Théorie des langages et compilation

Cyril Banderier
Université de Paris XIII
Contact : Cyril.Banderier@inria.fr

Mars 2000

Voici un rapide panorama de choses que vous devez connaître pour l'examen et d'autres dont vous n'entendrez peut-être plus jamais parler.

1 Théorie des langages

La structure de base de la TL sont les mots, on peut en donner une définition mathématique

Définition 1 (Monoïde) *Un monoïde est un ensemble avec une loi interne associative (notée "."") et un élément neutre noté* ϵ.

Définition 2 (Liberté) *Le monoïde est dit libre s'il possède une base (un sous-ensemble) dont les éléments sont indépendants (c'est-à-dire qu'il n'y a pas de relations entre les éléments de la base ; néanmoins, doter une telle base de relations n'est pas stupide et donne en fait lieu à la riche théorie des groupes de représentation). Moralement, on a donc existence et unicité d'une factorisation sur un monoïde libre.*

Définition 3 (Langage, TP1) *En TL, la base est appelée alphabet, les éléments de cette base sont appelés lettres, la loi du monoïde est appelée concaténation (du latin catena=chaîne). Une concaténation de lettres forme un mot, l'élément neutre* ϵ *est ainsi logiquement dénommé le mot vide. Un ensemble de mots est appelé langage.*

Définition 4 (Rationnel) *On appelle* **Rat** *la plus petite classe des langages stable par union, produit, étoile et qui contient les lettres de l'alphabet. Un langage de cette classe est appelé rationnel.*

Un langage rationnel est associé à une expression rationnelle (du type $b(a+b)^*ab$).

Définition 5 (Reconnaissable) *On appelle* **Rec** *la classe des langages reconnus par un automate. Un langage de cette classe est appelé reconnaissaible.*

Il existe par ailleurs une théorie des automates qui reconnaissent des mots infinis (une définition due à Büchi dit qu'il suffit, pour qu'un mot soit accepté, qu'il passe une infinité de fois dans des états terminaux).

Théorème 6 (Kleene 1951, RAND reseach memorandum # 704, p. 80) **Rat**=**Rec**, *i.e. on a équivalence entre un langage reconnu par un automate et un langage correspondant à une expression rationnelle.*

Définition 7 (Grammaire algébrique) *On appelle* **Alg** *la classe des langages engendrés par une grammaire. Un langage de cette classe est appelé algébrique.*

Théorème 8 (Alg=Pile) *La classe des langages engendrés par une grammaire correspond à la classe des langages reconnus par un automate à pile.*

La classe des langages rationnels est incluse dans la classe des langages algébriques, comme le montre la proposition suivante.

Proposition 9 (Grammaire régulière, ex.1 TD2) **Reg**=**Rat**, *i.e. la classe des langages engendrés par une grammaire régulière (i.e. dont les membres droits sont du type aB ou b) correspond aux langages rationnels.*

La TL n'est pas purement "algébrique", il existe un lien avec l'analyse via ce qui suit. Soit a_n le nombre de mots de L de longueur n, on appelle série génératrice de L la fonction $F(z) = \sum_{n \geq 0} a_n z^n$

Théorème 10 (Chomsky et Schützenberger 1963) *Un langage rationnel a une série génératrice F rationnelle (i.e. F=un quotient de deux polynômes). Un langage algébrique a une série génératrice F algébrique (i.e. $\exists P \in Q[z][F]$,avec $P(F) = 0$).*

Exemple 11 (Dyck et Łukasiewicz) *Le langage de Dyck $S \rightarrow (S)S|\epsilon$, qui correspond aux mots bien parenthésés, a une série génératrice F qui vérifie $F(z) = zF(z)zF(z) + 1$, donc $z^2F(z) - F(z) + 1 = 0$ et ainsi*

$$F(z) = \frac{1 - \sqrt{1 - 4z^2}}{2z^2} = 1 + z^2 + 2z^4 + 5z^6 + 14z^8 + O(z^{10}).$$

Le langage de Łukasiewicz (cf notation polonaise des calculatrices Hewlett Packard) $S \rightarrow aSS|b$ a quant à lui la série génératrice algébrique

$$F(z) = \frac{1 - \sqrt{1 - 4z^2}}{2z} = z + z^3 + 2z^5 + 5z^7 + 14z^9 + O(z^{11}).$$

Dans les deux cas, on reconnaît la suite des nombres de Catalan $\frac{\binom{2n}{n}}{n+1}$.
Une suite quelconque de tirage à pile ou face correspond à l'expression rationnelle $(P + F)^*$ et donc à la série génératrice rationnelle $\frac{1}{1-2z}$. Remarquez que dans tous les cas, il y a un lien entre le rayon de convergence ρ de F et le nombre de mots de grande longueur ($a_n \sim \rho^{-n}$).

Théorème 12 (Propriétés de stabilité) *Les langages rationnels et algébriques sont stables par union, produit, étoile, miroir, homomorphisme, homomorphisme inverse, substitution rationnelle, résiduel... Les langages rationnels sont également clos par passage au complémentaire et intersection.*

On peut toutefois donner l'exemple de stabilités bien moins triviales : L étant un langage rationnel sur un alphabet Σ de cardinalité quelconque, on peut montrer avec la notion sophistiquée de transduction rationnelle (hors programme) que les langages suivants sont rationnels $\sqrt{L} = \{w \in \Sigma^* | ww \in L\}$, et $\frac{1}{2}L = \{u \in \Sigma^* | \exists v \in \Sigma^* \text{ avec } uv \in L \text{ et } |u| = |v|\}$.

En dehors des propriétes de stabilité (on dit aussi clôture ou fermeture), un moyen courant et pratique pour montrer qu'un langage (n') est (pas) rationnel (respectivement algébrique) sont les lemmes d'itérations.

Proposition 13 (Lemme d'itération pour les rationnels, ex.2 TD2) *Si L est rationnel alors il existe un certaine longueur k à partir de laquelle pour tout mot w de longueur $\geq k$ qui admet une décomposition $w = \alpha u \beta$ (avec α, u et β non vides), on a $\forall n, \alpha u^n \beta \in L$. Connu aussi sous le nom de lemme de l'étoile (pumping lemma). Application : $\{a^n b^n\}$ n'est pas rationnel.*

Proposition 14 (Lemme d'itération pour les algébriques) *Si L est algébrique, alors il existe une certaine longueur k à partir de laquelle pour tout mot w de longueur $\geq k$ qui admet une décomposition $w = \alpha u \beta v \gamma$ (avec soit u soit v non vide et soit α soit γ non vide), on a $\forall n, \alpha u^n \beta v^n \gamma \in L$. Applications : $\{a^n b^n c^n\}$ n'est pas algébrique ;* **Alg** *n'est pas stable par intersection et complémentaire.*

Définition 15 (Ambigüité, TD1) *Si dans une grammaire, il existe un mot qui peut être obtenu de deux façons différentes à partir de l'axiome (c'est-à-dire plus rigoureusement si ce mot possède plusieurs arbres de dérivation), on dit que la grammaire est ambigüe. Il existe des langages algébriques dont toutes les grammaires qui les engendrent sont ambigües ; un exemple de langage ainsi inhéremment ambigu est donné par $\{a^n b^m c^p \text{ avec } n = m \text{ ou } m = p\}$.*

Définition 16 (Complétude, TD3) *Un automate est dit complet si chacun de ses états possède une transition étiquetée par chacune des lettres de l'alphabet. Quitte à rajouter un état puits, tout automate peut être rendu complet.*

Définition 17 (Déterminisation, TD3) *Un automate est dit déterministe si on n'a jamais de choix, d'hésitation dans les transitions. Plus formellement, on a un seul état initial puis, pour chaque état, pas plus d'une transition par lettre. Tout automate peut être rendu déterministe.*

Définition 18 (Minimisation) *Un automate est dit minimal s'il n'y a pas d'automate avec moins d'états qui engendre le même langage. Tout automate peut être minimisé. Tout langage rationnel admet un unique automate minimal.*

Définition 19 (Récursivement énumérable) *Les grammaires contextuelles (i.e. dont les membres* gauches *des productions peuvent contenir des terminaux et des non-terminaux) engendre une classe de langages, appelés récursivement énumérables. Ces derniers correspondent aux machines de Turing.*

Conjecture 20 (Thèse de Church) *On conjecture que tout ce que le cerveau humain peut calculer est effectivement calculable par une machine de Turing. Tout langage humain est ainsi récursivement énumérable.*

2 Compilation

La compilation d'un programme est la succession des étapes suivantes :

programme source

⇓

analyse lexicale ("scanner")	Lex
analyse syntaxique ("parser")	Yacc
analyse sémantique génération du code intermédiaire optimisation du code génération de code	tpc

⇓

programme exécutable

L'optimisation de code consiste essentiellement à compacter l'arbre de dérivation (on le transforme en DAG, directed acyclic graph, graphe orienté sans cycle) afin de ne pas refaire deux fois un même calcul. Notez que le programme exécutable dépend profondément de la machine (architecture PC/Mac/Alpha...) et de son système d'exploitation (Dos/Windows/Unix/Linux) alors que le programme source tend vers une certaine universalité.

L'analyse lexicale lit les lettres du programme source une à une et reconnaît des "mots-clefs" ou des "nombres" (lexèmes ou token en anglais). On a alors traduit le programme source en un nouveau mot u (sur l'alphabet des lexèmes).

L'analyse syntaxique (qui présuppose que l'on s'est fixé une grammaire non-ambigüe) essaie alors de trouver *la* dérivation de l'axiome en u. L'idée de base consiste à promener une fenêtre de longueur k sur le mot u et d'en déduire quelle suite de règles de récritures ont été employées pour produire ce mot. Il y a deux stratégies.

Définition 21 (analyse montante) *On part de u et on essaie d'atteindre l'axiome.*

Exemple : analyse LR(k). La suite des réductions, prise à l'envers, donne la suite des règles à appliquer pour avoir une dérivation droite de l'axiome en u.

Définition 22 (analyse descendante) *On part de l'axiome et on essaie d'atteindre u.*

Exemple : analyse LL(k). La suite des règles donne une dérivation gauche de l'axiome en u.

Définition 23 (Tables d'analyse, TD4) *Pour nous aider dans cette tâche, il existe des tables d'analyse. Elles se construisent en calculant les Follow(k), First(k)...*

Il existe de nombreuses analyses : LL(k), LR(k), SLR... Signalons que Yacc (construit) et utilise une table LALR(1). Voir le TD4 pour une analyse LR(1).

Quand on compile un programme, on est amené à sauvegarder les différentes valeurs/types des variables, tout ceci se fait via une pile (voir le TD5 pour la gestion de la pile). On peut ainsi facilement faire les calculs associés à l'arbre d'évaluation lié à l'arbre de dérivation que vient de trouver l'analyse syntaxique (les $\$\$:= \$1 + \$3$ de Yacc).

Remarque 24 (Puissance/pile) *Quand on n'a pas de pile, on a la puissance des langages rationnels ; quand on a une pile ($\sim$ un compteur) on a la puissance des langages algébriques ; quand on a plusieurs piles, on a la puissance d'une machine de Turing.*

3 Bibliographie

Puisque vous rechignez à lire des livres/articles en anglais, je puis vous conseiller, en plus du cours en amphi et des TD/TP, les ouvrages suivants :

- Compilation. Cours de licence de Dominique Perrin. Jussieu, 1986-1987.
- Théorie des langages et des automates. J.-M. Autebert. Masson, 1994.
- Le petit manuel sur Lex et Yacc qui vous a été distribué.

4 Examen

J'espère que vous aurez un aperçu plus clair de la théorie des langages et de la compilation avec ces quelques pages. Après avoir lu ces quelques pages, assurez-vous que vous maîtrisez les techniques de bases (manipulation d'expressions rationnelles, déterminisation d'un automate, utilisation d'une table LR(1), trouver une dérivation d'un mot pour une grammaire donnée) et pouvez interpréter ce que fait un .l et un .y (e.g. exprlex.l et expr.y). Nota bene : ceci peut ne vous aider en rien a l'examen, ce n'est pas moi qui fais le sujet !
Bonnes révisions ! :-)