Année 2018/2019

Métaplans d'Informatique pour l'Agrégation

Pierre Le Barbenchon

ENS Rennes

TABLE DES MATIÈRES

CHAPITRE 1

PLANS DE LEÇONS

1.1 Conseils

— C'est vous qui choisissez le niveau de votre leçon

— Ne mettre que des choses que l'on maitrise !

— Ce n'est pas obligatoire de remplir les 3 pages

— Il vaut mieux avoir moins de pages mais que des choses que l'on maitrise

— Je vous présente ici mes plans, ils sont loin d'être parfait

— Il ne sont d'ailleurs là que pour vous aider et vous guider, il faut que vous vous appropriiez des leçons [1]

— Personnalisez vos plans, car c'est finalement très personnel et c'est que comme ça que vous ferez un travail en profondeur !

ATTENTION : Ici, je mets le squelette de mes plans, ils sont volontairement très légers [2], c'est juste pour aider, donner les grandes parties et l'articulation des idées. En aucun cas, il ne faut les utiliser tel quel. Il ne reste plus qu'à les travailler ! [3]

1. tiens, ça fait deux "i" collés, c'est rigolo
2. certains plus que d'autres, en fonction du temps de travail dessus
3. Autrement dit tout faire

1.2 Métaplans d'informatique

Leçon 901
Structures de données. Exemples et applications.

Rapport du jury :

Le mot algorithme ne figure pas dans l'intitulé de cette leçon, même si l'utilisation des structures de données est évidemment fortement liée à des questions algorithmiques. La leçon doit donc être orientée plutôt sur la question du choix d'une structure de données. Le jury attend du candidat qu'il présente différents types abstraits de structures de données en donnant quelques exemples de leur usage avant de s'intéresser au choix de la structure concrète. Le candidat ne peut se limiter à des structures linéaires simples comme des tableaux ou des listes, mais doit présenter également quelques structures plus complexes, reposant par exemple sur des implantations à l'aide d'arbres. Les notions de complexité des opérations usuelles sur la structure de données sont bien sûr essentielles dans cette leçon.

Questions possibles :

- Donner les types abstraits, concrets et implantation possible pour une certaine structure.
  Réponse : Par exemple, type abstrait : file de priorité ; type concret : tas_min ; implantation : tableau
- Connaissez vous d'autres arbres que les ABRO et AVL ?
  Réponse : Arbres Rouge Noir, Arbre Splay
- Donner des utilisations de Union Find.
  Réponse : Algorithme de Kruskal

Plan détaillé :

I - Type abstrait de base
  1) opérations souhaitées
     find, create, insert, suppr, union, find_min, find_earlest
  2) tableau
     définition
     coût des opérations
  3) liste
     définition
     coût des opérations

II - Structure de données d'ordonnancement
  1) pile
     définition
     parcours en profondeur
  2) file
     définition
     parcours en largeur

3) file de priorité
   définition
   Tri par tas
   PRIM
   codage de HUFFMAN
4) graphe
   matrice d'adjacence
   liste d'adjacence

III - Structure de recherche : dictionnaire
1) arbres
   ABR
   AVL
   ABR optimaux
   B-arbres
2) table de hachage
   hachage parfait

IV - Partitionner un ensemble
   algorithmes liés à UNIONFIND
   KRUSKAL

Développements :

- Tri par tas
- B-arbres

Références :

♥♥♥♥ - [8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST et Clifford STEIN, *Algorithmique*

♥♥♥♥ - [12] Christine FROIDEVAUX, Marie-Claude GAUDEL et Michèle SORIA , *Type de données et algorithmes*

Leçon 903
Exemples d'algorithmes de tri. Correction et complexité.

Rapport du jury :

Sur un thème aussi classique, le jury attend des candidats la plus grande précision et la plus grande rigueur.

Ainsi, sur l'exemple du tri rapide, il est attendu du candidat qu'il sache décrire avec soin l'algorithme de partition et en prouver la correction en exhibant un invariant adapté. L'évaluation des complexités dans le cas le pire et en moyenne devra être menée avec rigueur : si on utilise le langage des probabilités, il importe que le candidat sache sur quel espace probabilisé il travaille.

On attend également du candidat qu'il évoque la question du tri en place, des tris stables, des tris externes ainsi que la représentation en machine des collections triées.

Questions possibles :

- Appliquer les tris naïfs sur la liste `[2; 5; 1; 8; 3; 10]`.
- Montrer que la complexité minimale d'un tri par comparaison est $O(n \log n)$.
  Réponse : voir appendice
- Montrer la complexité du tri rapide.

Plan détaillé :

I - Le problème du tri
- définition du problème de tri
- définition de "stable"
- définition de "en place"
- critère de comparaison

II - Tri par comparaison
- propriété
- complexité minimale (voir appendice)

1) naïf
   - tri par sélection
   - tri par insertion
   - tri bulle
2) diviser pour régner
   - tri fusion
   - tri rapide
3) structure de données
   - Tri par tas

III - Tri linéaire
1) par dénombrement
2) par paquet
3) par base

IV - Autres tris
1) Tri topologique
2) tri externe

Développements :

- Tri topologique
- Tri par tas

Références :

♥♥♥♥ - [8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST et Clifford STEIN, *Algorithmique*

♥♥♥ - [12] Christine FROIDEVAUX, Marie-Claude GAUDEL et Michèle SORIA , *Type de données et algorithmes*

Leçon 907
Algorithmique du texte. Exemples et applications.

Rapport du jury :

Cette leçon devrait permettre au candidat de présenter une grande variété d'algorithmes et de paradigmes de programmation, et ne devrait pas se limiter au seul problème de la recherche d'un motif dans un texte, surtout si le candidat ne sait présenter que la méthode naïve.

De même, des structures de données plus riches que les tableaux de caractères peuvent montrer leur utilité dans certains algorithmes, qu'il s'agisse d'automates ou d'arbres par exemple.

Cependant, cette leçon ne doit pas être confondue avec la 909, "*Langages rationnels et Automates finis. Exemples et applications.*". La compression de texte peut faire partie de cette leçon si les algorithmes présentés contiennent effectivement des opérations comme les comparaisons de chaînes : la compression LZW, par exemple, est plus pertinente dans cette leçon que la compression de Huffman.

Questions possibles :

- Donner le principe de l'algorithme de la plus longue sous séquence commune.
- Quelle est la différence entre Morris Pratt et Knuth Morris Pratt ?

  Réponse : la complexité spatiale

Motivations :

L'algorithmique du texte est présente partout dans notre quotidien. J'ai séparé mon plan en 4 parties. Une première sur la recherche de motif, essentielle pour les `Ctrl+F` afin de chercher un mot sur une page web, sur un document, dans un fichier de 500 pages qui recense plein de résultats utiles pour passer l'agrégation, etc. ; une deuxième sur l'analyse lexicale et syntaxique qui est nécessaire lors de la compilation de programmes informatiques ; une troisième sur la comparaison de texte, notamment dans l'étude des séquences de l'ADN où l'on cherche les ressemblances ; une quatrième sur la compression de texte, afin d'améliorer le stockage de données.

Plan détaillé :

I - Recherche d'un motif

algorithme naïf

Automate des occurrences pour l'algorithme de Morris Pratt

Algorithme de Knuth Morris Pratt

II - Analyse lexicale et syntaxique

1) analyse lexicale

utilisation de l'automate du motif

2) analyse syntaxique

Cocke-Younger-Kasami

III - Comparaison de texte

distance d'édition

plus longue sous séquence commune

IV - Compression

1) codage de HUFFMAN

2) compression LZW

Développements :

- Automate des occurrences
- Cocke-Younger-Kasami

Références :

♥♥♥ - [8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST et Clifford STEIN, *Algorithmique*

♥♥♥ - [9] Maxime CROCHEMORE, Christophe HANCART et Thierry LECROQ , *Algorithmique du texte*

Leçon 909
Langages rationnels et automates finis. Exemples et applications.

Rapport du jury :

Pour cette leçon très classique, il importe de ne pas oublier de donner exemples et applications, ainsi que le demande l'intitulé.

Une approche algorithmique doit être privilégiée dans la présentation des résultats classiques (déterminisation, théorème de Kleene, etc.) qui pourra utilement être illustrée par des exemples. Le jury pourra naturellement poser des questions telles que : connaissez-vous un algorithme pour décider de l'égalité des langages reconnus par deux automates ? quelle est sa complexité ?

Des applications dans le domaine de l'analyse lexicale et de la compilation entrent naturellement dans le cadre de cette leçon.

Questions possibles :

- Donner des idées de preuve pour le théorème de Kleene.
- Déterminiser un automate.
- Donner un exemple d'automate à n états dont l'automate déterminisé possède 2^n états.

Plan détaillé :

I - Langages rationnels, langages reconnaissables

1) langage rationnel

expression rationnelle

2) langage reconnaissable

automate

déterminisme

complet

3) équivalence entre les deux langages

théorème de Kleene

II - Propriétés des langages rationnels

1) stabilité et décidabilité

Décidabilité de Presburger

2) résiduels

Automate des occurrences

Algorithme de Hopcroft

recherche de motif

3) Lemme de l'étoile

En annexe :

- automate du complémentaire
- automate de l'union
- automate de l'intersection

Développements :

- Automate des occurrences
- Décidabilité de Presburger

Références :

♥♥♥♥♥ - [6] Olivier CARTON, *Langages Formels : Calculabilité et complexité*

Leçon 912
Fonctions récursives primitives et non primitives. Exemples.

Rapport du jury :

Il s'agit de présenter un modèle de calcul : les fonctions récursives. S'il est bien sûr important de faire le lien avec d'autres modèles de calcul, par exemple les machines de TURING, la leçon doit traiter des spécificités de l'approche. Le candidat doit motiver l'intérêt de ces classes de fonctions sur les entiers et pourra aborder la hiérarchie des fonctions récursives primitives. Enfin, la variété des exemples proposés sera appréciée.

Questions possibles :

- Comment écrire la fonction mod en fonction μ-récursive ?

  Réponse : voir appendice

Plan détaillé :

I - Fonctions primitives récursives

1) syntaxe

   les fonctions de bases

2) sémantique

   la boucle for

   prédicat

   minimisation bornée

   bijection primitive récursive dont la réciproque est primitive récursive (voir appendice)

3) limite

   fonction Ackermann non primitive récursive

II - Fonctions μ-récursives

   minimisation non bornée

   prédicat sûr

   fonction totale/partielle

   boucle while

   la fonction Ackermann est μ-récursive

III - Modèle de calcul

1) thèse de Church

2) machine de Turing

   Machine de Turing $\implies$ μ-recursif

   Caractérisation de RE

3) λ-calcul

   μ-recursif $\implies$ λ-définissable

Développements :

- μ-recursif $\Longrightarrow$ λ-définissable
- Machine de Turing $\Longrightarrow$ μ-recursif

Références :

♥♥♥ - [16] Richard LASSAIGNE et Michel DE ROUGEMONT, *Logique et fondements de l'informatique logique du 1er ordre, calculabilité et lambda-calcul*

♥♥♥♥ - [20] Pierre WOLPER, *Introduction à la calculabilité*

Leçon 913
Machines de Turing. Applications.

Rapport du jury :

Il s'agit de présenter un modèle de calcul. Le candidat doit expliquer l'intérêt de disposer d'un modèle formel de calcul et discuter le choix des machines de Turing. La leçon ne peut se réduire à la leçon 914 ou à la leçon 915, même si, bien sûr, la complexité et l'indécidabilité sont des exemples d'applications. Plusieurs développements peuvent être communs avec une des leçons 914, 915, mais il est apprécié qu'un développement spécifique soit proposé.

J'ai rédigé le plan de cette leçon (voir ici)

Plan détaillé :

I - Machine de Turing
- 1) Définitions
  - Def Machine de Turing
  - Def Configuration
  - Def Exécution
- 2) Variante
  - Plusieurs rubans
  - Double ruban infini
  - Machine non déterministe

II - Décidabilité et indécidabilité
- Définition Classe R
- Définition Classe RE
- Problème de l'arrêt
- Définition de la réduction
- Problème du mot

III - Calculabilité et thèse de Church
- 1) Thèse de Church
  - Énoncé
- 2) Fonctions μ-récursives
  - Caractérisation de RE
  - Machine de Turing $\Longrightarrow$ μ-recursif
- 3) λ-calcul
  - Théorème d'équivalence

IV - Complexité
- Classe P
- Classe NP
- Réduction polynomiale
- P $\neq$ NP
- Classe NP-complet
- Théorème de Cook

Développements :

- Théorème de Cook
- Machine de Turing $\Longrightarrow$ μ-recursif

Références :

♥♥♥♥♥ - [6] Olivier CARTON, *Langages Formels : Calculabilité et complexité*

♥♥ - [16] Richard LASSAIGNE et Michel DE ROUGEMONT, *Logique et fondements de l'informatique logique du 1er ordre, calculabilité et lambda-calcul*

♥♥♥ - [20] Pierre WOLPER, *Introduction à la calculabilité*

Leçon 914
Décidabilité et indécidabilité. Exemples.

Rapport du jury :

Le programme de l'option offre de très nombreuses possibilités d'exemples. Si les exemples classiques de problèmes sur les machines de TURING figurent naturellement dans la leçon, le jury apprécie des exemples issus d'autres parties du programme : théorie des langages, logique,...

Le jury portera une attention particulière à une formalisation propre des réductions, qui sont parfois très approximatives.

Questions possibles :

- Montrer l'indécidabilité du problème de l'arrêt.

Motivations :

La frontière entre décidabilité et indécidabilité sépare certains problèmes en terme de possibilité de résolution grâce à une machine de Turing. En effet, on va avoir une correspondance entre problème et langage et la notion de décidabilité se transposera entre langages décidés par une machine de Turing et les problèmes décidables. La classe R est l'ensemble des problèmes décidables, cependant la classe RE représente les problèmes acceptés par une machine de Turing, il existe des problèmes qui sont indécidables et qui ne sont pas dans RE comme l'arithmétique de PEANO par exemple.

J'ai rédigé le plan de cette leçon (voir ici)

Plan détaillé :

I - Généralisation et cadre
  1) Classe R et RE
    Définitions
    Caractérisation de RE
  2) Fonctions calculables et réductions
    Définitions

II - Technique pour prouver l'indécidabilité
  1) Problème de l'arrêt
    Définition
    Idée de preuve
  2) Utilisation de la réduction
    Propriétés

III - Décidabilité et indécidabilités de quelques problèmes (voir appendice)
  1) Machines de Turing
    Acceptation du mot (indécidable)
    Problème de POST (indécidable)

2) Automates finis

Acceptation du mot (décidable)

Universalité (décidable)

3) Grammaires algébriques

Acceptation du mot (décidable)

Vide (décidable)

Intersection (indécidable)

Universalité (indécidable)

4) Logique

Décidabilité de Presburger (décidable)

SAT (décidable)

Indécidabilité de la validité d'une formule du premier ordre (indécidable)

PEANO (indécidable)

En annexe :

- un arbre des réductions des problèmes indécidables (voir appendice).
- les inclusions des classes R, RE et indécidables (voir Carton [6]).

Développements :

- Indécidabilité de la validité d'une formule du premier ordre
- Décidabilité de Presburger

Références :

♥♥♥♥ - [6] Olivier CARTON, *Langages Formels : Calculabilité et complexité*

♥♥♥ - [20] Pierre WOLPER, *Introduction à la calculabilité*

Leçon 915
Classes de complexité. Exemples.

Rapport du jury :

Le jury attend que le candidat aborde à la fois la complexité en temps et en espace. Il faut naturellement exhiber des exemples de problèmes appartenant aux classes de complexité introduites, et montrer les relations d'inclusion existantes entre ces classes, en abordant le caractère strict ou non de ces inclusions. Le jury s'attend à ce que les notions de réduction polynomiale, de problème complet pour une classe, de robustesse d'une classe vis à vis des modèles de calcul soient abordées.

Se focaliser sur la décidabilité dans cette leçon serait hors sujet.

Questions possibles :

- Donner une autre définition de NP.
- Connaissez-vous un autre modèle de calcul pour calculer la complexité ?
- Donner un exemple de problème dans NP dont on ne sait pas s'il est dans P ou NP-dur.
- Expliquer QBF_SAT. Pourquoi il est PSpace-complet ?
- Qu'est ce qu'on appelle un problème de décision ?
- Donner une définition de la NP-difficulté d'un problème d'optimisation
- Pourquoi PSpace $\subset$ ExpTime ?
- Si on change de codage de l'entrée, est ce que la complexité change ?

  Réponse : oui, par exemple, si le codage est en unaire, le problème SubsetSum est dans P, on parle de problème pseudo-polynomiaux

Motivations :

On a classifié pendant la leçon **914**, les problèmes décidables et indécidables, on veut maintenant trouver une hiérarchie plus fine des problèmes décidables. On se place donc parmi les problèmes décidables. Il va falloir parler de modèle de calcul, car la complexité dépend a priori du modèle de calcul (cela donnera la séparation entre P et NP par exemple). On s'intéressera aux complexités temporelle et spatiale et le lien qu'il y a entre les deux.

Je suis tombé sur cette leçon le jour de l'agrégation (voir section **??**)

Plan détaillé :

I - Outil : Machine de Turing

Définition Machine de Turing

Définition Complexité spatiale et temporelle

Lemme de lien entre les deux complexités

II - Complexité temporelle

1) Changement de modèle de calcul

Théorème d'accélération

Équivalence avec machine de Turing à ruban biinfini (et même complexité temporelle)

Équivalence avec machine de Turing à k rubans (et facteur carré sur la complexité)

Équivalence avec machine de Turing non déterministe (et facteur exponentiel sur la complexité)

2) Classes temporelles

Définition de TIME et NTIME

Définition de P, NP, EXPTIME et NEXPTIME

Exemple du problème du mot des langages algébriques qui est dans P Cocke-Younger-Kasami

Exemple de SAT qui est NP.

3) NP-complétude

Définition réduction polynomiale

Propriété, si A se réduit à B et B est dans P alors A est dans P

Définition NP-dureté

Propriété, si A se réduit à B et A est NP-dur, alors B est NP-dur

Définition NP-complétude

Théorème de Cook

Exemple CNF_SAT et 3_SAT sont NP-complets

Exemple 3-Coloration d'un graphe, Vertex_Cover, Circuit Hamiltonien et Voyageur de Commerce sont NP-complets

III - Complexité spatiale

1) Classes spatiales

Théorème de Savitch

Définition de SPACE et NSPACE

Définition de PSPACE, NPSPACE, EXPSPACE et NEXPSPACE

Exemple QBF_SAT est PSPACE

Définition de PSPACE-dureté

Définition de PSPACE-complétude

Exemple QBF_SAT est PSPACE-complet

2) Classe logarithmique

Définition des machines à 3 bandes qui servent à la réduction logarithmique

Définition de L et NL

Exemple PATH est dans NL

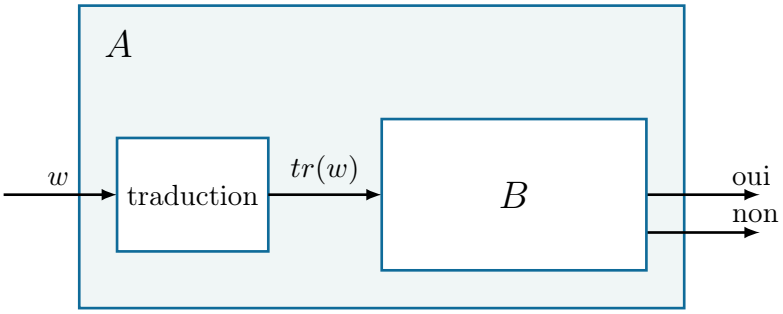
IV - Hiérarchie des complexités

Théorème des hiérarchies

inclusion de toutes les classes avec certaines inclusions strictes

En annexe :

- un schéma de réduction



- un arbre des réductions des problèmes NP-complet (voir appendice)
- les inclusions des classes (voir Carton [6])

Développements :

- Cocke-Younger-Kasami
- Théorème de Cook
- 3-Coloration d'un graphe

Références :

♥♥♥♥♥ - [6] Olivier CARTON, *Langages Formels : Calculabilité et complexité*

♥♥♥ - [20] Pierre WOLPER, *Introduction à la calculabilité*

♥♥ - [13] Michael GAREY et David JOHNSON, *Computers and Intractability : A Guide to the Theory of Np-Completeness*

Leçon 916
Formules du calcul propositionnel : représentation, formes normales, satisfiabilité. Applications.

Rapport du jury :

Le jury attend des candidats qu'ils abordent les questions de la complexité de la satisfiabilité.

Pour autant, les applications ne sauraient se réduire à la réduction de problèmes NP-complets à SAT.

Une partie significative du plan doit être consacrée à la représentation des formules et à leurs formes normales.

Questions possibles :

- Montrer rigoureusement la réduction de CnfSat à 3Sat.
- Montrer qu'une formule sous CNF équivalente à

$$(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \cdots \vee (x_n \wedge y_n)$$

  est de taille exponentielle en la taille de la formule de départ.
- Montrer que le système $\{\leftrightarrow, \neg\}$ n'est pas complet.
- Donner un système complet de connecteurs logiques qui ne contient qu'un seul connecteur logique.

J'ai rédigé le plan de cette leçon (voir ici)

Plan détaillé :

I - Syntaxe
  1) Langage
     Vocabulaire
     Définition par induction
  2) Représentation
     Formule
     Arbre
  3) Résolution par réfutation
     Règle de coupure
     Règle de factorisation

II - Sémantique
  1) Valuation
     Définition
     Table de vérité
  2) Validité et satisfiabilité
     Définition validité

Définition satisfiabilité

3) Équivalence sémantique

Définition de tautologie

Lois de De Morgan

4) Lien entre syntaxe et sémantique

Théorème de correction et complétude

III - Satisfiabilité et formes normales

Théorème de Cook

1) Mise sous forme normale conjonctive

Définition

Propriété d'équivalence avec des formules de tailles exponentielles

Transformation de Tseitin

2) D'autres formes normales

Définition de forme normale disjonctive

Problème DnfSat dans P

Développements :

- Théorème de Cook
- Transformation de Tseitin

Références :

♥♥♥♥ - [16] Richard LASSAIGNE et Michel DE ROUGEMONT, *Logique et fondements de l'informatique logique du 1er ordre, calculabilité et lambda-calcul*

♥♥ - [5] Mordechai BEN ARI, *Mathematical Logic for Computer Science*

♥♥♥ - [7] René CORI et Daniel LASCAR, *Logique Mathématique*

♥♥ - [11] Jacques DUPARC, *Logique pas à pas*

Leçon 918
Systèmes formels de preuve en logique du premier ordre. Exemples.

IMPASSE

Ce n'est pas par difficulté de la leçon mais car mes couplages m'obligeaient à apprendre deux nouveaux développements à un mois des oraux. Donc cela justifie le fait qu'il faut faire ces couplages tôt dans l'année et bien travailler ces développements.

Après l'agreg, je vous écris ce que je pense que j'aurais fait comme plan :

Rapport du jury :

Le jury attend du candidat qu'il présente au moins la déduction naturelle ou un calcul de séquents et qu'il soit capable de développer des preuves dans ce système sur des exemples classiques simples. La présentation des liens entre syntaxe et sémantique, en développant en particulier les questions de correction et complétude, et de l'apport des systèmes de preuves pour l'automatisation des preuves est également attendue.

Le jury appréciera naturellement si des candidats présentent des notions plus élaborées comme la stratégie d'élimination des coupures mais est bien conscient que la maîtrise de leurs subtilités va au-delà du programme.

Plan détaillé :

I - Logique du premier ordre
- Syntaxe
- Sémantique

II - Système de preuves avec règles syntaxiques

1) Déduction naturelle
- Règles en annexe
- Théorème de correction et complétude

2) Calcul des séquents
- Règles en annexe
- Théorème de correction et complétude

III - Preuve par résolution
- Forme prénexe
- Skolémisation
- Unificateur
- Unificateur principal
- Règle de résolution et factorisation
- Exemples
- Modèles de Herbrand
- Théorème de Correction et complétude

Développements : Bah j'en ai pas ! On pourrait mettre :

- Théorème de complétude de la déduction naturelle
- Algorithme d'unification

Références :

♥♥♥♥ - [10] René DAVID, Karim NOUR et Christophe RAFFALLI, *Introduction à la logique : Théorie de la démonstration*

♥♥♥♥♥ - [15] René LALEMENT, *Logique, Réduction, Résolution*

Leçon 921
Algorithmes de recherche et structures de données associées.

Rapport du jury :

Le sujet de la leçon concerne essentiellement les algorithmes de recherche pour trouver un élément dans un ensemble : l'intérêt des structures de données proposées et de leur utilisation doit être argumenté dans ce contexte.

La recherche d'une clé dans un dictionnaire sera ainsi par exemple l'occasion de définir la structure de données abstraite "dictionnaire", et d'en proposer plusieurs implantations concrètes. De la même façon, on peut évoquer la recherche d'un mot dans un lexique : les arbres préfixes (ou *digital tries*) peuvent alors être présentés. Mais on peut aussi s'intéresser à des domaines plus variés, comme la recherche d'un point dans un nuage (et les *quad-trees*), et bien d'autres encore.

Plan détaillé :

I - Recherche dans un dictionnaire
- 1) Structures simples
  - liste non triée
  - tableau trié
- 2) Arbres
  - ABR
  - AVL
  - ABRO
  - B-arbres
- 3) Table de Hachage
  - Principe
  - Hachage parfait

II - Recherche dans un texte
- 1) Recherche naïve
  - Algo
  - Complexité
- 2) Automate des occurrences
  - Algorithme de Morris-Pratt
- 3) Algorithme de Knuth-Morris-Pratt
  - Principe
  - Complexité

Développements :

- B-arbres
- Automate des occurrences

Références :

♥♥♥♥ - [8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST et Clifford STEIN, *Algorithmique*

♥♥♥♥ - [12] Christine FROIDEVAUX, Marie-Claude GAUDEL et Michèle SORIA , *Type de données et algorithmes*

Leçon 923
Analyses lexicale et syntaxique. Applications.

Rapport du jury :

Cette leçon ne doit pas être confondue avec la 909, qui s'intéresse aux seuls langages rationnels, ni avec la 907, sur l'algorithmique du texte.

Si les notions d'automates finis et de langages rationnels et de grammaires algébriques sont au cœur de cette leçon, l'accent doit être mis sur leur utilisation comme outils pour les analyses lexicale et syntaxique. Il s'agit donc d'insister sur la différence entre langages rationnels et algébriques, sans perdre de vue l'aspect applicatif : on pensera bien sûr à la compilation. Le programme permet également des développements pour cette leçon avec une ouverture sur des aspects élémentaires d'analyse sémantique.

Questions possibles :

- Calculer la fonction Premier pour la grammaire suivante $G = (\Sigma, \mathcal{N}, \mathcal{T}, S, \mathcal{R})$ où $\Sigma = \{a, b, c, d\}$, $\mathcal{N} = \{S, A, B, C\}$, $\mathcal{T} = \Sigma$ et $\mathcal{R}$ est décrit par

$$\begin{aligned} S &\to AB \mid Ca \\ A &\to aAb \mid \varepsilon \\ B &\to bB \mid b \\ C &\to cC \mid d. \end{aligned}$$

- Donner la table d'analyse syntaxique de la grammaire précédente.
- Donner une grammaire qui n'est pas LL(1).
- Calculer la fonction Suivant pour la grammaire suivante $G = (\Sigma, \mathcal{N}, \mathcal{T}, S, \mathcal{R})$ où $\Sigma = \{a, b\}$, $\mathcal{N} = \{S\}$, $\mathcal{T} = \Sigma$ et $\mathcal{R}$ est décrit par

$$S \to \varepsilon \mid aSb.$$

Motivations :

Quand un ordinateur compile un programme dans un certain langage, il passe par trois étapes avant la compilation : l'analyse lexicale, l'analyse syntaxique et l'analyse sémantique. On va présenter les deux premiers. L'analyse lexicale permet de regrouper les caractères en des groupes de caractères appelés « lexèmes », cette analyse permet de renvoyer une erreur si le programme contient des caractères que le langage informatique ne reconnait pas (par exemple ¿ou ±). L'analyse syntaxique permet de vérifier que le programme est une succession de lexèmes qui est un mot de la grammaire qui engendre le langage utilisé, cette analyse permet de renvoyer une erreur si le programme contient des successions invalides de lexèmes (par exemple, le symbole « + » doit être entre deux lexèmes de type « entiers », sinon cela déclenche une erreur syntaxique).

Plan détaillé :

I - Langages rationnels et algébriques [6]

1) Langage rationnel

Définition

Automate

Exemple automate du motif (utilité dans l'analyse lexicale)

Expression régulière

Théorème de Kleene

2) Langage algébrique

Définition grammaire

Définition dérivation

Arbre de dérivation

Lemme de factorisation

Exemples de grammaire préfixe et postfixe (utilité dans l'analyse syntaxique)

II - Analyse lexicale [17]

But : transformer un texte en liste de lexèmes

Définitions règles lexicales

Algorithme d'analyse lexicale en liste de lexèmes

III - Analyse syntaxique [17]

But : savoir si un texte est engendré par une grammaire et construire l'arbre de dérivation

1) Algorithme générique

Approche néïve

Cocke-Younger-Kasami

2) Analyse descndante

Grammaire $LL(1)$

Fonctions PREMIER, SUIVANT

Caractérisation de PREMIER pour l'analyse $LL(1)$

Présentation de l'algorithme

3) Analyse ascendante

Grammaire $LR(0)$

En annexe :

- Exemples de grammaires $LL(1)$

Développements :

- Cocke-Younger-Kasami
- Caractérisation de PREMIER pour l'analyse $LL(1)$

Références :

♥♥♥♥♥♥♥ - [17] Romain LEGENDRE et François SCHWARZENTRUBER, *Compilation : Analyse Lexicale et Syntaxique du Texte à sa Structure en Informatique*

♥♥ - [6] Olivier CARTON, *Langages Formels : Calculabilité et complexité*

Leçon 924
Théories et modèles en logique du premier ordre. Exemples.

Rapport du jury :

Le jury s'attend à ce que la leçon soit abordée dans l'esprit de l'option informatique, en insistant plus sur la décidabilité/indécidabilité des théories du premier ordre que sur la théorie des modèles.

Il est attendu que le candidat donne au moins un exemple de théorie décidable (respectivement complète) et un exemple de théorie indécidable.

Le jury appréciera naturellement si des candidats connaissent l'existence du premier théorème d'incomplétude mais est bien conscient que la démonstration va au-delà du programme.

Questions possibles :

- Montrer que si T est récursive, alors $T-$valid est dans RE.
- Montrer que si T est complète et récursive, alors $T-$valid est dans R.
- Comment montrer le théorème de compacité à partir du théorème de correction et complétude ?
- Donner des applications du théorème de compacité.

Motivations :

La logique du premier ordre peut être vue comme une extension de la logique propositionnelle, cependant l'approche classique est une nouvelle structure syntaxique comprenant des symboles de fonctions et de prédicats, ensuite quand on ajoute la sémantique, les interprétations des fonctions seront des fonctions (mathématiques) prenant en argument des éléments du domaine et renvoyant un élément du domaine tandis que les interprétations des prédicats seront des fonctions prenant en argument des éléments du domaine et renvoyant un booléen True ou False. La donnée de modèles (le penchant de ce qu'on appelle « valuations » en logique propositionnelle) qui correspond à un domaine et des interprétations de tous les symboles de fonctions et de prédicats permettra de dire si une formule est satisfiable ou valide. On définit des théories en ce donnant un ensemble clos par conséquence logique de formules closes. Certaines théories sont axiomatisables, c'est-à-dire que la théorie sera l'ensemble des formules qui sont conséquence logique des axiomes.

Plan détaillé :

I - Logique du premier ordre [10]

1) Syntaxe

Définition de signature

Définition de terme

Définition de formules

Définition de variables libre et liée

Définition formule close

2) Modèles

Définition de structure

Définition de valuation

Définition d'interprétation

Définition de modèle

Définition de validité

3) Théories

Définition de théorie

Définition de modèle d'une théorie

Définition de théorie cohérente/contradictoire

$T \vDash F$ si et seulement si $T \cup \{\neg F\}$ contradictoire

Définition de $T-$validité

Définition de théorie complète

Définition de théorie décidable

Indécidabilité de la validité d'une formule du premier ordre en disant que la théorie vide est indécidable

II - Résolution

1) Méthode de la résolution

Forme prénexe

Skolémisation

Unificateur principal

Règle de coupure et de factorisation

2) Théorème de correction et complétude

Modèle de Herbrand

$T \cup \{\neg F\}$ contradictoire si et seulement si $T \cup \{\neg F\} \vdash \bot$

3) Conséquences [10]

Théorème de compacité

Théorème de Lowenheim Skolem

Définition théorie récursive

Si T est récursive, alors $T-$valid est dans RE

Si T est complète et T est récursive, alors $T-$valid est dans R

III - Théories arithmétiques

1) Arithmétique de Presburger

Définition de la théorie de Presburger

$\text{T}_{\text{PRES}} \vDash F$ si et seulement si $\mathbb{N} \vDash F$

Décidabilité de Presburger

2) Arithmétique de Peano

Définition de la théorie de Peano

T_{PEANO} est indécidable

Théorème d'incomplétude de Gödel : Si T est récursive, cohérente et contient T_{PEANO}, alors T n'est pas complète.

Développements :

- Indécidabilité de la validité d'une formule du premier ordre
- Décidabilité de Presburger

Références :

♥♥♥♥♥ - [10] René DAVID, Karim NOUR et Christophe RAFFALLI, *Introduction à la logique : Théorie de la démonstration*

♥♥♥♥ - [15] René LALEMENT, *Logique, Réduction, Résolution*

♥♥♥ - [16] Richard LASSAIGNE et Michel DE ROUGEMONT, *Logique et fondements de l'informatique logique du 1er ordre, calculabilité et lambda-calcul*

Leçon 925
Graphes : représentations et algorithmes.

Rapport du jury :

Cette leçon offre une grande liberté de choix au candidat, qui peut choisir de présenter des algorithmes sur des problèmes variés : connexité, diamètre, arbre couvrant, flot maximal, plus court chemin, cycle eulérien, etc. mais aussi des problèmes plus difficiles, comme la couverture de sommets ou la recherche d'un cycle hamiltonien, pour lesquels il pourra proposer des algorithmes d'approximation ou des heuristiques usuelles. Une preuve de correction des algorithmes proposés sera évidemment appréciée. Il est attendu que diverses représentations des graphes soient présentées et comparées, en particulier en termes de complexité.

Questions possibles :

- Faire tourner les algorithmes présents dans le plan.
- Pourquoi 2Col est dans P ?

Plan détaillé :

I - Généralité et cadre
- Définition de graphe, sommet, arête
- Définition de adjacence, voisins
- Définition de orienté
- Définition chemin, cycle, clique
- Définition de connexe
- Définition de poids
- Représentation des graphes en machine

II - Parcours
- Parcours en largeur
- Exemple du plus court chemin (voir partie 4)
- Parcours en profondeur
- Exemple des composante fortement connexe
- Tri topologique

III - Arbres couvrants minimaux
- Définition
- Explication algorithme de Prim
- Explication algorithme de Kruskal

IV - Plus court chemin
- Explication algorithme de Dijkstra
- Explication algorithme de Floyd-Warshall

V - Problèmes NP-complets (voir appendice)
- 3-Coloration d'un graphe
- Cycle Hamiltonien
- Problème du voyageur de commerce

En annexe :

- Dessins de Graphe
- Parcours en largeur et en profondeur
- Composantes fortement connexe
- Arbre couvrant
- Prim et Kruskal

Développements :

- 3-Coloration d'un graphe
- Tri topologique

Références :

♥♥♥♥ - [8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST et Clifford STEIN, *Algorithmique*

♥♥♥♥ - [12] Christine FROIDEVAUX, Marie-Claude GAUDEL et Michèle SORIA , *Type de données et algorithmes*

♥♥♥ - [13] Michael GAREY et David JOHNSON, *Computers and Intractability : A Guide to the Theory of Np-Completeness*

Leçon 926
Analyse des algorithmes : complexité. Exemples.

Rapport du jury :

Il s'agit ici d'une leçon d'exemples. Le candidat prendra soin de proposer l'analyse d'algorithmes portant sur des domaines variés, avec des méthodes d'analyse également variées : approche combinatoire ou probabiliste, analyse en moyenne ou dans le pire cas.

Si la complexité en temps est centrale dans la leçon, la complexité en espace ne doit pas être négligée. La notion de complexité amortie a également toute sa place dans cette leçon, sur un exemple bien choisi, comme union find (ce n'est qu'un exemple).

J'ai rédigé le plan de cette leçon (voir ici)

Plan détaillé :

I - Généralité et cadre
- Définitions
- Notation de Landau
- Différentes approches (au pire, moyenne, ...)

II - Premières méthodes de calcul de complexité
- Calcul direct
- Complexité moyenne vu comme une espérance
- Résolution de récurrence
- masterthrm

III - Complexité amortie
- Exemple filé sur le tableau dynamique

1) Agrégat
   - Expliquer sur le tableau dynamique
2) Comptable
   - Définir les coûts et dépenses
3) Potentiel
   - Exprimer la valeur du potentiel dans l'exemple

IV - Amélioration de la complexité
- Borne inférieure pour les tris par comparaison
- Tri par tas
- Amélioration de la structure de donnée
- Exemple entre Morris-Pratt et Knuth-Morris-Pratt
- Compromis entre espace et temps
- Exemple de Fibonacci

Développements :

- Tri par tas
- Master Theorem

Références :

♥♥♥♥ - [8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST et Clifford STEIN, *Algorithmique*

♥♥♥♥ - [12] Christine FROIDEVAUX, Marie-Claude GAUDEL et Michèle SORIA , *Type de données et algorithmes*

♥♥♥ - [2] Luc ALBERT et Daniel KROB, *Cours et exercices d'informatique*

Leçon 927
Exemples de preuve d'algorithme : correction, terminaison.

Rapport du jury :

Le jury attend du candidat qu'il traite des exemples d'algorithmes récursifs et des exemples d'algorithmes itératifs.

En particulier, le candidat doit présenter des exemples mettant en évidence l'intérêt de la notion d'invariant pour la correction partielle et celle de variant pour la terminaison des segments itératifs.

Une formalisation comme la logique de Hoare pourra utilement être introduite dans cette leçon, à condition toutefois que le candidat en maîtrise le langage. Des exemples non triviaux de correction d'algorithmes seront proposés. Un exemple de raisonnement type pour prouver la correction des algorithmes gloutons pourra éventuellement faire l'objet d'un développement.

J'ai rédigé le plan de cette leçon (voir ici)

Plan détaillé :

I - Terminaison
- 1) Définitions
  - Exemples de Syracuse
  - Problème de l'arrêt
- 2) Ensemble bien fondé et algo récursif
  - Exemple de PGCD et Ackermann
- 3) Variant de boucle et algo itératif
  - Définition Variant de boucle
  - Théorème de terminaison
  - Exemple de la division euclidienne

II - Correction
- 1) Définitions
  - Différence entre partiel et total
  - Tri topologique
- 2) Algorithmes récursifs
  - Exemple PGCD et tri par insertion
- 3) Algorithmes itératifs
  - Définition invariant de boucle
  - Théorème pour la correction
  - Exemple Dijkstra

III - Logique de Hoare (voir appendice)
- 1) Langage IMP
  - Définitions
- 2) Sémantique naturelle (ou opérationnelle à grands pas)

Règles

Propriétés de langage déterministe

3) Triplets de Hoare (partiel)

Définition

Règles

Théorème de correction

Théorème de Complétude de la logique de Hoare

Développements :

- Tri topologique
- Complétude de la logique de Hoare

Références :

♥♥♥♥ - [8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST et Clifford STEIN, *Algorithmique*

♥♥♥♥ - [12] Christine FROIDEVAUX, Marie-Claude GAUDEL et Michèle SORIA , *Type de données et algorithmes*

♥♥♥ - [2] Luc ALBERT et Daniel KROB, *Cours et exercices d'informatique*

♥♥♥ - [18] Hanne Riis NIELSON et Flemming NIELSON, *Semantics with applications*

Leçon 928
Problèmes NP-complets : exemples et réduction.

Rapport du jury :

L'objectif ne doit pas être de dresser un catalogue le plus exhaustif possible ; en revanche, pour chaque exemple, il est attendu que le candidat puisse au moins expliquer clairement le problème considéré, et indiquer de quel autre problème une réduction permet de prouver sa NP-complétude.

Les exemples de réduction polynomiale seront autant que possible choisis dans des domaines variés : graphes, arithmétique, logique, etc. Si les dessins sont les bienvenus lors du développement, le jury attend une définition claire et concise de la fonction associant, à toute instance du premier problème, une instance du second ainsi que la preuve rigoureuse que cette fonction permet la réduction choisie et que les candidats sachent préciser comment sont représentées les données.

Un exemple de problème NP-complet dans sa généralité qui devient P si on contraint davantage les hypothèses pourra être présenté, ou encore un algorithme P approximant un problème NP-complet.

Questions possibles :

- Donner un exemple de problème dans NP dont on ne sait pas s'il est dans P ou NP-dur.
- Donner un exemple de problème qui est NP-dur mais pas dans NP.
- Montrer l'une des réductions du plan.
- Comment passe-t-on de CnfSat à 3Sat ?

Motivations :

On a classifié les problèmes décidables et indécidables durant la leçon 914. Ensuite on a étudié les problèmes décidables durant la leçon 915 en hiérarchisant les problèmes afin de déterminer leur complexité (liée au modèle de calcul : machine de Turing). Une classe particulière de cette hiérarchie est la classe des problèmes NP-complets. Ces problèmes sont dans la classe NP et ont la particularité que tout problème de NP peut être réduit à lui. Autrement dit, si on sait résoudre de manière efficace un problème NP-complet, on pourra résoudre tous les autres problèmes NP. Une grande question en informatique est de savoir si $\mathsf{P} = \mathsf{NP}$ ou si $\mathrm{P} \neq \mathsf{NP}$. Tous les problèmes ci-dessous sont détaillés dans l'appendice sur les problèmes NP-complets.

Plan détaillé :

I - Définitions et outils

1) Les classes P et NP

Définition de P

Exemple de CYK

Définition de NP

Exemple de Sat

2) Réduction polynomiale

Définition

Exemple

3) NP-complétude

Définition de la NP-complétude

Théorème de Cook

II - Quelques problèmes NP-complets

1) Logique

SAT est NP-complet par le théorème de Cook

Réduction à CNFSAT par la Transformation de Tseitin

Réduction à 3SAT

2) Graphes

3-Coloration d'un graphe

Réduction de 3SAT à VERTEXCOVER

Équivalence entre VERTEXCOVER, INDEPENDANTSET et CLIQUE

Réduction de VERTEXCOVER à HAMCIRC

Réduction à PVC

3) Paquets

Réduction de 3SAT à SUBSETSUM

Réduction de SUBSETSUM à BINPACKING

Réduction de SUBSETSUM à PARTITION

Réduction de PARTITION à SACÀDOS

III - Que faire en cas de NP-difficulté ?

1) Restreindre les entrées

Le problème 2SAT est dans P

Le problème HORNSAT est dans P

Le problème 2COL est dans P

2) Approximation (on va parler de problème d'optimisation ici)

Définition d'un problème d'optimisation

2-approximation de VERTEXCOVER

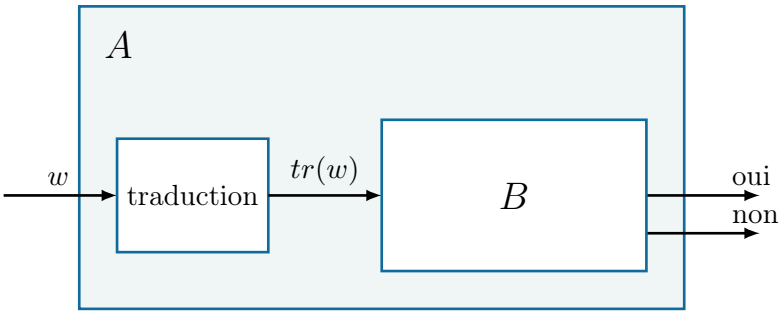
2-approximation du problème du voyageur de commerce euclidien

Remarques :

Pour la deuxième partie, j'écris dans le plan les définitions des problèmes avec entrée et sortie comme c'est montré dans l'appendice des problèmes NP-complets.

En annexe :

- un schéma de réduction



- un arbre des réductions des problèmes NP-complet (voir appendice)
- un exemple de graphe pour les problèmes VertexCover, IndependantSet et HamCirc
- un exemple de graphe biparti pour le problème 2Col

Développements :

- 3-Coloration d'un graphe
- Théorème de Cook
- Transformation de Tseitin

Remarques :
Il n'est sans doute pas très judicieux de mettre seulement le théorème de Cook et la transformation de Tseitin car ce sont tous les deux des développements sur la logique propositionnelle.

Références :

♥♥♥♥ - [13] Michael GAREY et David JOHNSON, *Computers and Intractability : A Guide to the Theory of Np-Completeness*

♥♥♥♥ - [8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST et Clifford STEIN, *Algorithmique*

♥♥♥ - [19] Christos PAPADIMITRIOU, *Computational Complexity*

Leçon 929
Lambda-calcul pur comme modèle de calcul. Exemples.

Rapport du jury :

Il s'agit de présenter un modèle de calcul : le lambda-calcul pur. Il est important de faire le lien avec au moins un autre modèle de calcul, par exemple les machines de TURING ou les fonctions récursives. Néanmoins, la leçon doit traiter des spécificités du lambda-calcul. Ainsi le candidat doit motiver l'intérêt du lambda-calcul pur sur les entiers et pourra aborder la façon dont il permet de définir et d'utiliser des types de données (booléens, couples, listes, arbres).

Questions possibles :

- Donner un terme qui n'a pas de forme normale
- Expliquer if_then_else
- Expliquer la fonction prédécesseur sur les entiers de Barendregt
- Expliquer l'équivalence avec les machines de Turing

Points essentiels :

- les subtilités de la substitution
- théorèmes de Church Rosser
- lien avec les autres modèles de calcul

J'ai fait un résumé des bases du λ-calcul dans l'appendice sur le λ-calcul.

J'ai rédigé le plan de cette leçon (voir ici)

Plan détaillé :

I - Syntaxe
- Termes
- Sous-termes
- Longueur
- Substitution
- α-équivalence

II - β-réduction
- β-réduction
- Exemples
- Théorème de Church Rosser I
- β-équivalence
- Formes normales
- Théorème de Church Rosser II

III - Représentations de données

booléens

$T := \lambda xy.x$

$F := \lambda xy.y$

if_then_else

Listes

Paires

Entiers de Church

Opérations sur ces entiers

Entiers de Barendregt

Opérations sur ces entiers

IV - Modèles de calcul

Thèse de Church

1) Fonctions μ-récursives

Définitions

μ-recursif $\implies$ λ-définissable

Théorème de Scott–Curry

2) Machine de Turing

équivalence

Développements :

- Théorème de Scott–Curry
- μ-recursif $\implies$ λ-définissable

Références :

♥♥♥ - [16] Richard LASSAIGNE et Michel DE ROUGEMONT, *Logique et fondements de l'informatique logique du 1er ordre, calculabilité et lambda-calcul*

♥♥♥♥ - [3] Henk BARENDREGT, *The Lambda Calculus*

♥♥♥♥ - [14] Roger HINDLEY et Jonathan SELDIN, *Lambda-Calculus and combinators, an introduction*

Leçon 930
Sémantique des langages de programmation. Exemples.

Rapport du jury :

L'objectif est de formaliser ce qu'est un programme : introduction des sémantiques opérationnelle et dénotationnelle, dans le but de pouvoir faire des preuves de programmes, des preuves d'équivalence, des preuves de correction de traduction.

Ces notions sont typiquement introduites sur un langage de programmation (impératif) jouet. On peut tout à fait se limiter à un langage qui ne nécessite pas l'introduction des CPOs et des théorèmes de point fixe généraux. En revanche, on s'attend ici à ce que les liens entre sémantique opérationnelle et dénotationnelle soient étudiés (toujours dans le cas d'un langage jouet). Il est aussi important que la leçon présente des exemples d'utilisation des notions introduites, comme des preuves d'équivalence de programmes ou des preuves de correction de programmes.

Questions possibles :

- Montrer que les instructions while b do S et if b then S; while b do S else skip sont équivalentes pour la sémantique opérationnelle à grands pas.
- Écrire des règles de sémantique à grands et petits pas pour gérer les instructions repeat S until b, et montrer que l'équivalence est préservée.
- Montrer que le langage IMP muni de la sémantique opérationnelle à petits pas (ou à grands pas) est déterministe, *i.e.* il y a unicité de l'état sortant s' après exécution d'une instruction S à partir d'un état s.

Motivations :

Une sémantique d'un langage de programmation est une fonction $\mathcal{S}$ qui à une instruction S et un état mémoire s associe un état mémoire s'. Ce dernier état mémoire est interprété comme un état mémoire sortant après exécution de l'instruction S sur l'état mémoire initial s. On peut définir une fonction de sémantique $\mathcal{S}$ de différentes manières. Nous verrons deux manières différentes :

- la sémantique « opérationnelle », qui décrit la suite des pas de calcul pendant l'exécution du programme.
- la sémantique « dénotationnelle », qui décrit le programme informatique comme une fonction au sens mathématique du terme.

On va étudier trois sémantiquesdifférentes (deux sémantiques opérationnelleset une sémantique dénotationnelled'un langagede programmation « jouet » dénoté IMP pour langage « IMPératif ».

J'ai fait un résumé des bases de sémantiques des langages de programmation dans l'appendice sur la sémantique.

J'ai rédigé le plan de cette leçon (voir ici)

Plan détaillé :

I - Le langage IMP
- 1) Syntaxe de IMP
  - Nombres
  - Variables
  - Expressions Arithmétiques
  - Expressions booléennes
  - Instructions
- 2) Sémantique dénotationnelle des expressions
  - $\mathbb{N}$, Var, $\mathcal{A}$, $\mathcal{B}$

II - Sémantique opérationnelle
- 1) A grands pas
  - Règles
  - Langage déterministe
  - Définition de la fonction sémantique
  - Définition d'instructions équivalentes sémantiquement
- 2) A petits pas
  - Règles
  - Langage déterministe
  - Définition de la fonction sémantique
  - Définition d'instructions équivalentes sémantiquement
  - Petites propriétés liées à cette sémantique
- 3) Équivalence des sémantiques
  - Équivalence entre sémantiques opérationnelles

III - Sémantique dénotationnelle
- 1) Fonction sémantique
  - Définition
- 2) Théorie de point fixe
  - CPO
  - Point fixe
- 3) Équivalence des sémantiques
  - Théorème : Équivalence entre sémantique dénotationnelle et à petits pas
  - Corollaire : Équivalence entre sémantique dénotationnelle et à grands pas

IV - Logique de Hoare
- 1) Triplets de Hoare
  - Définition
  - Précondition et postcondition
- 2) Règles de Hoare
  - Règles
- 3) Correction et complétude
  - Théorème de correction
  - Théorème de Complétude de la logique de Hoare

Développements :

- Équivalence entre sémantiques opérationnelles
- Complétude de la logique de Hoare

Références :

♥♥♥♥♥ - [18] Hanne Riis NIELSON et Flemming NIELSON, *Semantics with applications*

Leçon 931
Schémas algorithmiques. Exemples et applications.

Rapport du jury :

Cette leçon permet au candidat de présenter différents schémas algorithmiques, en particulier "diviser pour régner", programmation dynamique et approche gloutonne. Le candidat pourra choisir de se concentrer plus particulièrement sur un ou deux de ces paradigmes. Le jury attend du candidat qu'il illustre sa leçon par des exemples variés, touchant des domaines différents et qu'il puisse discuter les intérêts et limites respectifs des méthodes. Le jury ne manquera pas d'interroger plus particulièrement le candidat sur la question de la correction des algorithmes proposés et sur la question de leur complexité, en temps comme en espace.

Questions possibles :

- Donner d'autres paradigmes de programmation.
  Réponse : back tracking, branch and bound, ...
- Est-ce qu'on peut utiliser les trois paradigmes que je présente pour faire des algos d'approximation ?
  Réponse : oui, sac à dos avec programmation dynamique, vertex cover avec algorithmie gloutonne, TSPE avec diviser pour régner

J'ai dressé une liste non exhaustive de différents algorithmes suivant les 3 paradigmes que je présente dans mon plan dans l'appendice sur les schémas algorithmiques.

J'ai rédigé le plan de cette leçon (voir ici)

Plan détaillé :

I - Diviser pour régner
- Master Theorem
- Tri fusion
- Tri Rapide
- Dichotomie
- Strassen
- Fibonacci

II - Programmation dynamique
- Découpe de barre
- Plus longue sous séquence commune
- Fibaonacci
- Triangle de Pascal
- Cocke-Younger-Kasami
- Floyd Warshall

III - Algorithmie gloutonne
- Problème du sac à dos
- Prim
- Kruskal
- Codage de Huffman

Développements :

- Cocke-Younger-Kasami
- Master Theorem

Références :

♥♥♥♥ - [8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST et Clifford STEIN, *Algorithmique*

♥♥♥ - [4] Danièle BEAUQUIER, Jean BERSTEL et Philippe CHRETIENNE , *Éléments d'algorithmique*

Leçon 932
Fondements des bases de données relationnelles.

Rapport du jury :

Le cœur de cette nouvelle leçon concerne les fondements théoriques des bases de données relationnelles : présentation du modèle relationnel, approches logique et algébrique des langages de requêtes, liens entre ces deux approches.

Le candidat pourra ensuite orienter la leçon et proposer des développements dans des directions diverses : complexité de l'évaluation des requêtes, expressivité des langages de requête, requêtes récursives, contraintes d'intégrité, aspects concernant la conception et l'implémentation, optimisation de requêtes...

Questions possibles :

- Montrer l'indécidabilité du problème RelEqu.

  Réponse : voir commentaires de Indécidabilité de la satisfiabilité d'une requête

Motivations :

Une base de données est une structure qui permet d'organiser un grand nombre de données. Elles seront répertoriées par catégorie, appelée *attribut*, dans différents tableaux, appelés *relations*, ce qui formera la *base de données*. Cette dernière permettra de faire des liens entre les différentes informations qui y sont présentes. On va présenter différents moyens de récupérer des informations dans ces bases de données ainsi que les avantages et inconvénients de chaque moyen.

J'ai fait un résumé de ce qu'il faut savoir sur les bases de données dans l'appendice sur les bases de données.

J'ai rédigé le plan de cette leçon (voir ici)

Plan détaillé :

I - Formalisme

- Définitions de variable, attribut, domaine, schéma de relations
- Définition d'arité
- Définition de schéma de relation
- Définition de schéma de base de données
- Définition de tuple
- Définition de relation
- Définition d'une base de données

II - Requêtes conjonctives

1) Règles conjonctives

- Définition d'un tuple libre
- Définition d'une règle conjonctive
- Définition d'une valuation
- Définition de la sémantique d'une règle conjonctive

Définition du domaine actif

Propriété : $adom(q(\mathcal{I})) \subset adom(q) \cup adom(\mathcal{I})$

2) Calcul conjonctif

Définition d'une formule de calcul conjonctif

Définition d'une requête en calcul conjonctif

Définition d'une valuation

Définition de la sémantique d'une requête de calcul conjonctif

Théorème de Codd(1) : Le calcul conjonctif et les règles conjonctives sont équivalentes.

III - Algèbre SPC

Définitions de la sélection, la projection et le produit carthésien qui forment l'algèbre SPC

Prop : on peut effectuer la joointure naturelle, l'intersection avec les trois opérations

Remarque : il existe des requêtes insatisfiables

Théorème de Codd (2) : Il y a équivalence entre les requêtes par règles conjonctives, le calcul conjonctif et les requêtes satisfiables de l'algèbre SPC

IV - Calcul et algèbre relationnelle

1) Ajout de l'union

Définition de l'union U

Ajout du $\vee$ dans le calcul conjonctif

Définition d'une requête sûre

2) Ajout de la négation

Définition de la différence ensembliste D

Ajout du $\neg$ dans le calcul conjonctif

Restreint au domaine actif, l'algèbre relationnelle SPCUD et le calcul relationnnel sont équivalents

Indécidabilité de la satisfiabilité d'une requête

V - Optimisation

1) Stockage

Définition d'un B-arbre

B-arbres

2) Optimisation locale des requêtes SPC

Ordre des opérations pour minimiser les requêtes et le nombre de données que l'on manipule

En annexe :

- deux relations qui forment une base de données
- un B-arbre avec $t = 2$

Développements :

- B-arbres
- Indécidabilité de la satisfiabilité d'une requête

Références :

♥♥♥♥ - [1] Serge ABITEBOUL, Richard HULL et Victor VIANU, *Foundations of Databases*

♥ - [8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST et Clifford STEIN, *Algorithmique*

BIBLIOGRAPHIE

[1] Serge ABITEBOUL, Richard HULL, and Victor VIANU. *Foundations of Databases*. Addison-Wesley Publishing Company, 1995.

[2] Luc ALBERT and Daniel KROB. *Cours et exercices d'informatique*. Vuibert, 1998.

[3] Henk BARENDREGT. *The Lambda Calculus*. College Publications, 2012.

[4] Danièle BEAUQUIER, Jean BERSTEL, and Philippe CHRETIENNE. *Éléments d'algorithmique*. Masson, 1992.

[5] Mordechai BEN ARI. *Mathematical Logic for Computer Science*. Springer, 2012.

[6] Olivier CARTON. *Langages Formels : Calculabilité et complexité*. Vuibert, 2014.

[7] René CORI and Daniel LASCAR. *Logique Mathématique*. Dunod, 2003.

[8] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST, and Clifford STEIN. *Algorithmique*. Dunod, 2010.

[9] Maxime CROCHEMORE, Christophe HANCART, and Thierry LECROQ. *Algorithmique du texte*. Vuibert, 2001.

[10] René DAVID, Karim NOUR, and Christophe RAFFALLI. *Introduction à la logique : Théorie de la démonstration*. DUNOD, 2004.

[11] Jacques DUPARC. *Logique pas à pas*. PPUR, 2015.

[12] Christine FROIDEVAUX, Marie-Claude GAUDEL, and Michèle SORIA. *Type de données et algorithmes*. McGraw-Hill, 1990.

[13] Michael GAREY and David JOHNSON. *Computers and Intractability : A Guide to the Theory of Np-Completeness*. Freeman, 1979.

[14] Roger HINDLEY and Jonathan SELDIN. *Lambda-Calculus and combinators, an introduction*. Cambridge University Press, 2008.

[15] René LALEMENT. *Logique, Réduction, Résolution*. Masson, 2000.

[16] Richard LASSAIGNE and Michel DE ROUGEMONT. *Logique et fondements de l'informatique logique du 1er ordre, calculabilité et lambda-calcul*. Hermes Science Publications, 1993.

[17] Romain LEGENDRE and François SCHWARZENTRUBER. *Compilation : Analyse Lexicale et Syntaxique du Texte à sa Structure en Informatique*. Ellipses, 2015.

[18] Hanne Riis NIELSON and Flemming NIELSON. *Semantics with applications*. Springer, 2007.

[19] Christos PAPADIMITRIOU. *Computational Complexity*. Pearson, 1993.

[20] Pierre WOLPER. *Introduction à la calculabilité*. Dunod, 2006.