Mathématiques Discrètes 2

ESIAL 1^{ere} année

Rappels de cours et méthodes pratiques

Tony Bourdier alias GlanDyL
Version 3.0 – décembre 2005
Document rédigé en LaTeX

Table des matières

Première partie
Préambule

Ce document a été écrit dans le but de faciliter les révisions des étudiants en *Mathématiques Discrètes* et d'apporter quelques explications supplémentaires sur certains points. Il a été très largement "pompé" sur l'ouvrage de Pierre Marchand (*Mathématiques Discrètes - Dunod*) et n'a pas la prétention d'en dispenser la lecture.

Je tiens à remercier Pierre Marchand pour les corrections qu'il a apportées à la précédente version de ce document et pour l'indulgence dont il a fait preuve lors de cet exercice ;)

Enfin, je suis ouvert à toutes suggestions concernant ce poly et suis prêt à répondre à toutes les questions que le lecteur sera amené à se poser au cours de sa lecture.

ESIALement,

Tony Bourdier alias GlanDyL

Deuxième partie

Logique des propositions

I Rappels & définitions

I.1 Définition syntaxique

Définition : Soit P un ensemble quelconque, on appelle formule du calcul des propositions sur P et l'on note Prop(P) l'ensemble des mots engendrés par la grammaire suivante :
$Y \to V \mid F \mid a \mid \neg Y \mid \Rightarrow YY$ où a est un élément quelconque de P

I.2 Sémantique d'une formule du calcul des propositions

Définition : Soit $\delta : P \to \{0,1\}$ une application qui donne à chaque variable propositionnelle une valeur de vérité (0 ou 1). On dit que δ est une valuation des variables propositionnelles de P. Pour tout α de Prop(P), on appelle valeur de la sémantique de α sur la valuation δ et l'on note $[\alpha](\delta)$ la valeur obtenue en remplaçant dans α chaque symbole par son interprétation. Pour simplifier les notations, on écrit $\delta(\alpha)$ à la place de $[\alpha](\delta)$

Remarque : α est donc une écriture et $\delta(\alpha)$ est sa sémantique.

Définition : Soit α une formule du calcul des propositions
(1) on dit que $\delta : P \to \{0,1\}$ est un modèle de α ssi $[\alpha](\delta) = 1$
(2) on dit que α est une tautologie ssi tout $\delta : P \to \{0,1\}$ est un modèle de α
(3) on dit que α est non-contradictoire ssi il existe un modèle de α

Remarque :
(1) $\forall \mathcal{A} \subset$ Prop(P), $\delta : P \to \{0,1\}$ est un modèle de $\mathcal{A}$ ssi $\forall \alpha \in \mathcal{A}$, δ est un modèle de α
(2) $\forall \mathcal{A} \subset$ Prop(P), on dit que A est non-contradictoire ssi il existe un modèle de $\mathcal{A}$

Définition : Soit $\mathcal{A} \subset$ Prop(P) et α une formule du calcul des propositions, on dit que de $\mathcal{A}$ on déduit sémantiquement α et on note $\mathcal{A} \models \alpha$ ssi tout modèle de $\mathcal{A}$ est un modèle de α :

$$\forall \delta : P \to \{0,1\}, \quad (\forall \beta \in \mathcal{A}, [\beta](\delta) = 1) \Rightarrow ([\alpha](\delta) = 1)$$

Remarque :
(1) $\mathcal{A} \models \alpha$ ssi $\mathcal{A} \cup \{\neg\alpha\}$ est contradictoire.
(2) $\mathcal{A} \cup \{\alpha\} \models \beta$ ssi $\mathcal{A} \models (\alpha \Rightarrow \beta)$

Théorème : (de finitude ou de compacité) Soit $\mathcal{A} \subset$ prop(P)
(1) $\mathcal{A}$ admet un modèle ssi tout sous-ensemble de $\mathcal{A}$ admet un modèle

(2) soit α une formule du calcul des propositions, $\mathcal{A} \models \alpha$ ssi α se déduit sémantiquement d'un sous-ensemble fini de $\mathcal{A}$
(3) $\mathcal{A}$ est contradictoire ssi l'un de ses sous-ensemble finis est contradictoire

I.3 Systèmes formels, déduction syntaxique

I.3.1 Systèmes formels

Définition : On appelle système formel $\mathcal{S}$ un triplet $(\mathcal{E}, \mathcal{A}, \mathcal{R})$ où
- $\mathcal{E}$ est un ensemble non vide quelconque
- $\mathcal{A}$ est est sous-ensemble de $\mathcal{E}$ appelé ensemble des axiomes de $\mathcal{S}$
- $\mathcal{R}$ est un ensemble de relations sur $\mathcal{E}$ d'arité au moins deux

Un élément r de $\mathcal{R}$ s'appelle une règle de déduction de $\mathcal{S}$

Remarque : Si $(e_1, e_2, ..., e_n, e_{n+1})$ sont des éléments de $\mathcal{E}$ en relation suivant r, on écrit :

$$\frac{e_1, e_2, ..., e_n}{e_{n+1}}(r)$$

Ce qui se lit "de $e_1, e_2, ..., e_n$ se déduit e_{n+1} en utilisant la règle de déduction (r)"

Définition : Soit $\mathcal{S}$ un système formel $(\mathcal{E}, \mathcal{A}, \mathcal{R})$ et $\mathcal{E}' \subset \mathcal{E}$, on appelle preuve dans $\mathcal{S}$ en utilisant $\mathcal{E}'$ comme ensemble de prémisses une suite finie $e_1, e_2, ..., e_n$ d'éléments de $\mathcal{E}$ telle que pour tout e_i :
- soit e_i est un élément de $\mathcal{A}$, *ie* un axiome de $\mathcal{S}$
- soit e_i est un élément de $\mathcal{E}'$, *ie* une prémisse spécifique de la preuve dans $\mathcal{S}$
- soit il existe une règle de déduction (r) d'arité k+1 dans $\mathcal{R}$ telle que pour des indices $j_1, j_2, ..., j_k$ (strictement inférieurs à i) :

$$\frac{e_{j_1}, e_{j_2}, ..., e_{j_k}}{e_i}(r)$$

Remarque :
- On dit que e_n est la conclusion de la preuve et on note $\mathcal{E}' \vdash_S e_n$ et on lit : "l'élément de e_n de $\mathcal{E}$ se démontre dans $\mathcal{S}$ en utilisant les prémisses de $\mathcal{E}'$"
- On dit qu'un élément e de $\mathcal{E}$ est un théorème du système formel $\mathcal{S}$ ssi e se démontre dans $\mathcal{S}$ sans utiliser de prémisse. On écrira $\vdash_S e$ au lieu de $\emptyset \vdash_S e$

I.3.2 Utilisation des systèmes formels en logique

Définition : On appelle système formel du calcul des propositions un système formel $\mathcal{S} = (\mathcal{E}, \mathcal{A}, \mathcal{R})$ où $\mathcal{E}$ = Prop(P)

Remarque :
(1) $\mathcal{S}$ = (Prop(P), $\mathcal{A}, \mathcal{R}$) est un système formel valide ssi $\forall \mathcal{E}' \subset$ Prop(P) on a :

$$\mathcal{E}' \vdash_S \alpha \quad \Rightarrow \quad \mathcal{E}' \models \alpha$$

(2) $\mathcal{S}$ = (Prop(P), $\mathcal{A}, \mathcal{R}$) est un système formel complet ssi $\forall \mathcal{E}' \subset$ Prop(P) on a :

$$\mathcal{E}' \vdash_{\mathcal{S}} \alpha \quad \Leftrightarrow \quad \mathcal{E}' \models \alpha$$

II La Méthode de Résolution

II.1 Clauses, forme clausale d'une formule

Définition : Soit P un ensemble quelconque, on appelle clause sur P une formule c de Prop(P) telle que c s'écrive :

$$c = a_1 \vee a_2 \vee ... \vee a_k \vee \neg a_{k+1} \vee \neg a_{k+2} \vee ... \vee \neg a_{k+r}$$

où les a_i sont des éléments distincts de P que l'on appelle atomes (ou littéraux) de c. $\{a_1, ..., a_k\}$ sont les atomes positifs et $\{a_{k+1}, ..., a_{k+r}\}$ sont les atomes négatifs de c. On note CL(P) l'ensemble des clauses à littéraux dans P.

Définition : (Relation d'ordre de subsomption)
Soient deux clauses $c = a_1 \vee a_2 \vee ... \vee a_k \vee \neg a_{k+1} \vee \neg a_{k+2} \vee ... \vee \neg a_{k+r}$
et $c' = a'_1 \vee a'_2 \vee ... \vee a'_k \vee \neg a'_{k+1} \vee \neg a'_{k+2} \vee ... \vee \neg a'_{k+r}$
On dit que c' est subsumée par c ssi (conditions équivalentes) :
(1) (condition syntaxique) tout littéral de c apparaît avec le même signe dans c'
(2) (condition sémantique) tout modèle de c est modèle de c'

Remarque : Toute formule de Prop(P) est équivalente à une formule qui s'écrit sous la forme $c_1 \wedge c_2 \wedge ... \wedge c_q$ où les c_j sont des clauses.

Définition : Mettre une formule α sous forme clausale est, par définition, trouver un ensemble de clause $C(\alpha)$ dont la conjonction donne une formule équivalente à la formule de départ.

Proposition : $\alpha \in$ Prop(P) est une tautologie ssi le calcul de $C(\alpha)$ donne le résultat $\emptyset$

II.2 Méthode de résolution

Remarque : On note ■ la clause vide (clause toujours fausse)

Remarque : Le système formel que nous considérons n'a pas pour but de démontrer une formule quelconque mais seulement la clause vide ■.

Théorème : Soit $\mathcal{A} \subset$ Prop(P). On a $\mathcal{A} \models \alpha$ ssi l'union des ensembles de clauses associées aux formules de $\mathcal{A} \cup \{\neg\alpha\}$ est contradictoire.

Théorème : (Théorème de Robinson) Soit C un ensemble de clause, cet ensemble est contradictoire ssi en prenant C comme ensemble de prémisses, on peut déduire ■ en utilisant la résolution comme seule règle de déduction.

Définition : (Règle de résolution version Pierre Marchand)
Soient c, c' et c'' trois clauses, on écrit :

$$\frac{c, c'}{c''}(resolution)$$

ssi l'on peut trouver quatre clauses c_1, c_2, c_3, c_4 et un élément a de P tels que :

$$c = c_1 \vee a \vee c_2 \quad c' = c_3 \vee \neg a \vee c_4 \quad c'' = c_1 \vee c_2 \vee c_3 \vee c_4$$

Remarque : On dit que deux clauses peuvent entrer en résolution s'il en existe une troisième qui s'en déduit par la règle de résolution. Deux clauses peuvent entrer en résolution ssi elles ont un même littéral "a" qui est positif dans l'une et négatif dans l'autre. Alors la troisième clause, déduite des deux premières par résolution, est obtenue en recopiant à la suite des deux clauses de départ sauf le littéral commun qui est effacé.

Définition : (Règle de résolution version GlanDyL)

$$\frac{a \vee A, \neg a \vee B}{A \vee B}(resolution)$$

Ce qui donne en français :
Soient C_1 et C_2 deux clauses appartenant à une formule F mise sous forme clausale. S'il existe un atome a tel que $a \in C_1$ et $\neg a \in C_2$ alors la clause $R \equiv C_1 \setminus \{a\} \cup C_2 \setminus \{\neg a\}$ dite résolvante de C_1 et C_2 est une conséquence logique de F.

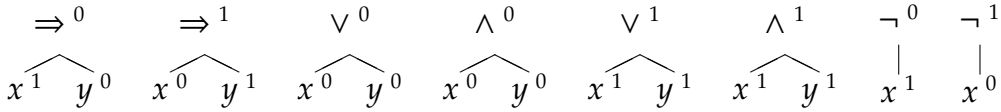
III Méthodes pratiques

III.1 Montrer qu'une formule est une tautologie

III.1.1 Algorithme de mise sous forme clausale

On considère une formule.
1) on dresse l'arbre de l'expression
2) on utilise la typologie suivante :
   on met 1 devant un symbole pour le nier
   on met 0 sinon
3) on transmet les négations en utilisant les règles de transmission
4) on transforme les symboles en $\wedge$ et en $\vee$ selon les règles de transformation
5) on associe à chaque noeud un ensemble de clauses calculé avec les clauses de ses fils en respectant la règle d'association des clauses. On remonte ainsi jusqu'à la racine dont le résultat est la forme clausale de la formule. Il faut arriver à Ø (ensemble vide de clause) pour pouvoir affirmer que la formule est une tautologie.

Règles de transmission de la négation par les symboles $\Rightarrow, \wedge, \vee$ et $\neg$ au niveau de l'arbre :



Règles de transformation des symboles $\Rightarrow, \wedge, \vee$ et $\neg$:

symbole d'origine	symbole de substitution
$\Rightarrow^0$	$\vee$
$\Rightarrow^1$	$\wedge$
$\vee^0$	$\vee$
$\vee^1$	$\wedge$
$\wedge^0$	$\wedge$
$\wedge^1$	$\vee$
$\neg$	*aucun* (après transmission, on supprime les $\neg$)

Règles d'association des clauses : Soient E_1 et E_2 deux ensembles de clauses :

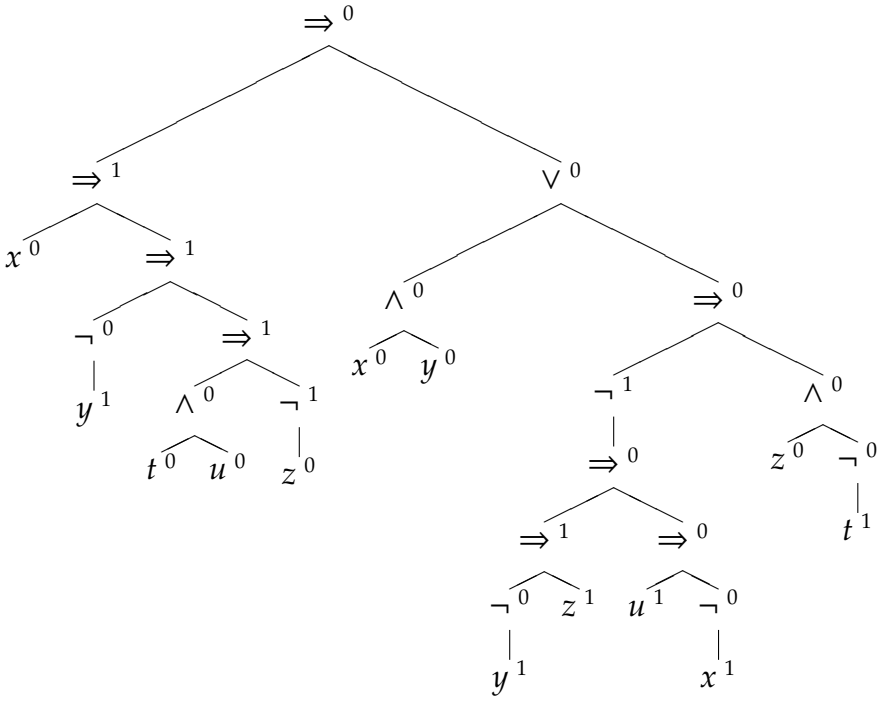
$$E_1 = \left|\begin{array}{c} A \\ B \end{array}\right. \text{ et } E_2 = \left|\begin{array}{c} C \\ D \end{array}\right. \text{, alors } E_1 \wedge E_2 = \left|\begin{array}{c} A \\ B \\ C \\ D \end{array}\right. \text{ et } E_1 \vee E_2 = \left|\begin{array}{c} A \vee C \\ B \vee C \\ A \vee D \\ B \vee D \end{array}\right.$$

Compte tenu des propriétés évidentes suivantes (qui ne sont que des rappels de l'algèbre de Boole avec des notations 'arborescentes' et la règle de résolution) :

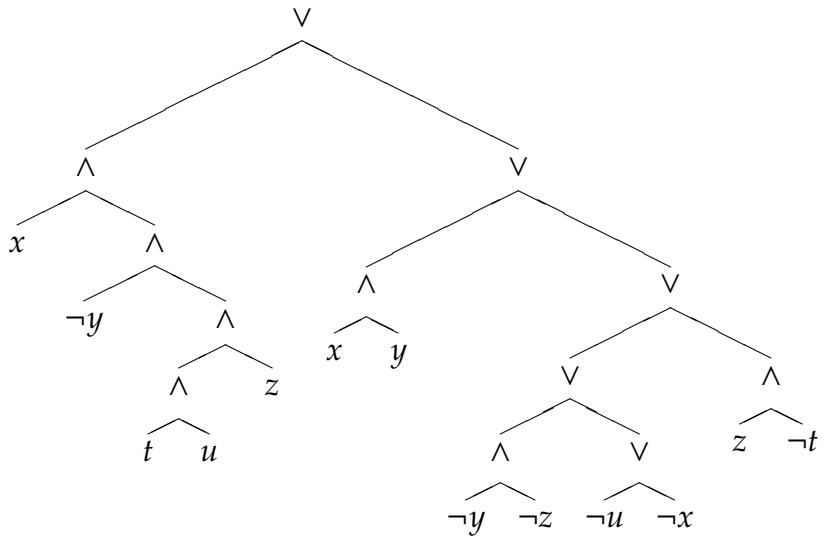
$$\left|\begin{array}{c} x \\ \neg\, x \end{array}\right. \models \left|\ \blacksquare\ \right. , \quad \left|\begin{array}{c} A \\ x \vee \neg x \end{array}\right. \models \left|\ A\ \right. , \quad \left|\ x \vee \neg x \vee z \vee t\ \right. \models \varnothing, \quad \left|\ x \vee \neg\, x\ \right. \models \varnothing$$

III.1.2 Exemple

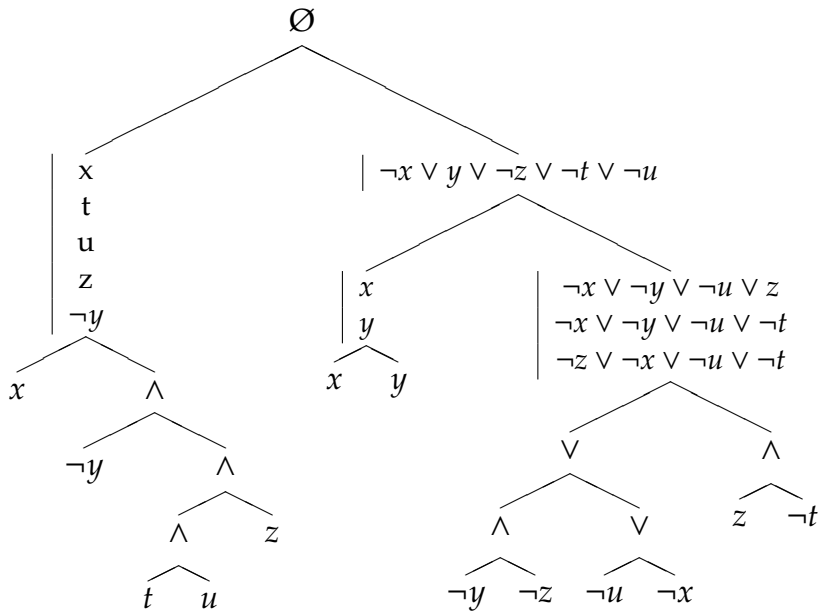
On considère la formule suivante :
$A = [x \Rightarrow (\neg y \Rightarrow ((t \wedge u) \Rightarrow \neg z))] \Rightarrow [(x \wedge y) \vee \{\neg((\neg y \Rightarrow z) \Rightarrow (u \Rightarrow \neg x)) \Rightarrow (z \wedge \neg t)\}]$
On commence donc par dresser l'arbre de l'expression et par transmettre les négations.



on utilise maintenant les règles de transformation :



A défaut d'écrire l'ensemble des clauses de chaque noeud, voici les principaux :



Troisième partie
Calcul des prédicats du premier ordre

IV Rappels & définitions

IV.1 Introduction à la GlanDyL

Je vous propose une petite mise au point de vocabulaire et de syntaxe afin de faciliter la lecture et la compréhension des définitions ultérieures.

IV.1.1 Langage du premier ordre

Un langage $\mathcal{L}$ du premier ordre est défini par :

– **des connecteurs propositionnels** : Ce sont les symboles que l'on s'autorise en 'Calcul des Propositions', à savoir $\neg$, $\vee$, $\wedge$, $\Rightarrow$ et leurs composés.

– **des quantificateurs** : Il s'agit des symboles 'pour tout' : $\forall$ et 'il existe' : $\exists$

– **un ensemble** C **de constantes** : Il s'agit des symboles qui commencent par une majuscule (terminologie pas toujours respectée) et qui ont une interprétation constante : $\mathcal{V}$, $\mathcal{F}$, *Rocco*, *Ginette* ...

– **un ensemble** X **de variables (propositionnelles) :** Ce sont les symboles commençant par une minuscule : x, y... et dont la sémantique dépend de l'interprétation

Rappel : L'arité d'une fonction est le nombre d'arguments de la fonction. (être d'arité n $\Leftrightarrow$ être n-aire)

– **un ensemble** F **de symbole de fonction** : Comme dans la logique des propositions, l'ensemble des symboles de fonctions ou schémas fonctionnels est $F = (F_n)_{n\in\mathbb{N}}$ où F_n est l'ensemble des symboles de fonctions d'arité n.

– **un ensemble** R **de symbole de relation** ou encore symboles de prédicat.

IV.1.2 Termes

Définition : L'ensemble des termes du langage, noté $\mathcal{T}(\mathcal{L})$ est le plus petit ensemble de mots écrits avec le vocabulaire $X \cup F \cup C \cup \{(\} \cup \{)\} \cup \{,\}$ tel que :
- chaque variable est un terme
- chaque constante est un terme
- si $f \in F_n$ et si $t_1, t_2, ..., t_n$ sont des termes, alors $f(t_1, t_2, ..., t_n)$ est un terme.

Définition : On appelle substitution une application remplaçant dans un schéma fonctionnel des variables par des termes :

$$\sigma : X \to F(X) \text{ telle que } \sigma(f(x_1, ..., x_n)) = f(t_1, ..., t_n)$$

où $t_1 = \sigma(x_1), ..., t_n = \sigma(x_n)$

IV.1.3 Atomes

Définition : On appelle atome un objet de la forme $r(t_1, ..., t_n)$ où r est un symbole de relation d'arité n et $t_1, ..., t_n$ des termes de $\mathcal{L}$. On note $At(\mathcal{L})$ cet ensemble.

Remarque : Une formule atomique est une suite d'atomes[1].

Remarque : L'ensemble des formules du langage $\mathcal{L}$ noté $For(\mathcal{L})$ est le plus petit ensemble de mots qui contient les formules atomiques et qui est clos par les opérations suivantes :
- Si F et G sont des formules, alors $\neg F$, $(F \vee G)$ et $(F \wedge G)$ sont des formules
- Si F est une formule et x une variable, alors $\exists x\ F$ et $\forall x\ F$ sont des formules

IV.1.4 Liberté

Définition : L'occurence d'une variable $x \in X$ est dite libre ssi elle n'est pas à la portée d'un quantificateur. Sinon, elle est dite liée.

Remarque : La plupart du temps, une variable est à la portée ou dans le champs d'un quantificateur si elle se trouve à droite de ce dernier.

Définition : Une formule est dite close ssi elle n'a aucune variable libre. Une formule est ouverte ssi elle comporte au moins une variable libre.

Définition : Pour toute formule α de $For(\mathcal{L})$, on note $V(\alpha)$ le sous-ensemble de X des variables apparaissant dans α, $VL(\alpha)$ l'ensemble des variables libres de α et $VM(\alpha)$ l'ensemble des variables liées.

Proposition : Par définition, les variables libres d'un terme sont exactement les variables de ce terme : $\forall t \in \mathcal{T}(\mathcal{L}), VL(t) = V(t)$. De même, les variables libres d'un atome sont exactement les variables de cet atome : $\forall a \in At(\mathcal{L}), VL(a) = V(a)$.

[1] Définition différente de celle de P. Marchand

IV.1.5 Formule polie

Définition : Une formule α de $For(\mathcal{L})$ est polie ssi :
– les deux ensembles $VL(\alpha)$ et $VM(\alpha)$ sont disjoints, autrement dit, il n'existe pas de variables dont une occurence soit libre et une autre occurence soit liée.
– deux occurences d'une même variable liée correspondent à la même occurence de quantificateur.

Définition : Soit α une formule dont les variables libres sont $VL(\alpha) = \{x_1, ..., x_n\}$
– On appelle clôture universelle de α de l'on note $\forall_{VL(\alpha)}(\alpha)$ la formule :

$$\forall x_1 \forall x_2 ... \forall x_n, (\alpha)$$

– On appelle clôture existentielle de α de l'on note $\exists_{VL(\alpha)}(\alpha)$ la formule :

$$\exists x_1 \exists x_2 ... \exists x_n, (\alpha)$$

Remarque : Si α' est la cloture universelle de α, alors pour tout A, $A \models \alpha$ ssi $A \models \alpha'$

IV.2 Procédures de base

Définition : L'interprétation de Herbrand de F(X), notée $\mathbb{H}$ est telle que
– l'ensemble de base de cette interprétation est $F(X)$
– chaque symbole f_n de F_n est interprété par l'application $\mathbb{H}(f_n) : F(X)^n \to F(X)$ telle que :

$$\mathbb{H}(f_n)(t_1, t_2, ..., t_n) = f_n(t_1 t_2 ... t_n)$$

Remarque : Dans le cadre de l'interprétation de Herbrand, une valuation des variables est exactement la définition d'une substitution.
De plus, on a pour tout schéma fonctionnel α de $F(X)$: $[\alpha](\mathbb{H})(\sigma) = \sigma(\alpha)$

Définition : Soient t et t' deux schémas fonctionnels. On dit que t' est filtré par t ssi il existe une substitution σ telle que $\sigma(t) = t'$

Définition : Soient t et t' deux schémas fonctionnels. On dit que t et t' s'unifient ssi il existe une substitution σ telle que $\sigma(t) = \sigma(t')$

V Interpretation et sémantique des formules d'un langage $\mathcal{L}$

Rappel : $E^X = \{f, f : X \to E\}$ (ensemble des applications de X dans E)

Définition : On dit que le sous-ensemble E' de E^X ne dépend que de $X' \subset X$ ssi δ et δ' étant deux éléments de E^X,

$$\delta_{|X'} = \delta'_{|X'} \text{ et } \delta \text{ dans } E' \text{ implique } \delta' \text{ dans } E'$$

V.1 Définition d'une interprétation

Définition : Pour définir une interprétation I d'un langage $\mathcal{L}$, on considère :
– un ensemble E non vide appelé ensemble de base de l'interprétation I. C'est dans cet ensemble que les variables pourront prendre une valeur.
– pour tout symbole f_n de F_n, une application $I(f_n) : E^n \to E$ qui est l'interprétation du symbole f_n. $I(f_n)$ est donc la fonction de symbole f_n qui prend en argument les n interprétations des n variables de f_n
– pour tout symbole r_n de R_n, une application $I(r_n) : E^n \to \{0, 1\}$ qui est l'interprétation du symbole r_n et qui pour n valeurs de E nous dit si la relation est vérifiée ... ou pas.

Définition : Etant donné une interprétation I, on a associe à toute formule α de $\mathcal{L}$ un sous-ensemble de E^X noté $[\![\alpha]\!](I)$ et appelé valeur de α dans la sémantique I. La sémantique d'une formule α est donc l'ensemble des façons de donner des valeurs aux variables de α pour rendre la formule vraie.

Remarque : $[\![\alpha]\!](I)$ est un sous-ensemble de E^X et pas 0 ou 1 comme dans la logique des propositions !

Remarque : L'ensemble $[\![\alpha]\!](I)$ ne dépend que des variables libres de α

V.2 Cas pratique et exemple

On part d'une formule α, on choisit une interpretation I, on réécrit α en remplaçant chaque symbole de fonction et de relation par son interprétation dans I mais en gardant les symboles de logique propositionnelle ($\vee, \wedge, \Rightarrow, \neg, ...$) et de logique du premier ordre ($\forall, \exists$). On obtient donc une formule interprétée que l'on notera $I(\alpha)$ qui s'appelle le prédicat associé à α dans l'interprétation I. On réfléchit aux valeurs rendant vrai ce prédicat et, dans les cas simples, on trouve ainsi la valeur de la sémantique de α dans I.

Exemple : Soit α la formule $s(x, a) \wedge (\forall y)(d(y, x) \Rightarrow eq(y, b) \vee eq(y, x))$
Dans cette formule, a et b sont des symboles de constantes, s, d et eq sont des symboles de relation d'arité deux. On va étudier, parmi l'infinité d'interprétations possibles de cette formule, une interprétation possible de cette formule.
– Ensemble de base E = $\mathbb{N}$
– $I(a) = 2, \quad I(b) = 1$
– $I(s) : (p, q) \to 1$ ssi $p \geq q$
– $I(d) : (p, q) \to 1$ ssi p est un diviseur de q
– $I(eq) : (p, q) \to 1$ ssi $p = q$

Dans cette interprétation, la formule α devient le prédicat suivant :

$$I(\alpha) = x \geq 2 \wedge (\forall y)(y|x \Rightarrow y = 1 \vee y = x)$$

Remarque : Les différents prédicats $[\![\alpha]\!](I)$ obtenus pour une même formule α avec des interprétations I différentes peut être :
− toujours faux. On dira alors que la sémantique de α dans cette interprétation est 0 (ou faux). (par exemple, si l'on obtient un prédicat du style $(\forall x \geq y, y \geq 0 \Rightarrow x < 0)$)
− toujours vrai. On dira alors que la sémantique de α dans cette interprétation est 1 (ou vrai). (par exemple, si l'on obtient un prédicat du style $\forall y \geq 0, \exists x, x = y + 1)$)
− ni l'un, ni l'autre (cas général qui est le cas de l'exemple ci-dessus). Le prédicat obtenu ne peut alors être qualifié ni de vrai, ni de faux. Il spécifie en fait un sous-ensemble de E^X qui est l'ensemble des façons de donner des valeurs aux variables pour rendre la formule vraie.

V.3 Modèle d'une formule

Définition : Soit $\mathcal{L}$ un langage du premier ordre.
(1) Soit α une formule de $\mathcal{L}$, on dit que l'interprétation I est un modèle de α ssi $[\![\alpha]\!](I) = E^X$
(2) Soit A un sous-ensemble de $\mathcal{L}$, on dit que l'interprétation I est un modèle de A ssi $\forall \alpha \in A, [\![\alpha]\!](I) = E^X$, c'est à dire si I est un modèle de chacune des formules de A.
(3) Soit A un sous-ensemble de $\mathcal{L}$, on dit que A est contradictoire ssi A n'a pas de modèle.

Définition : Soit α une formule, A un ensemble de formules.
(1) On dit que α se déduit sémantiquement de A et l'on écrit A $\models \alpha$ ssi tout modèle de l'ensemble A est un modèle de α
(2) En particulier, on dit que α est un théorème de la logique du premier ordre ssi toute interprétation est un modèle de α. On écrira alors $\models \alpha$ plutôt que $\emptyset \models \alpha$
(3) On dit que deux formules α et β sont équivalents ssi la formule $\alpha \Leftrightarrow \beta$ est un théorème de logique, à savoir ssi $\models \alpha \Leftrightarrow \beta$

VI Vers les méthodes de déduction syntaxique

VI.1 Système formel de déduction en logique du premier ordre

On suppose que les opérateurs primaires sont $\neg$, $\Rightarrow$ et $\exists$. Les autres symboles étant considérés comme des raccourcis d'écriture.

Définition : On considère le système formel $\mathcal{S} = (For(\mathcal{L}), A, R)$ avec
$A = \{\mathcal{V}, \neg\mathcal{F}\} \bigcup_{i=1}^{4} A_i \cup \{A_x[a] \Rightarrow \exists x A; A \in For(\mathcal{L}) \text{ et } a \in F_0\}$
($A_x[a]$ signifie simplement que l'on a remplacé dans A toutes les occurences libres de la variable x par la constante a. Les axiomes $A_x[a] \Rightarrow \exists x A$ s'appellent axiomes de substitution)
$A_1 = \{A \Rightarrow (B \Rightarrow A); A, B \in For(\mathcal{L})\}$
$A_2 = \{(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C)); A, B, C \in For(\mathcal{L})\}$
$A_3 = \{A \Rightarrow (\neg A \Rightarrow B); A, B \in For(\mathcal{L})\}$
$A_4 = \{(A \Rightarrow B) \Rightarrow ((\neg A \Rightarrow B) \Rightarrow B); A, B \in For(\mathcal{L})\}$
R = {modus ponens, $\exists$−introduction} où ces deux règles de déductions sont définies par :
− modus ponens : $(A, A \Rightarrow B) \rightarrow B$
− $\exists$−introduction : on déduit β de α ssi $\alpha = A \Rightarrow B$ et $\beta = \exists x A \Rightarrow B$ et x n'est pas une variable libre de B et A et B sont des formules quelconques de $For(\mathcal{L})$

Définition : Soit $\mathcal{L}$ un langage du premier ordre. On appelle interprétation libre ou de Herbrand une interprétation $\mathbb{H}$ tel que :
(1) le domaine de $\mathbb{H}$ est l'ensemble des termes de $\mathcal{L}$: $|\mathrm{H}| = \mathcal{T}(\mathcal{L})$
(2) chaque terme $f(t_1, ..., t_n)$ est interprété par $f(t_1 ... t_n)$
(3) l'interprétation des relations est libre

Théorème de Herbrand : Soit A un ensemble des formules sans quantificateur. Cet ensemble admet un modèle ssi il admet un modèle libre.

Proposition : Soit α une formule du premier ordre sans quantificateur (ou A un ensemble de formules du premier ordre sans quantificateur), il existe un ensemble de clauses C tel que les modèles de α (ou de A) soient exactement ceux de C.

VI.2 Transformation de Skolem - Skolemisation

Le but de cette partie est de supprimer les quantificateurs d'une formule pour pouvoir se ramener à des cas que l'on a déjà étudiés afin de savoir si une formule est ou non un théorème.

VI.2.1 Formes prénexes

Définition : α est une formule prénexe ssi α s'écrit sous la forme :

$$(Q_1 x_1)(Q_2 x_2)...(Q_n x_n)\alpha'$$

où α' est une formule sans quantificateur et $Q_1, ..., Q_n$ une suite de quantificateur ($\forall$ ou $\exists$)

Remarque GlanDyLesque : Mettre sous forme prénexe consiste tout simplement à regrouper tous les quantificateurs au début de la formule ... en veillant bien à ne pas modifier les relations qu'ont les quantificateurs entre eux !
($(\exists x, \forall y, x \Rightarrow y)$ [ici, y dépend du x] n'est pas équivalent à $(\forall y, \exists x, x \Rightarrow y)$ [cette fois, y ne dépend de personne et x dépend de y])

Proposition : Pour toute formule, il existe une formule prénexe équivalente.

VI.2.2 Skolémisation

Définition : (Version GlanDyL) Version de P.Marchand p.196
La Skolémisation ou mise en forme de Skolem d'une formule prénexe consiste à supprimer les quantificateurs existentiels en remplaçant toutes les occurrences des variables associées à ce prédicat par un symbole de fonction qui dépend de toutes les variables quantifiées universellement qui précèdent. ($Q_1, ..., Q_n$ sont des quantificateurs)

$$Q_1x_1...Q_nx_n\exists xP(x_1, ..., x_n, x) \text{ donne } Q_1x_1...Q_nx_nP(x_1, ..., x_n, f(x_{j_1}, ..., x_{j_n}))$$

où $x_{j_1}, ..., x_{j_n}$ sont les variables correspondant aux quantificateurs universels.

Théorème de Skolem : Soient α une formule et α' une formule équivalement skolémisée, alors α est satisfiable ssi α' est satisfiable[2].

VI.3 La méthode de résolution

Il s'agit de démontrer la validité d'un raisonnement de la forme $A \models \alpha$ en utilisant des méthodes syntaxiques. Pour cela, il faut :
(1) prendre la clôture universelle α' de α et utiliser le fait que $A \models \alpha$ ssi $A \models \alpha'$
(2) utiliser le lemme disant que $A \models \alpha'$ est contradictoire ssi $A \cup \{\neg\alpha'\}$ est un ensemble contradictoire de formules
(3) utiliser le fait que $A \cup \{\neg\alpha'\}$ est contradictoire ssi les formes skolémisées des formules considérées forment aussi un ensemble contradictoire.
(4) utiliser l'algorithme de mise sous forme clausale et skolémisée et le fait qu'un ensemble de formules est contradictoire ssi l'union des ensembles de clauses associées aux formules traitées est aussi contradictoire
(5) utiliser le système formel décrit ci-après pour démontrer que la clause ■ se déduit de l'ensemble de clauses construit précédemment.

Définition : Le système formel $\mathcal{S} = (Cl(\mathcal{L}), \mathcal{A}, R)$, dit système de Robinson, fonctionne sur $Cl(\mathcal{L})$ qui désigne l'ensemble des clauses du langage $\mathcal{L}$, n'a pas d'axiome ($\mathcal{A} = \emptyset$) et a trois règles de déduction formant l'ensemble $R = \{res, fac^+, fac^-\}$ décrites ci-après.

Règle de résolution 'res' en LP1 (Version GlanDyL) : version de P. Marchand p.200
Soient C_1 et C_2 deux clauses appartenant à une formule F mise sous forme clausale. S'il existe deux atomes (ou littéraux) a_1 et a_2 unifiables de susbtitution σ tels que $a_1 \in C_1$ et $\neg a_2 \in C_2$ alors la clause $R \equiv \sigma(C_1) \setminus \{a_1\} \cup \sigma(C_2) \setminus \{\neg a_2\}$ dite résolvante de C_1 et C_2 est une conséquence logique de F, autrement dit, si $\sigma(a_1) = \sigma(a_2)$:

$$\frac{a_1 \vee A, \neg a_2 \vee B}{\sigma(A) \vee \sigma(B)}(resolution)$$

[2]Rappel : Une formule est satisfiable ssi elle possède au moins un modèle.

Règle de factorisation positive 'fac$^+$' (Version GlanDyL) : version de P. Marchand p.200
Soit C une clause telle que $C = A \vee a_1 \vee a_2$. Si $\sigma(a_1) = \sigma(a_2)$, (si a_1 et a_2 s'unifient) alors

$$\frac{A \vee a_1 \vee a_2}{\sigma(A) \vee \sigma(a_1)}(fac^+)$$

Règle de factorisation négative 'fac$^-$' (Version GlanDyL) : version de P.Marchand p.200
Soit C une clause telle que $C = A \vee \neg a_1 \vee \neg a_2$. Si $\sigma(\neg a_1) = \sigma(\neg a_2)$, (si $\neg a_1$ et $\neg a_2$ s'unifient) alors

$$\frac{A \vee \neg a_1 \vee \neg a_2}{\sigma(A) \vee \sigma(\neg a_1)}(fac^-)$$

Théorème de Robinson (idem Logique des Propositions) :
Soit C un ensemble de clauses. Cet ensemble est contradictoire ssi le système formel ci-dessus permet de démontrer la clause vide ■ en utilisant C comme ensemble de prémisses.

VII Méthodes pratiques

VII.1 Montrer qu'une formule est ou non un théorème

VII.1.1 Méthode générale

Pour montrer qu'une formule est un théorème, il faut procéder en 7 étapes :
(1) écrire la formule sous forme arborescente
(2) rendre la formule polie
(3) chercher les variables libres et considérer la clôture universelle de la formule
(4) nier la formule
(5) descendre les négations
(6) skolémiser la formule
(7) remonter les clauses

Examinons chacune des ces étapes.
(1) Ecrire sous forme arborescente, tout le monde sait faire. Passons.
(2) Rendre la formule polie.[3] Autrement dit, si une variable apparaît sous l'influence de quantificateurs dans des branches différentes, il suffit de donner un nom différent à cette variable pour chaque branche.

(3) Chercher les variables libres. Autrement dit, on parcourt l'arbre et on retient les variables dont aucune occurence n'accompagne un quantificateur. On les note toutes (en général, peu nombreuses) à la racine pour les considérer comme quantificateurs universels (c'est à dire des $\forall$)[4].

[3] Faisons un petit rappel de cours : Une formule α de $For(\mathcal{L})$ est polie ssi il n'existe pas de variables dont une occurence soit libre et une autre occurence soit liée et si deux occurences d'une même variable liée correspondent à la même occurence de quantificateur.

[4] Ceci n'est pas du bricolage. On considère la clôture universelle de la formule, puisque l'on sait (confère quelques pages plus haut) qu'une formule est satisfiable ssi sa clôture universelle l'est

(4) Nier la formule ... il n'agit juste de rajouter un ¬ à la racine ou de mettre un 1 à côté du premier symbole.

(5) Descendre les négations. Vu et revu. Pour les boulets, se reporter au chapitre précédent mutatis mutandis [5].

(6) Skolémiser la formule ... hahaha ... tremblez ! Mais non ... allez, un petit rappel pour vous rafraîchir la mémoire : skolémiser (ou mettre sous forme de Skolem) consiste à supprimer tous les quantificateurs existentiels et à remplacer les variables qui dépendent de ces quantificateurs par des prédicats[6].
Pour faire simple, on a deux cas de figure :
− la variable en question ne dépendait que de ce quantificateur, autrement dit, il n'y avait aucun ∀ en amont du noeud. Dans ce cas, on remplace purement et simplement la variable qui est une lettre minuscule par une fonction sans paramètre (lettre majuscule). $\exists y$ se transforme en Y.
− si la variable dépend de n quantificateurs, à savoir on est dans un cas de la forme : $\forall x_1 \forall x_2 ... \forall x_n \exists y$, alors, $\exists y$ se transforme en $Y(x_1, ..., x_n)$ et les $\forall x_i$ disparaissent ![7]

(7) Idem la logique des propositions mutatis mutandis[8]

[5] En fait, il faut juste rajouter une règle pour les quantificateurs. ∀ nié donne ∃ et vice-versa. De plus, les quantificateurs transmettent la négation qu'ils ont reçue

[6] Simple non ? Qui a dit "qu'est-ce qu'un prédicat" ? ? ?

[7] Il ne s'agit ni de magie, ni de bricolage. Cette propriété est explicitée dans le *VI*.2

[8] Il faut juste faire attention d'utiliser la règle de résolution de la logique du premier ordre et pas celle du calcul des propositions, à savoir si $\sigma(a_1) = \sigma(a_2)$: $\frac{a_1 \vee A, \neg a_2 \vee B}{\sigma(A) \vee \sigma(B)}(res)$

Quatrième partie
Compléments sur les langages algébriques, analyse syntaxique

VIII Rappels et notations

Définition : Une grammaire est un quadruplet $G = (N, T, \rightarrow, X)$ où
- N désigne le vocabulaire non-terminal
- T désigne le vocabulaire terminal
- $\rightarrow$ désigne la relation de production de la grammaire algébrique
- X qui désigne l'axiome de la grammaire

Remarque : On note : $L(G)$ le langage engendré par la grammaire algébrique
$\rightarrowtail$ la relation de réécriture (application d'une règle de la grammaire à un mot sur le vocabulaire $N \cup T$)
$\rightarrowtail^*$ la relation de dérivation (fermeture réflexive et transitive de la précédente)
$\rightarrowtail^+$ la relation de dérivation stricte
$L(G, \alpha)$ l'ensemble des mots de T^* dérivant de α (en particulier, $L(G) = L(G, X)$)
$Arb(G)$ l'ensemble des arbres syntaxiques de racine X engendrés par G
$Arb(G, \alpha)$ l'ensemble des arbres syntaxiques de racine α engendrés par G
On appelle ϕ la fonction qui donne le mot lu aux feuilles d'un arbre.

IX Réduction des grammaires algébriques

IX.1 Réduites inférieures et supérieures d'une grammaire algébrique

IX.1.1 Algorithme de la réduction supérieure

On définit une suite $(N_i)_{i\geq 0}$ de sous-ensembles de N[9] en posant :
i) $N_0 = \{X\}$
ii) $N_{i+1} = N_i \cup \{A; A \in N$ et $(\exists B \in N_i)(\exists \alpha, \beta \in (N \cup T)^*)(B \rightarrow \alpha A \beta$ dans $G)\}$[10]
Cette suite va devenir stationnaire, donc dès que $N_i = N_{i+1}$, on obtient un ensemble N'. La grammaire réduite de G est la grammaire $G' = (N', T, \mapsto, X)$ où $\mapsto$ est la restriction à N' de $\rightarrow$.

IX.1.2 Algorithme de la réduction inférieure

On définit une suite $(N_i)_{i\geq 0}$ de sous ensemble de N en posant :
i) $N_0 = \emptyset$
ii) $N_{i+1} = N_i \cup \{A; A \in N$ et $(\exists \alpha \in (N_i \cup T)^*$ tel que $A \rightarrow \alpha$ soit une règle de G)$\}$[11]

[9] N désigne l'ensemble des non-terminaux !
[10] N_{i+1} est simplement N_i auquel on ajoute tous les non-terminaux que l'on voit apparaître dans un 2^{nd} membre de règle dont le premier membre est un non-terminal
[11] N_{i+1} est simplement N_i auquel on ajoute tous les non-terminaux dont un second membre de règle est formé uniquement de terminaux ou de non-terminaux de N_i

Cette suite va devenir stationnaire, donc dès que $N_i = N_{i+1}$, on obtient un ensemble N'. La grammaire réduite de G est la grammaire $G' = (N' \cup \{X\}, T, \mapsto, X)$.

Définition : Réduire une grammaire consiste à effectuer dans l'ordre que l'on souhaite les deux réductions (inférieures et supérieures).

Réduction de Chomsky[12] : On dit qu'une grammaire est réduite sous la forme de Chomsky ssi ses règles sont d'une des formes suivantes[13] : $A \to \wedge, A \to a$ où $a \in T, A \to BC$ où $B, C \in T$

Réduction de Greibach : On dit qu'une grammaire est réduite sous forme de Greibach ssi ses règles sont d'une des formes suivantes : $A \to \wedge, A \to a\alpha$ où $a \in T$ et $\alpha \in N^*$

Suppression des branches filiformes[14] : Pour toute grammaire G, il existe une grammaire G' équivalente à G ne contenant pas de règle de la forme $A \to B$ où $A, B \in N$.

X Analyse syntaxique

X.1 A quoi sert l'analyse syntaxique

Le but de l'analyse syntaxique d'un langage algébrique ou plutôt d'une grammaire algébrique est de trouver, a posteriori la façon dont un mot a été engendré par la grammaire considérée et de rejeter les mots qui ne sont pas engendrés par cette grammaire. Le problème de l'analyse syntaxique contient donc celui de la reconnaissance du langage engendré par une grammaire.

X.1.1 Automate à pile

Définition : On appelle automate à pile de vocabulaire d'entrée T et de vocabulaire de sortie T' un quintuplet $A = (S, s_0, z_0, \delta, S')$ tel que :
i) S est l'ensemble fini des états de l'automates
ii) s_0 est un état particulier de S appelé état initial de l'automate
iii) z_0 est une lettre particulière de T' appelée marqueur de fond de pile
iv) S' est un sous-ensemble de S appelé sous-ensemble des états de satisfaction de l'automate.
v) δ est une application de $S \times (T \cup \{\wedge\}) \times T'$ dans l'ensemble des sous-ensembles finis de $D \times T'$. Cette application δ s'appelle la loi de transition de l'automate.

Remarque : Un langage L est algébrique ssi il est formé par l'ensemble des mots reconnus par un automate à pile[15]. De plus, si G est une grammaire algébrique engendrant L, il existe

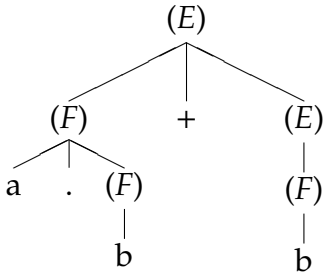
[12] Cette réduction est toute simple. La règle $A \to aABcB$ sera réduite avec les règles :
$A \to A_1A_2, A_1 \to a, A_2 \to AA_3, A_3 \to BA_4, A_4 \to A_5B, A_5 \to c$
[13] Attention, le symbole '$\wedge$' désigne maintenant la lettre vide.
[14] Rien de plus trivial, il suffit d'injecter le second membre de la règle de B dans celui de A
[15] Le fonctionnement d'un automate à pile est décrit dans le livre de Pierre Marchand p.230

un algorithme permettant de construire un automate à pile reconnaissant L à partir de la grammaire G.

X.1.2 Les arbres syntaxiques et leur représentation postfixée

Il y a de nombreuses façon de représenter un arbre. Dans le but de faire de l'analyse syntaxique , la représentation la plus adaptée est la représentation postfixée que nous allons adapter aux grammaires algébriques.

Si $G = (N, T, \rightarrow, X)$ est une grammaire algébrique, on ajoute à chaque règle un numéro propre à cette règle. La connaissance de ce numéro permet de retrouver la règle et c'est ce numéro que l'on fait figurer dans la représentation postfixée des arbres syntaxiques de cette grammaire. Considérons par exemple la grammaire $G = (N, T, \rightarrow, X)$ avec $N = \{E, F\}$, $T = \{a, b, ., +\}$ et $E \rightarrow F + E|1|F|2 \quad F \rightarrow a.F|3|b.F|4|a|5|b|6$.
Le mot $\alpha = a.b63{+}b621$ sera représenté par l'arbre :



X.1.3 Principes de l'analyse syntaxique

Définition : Soit G une grammaire algébrique. Le problème de l'analyse syntaxique pour G est de découvrir un algorithme qui, pour chaque mot α, dit si ce mot est ou non engendré par la grammaire et en cas de réponse positive fournit l'ensemble des arbres syntaxiques engendrés par la grammaire et ayant α comme feuille. Donc, si l'on appelle Ana_G cet algorithme, on a : $Ana_G(\alpha) = (b, res)$ où b est un booléen et res un ensemble d'arbres syntaxiques tels que :
$b = 1 \equiv \alpha \in L(G)$ et si b alors $res = \{t, t \in Arb(G)$ et $\phi(t) = \alpha\}$ sinon $res = \emptyset fsi$

Définition : On appelle méthode d'analyse syntaxique un algorithme qui, pour des grammaires particulières que l'on peut déterminer par algorithme, fournit pour chaque G de ce type un algorithme Ana_G

Rappel : Une grammaire algébrique est non ambiguë ssi pour tout mot, il existe au plus un arbre syntaxique de G associé à ce mot.

Définition : On dit qu'une procédure d'analyse syntaxique pour une grammaire G est
– globale ssi cette procédure manipule des ensembles d'arbres et fournit directement l'ensemble des arbres syntaxiques associés à un mot donné
– locale ssi cette procédure manipule des arbres (mais un seul à la fois) et ne peut fournir l'ensemble des arbres syntaxiques associés à un mot donné que grâce à l'utilisation de l'indeterminisme.

Théorème : Soit G une grammaire algébrique. Si l'on sait construire pour G des procédures d'analyse syntaxique locales et deterministes alors la grammaire est non ambigüe.

Remarque : Nous n'étudierons dans la suite que des procédures d'analyse syntaxique de type local.

Définition : On dit que les procédures d'analyse syntaxique sont de type
– descendant ssi la recherche des arbres syntaxiques associés à un mot α donné se fait en partant de l'axiome X de la grammaire et en essayant les diverses réécritures sur les différents non-terminaux pour atteindre le mot α
– ascendant ssi la recheche des arbres syntaxiques associés à un mot α donné se fait en partant du mot α et en essayant les diverses réductions sur les différents facteurs de ce mot pour atteindre l'axiome X de la grammaire.

Définition : Soit L un langage algrébique. On dit que L possède une ambigüité inhérente ssi tout grammaire algébrique engendrant L est ambigüe.

X.2 Méthodes descendantes d'analyse syntaxique (indeterministes)

X.2.1 Première version indeterministe de l'algorithme d'analyse syntaxique

On considère une grammaire algébrique $G = (N, T, \rightarrow, X)$ quelconque. On va écrire pour chaque non-terminal A une procédure d'analyse syntaxique que l'on appellera Ana-A et qui est chargée de générer un arbre de $Arb(G, A)$ et dont le mot des feuilles est un facteur du mot α à analyser.

Précisons que l'on utilise une variable globale[16] que l'on note *res* et qui est une pile sur le vocabulaire $T \cup \mathbb{N}_R$ [17] où $\mathbb{N}_R \subset \mathbb{N}$ est l'ensemble des numéros de règles de la grammaire G. On notera $\oplus$ l'opération d'adjonction de texte en fin de *res*. Pour simplifier la reconnaissance de la fin du mot α, on suppose que celui-ci est suivi du caractère spécial que nous noterons $\dashv$. Enfin, on dispose d'une procédure 'lire(y :T)' qui lit la lettre en cours dans le mot α et range la lettre obtenue dans la variable y[18]. Décrivons maintenant la procédure Ana-A par des préconditions et des postconditions.

[16]Donc toutes les procédures sont autorisées à la modifier
[17]La seule opération permise sur *res* est d'ajouter du texte à la fin de ce mot
[18]Pour les fanatiques du C, ça correspond à $scanf("\%c", \&y)$;
et pour les fanatique d'Eiffel : *io.read_character*; y := *io.last_character*;
Donc pour savoir si on est en fin de mot, il suffit de faire lire(y) ; si y='$\dashv$' ...

Préconditions de Ana-A :

si *erreur*[19]
alors
la procédure ne fait rien[20]
sinon
la donnée α[21] s'écrit $\alpha = \alpha_1\alpha'$[22] déjà exploité avant l'appel de Ana-A
le résultat *res*[23] est une représentation postfixée d'un arbre associé à la grammaire G et dont le mot des feuilles vaut α_1.

Postconditions de Ana-A :

La donnée α s'écrit $\alpha = \alpha_1\alpha_2\alpha''$ où α_2[24] a été exploité par la procédure Ana-A

La procédure Ana-A a pu positionner le booléen *erreur* à vrai, auquel cas, le processus entier s'écroule en erreur.

Si le booléen *erreur* est resté faux, alors $res \leftarrow res \oplus res_A$ où res_A est la représentation postfixée d'un arbre de $Arb(G, A)$ dont le mot des feuilles est α_2 fsi

Programme principal :
Initialisation : *erreur* ← 'faux' ; *res* ← ∧ ;
Appel : Ana-X ;
Rsultat :

```
si erreur
alors
   ecrire("Le mot n'est pas engendré par la grammaire") ;
sinon
   lire(y)
   si y =⊣
   alors
      ecrire("Le mot est engendré par la grammaire",res) ;
   sinon
      ecrire("Le mot n'est pas engendré par la grammaire") ;
   fsi
fsi
```

Il faut maintenant la procédure Ana($x : T$) qui s'occupe des terminaux et Ana-A qui s'occupe du non-terminal A.

[19] *erreur* est un booléen
[20] Le cas échéant, elle rend le controle à la procédure appelante
[21] α est comme toujours un mot sur le vocabulaire T suivi du marqueur de fin de mot ⊣
[22] α_1 est un facteur gauche de α
[23] *res* est en cours d'élaboration
[24] α'' est le facteur gauche de α'

Procédure Ana-A
si ¬*erreur*
alors
Choisir une règle de premier membre A.
Soit k le numéro de cette règle qui s'écrit $A \rightarrow A_{k_1}A_{k_2}...A_{k_n}$.
Effectuer la suite des appels suivants : Ana-A_{k_1} ;Ana-A_{k_2} ;...Ana-A_{k_n} ;
$res \leftarrow res \oplus k$;
fsi

Procédure Ana(x : T**)**
si ¬*erreur*
alors
lire(y) ;
si y=x
alors
$res \leftarrow res \oplus x$
sinon
erreur ← *vrai*
fsi

Exemple[25] **de procédure Ana-A.** On considère la règle A→ BCa |1| BCd |2| BC |3| Bd |4|
choix
debut Ana-B ; Ana-C ; Ana(a) ; $res \leftarrow res \oplus 1$ *fin*
debut Ana-B ; Ana-C ; Ana(d) ; $res \leftarrow res \oplus 2$ *fin*
debut Ana-B ; Ana-C ; $res \leftarrow res \oplus 3$ *fin*
debut Ana-B ; Ana(d) ; $res \leftarrow res \oplus 4$ *fin*
finchoix

Remarque GlanDyLesque : On peut faire l'analogie avec le backtracking, c'est à dire que l'on teste toutes les possibilités que la grammaire nous offre (toutes les règles) une par une et on s'arrête dès que l'on a trouvé, mais en version bête puisque à chaque fois, on refait des tests que l'on a déjà faits. Si on arrive à la fin sans avoir trouvé, c'est que le mot n'appartient pas au langage engendré par la grammaire.

X.2.2 Seconde version indeterministe avec factorisation des choix

On s'aperçoit en regardant notre exemple que pour chaque choix, on appelle Ana-B, et pour trois d'entre eux, on appelle Ana-C, ce qui est complètement ridicule. Pour 'alléger' le programme, on factorise certaines procédures. Sur notre exemple, ça donne :

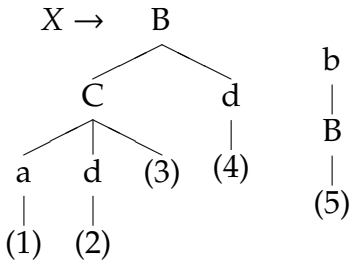
[25] Vous allez tout de suite comprendre la simplicité de l'algorithme avec exemple. Il suffit juste de recopier les règles une par une.

Ana-A
Ana-B ;
choix
debut
Ana-C ;
choix
debut Ana(a) ; $res \leftarrow res \oplus 1$ *fin*
debut Ana(d) ; $res \leftarrow res \oplus 2$ *fin*
debut $res \leftarrow res \oplus 3$ *fin*
finchoix
fin
debut Ana(d) ; $res \leftarrow res \oplus 4$ *fin*
finchoix

Regardons comment, dans le cas général, on peut effectuer cette factorisation des choix dans une grammaire quelconque. Pour cela, on regroupe les règles de la grammaire dans un arbre.

Définition : Soit t un arbre dont les étiquettes sont dans un vocabulaire V. On appelle chemin de cet arbre t les mots sur V obtenus en lisant les étiquettes le long d'un parcours qui va d'une racine à une feuille de l'arbre t.

Comment écrire une grammaire sous forme arborescente.
Il n'y a rien de plus simple. Au lieu de perdre notre temps dans une explication générale, la vue d'un exemple simple vous permettra de comprendre tout le processus. On considère donc une règle A → BCa |1| BCd |2| BC |3| Bd |4| bB |5|. La forme arborescente de cette règle sera :



Mettre une grammaire sous forme arborescente consiste à mettre toutes ses règles sous cette forme.

Condition nécessaire et suffisante pour que les procédures s'arrêtent : Les procédures d'analyse syntaxiques d'une grammaire G s'arrêtent pour toute donnée α ssi la grammaire n'est pas récursive gauche[26].

Définition : Une grammaire est récursive gauche ssi elle contient un non-terminal A tel qu'il existe une dérivation $A \rightarrowtail^{+} A\alpha$

[26]Nous verrons un peu plus loin comment 'dérécursiver' une grammaire

X.2.3 Méthodes descendantes avec réduction de l'indeterminisme

Remarque : Pour des questions de lisibilité, nous emploierons à la place des *si ... alors ... sinon ... fsi* l'écriture suivante :

cas
cas$_1$ → action$_1$;
cas$_2$ → action$_2$;
...
cas$_n$ → action$_n$;
autres cas$_n$ → *erreur* ← vrai ;
sac[27]

Remarque : Dans la suite, on considère une grammaire algébrique $G = (N, T, \rightarrow, X)$ sans $\wedge$. On pose $V = N \cup T \cup \{\dashv\}$. Soient A et B $\in V$.

Définition : B est un initial de A ssi il existe une règle de G de la forme $A \rightarrow B\alpha$

Définition : B est un initial large de A ssi il existe une dérivation de G telle que $A \rightarrowtail^* B\alpha$

Définition : B est un initial stricte de A ssi il existe une dérivation stricte de G telle que $A \rightarrowtail^+ B\alpha$

Définition : B est un final de A ssi il existe une règle de G de la forme $A \rightarrow \alpha B$

Définition : B est un final large de A ssi il existe une dérivation de G telle que $A \rightarrowtail^* \alpha B$

Définition : B est un final stricte de A ssi il existe une dérivation stricte de G telle que $A \rightarrowtail^+ \alpha B$

Remarque : L'ensemble des initiaux larges et celui des finaux larges d'un élément de $T \cup \{\dashv\}$ sont réduits à cet élément.

Définition : On dit que B est un suivant de A ssi il existe une règle de la forme $C \rightarrow \alpha AB\beta$

Définition : On dit que B est un voisin de A ssi une des conditions suivantes est vérifiée :
i) (cas général) $\exists C, D \in V$ tels que A soit final large de C, D suivant de C et B initial large de D.
ii) (cas particulier) B est le marqueur de fin de mot $\dashv$ et A est final large de l'axiome X.

[27] sac est l'anagramme de cas. Il correspond à *fincas*

Remarque : Pour trouver les voisins de A, Pierre Marchand propose la petite comptine suivante :
1) De quoi A est-il final large ? On obtient une liste l_1 qui contient A.
2) Quels sont les suivants de chaque élément de l_1 ? On obtient une seconde liste l_2 éventuellement vide.
3) Quels sont les initiaux larges des éléments de l_2 ? On obtient une troisième liste l_3 qui est l'ensemble des voisins de A.
4) On rajoute à l_3 le marqueur de fin si A est final large de X.

Pour être sûr d'avoir bien compris toutes ces notions, nous allons étudier un exemple. On considère la grammaire $G = (N, T, \rightarrow, X)$ où $N = \{E, T, F, P\}$, $T = \{a, b, c, +, -, \times, /, (,)\}$ avec :
$E \rightarrow$ T+E |1| T − E |2| T |3|
$T \rightarrow$ F × T |4| F/T |5| F |6|
$F \rightarrow$ P |7| − P |8|
$P \rightarrow$ a |9| b |10| c |11| (E) |12|

On obtient donc le tableau d'analyse suivant :

	I	I*	I+	F	F*	F+	suivants (S)	voisins (V_s)
E	T	ETFP-abc(	TFP-abc(	ET	ETFPabc)	ETFPabc)	)	⊣)
T	F	TFP - abc(	FP-abc(	TF	TFP abc)	TFPabc)	+ -	⊣+-)
F	P	FP-abc(	P-abc(	P	FPabc)	Pabc)	× /	⊣ ×/+-)
P	abc(	Pabc(	abc(	abc)	Pabc)	abc)		⊣ ×/+-)
a		a			a			⊣ ×/+-)
b		b			b			⊣ ×/+-)
c		c			c			⊣ ×/+-)
+		+			+		E	ETFP-abc(
-		-			-		E	ETFP-abc(
×		×			×		T	TFP-abc(
/		/			/		T	TFP-abc(
(		(			(		E	ETFP-abc(
)		)			)			⊣ ×/+-

On rappelle que le but du présent chapitre est de réduire l'indeterminisme. Pour cela, on décide de lire un caractère en avance. Donc le programme principal ainsi que la procédure Ana($x \in T$) ont été modifiés en conséquence.

Condition de déterminisme 1 : Dans le cas où aucun des choix n'est une action correspondant à la fin de l'exploitation d'une règle de la grammaire, la condition de déterminisme est :

$$\forall i, j, i \neq j \Rightarrow I^*(A_i) \cap I^*(A_j) \cap T = \emptyset$$[28]

Les instructions dans Ana-A de la forme *debut* Ana-A_1 ; *res* ← *res* ⊕ i *fin* seront donc remplacées par :

cas
y ∈ $I^*(A_1) \cap T \rightarrow$ Ana-A_1[29] ;*res* ← *res* ⊕ i ;

[28] C'est à dire les ensembles d'initiaux terminaux des différentes lettres concernées par le choix à rendre déterministe sont deux à deux disjoints

[29] Ce qui veut dire que l'on n'effectuera Ana-A_1 et les actions qui vont avec seulement si le caractère lu en avance est un initial de A_1

Condition de déterminisme 2 : Dans le cas d'un choix qui correspond à la fin d'exploitation d'une règle de la grammaire, la condition de déterminisme est :

$$I^*(A_i) \cap V_s(A) \cap T = \emptyset$$[30]

Les instructions dans Ana-A de la forme *debut res* $\leftarrow$ *res* $\oplus$ i *fin*[31] seront remplacées par :

cas
... [32]
y $\in V_s(A) \cap T \rightarrow$ *res* $\leftarrow$ *res*$\oplus$ i ;
... [33]
sac

Définition : Une grammaire algébrique G, réduite et non récursive gauche, pour laquelle les procédures d'analyse syntaxique deviennent toutes déterministes en utilisant la méthode décrite ci-dessus est dite LALL(1). Une grammaire est dite LL(1) ssi on a le même résultat sans qu'il soit nécessaire de factoriser des choix.

Proposition : Aucune grammaire algébrique LALL(1) n'est ambigüe.

X.2.4 Cas des grammaire avec $\wedge$

not_yet_implemented −− Patience ... vous verrez ça en *Traduction* l'année prochaine ...

X.2.5 Dérécursivation à gauche d'une grammaire algébrique

On a vu que l'analyse descendante ne pouvait pas se faire si la grammaire en question est récursive gauche. C'est pourquoi il est toujours nécessaire lors d'une analyse descendante de vérifier que la grammaire n'est pas récursive gauche et le cas échéant de la dérécursiver. Pour cela, plusieurs possibilités s'offrent à nous.

Premier cas de figure.[34]
On suppose que la récursivité gauche est localisée sur des règles d'un non-terminal. Autrement dit, on est dans la situation suivante : $A \rightarrow A\alpha_1|...|A\alpha_n|\beta_1|...|\beta_n$
On remarque donc la récursivité à gauche des règles :

- $A \rightarrow A\alpha_1$
- ...
- $A \rightarrow A\alpha_n$

Eh bien il suffit de créer un nouveau non-terminal A' et de transformer les règles de A de la façon suivante. Les nouvelles règles de A sont constituées de toutes les anciennes règles de A qui ne concernent pas la récursivité $\beta_1|...|\beta_n$
et toutes ses règles concaténées à A' : $\beta_1 A'|...|\beta_n A'$
Les règles de A' ne seront que les anciennes règles de A qui étaient récursives : $A\alpha_1|...|A\alpha_n|$

[30] On remarque que ce sont les voisins de A et pas ceux de A_i qui interviennent !
[31] Ce cas correspond aux règles de la forme $A \rightarrow \beta$ |i|. Effectuer l'action *res* $\leftarrow$ *res* $\oplus$ i signifie donc que l'on va terminer la procédure Ana-A et rendre le contrôle à la procédure qui l'a appelée
[32] Instructions de la forme : y $\in I^*(A_k) \cap T \rightarrow$ Ana-A_k ; *res* $\leftarrow$ *res* $\oplus$ k *fin*
[33] Instruction : autre cas $\rightarrow$ *erreur* $\leftarrow$ vrai
[34] Le plus courant et certainement le seul que l'on utilisera dans la pratique ... du moins je l'espère :)

auxquelles on enlève le A puis auxquelles on concatène A' à droite de chaque règle : $\alpha_1|...|\alpha_n|\alpha_1 A'|...|\alpha_n A'|$
Ce qui donne au final : $A \to \beta_1|...|\beta_n|\beta_1 A'|...|\beta_n A'$ $A' \to \alpha_1|...|\alpha_n|\alpha_1 A'|...|\alpha_n A'|$

Second cas de figure. La récursivité à gauche de la grammaire concerne simultanément plusieurs règles.
not_yet_implemented

X.2.6 Les langages LL(1) et LL(k)

Définition : Soit L un langage sur un vocabulaire T. On dit que L est LL(1) ssi il existe une grammaire algébrique $G = (N, T, \to, X)$ engendrant L et qui est LL(1). De même, on dit que L est LL(k) ssi il existe une grammaire algébrique engendrant L et qui est LL(k)

Théorème :
– Toute grammaire / langage LL(k) est LL(k+1)
– Pour tout k, il existe des grammaires / langages qui sont LL(k) dans être LL(k+1)

X.3 Méthodes ascendantes d'analyse syntaxique

On rappelle que les procédures ascendantes d'analyse syntaxique partent du texte à analyser, effectuent les réductions sur celui-ci et essaient d'atteindre l'axiome de la grammaire.[35]

X.3.1 Version indéterministe de l'algorithme ascendant d'analyse syntaxique

Pour comprendre le principe de l'analyse syntaxique ascendante, on va mettre en place un algorithme indeterministe.

Programme principal
Initialisation :
$z \leftarrow \wedge$; *res* $\leftarrow \wedge$; *erreur* $\leftarrow$ *faux* ; *succes* $\leftarrow$ *faux* ;
Itération :
tant que ¬*erreur* et ¬*succes* faire
choix
e : *debut erreur* $\leftarrow$ *vrai fin* ;
s : *debut succes* $\leftarrow$ *vrai fin* ;
lect : *debut* lire(y) ; z $\leftarrow$ zy ; *res* $\oplus$ y *fin* ;
red : *debut*
choix
choisir un mot β ;
écrire z = $z_1\beta$;
choisir A tel que $A \rightarrow \beta$ est la règle i de G ;
z $\leftarrow$ z_1A ;
res $\leftarrow$ *res* $\oplus$ i ;
finchoix
fin
finchoix
fintantque
Resultat
lire(y) ;
cas
erreur $\rightarrow$ ecrire(Le mot n'est pas engendré par la grammaire) ;
¬*succes* $\rightarrow$ ecrire(Le mot n'est pas engendré par la grammaire) ;
$y \neq \dashv \rightarrow$ ecrire(Le mot n'est pas engendré par la grammaire) ;
$z \neq X \rightarrow$ ecrire(Le mot n'est pas engendré par la grammaire) ;
¬*erreur* et *succes* et $y = \dashv$ et $z = X \rightarrow$ ecrire(Le mot est engendré par la grammaire,res) ;
fincas

Remarque : La variable *res* est la représentation postfixée d'un arbre associé à la grammaire G dont la racine est z. En début de traitement, ces deux variables sont initialisées à $\wedge$

Théorème : Soit $G = (N, T, \rightarrow, X)$ une grammaire algébrique réduite. Pour qu'un algorithme d'analyse syntaxique de type ascendant s'arrête pour tout $\alpha \in T^*$, il faut et il suffit qu'aucun non-terminal ne dérive strictement de lui-même. Plus formellement, si

[35] Pour une description du principe de ces méthodes, voir p.274 du livre de P.Marchand

Ana_G est un algorithme d'analyse syntaxique ascendant, alors : $\forall \alpha \in T^*, Ana_G(\alpha)$ s'arrête $\Leftrightarrow \neg(\exists A \in N, A \rightarrowtail^+ A)$

Remarque : Dans la suite du chapitre, on considerera des grammaires réduites

X.3.2 Principe des méthodes utilisant des automates

Les automates utilisés en analyse syntaxique sont de la forme $A = (S, s_0, .)$[36] et fonctionnent sur le vocabulaire $N \cup T$

La première donnée d'un analyseur de type ascendant est une table d'automate qu'on appelera table "trans-item" et qui est de la forme suivante :

Noms des états $\downarrow$ | Lettres de $N \cup T \rightarrow$	X	...	A
s_0	$s_0.X$	...	$s_0.X$
...			
s_i	$s_i.X$	...	$s_i.X$
...			
s_n	$s_n.X$	...	$s_n.X$

La plupart du temps, le nom d'un état est un entier. Par ailleurs, il apparaît très souvent un état spécial noté " − " qui est un état de refus et fait passer *erreur* à vrai.

On construit ensuite une seconde table nommée "table-action" qui, en fonction d'un état et d'une lettre terminale ou du marqueur de fin de mot $\dashv$ donne une action d'analyse à effectuer.

Remarque : La racine z n'est pas seulement un mot sur $N \cup T$. On insère systématiquement dans z les états de l'automate qui correspondent aux résultats des calculs intermédiaires sur z. Le nouveau z s'écrit donc $z = s_0 A_1 s_1 ... s_i A_{i+1} s_{i+1} ... s_{k-1} A_k s_k$ avec $s_i = s_0.A_1...A_i$. On utilise la fonction θ pour effacer les états dans un mot telle que $\theta_{|N \cup T} = id$ et $\forall s \in S, \theta(s) = \wedge$.[37]

[36] On ne distingue pas les états finaux (de satisfaction)

[37] Que de formalités pour dire que $\theta(z)$ est le mot z sans les numéros des états

Programme principal
Initialisation
z←s_0 ; *res* ← ∧, *erreur* ← *faux* ; *succes* ← *faux* ; lire(y) ;
Iteration
tantque ¬*erreur* et ¬*succes faire*
s← sommet(z) (1)
cas (2)
Action(s,y) = succès → *succes* ← *vrai*
Action(s,y) = Lect. → *z* ← *zys.y* ; *res* ← *res* ⊕ y ; lire(y) ;
Action(s,y) = A → α |i| → z ← réduction(z,α,A) ; *res* ← *res* ⊕ i ;
Autres cas → *erreur* ← *vrai*
fincas
fintantque
Resultat
si erreur alors écrire(Le mot n'est pas engendré par la grammaire)
sinon écrire(Le mot est engendré par la grammaire,*res*)
fsi

Remarque : La variable *res* est la représentation postfixée d'un arbre associé à la grammaire G dont la racine est $\theta(z)$. En début de traitement, ces deux variables sont initialisées à ∧. On lit un caractère en avance. Le point (1) va chercher sur la pile z le dernier état s. Cette procédure ne dépile pas la pile. Le point (2) cherche dans la table action le traitement qui dépend de s et de y qui doit être effectué à ce stade de la procédure.

X.3.3 Méthodes de réduction de l'indeterminisme

Dans tout ce paragraphe, on considère que les grammaires possèdent la propriété suivante :
L'axiome X n'apparaît que dans une seule règle de la forme $X \to Y$
La raison de cette contrainte est de faciliter la gestion du booléen *succes*.
Si la grammaire ne vérifie pas cette contrainte, il suffit de rajouter une règle "artificielle" $X' \to X$[38].

X.3.4 La méthode SLR(1)

Cette méthode est la plus simple à mettre en oeuvre. Elle utilise la notion d'item dit LR(0). Pour chaque règle $A \to \alpha$ |i|, on construit tous les objets de la forme $A \to \alpha_1.\alpha_2$ avec $\alpha = \alpha_1\alpha_2$. Le nombre d'items associés à une règle est dont $|\alpha| + 1$[39]. Il faut faire attention au fait que si la règle est $A \to \wedge$, il ne lui est associé qu'un seul item qui s'écrit $A \to .$
On va donc construire un automate $A = (S, s_0, .)$ dont les états sont des ensembles d'items. Intuitivement, si s est un état de l'automate et si $It = A \to \alpha_1.\alpha_2$ est un item appartenant à s, cela signifie que l'analyseur cherchera à faire la réduction $A \to \alpha_1\alpha_2$ et que dans ce but, il a déjà lu un texte qui s'est réduit en α_1. Donc un facteur droit z' de z vérifie $\theta(z') = \alpha_1$. Il faudra donc effectuer des lectures dans le mot à analyser pour obtenir un mot se réduisant α_2 avant de pouvoir effectuer la réduction $A \to \alpha_1\alpha_2$. Un état est formé de tous les items qui

[38]Cette opération banale qui donne évidemment une grammaire équivalente s'appelle "augmenter la grammaire"
[39]Je rappelle que $|\alpha|$ signifie : "longueur du mot α"

sont compatibles avec ce qui précède[40].

Définition : Soit I un ensemble d'items. On dit que l'ensemble de I est fermé ssi pour tout item *it* de I de la forme $A \to \alpha_1.B\alpha_2$ et toute règle de la forme $B \to \beta$ alors l'item $B \to .\beta$ est dans l'ensemble I.

Parlons peu, parlons bien. La construction de l'automate se fait maintenant de manière algorithmique en suivant le procédé ci-dessous :

i) Construction de l'état initial : On considère l'item $X \to .Y$ associé à la seule règle où intervient l'axiome X. Conformément à l'interprétation intuitive des items, cet item signifie que l'on va chercher à faire la réduction $X \to Y$, mais que, pour l'instant, l'analyseur n'a encore rin lu dans le texte à analyser. L'état initial s_0 est alors la fermeture de l'ensemble d'item $\{X \to .Y\}$

ii) Progression sur une ligne de l'automate : Ayant construit un état s, il faut construire les états s.x où x est une lettre quelconque de $N \cup T$. Pour cela, on considère tous les items de s où le point est devant la lettre x et qui sont donc de la forme $A \to \alpha.x\beta$. On crée un ensemble intermédiaire à l'aide des items de la forme $A \to \alpha x.\beta$ et l'état s.x est par construction la fermeture de cet ensemble intermédiaire. Plus formellement, on considère les deux fonctions suivantes :

Remarque : On notera IT l'ensemble des ensembles d'items.[41]

Fonction ens-intermédiaire(I : IT ; x : $N \cup T$) : IT
var I_1 : IT ; it : item
debut
Initialisation : $I_1 \leftarrow \emptyset$;
Iteration :
    Pour it dans I *faire*
        si $\exists A, \alpha, \beta$, it = $A \to \alpha.x\beta$ alors $I_1 \leftarrow I_1 \cup \{A \to \alpha x.\beta\}$ *fsi*
    fait
Resultat : ens-interm $\leftarrow I_1$
fin

Fonction tran-item(I : IT ; x : $N \cup T$) : IT
trans-item ← Fermeture(ens-intermédiaire(I ;x)) ;

Exemple : Soit $G = (\{E, T, F, P\}, \{a, b, c, (,), +, 6, \times, /\}, \to, E)$ une grammaire telle que :
$E \to E + T$ |1| $E - T$ |2| T |3|
$T \to T \times F$ |4| T/F |5| F |6|
$F \to P$ |7| $-P$ |8|
$P \to a$ |9| b |10| c |11| (E) |12|
Cette grammaire ne vérifie pas la condition imposée par l'axiome. Donc qu'est ce que l'on fait ? On ajoute la règle $X \to E$.

[40] Il est fort possible que vous n'ayez pas compris toute cette phrase. P.Marchand suggère que vous testiez le petit programme suivant : *tantque* ¬ compris *faire* relire la phrase *fait* ... huhuhu

[41] $I \in IT$ signifie : I est un ensemble d'items

i) L'état initial s_0 est donc le suivant :

$$s_0 = \left| \begin{array}{ccccc} X \to .E & E \to .E+T & E \to .E-T & E \to .T & T \to .T \times F \\ T \to .T/F & T \to .F & F \to .P & F \to .-P & P \to .a \\ P \to .b & P \to .c & P \to .(E) & & \end{array} \right|$$

Cet état initial contient donc 13 items.

ii) On peut donc construire les états suivants :
$s_0.E = s_1$ $s_0.T = s_2$ $s_0.F = s_3$ $s_0.P = s_4$ $s_0.- = s_5$
$s_0.a = s_6$ $s_0.b = s_7$ $s_0.c = s_8$ $s_0.(= s_9$[42]
On obtient donc par exemple :

$$s_1 = s_0.E = \left| \begin{array}{c} X \to E. \\ E \to E.+T \\ E \to E.-T \end{array} \right|$$

En résumé : La table de l'automate se construit ligne par ligne, en commençant par la ligne de s_0. Pour la ligne s, on calcule tous les état de la forme s.x pour remplir la ligne de s. Pour chaque nouvel état, on crée une nouvelle ligne. La construction s'arrête lorsque toutes les lignes correspondant aux états ont été remplies.

La construction étant terminée, il faut maintenant construire la table action. Il s'agit d'une procédure de décision permettant de savoir quelle action doit être effectuée en fonction de l'état de l'automate et de la lettre lue en avance qui est dans le vocabulaire $T \cup \{\dashv\}$. Pour cela, on réexamine chacun des états de l'automate précédemment créé et on construit une table Action[s : S, x : $T \cup \{\dashv\}$]. On distingue plusieurs cas :

i) L'état de l'automate est vide[43]. La table action fait sortir en *erreur*. L'analyseur s'arrête en écrivant que le mot n'est pas reconnu. Donc, $\forall x \in T \cup \{\dashv\}, Action[-; x] \to erreur \leftarrow vrai$

ii) L'état de l'automate est $\{X \to Y.\}$. L'analyseur a donc lu une donnée qui peut se réduire en X. Pour que le mot fourni à l'analyseur soit engendré par la grammaire, il faut et il suffit que le caractère lu en avance soit le marqueur de fin de mot $\dashv$. Dans ce cas on effectue la dernière réduction $X \to Y$ et on sort en succès de l'analyseur.
Donc : Action[$\{X \to Y.\}$; $\dashv$] $\to$ *succes* $\leftarrow$ *vrai*
et $\forall x \in T$, Action[$\{X \to .\}$; x] $\to$ *erreur* $\leftarrow$ *vrai*

iii) Le point n'est en fin d'item dans aucun des items de l'état en cours. Un tel état est dit *etat de lecture*. Si la lettre lue en avance est compatible avec les lectures prévues dans l'état (c'est à dire si $y = a$ et si au moins un des items est de la forme $A \to \alpha.a\beta$), on effectue cette lecture. Sinon, on sort en *erreur*. On indique cela *lect* dans la table action.

iv) L'état en cours ne contient qu'un item de la forme $\{A \to \alpha.\}$. On dit que cet état est un *etat de reduction*. On effectue cette réduction ssi la lettre lue en avance est un voisin de A ![44]
Dans ce cas, on effectue la réduction $A \to \alpha$. On codifie cela en écrivant la règle numérotée $A \to \alpha$ |i| dans la table Action. Dans les autres cas, l'analyseur sort en *erreur*.

[42] On aurait également pu considérer 's_0.+' mais l'ensemble d'items obtenu est vide. On indiquera cela dans la table de l'automate par un caractère spécial, par exemple '−'

[43] Généralement codifié par '−' dans la table

[44] A retenir ...

v) Ce dernier cas est le plus important. Il y a dans ce cas 3 possibilités.

– L'état contient un item avec le point en fin du type $A \to \alpha.$ [45] et d'autres items avec le point ailleurs qu'à la fin[46]. On examine alors les lettres que l'on peut lire dans cet état et les voisins de A. Si les deux ensembles obtenus sont disjoints, alors le conflit est levé. Pour les "x" qui sont les voisins de A, on remplit la table Action avec la règle numérotée $A \to \alpha$ |i|. Pour les x qui sont des lectures possibles, on remplit la table Action avec *lect*. On continue alors sur les autres états.[47]

– L'état contient seulement des items avec le point en fin du type $A \to \alpha.$ [48]. On examine alors les voisins de A pour chacun des items de la forme $A \to \alpha.$ de l'état en cours d'examen. Si les ensembles obtenus sont deux à deux disjoints, alors le conflit est levé. Pour les x qui sont voisins de A, on remplit la table Action avec la règle numérotée $A \to \alpha$ |i|. On continue sur les autres états.[49]

– L'état contient au moins deux items avec le point en fin : $A \to \alpha.$ [50] et d'autres items avec le point ailleurs qu'à la fin[51]. On examine alors les lettres que l'on peut lire dans cet état et les voisins de chacun des A tels que $A \to \alpha.$ est un item de l'état en cours d'examen. Si tous les ensembles obtenus sont disjoints, alors les conflits sont levés. On continue sur les autres états[52].

X.3.5 La méthode LR(1)

Si on examine les raisons des échecs de la méthode cu-dessus, on remarque que l'on calcule trop tard les lettres qui sont des voisins de non-terminaux et qui permettent de savoir si une réduction est possible ou non à cet endroit. La méthode LR(1) consiste en l'utilisation d'items contenant plus d'informations. Ces nouveaux items dits LR(1) sont de la forme ($A \to \alpha_1.\alpha_2$ |a) où $A \to \alpha_1\alpha_2$ est une règle de la grammaire à analyser et "a" est une lettre de $T \cup \{\dashv\}$. On va construire un automate $A = (S, s_0, .)$ dont les états sont des ensembles d'items. Avant d'allez plus loin, assurez-vous de bien avoir compris les points suivants :
Intuitivement, si s est un état de l'automate et si $It = (A \to \alpha_1.\alpha_2|a)$ est un item appartenant à s, cela signifie que :

i) L'analyseur cherchera à faire la réduction $A \to \alpha_1\alpha_2$ et pour cela, il a déjà lu un texte qui s'est réduit en α_1. Donc un facteur à droite z' de z vérifie $\theta(z') = \alpha_1$. Il faudra donc effectuer

[45] Possibilité de réduction $A \to \alpha$

[46] Possibilité de lecture. On dit que l'état contient, de ce fait, un potentiel conflit de *lecture – reduction*

[47] Dans le cas contraire, à savoir lorsqu'une lettre est à la fois voisine de A et compatible avec une lecture, la méthode est en échec en raison d'un conflit lecture-réduction. On arrête tout le precessus soit pour changer de grammaire, sois pour changer de méthode d'analyse.

[48] Possibilité de réduction $A \to \alpha$. On dit que l'état contient un potentiel conflit *reduction – reduction*

[49] Dans le cas contraire, c'est à dire quand il existe deux items $A \to \alpha.$ et $B \to \beta.$ et qu'une même lettre est à la fois voisin de A et de B, la méthode est mise en échec en raison d'un conflit *reduction – reduction*. On arrête tout le processus soit pour changer la grammaire, sois plutôt pour changer de méthode.

[50] Possibilité de réduction $A \to \alpha$

[51] Possibilité de lecture. On dit que l'état contient des potentiels conflits *reduction–reduction* et *lecture–reduction*

[52] Dans le cas contraire, la méthode est mise en échec en raison soit d'un conflit *reduction – reduction* soit d'un conflit *lecture – reduction*. On arrête tout le processus soit pour changer de grammaire, soit plutôt pour changer de méthode.

des lectures dans le mot à analyser pour obtenir un mot se réduisant α_2 avant de pouvoir effectuer la réduction $A \to \alpha_1\alpha_2$. On reconnaît la signification intuitive d'un item LR(0).

ii) Quand on effectuera la réduction $A \to \alpha_1\alpha_2$, la lettre lue en avance devra être "a". C'est l'information supplémentaire contenue dans ces items.

On va donc maintenant remettre en place ce qui a été fait précédemment pour la méthode SLR(1), à savoir trouver la construction de l'automate et de la table action.

Définition : Soit α un mot non vide sur le vocabulaire $N \cup T \cup \{\dashv\}$. On appelle ensemble des premières lettres des mots dérivant de α dans F et on note $Prem(\alpha)$ le sous-ensemble de T : $Prem(\alpha) = \{b; b \in T \text{ et } \exists\gamma, \alpha \rightarrowtail^* b\gamma\}$

Définition : Soit I un ensemble d'items LR(1), on dit que I est fermé ssi il vérifie la condition suivante : si $It = (A \to \alpha_1.B\alpha_2|a)$ est un item appartenant à I et si $B \to \beta$ est une règle de G alors tous les items de la forme $(B \to .\beta|b)$ avec b qui est une lettre quelconque de $Prem(\alpha_2 a)$ sont aussi dans I

Remarque : L'item $(B \to .\beta|b)$ doit accompagner l'item $(A \to \alpha_1.B\alpha_2|a)$ dans tout état de l'automate où ce dernier figure.

Fonction fermeture : Pour chacun des items non encore examinés, on regarde si le point des devant un non-terminal. Si oui, on ajoute à l'ensemble d'items en construction les items de la forme $(B \to .\beta|b)$ avec b dans $Prem(\alpha)$ qui sont considérés comme non encore examinés et produiront peut-être de nouveaux items.

De même, la construction de l'automate associé est une généralisation simple du cas SLR(1).

i) Calcul de l'état initial. On considère la seule règle du type $X \to Y$ et l'item LR(1) $(X \to .Y| \dashv)$ qui signifie que l'on souhaite effectuer la réduction $X \to Y$, que rien n'a encore été lu dans ce but et que lorsque cette réduction sera effectuée, le caractère lu en avance sera le marqueur de fin de mot $\dashv$[53]. L'état initial de l'automate LR(1) est la fermeture de cet unique item, c'est à dire : $s_0 = Fermeture(\{(X \to .Y| \dashv)\})$

ii) Si un état s a été généré par l'algorithme, on calcule pour chaque x de $N \cup T$ les états s.x en effectuant la fonction ens-intermédiaire(s ;x) puis la fermeture sur le résultat obtenu. On obtient ainsi la fonction trans-item[54].

iii) On continue cette construction tant que de nouveaux états sont générés par le processus.

Remarque : Pour alléger les notations, on simplifie
$A \to \alpha.\beta|a_1 \quad ... \quad A \to \alpha.\beta|a_n \quad$ par $\quad A \to \alpha.\beta|a_1, ..., a_n$

[53] Le mot aura été complètement lu

[54] Il n'y a pratiquement pas de modification par rapport au cas SLR(1).

Enfin, la méthode se termine par la mise en place de la **table Action**. Elle est beaucoup plus simple que dans le cas SLR(1)[55] :
Action(s ;x) = "*lect*" ssi s contient un item de la forme $(A \to \alpha.x\beta|a)$.
Action(s ;x) = "*reduction* $A \to \alpha$" ssi s contient un item de la forme $(A \to \alpha.|x)$.
Si ces conditions ne se contredisent pas, la méthode LR(1) a réussi sinon elle est en échec.

X.3.6 La méthode LALR(1)

Remarque : Les deux méthodes précédentes ont des défauts. La méthode SLR(1) est simple mais manque de puissance. La méthode LR(1) est puissance mais de grande complexité. Nous allons donc étudier un compromis entre ces deux méthodes. Pour simplifier, nous considererons la méthode qui construit l'analyseur LR(1) et à la dégrader pour obtenir l'analyseur LALR(1). L'idée de cette méthode est fondée sur la remarque que les ensembles d'items LR(1) utilisés dans l'automate LR(1) contiennent déjà l'information de l'automate SLR(1). On introduit la fonction κ appelée "corps" telle que $\kappa(A \to \alpha.\beta|a) = A \to \alpha.\beta$[56]. Si on considère $A = (S, s_0, .)$ et $A' = (Q, q_0, .)$ respectivement les automates LR(1) et SLR(1) d'une grammaire G, alors : $\kappa(s_0) = q_0$ et $\forall s, x \in S \times N \cup T, \kappa(s.x) = \kappa(s).x$

Faisons fonctionner l'analyseur ascendant en utilisant l'automate $A' = (Q, q_0, .)$ de la méthode SLR(1) mais en utilisant la table Action suivante :

Action(q ;x) = '*lect*' $\Leftrightarrow$ il existe un item de la forme $(A \to \alpha.x\beta|a)$ dans un des états s tel que $\kappa(s) = q$
Action(q ;x) = '*reduction* $(A \to \alpha')$ $\Leftrightarrow$ il existe un item de la forme $(A \to \alpha.|x)$ dans un des états s tel que $\kappa(s) = q$

Si cette table Action est sans conflit, la méthode LALR(1) réussit sur cette grammaire. On remarquera que l'on peut concevoir cette table action en écrivant des état LALR(1) obtenus en fusionnant les listes de contextes de réduction des états de l'automate LR(1) qui ont le même corps. Autrement dit, si $s_1, ..., s_n$ sont des états de l'automate LR(1) de G tels que $\kappa(s_1) = ... = \kappa(s_n) = q$, on forme un état LALR(1) noté $s_{1,...,n}$ et dont les items sont définis par : $(A \to \alpha.\beta|l_1) \in s_1$ et ... et $(A \to \alpha.\beta|l_n) \in s_n \Leftrightarrow (A \to \alpha.\beta|l_1 \cup ... \cup l_n) \in s_{1,...,n}$

Avec cette nouvelle notation de ces états, la table Action se définit exactement comme dans la méthode LR(1)[57].

[55] Il n'y a plus rien à calculer car toutes l'information nécessaire est dans les items ! Chouette :)

[56] Encore une fois, ce n'est qu'un formalisme pour dire très peu de chose. κ prend juste la règle sans les contexte... rien de plus. On dit que κ transforme les items LR(1) en items LR(0)

[57] On remarque que cette méthode LALR(1) peut introduire des conflits *reduction – reduction* qui n'existaient pas dans la version LR(1).

Cinquième partie

Remarques d'ordre général sur ce poly

Ce poly ne vous aura servi à rien[58] si :

- Vous avez tout recopié
- Vous n'avez pas noté et ne pouvez pas restituer les définitions de
  "clause",
  "règle de résolution" en logique des propositions et en LP1,
  "skolémisation", ...
- Vous ne savez pas montrer que
  $\alpha = \neg(\neg a \vee (b \Rightarrow c)) \vee ((a \vee \neg b) \vee (\neg a \vee c))$ est une tautologie
- Vous ne connaissez pas la CNS pour que les procédures d'analyse syntaxique descendante se terminent
- Vous ne savez pas réduire une grammaire
- Vous ne savez pas dérécursiver à gauche une grammaire
- Vous ne savez pas mettre une grammaire sous forme arborescente
- Vous n'avez pas noté la structure de tous les programmes d'analyse et ne pouvez pas les écrire de vous même.
- Vous n'avez pas recopié et appris la petite comptine pour rechercher les voisins
- Vous n'avez pas refait le tableau d'analyse de la page 28.

Donc s'il subsiste ne serait-ce qu'un seul point noir parmi cette liste, c'est qu'il vous faut relire plus attentivement ce poly.

GlanDyL vous souhaite
bonne chance pour le partiel

[58] L'auteur dit ça ... mais il ne dit rien ... ;)