Questions examen

Michel Donnet

January 11, 2024

Contents

1 Principes généraux

1.1 Q1.1 Analyseur lexical, qu'est ce que c'est ?

L'analyse lexicale lit les caractères du source, et détermine la `séquence des terminaux` le composant.

Le lexique d'un langage définit les mots qui le composent

1. De manière explicite, en extension, comme: `( procedure if >= . } ;` etc…
2. De manière générique, en compréhension, comme:

a. identificateur
b. chaînes de caractères
c. constantes numériques

(Les langues naturelles ont par exemple des règles génériques pour le nombre (singulier-pluriel) et le genre (masculin-féminin)

En informatique, on parle de symboles terminaux. La justification de ce terme se verra sur les arbres de dérivation.

Cela conduit souvent à des grammaires à 2 niveaux:

1. grammaire définissant les classes de terminaux
2. niveau syntaxique lui-même

1.2 Q1.2 Analyseur syntaxique, qu'est ce que c'est ?

L'analyse syntaxique vérifie que la structure de cette séquence (définie par l'analyseur lexical) est coforme à la syntaxe du langage

Principes de l'analyse syntaxique:

1. reçoit de l'analyseur lexical une suite de symboles
2. reconnait dans cette suite la structure d'un texte

La construction d'un arbre syntaxique permet uniquement d'avoir une représentation commode pour comprendre l'analyse syntaxique.

feuilles: symboles terminaux

noeuds: symboles non-terminaux

C'est un arbre résultant de l'analyse syntaxique.

Fait apparaître la **sturcture syntaxique** d'un mot.

C'est une notion très importante !!!!!

On parle aussi d'**arbre de dérivation**.

$$\text{Soit } m \in V_T^*$$

alors $m \in L(G)$ ss'il existe un arbre syntaxique selon G dont le mot aux feuilles est m.

On dit alors que cet arbre est un arbre syntaxique pour m, ou que m admet cet arbre.

Répondre à $m \in L(G)$ = chercher un arbre syntaxique pour m.

En pratique:

Les analyseurs syntaxiques ne construisent pas un arbre syntaxique. L'arbre est simplement une représentation commode pour comprendre l'analyse syntaxique. Ils construisent une dérivation particulière, appelée gauche ou droite.

À une dérivation correspond *un seul arbre...*

À un arbre correspond potentiellement plusieurs dérivations. Les dérivations équivalent correspondent au même arbre. Ce qui change est simplement l'ordre des dérivations.

À chaque arbre syntaxique correspond une unique *dérivation gauche* et une unique *dérivation droite*

La syntaxe régit la forme des phrases

1. les phrases acceptables au vu de la définition syntaxique appartiennent au langage
2. les autres n'y appartiennent pas

La définition syntaxique d'un langage s'appuie sur:

1. les terminaux, définis au niveau lexical
2. des règles de bonne forme pour des séquences de terminaux, appelées `notions non-terminales`

La justification du terme `non-terminal` se verra sur les arbres de dérivation.

Toute description grammaticale textuelle d'un langage s'appuie sur une syntaxe spécifique, comme par exemple: (Bison)

Expression: non-terminal Expression PLUS Terme; PLUS est un terminal

Dans cette exemple, on décrit un fragment de la syntaxe d'expressions algébriques usuelles au moyen d'une spécification écrite

1.3 Rappel

Sémantique: signification véhiculée par les phrases du langage

analyse sémantique: contrôle la signification du code source

Sur-langage d'un langage donné: contient toutes les phrases de ce langage.

statique: à la compilation

dynamique: à l'exécution

(S'applique au typage)

Selon le langage, on mène toutes les tâches de compilation

1. de front en une passe: on ne fait qu'un passage sur le texte source
2. successivement en plusieurs passes: on fait des passes successives sur des formes intermédiaires représentant le code source. Certaines créent des structures de données utilisées par des passes ultérieures. On peut ainsi faire une analyse sémantique plus fine et du meilleur code objet.

`Tendance moderne` de compilation: contrôler le plus de choses statiquement, à la compilation,

Un `auto-interprète` ou `interprète méta-circulaire` est écrit dans le langage dont il peut interpréter les programmes.

Un `auto-compilateur` est écrit dans le langage qu'il peut compiler. => problème du bootstrap ou amorçage de la pompe… Si on veut écrire le compilateur dans son propre langage S

1. écrire d'abord un compilateur d'un sous-ensemble S' de S en E, et le compiler
2. écrire ensuite en S' le compilateur de S
3. c'est ce qu'on appelle un bootstrap du compilateur

Générateur de compilateurs: C'est une grammaire que l'on compile pour obtenir un compilateur

1.4 Notation postfixée

Incontournable !

C'est une écriture linéaire d'un graphe qui décrit l'ordre d'évaluation des cupérandes et cupérateurs.

1.4.1 Fonction stricte

Évalue toujours tous ses arguments d'appel

Ne peut donc être évaluée si l'un des arguments d'appel ne peut pas l'être

Un exemple typique de fonction non stricte est la conditionnelle "Si":

1. elle évalue toujours son premier argument d'appel, la condition
2. elle n'évalue que l'un ou (exclusif) l'autre de ses deuxièmes et troisième arguments, selon la valeur de la condition

2 Grammaires algébriques

2.0.1 Limites des langages réguliers

Les langages réguliers ne suffisent pas. ex: $\{a^n b^n | n \geq 0\}$: pas régulier (Pourrait servir pour vérifier si le nombre de parenthèses est correct dans une expression…)

Au fait on a besoin d'une forme de mémoire infini pour conter les a et les b car c'est non borné -> langage non régulier

Si la mémorisation est bornée, c'est un langage régulier

Lemme de pompage

Soit L un langage régulier. Alors il existe un entier k tq pour tout mot m de L de longueur >= k, il existe des mots

$$x, u, y \in \Sigma^*$$

avec $m = xuv$, $u \neq \epsilon$ et $\forall n$, $xu^n y \in L$

2.1 Q3.1: Définir formellement ce qu'est une grammaire

Une grammaire algébrique ou hors-contexte est un quadruplet

$$G = (V_T, V_N, P, S)$$

1. V_T et V_N sont des vocabulaires disjoints: V_T est l'ensemble des terminaux, V_N l'ensemble des non-terminaux
2. $S \in V_N$ est l'axiome, ou symbole de départ
3. $$P \subseteq V_N \times (V_N \cup V_T)^*$$
   est l'ensemble des (règles de) productions.

La grammaire engendre des mots de V_T

2.2 Définitions

2.2.0.1 Dérivation Une dérivation est une suite de dérivations directes (Suite d'application des règles d'inférence définies pour arriver à une preuve...). On peut avoir plusieurs dérivations pour un même mot.

2.2.0.2 Langage engendré par une grammaire Soit G une grammaire algébrique d'axiome S. Le *langage engendré* par G est défini par:

$$L(G) = m \in V_T^* | S \Rightarrow^* m$$

Déterminer si un mot m appartient au langage engendré, c'est déterminer si: $S \Rightarrow_G^* m$?

2.2.0.3 Dérivation directe (formellement) Dérivation directe: Soient

$$\beta, \beta' \in (V_N \cup V_T)^*$$

β se dérive directement en β' selon G, noté $\beta \Rightarrow_G \beta'$, s'il existe des mots

$$\gamma, \gamma' \in (V_N \cup V_T)^*$$

et une production

$$X \to \alpha$$

tels que:

1. $$\beta = \gamma X \gamma'$$
2. $$\beta' = \gamma \alpha \gamma'$$

(En gros, un symbole est remplacé par un ou plusieurs symboles...)

2.2.0.4 Dérivation (formellement) Soient $\beta, \beta' \in (V_N \cup V_T)$. Une suite de mots $\beta_0, \beta_1, \dots, \beta_n$ $(n \geq 0)$ est une dérivation de β en β' selon G si $\beta_0 = \beta, \beta_n = \beta'$, et

$$\beta_0 \Rightarrow \beta_1 \Rightarrow \cdots \Rightarrow \beta_n$$

2.2.0.5 Relation de dérivation

La relation de dérivation

$$\Rightarrow_G^*$$

est la fermeture réflexive et transitive de la relation de dérivation directe $\Rightarrow_G$

2.2.0.6 Langages algébriques

Les langages engendrés par les grammaires algébriques sont appelés *langages algébriques*

2.2.0.7 Grammaires équivalentes

On peut trouver 2 grammaires algébriques qui engendrent le même langage.

Deux grammaires sont *équivalentes* si elles engendrent le même langage

2.2.0.8 Remarques

Tout langage régulier est algébrique

1. possible d'utiliser une grammaire algébrique au lieu d'expressions régulières
2. Mais AF plus efficaces

Grammaire régulière: peut pas avoir plus d'un symbole non terminal. Ex: A -> a régulière; A -> aB régulière; A -> aBC non régulière

2.2.0.9 Dérivation gauche / droite (formellement)

Soient

$$\beta, \beta' \in (V_N \cup V_T)^*$$

et

$$\beta_0 \Rightarrow \beta_1$$

une dérivation de β en β'. Si à chaque étape on choisit de remplacer dans β_i le non-terminal

1. le plus à gauche: dérivation gauche (leftmost derivation) notée $\beta lm \Rightarrow^* \beta'$
2. le plus à droite: dérivation droite (rightmost derivation) notée $\beta rm \Rightarrow^* \beta'$

2.2.0.10 Récapitulatif

1. 1 dérivation -> 1 seul arbre
2. 1 arbre -> potentiellement plusieurs dérivations
3. 1 arbre -> unique dérivation gauche et unique dérivation droite
4. 1 mot -> potentiellement plusieurs arbres/interprétations

2.3 Q3.2: Classification des grammaires

Classification de Chomsky

$$\text{régulier} \subset \text{algébrique} \subset \text{contextuel} \subset \text{arbitraire}$$

Les grammaires algébriques expriment:

1. les prcupriétés structurelles des langages
2. mais pas leurs prcupriétés contextuelles
3. on les appelle aussi grammaires hors contexte

Par exemple, on ne peut pas exprimer par une grammaire algébrique

1. que toute variable utilisée a été déclarée
2. les vérifications de typage etc… Ces prcupriétés relèvent de l'analyse sémantique

Pour être sensible au contexte, il faut utiliser une grammaire contextuelle (Productions de la forme $\alpha \rightarrow \beta$ avec

$$|\alpha| \leq |\beta|$$

et

$$\alpha, \beta \in (V_N \cup V_T)^*$$

Ex: AB $\rightarrow$ BA

Les grammaires contextuelles

1. engendrent les langages contextuels
2. ne sont pas utilisées lors de l'analyse syntaxique
3. pas d'algorithme polynomial connu qui, pour tout mot, détermine si ce mot est engendré par une grammaire contextuelle donnée.

Il existe des grammaires arbitraires. Productions de la forme

$$\alpha \rightarrow \beta$$

avec

$$\alpha, \beta \in (V_N \cup V_T)^*$$

Ex: AB $\rightarrow \epsilon$

Ces grammaires engendrent l'ensemble de tous les langages.

2.4 Q3.3: Grammaires ambigües

Un mot est ambigü s'il admet plusieurs arbres syntaxiques.

Une grammaire est ambiguë si elle permet de dériver au moins un mot ambigu.

Un langage est ambigu si toutes les grammaires qui l'engendrent sont ambiguës.

On ne travaille que avec des grammaires non ambiguës !!!

Certaines grammaires sont clairement ambiguës, et d'autres sont mal conçues

$$G_1$$

$$\text{listInst} \rightarrow \epsilon | \text{inst listInst}$$
$$\text{inst} \rightarrow \text{affect PV} | \text{lecture PV}$$

$$\text{lecture} \rightarrow \text{READ IDENT}$$

$$G_2$$
$$\text{listInst} \rightarrow \text{inst listInst} | \text{inst}$$
$$\text{inst} \rightarrow \epsilon | \text{affect PV} | \text{lecture PV}$$

G_1: on ne peut pas ajouter ϵ où on veut !!!! (non ambiguë)

G_2: on peut ajouter ϵ où on veut !!! (ambiguë)

Prouver qu'une grammaire est ambiguë: trouver un mot qui admet au moins 2 arbres syntaxiques.

Prouver qu'une grammaire n'est pas ambiguë: faire une preuve… Très difficile…

⇒ Décider de l'ambiguité d'une grammaire est indécidable !!!!

Il y a possibilité de transformer une grammaire ambiguë en grammaire non-ambiguë.

Grammaire à cupérateurs: grammaire faisant intervenir des cupérateurs avec associativité et priorité.

Pour rendre une grammaire à cupérateur non ambiguë:

1. on ajoute un non terminal par niveau de priorité (S pour +, P pour * , etc)
2. les moins prioritaires en haut de l'arbre, proches de l'axiome
3. les plus prioritaires en bas de l'arbre, proches des feuilles
4. les axiomes et terminaux tout en bas (les atomes)
5. associativité gauche/droite ⇒ récursivité gauche/droite

Grammaires pathologiques: certains non-terminaux ne servent à rien. Cela est souvent dû à une erreur de conception. les règles improductives ne sont pas détectées…

2.4.0.1 Définition Une grammaire est dite réduite si elle ne contient pas de non-terminal improductif ou inaccessible.

2.4.0.2 Définition Un non-terminal $X \in V_N$ est improductif s'il n'existe pas de mot

$$u \in V_T^* \ tq \ X \Rightarrow^* u$$

(le langage engendré par X est vide). Il est productif sinon.

2.4.0.3 Calcul des productifs: idée X est productif:

1. s'il existe une production
$$X \to u \text{ avec } u \in V_T^*$$
2. ou s'il existe une production
$$X \to \alpha \text{ avec } \alpha \in (V_N \cup V_T)^*$$
tel que tous les non-terminaux apparaissant dans α sont productifs.

2.4.0.4 Définition d'un inaccessible Soit G une grammaire algébrique d'axiome S. Un non-terminal $X \in V_N$ est inaccessible s'il n'existe pas de mots

$$\alpha, \beta \in (V_N \cup V_T)^*$$

tels que $S \Rightarrow^* \alpha X \beta$. Il est accessible sinon.

2.4.0.5 Calcul des accessibles: idée X est accessible si:

1. c'est l'axiome
2. ou il existe une production $Y \to \alpha X \beta$ telle que Y est accessible.

Même principe d'itérations de point fixe que pour les accessibles mais on cherche les candidats en partie **droite** de production

Si on souhaite calculer une grammaire réduite, il faut supprimer d'abord les improductifs, puis les inaccessibles.