Gouvernement du Canada
Ministère des Communications

Government of Canada
Department of Communications

Le Centre canadien de recherche sur l'informatisation du travail
Canadian Workplace Automation Research Centre

Bidirectionnalité des grammaires par déduction arrière contrôlée.

Marc Dymetman

Pierre Isabelle

QUEEN
P
308
.D96514
1990
c.2

Canada

Queen
P
308
D965f4
1990
c.2

Industry Canada
Library Queen
JUL 24 1998
Industrie Canada
Bibliothèque Queen

Bidirectionnalité des grammaires par déduction arrière contrôlée

Marc Dymetman

Pierre Isabelle

Centre Canadien de Recherches sur
l'Informatisation du Travail

Communications Canada

Laval

COMMUNICATIONS CANADA
FEB 14 1992
LIBRARY – BIBLIOTHÈQUE

Mars 90

Ce rapport est la traduction de:

Dymetman, Marc and Pierre Isabelle. 1990. Grammar Bidirectionality through Controlled Backward Deduction. In *Logic and Logic Grammars for Language Processing*, eds. Saint Dizier, P. and S. Szpakowicz. Chichester, England: Ellis Horwood.

N° de cat: Co 28-1/48-1990F

ISBN: 0-662-96089-0

P
308
D965
1990
C.2

DD 9601567
DL 11168144

Les opinions exprimées dans ce rapport n'engagent que ses auteurs.

This report is also available in English.

Résumé

Une grammaire peut être vue comme une théorie logique, et un analyseur syntaxique comme un démonstrateur de théorèmes spécialisé pour cette théorie. Un générateur peut et doit être vu de façon analogue. Dans notre approche, le même démonstrateur par chaînage arrière sert à l'analyse et à la génération. Les règles de la grammaire (c'est-à-dire les axiomes de la théorie) sont annotées avec des directives de contrôle s'appliquant soit au mode d'analyse, soit au mode de génération. Ceci assure une équivalence déclarative stricte entre les deux modes, maximise la clarté des spécifications linguistiques, mais laisse au linguiste une certaine souplesse dans le contrôle du comportement algorithmique de chacun des deux modes. Cette souplesse s'avère utile compte tenu de nos objectifs pratiques: construire un système bidirectionnel de traduction anglais-français capable de traiter les textes d'un sous-domaine restreint.

1. Introduction

La traduction automatique est un domaine naturel d'application des grammaires bidirectionnelles, aussi bien pour des raisons économiques - utiliser un système *unique* pour traduire d'une $langue_1$ à une $langue_2$ et vice versa - que pour des raisons théoriques - maintenir une spécification linguistique claire de laquelle analyse et génération sont simplement considérées comme deux réalisations computationnelles.

Historiquement, l'une des premières tentatives d'utilisation de grammaires réversibles furent les systèmes Q (Colmerauer 71), base algorithmique du système de traduction TAUM-METEO et précurseurs de Prolog. L'article de Colmerauer donnait un exemple de traitement réversible, mais l'applicabilité de l'approche était limitée par l'absence d'un concept général d'unification et par le manque d'un statut logique clair pour les systèmes Q, propriétés qui apparurent avec l'arrivée de Prolog.

Le projet CRITTER (Isabelle et al. 88) a pour objectif de traduire de l'anglais ou français des rapports de marchés agricoles produits par le ministère canadien de l'Agriculture.

CRITTER a recours à 2 grammaires bidirectionnelles (une pour l'anglais et une pour le français), à 2 lexiques unilingues bidirectionnels (qui permettent un accès efficace à la fois aux chaînes et aux unités sémantiques), et à un lexique de transfert bidirectionnel (qui décrit les correspondances entre les unités sémantiques propres à chaque langue).

A titre d'exemple de la couverture actuelle du système, considérons la phrase suivante:

> *it is reported by the OHMB that the market remained strong in Toronto ;*

CRITTER donne une traduction non déterministe de cette phrase et produit les équivalents français suivants:

{ (f1) l'OHMB signale que le marché est resté soutenu à Toronto,
(f2) il est signalé par l'OHMB qu'à Toronto le marché est resté soutenu,...}.

En *retraduisant* en anglais (f1) (par exemple), on obtient l'ensemble de phrases suivant:

{ (e1) the OHMB reports that in Toronto the market remained strong,
(e2) it is reported by the OHMB that the market remained strong in Toronto,
(e3) the market is reported by the OHMB to have remained strong in Toronto,...}.

Comme on le voit (e2) est identique à la phrase de départ, et (e1) et (e3) peuvent être considérées comme des paraphrases de (e2).

Paraphrase et traduction. Conçu au départ comme système de traduction, CRITTER peut aussi fonctionner — grâce à sa bidirectionalité — comme système de paraphrasage pour l'anglais ou pour le français.
Le paraphrasage et la traduction sont deux processus étroitement liés: alors que le paraphrasage met en rapport à l'intérieur d'une même langue des expressions partageant la même forme sémantique, la traduction, elle, met en rapport des expressions de deux langues qui, soit partagent la même forme sémantique (approche *interlinguistique*), soit possèdent des formes sémantiques entre lesquelles on peut établir une relation directe (approche par *transfert*).

Paraphrasage et Bidirectionnalité. Considérez les phrases suivantes[1]:

(p1) Jack seems to be given a black cat by Mary

(p2) Mary seems to give a black cat to Jack

...

(p3) It seems that Jack is given a black cat by Mary

(p4) It seems that Mary gives a black cat to Jack

...

Dans le système CRITTER, ces phrases sont toutes analysées en la même représentation sémantique "superficielle" (voir Figure 1). Le lecteur trouvera une explication détaillées de ces représentations dans (Isabelle et al. 1988).

[1] Les connotations agricoles de nos exemples sont quelquefois subtiles.

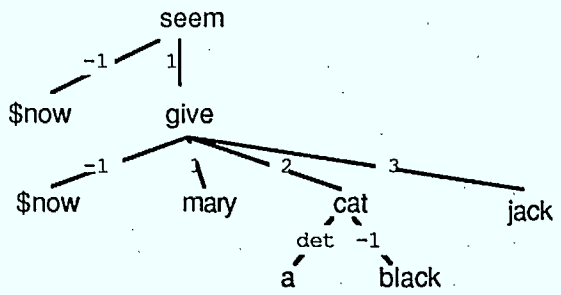


Fig. 1. Dans cette représentation sémantique, les étiquettes 1,2,3 denote des positions argumentaires relativement à un prédicat: par exemple, 'mary' est le premier argument de 'give', etc... Les étiquettes -1, -2, -3 (dont seulement la première est illustrée ici) ont la même interprétations, à ceci près qu'elles doivent être lues "de bas en haut": ainsi '$now'est un prédicat à une place (s'appliquant à un objet de type *event*, et signifiant que cet évènement se situe au temps présent). Dans l'exemple, une des deux instances du prédicat '$now' s'applique à l'évènement "don". Ce mécanisme notationnel permet de donner une forme prédicat-argument uniforme à la représentation sémantique, tout en gardant une structure *arborescente*. Cette structure arborescente a une *racine* bien définie, contrairement au cas d'un graphe orienté, pour lequel la notion de racine n'a pas de sens (pour plus de détails. voir (Isabelle et al. 1988)).

En vertu de la définition précédente, on peut dire que (p1), (p2), ..., sont des paraphrases les unes des autres, relativement à une relation d'interprétation sémantique donnée.

La bidirectionnalité du système se manifeste lorsque le processus de génération reçoit en entrée la structure sémantique de la Figure 1: le processus de génération produira alors — de façon non déterministe — *chacune* des paraphrases (pi), et *uniquement* celles-ci.

2. Le problème de la réversibilité computationnelle

Abstraitement, une grammaire est une spécification d'un ensemble récursivement énumérable de structures linguistiques (encodées de façon finie) A_v, ventilées selon leur catégorie syntaxique v (terminale ou non-terminale).

On suppose que ces structures linguistiques encodent des informations de divers types, telles que: syntaxe, sémantique, chaîne de caractères, ..., auxquelles il est possible d'accéder par des fonctions *syn*, *sem*, *string*... Ainsi pour exprimer que le contenu syntaxique de *A* est *Syn*, on écrira *syn(A)=Syn*, ou encore, dans une notation[2] "pointée": *A.syn=Syn*.

On supposera que (au moins) les champs *A.sem* et *A.string* ont une valeur bien définie pour toute structure linguistique *A*.

Le problème de la *réversibilité computationnelle* est de parvenir à dériver de la spécification d'une grammaire G deux programmes $Parse_G$ et $Generate_G$ (pouvant être identiques) efficaces[3], tels que:

- Si $Parse_G$ reçoit *String* en entrée, il énumère en sortie tous les *S* bien formés tels que *S.string=String*.

---

2 Cette notation est une variante de la notation fonctionnelle de LFG (Kaplan et Bresnan 1982), ou de celle de PATR (voir Shieber 1986). Voir aussi (Dymetman 1989).

3 De toute évidence, trouver des programmes quelconques qui satisfassent à ces exigences est trivial: le filtrage d'un ensemble récursivement énumérable par un critère récursif donne un autre ensemble r.e. En revanche, il peut être difficile de dériver des programmes de coût non prohibitif, et cela dépend fortement de la forme de la spécification initiale. Considérez par exemple la *cryptographie à clé publique*, dans laquelle la *clé publique* et la *clé privée* ont des spécifications fonctionnellement équivalentes, mais où seule la *clé privée* permet la dérivation de processus de codage et de décodage efficaces.

- Si *Generate*$_G$ reçoit *Sem* en entrée, il énumère en sortie tous les *S* bien formés tel que *S.sem=Sem*.

3. Les grammaires comme théories logiques

Il est devenu populaire de considérer les grammaires comme des formes particulières de théories axiomatiques, étroitement associées au sous-ensemble de la logique de premier ordre connu sous le nom de logique des *clauses de Horn* (ou des *clauses définies*).

Si l'on néglige les différences entre les divers formalismes appartenant à ce paradigme, une règle de grammaire peut se voir comme un axiome logique de la forme générale :

(1) $nt(NT) <= v_1(V_1), v_2(V_2), \ldots, v_n(V_n), constraint(NT,V_1,\ldots,V_n).$

Cet axiome exprime que si $V_1, \ldots, V_n$ sont des structures linguistiques bien formées de catégories $v_1,\ldots,v_n$, et que si la contrainte *constraint* est vérifiée entre $NT,V_1,\ldots,V_n$, alors NT est une structure linguistique bien formée de catégorie *nt*.

Notez que les littéraux du membre de droite de (1) ne sont pas ordonnés; c'est à la contrainte d'assurer, par exemple, que $NT.string$ est la concaténation de $V_1.string,\ldots,V_n.string$, et que les relations de compositionnalité syntaxique et sémantique sont vérifiées entre NT et ses filles[4].

Par la suite, nous supposerons que (1) est une clause de type défini, c.à.d. i) que les interprétations visées sont des interprétations de Herbrand (Kowalski 79b), ii) que les équations de la forme $X.function=Y$ sont des abréviations pour des expressions de la forme: $X = t(\ldots Y\ldots)$,

---

[4] Dans cette discussion, nous passons sous silence les spécificités des différents formalismes, plus particulièrement: i) la nature formelle des objets linguistiques manipulés (*DAGs* dans les grammaires dites "d'unification", *termes* dans les grammaires dites "logiques"); ii) la nature des contraintes: peut-on les définir récursivement comme dans le cas des DCGs ou faut-il les définir à l'aide de moyens expressifs plus limités (par exemple uniquement à l'aide d'équations, peut-être à l'exception de la relation de concaténation de chaîne, vue comme une primitive), ce qui permet d'obtenir des résultats de décidabilité plus forts.

où $t(\ldots Y \ldots)$ est un terme contenant la variable Y à un certain niveau d'enchâssement.

Sous cette simplification, l'ensemble des structures linguistiques bien formées défini par la grammaire est l'ensemble des termes clos NTa tel que nt(NTa) soit une conséquence valide des axiomes.

Classes de stratégies déductives. La plupart des stratégies déductives utilisées dans le traitement grammatical peuvent se répartir dans l'une des trois classes suivantes, en fonction de leur *règle d'inférence* fondamentale:

Déduction régressive (ou *par chaînage arrière*):

$$\frac{<\!\!-\!\!-\ B_1 \ldots B_m \ ,\ \ B'_i \ :\!-\ C_1 \ldots C_n}{q[\ <\!\!-\!\!-\ C_1 \ldots C_n\ B_1 \ldots B_{i-1}\ B_{i+1} \ldots B_m\ ]}$$

Déduction progressive (ou *par chaînage avant*):

$$\frac{A\ :\!-\ B_1 \ldots B_n \ ,\ \ B'_1 <\!\!-\!\!-, \ \ldots\ , B'_n <\!\!-\!\!-}{q[\ A <\!\!-\!\!-\ ]}.$$

Déduction assertive (ou *par lemmes*):

$$\frac{A <\!\!-\!\!-\ B_1 \ldots B_n \ ,\ \ B'_i <\!\!-\!\!-}{q[\ A <\!\!-\!\!-\ B_1 \ldots B_{i-1}\ B_{i+1} \ldots B_n\ ]}$$

Remarques sur la notation:

i) Dans chaque cas, θ est la substitution qui réalise l'unification la plus générale entre B_i et B'_i.

ii) La différence entre les symboles '<--' et ':-' est la suivante: '<--' correspond à une clause ajoutée dynamiquement dans "l'agenda", tandis que ':-' correspond à un axiome du programme.

La *déduction régressive* (ou *par chaînage arrière*) et la *déduction progressive* (ou *par chaînage avant*) font depuis longtemps partie de la tradition en programmation logique; par contre, la déduction dite ici *assertive* (ou encore *déduction par lemmes*) a été introduite par Pereira et Warren (83) sous le nom de "déduction de Earley"[5] à cause de l'analogie avec l'algorithme de Earley d'analyse syntaxique des grammaires hors-contexte (1970).

Dans les cas de la déduction *progressive* et de la déduction *assertive*, une question importante, au niveau du contrôle, est d'être capable de fournir, à partir d'une analyse *a priori* des buts activables ainsi que des propriétés de la théorie, des *filtres* efficaces capables d'éliminer de toute considération future la plupart des instanciations potentielles d'axiomes, pour ne conserver comme candidates que celles qui ont une chance de contribuer à une déduction réussie:

- *Filtres universels*: ainsi la déduction progressive ou assertive appliquée à l'analyse syntaxique doit recourir à des relations de projection universelles (comme celles imposées par une grammaire hors-contexte) entre la chaîne associée à un noeud et les chaînes associées aux filles de ce noeud, de façon à éliminer la plupart des instanciations d'axiomes[6] lexicaux et à exprimer des prédictions[7] fortes sur les

---

[5] Ce que ces auteurs désignent par déduction de Earley est l'exemple de notre déduction assertive (qui correspond à leur règle de réduction) appliquée à une règle de prédiction spécifique, que nous appellerons règle de prédiction de type Earley (voir note 3 de cette page).

[6] Le lexique étant défini comme l'ensemble des axiomes de grammaire "non conditionnelles".

[7] Comme la règle suivante d'instanciation de (Pereira et Warren 1983) (appelée règle de prédiction dans (Shieber 88)) :

Prédiction de type Earley: $\dfrac{A \text{ <-- } B_1 \ldots B_n \ , \ B'_i \text{ :- } C_1 \ldots C_m}{\theta[\ B_i \text{ <-- } C_1 \ldots C_m\ ]}$

Sur la même base, on peut introduire une règle de prédiction de type left-corner inspirée de l'analyse syntaxique *left-corner* (que l'on peut relier soit à la famille de la déduction progressive soit à celle de la déduction assertive, selon la version) :

Prédiction de type left-corner: $\dfrac{B'_i \text{ <-- } \ , \ A \text{ :- } B_1 \ldots B_m}{\theta[\ A \text{ <-- } B_1 \ldots B_m\ ]}$

instances de règles de la grammaire qui peuvent participer utilement à une démonstration. De même, l'application de la déduction assertive à la génération comme dans (Shieber 88) suppose certaines conditions restrictives (monotonicité sémantique) sur le lien entre la sémantique d'un noeud et les sémantiques de ses filles.

- *Relations d'accessibilité*: d'autres types de filtres qui ont été utilisés font intervenir des *tables d'accessibilité* [*reachability tables*] (Kay, 1980), ou *liens [links]*, qui représentent des relations précompilées à partir de la spécification grammaticale. Ces relations expriment des contraintes entre la structure linguistique associée à un noeud et les structures linguistiques associées à certains descendants de ce noeud dans l'arbre de dérivation. De telles méthodes ont été utilisées dans un contexte d'analyse dans BUP (Matsumoto et al., 1983) (liens a priori entre les catégories syntaxiques) et dans un contexte de génération dans BUG (Van Noord, 1988) (liens a priori entre les structures sémantiques).

A l'opposé, une conséquence naturelle de l'approche par *déduction régressive* est la "pertinence" de chaque étape d'inférence relativement au théorème à démontrer; cette propriété permet de relaxer les exigences concernant la relation entre la sémantique d'un noeud donné et les sémantiques de ses filles, sans rendre inopérant le processus de génération[8]. Le problème de contrôle[9] principal dans le cas de la déduction régressive consiste en la question de déterminer le choix du but B_i à lancer à la prochaine étape d'inférence. Le non-déterminisme

---

(Remarques sur la notation: comme précédemment est la substitution qui réalise l'unification la plus générale entre B_i et B'_i.)

[8] Ce qui peut être utile dans le traitement d'idiomes (voir section 7).

[9] Mais voir également la discussion de la récursion à gauche (section 5).

potentiel et le niveau d'instanciation de chaque but par rapport à celui des autres sont alors les paramètres essentiels de ce choix.

4. La déduction régressive et la réversibilité computationnelle

Problèmes d'ordonnancement. Dans le contexte du traitement des langues, les grammaires à clauses définies (DCGs), munies de leur interprétation Prolog standard constituent l'exemple le plus connu de déduction régressive. Soit la règle DCG (2) et la procédure Prolog correspondante (3):

```
(2)   s(S) -->                    (3) s(S, L1, L3) :-
          np(NP),                         np(NP, L1, L2),
          vp(VP),                         vp(VP, L2, L3),
          {combine1(NP, VP, S)}.          combine1(NP, VP, S).
```

On a supposé ici que *combine1* définit la relation entre les structures de chaque constituant et en particulier qu'il stipule comment *S.sem* est construit à partir de *NP.sem* et *VP.sem*.

On remarque que d'après l'interprétation (3), la règle (2) amalgame deux types différents d'ordre: a) l'ordre relatif des chaînes associées aux non-terminaux (le *np* précède le *vp*); et b) l'ordre relatif du lancement des buts Prolog correspondants (le but *np* est évalué avant le but *vp*). Si l'ordre des non-terminaux fait partie intégrante de la *spécification grammaticale*, celui des buts est exclusivement une question de *contrôle*.

Le procédure (3), interprétée de façon standard, est bien adaptée aux objectifs d'analyse syntaxique, mais donnerait lieu à un programme de génération totalement inefficace. Par exemple, ce serait le cas dans une grammaire basée sur un modèle "lexicaliste"[10]. Dans une telle grammaire, la mise en correspondance des structures sémantiques et des configurations syntaxiques (ou fonctions) est régie par le lexique. En particulier, la relation entre la structure sémantique de la phrase

---

[10] Pour une introduction aux notions linguistiques utilisées par la suite, voir (Sells 1985), et pour une théorie linguistique à laquelle notre approche est liée, voir (Pollard and Sag 1987).

S.sem et la structure sémantique du sujet *S.suj.sem* ne peut être déterminée qu'à partir de l'entrée lexicale du verbe principal. Compte tenu de l'existence de phénomènes comme la *montée* [raising], *S.suj.sem* peut être enchâssé arbitrairement bas dans *S.sem*:

(4) a) Max seemed to begin to be tougher to convince.
b) past'(seem'(begin'(tougher'(convince'(one',Max')))))

Les mêmes modèles linguistiques postulent une relation beaucoup plus directe entre *S.sem* et *VP.sem* (ou plus généralement entre *X.sem* et *X.head.sem*).

Le procédure (3) se révèle donc réaliser une stratégie très médiocre en génération, puisqu'alors seul *S.sem* est connu au moment de l'appel de cette procédure.

Annotations de contrôle et compilation spécifique au mode. Si on désire se servir d'une même grammaire aussi bien en mode d'analyse qu'en mode de génération, un contrôle du processus de déduction spécifique au mode est nécessaire. Dans ce but, on pose que chaque règle de grammaire est constituée de *deux composantes* complémentaires: 1) une description du contenu grammatical (déclaratif), commun aux deux modes; et 2) une description de la façon dont le processus de déduction doit être réalisé dans chacun des deux modes. Ces deux composantes réalisent ainsi la séparation logique/contrôle défendue dans (Kowalski 1979a).

Les règles DCG sont augmentées de la façon suivante: a) les buts sont préfixés avec des identificateurs uniques; et b) les règles contiennent des annotations de contrôle qui stipulent l'ordre de traitement des buts pour chaque mode. Par exemple, la règle DCG (5) est augmentée pour devenir (6):

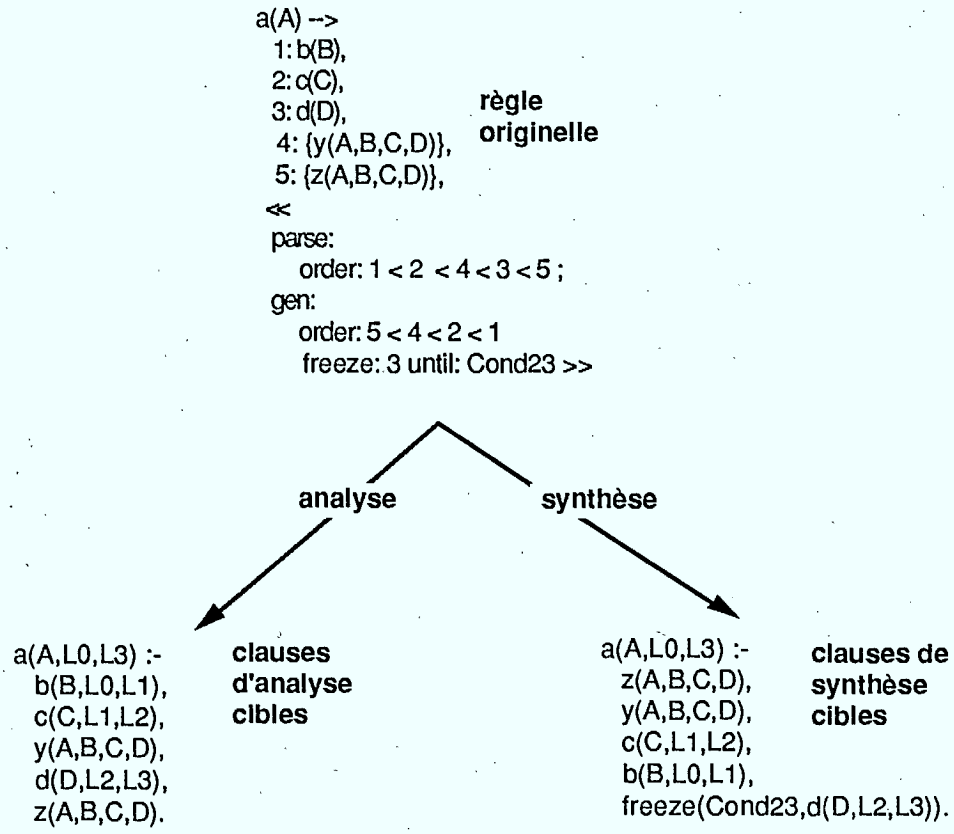
(5)
```
a(A) -->
    b(B),
    c(C),
    d(D),
    {y(A,B,C,D)},
    {z(A,B,C,D)}.
```

(6)
```
a(A) -->
    1: b(B),
    2: c(C),
    3: d(D),
    4: {y(A,B,C,D)},
    5: {z(A,B,C,D)},
    <<
    parse:
        order: 1 < 2 < 4 < 3 < 5;
    gen:
        order: 5 < 4 < 2 < 1
        freeze: 3 until: cond(A,B,C,D) >>
```

A l'intérieur de l'annotation de contrôle "<< ... >>", *parse* et *gen* désignent deux modes de traitement différents. Dans le mode *parse* (analyse), les buts 1, 2 et 3 sont déclenchés dans cet ordre (l'ordre des autres buts est indifférent). Dans le mode *gen*, le but 2 doit être déclenché avant le but 1; en outre, le but 3 doit être gelé (Colmerauer 82) jusqu'à ce que la précondition *cond* soit satisfaite.

Les règles annotées sont précompilées vers des programmes Prolog spécifiques à chaque mode[11]. Compte tenu de la nature des annotations de contrôle, ces programmes possèdent des contenus déclaratifs équivalents, mais chacun est optimisé en fonction de son mode d'utilisation (voir Figure 2).

[11] Ces programmes sont interprétés par un interpréteur Prolog étendu qui traite le gelage de but (voir Cochard et Dymetman, à paraître).



(Fig. 2)

La règle (2) sera ainsi reformulée de la façon suivante:

```
(7)     s(S) -->
              1: np(NP),
              2: vp(VP),
              3: {combine1(NP, VP, S)},
                   <<
                    parse:
                         order: 1 < 2 < 3 ;
                    gen:
                         order: 3 < 2 < 1 >>
```

Cette règle sera compilée pour l'analyse en la procédure Prolog (3), mais pour la génération en la procédure (8):

```
(8)     s(S, L1, L2) :-
            combine1(NP, VP, S),
            vp(VP, L3, L2),
            np(NP, L1, L3).
```

Ainsi, pour la génération, les buts sont réordonnés de telle façon que: *combine1* est appelé en premier, ce qui provoque l'instanciation de *VP.sem; vp* est appelé ensuite, ce qui finit par provoquer un accès au lexique (entrée lexicale du verbe principal) dont l'une des conséquences est l'instanciation de *NP.sem*. Finalement, *np* est appelé avec une sémantique connue.

De façon assez surprenante, ce simple réordonnancement *statique* des buts, *local* aux règles, suffit dans bien des cas à garantir que la grammaire se comporte raisonnablement bien en analyse comme en synthèse.

Évaluation asynchrone. Malgré tout, des demandes conflictuelles peuvent apparaître en ce qui a trait à l'ordre optimal des buts, et il est parfois nécessaire de tirer profit du réordonnancement dynamique *non local* possible grâce à un gelage [goal-freezing] de buts. Par exemple, si la génération de *s* est faite de la façon qui vient d'être décrite, au moment de l'accès lexical au verbe, le sujet n'a pas encore été généré; certaines informations concernant l'accord du verbe (nombre, personne, genre) ne sont donc pas disponibles, ce qui aurait normalement pour conséquence la génération aveugle d'une inflexion arbitraire sur le radical verbal. Le gelage de buts permet de résoudre ce problème:

```
(9)   v(V) -->
            1: rad(RAD),
            2: suff(SUFF),
            3: {combine3(RAD, SUFF, V)},
                 <<
                  parse:
                       order: 1 < 2 < 3 ;
                  gen:
                       order: 3 < 1
                       freeze: 2 until: nonvar(SUFF.agr) >>
```

En mode de génération, *combine3* est appelé en premier. Entre autres choses, cette opération instancie *RAD.sem* et unifie *V.agr* avec *SUFF.agr*. Il est dès lors possible de générer le radical du verbe en consultant le lexique (appel à *rad*). Cependant, la génération du suffixe inflectionnel (but *suff*) est reportée jusqu'à l'instanciation de

SUFF.agr. Cette condition n'est remplie qu'après qu'une partie suffisante du groupe nominal sujet a été générée pour que *NP.agr* soit instancié. Au même moment, *SUFF.agr* devient instancié, en vertu de l'action des règles d'accord (implicites dans *combine1*) et de la transmission des attributs de tête (head features) de *VP* à *V*.

Ordonnancement statique, ordonnancement dynamique et déduction régressive. Réexaminons à présent les rapports de i) l'*ordonnancement statique des buts* et de ii) l'*ordonnancement dynamique des buts par évaluation asynchrone* à la règle de déduction régressive de la section 3, reproduite ici:

$$\frac{<\!\!-\!\!-\ B_1 \ldots B_m \ ,\ B'_i :\!- C_1 \ldots C_n}{\theta[\ <\!\!-\!\!-\ C_1 \ldots C_n\ B_1 \ldots B_{i-1}\ B_{i+1} \ldots B_m\ ]}$$

Nous supposerons maintenant que la liste des buts dans le membre droit d'une clause est ordonnée; il nous faut alors décrire comment la sélection de B_i se fait à chaque pas de déduction et comment la liste de buts en sortie (le dénominateur de la règle de déduction ci-dessus) est ordonnée. Le rapport entre l'ordonnancement statique, l'ordonnancement dynamique et la règle de déduction précédente est défini par les spécifications suivantes :

- *sélection de* B_i: parmi les $B_1 \ldots B_m$, il se peut qu'il y ait un but B_f "gelé" dont la précondition de déclenchement est maintenant satisfaite; dans ce cas, on prend $B_i = B_f$, autrement on prend $B_i = B_1$.

- *ordonnancement de la liste de buts résultante*:

i) on divise le corps de l'axiome $C_1 \ldots C_n$ en deux listes: $Ls = [S_1 \ldots S_k]$ et $Lf = [F_{k+1} \ldots F_n]$, où les F_i sont les buts gelés dans le corps de l'axiome, et où les S_i sont les buts restants, ordonnés selon leurs annotations d'ordre (propres au mode de traitement considérée). Contrairement à celui de Ls, l'ordre de Lf est considéré comme non significatif;

ii) on produit en sortie la liste de buts:

$\theta[\text{ <-- } S_1 \ldots S_k \; B_1 \ldots B_{i-1} \; B_{i+1} \ldots B_m \; F_{k+1} \ldots F_n \;]$.

5. Traitement de la récursivité à gauche

Un problème bien connu des analyseurs syntaxiques par déduction régressive concerne le traitement des constructions récursives à gauche dans la grammaire.

Soient les règles suivantes :

```
(10)  np(NP) -->
            1: np(NP'),
            2: pp(PP),
            3: {combine2(PP, NP', NP)},
                  <<
                   parse:
                        order: 1 < 2 < 3 ;
                   gen:
                        order: 3 < 2 < 1 >>.
      np(NP) -->
                  n1(NP).
```

Le but des règles ci-dessus est d'intégrer dans un *np* un nombre non spécifié de modificateurs prépositionnels (*the hotel in Vancouver with a Chinese roof*). *n1* est le *np* "nu" (*the hotel*), et *combine2* construit les structures syntaxique et sémantique de *NP* à partir des structures de *PP* et de *NP'*.

Cependant, le problème évident avec cette règle est qu'elle est "récursive à gauche": bien que sa lecture déclarative soit parfaitement naturelle et représente directement la signification visée de la règle, la traduction DCG standard conduit à une boucle en mode d'analyse. Pour cette raison, aucun linguiste computationnel n'écrira de telles règles. Il les transformera plutôt en des règles équivalentes qui, bien que plus complexes, auront le comportement algorithmique désiré (voir Figure 3 ci-après).

Toutefois, il est intéressant de noter que la règle (10), lorsqu'elle est utilisée en synthèse, fonctionnera parfaitement bien, si seulement

on veille - comme dans l'exemple précédent *vp* - à ordonner le but *combine2* avant les buts *pp* et *np*[12].

Heureusement, il y a moyen de tourner cette difficulté: le grammairien doit avoir la possibilité d'écrire la règle *np* de façon naturelle et le compilateur aura la responsabilité d'exécuter les transformations nécessaires pour l'analyse et pour la génération.

C'est précisément ce qui est fait dans le système CRITTER, comme l'illustre la figure 3 :

---

[12] Cela ne signifie pas qu'un problème analogue à la récursivité à gauche ne peut pas *en principe* se poser en *génération*, mais plutôt qu'il n'apparaîtra pas si la grammaire respecte les conditions énumérées à la section 6 ci-dessous. Le proglème de l'élimination en génération de la récursivité à gauche dans des grammaires qui ne respectent pas ces conditions déborde du cadre de cet article.

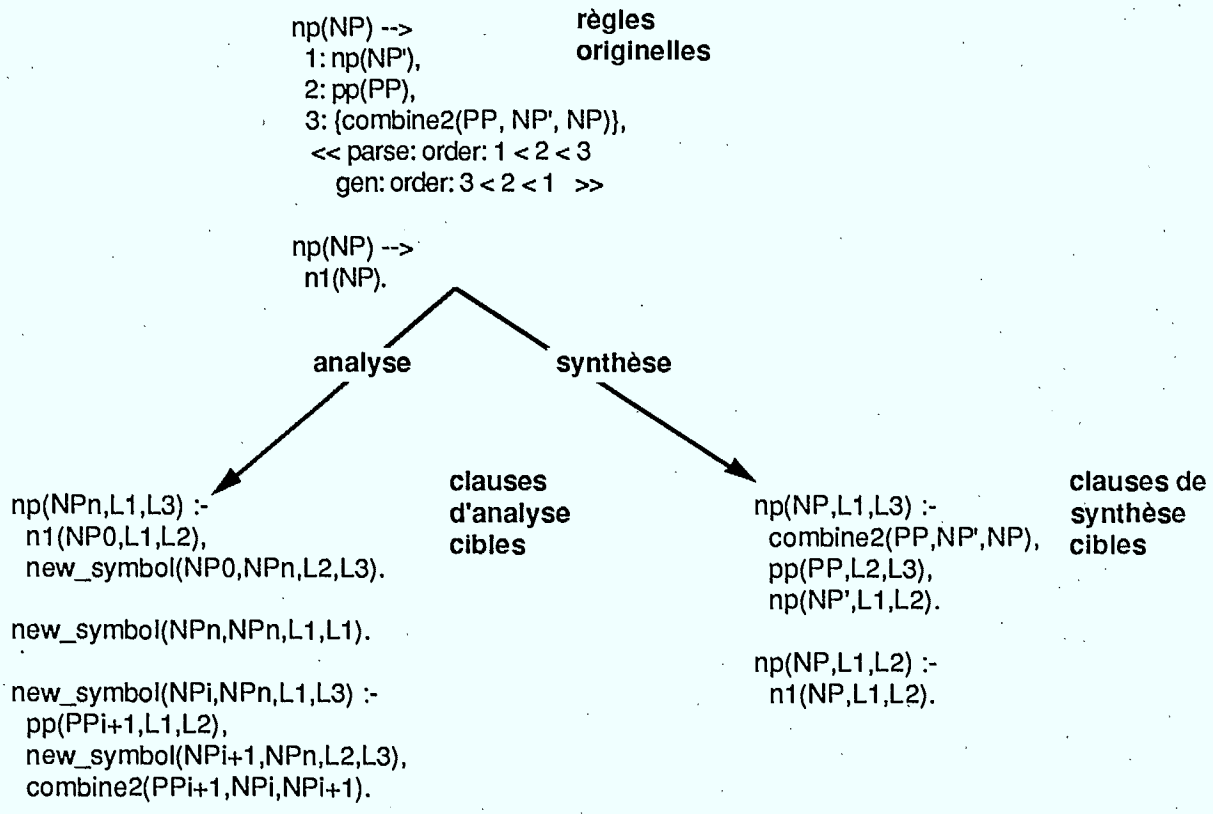


Fig. 3

Dans l'implémentation actuelle, il faut signaler au compilateur les procédures[13] récursives à gauche. La compilation spécifique à l'analyse introduit un non-terminal "factice" *new_symbol* et réalise une transformation vers les clauses d'analyse de la figure ci-dessus. Nous ne justifierons pas ici cette transformation, mais nous avons pu démontrer que les clauses résultantes gardent inchangé le contenu déclaratif de la règle[14].

Quant aux clauses de synthèse, elles sont analogues aux règles originelles, à ceci près que les annotations d'ordre dans ces règles ont

---

[13] Comme toujours dans le jargon Prolog, une procédure est le paquet de clauses définissant un prédicat.

[14] Le processus est apparenté aux techniques de réécriture d'une grammaire hors-contexte sous une forme normale, la partie délicate étant la façon de transférer les contraintes entre variables.

servi à ordonner les sous-buts, de la même façon que dans le cas de la Figure 2.

6. Conditions pour la réversibilité

Notre expérience a montré que les techniques présentées ci-dessus suffisent à assurer la bidirectionalité pratique des grammaires du français et de l'anglais appliquées à des textes réels de complexité moyenne. Nous allons maintenant prendre un peu de recul et examiner les propriétés générales des grammaires qui rendent cela possible.

Formes normales sémantiques. Comme l'ont souligné nombre de chercheurs, le problème de la réversibilité dépend de certaines hypothèses de base concernant le niveau des représentations sémantiques. Ainsi, dans le cas où l'on suppose qu'une représentation sémantique est une *proposition* abstraite de la logique du premier ordre (c-à-d. une classe d'équivalence de *formules* de logique du premier ordre équivalentes), il n'est pas possible de coder (*bijectivement* et *effectivement*) cette proposition en une représentation finie. Ce fait est à l'origine de difficultés insurmontables pour le processus de génération (voir Appelt 1987).

L'approche présentée ici s'applique uniquement au cas d'objet sémantiques possédant une forme normale encodée de façon finie. Ces objets ne sont pas aussi abstraits que des propositions logiques; en simplifiant, on peut dire qu'ils sont justes suffisamment abstraits pour "ignorer" les idiosyncrasies grammaticales de deux (ou plusieurs) langues différentes[15].

Propriétés des grammaires suffisantes pour la réversibilité. Revenons au problème de la génération et supposons, sans perte de généralité, que les arbres de dérivation sont des arbres binaires.

---

[15] La notion de paraphrase que nous voulons caractériser est bien entendu beaucoup plus restrictive que ce qui découlerait de l'équivalence en logique du premier ordre; dans ce contexte, "paul drinks" et "paul drinks and ideas sleep or ideas do not sleep" ne peuvent être considérées comme des paraphrases, pas plus que "paul drinks" et "paul boit et les idées dorment ou ne dorment pas" ne peuvent être considérées comme des traductions valables.

Les quatre propriétés suivantes suffisent alors à garantir qu'un ordonnancement local des buts (interne aux règles de grammaire) se traduira par une procédure de génération complète (complète dans le sens où cette procédure énumèrera toutes les structures phrastiques S partageant une structure sémantique donnée.)

(Prop.1) Toutes les structures linguistiques *A* possèdent un attribut sémantique Sem ainsi qu'un attribut syntaxique Syn.

(Prop.2) Dans un arbre de dérivation, si la structure linguistique *A* a des filles *B* et *C*, la sémantique de l'une des filles - disons *C* - (la tête) est identique à celle de *A* (cas d'incorporation d'un argument: *B* est un argument) ou est "contenue" dans celle de *A* - relativement à un certain ordre partiel (la *taille*) des structures sémantiques n'admettant aucune chaîne descendante infinie - (cas de l'incorporation d'un modifieur: *B* est un modifieur).

(Prop.3) Avec la même notation que précédemment, si la sémantique de *A* est connue et si la sémantique *et* la syntaxe de *C* sont connues, la sémantique de *B* est déterminée de manière univoque, *et* est de taille strictement inférieure à la taille de *A*. (Voir l'exemple (8) ci-dessus, où la sémantique du sujet *NP* (*B*) sera déterminée seulement au moment où *VP* (*C*) "aura pris une décision", reflétée à la fois dans sa syntaxe et dans sa sémantique, concernant la mise en correspondance des arguments avec des positions syntaxiques déterminées.)

(Prop.4) Il existe un mécanisme dans la grammaire permettant d'exprimer le "niveau de bar" d'une phrase[16] - soit en recourant aux noms des non-terminaux ou encore grâce à un indice interne aux structures linguistiques - ce niveau de bar étant incrémenté à chaque incorporation d'argument.

---

[16] Voir par exemple (Sells 1985, p.28), pour une explication de cette notion.

Preuve de complétude (esquissée): Les Props. 2 et 4 garantissent qu'il est possible de transférer de l'information sémantique d'une mère à sa fille tête et que la répétition de tête en tête de cette opération aboutira finalement à une structure lexicale *C*, c'est à dire une structure déterminée dans tous ses aspects (en particulier syntaxique et sémantique) par simple examen du dictionnaire. La propriété 3. ci-dessus assure alors que la sémantique de la soeur *B* de la tête *C* est dès lors connue, et que sa taille est inférieure à celle de la sémantique de A. Par récurrence, B finira donc par être complètement connue, et la mère A de *B* et *C* sera complètement déterminée à partir de *B* et *C;* ainsi la détermination complète de tous les noeuds de l'arbre de dérivation se propagera vers le haut jusqu'à ce que finalement *S* devienne complètement connu.

7. Recherches futures.

Idiomes. Il est possible d'assouplir l'exigence qui veut que les filles et la mère soient aient une relation d'inclusion (ce qui est en substance la condition de *monotonicité sémantique* de (Shieber 88)). Par exemple, on pourrait définir un prédicat "de réduction idiomatique" *idiom_red(Sem_init, Sem_fin)*. A partir de la "sémantique initiale" de *A*, obtenue compositionnellement de façon standard à partir de la "sémantique finale" de ses filles *B* et *C*, *idiom_red* pourrait produire la sémantique finale de *A*, grâce à la consultation d'une table de "formations idiomatiques".

Un dispositif de ce genre permettrait diverses opérations de normalisation, comme la réduction de *kick(bucket)* en *die*, ou la normalisation de "dormir volontiers" (l'équivalent français de la célèbre expression hollandaise "graag slapen") en "aimer dormir".

Un tel mécanisme n'empêcherait pas la réversibilité, à condition que *idiom_red*, inversé en génération pour permettre de passer de la sémantique finale à la sémantique initiale, ait un non-déterminisme limité. Il faut remarquer que ce type de mécanisme serait par contre difficile à implanter dans un paradigme de déduction progressive ou de type Earley, car la sémantique de la phrase n'a pas alors de relation évidente avec la sémantique des éléments lexicaux utilisés pour la construire, une conséquence apparemment inévitable de ces approches. (Mais voir cependant (Van Noord 1989) pour une approche de l'idiomaticité dans le paradigme de la déduction progressive).

Réversibilité computationnelle et réversibilité linguistique. Le problème que cette communication a abordé jusqu'ici est celui de la *réversibilité computationnelle*, à savoir: comment dériver des *programmes* d'analyse et de synthèse efficaces à partir d'une même grammaire déclarative.

Cependant, même supposant le problème de la réversibilité computationnelle résolu, un autre problème reste entier: celui de la *réversibilité linguistique*. Ce problème consiste à spécifier une grammaire bidirectionnelle d'une façon qui soit *adéquate linguistiquement* à la fois pour l'analyse et la génération. Comme les remarques suivantes vont tenter de le montrer, cela est loin d'être une question triviale.

Une grammaire BG, si elle est bidirectionnelle, doit nécessairement se rapprocher du critère d'*adéquation observationnelle* plus que ne devraient le faire deux grammaires différentes, l'une pour l'analyse (AG), l'autre pour la synthèse (SG). La raison pour cela est que la grammaire d'analyse AG peut sans danger se montrer hyper-permissive quant aux chaînes qu'elle accepte, puisque, dans une large mesure, elle peut compter sur le fait que le texte d'entrée soit bien formé[17]. La grammaire de génération SG, quant à elle, peut se permettre d'être sous-permissive quant aux chaînes qu'elle génère, car on ne lui demande que de garantir une sortie bien formée et *non* de fournir toutes les variations possibles exprimant le même contenu. A l'opposé, le recours à une grammaire bidirectionnelle BG impose plus de contraintes au travail du linguiste qui est obligé de raffiner ses descriptions afin de représenter précisément la réalité linguistique: si un texte peut être analysé avec une BG, il peut aussi être généré par une BG et vice versa.

L'exigence pour une grammaire bidirectionnelle de respecter l'adéquation observationnelle est bien fondée du point de vue théorique, mais il peut arriver que la discipline ainsi imposée au travail de description linguistique soit délicate à respecter en pratique.

Cela est particulièrement vrai pour certains phénomènes semi-productifs, que ce soit en syntaxe, ou en morphologie dérivationnelle ou

---

[17] Evidemment, cela ne va pas sans problème; la sous-spécification de AG finira par se trahir et produira de fausses ambiguités syntaxiques, même lorsqu'on peut se fier à la bonne formation du texte analysé.

compositionnelle, phénomènes pour lesquels il est difficile de produire une description complète en pratique. Considérons les exemples suivants:

(a_1) Reagan supporter	(a'_1) Roosevelt supporter
(b_1) reaganite	(b'_1) *rooseveltite
(a_2) africain de l'ouest	(a'_2) américain de l'ouest
(b_2) ouest-africain	(b'_2) *ouest-américain
(a_3) portrait by Goya of the duchess of Alba	(a'_3) portrait of Saskia by the greatest Dutch painter
(b_3) portrait of the duchess of Alba by Goya	(b'_3) *portrait by the greatest Dutch painter of Saskia

Dans chacun des exemples 1, 2 et 3, nous avons supposé :

- que a_i et b_i partagent la même sémantique (même chose pour a'_i et b'_i);
- que si on exprime une sémantique donnée par le procédé a, on a affaire à un mécanisme entièrement productif, mais que si on l'exprime par le procédé b, à un mécanisme semi-productif: ainsi a_i et a'_i sont tous les deux possibles, mais b_i et b'_i ont des statuts d'acceptabilité différents.
- une description complète des conditions dans lesquelles le procédé de type b serait acceptable est difficile à obtenir en pratique.

Dans ces conditions, une stratégie acceptable pourrait être:

- en analyse, admettre le procédé de type a aussi bien que celui de type b; il en résultera une analyse sur-génératrice, mais cela sera compensé par le fait que les entrées mal formées seront rares;
- en génération, n'admettre que le procédé de type b; il en résultera une analyse sous-génératrice, un prix

raisonnable à payer pour garantir un haut degré d'acceptabilité.

Une telle approche suppose un relâchement de la condition que la grammaire d'analyse syntaxique et celle de génération (et leurs composants lexicaux respectifs) soient déclarativement *équivalentes*. On exigera plus faiblement que la grammaire de génération "*implique*" la grammaire d'analyse, dans le sens que l'ensemble des structures linguistiques décrites par la grammaire de synthèse devra être un sous-ensemble de celui décrit par la grammaire d'analyse. Une façon de réaliser cela pourrait être d'utiliser une seule et unique grammaire "bidirectionnelle", comme plus haut, mais de pouvoir signaler un but *gc* dans le corps d'une règle avec une nouvelle annotation *generation_constraint*, qui aurait pour effet d'imposer la vérification de ce but *seulement* en mode génération[18].

Extraposition. Pour les besoins de l'exposé, nous avons restreint la discussion précédente au cas des DCGs, alors qu'en réalité nous utilisons une extension des DCGs connue sous le nom de Grammaires d'Extraposition (XGs), qui permettent de traiter certains cas intéressants de *dépendances non bornées* (Pereira 1981).

Les XGs posent de sérieux problèmes d'efficacité en génération: en effet, l'ordre des appels dans ce mode, comme nous l'avons vu plus haut, ne respecte pas nécessairement l'ordre gauche droite des constituants. En conséquence, au moment de l'appel d'un non-terminal *nt*, les listes d'extraposition en entrée et en sortie de ce non-terminal sont *toutes les deux* inconnues; la génération n'est donc pas en mesure de "savoir" si le constituant doit être généré comme une chaîne ou extraposé vers la

---

[18] L'exemple extrême de l'utilisation de telles contraintes serait d'appeler le but 'fail' seulement en mode génération, par le fait même invalidant la règle de grammaire contenant la contrainte. Une généralisation de l'annotation *génération_constraint* serait d'imposer des annotations de *niveau d'acceptabilité* aux différents buts d'une clause.

gauche. Les conséquences algorithmiques de cet état de fait peuvent être désastreuses.

Nous avons mis au point certaines techniques qui rendent les XGs raisonnablement efficaces en génération. Elles impliquent un "marquage" de la sémantique des expressions extraposées de telle sorte que durant la génération, la décision de réaliser l'expression directement comme une chaîne ou de l'extraposer à gauche est complètement déterminée par la présence ou la non-présence de cette marque dans la sémantique de l'expression. Nous avons pu ainsi traiter plusieurs types de clauses relatives, y compris certains cas de *pied piping* (eg.: "the owner of which").

Nous espérons que ces techniques pourront être généralisées, bien qu'à ce stade, nous n'en ayions pas encore obtenu une rationalisation théorique satisfaisante. C'est un domaine où d'autres recherches s'imposent.

8. Travaux connexes

Les approches au problème de la réversibilité grammaticale peuvent être approximativement classées le long de deux dimensions[19]:

- réversibilité *complète* ou *partielle* : les modes d'analyse et de génération sont-ils ou non deux réalisations algorithmiques de la même relation abstraite?

- réversibilité *automatique* ou *manuelle*: les programmes d'analyse et de génération sont-ils dérivés automatiquement (soit par *compilation* dépendante du mode ou encore par *interprétation* dépendante du mode) de la spécification commune, ou le sont-ils manuellement?

Sans prétendre à l'exhaustivité, certaines des approches représentatives du problème de la réversibilité sont les suivantes:

PHRED (Jacobs 1985), Ariane (Vauquois et Chappuy 1985), et Rosetta (Landsbergen 1987) peuvent tous être considérés - à des degrés divers - comme des exemples de l'approche "*manuelle*" à la réversibilité.

---

19 Remarque : Dans l'abstrait, toute théorie linguistique postulant un niveau sémantique quelconque devrait se prêter parfaitement à une réalisation réversible complète tout au moins manuelle. En pratique, toutefois, ceci supposerait: 1) qu'il existe une "grammaire de référence" stable pour cette théorie, qui spécifie exactement les constructions admises (c.à.d. qui soit une grammaire *formelle*) et que cette grammaire soit neutre par rapport à son mode d'utilisation (analyse ou génération); 2) qu'il soit possible d'obtenir une implantation en analyse de cette grammaire raisonnablement efficace; 3) qu'il soit possible d'obtenir une implantation en génération de cette grammaire raisonnablement efficace.
L'expérience démontre qu'il est difficile de réaliser simultanément ces trois conditions. La plupart du temps, les linguistes computationnels se contenteront de réalisations partielles de la grammaire formelle, et exploiteront la latitude ainsi offerte pour rendre efficace le mode d'implantation choisi - analyse *ou* génération. Ou bien (ce qui est pire), la spécification de la grammaire formelle sera biaisée en vue de favoriser un mode, rendant nébuleuse la distinction entre spécification formelle et programme et augmentant d'autant la difficulté de réalisation de l'autre mode.

PHRED. PHRED et PHRAN (Wilensky et Arens 1980) sont respectivement les composants génération et analyse d'une interface en langue naturelle au système Unix: PHRAN analyse les questions de l'utilisateur touchant le fonctionnement du système et PHRED génère les réponses.
PHRED a été élaboré après PHRAN, et bien que ces deux programmes semblent réaliser une certaine réversibilité, grâce au partage d'une "base de connaissances linguistiques communes", rien n'indique qu'ils permettent une réversibilité complète dans le sens indiqué ci-dessus.

Ariane et Rosetta sont des systèmes de traduction automatique dans lesquels on trouve un certain degré de réversibilité.

Ariane. Dans l'approche Ariane[20], la première phase de la réalisation des programmes d'analyse et de génération pour un langue (ou sous-langue) consiste à décrire (dans un formalisme adapté) une spécification linguistique déclarative, dite "grammaire statique". Cette grammaire statique sert de structure de base pour la dérivation manuelle de deux grammaires d'analyse et de génération dites "dynamiques". Elles sont toutes deux exprimées dans le langage de programmation linguistique ROBRA (un langage spécialisé permettant de spécifier des transductions d'arborescences) (Boitet et Nedobejkine 1981). Bien que les deux grammaires dynamiques réflètent de près la spécification statique, leur équivalence stricte est probablement difficile à garantir en pratique.

Rosetta. Dans l'approche Rosetta, une structure sémantique revêt la forme d'un "arbre de dérivation sémantique", dont les noeuds sont étiquetés par des "règles sémantiques". L'arbre de dérivation sémantique sert de base à la construction compositionnelle des structures syntaxiques de surface (appelées *S-trees*), réalisée au moyen d'une fonction dite "fonction compositionnelle"[21]. Il incombe au linguiste de

---

20 Développée par le GETA (Groupe d'Etudes pour la Traduction Automatique, Grenoble).

21 Cette approche par "syntaxe compositionnelle" peut être considérée comme duale de l'approche plus courante par "sémantique compositionnelle".

s'assurer que cette fonction compositionnelle respecte la "condition de réversibilité", c.à.d. qu'il existe une "fonction analytique", inverse de la fonction compositionnelle. Cette fonction analytique doit être définie manuellement "en tandem" avec la fonction compositionnelle.
Dans ces conditions, il devient possible soit de partir d'une structure sémantique et d'obtenir un S-tree pour la phrase (mode de génération), ou inversement, de partir d'un S-tree et d'obtenir une structure sémantique (mode d'analyse), de façon garantie.
Il semble toutefois que cette approche suppose une certaine assymétrie entre l'analyse et la génération, ce qui est probablement une source de difficultés. La génération est immédiate: à partir de la structure sémantique, un S-tree est obtenu, et, à partir de ce *S-tree*, la chaîne phrastique est produite par simple énumération des feuilles de ce S-tree. En revanche, dans le mode analyse, il serait totalement inefficace, partant de la chaîne phrastique, de faire une hypothèse aveugle de S-tree sans aucun moyen de vérifier sa bonne formation syntaxique, et ensuite de tenter d'appliquer les fonctions analytiques jusqu'à tomber sur une structure sémantique. Voilà pourquoi il est nécessaire de compléter le système avec une "composante de syntaxe de surface", qui limite considérablement les S-trees dont on peut faire l'hypothèse sur la base d'une chaîne phrastique.

Parmi les exemples d'approche *automatique* de la réversibilité, on peut citer CRITTER (abordé dans la présente communication), l'approche discutée dans (Shieber 1988) et le modèle BUG (Van Noord 1988).[22]

Toutes ces approches ont les traits communs suivants: i) elles visent la réversibilité complète et ii) elles se fondent sur des formalismes qui reposent sur l'unification – soit grammaires logiques, soit grammaires d'unification.

Shieber 1988. L'approche de (Shieber 1988) diffère de la nôtre en ce qu'elle utilise un mode de déduction de type Earley à la fois pour

[22] Récemment, un article (Shieber, Van Noord and al., 1989) est apparu, qui développe ces deux approches.

l'analyse et pour la génération. Dans le mode de génération, un "filtre sémantique" est utilisé pour restreindre les lemmes qui sont ajoutés à l'agenda. Pour que ce filtre sémantique puisse fonctionner, il est nécessaire que la grammaire respecte un "critère de monotonicité sémantique", c'est-à-dire que la sémantique d'une sous-phrase *SP* de la phrase *P* soit "subsumée par une portion de la sémantique de *P*". Sous cette condition, un lemme ' A <-- $B_1 \ldots B_n$ ' ne peut faire partie d'une preuve réussie du but principal *S* que lorsque la sémantique (partiellement instanciée) de A subsume une partie de la sémantique de *S*. C'est cette propriété que le filtre sémantique vérifie, et qui permet que seuls les lemmes "prometteurs" soient ajoutés à l'agenda.
Aux dires mêmes de l'auteur, la monotonicité sémantique est peut-être une condition trop contraignante pour les grammaires réelles et "la découverte d'une contrainte moins rigide est un important objectif de recherche". A l'opposé, dans une approche par déduction régressive, une telle contrainte n'est pas une condition nécessaire au fonctionnement de la génération (voyez la section 7). En revanche, l'analyse par déduction régressive possède des inconvénients que n'a pas une approche progressive (ou assertive), en particulier le problème de récursion à gauche (voir section 5).

BUG. Le modèle BUG (Van Noord 1989) cherche à résoudre le problème de la génération (l'article ne traite pas de l'analyse) dans une approche par déduction progressive, inspirée de l'algorithme *left-corner* utilisé (dans un cadre d'analyse) par (Matsumoto et. al., 1983), (voir aussi Pereira et Shieber (1987)). L'algorithme fait appel à un filtre sémantique 'link', qui est une relation précompilée d'accessibilité sémantique, reliant la sémantique d'un noeud à la sémantique de sa "tête sémantique" (et transitivement de la tête sémantique de cette tête sémantique, etc...). Cette relation permet d'éliminer par filtrage les constituants dont la sémantique n'est pas liée à la sémantique (connue au préalable) de leur projection maximale, et par conséquent permet d'établir des prédictions fortes concernant la tête lexicale d'une phrase. Par hypothèse, la grammaire doit obéir à la "condition de tête sémantique", assez semblable aux conditions 2) et 3) de la section 6, et

qui n'implique pas de monotonicité sémantique, ce qui permet le traitement de certains cas d'idiomes.

9. Conclusions

Nous avons décrit une solution pratique au problème computationnel de la *bidirectionnalité des grammaires*.

Une grammaire est formalisée comme une théorie logique, de laquelle analyse et génération sont vues comme des procédures de preuve spécialisées de type régressif. Le processus de déduction est rendu efficace dans chacun des deux modes grâce à l'utilisation d'*annotations de contrôle spécifiques au mode*.

La bidirectionnalité procure des avantages théoriques et pratiques importants:

- Au plan *théorique*, l'hypothèse qu'une *compétence grammaticale commune* est sous-jacente à l'analyse et à la génération est vraisemblable. La contrainte de bidirectionnalité incite le linguiste à une plus grande rigueur dans son travail de spécification de cette compétence. La plupart des *analyseurs* unidirectionnels sont associés à des grammaires qui, prises en elles-mêmes, sont sérieusement *surgénératrices*. Et cela, parce que le rédacteur de la grammaire peut supposer, parfois de façon préjudiciable, que l'entrée ne contient que des phrases bien formées. De façon duale, la plupart des *générateurs* unidirectionnels sont associés à des grammaires qui sont sérieusement *sous-génératrices*. Le rédacteur de la grammaire sélectionne arbitrairement un sous-ensemble restreint de la langue. A l'opposé, comme il est impossible d'avoir une *même* grammaire qui tout à la fois surgénère et sous-génère, une grammaire bidirectionnelle devra produire le *bon* ensemble de phrases.

- Au plan *pratique, la bidirectionnalité est un outil facilitant l'obtention d'une grammaire plus précise*. Sans elle, il est bien plus difficile d'apprécier de quelle manière un analyseur unidirectionnel est sur-permissif: en effet, il

devient alors nécessaire de vérifier sur un grand nombre de phrases non-grammaticales qu'elles échouent bien en analyse, une entreprise fort peu naturelle! Il est tout aussi difficile de tester un générateur unidirectionnel par lui-même: il faudrait entrer manuellement des structures sémantiques compliquées. Cependant, quand à partir du résultat d'une analyse on peut générer toutes les paraphrases admises par la grammaire, les lacunes sautent aux yeux. Une grammaire bidirectionnelle est donc en un certain sens *auto-correctrice* [self-debugging].

Nous insistons sur le fait que les techniques présentées ci-dessus ont une importance pratique. Elles ont fait l'objet de tests exhaustifs, et les résultats ont été probants. Les annotations de contrôle ont permis le développement de grammaires réversibles de taille honorable pour le français et pour l'anglais. Ces grammaires sont utilisées dans le système CRITTER, un système expérimental de traduction automatique qui traduit de l'anglais au français et du français à l'anglais des rapports des marchés agricoles (Isabelle et al. 1988). Le système est encore incomplet, mais la couverture des grammaires est déjà assez large: verbes de contrôle et de montée, passivisation, movement datif [dative movement], insertion du "there' [there insertion], ainsi que certains types de relatives (réduites ou non réduites)[23], certains types de coordination, divers types de modification, etc... Les dictionnaires contiennent environ 1000 entrées lexicales chacun. CRITTER est implanté en Quintus Prolog. Les temps de traduction types sont inférieurs à un mot par seconde sur un SUN 3/60, et pour des phrases types, les temps d'analyse et de génération sont presque égaux.

Remerciements.

Nous remercions Elliot Macklovitch de ses suggestions et commentaires, et Jean-Luc Cochard, François Perreault et Michel Simard de leur apport au travail ici décrit.

[23] Voir la discussion de l'extraposition en section 7.

References.

Appelt, Douglas E. 1987. Bidirectional Grammars and the Design of Natural Language Generation Systems. In *Theoretical Issues in Natural Language Processing 3 (TINLAP-3)*, 206-12. Las Cruces, NM: Association for Computational Linguistics, January.

Beaven, John L. and Whitelock, Peter. 1988. Machine Translation using Isomorphic UCGs. In *Proceedings of the 12th International Confererence on Computational Linguistics*, 32-35. Budapest, August.

Boitet, Christian, and Nedobejkine, Nicolas. 1981. Recent developments in Russian-French machine translation at Grenoble. *Linguistics* 19 (3/4): 199-271.

Colmerauer, Alain. 1971. *Les systèmes-Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur* . Montreal: Université de Montréal, Projet TAUM, Rapport de Recherche.

Colmerauer, Alain. 1982. *PROLOG II : Manuel de référence et modèle théorique* . Marseilles, France: Faculté des Sciences de Luminy, Groupe d'Intelligence Artificielle.

Dymetman, Marc and Isabelle, Pierre. 1988. Reversible Logic Grammars for Machine Translation. In *Proceedings of the Second International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*. Pittsburgh: Carnegie Mellon University, June.

Dymetman, Marc. 1989. *RATP: Un nouveau formalisme de la famille des grammaires d'unification*. Laval, Québec: Ministère des Communications Canada, Centre Canadien de Recherches sur l'Informatisation du Travail, Co 28-1/32-1989F.

Earley, J. 1970. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM* 13:2 (February): 94-102.

Hasida, K. and Isizaki, S.. 1987. Dependency propagation : a unified theory of sentence comprehension and generation. In *Proceedings of AAAI*, 664-670. Seattle, WA, July.

Isabelle, Pierre, Dymetman, Marc and Macklovitch, Elliott. 1988. CRITTER: a Translation System for Agricultural Market Reports. In *Proceedings of the 12th International Confererence on Computational Linguistics*, 261-266. Budapest, August.

Jacobs, P. 1985. PHRED: a generator for natural language interfaces. *Computational Linguistics* 11:4 (October-December): 219-42.

Kay, Martin. 1975. Syntactic Processing and Functional Sentence Perspective. In *Theoretical Issues in Natural Language Processing (TINLAP), Supplement to the Proceedings*, 12-15. Cambridge, MA: Association for Computational Linguistics, June.

Kay, Martin. 1980. Algorithm Schemata and Data Structures in Syntactic Processing. In *Readings in Natural Language Processing*, eds. Grosz B., K. Sparck Jones and B. Lynn Webber (1986). Los Altos, CA: Morgan Kaufmann.

Kowalski, R.A. 1979a. Algorithm = Logic + Control. *Communications of the ACM* 22 (July): 424-436.

Kowalski, R.A. 1979b. Logic for Problem Solving. Reading, New York, NY: North-Holland.

Landsbergen, J. 1987. *Montague Grammar and Machine Translation*. Eindhoven, Holland: Philips Research M.S. 14.026.

Matsumoto Y., Tanaka, H., Hirikawa, H., Miyoshi, H. and Yasukawa, H. 1983. BUP: a bottom-up parser embedded in Prolog. *New Generation Computing* 1:2, 145-158.

Pereira, Fernando C. N. 1981. Extraposition Grammars. *Computational Linguistics* 7:4, 243-56.

Pereira, Fernando C. N. and Warren, David H. D. 1980. Definite Clause Grammars for Language Analysis. *Artificial Intelligence*:13, 231-78.

Pereira, Fernando C. N. and David Warren, David H. D. 1983. Parsing as Deduction. In *Proceedings of the 21th Annual Meeting of the Association for Computational Linguistics*, 137-44. Cambridge, MA, June.

Pereira, Fernando C. N. and Shieber, Stuart M. 1987. *Prolog and Natural Language Analysis*. CSLI lecture note No. 10. Stanford, CA: Center for the Study of Language and Information.

Pollard, Carl and Ivan A. Sag. 1987. *Information-Based Syntax and Semantics*, vol. 1. CSLI lecture note No. 13. Stanford, CA: Center for the Study of Language and Information.

Sells, Peter. 1985. *Lectures on Contemporary Linguistic Theories*. CSLI lecture note No. 3. Stanford, CA: Center for the Study of Language and Information.

Shieber, Stuart M. 1986. A Uniform Architecture for Parsing and Generation. In *Proceedings of the 12th International Confererence on Computational Linguistics*, 614-19. Budapest, August.

Shieber, Stuart M. 1986. *An Introduction to Unification-Based Approaches to Grammar*. CSLI lecture note No. 4. Stanford, CA: Center for the Study of Language and Information.

Shieber, Stuart, M., van Noord, Gertjan, Moore, Robert and Pereira, Fernando. 1989. A Semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, 7-17. Vancouver, BC, Canada, June.

Strzalkowski, Tomek. 1989. *Automated Inversion of a Unification Parser into a Unification Generator*. New York: Courant Institute of Mathematical Sciences, Department of Computer Science, Technical Report 465.

Van Noord, Jan. 1989. *BUG: A Directed Bottom-up Generator for Unification Based Formalisms*. Working Papers in Natural Language Processing No. 4. Utrecht, Holland: RUU, Department of Linguistics.

Vauquois, Bernard and Chappuy, Sylviane. 1985. Static Grammars. A Formalism for the Description of Linguistic Models. In *Proceedings of the International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*, 298-322. Hamilton, NY: Colgate University, August.

Wilensky, R. and Arens, Y.1980. *PHRAN: a Knowledge-Based Approach to Natural Language Analysis*. Berkeley, CA: University of California, Electronics Research Laboratory Memorandum #UCB/ERL M80/34.

QUEEN P 308 .D96514 1990 c.2
Dymetman, Marc
Bidirectionnalité des gramm

DYMETMAN, MARC
--Bidirectionnalité des grammaires
par déduction arrière contrôlée

P
308
D965f
1990
c.2

DATE DUE

Pour plus de détails, veuillez communiquer avec :

Le Centre canadien de recherche sur l'informatisation du travail
1575, boulevard Chomedey
Laval (Québec)
H7V 2X2
(514) 682-3400

For more information, please contact:

Canadian Workplace Automation Research Centre
1575 Chomedey Blvd.
Laval, Quebec
H7V 2X2
(514) 682-3400