_1, gagnés pour J_2 ou partie nulle.
- Savoir dessiner l'arène d'un jeu d'accessibilité, pour des petits cas seulement.
- Notion de partie comme un chemin dans le graphe du jeu.
- Stratégie. Stratégie gagnante.
- Attracteurs : savoir calculer les attracteurs sur un petit jeu et en déduire l'existence de stratégies gagantes. Vus en cours : jeu de Nim, jeu de Chomp.
- Algorithme min-max : savoir le pseudo-code et l'appliquer sur de petits arbres.