IV. Grammaires BP (*BP grammars*)[81]

Nous désignons par "**grammaires BP**" une classe de grammaires formelles appartenant à la famille des systèmes de réécriture, permettant une représentation compréhensible de formes syntaxiques utilisées dans les "langages de percussions": contextes, répétitions, répétitions avec image homomorphique. Ces grammaires sont présentées ici comme une extension de la notion de **motif** (*pattern*) proposée par Angluin[82].

Pour ces grammaires, un algorithme de **test d'appartenance** (*membership test*) déterministe a été implémenté dans la première version du *Bol Processor* (BP1). Le test ne peut être appliqué que si les règles sont partiellement ordonnées. Cet ordre correspond à une heuristique de reconnaissance de formes ("la priorité aux plus grands agrégats de symboles") et peut être géré par le moteur d'inférences. En mode production les grammaires BP peuvent donc être considérées comme une forme de connaissance **déclarative**, alors qu'en analyse c'est l'aspect **procédural** qui est mis en évidence. L'algorithme déterministe de test d'appartenance permet ainsi, en suivant l'analyse pas à pas, de fournir une explication de l'échec et donc, le cas échéant, d'envisager une généralisation de la grammaire.

1. Aperçu historique (*historical survey*)

La conception "atomiste" de la musique — "la mélodie n'est autre chose qu'une succession de sons déterminés" (Vincent d'Indy) — est à l'origine du sérialisme de Schoenberg[83], suivi par Webern, puis par les tenants de l'approche stochastique:[84] Elisabeth Shannon[85], Meyer[86], Coons & Kraehenbuehl[87], Fuchs[88], Moles[89], Xenakis[90], Philippot[91], etc.

Une autre manière d'appréhender un idiome musical consiste à supposer l'existence de structures profondes, autrement dit un ensemble de relations syntagmatiques et

---

81 Les quatre premiers paragraphes de ce chapitre ont fait l'objet d'une communication au *Workshop on AI & Music, 11th International Joint Conference for Artificial Intelligence* (IJCAI). Voir Bel 1989a. Les paragraphes suivants ont été exposés au 6ème Congrès de l'AFCET — *Reconnaissance des Formes et Intelligence Artificielle* (Bel 1987b). Une version anglaise simplifiée (Bel 1991b) sera publiée.

82 1980b

83 Voir Chemillier 1990b pour un commentaire sur l'aspect formel de la théorie sérielle.

84 Références données par Vecchione.

85 1951

86 1956; 1957

87 1958

88 1961

89 1958

90 1963

91 1970

paradigmatiques qui permettent de caractériser les productions d'un idiome musical lorsque celles-ci sont représentées par des structures discrètes.

L'idée de formaliser une “syntaxe musicale” est très ancienne. On peut citer — parmi les modèles qui n'utilisent pas le formalisme des règles de réécriture — le Jeu de dés musical (*Musikalisches Würfelspiel*) de W.A. Mozart, puis, au 20ème siècle, les travaux de Schenker[92] qui servent encore de substrat à de nombreuses approches théoriques. La **théorie générative de la musique tonale**[93] est une synthèse de travaux en linguistique formelle, en psychologie et en musicologie: partant d'une monodie tonale, le modèle permet de déterminer à la fois une structure de surface (à l'aide de règles de structure métrique et de règles de regroupement, ces dernières étant empruntées à la psychologie gestaltiste) et une structure profonde (réduction prolongationnelle Schenkerienne et structure prosodique inspirée de la théorie des rythmes de Cooper et Meyer[94]). Le terme “génératif”, dans ce contexte, se réfère donc à l'**analyse** d'une pièce unique, et non à la production musicale d'un ensemble de variantes:

> Our theory of music is therefore base on structural considerations; it reflects the importance of structure by concerning itself not with the composition of pieces but with assigning structures to already existing pieces.[95]

Pour une œuvre donnée il existe plusieurs analyses possibles. Dans un deuxième temps on fait donc appel à des règles d'un ordre supérieur (règles de préférence[96]), tenant compte des consonances/dissonances, cadences tonales, etc., pour déterminer la *meilleure* analyse.[97]

La théorie de Lerdahl et Jackendoff, que les auteurs ont plus tard commencé à étendre à la musique atonale, reste une des tentatives les plus achevées de modélisation de la musique tonale.[98] Elle est à la source de nombreux travaux actuels, certains visant à évaluer expérimentalement sa validité psychologique, d'autres à concevoir des systèmes experts pour la composition[99] ou l'analyse[100]. Une critique et une extension de la théorie générative ont été formulées par Célestin Deliège[101].

La recherche de grammaires formelles servant à modéliser une **activité compositionnelle** (ou improvisationnelle) a suscité quelques travaux dans le domaine de l'ethnomusicologie.[102] Feld s'est élevé toutefois contre ce qu'il appelle “la coquille vide du formalisme” (*the hollow shell of formalism*) dans les emprunts de la musicologie à la linguistique générative. A propos des travaux de Lidov, il écrit notamment:

---

92 1935

93 Jackendoff & Lerdahl 1982, pp.92-ff; Lerdahl & Jackendoff 1983.

94 1960

95 Jackendoff & Lerdahl 1982, p.85

96 *op.cit.* pp.99-102.

97 Reprenant ainsi l'hypothèse Chomskienne d'un “sujet idéal”.

98 C'est aussi à ce travail sur la musique que l'on doit l'introduction des règles de préférence en linguistique formelle.

99 Par exemple, Camilleri 1984.

100 Notamment pour simuler la perception du rythme: voir Jones et al. 1990.

101 1983; 1984.

102 Laloum & Rouget 1965; Lidov 1972; etc., cités par Feld 1974.

> […] I find the approach absurdly formal and totally pretentious, not because I can't appreciate his mathematics, but because he purports to do a theoretical task and then says nothing about what kind of ethnomusicological theory he is talking about. Yet worse, the music is considered as nothing more than one dimensional transcriptions — a set of abstractions which a mathematician may retranslate into other abstractions of another logical order. This method of analysis is based on the completely false assumption that transcriptions of music have some sort of objective reality [,…][103]

Dans le même article, Feld cite Blacking pour ce qu'on pourrait appeler un fondement épistémologique de l'ethnomusicologie scientifique:

> Analytic tools cannot be borrowed freely and used as short cuts to greater achievements in ethnomusicological research as can electronic devices such as the tape recorder: they must emerge from the nature of the subject studied.[104]

Un problème essentiel des théories de la performance ou de la compétence est celui de leur capacité explicative (*explanatory adequacy*). Selon Laske[105], une théorie de la performance peut être:

— adéquate pour les observations (*observationally adequate*) si elle donne un compte-rendu compréhensible des données;

— descriptivement adéquate (*descriptively adequate*) si la sortie est conçue en accord avec une condition de bonne formation (*a well-formedness condition*), le cas échéant, avec une mesure de la grammaticalité;

— explicatoire (*explanatory adequate*) si et seulement si elle est capable de définir la structure interne de la performance en relation avec les règles de compétence dont cette performance dépend — au moins partiellement.

Une grammaire formelle qui, pour un corpus (fini) de productions donné, permet de décider si une production arbitraire appartient ou non au corpus, n'a pas de capacité explicatoire. Pour accéder au niveau de l'adéquation explicatoire il faut en fait disposer d'un modèle susceptible d'engendrer de nouvelles productions correctes ou de prédire l'appartenance d'une production au langage. Un tel modèle doit donc se prêter à la généralisation, opération que l'analyste peut tenter d'effectuer lui-même si la représentation est assez explicite.[106]

2. Grammaires de motifs (*pattern grammars*)

2.1 Définitions et notations (*definitions and notations*)

Soient Vt un alphabet terminal fini de **symboles terminaux**, et Vn un alphabet dénombrable de symboles non-terminaux (des **variables**). Si l'on note H l'ensemble des homomorphismes non-effaçants pour la concaténation (λ-*free homomorphisms*) de (Vn ∪ Vt)* sur lui-même, tout élément de H dont la restriction à Vt est l'application

---

[103] Feld 1974, pp.209-10.

[104] Blacking 1972, p.1. Cité par Feld 1974.

[105] 1972a, p.5. Laske reprend ici une classification des théories de la compétence proposée par Chomsky (1964).

[106] Une autre approche consiste à implémenter une technique d'inférence inductive. (Voir chapitre VI)

identique est appelé une **substitution**. Si l'ensemble image d'une substitution est Vt*, cette substitution est dite **terminale**. Toute substitution dont la restriction à Vn est une bijection de Vn dans Vn est un **réétiquetage de variables** (*renaming of variables*).

On appelle **motif** (*pattern*) tout élément de (Vn ∪ Vt)*. Si *p* est un motif et *s* une substitution, alors s(p) est appelé une **dérivation** de *p*. Un motif qui ne contient aucune variable est appelé **dérivation terminale** ou **phrase**. Deux motifs, *p* et *q*, sont **équivalents** (on note p ≈ q) si et seulement s'il existe un réétiquetage de variables *r* tel que p = r(q). Une autre relation binaire (notée p ≤ q) est définie de la manière suivante: *p* est **moins général que** (ou **plus spécifique que**) *q* si et seulement si pour une substitution *s*, p = s(q). Puisque la substitution est un homomorphisme non effaçant, p ≤ q => |p| ≥ |q|.

Le **langage** généré par le motif *p* est l'ensemble des dérivations terminales de *p*, à savoir L(p) = {s ∈ Vt^+ : s ≤ p}. On peut prouver que:

≤ est transitive
p ≤ q => L(q) ⊇ L(p)
p ≈ q <=> p ≤ q et q ≤ p

Toutefois, la question de trouver une procédure effective pour décider si L(q) ⊇ L(p), étant donnés deux motifs arbitraires *p* et *q*, semble être ouverte.[107]

2.2 Application à un exemple musical (*musical example*)

Les ensembles de variations du *qa'ida* présenté au chapitre III §2 peuvent être, en première analyse, décrits à l'aide des motifs *p1* et *p2* suivants, avec les variables X, Y et Z:

Variations simples:

p1 = X tagetirakitadhin--dhagenadhatigegenakateeneteenakena
Y tagetirakitadhin--dhagenadhatigegenakadheenedheenagena

Variations doubles:

p2 = dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena
tagetirakitadhin--dhagenadhatigegenakateeneteenakena
Z dhatigegenakateeneteenakena
tagetirakitadhin--dhagenadhatigegenakateeneteenakena
tin--takenata--takenatatikekenakateeneteenakena
taketirakitatin--takenatatikekenakateeneteenakena
Z dhatigegenakateeneteenakena
tagetirakitadhin--dhagenadhatigegenakadheenedheenagena

Soit L le langage représentant toutes les variations acceptables du *qa'ida*, L1 le sous-ensemble des variations simples et L2 celui des variations doubles. On a évidemment: L = L1 ∪ L2. L(p1) et L(p2) sont les langages générés respectivement par *p1* et *p2*. Admettons (sur la base d'une étude expérimentale) que L(p1) et L(p2) contiennent des sous-ensembles non triviaux de L1 et L2.

On cherche, de manière idéale, des motifs *descriptifs* de L1 et L2. Un motif *d* est **descriptif** d'un langage L si L(d) ⊇ L et si pour tout motif *q* tel que L(q) ⊇ L, L(q) n'est pas strictement contenu dans L(d).

L est en fait un ensemble fini. Par exemple, dans ce *qa'ida*, toutes les chaînes acceptables de Vt* ont des longueurs inférieures ou égales à 32 x 6 = 192 symboles, ce

[107] Angluin 1980b, pp.49-52.

qui permet de calculer un majorant de card(L).[108] On ne connaît toutefois de L qu'un ensemble S^+ d'**exemples** (*positive instances*) et un ensemble S^- de **contrexemples** (*negative instances*). On appelle présentation l'ensemble $S = S^+ \cup S^-$. On dit qu'un motif *p* est **compatible avec** (*matches*) une présentation si et seulement si *p* est descriptif de S^+ et que $S^- \cap L(p) = \varnothing$. Le motif *p* est **ajusté sur** S (*tight fit*) si $L(p) = S^+$. De manière évidente, tout motif ajusté sur S est descriptif de S^+ et compatible avec S.

Il existe une procédure effective pour inférer un motif descriptif à partir d'un ensemble d'exemples S^+, mais le problème est NP-difficile dans la plupart des cas.[109] De plus, la classe des langages de motifs n'est pas fermée pour la plupart des opérations ensemblistes de base: union, complément et intersection. En sorte qu'il n'est pas facile de construire de manière systématique une description par motifs d'un langage formel.

2.3 Langages de motifs restreints (*restricted pattern languages*, RPL)

Etant donné un motif *p*, toute sous-classe de L(p) peut être définie en contraignant les dérivations acceptables de p. Par exemple, on peut définir la sous-classe de longueur *n*: $L_n(p) = \{s \in Vt^+ : s \leq p \text{ et } |s| = n\}$. Dans l'exemple traité, évidemment |X| = |Y| = 24 et |Z| = 12. De plus on peut écrire Y = mir(X).

Une autre manière de formuler des contraintes consiste à utiliser des **règles de réécriture**. Nous notons

$$p \longrightarrow q$$

pour indiquer que toute dérivation acceptable de *q* est une dérivation acceptable de *p*, autrement dit $q \leq p$. Si l'on note S l'ensemble de toutes les substitutions, $p \longrightarrow q$ dénote le sous-ensemble

$$S_{p\to q} \quad \text{tel que} \quad \forall s \in S_{p\to q},\ s(p) = q.$$

Puisque le problème de décider si $p \leq q$ pour deux motifs arbitraires p et q est NP-complet,[110] on doit se contenter de **règles bien formées**:

Définition:

> La règle $p \longrightarrow q$ est bien formée si et si seulement si $p \in Vn^+$, $q \in (Vt \cup Vn)^+$, $|q| \geq |p|$, et aucune variable n'apparaît plus d'une fois dans p.

Théorème:

> Si la règle $p \longrightarrow q$ est bien formée alors $L(p) \supseteq L(q)$.

Preuve:

Il suffit de trouver une substitution *s* telle que q = s(p), ce qui entraîne $q \leq p$. Une procédure pour construire *s* est la suivante:

---

[108] Le nombre de phrases de longueur inférieure ou égale à *n* sur un alphabet de cardinal k > 1 est en effet:

$$\frac{k \cdot (k^n - 1)}{k - 1}$$

[109] Angluin 1980b, pp.54-55.

[110] *Op.cit.* p.52

(1) remplacer les |p|-1 variables les plus à gauche (leftmost) de *p* par les |p|-1 variables les plus à gauche de *q*;

(2) remplacer la variable la plus à droite dans *p* par la chaîne formée des |q|-|p|+1 symboles les plus à droite (rightmost) de *q*.

Puisqu'aucune variable ne figure deux fois dans *p*, il n'existe aucune contrainte sur *q* qui rende impossibles ces remplacements.[111] Par exemple, une substitution de XYZ produisant abcX serait {X/a, Y/b, Z/cX}.■

Soit L(p) un langage de motif et R un ensemble de règles bien formées. Toute règle (p —> q) définit un ensemble de substitutions $S_{p->q}$. Pour construire une phrase du **langage de motif restreint** $L_R(p)$ on procède comme suit:

(a) sélectionner un sous-ensemble R_o de R ;

(b) soit $S_o = \cap_i (S_{pi->qi})$ tel que $(p_i \text{ —> } q_i) \in R_o$;

(c) si $S_o \neq \varnothing$ et S_o est fini, alors $S_o = \{s\}$ tel que la phrase est w = s(p).

La sélection de R_o appelle certains commentaires:

(1) Si une variable X n'apparaît dans la partie gauche d'aucune règle sélectionnée, alors elle peut être remplacée par n'importe quel $q \in (Vt \cup Vn)^+$. Dans ce cas, S_o est infini et par conséquent $L_R(p) = \varnothing$.

(2) Il existe des sous-ensembles de R qui donnent $S_o = \varnothing$, autrement dit des **dérivations avortées**. Par exemple, si une variable X apparaît dans le membre gauche de deux règles distinctes de R_o, par exemple X —> q_i et X —> q_j, alors $S_{X->qi} \cap S_{X->qj} = \varnothing$, et donc $S_o = \varnothing$.

(3) Dans tous les cas autres que (1) et (2), toute variable apparaît dans un motif avec une dérivation unique, de sorte que $\text{card}(S_o) = 1$.

L'exemple suivant est destiné à clarifier les points (2) et (3). Un RPL pour les variations doubles serait $L_R(p2)$ avec:

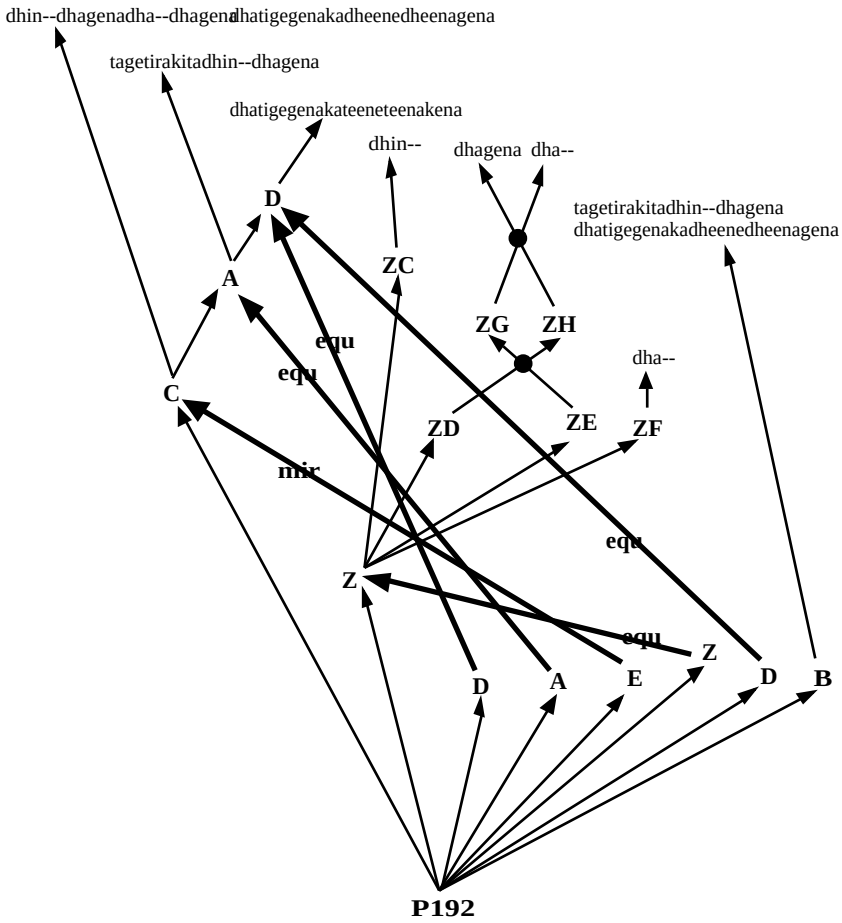
p2 = P192 et R est l'ensemble de règles:
(1) P192 —> C Z D A E Z D B avec E = mir(C)
(2) A —> tagetirakitadhin--dhagena D
(3) B —> tagetirakitadhin--dhagenadhatigegenakadheenedheenagena
(4) C —> dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena A
(5) D —> dhatigegenakateeneteenakena
(6) Z —> tirakitatirakitatirakita (7) Z —> ZA ZB (8) Z —> ZC ZD ZE ZF
(9) ZC ZD —> ZG ZH (10) ZD ZE —> ZG ZH (11) ZE ZF —> ZG ZH
(12) ZG ZH —> dhagenadhin-- (13) ZG ZH —> dhagenadha--
(14) ZC —> dhin-- (15) ZD —> dhin-- (16) ZE —> dhin-- (17) ZF —> dhin--
(18) ZC —> dha-- (19) ZD —> dha-- (20) ZE —> dha-- (21) ZF —> dha--
(22) ZA —> dheenedheenedheene (23) ZB —> dheenedheenedheene
(24) ZA —> dha-dha-dha- (25) ZB —> dha-dha-dha-

Si l'on choisit R_o ={1,2,3,4,5,8,10,13,14,21}, on obtient la phrase:

[111] Cette procédure est inspirée de l'algorithme de Makanin (Makanin 1977, Roussel 1987).

dhin--dhagena dha--dhagena dhatigegenaka dheenedheenagena **C**
tagetirakita dhin--dhagena dhatigegenaka teeneteenakena
dhin-- dhagena dha--dha-- dhatigegenaka teeneteenakena **Z D**
tagetirakita dhin--dhagena dhatigegenaka teeneteenakena **A**
tin--takena ta--takena tatikekenaka teeneteenakena **E**
taketirakita tin--takena tatikekenaka teeneteenakena
dhin--dhagena dha--dha-- dhatigegenaka teeneteenakena **Z D**
tagetirakita dhin--dhagena dhatigegenaka dheenedheenagena **B**

qui peut être représentée par le graphe syntaxique (sensible au contexte) suivant, où ‘equ’ et ‘mir’ indiquent les structures de répétition stricte et de répétition avec miroir:



Les ensembles de substitutions des règles 10 et 14 peuvent être représentés par:

$$S_{\text{p10->q10}} = \{..., \text{ZD ZE/ZG ZH}, ...\}$$

et

$$S_{\text{p14->q14}} = \{..., \text{ZC/}dhin\text{-- },...\}$$

où ‘...’ indique l'ensemble (dénombrable) de toutes les substitutions possibles des chaînes de $(Vt \cup Vn)^+$ à l'exception de celles qui ont déjà été spécifiées (ZD ZE et ZC). On voit que

$$S_{\text{p10->q10}} \cap S_{\text{p14->q14}} = \{..., \text{ZD ZE/ZG ZH}, ... , \text{ZC/dhin-- },...\}.$$

Le résultat des intersections est:

S_o = {P192/C Z D A E Z D B, A/tagetirakitadhin--dhagena D,
B/tagetirakitadhin--dhagenadhatigegenakadheenedheenagena,
C/dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena A, D/dhatigegenakateeneteenakena,
Z/ZC ZD ZE ZF, ZC/dhin--, ZD ZE/ZG ZH, ZF/dha--, ZG ZH/dhagenadha--}

où chaque dérivation est explicite. Par conséquent, S_o contient une substitution unique qui produit la phrase considérée.

Si l'on sélectionne R_o = {1,2,3,4,5,8,10,13,14,15,21} on obtient $S_{p10->q10}$ = {..., ZD ZE/ZG ZH, ...}, $S_{p15->q15}$ = {..., ZD/*dhin*-- ,...}, et $S_{p20->q20}$ = {..., ZE/*dha*-- ,...}. Appelons $S_{15,20} = S_{p15->q15} \cap S_{p20->q20}$ = {..., ZD/*dhin*-- ,..., ZE/*dha*-- ,...}. Pour toute substitution s $\in S_{15,20}$, s(ZD ZE) = s(ZD) s(ZE) = *dhin--dha--*. Par conséquent, ZD ZE ne peut jamais être substitué à ZG ZH, ce qui contredit $S_{p10->q10}$. Dans ce cas, $S_o = \varnothing$ et la dérivation est avortée.

Remarque: on pourrait aussi dire de S_o qu'elle est l'expression conjonctive maximale (donc la plus spécifique) utilisant les prédicats ($p_i = q_i$) tels que (p_i —> q_i) $\in R$.

Théorème:

> La classe des langages finis coincide exactement avec celle des RPLs.

Preuve:

Tout sous-ensemble de l'ensemble (fini) de règles dans un RPL produit au plus une dérivation terminale. L'ensemble des dérivations terminales est donc fini. Réciproquement, pour tout langage fini L il existe une grammaire linéaire à droite sans règle récursive (*a non-imbedding right-linear grammar*) qui génère exactement L. Soit G = (Vt, Vn, S, R) où S est le symbole de départ et R un ensemble fini de règles de format A —> a ou A —> aB tel que A,B $\in$ Vn, A≠B, et a $\in$ Vt. Il est facile de montrer que G engendre exactement le RPL L_R(p) tel que p = S, ce qui complète la preuve.■

Corollaire:

> La classe des RPL est récursive et fermée pour l'union, la concaténation et l'intersection.

On construit une procédure effective pour construire $L_R(p) = L_{R1}(p1) \cup L_{R2}(p2)$ de la manière suivante:

Supposons que L_{R1}(p1) et L_{R2}(p2) sont définis avec:
p1 = S1 et $R1$ = {S1 —> q1, ...}
p2 = S2 et $R2$ = {S2 —> q2, ...} dans lequel on a renommé les variables de $R2$ de sorte qu'aucune variable n'apparaisse à la fois dans $R1$ et dans $R2$. On construit L_R(p) défini avec:
p = S et R = {S —> S1, S —> S2} $\cup$ $R1$ $\cup$ $R2$.
L'ensemble L_R(p) des dérivations terminales de S est l'union des ensembles de dérivations terminales de S1 et S2, par conséquent $L_R(p) = L_{R1}(p1) \cup L_{R2}(p2)$. De plus, puisque $R1$ et $R2$ ne contiennent que des règles bien formées, R ne contient aussi que des règles bien formées. Par conséquent, L_R(p) est un RPL.

Les RPLs forment une classe de langages formels fermée sur l'union, et il existe une procédure permettant de construire un langage à partir de ses sous-ensembles. La classe

est par ailleurs récursive et par conséquent le test d'appartenance est algorithmique. Ces propriétés sont intéressantes pour construire un modèle de représentation qui permette d'opérer des généralisations descriptives.

3. Grammaires BP (*BP grammars*)

3.1 Règles de motifs (*pattern rules*)

Les règles de réécriture du §2.3 peuvent être remplacées par des **règles de motifs**:

P96 —> X A Y B *avec* mirror(X,Y) et longueur(X,24)
P192 —> C Z1 D A E Z2 D B *avec* equal(Z2,Z1) et longueur(Z1,12) et mirror(C,E)

Ces règles comportent une partie de réécriture classique et des prédicats *mirror*, *longueur* et *equal* qui expriment des contraintes.

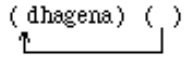
Il est clair que si deux variables liées par une prédicat *equal/mirror* apparaissent dans la partie droite d'une règle, la variable la plus à gauche dénote une chaîne qui sert de référence, et l'autre une copie de cette référence. Par exemple, avec les règles suivantes:

A —> B C where equal(B,C)
B —> dhagena

la dérivation terminale *dhagenadhagena* peut être notée

(= dhagena) (: dhagena)

où la première parenthèse (marquée avec "=") sert de référence (**maître**), et la suivante (marquée avec ":") de copie (**esclave**). Dans un implémentation informatique, la seconde parenthèse contient en fait un pointeur vers la référence:



La même convention d'écriture est utilisée dans les règles, par exemple:

A —> (= B) (: B)

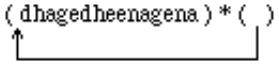
Pour ce qui est des homomorphismes, on utilise un symbole réservé pour indiquer que le contenu de la parenthèse suivante (maître ou esclave) est une image homomorphique. Par exemple, le miroir des langages de percussion est noté "*". Etant donné l'homomorphisme miroir défini au chapitre III §2, dans la grammaire suivante

S —> (= D) * (: D)
D —> dhagedheenagena

l'unique dérivation terminale est

(= dhagedheenagena) * (: taketeenakena)

et sa représentation interne:



A l'aide de ces conventions il est possible de simplifier le formalisme de représentation des RPLs:

(1) Si une variable apparaît plusieurs fois dans une règle de RPL, elle doit être indiquée avec des marqueurs de répétition. Par exemple,
A —> B C B C B doit s'écrire: A —> (=B) (=C) (:B) (:C) (:B).

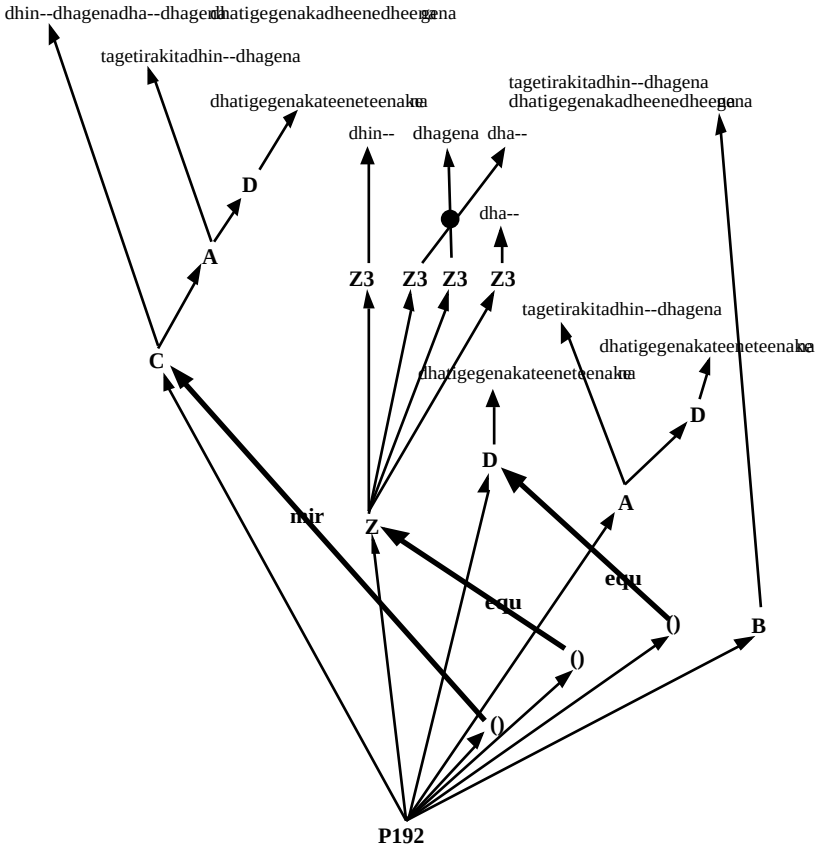
(2) La même variable peut apparaître deux fois dans un membre gauche de règle (si ces deux occurences ne sont pas liées par un marqueur de répétition). Ainsi, la règle A A —> q est correcte, mais pas la règle (=A) (:A) —> q.

(3) Les miroirs sont représentés par un astérisque.

Avec ces conventions, la grammaire BP générant le langage RPL défini au §2.3 est:

(1) P192 —> (=C) (=Z) (=D) A * (:C) (:Z) (:D) B
(2) A —> tagetirakitadhin--dhagena D
(3) B —> tagetirakitadhin--dhagenadhatigegenakadheenedheenagena
(4) C —> dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena A
(5) D —> dhatigegenakateeneteenakena
(6) Z —> tirakitatirakitatirakita
(7) Z —> Z6 Z6
(8) Z —> Z3 Z3 Z3 Z3
(12) Z3 Z3 —> dhagenadhin--
(13) Z3 Z3 —> dhagenadha--
(14) Z3 —> dhin--
(18) Z3 —> dha--
(22) Z6 —> dheenedheenedheene
(24) Z6 —> dha-dha-dha-

dans laquelle les règles 9, 10, 11, 15, 16, 17, 20, 21 et 23 ont été supprimées. Le graphe syntaxique de la phrase générée au §2.3 est maintenant:

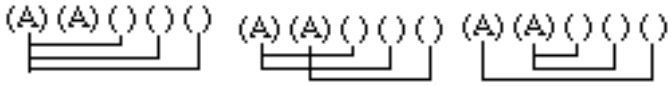


3.2 Affectation maître-esclave (*master-slave assignment*)

La représentation de motifs définie au §3.1 peut être ambiguë. Par exemple, l'expression

(= A) (= A) (: A) (:A) (:A)

peut représenter diverses affectations de pointeurs:



etc…

Seule la première affectation est prise en compte dans la version BP2 du *Bol Processor*. Dans la version BP1, toutes les affectations sont réalisables au moyen de l'éditeur. Dans les deux cas, l'ambiguïté est levée même si l'affectation réalisée n'est pas explicitement visible à l'écran.

L'affectation d'une parenthèse est caractérisée par un pointeur (un entier positif ou nul) *i*, que nous appelons **marqueur de parenthèse**, calculé de la manière suivante:

1) Parenthèse maître: i = 0 (“**marqueur zéro**”)
2) Parenthèse esclave: (i-1) est le nombre de marqueurs zéro qui séparent la parenthèse esclave de la parenthèse maître à laquelle elle est rattachée.

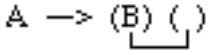
Les valeurs de *i* pour les parenthèses esclaves des trois affectations précédentes sont les suivantes:

(=A) (=A) (2) (2) (2) (=A) (=A) (2) (2) (1) (=A) (=A) (1) (1) (2) .

Nous montrons maintenant comment ces affectations sont recalculées lors des réécritures. Supposons qu'à l'expression précédente soit appliquée la règle:

A —> (= B) (:B)

à la deuxième position de dérivation, i.e. sur la deuxième occurrence de A. La règle s'écrit



ou encore:

A —> (=B) (1)

Après réécriture on obtient:

(=A) (=(=B) (:B)) (:A) (:A) (:A)

pour la première affectation, ou

(=A) (=(=B) (:B)) (:A) (:A) (:(=B) (:B))

pour la seconde, ou enfin

(=A) (=(=B) (:B)) (:(=B) (:B)) (:(=B) (:B)) (:A)

pour la troisième.

Ces formules se représentent plus simplement avec les marqueurs, respectivement:

(=A) (=(=B) (1)) (**3**) (**3**) (**3**) (=A) (=(=B) (1)) (**3**) (**3**) (**2**) (=A) (=(=B) (1)) (**2**) (**2**) (**3**)

Il est facile de voir que les marqueurs indiqués en gras ont été incrémentés d'une unité après le remplacement de A par (=B) (1).

L'algorithme de réaffectation des marqueurs dans la chaîne de travail (*workstring*) X(i) après une réécriture est le suivant:

```
inmark = n2 - n1; /* n1 et n2 représentent les nombres de marqueurs
   zéro dans les membres gauche et droit de la règle, respectivement.
   */
i0 = position du symbole le plus à droite réécrit par la règle;
q = 0;
Pour (i = i0; i < longueur de l'expression)
|    Si (X[i] est un marqueur zéro)
|    alors
|    |    q = q +1;
|    sinon
|    |    Si(X[i] est un marqueur esclave et que X[i] ≥ q)
|    |    alors
|    |    |    X[i] = X[i] + inmark;
|    |    Finsi
|    Finsi
|    i = i + 1;
Finpour
```

Il est facile de voir que seuls les marqueurs pointant vers des parenthèses maîtres à gauche de la partie réécrite sont incrémentés de *inmark*.

En conclusion, la réécriture dans une grammaire BP se fait en deux étapes:

1) Réécriture des symboles et des marqueurs;

2) Correction des marqueurs avec l'algorithme précédent.

Le formalisme que nous avons introduit pour la représentation de motifs ne modifie donc pas la procédure de réécriture (étape 1): les marqueurs sont copiés comme des symboles quelconques de l'alphabet Vn. Dans ce qui suit, on peut donc traiter toute grammaire BP comme une grammaire formelle de type 0, sachant que l'étape 2 doit être effectuée après chaque réécriture.

4. Grammaires BP transformationnelles (*BP transformational grammars*)

Ce terme est utilisé ici dans un sens plus restrictif qu'en linguistique. L'idée, empruntée à Kain[112], consiste à considérer toute dérivation comme une séquence de dérivations dans plusieurs grammaires; les symboles de départ A_D d'une grammaire, sauf “S”, le symbole initial, sont des symboles terminaux d'une grammaire de niveau supérieur.

Définition

> Une **grammaire transformationnelle** (sans règle d'effacement) G est un quintuplet ordonné (Ve,Vn,Vt,Vd,F) tel que:

[112] 1981, p.24

$V = Ve \cup Vn \cup Vt \cup Vd$, $Vt \neq \emptyset$, $Vd \neq \emptyset$

(Ve,Vn,Vt,Vd) est une partition d'un alphabet fini V,

et F est un ensemble fini de couples ordonnés (LCR, LDR) tel que:

$C \in (Vn \cup Vt \cup Vd)^+$, $L \in V^*$, $R \in V^*$ et $D \in (Vn \cup Vt)^+$

Vd est l'alphabet de départ, Vt l'alphabet terminal, Vn l'alphabet intermédiaire (les variables), Ve l'alphabet extérieur à G, et F l'ensemble des règles de production.

L'alphabet de départ d'une grammaire est nécessairement inclus dans la réunion des alphabets terminaux des grammaires précédentes, sauf pour la première où Vd = {S}. Si le langage est décrit par *n* grammaires, l'alphabet terminal de G_n est inclus dans l'alphabet terminal du langage. Nous convenons d'autre part que l'alphabet extérieur de G_1 est vide et que tout symbole de l'alphabet extérieur d'une grammaire G_i est un terminal d'une grammaire G_j avec $j < i$. Les symboles terminaux qui ne représentent pas des *bol*s sont appelés **symboles structurels**. Nous verrons plus loin (chapitre V §5) comment ces symboles sont gérés par les *gabarits*. Les symboles structurels reconnus par le *Bol Processor* BP1 sont:

les parenthèses et pointeurs de répétition: **(=) (:)**

le symbole de miroir: *****

les entiers (1..99) qui représentent la vitesse courante

des **symboles spéciaux** choisis parmi: {**+ : ; =** }

5. Contrôle des dérivations dans les grammaires BP (*derivation control in a BP grammar*)

Diverses stratégies de contrôle ont été implémentées dans les versions BP1 et BP2 du *Bol Processor*. Une justification détaillée de ces stratégies a fait l'objet d'une publication.[113] On présente en premier les dérivations en mode “production”. Le mode “reconnaissance” est étudié au §6.

5.1 Grammaires ‘RND’ (‘*RND’ grammars*)

La production de phrases avec un contrôle stochastique de l'ordre des règles et de la position de dérivation utilise la procédure suivante:

```
Procédure GENERER
Début
  Tant que (une règle est appliquable)
  |    Choisir une règle candidate;
  |    Choisir une position de dérivation;
  |    Si (le membre gauche de la règle est reconnu)
  |    alors
  |    |    Remplacer l'occurrence trouvée par le membre droit;
  |    Finsi
  Fintant
Fin
```

[113] Kippen & Bel 1990.

Le choix d'une règle se fait par un tirage pondéré par les poids des règles candidates (voir chapitre V §6). La position de dérivation est aléatoire. Le moteur d'inférence du BP1 peut aussi générer dans l'ordre toutes les phrases du langage.

Ce mode de contrôle est indiqué en plaçant l'instruction 'RND' (*random*) au dessus des règles de la grammaire transformationnelle.

5.2 Contrôle de position de dérivation: règles 'LEFT' et 'RIGHT' (*controlling the derivation position: 'LEFT' and 'RIGHT' rules*)

Il est souvent nécessaire, et dans tous les cas plus rapide, de prendre l'occurrence la plus à gauche (*leftmost*) ou la plus à droite (*righmost*) de l'argument gauche de la règle sélectionnée dans la chaîne de travail. Ce contrôle peut s'effectuer au niveau de chaque règle, en plaçant l'instruction correspondante, 'LEFT' ou 'RIGHT' en début de règle.

5.3 Grammaires 'LIN' (*'LIN' grammars*)

En production, le contrôle des dérivations d'une grammaire 'LIN' est identique à celui d'une grammaire'RND' dont toutes les règles seraient de type 'LEFT'.

5.4 Grammaires 'ORD' (*'ORD' grammars*)

Dans une grammaire 'ORD' les règles sont choisies dans l'ordre où elles apparaissent dans la grammaire. Chaque règle est appliquée jusqu'à saturation, puis la grammaire est analysée de nouveau pour trouver la première règle candidate. La position de dérivation est choisie comme avec l'instruction 'LEFT'.

Les grammaires 'ORD' sont utilisées surtout dans le cas où aucun choix stochastique, ni sur la règle candidate ni sur la position de dérivation, n'est nécessaire.

6. Test d'appartenance des grammaires BP (*membership test for a BP grammar*)

Un exemple d'analyse syntaxique par une grammaire BP (formalisme étendu) est proposé en annexe.

6.1 Algorithme (*algorithm*)

Le principe général de l'algorithme de test d'appartenance est le suivant:

1) Etant donnée une grammaire transformationnelle G, permuter les membres gauche et droit de toutes les règles de G. Soit G' la grammaire ainsi obtenue (**grammaire duale**).
2) Sachant que le langage est généré par l'application des grammaires transformationnelles G_1, ... G_n dans cet ordre, le test d'appartenance est le résultat de la **dérivation canonique à droite** de la chaîne analysée par G'_n, ... G'_1 dans cet ordre.
3) La chaîne appartient au langage si et seulement si le test d'appartenance se termine avec le symbole de phrase S.

La permutation des arguments dans les règles justifie l'utilisation de flèches doubles dans les grammaires: <—>

L'intérêt de cet algorithme est multiple. En premier lieu, il permet d'utiliser sensiblement les mêmes procédures qu'en synthèse pour réaliser le test d'appartenance. De plus, il est déterministe, économique en espace mémoire, et le nombre de réécritures est inférieur à la longueur de la chaîne analysée.

La seule difficulté est de définir une dérivation canonique pour des grammaires contextuelles. Par *dérivation à droite* il faut entendre une dérivation qui réécrit le symbole non terminal situé le plus à droite dans la chaîne. Cette définition est ambiguë: le contexte fait-il partie des symboles réécrits? Si l'on répond 'oui' à cette question, il n'est pas possible de donner une solution générale au problème de l'ambiguïté de la grammaire contextuelle. Par contre, une définition plus générale des dérivations canoniques[114] permet de définir des contraintes sur les règles de production qui garantissent que la grammaire ne sera pas ambiguë.

6.2 Dérivation contextuelle à droite (*context-sensitive rightmost derivation*)

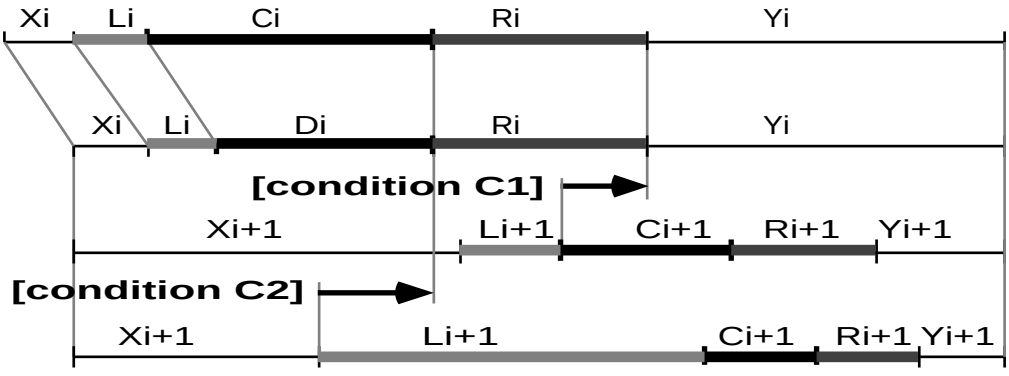
Soit G une grammaire contextuelle. La dérivation de G:

$W_0 => W_1 => ... => W_n$

est **contextuelle à droite** (*context-sensitive rightmost*) si et seulement si

$\forall i \in [0, n-1]$, $W_i = X_iL_iC_iR_iY_i$
et $W_{i+1} = X_iL_iD_iR_iY_i$ en appliquant la règle f_i: $L_iC_iR_i \rightarrow L_iD_iR_i$,
et l'une au moins des conditions suivantes est réalisée:
(C1) $|C_{i+1}R_{i+1}Y_{i+1}| > |Y_i|$
(C2) $|L_{i+1}C_{i+1}R_{i+1}Y_{i+1}| > |R_iY_i|$

Cette définition est adaptée de celle de Hart[115]. Hart n'étudie que le cas des grammaires **strictement contextuelles** pour lesquelles $|C_i| = |C_{i+1}| = 1$, et la condition C1 peut alors s'écrire $|R_{i+1}\ Y_{i+1}| \geq |Y_i|$. La figure ci-dessous illustre les conditions **C1** et **C2** qui seront justifiées plus loin:



Supposons que les conditions **C1** et **C2** ne soient pas respectées. Du fait que **C2** n'est pas vérifiée, la règle f_{i+1} aurait pu être appliquée avant f_i puisque $L_{i+1}C_{i+1}R_{i+1}$ serait une sous-chaîne de R_iY_i. D'autre part, puisque **C1** n'est pas vérifiée, l'application de la

114 Hart 1980.
115 *op.cit.* p.82

règle f_{i+1} ne modifierait que la chaîne Y_i sans toucher au contexte R_i. Dans ce cas, l'ordre de f_i et de f_{i+1} pourrait être inversé. Cette permutation serait légitime puisque les symboles réécrits par f_{i+1} se trouvent à droite de ceux réécrits par f_i.

6.3 Application de la dérivation canonique à l'algorithme de reconnaissance (*applying the canonic derivation to the membership test*)

Supposons que pour une chaîne Wi deux règles soient appliquables:

f_i $L_iC_iR_i$ -> $L_iD_iR_i$

f'_i $L'_iC'_iR'_i$ -> $L'_iD'_iR'_i$

sachant que $X_iL_iC_iR_iY_i = X'_iL'_iC'_iR'_iY'_i = W_i$

Ordonnons ces règles selon les critères suivants: f_i sera choisie en priorité sur f'_i si l'une des conditions suivantes, prises dans l'ordre, est vérifiée:

(D1) $|X_iL_iC_i| > |X'_iL'_iC'_i|$
(D2) $|X_iL_iC_iR_i| > |X'_iL'_iC'_iR'_i|$
(D3) $|L_iC_iR_i| > |L'_iC'_iR'_i|$
(D4) $i > i'$

Commentaire:

D1 permet à un nombre maximum de règles d'être candidates à l'étape suivante de réécriture en remplissant la condition **C2**.

D2 permet à un nombre maximum de règles d'être candidates à l'étape suivante de réécriture en remplissant la condition **C1**.

Lorsque **D3** est appliqué, les relations **D1** et **D2** se ramènent à des égalités, et par conséquent $R_i = R'_i$ et $L'_iC'_iR'_i$ est une sous-chaîne de $L_iC_iR_i$. L'ambiguïté est alors levée en appliquant le principe suivant:

> Les motifs les plus longs (contextes inclus) sont reconnus en priorité.

Lorsque **D3** ne permet pas de lever l'ambiguïté, les membres droits des règles de production sont identiques. Il y a réellement ambiguïté, mais cette situation peut être évitée par une écriture attentive des grammaires. Dans ce cas, le moteur d'inférences utilise un critère de sélection arbitraire: l'ordre inverse d'apparition des règles dans la grammaire.

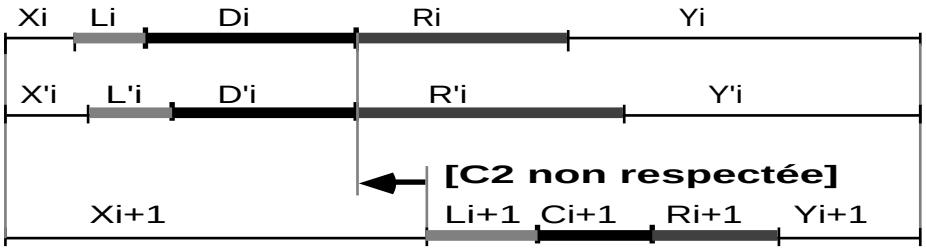
6.4 Simplification des critères dans le *Bol Processor* (*simplifying these criteria in the BP*)

Supposons qu'après l'application du critère **D1** deux règles f_i et f'_i soient candidates, et donc que $|X_iL_iC_i| = |X'_iL'_iC'_i|$. Après réécriture par f_i et f'_i respectivement, les chaînes seraient:

$W_{i+1} = X_iL_iD_iR_iY_i$ et $W'_{i+1} = X'_iL'_iD'_iR'_iY'_i$

avec $R_iY_i = R'_iY'_i$

Est-il nécessaire d'utiliser le critère **D2** pour choisir entre f_i et f'_i ? Ce critère permet de réaliser au mieux la condition **C1** sur les règles candidates f_{i+1}. Prenons l'exemple d'une règle f_{i+1} qui vérifie **C1** et pas **C2**:



Il est clair que la situation ci-dessus ne peut se produire car dans ce cas la règle f_{i+1} devait être appliquée avant f_i ou f_{i+1} selon le critère **D1**. Il s'ensuit que l'examen du critère **D2** n'est pas utile pour lever les ambiguïtés nées de l'application de **D1**.

Les situations d'ambiguïté découlant de l'application du critère **D1** seront donc immédiatement traitées par **D3**. Dans ce cas, $L'_iC'_iR'_i$ n'est pas nécessairement une sous-chaîne de $L_iC_iR_i$. Dans la version BP1, le critère **D3** n'est pas pris en considération par le moteur d'inférences, et c'est en définitive **D4**, c'est à dire l'ordre des règles, qui lève les ambiguïtés. Il s'ensuit que l'utilisateur doit respecter la règle suivante:

Règle des chunks

> Le membre droit d'une règle f_i ne peut être une sous-chaîne de celui d'une règle f_j telle que j<i.

6.5 Dépendance du contexte et dérivation canonique (*context dependency and canonic derivation*)

Les dérivations dans les grammaires contextuelles peuvent être représentées par des arbres syntaxiques.[116] Toutefois, ces arbres ne permettent pas de déduire la forme canonique d'une dérivation, c'est à dire l'ordre d'application des règles. Une solution satisfaisante a été proposée par Hart[117]. Elle consiste à insérer dans l'arbre syntaxique des arcs orientés exprimant deux types de contraintes:

> $<_{\mathbf{c}}$ la dépendance de contexte: la règle pointée par l'extrêmité de cet arc ne peut être appliquée que si le symbole placé à son origine est présent;
> $<_{\mathbf{f}}$ la 'libération' de contexte: la règle pointée par l'extrêmité de cet arc est 'gelée' jusqu'à ce que la règle pointée par son origine ait été appliquée.

On peut en outre ajouter à l'arbre syntaxique des arcs orientés $<_{\mathbf{d}}$ définis ainsi:

> Un arc relie f_i et f_j s'il n'existe pas de chemin entre ces noeuds;
> $f_i <_{\mathbf{d}} f_j$ si la dérivation de f_i se fait à droite de celle de f_j.

Ces arcs représentés simultanément forment un **graphe** 3-coloré **doublement ordonné** (*doubly ordered graph*), et ce graphe permet de déduire un ordre des règles qui correspond à la dérivation canonique.[118] Considérons par exemple la grammaire **LIN** suivante:

$\mathbf{f_1}$ S <—> EBA
$\mathbf{f_2}$ B <—> CD

---

[116] Révész 1985, p.57
[117] 1980
[118] Révész 1985, p.182

f_3 DA <—> DEF
f_4 EC <—> AAC

dans laquelle nous avons souligné les contextes, et la dérivation canonique à droite du mot:

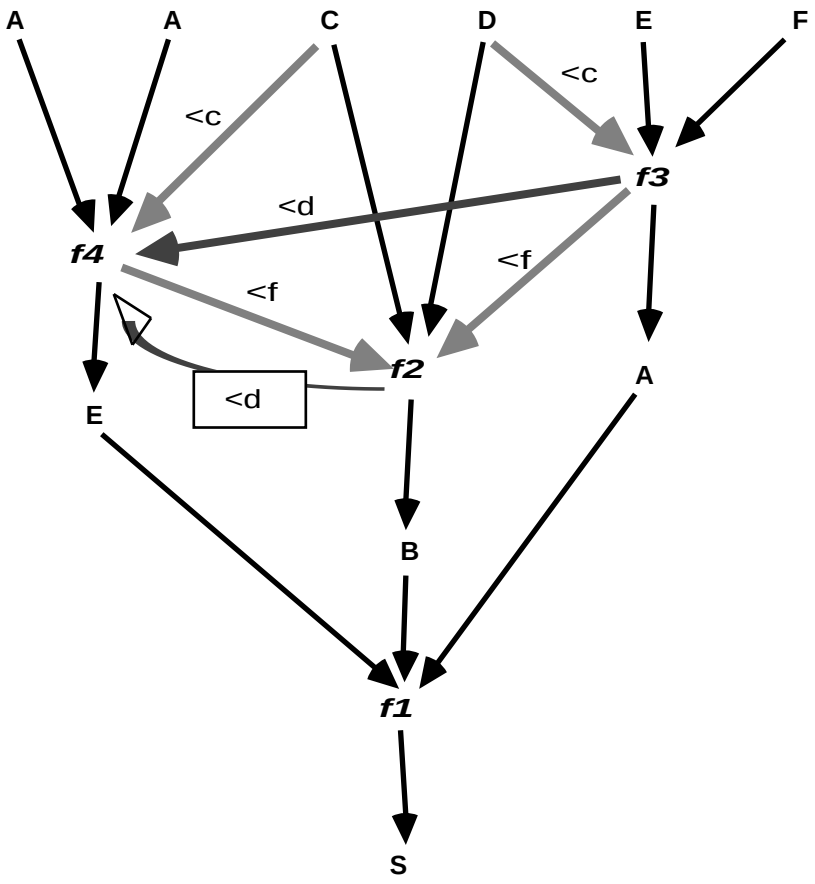
AACDEF => AACDA => ECDA => EBA => S
f_3 f_4 f_2 f_1

Si l'on appliquait le critère **D1** défini au paragraphe précédent, la dérivation serait:

AACDEF => AACDA => AABA
f_3 f_2

Par conséquent, le critère **D1** est insuffisant ici pour définir la dérivation canonique à droite. Nous avons représenté ci-dessous le graphe syntaxique contextuel (*context-sensitive syntactical graph*)[119] correspondant à l'analyse correcte. L'arc marqué $<_{\mathbf{d}}$ encadré ne fait pas partie de ce graphe, puisqu'il existe déjà un arc $<_{\mathbf{f}}$ entre f_4 et f_2. Il est clair que le gel de la règle f_2 contredit l'ordre imposé par la contrainte **D1**. Il est clair, d'autre part, que les arcs $<_{\mathbf{f}}$ qui proviennent d'un noeud f_i représentant une règle avec contexte à gauche sont tous orientés de droite à gauche, c'est à dire dans le même sens que les arcs $<_{\mathbf{d}}$. Enfin, il n'existe pas de paire d'arcs ($<_{\mathbf{c}}$,$<_{\mathbf{f}}$) ayant pour origine un contexte qui n'appartient pas à l'alphabet d'une grammaire transformationnelle.

[119] *ibid.*



Nous en tirons la règle suivante:

Règle des contextes

> Dans une grammaire LIN, un contexte à droite ne peut être composé que de symboles de l'alphabet extérieur de cette grammaire.

6.6 Test d'appartenance des grammaires ‘RND’ et ‘ORD’ (*membership test for ‘RND’ and ‘ORD’ grammars*)

Les grammaires ‘LIN’ permettent, à condition de respecter la règle des chunks et celle des contextes, de réaliser la dérivation canonique, en analyse, de n'importe quelle phrase du langage. Dans les grammaires ‘RND’ et ‘ORD’, c'est l'ordre des règles qui a priorité sur la position de dérivation (critère **D4**). De plus, lorsqu'une règle est sélectionnée, elle est appliquée jusqu'à saturation.

L'algorithme correspondant est le suivant:

```
Procédure REECRIRE(i)
Début
  Chercher le membre droit de f_i dans la phrase à partir de la droite;
  Remplacer l'occurence trouvée par le membre gauche de f_i ;
Fin
```

```
Procédure SATURER(i)
Début
  Tant que (la règle f_i est applicable)
  |    REECRIRE(i);
  |    test = 1;
  Fintant
Fin

Procédure ANALYSER
Début
  Répéter
  |    i = numéro de la dernière règle;
  |    test = 0;
  |    Répéter
  |    |    SATURER(i);
  |    |    i = i - 1;
  |    jusqu'à ce que (i = 0) ou (test = 1)
  jusqu'à ce que (test = 0)
Fin
```

Cet algorithme est acceptable lorsque tous les contextes des règles utilisent exclusivement des symboles de l'alphabet extérieur, et lorsque les motifs des membres droits des règles ne se chevauchent pas partiellement. Le problème du chevauchement de motifs est traité ailleurs.[120]

[120] Bel 1987.