

## Programme de colle - Semaine 19 (26 mars)

*La démonstration des énoncés marqués d'une étoile est exigible*

### 1 Complexité et décidabilité

- Notion de problème de décision.
- Problèmes décidables. La notion de machine de Turing est hors programme, une machine est un algorithme ou un programme écrit en C ou en OCaml s'exécutant sur une machine à mémoire infinie.
- Classe **P**.
- Principe de réduction d'un problème  $A$  à un problème  $B$  noté  $A \leq B$ .
- Principe de réduction polynomiale d'un problème  $A$  à un problème  $B$  noté  $A \leq_P B$ .
- Si  $B \in \mathbf{P}$  et  $A \leq_P B$  alors  $A \in \mathbf{P}$  (\*)
- Classe **NP** : définie comme la classe des problèmes admettant dont les instances positives admettent des certificats qui peuvent être vérifiés en temps polynomial. Notion de certificat pour un problème de décision.
- $\mathbf{P} \subset \mathbf{NP}$  (\*)
- Problèmes **NP-complets** : définis comme les problèmes de **NP** les plus difficiles.
- Si  $A$  est **NP-complet**,  $B \in \mathbf{NP}$ , et  $A \leq_P B$  alors  $B$  est **NP-complet** (\*)
- Théorème de Cook-Levin.
- Exemples de problèmes **NP-complets** vus en cours : **CNF-SAT**, **3-SAT** (réduction faite en cours), **CLIQUE** (réduction faite en cours), **INDEPENDANT**, **SUBSET-SUM**
- Notion de machine universelle : les machines  $M$  sont représentables sous forme d'une donnée  $\langle M \rangle$  et une machine dont l'entrée est  $\langle M \rangle$  peut simuler le comportement de la machine  $M$  sur toute entrée.
- Problèmes indécidables. Problème de l'arrêt **HALT** prenant en entrée le code d'une machine  $\langle M \rangle$  une entrée  $I$  pour  $M$  et décidant si la machine  $M$  termine sur l'entrée  $I$ . **HALT est indécidable** (\*).

### 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  $X \rightarrow u$ .
- Dérivation immédiate  $u \Rightarrow v$ , dérivation  $u \Rightarrow^* v$ , dérivation gauche, dérivation droite. Si  $X \Rightarrow^* u \in \Sigma^*$  alors  $X \Rightarrow_g^* u$  et  $Y \Rightarrow_d^* v$ .
- 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.  $X \Rightarrow^* u(\Sigma \cup V)^*$  si et seulement si il existe un arbre d'analyse de racine  $X$  dont la concaténation des feuilles donne  $u$ . Savoir passer en pratique d'un arbre de dérivation à une séquence de dérivations et réciproquement.
- Savoir sur des exemples simples et concrets : représenter un langage à l'aide d'une grammaire, proposer un algorithme pour construire l'arbre d'analyse d'un mot engendré par la grammaire (sur des cas simples : expressions arithmétiques ou logiques avec parenthésage strict, langage balisés, ...).

### 3 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 pour la colle).
- 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**.

### 4 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*

- 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`) et  $P$  (`sem_wait`), applications (verrous, jauges, communication par signaux, ...)
- Algorithme de Peterson : pour une exclusion mutuelle algorithmique de deux processus par une attente active. **L'algorithme de Peterson garantit l'exclusion mutuelle et l'absence de famine (\*)**
- Algorithme de la Boulangerie de Lamport : pour une exclusion mutuelle algorithmique de  $N$  processus par une attente active.

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