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

La démonstration des énoncés marqués d'une étoile est exigible

1 Décidabilité et complexité

Dans ce cours on s'intéresse à la hiérarchisation des ***problèmes de décision****. On ne parle pas de machine de Turing, le modèle de calcul a été nommé* machine *et peut désigner au choix un algorithme décrit en pseudo-code ou une fonction écrite en C ou en OCaml.*

- Notion de problème de décision.
- Problèmes décidables : ce sont les problèmes de décision pour lesquels on a un algorithme qui répond à la question.
- Classe **P** : ce sont les problèmes de décision pour lesquels on a un algorithme de complexité *polynomiale* qui répond à la question.
- Principe de réduction d'un problème A à un problème B noté $A \leq B$: il existe un algorithme qui transforme les instances de A en instances de B en préservant la réponse oui/non. Ceci signifie que le problème B est plus difficile que le problème A.
- Principe de réduction polynomiale d'un problème A à un problème B noté $A \leq_P B$.
- **Si $A \leq_P B$ et $B \leq_P C$ alors $A \leq_P C$ (*)** (par composition des deux réductions)
- **Si $B \in$ P et $A \leq_P B$ alors $A \in$ P** (*)
- Classe **NP** : définie comme la classe des problèmes dont les instances positives admettent des certificats qui peuvent être vérifiés par un algorithme en temps polynomial. Notion de certificat pour un problème de décision.
- **P** $\subset$ **NP** (*)
- Classe **NP**-complet : définie comme la classe des problèmes qui sont dans **NP** et qui sont **NP-durs** (c'est-à-dire que tout problème de **NP** se réduit à lui). Exemples faits en classe : 3-SAT (réduction difficile avec la transformation de Tseitin), CNF-SAT (réduction triviale depuis 3-SAT), CLIQUE (réduction depuis 3-SAT).
- Preuve de NP-complétude par réduction avec le théorème : **Si A est NP-complet, si $B \in NP$ et si $A \leq_P B$ alors B est NP-complet (*)**
- Problèmes indécidables : problème de l'arrêt HALT, **HALT est indécidable (*)** (preuve par argument diagonal) NB: les grammaires algébriques n'ont pas été vues, le problème de correspondance de Post n'est pas au programme.

À savoir faire :

- Identifier un problème de décision.
- Montrer qu'un problème est décidable.
- Montrer qu'un problème est dans P
- Établir une réduction entre deux problèmes de décision. En autonomie si la réduction est simple, avec un exercice guidé dans les cas plus difficiles. Exemples vus en TD : réductions

entre CLIQUE et INDEPENDENT-SET, réduction de 3-COLOR vers 4-COLOR, réduction de 3-COLOR vers k-COLOR avec $k \geq 4$.
- Montrer qu'un problème est dans NP (SAT, CLIQUE, k-COLOR vus en cours ou en TD) : l'élève doit savoir identifier les certificats pour un problème de décision et donner un vérificateur polynomial.
- Montrer qu'un problème est NP-complet (3-SAT, CNF-SAT et CLIQUE faits en cours) : encore une fois soit la réduction est facile (voire très facile dans le cas de l'identité), soit la réduction est difficile et dans ce cas l'exercice guide l'élève. L'exercice peut admettre qu'un problème est NP-complet pour montrer la NP-complétude d'un autre.
- Montrer qu'un problème est indécidable avec une preuve par l'absurde : le programme officiel est très peu développé sur ce point, seul **HALT** est au programme, le théorème de Rice a été abordé sous forme d'exercice. On ne peut faire que très peu de choses...

2 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

3 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.