IDIAP
Rapport technique

Un environnement d'analyse linguistique robuste

CPD, version 1.7

Jean-Luc Cochard

Octobre 1992

INSTITUT DALLE MOLLE D'INTELLIGENCE ARTIFICIELLE PERCEPTIVE
CASE POSTALE 609 - 1920 MARTIGNY - VALAIS - SUISSE
TELEPHONE : ++41 26 22.76.64 - FAX : ++41 26 22.78.18
E-MAIL : IDIAP@IDIAP.CH

Numéro : 92-03

Un environnement d'analyse linguistique robuste
CPD, version 1.7

Jean-Luc Cochard

Institut Dalle Molle d'Intelligence Artificielle Perceptive (IDIAP)
Case postale 609, CH-1920 Martigny, Suisse
Adresse électronique : `cochard@idiap.ch`

Résumé

Ce document présente un environnement de développement de grammaires construit dans le cadre du projet *Specification and Prototyping of a System for the Intelligent Management of Information*[1]. Cet environnement se nomme CPD ("Chart Parsing and Debugging") et permet la formulation, la compilation et la mise au point de grammaires formelles pour une langue naturelle. La stratégie fixe d'analyse adoptée pour le projet est une analyse ascendante et parallèle.

Cet environnement construit autour de SICStus-Prolog, offre un affichage pseudo-graphique des résultats d'analyse et met à disposition des outils de suivi de l'évolution du chart depuis son initialisation jusqu'à l'obtention des résultats finals.

Mots-clé : programmation logique, Prolog, traitement de la langue naturelle, analyse ascendante, environnements de mise au point.

[1] Projet № 4023-26996, PNR 23, FNSRS

Table des Matières

1 Introduction **3**

2 Le formalisme de la grammaire **3**
2.1 L'entête . . . 3
2.2 La syntaxe d'une règle . . . 4
2.3 Le post-scriptum . . . 6
2.4 La mise en forme de la grammaire linguistique . . . 6

3 Les principes de génération du chart **6**
3.1 Le chart, les arcs du chart . . . 6
3.2 Le processus d'analyse . . . 7

4 Le mode d'emploi de CPD **10**
4.1 Le chargement des grammaires et le prédicat d'analyse `parse/3` . . . 10
4.2 Le mode *de base* . . . 10
4.3 Les "événements" du processus d'analyse . . . 13
4.3.1 Le format d'un arc . . . 13
4.3.2 Le format d'une *création d'arc* . . . 14
4.3.3 Le format d'un *échec de création* . . . 14
4.4 Les prédicats de configuration de CPD . . . 14
4.4.1 La trace exhaustive : `<>trace/0`, `<>notrace/0` . . . 15
4.4.2 La trace sélective : `<>debug/0`, `<>nodebug/0` . . . 15
4.4.3 Les points d'arrêt et les points de contrôle : `<>spy/1`, `<>nospy/1`, `<>nospyall/0` . . . 16
4.4.4 L'état courant de CPD : `<>debugging/0` . . . 17
4.4.5 L'analyse détaillée des résultats : `<>spy result` . . . 17
4.5 Les contextes d'interaction de CPD . . . 18
4.5.1 Le contexte d'interaction sur les événements . . . 18
4.5.2 Le contexte d'interaction sur les entités . . . 19
4.5.3 Le contexte d'interaction sur les structures d'attributs . . . 20

5 Conclusion **21**

6 Résumé des commandes **23**

1 Introduction

Il existe, dans la littérature associée à Prolog, plusieurs publications traitant du "chart parsing". Cette technique est née des travaux de J. Earley [2] et de M. Kay [3] et permet, dans son utilisation en linguistique, de conserver une trace de toutes les étapes de vérification d'un ensemble de règles sur un texte donné. C'est cette caractéristique qui donne cette qualité de *robustesse* au système car, en tout temps, le système génère un résultat même partiel, ce qui n'est pas le cas avec la technique standard de résolution de Prolog.

La technique de "chart parsing" a été mise en œuvre en Prolog depuis quelques années déjà. Il existe au moins deux publications incontournables sur le sujet : celle de P. Ross [5] et celle de G. Mellish & C. Gazdar [4]. Dans l'ensemble, les questions auxquelles répondent ces documents sont : *Comment implanter un "chart" en Prolog et comment manipuler efficacement cette structure de données en présence de différentes stratégies d'analyse ?* Ce sont évidemment là des problèmes majeurs qu'il faut résoudre avant toute autre chose. Mais une fois ce stade atteint, la technique de "chart parsing" pose de gros problèmes d'utilisation. En effet, il est extrêmement difficile de suivre les étapes d'une vérification des règles sur la simple base des différents états du chart — ce qui reviendrait, par ailleurs, à suivre l'exécution d'un programme en C sur l'unique base de l'évolution de la pile d'exécution et de l'état de la mémoire. Et pourtant, on ne peut pas supposer qu'un ensemble de règles soit exempt de "bugs".

C'est donc pour des raisons analogues à celles qui ont motivées le développement d'outils de mise au point pour C, Lisp ou Prolog, que nous nous sommes intéressés au problème de la mise au point de grammaires dans un environnement d'analyse ascendante et parallèle. Pour atteindre ce but de manière satisfaisante, il est indispensable de définir au préalable un modèle du chart parsing, comparable au modèle des boîtes — la "Procedure box" inventé par Lawrence Byrd — est le modèle classique du mécanisme de résolution de Prolog.

Pour le projet *Specification and prototyping of a System for the Intelligent Management of Information*[2], nous avons construit à la fois une infrastructure de "chart parsing" en Prolog, proposé un modèle du fonctionnement du "chart parser" et implanté un environnement de mise au point s'appuyant sur ce modèle facilitant le travail du linguiste qui écrit et teste ses grammaires.

Cet environnement d'analyse linguistique se nomme CPD — "chart parsing and debugging". Il est basé sur le "chart parser" proposé par P. Ross dans [5]. Quelques adaptations et simplifications y ont été apportées afin de tenir compte des besoins spécifiques d'une analyse linguistique ascendante. La particularité de CPD réside principalement dans l'existence d'outils de mise au point spécialisés.

La suite de ce rapport est organisé de la manière suivante : la section 2 décrit la syntaxe d'une grammaire linguistique qui constitue le programme accepté par CPD; la section 3 présente les principes de la construction d'un chart ainsi qu'un modèle de comportement du système et la section 4 décrit en détail toutes les commandes de l'environnement de mise au point de CPD.

2 Le formalisme de la grammaire

Un "programme" pour CPD — une "grammaire linguistique" dans la suite de ce rapport — contient des définitions de *règles syntaxiques* et des définitions de clauses Prolog. Une règle syntaxique est un cas particulier de clause Prolog, utilisant l'opérateur `--->/2` comme foncteur. En plus de la syntaxe de Prolog, CPD impose des règles spécifiques de rédaction d'une grammaire linguistique qui portent sur trois parties du fichier : le préambule, la règle et le post-scriptum.

2.1 L'entête

Une grammaire linguistique valide pour CPD doit débuter par l'entête reproduit dans la Figure 1). Cette séquence constitué d'un but et d'un fait est très inélégante et devra faire place, dans une version

[2] Projet N° 4023-26996, PNR 23, FNSRS.

ultérieure, à une solution plus simple. Les raisons de cette séquence sont abordées dans la section 2.4, où nous expliquons les principes de mise en forme des grammaires linguistiques.

```
:- init_rule_compilation.
grammar.
```

Figure 1 Entête indispensable au bon fonctionnement de CPD : le but ``:- init_rule_compilation.'' sert à initialiser des compteurs internes de CPD et la définition de `grammar/0` sert de définition témoin pour le chargement de règles.

2.2 La syntaxe d'une règle

La forme générale d'une règle est décrite dans la Figure 2, conformément à la notation adoptée pour décrire Prolog dans la documentation de Sicstus-Prolog [1].

règle	$\longrightarrow$	*membre-gauche* `'--->'` *membre-droit* `'.'`
membre-gauche	$\longrightarrow$	*non-terminal*
membre-droit	$\longrightarrow$	*des-sous-problèmes*
des-sous-problèmes	$\longrightarrow$	*sous-problème*
	$\mid$	*sous-problème* `','` *des-sous-problèmes*
sous-problème	$\longrightarrow$	*non-terminal*
	$\mid$	`'{'` *des-buts-Prolog* `'}'`
des-buts-Prolog	$\longrightarrow$	*but-Prolog*
	$\mid$	*but-Prolog* `','` *des-buts-Prolog*
non-terminal	$\longrightarrow$	*foncteur* `'('` *poids* `','` *structure-d'attributs* `')'`
but-Prolog	$\longrightarrow$	*valeur-d'attribut* `'=..='` *valeur-d'attribut*
	$\mid$	*but*
valeur-d'attribut	$\longrightarrow$	*structure-d'attributs* `'..'` *attribut*
	$\mid$	*valeur*

Les définitions des non-terminaux suivants s'appuient sur la syntaxe de Prolog

but	$\longrightarrow$	*goal*
foncteur	$\longrightarrow$	*functor*
valeur	$\longrightarrow$	*name*
	$\mid$	*variable*
poids	$\longrightarrow$	*variable*
	$\mid$	*integer*
structure-d'attributs	$\longrightarrow$	*variable*
attribut	$\longrightarrow$	*word*

Figure 2 Syntaxe d'une règle linguistique de CPD.

La forme des constituants d'une règle mérite quelques explications complémentaires en particulier sur le choix de la représentation des non-terminaux. Un *non-terminal* est défini comme un terme à deux arguments. Le *foncteur* de ce terme détermine le nom d'une règle de la grammaire. Plusieurs règles peuvent avoir le même nom; elles sont considérées comme des alternatives numérotées de manière interne à partir de "1". Le premier argument du terme, le *poids*, est automatiquement unifié avec l'attribut `poids` de la *structure-d'attributs*. Le deuxième argument est précisément la *structure-d'attributs*. Cette unification entre un argument du terme *non-terminal* et un constituant d'un autre argument sert uniquement à amener "à la surface" du terme le poids qui joue un rôle essentiel dans l'ordonnancement des arcs de l'agenda (cf. section 3.2, pour la définition de la notion d'agenda). Ainsi lorsque le terme est affiché par l'interface de mise au point, le poids apparaît en premier lieu. Il n'existe cependant aucune raison technique de pratiquer de la sorte et ce premier argument d'un *non-terminal* va disparaître dans une prochaine version du système.

Pratiquement, la définition de la *règle* permet de définir deux groupes distincts de règles qui peuvent cohabiter dans une grammaire : des règles *normales* et des règles *vides*. Une règle normale est une règle dont le membre droit contient des non-terminaux (cf. Figure 3).

```
% COMPLEMENTS optionnels : un groupe prépositionnel quelconque
%                          suivi d'un groupe prépositionnel
%                          débutant par "par".

cnp(V,CNp) ---> opt_adjp(V0,Adj0),
                noun(V1,N),
                opt_adjp(V2,Adj2),
                { N..form_cit =..= NN,
                  relational(NN,1,Prep,_,_)},
                opt_subcat_pp(V3,Pp1),
                opt_subcat_pp(V4,Pp2),
                { first_daughter(Pp1,Prep),
                  first_daughter(Pp2,par),
                  CNp..agreement =..= N..agreement,
                  agreement(N, Adj0, Adj2),
                  CNp..cat =..= cnp,
                  CNp..daughters =..= [Adj0,N,Adj2,Pp1,Pp2],
                  CNp..head =..= N..head,
                  compute(V0,V1,V2,V3,V4,0,V)}.
```

Figure 3 Exemple d'une règle typique de description d'un groupe nominal complexe : cet exemple illustre les possibilités de construction du membre droit d'une règle avec des non-terminaux comme "`opt_adjp(V0,Adj0)`" ou "`noun(V1,N)`", des unifications sur les structures d'attributs comme "`N..form_cit =..= NN`" et aussi des buts Prolog habituels comme "`first_daugthers(Pp1,Prep)`" et "`compute(...)`".

Une règle vide est un cas particulier de règle, utile lorsqu'il faut définir des constituants optionnels. La caractéristique d'une telle règle est qu'elle ne contient aucun non-terminal dans le membre droit mais uniquement des buts Prolog qui servent normalement à décrire la structure d'attributs résultante (cf. Figure 4).

```
opt_subcat_pp(0,Pp) ---> { Pp..cat =..= []}.
opt_subcat_pp(V,Pp) ---> pp(V0,Pp),
                         { Pp..daughters =..= [Prep|_],
                           Prep..form_cit =..= PrepForm,
                           preference(opt_subcat,PrepForm,Pf),
                           compute(V0,Pf,V)}.
```

Figure 4 Exemple de description d'un groupe prépositionnel optionnel à l'aide de deux instances de la même règle : la première étant une règle vide; la deuxième, une règle normale.

La caractérisation de la structure d'attributs associée à une règle vide est laissée à la responsabilité de l'utilisateur. Dans la Figure 4, le rédacteur de la grammaire a choisi d'unifier la catégorie morpho-syntaxique avec la valeur "`[]`", liste vide.

2.3 Le post-scriptum

Une grammaire linguistique valide, pour CPD, doit se terminer par le post-scriptum reproduit dans la Figure 5. Probablement, dans un avenir proche, ce but sera remplacé par une définition du prédicat `end_grammar/0`, ce qui rendrait la syntaxe de la grammaire plus simple et plus symétrique.

```
:- end_rule_compilation.
```

Figure 5 Code devant obligatoirement terminer la série des définitions de règles syntaxiques. Ce but indique la fin de la grammaire et produit une mise en forme optimale des règles.

2.4 La mise en forme de la grammaire linguistique

Une grammaire linguistique est un programme Prolog particulier (cf. section 2). Ce programme peut être chargé dans un état sauvé qui contient l'environnement CPD de manière tout à fait transparente pour l'utilisateur à l'aide des prédicats `consult/1` ou `compile/1`. Lors du chargement, CPD fait subir quelques transformations à ce programme.

Premièrement, les différentes instances d'une même règle linguistique — les définitions de règles ayant le même foncteur au membre gauche — sont explicitement numérotées. Cette numérotation permet de fournir des informations plus précises lors de la trace d'exécution (cf. section 4.3). Le but `":- init_rule_compilation."` sert à initialiser cette table de compteurs d'instances de règles.

Deuxièmement, lorsque toutes les règles sont chargées, c'est-à-dire lorsque Prolog exécute le but `":- end_rule_compilation."`, toutes les règles sont groupées sous forme d'une liste. Et c'est cette liste qui est finalement mémorisée, en tant qu'argument d'un prédicat spécial, `chart_rules/1`.

Troisièmement, les instances de règles peuvent être traitées par un évaluateur partiel, notamment les unifications sur les prédicats d'accès. Lorsque de telles évaluations partielles sont effectuées, le nombre de buts du membre droit de la règle est diminué et les arguments du membre gauche deviennent partiellement instanciés. Ce traitement améliore les performances du programme, soit la grammaire linguistique partiellement évaluée, en transformant la grammaire source en une grammaire équivalente plus compacte ayant la même sémantique mais plus la même séquence d'instructions. L'utilisateur ne peut donc pas se référer à sa grammaire source pour suivre l'évolution de son programme lorsque celui-ci a été partiellement évalué.

Nous avons donc mis en place un mécanisme de traitement différencié des grammaires "consultées" et des grammaires "compilées". Lorsqu'une grammaire est consultée, l'évaluation partielle est mise hors service ce qui garantit la conservation de la forme de la grammaire lors du chargement de celle-ci. Lorsqu'une grammaire est compilée, tout est mis en œuvre pour améliorer l'efficacité de son exécution. La grammaire est donc partiellement évaluée avec les avantages et les conséquences que l'on sait. La présence de la clause `grammar/0` dans l'entête du fichier joue le rôle de témoin pour connaître les intentions de l'utilisateur, à savoir s'il a demandé de consulter ou de compiler sa grammaire.

3 Les principes de génération du chart

Cette section décrit les principes de génération du chart avec l'objectif de définir un modèle d'analyse aussi simple et complet que possible. Ce modèle doit servir à comprendre les informations fournies et les options offertes par CPD.

3.1 Le chart, les arcs du chart

Un *chart* est un graphe constitué d'arcs orientés et étiquetés. Une telle structure de données, en relation avec un processus d'analyse, sert à garder une trace de toutes les étapes de vérification de l'ensemble des règles sur un texte donné. Chaque arc représente un état de vérification d'une règle sur une portion de texte. L'arc est étiqueté à l'aide du terme qui constitue le membre gauche d'une règle. Le "niveau" de vérification atteint par une règle est déterminé par la liste des *sous-problèmes* (cf. Figure 2) qui doivent encore être résolus. Une règle est complétement vérifiée lorsque cette liste est vide.

Chaque arc du chart est donc intimement lié à une règle de la grammaire. Le contenu d'un arc du chart en termes d'informations qu'il doit mémoriser se résume de la manière suivante :

un sommet initial – l'identification[3] du sommet de départ de l'arc;

un sommet final – l'identification du sommet d'arrivée de l'arc; cette distinction entre les rôles des deux sommets permet de définir une direction;

un nom – ce nom[4] est le lien effectif existant entre la règle et l'arc; il s'agit pratiquement d'une copie du *non-terminal* qui constitue le membre gauche de la règle;

des hypothèses – la liste des *sous-problèmes* du membre droit de la règle qui doivent encore être résolus.

De fait, il existe deux catégories d'arcs : premièrement, les arcs dont la liste des hypothèses est vide — ce sont les *arcs solution* car ils détiennent l'information sur les solutions de l'analyse — et, deuxièmement, les arcs dont la liste des hypothèses n'est pas vide — ce sont les *arcs supposition* car ils représentent une règle partiellement vérifiée.

3.2 Le processus d'analyse

Le processus d'analyse se décompose en trois étapes bien distinctes : la création d'un chart initial à partir d'un texte, l'addition de nouveaux arcs par l'application des règles et la sélection des résultats lorsque la condition d'arrêt de la deuxième étape est remplie.

L'initialisation morphologique

La création d'un chart initial est une opération qui fait intervenir l'analyseur morphologique et qui nécessite que toutes les analyses morphologiques de tous les mots du texte soient générées. Un résultat morphologique est une structure d'attributs déterminée par les dictionnaires, les définitions des classes flexionnelles et des valeurs par défaut — actuellement les seules valeurs par défaut sont : `poids` qui prend la valeur `1`, `daughters` et `head` qui prennent la valeur `[]`. Afin de permettre l'application de règles sur le résultat de la morphologie, il faut convertir ces structures d'attributs en arcs solution du chart. L'étiquette de l'arc est construite à partir de la structure d'attributs de la manière suivante :

> À partir de la structure d'attributs SA, on construit un terme à deux arguments dont le foncteur est unifié avec la catégorie morpho-syntaxique (attribut `cat`) de SA, le premier argument est unifié avec le poids (attribut `poids`) de SA et le deuxième argument est la structure d'attributs SA complète. Les *hypothèses* sont unifiées avec la liste vide `[]`, puisqu'il s'agit, par définition, d'un arc solution.

La détermination des numéros des sommets initial et final d'un arc est relativement simple. Le numéro du sommet initial d'un arc X s'unifie avec le numéro du sommet final d'un arc Y si Y représente un mot qui précède directement X dans la phrase. Dans le cas d'une phrase comme "Rapport sur les résultats d'une enquête", nous obtenons l'organisation des arcs telle qu'illustrée dans la Figure 6. Comme l'objectif de l'illustration est d'expliquer la technique de numérotation des arcs et non la présentation exhaustive du contenu du chart à un certain moment, les noms des arcs ne sont pas affichés. Par contre, le texte à analyser figure en transparence sur le chart pour en faciliter la lecture. De plus les arcs ne sont pas fléchés mais leur orientation est définie conventionnellement de gauche à droite.

[3] L'identification des sommets est faite à l'aide de nombres entiers. On parlera donc généralement du "numéro du sommet".

[4] À ne pas confondre avec le nom d'une règle qui n'est que le foncteur du membre gauche d'une règle

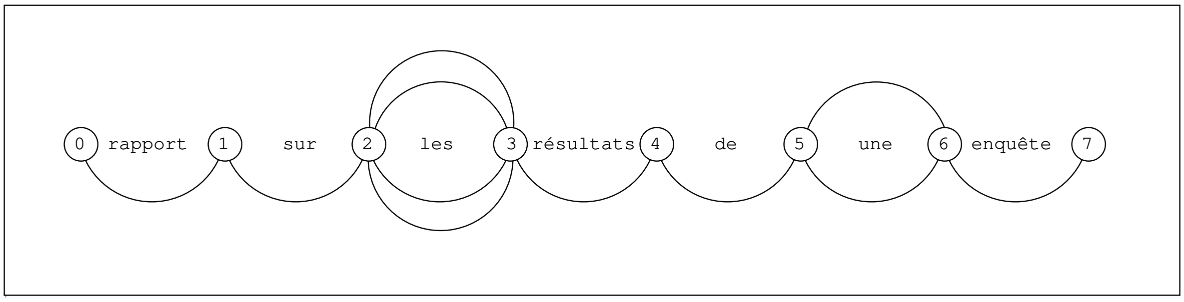

Figure 6 Représentation graphique du chart initial construit par CPD lors de l'exécution de `"parse(0, N, "Rapport sur les résultats d'une enquête").".` La valeur "0" du premier arc est déterminée par le 1er argument de `parse/3`.

La situation se complique légèrement lorsque le texte à analyser contient des contractions ambiguës comme "des" qui peut être soit la contraction de "de les", soit le pluriel de "un". L'algorithme de génération des arcs commence par construire les arcs n'impliquant pas de décomposition d'une contraction. Cette solution nous amène à des situations surprenantes avec des arcs dont le numéro du sommet initial est plus grand que celui du sommet final (cf. Figure 7).

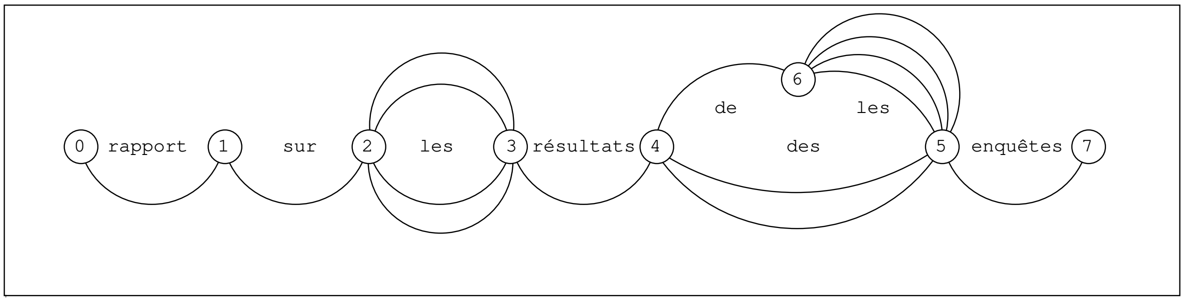

Figure 7 Représentation graphique du chart initial construit par CPD lors de l'exécution de `"parse(0, N, "Rapport sur les résultats des enquêtes").".`

L'initialisation syntaxique

Une deuxième initialisation fait suite à celle morphologique. Il s'agit de l'application des règles vides comme

```
opt_determiner(0, Det)  --> {Det..cat =..= []}.
```

Comme ces règles n'ont aucun prérequis — membre droit vide, ou du moins exempt de référence à d'autres non-terminaux — leur transcription en arcs du chart peut se faire immédiatement (cf. Figure 8). Les arcs engendrés par ces règles vides ont deux particularités. Premièrement, les sommets initial et final sont confondus car une règle vide ne recouvre aucune portion de texte. Deuxièmement, ces arcs sont génériques dans la mesure où ils peuvent se rattacher à n'importe quel sommet du chart. Pour mettre en place ce principe, la valeur de l'unique sommet est une variable non instanciée. Ainsi, pour chaque règle vide présente dans la grammaire, un seul exemplaire d'un tel arc est introduit dans le chart. Par ailleurs, ces arcs sont des arcs solution car tous les buts Prolog — entre {} — sont vérifiés automatiquement lors de la création de l'arc.

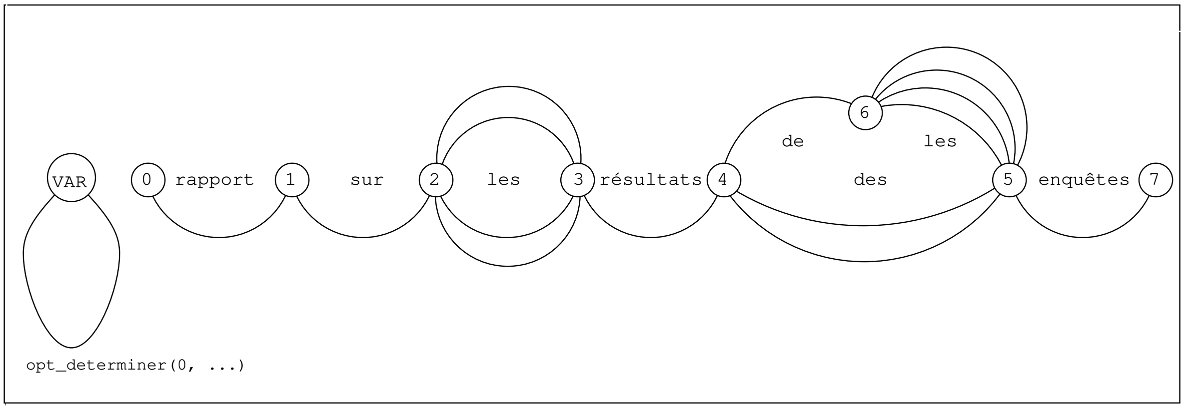

Figure 8 Reprise du chart initial de la Figure 7 assorti de l'arc généré par la règle `opt_determiner(0, Det) --> {Det..cat =..= []}`.

L'adjonction de nouveaux arcs

L'adjonction de nouveaux arcs dans le chart se fait par l'application de l'une ou de l'autre des deux règles suivantes. Pour simplifier l'exposé nous allons considérer, dans un premier temps, que les membres droits des règles ne contiennent que des non-terminaux.

Règle 1 : "nouvelle règle" – Soit une règle `nt0 ---> nt1, nt2` ...`ntn` et un arc solution entre X et Y ayant le nom `ntx` qui s'unifie avec `nt1`, alors un nouvel arc est créé entre X et Y, avec le nom `nt0` et une liste d'hypothèses `[nt2,` ...`ntn]`.

Règle 2 : "nouveau pas" – Soient deux arcs, un entre Y et Z ayant le nom `nt0` et une liste d'hypothèses `[nt1 | nts]` et un entre X et Y ayant le nom `ntx` qui s'unifie avec `nt1`, le second étant un arc solution, alors un nouvel arc est créé entre X et Z avec le nom `nt0` et la liste des hypothèses `nts`.

Maintenant, si nous voulons tenir compte des buts Prolog qui peuvent apparaître dans le membre droit, il faut compléter cette liste de règles en ajoutant une règle 2'.

Règle 2' : "nouveau pas bis" – Soit un arc entre X et Y ayant le nom `nt0` et une liste d'hypothèses `[buts1 | nts]`, si l'exécution de `buts1` réussit, alors un nouvel arc est créé entre X et Y avec le nom `nt0` et la liste des hypothèses `nts`.

Cet ensemble de règles garantit que toutes les solutions sont trouvées, pour autant que nous détections tous les candidats à l'application de l'une ou l'autre règle. Or, si nous implantons un mécanisme simple de recherche de candidats à l'aide d'un parcours systématique du chart, cette technique s'avère particulièrement inefficace. Nous avons donc adopté une solution décrite dans [5] et [4], utilisant un `agenda`. Un agenda est une file d'arcs. Ces arcs sont ordonnés selon le `poids` de la structure d'attributs — qui est aussi le premier argument du nom de l'arc. Cet agenda est alimenté par les arcs produits par l'application des règles 1, 2 et 2'. Le premier élément de la file est candidat à l'application des différentes règles. S'il s'agit d'un arc solution, le système tente de lui appliquer les règles 1 et 2. S'il s'agit d'un arc supposition, une des règles 2 et 2' sera appliquée suivant le contenu des hypothèses. Une fois l'arc traité par les différentes règles, il est versé au chart et l'opération recommence avec un nouvel arc.

La condition d'arrêt

Avec la technique de l'agenda, la condition d'arrêt est triviale. L'adjonction d'arcs est terminée lorsque l'agenda est vide, c'est-à-dire que tous les candidats ont été essayés sans qu'aucun nouvel arc n'ait été généré.

La sélection des résultats

Le chart contient deux catégories d'arcs, les arcs supposition et les arcs solution. Lorsque l'étape d'adjonction d'arcs prend fin, seuls les arcs solution ont un intérêt, puisqu'ils codent les résultats des règles vérifiées sur le texte. En général, cet ensemble d'arcs constitue une masse trop large d'informations pour être fournie telle quelle. Seuls les résultats les plus généraux ont de la valeur à la fin d'une analyse linguistique ce qui se traduit, en termes de chart et d'arcs, par les arcs qui recouvrent les plus grandes portions du chart.

À ce principe, il faut ajouter qu'une solution intéressante est constituée d'un arc ou d'une séquence d'arcs qui relient le sommet initial au sommet final du chart, soit, dans les illustrations graphiques, le sommet le plus à gauche au sommet le plus à droite.

4 Le mode d'emploi de CPD

Ce chapitre décrit l'ensemble des possibilités d'utilisation de CPD allant de l'analyse simple d'un texte accompagnée de l'affichage des résultats à l'utilisation d'outils plus sophistiqués du suivi de l'évolution du chart et du diagnostic d'erreurs dans les grammaires. L'ambition de CPD est de constituer une couche "étanche" au-dessus de Prolog qui évite à l'utilisateur d'un tel système de "chart parsing" d'être confronté directement avec Prolog et avec le code du moteur de construction des arcs. La première partie de ce chapitre (cf. sections 4.1 et 4.2) décrit les fonctions minimales de CPD. Dans la partie suivante, nous présentons les outils d'interaction avec CPD que sont les *prédicats de configuration de CPD* et les *contextes d'interaction* existant lors de l'exécution d'une analyse via `parse/3` (cf. sections 4.3, 4.4, 4.5).

4.1 Le chargement des grammaires et le prédicat d'analyse `parse/3`

La première opération à exécuter pour travailler avec l'environnement CPD, est le chargement d'une grammaire. Bien que les grammaires linguistiques pour CPD doivent respecter une syntaxe différente de celle des programmes Prolog (cf. chapitre 2), le chargement d'une grammaire se fait de manière tout à fait habituelle et transparente en utilisant les prédicats standards `consult/1` ou `compile/1` de Prolog.

Le prédicat principal de CPD `parse/3` permet l'exécution d'analyses linguistiques s'appuyant sur une grammaire précédemment chargée. Ce prédicat s'utilise de la manière suivante :

`parse(+`*Initial*`, -`*Final*`, +`*Texte*`)`

Initial Un nombre entier qui est interprété comme étant le numéro du sommet initial du chart.

Final Un nombre entier qui est interprété comme 'tant le numéro du sommet final.

Texte Une chaîne de caractères qui représente le texte à analyser. Aucune démarcation particulière des fins de mots autre que les espaces et la ponctuation n'est nécessaire au bon fonctionnement de ce prédicat. Il supporte aussi un texte qui s'étend sur plusieurs lignes.

Remarque Afin d'éviter toute confusion sur la notion de sommet dans le contexte du chart ou d'un arc, nous dirons qu'un arc a un *sommet gauche* et un *sommet droit* alors que le chart a un *sommet initial* — le sommet "le plus à gauche" — et un *sommet final* — le sommet "le plus à droite".

4.2 Le mode *de base*

CPD, comme Prolog d'ailleurs, dont nous nous sommes inspirés pour construire cet environnement, fonctionne selon plusieurs modes : un mode *de base* qui est le mode disponible lors du chargement du système, un mode *de trace exhaustive* des événements et un mode *de trace sélective* de ces mêmes événements. Nous allons décrire dans cette section le fonctionnement du mode *de base*.

Une séquence typique d'exécution de **parse/3** dans le mode *de base* de CPD se présente de la manière suivante :

```
SICStus 0.7 #6: Fri Oct 25 13:50:06 MET 1991

Projet des Archives: CHART PARSER V1.4 ++++++++
-------------------


yes
| ?- compile(fgram).
{compiling /home/arolla/cochard/ARCHIVES/Syntax/fgram.pl...}
{/home/arolla/cochard/ARCHIVES/Syntax/fgram.pl compiled, 870 msec 11484 bytes}

yes
| ?- parse(0, _, "Utilisation de la carte").

Parsing of :
(0) "Utilisation de la carte" (4)
1 constituent was recognized.
Parse 1 of 2, from 0 to 4
np
 |-- determiner()
 `-- cnp
      |-- noun(utilisation)
      |-- pp
      |    |-- prep(de)
      |    `-- np
      |         |-- determiner(le)
      |         `-- cnp
      |              `-- noun(carte)
      `-- pp()
           `-- _28447(par)
Parse 2 of 2, from 0 to 4
cnp
 |-- noun(utilisation)
 |-- pp
 |    |-- prep(de)
 |    `-- np
 |         |-- determiner(le)
 |         `-- cnp
 |              `-- noun(carte)
 `-- pp()
      `-- _17651(par)

yes
| ?-
```

L'affichage automatique des résultats par CPD fournit deux types d'informations : premièrement, une indication des sommets initial et final du chart; deuxièmement, les structures d'attributs les plus générales vues sous l'angle d'un arbre de dépendances syntaxiques.

L'avantage d'une analyse utilisant un chart avec la stratégie ascendante adoptée dans CPD est que l'analyse n'échoue jamais pour autant que la morphologie réussisse à initialiser correctement le chart. Cependant l'état final du chart ne contient pas toujours un résultat idéal, c'est-à-dire un ou plusieurs arcs dont les sommets gauche et droit s'unifient avec les sommets initial et final du chart. Dans ces conditions, il est nécessaire d'ordonner les résultats selon un critère de *généralité*.

Définition 1 (Relation d'ordre partiel des résultats) *Un* résultat valide *sur un chart est un chemin composé d'arcs solution reliant le sommet initial au sommet final du chart. Les résultats valides forment un ensemble sur lequel existe une relation d'ordre partiel nommée* "plus général que"*. Un résultat R_1 est* plus général qu*'un résultat R_2 si R_1 est constitué de moins d'arcs que R_2.*

En se basant sur cette définition, nous avons adopté la stratégie d'affichage des résultats qui consiste à n'afficher que les résultats appartenant au niveau *le plus général* atteint par l'analyse. CPD indique, premièrement, le niveau de généralité atteint : `"1 constituent was recognized."` — ce qui signifie que le chemin est constitué d'un seul arc — et affiche, deuxièmement, l'ensemble des arcs appartenant aux résultats retenus. Les arcs sont ordonnés suivant la numérotation du sommet gauche de chacun d'eux. Les arcs qui symbolisent des alternatives — mêmes sommets gauche et droit — sont numérotés de la manière suivante :

`Parse` *I* `of` *N*`, from` *Gauche* `to` *Droite*

I	Compteur de solutions alternatives allant de 1 à *N*.
N	Nombre de solutions alternatives rencontrées.
Gauche	Numéro du sommet gauche de l'arc.
Droite	Numéro du sommet droit de l'arc.

Pour chaque arc, CPD n'affiche qu'une partie de l'information contenue dans la structure d'attributs. Seul l'arbre des dépendances syntaxiques est représenté de manière pseudo-graphique. Chaque nœud de l'arbre est symbolisé par la catégorie grammaticale (attribut `cat`) qui lui est associé et si, pour un nœud donné, une forme de citation est présente (attribut `form_cit` instancié), sa valeur apparaît entre parenthèses.

Pour compléter cette introduction, voici l'illustration de l'analyse d'une phrase qui ne fournit pas un résultat "idéal" :

```
yes
| ?- parse(0, _, "Utilisation finale de la traditionnelle carte vaudoise").

Parsing of :
(0) "Utilisation finale de la traditionnelle carte vaudoise" (7)
2 constituents were recognized.

Parse 1 of 2, from 0 to 2
np
 |-- determiner()
 `-- cnp
      |-- noun(utilisation)
      `-- adjp
           `-- adjective(final)
Parse 2 of 2, from 0 to 2
cnp
 |-- noun(utilisation)
 `-- adjp
      `-- adjective(final)
```

```
Parse 1 of 1, from 2 to 7
pp
 |-- prep(de)
 `-- np
      |-- determiner(le)
      `-- cnp
           |-- adjp
           |    `-- adjective(traditionnel)
           |-- noun(carte)
           `-- adjp
                `-- adjective(vaudois)

yes
| ?-
```

4.3 Les "événements" du processus d'analyse

La description conceptuelle dans le chapitre 3 des mécanismes de construction d'un chart par le processus d'analyse met en évidence trois entités : une entité statique, les règles de la grammaire et deux entités dynamiques, le chart et l'agenda. C'est la gestion de l'agenda et, en particulier, la manière d'y introduire de nouveaux éléments qui détermine la stratégie d'analyse. L'évolution du chart dépend essentiellement de l'arc qui se trouve *en tête de* l'agenda et la prise en charge de cet arc par le système permet la *création* d'un ou plusieurs nouveaux arcs. La *création d'arcs* sera ainsi considérée comme un événement capital dans la discrétisation du processus d'analyse. Un autre événement directement opposé au premier est l'*échec dans la tentative de création* de nouveaux arcs, qui a comme effet de réduire la taille de l'agenda. Il est donc tout aussi utile de signaler les réussites et les échecs dans les tentatives de création de nouveaux arcs pour pouvoir suivre l'évolution des entités dynamiques de CPD.

4.3.1 Le format d'un arc

Un arc du chart est implanté dans CPD comme une structure de données relativement complexe qui contient les différentes informations suivantes : les valeurs de ses sommets, le nom de la règle qui l'a engendré, la structure d'attributs et finalement, comme chaque arc symbolise une étape dans la tentative de vérification d'une règle, les hypothèses à vérifier.

Définition 2 (Le statut d'un arc) *Un arc du chart est un* arc supposition *si sa liste d'hypothèses à vérifier n'est pas vide. Un arc du chart est un* arc solution *si sa liste d'hypothèses à vérifier est vide.*

Il n'est pas opportun de présenter l'intégralité des informations associées à un arc lors de l'affichage de son contenu à l'écran. Nous avons donc fait une sélection. Un arc est ainsi représenté par le schéma général suivant :

Gauche-Droite: *Non-term*`#`*Nb*`(`*Ps*`, <`*Form-cit* `agr(`*G*`,`*N*`,`*P*`)>), [`*NT*...`]`

Gauche	Numéro du sommet gauche de l'arc.
Droite	Numéro du sommet droit de l'arc.
Non-term	Le nom de la règle qui a engendré cet arc.
Nb	L'instance de la règle qui a engendré cet arc[5].
Ps	Le poids de l'arc.

[5] La 1$^{\text{ère}}$ instance d'une règle est numérotée par 1. La valeur 0 qui apparaît sur certains arcs, est réservée aux arcs générés par la morphologie.

Form-cit Variable unifiée avec la forme de citation associée à cet arc[6].

G, N, P Variables unifiées respectivement avec les attributs `gender`, `number` et `person`.

NT... Les hypothèses qui doivent encore être vérifiées. Chacune des hypothèses est symbolisée soit par un atome (le nom d'une règle de la grammaire), soit par la séquence de caractères `"{...}"` qui représente un groupe de buts Prolog présents dans la règle.

4.3.2 Le format d'une *création d'arc*

CPD signale la création d'un nouvel arc de la manière suivante :

```
[created] using 3-7: np#2(A, <B agr(fem,sing,E)>)
... 3-7: sent#1(A,<B agr(C,D,E)>), [vp {...} ]<++>
```

qui met en perspective l'arc qui a servi à la création d'un ou de plusieurs nouveaux arcs et le premier des arcs générés. Les autres arcs peuvent être affichés suivant la commande fournie par l'utilisateur à la suite du prompt `"<++>"` (cf. section 4.5.1 pour la description des commandes disponibles).

L'information fournie se rapportant à la création d'un arc est suffisante pour détermier laquelle des trois règles présentées dans la section 3.1 a été appliquée.

> Si l'arc qui a servi à la création d'un autre arc est un arc solution, comme c'est le cas dans l'exemple ci-dessus, nous avons appliqué la règle 1 "nouvelle règle" qui initie la vérification d'une règle sur une nouvelle portion du chart. Le nom de la règle ainsi que son instance sont déterminés par le contenu du nouvel arc.
>
> Si l'arc qui a servi à la création d'un autre arc est un arc supposition, c'est le contenu de la liste des hypothèses qui nous indique laquelle des règles 2 ou 2' a été appliquée : si le premier élément de cette liste est une référence à un nom de règle, il s'agira de la règle 2; si le premier élément représente un groupe de buts, il s'agira de la règle 2'.

4.3.3 Le format d'un *échec de création*

L'échec d'une tentative de création d'un nouvel arc est signalé de la manière suivante :

```
[fail to create] using ...
... 6-7: cnp#4(A, <B, agr(C,D,E)>), [noun adjp {...} ]<++>
```

Ici l'emphase, c'est-à-dire la possibilité d'obtenir des compléments d'information, est mise sur l'arc qui n'a pas permis la création de nouveaux arcs, ce qui autorise l'utilisateur à entreprendre des recherches pour comprendre cet échec.

4.4 Les prédicats de configuration de CPD

Nous avons mentionné plus haut que CPD peut fonctionner dans différents modes à l'instar de Prolog qui a un mode "trace" et un mode "debug". CPD reprend ces possibilités et les transposent dans le cadre des événements que constituent la création d'arcs ou l'échec.

Les prédicats qui permettent de modifier l'état interne et le mode de fonctionnement de CPD sont directement inspirés des prédicats équivalents dans l'environnement de mise au point de Prolog. Ainsi, il existe un mode *de trace exhaustive* des événements qui est activé par l'exécution de `"<>trace."` et qui a un effet comparable au prédicat `trace/0` de Prolog. Afin de s'épargner la tâche difficile de trouver des noms significatifs pour les prédicats de configuration de CPD, nous avons préféré utiliser les mêmes noms que pour Prolog mais en ajoutant le préfixe `"<>"` ce qui évite tout conflit entre les deux niveaux, CPD et Prolog.

[6] Seuls les arcs générés par la morphologie ont habituellement une forme de citation. Dans de très rares cas, les règles de la grammaire fixent une valeur à cet attribut. Malgré tout, cette information reste utile pour repérer l'emplacement des sommets dans le chart et, par là même, comprendre les portions recouvertes par les arcs.

4.4.1 La trace exhaustive : <>trace/0, <>notrace/0

L'activation du mode *de trace exhaustive* de CPD permet à l'utilisateur d'être informé de tous les événements qui interviennent lors de la construction du chart. Lorsque le prédicat **<>trace/0** d'activation de la trace exhaustive est exécuté, CPD fournit le message suivant :

```
| ?- <>trace.
Chart Parser Debugger will first creep ! -- showing everything

yes
| ?-
```

Dès ce moment toute exécution de **parse/3** produit une séquence de messages comparables à ceux décrits dans la section 4.3. Après chaque message, CPD se met en attente d'une commande interactive en affichant le prompt "<++>". La commande qui garantit un affichage exhaustif de tous les événements est la touche <RET>. Il existe de nombreuses autres commandes possibles à ce niveau. Nous proposons au lecteur de se reporter à la section 4.5.1 pour une description détaillée de toutes ces possibilités.

La trace exhaustive peut être désactivée par le prédicat **<>notrace/0**; CPD est alors remis dans son mode de fonctionnement de base et affiche le message suivant :

```
| ?- <>notrace.
Chart Parser Debugger is switched off !

yes
| ?-
```

4.4.2 La trace sélective : <>debug/0, <>nodebug/0

L'activation du mode *de trace sélective* de CPD permet à l'utilisateur d'être informé des événements directement reliés aux règles ou instances de règles qu'il a sélectionnées à l'aide de **<>spy/1** (cf. 4.4.3). L'exécution de **<>debug/0** donne lieu au message suivant :

```
| ?- <>debug.
Chart Parser Debugger will first creep ! -- showing spypoints

yes
| ?-
```

Dès ce moment toute exécution de **parse/3** produit une séquence de messages comparables à ceux décrits dans la section 4.3. Après chaque message, CPD se met en attente d'une commande interactive en affichant le prompt "<++>". La commande qui garantit un affichage sélectif des événements est la touche "s". Il existe de nombreuses autres commandes possibles à ce niveau. Nous proposons au lecteur de se reporter à la section 4.5.1 pour une description détaillée de toutes ces possibilités.

La trace sélective peut être désactivée par le prédicat **<>nodebug/0**; CPD est alors remis dans son mode de fonctionnement de base et affiche le message suivant :

```
| ?- <>nodebug.
Chart Parser Debugger is switched off !

yes
| ?-
```

4.4.3 Les points d'arrêt et les points de contrôle : `<>spy/1`, `<>nospy/1`, `<>nospyall/0`

Définition 3 (Un point d'arrêt vs un point de contrôle) *Un* point d'arrêt *est une marque fictive placée sur une instance d'une règle de la grammaire. Chaque fois qu'un événement concerne un arc issu d'une instance de cette règle, CPD génère un message et se met en attente d'une commande interactive. Un* point de contrôle *est aussi une marque fictive placée sur une instance d'une règle de la grammaire. Chaque fois qu'un événement concerne un arc issu d'une instance de cette règle, CPD génère un message mais continue son exécution sans attendre une commande interactive fournie par l'utilisateur.*

La définition des points d'arrêt et des points de contrôle se fait à l'aide du prédicat `<>spy/1`. Grâce au fait que `spy/1` est défini comme un opérateur préfixé, il est possible d'omettre les parenthèses dans l'énoncé d'un point d'arrêt.

`<>spy +`*Spec*, avec *Spec* qui peut prendre une des formes suivantes :

Nom`#`*Instance*
: Définit un point d'arrêt sur l'instance numérotée *Instance* de la règle nommée *Nom*.

Nom
: Définit un point d'arrêt sur toutes les instances de la règle ayant *Nom* comme nom.

`result`
: Définit un point d'arrêt juste avant l'affichage des résultats (cf. section 4.4.5 pour les détails d'utilisation de ce cas particulier).

`u` *Nom*`#`*Instance*
: Définit un point de contrôle ("u" pour "unleashed") sur l'instance numérotée *Instance* de la règle nommée *Nom*.

`u` *Nom*
: Définit un point de contrôle sur toutes les instances de la règle ayant *Nom* comme nom.

`[ ... ]`
: Une liste d'élements de la forme de ceux qui précèdent. Ceci permet de grouper en une commande plusieurs définitions de points d'arrêt et de points de contrôle.

Lors de l'exécution de `<>spy/1`, CPD réagit en mettant automatiquement le système en mode de trace sélective. D'autre part, si la spécification d'un point d'arrêt ne correspond à aucune règle momentanément chargée, CPD affiche le message d'erreur suivant :

```
| ?- <>spy sent#1.  Chart Parser Debugger will first leap ! -- showing
spypoints

yes
| ?- <>spy xxxxx.
CPD warning: No (instance of) rule matches this spec. !

yes
| ?-
```

L'élimination d'un point d'arrêt ou d'un point de contrôle se fait à l'aide du prédicat `<>nospy/1`. Comme pour `<>spy/1`, `nospy/1` est défini comme un opérateur préfixé et l'argument de `<>nospy/1` peut prendre exactement les mêmes formes que celui de `<>spy/1`. Lors de l'exécution de `<>nospy/1`, CPD ne fournit aucun message spécifique, sauf si la spécification du point d'arrêt ou de contrôle ne correspond à aucun point d'arrêt ou de contrôle existant. Dans ce cas, CPD affiche le message d'erreur suivant :

```
| ?- <>nospy sent#1.

yes
| ?- <>nospy xxxxx.
CPD warning: No existing spypoint matches this spec. !

yes
| ?-
```

L'élimination de tous les points d'arrêt et de contrôle est possible à l'aide du prédicat `<>nospyall/0` qui a comme effet auxiliaire de mettre CPD dans son mode de fonctionnement de base.

4.4.4 L'état courant de CPD : `<>debugging/0`

L'état courant de CPD peut être obtenu en exécutant le prédicat `<>debugging/0`. Le système indique alors le mode de fonctionnement actuel à l'aide d'un des trois messages suivants :

`Chart Parser Debugger is switched off !` Mode de base.

`Chart Parser Debugger will first creep ! -- showing everything` Mode de trace exhaustive.

`Chart Parser Debugger will first leap ! -- showing spypoints` Mode de trace sélective.

CPD indique ensuite les points d'arrêt ou de contrôle sous la forme d'une suite de lignes ayant chacune le format suivant :

Nom`#`*Instance* `[` *Type* `]`

Nom Le foncteur du membre gauche de la règle considérée.

Instance Le numéro de l'instance de la règle.

Type Une des deux valeurs, `"L"` pour "leashed" — point d'arrêt —, `"U"` pour "unleashed" — point de contrôle.

Un exemple typique d'informations fournies par CPD est le suivant :

```
| ?- <>debugging.
Chart Parser Debugger will first leap ! -- showing spypoints
sent#1 [ L ]
cnp#3 [ L ]
opt_determiner#1 [ U ]
opt_determiner#2 [ U ]

yes
| ?-
```

Lorsque CPD fonctionne en mode de trace exhaustive ou sélective il signale les événements pour lesquels un point d'arrêt est défini en remplaçant `"..."`, en début de ligne, par `".+."`.

4.4.5 L'analyse détaillée des résultats : `<>spy result`

CPD offre la possibilité de placer un point d'arrêt très particulier avant l'affichage des résultats afin de permettre une étude détaillée de tous les arcs solution du chart. Pour placer un tel point d'arrêt il faut exécuter le prédicat `<>spy result`. Dès ce moment, toute exécution de `parse/3` s'effectue de la manière suivante :

```
| ?- parse(0, _, "Utilisation de la carte").
Parsing of :
(0) "Utilisation de la carte" (4)

Results : (s/c/n/<RET>/e)
```

CPD est alors en attente d'une commande interactive pour initier l'affichage des arcs solution du chart. Le choix des commandes possibles est mentionné pour mémoire entre parenthèses[7] :

[7] Il existe d'autres situations dans lesquelles figurent des commandes ayant la même signification. Nous avons pris garde pour éviter toute confusion d'uniformiser les noms des commandes à tous les niveaux possibles d'interaction.

`s` pour "skip" : initie l'affichage sélectif des arcs solution sur lesquels portent un point d'arrêt ou de contrôle défini à l'aide de `<>spy/1` et met CPD en mode de trace sélective.

`c` pour "creep" : initie l'affichage de tous les arcs solution du chart et met CPD en mode de trace exhaustive.

`<RET>` joue le même rôle que `"c"`.

`n` pour "no trace" : termine l'opération d'affichage des arcs et met CPD au mode de base.

`e` pour "exit" : termine l'opération d'affichage, sans changer le mode de CPD.

Lorsque l'affichage des arcs solution est initié avec une des commandes ci-dessus, CPD entame l'affichage des arcs de la manière suivante :

```
Results : (s/c/n/<RET>/e) c
   >>> 3-4: np#2(A, <B agr(fem,sing,C)>), <++>
```

La mise en page des arcs avec, sur la gauche, `" >>>"` et le prompt `"<++>"` indique à l'utilisateur qu'il se trouve dans le *contexte d'interaction sur les entités* qui est décrit en détail dans la section 4.5.2 et qui est un des trois *contextes d'interaction* possibles de CPD.

4.5 Les contextes d'interaction de CPD

Lorsque CPD fonctionne en mode de trace exhaustive ou sélective, le système affiche des informations et se met éventuellement en attente d'une commande interactive. Cette commande est généralement constituée d'un caractère suivi de `<RET>`. Il existe dans CPD trois contextes dans lesquels le système se met en attente d'une commande, ce sont les trois contextes d'interaction de CPD.

4.5.1 Le contexte d'interaction sur les événements

Le contexte d'interaction sur les événements permet à l'utilisateur d'obtenir des compléments d'information sur les arcs générés par CPD lors d'une tentative de création d'un arc, qu'il s'agisse d'une tentative réussie ou d'un échec. Un exemple typique d'un tel contexte est le suivant :

```
[created] using 3-7: np#2(A, <B agr(fem,sing,E)>)
... 3-7: sent#1(A,<B agr(C,D,E)>), [vp {...} ]<++>
```

Le système est en attente d'une commande après avoir affiché le prompt `"<++>"` et l'utilisateur a alors à disposition toute la gamme de commandes suivantes :

`h` pour "help", la seule commande à mémoriser, qui affiche l'aide-mémoire suivant :

```
Help:
<i>A  Agenda edges
<i>C  Chart edges (actives & inactives)
  T   pretty print the tree structure
<RET> creep
  a   abort
  c   creep
  h   help
  n   no trace
  s   skip to a spypoint
  u   unleash this
  w   write current edge
  +   spy this
  -   unspy this
  =   debugging status
```

`A` pour "agenda" : affiche en séquence et sans interaction possible tous les arcs de l'agenda.

i`A` si `"A"` est précédé du nombre entier i, CPD affiche uniquement les arcs de l'agenda dont le sommet gauche s'unifie avec ce nombre et place CPD dans le mode d'interaction sur les entités (cf. section 4.5.2).

`C` pour "chart" : affiche en séquence et sans interaction possible tous les arcs du chart.

i`C` si `"C"` est précédé du nombre entier i, CPD affiche uniquement les arcs du chart dont le sommet gauche s'unifie avec ce nombre et place CPD dans le mode d'interaction sur les entités (cf. section 4.5.2).

`T` pour "tree structure" : affiche l'arbre des dépendances syntaxiques et met CPD dans le contexte d'interaction sur les structures d'attributs (cf. section 4.5.3).

`<RET>` joue le même rôle que `"c"`.

`a` pour "abort" : interrompt irrévocablement le processus d'analyse après avoir demandé une confirmation.

`c` pour "creep" : affiche le prochain événement du processus d'analyse et met CPD en mode de trace exhaustive.

`n` pour "no trace" : termine l'analyse sans nouvelle interaction et met CPD en mode de base.

`s` pour "skip" : affiche le prochain événement qui fait partie des points d'arrêts ou de contrôle et met CPD en mode de trace sélective.

`u` pour "unleash" : place un point de contrôle sur l'instance de la règle qui a engendré l'arc courant.

`w` pour "write" : réaffiche l'arc courant.

`+` place un point d'arrêt sur l'instance de la règle qui a engendré l'arc courant.

`-` enlève un point d'arrêt associé à l'arc courant.

`=` affiche l'état actuel de CPD.

4.5.2 Le contexte d'interaction sur les entités

Le contexte d'interaction sur les entités permet d'obtenir des compléments d'information sur les entités dynamiques de CPD que sont l'agenda et le chart. Un exemple typique d'un tel contexte est le suivant :

```
... 5-6: cnp#5(1, <A agr(fem,sing,B)>), [{...} ]<++> 1C
Active edges of the chart (left vertex = 1):


Inactive edges of the chart (left vertex = 1):

   >>> 1-2: adjective#0(1, <final agr(fem,sing,A)>), <++>
```

Ce contexte d'interaction est caractérisé par la présence de `" >>>"` sur la gauche de la description de l'arc et par le prompt `"<++>"`, sur la droite. L'utilisateur a alors à sa disposition un sous-ensemble des commandes du contexte d'interaction sur les événements :

`h` pour "help", encore et toujours la seule commande à mémoriser, qui affiche l'aide-mémoire suivant :

```
Help:
  T   pretty print the tree structure
<RET> creep
```

```
a   abort
c   creep
h   help
n   no trace
s   skip to a spypoint
w   write current edge
=   debugging status
```

`T` pour "tree structure" : affiche l'arbre des dépendants syntaxiques et met CPD dans le contexte d'interaction sur les structures d'attributs (cf. section 4.5.3).

`<RET>` joue le même rôle que `"c"`.

`a` pour "abort" : interrompt irrévocablement le processus d'analyse après avoir demandé une confirmation.

`c` pour "creep" : affiche le prochain arc de l'entité dynamique concernée. Si le dernier arc de l'entité vient d'être affiché, CPD se remet dans le contexte d'interaction englobant.

`n` pour "no trace" : termine l'analyse sans nouvelle interaction et met CPD en mode de base.

`s` pour "skip" : affiche le prochain arc de l'entité dynamique concernée pour lequel existe un point d'arrêt. Si le dernier arc de l'entité vient d'être affiché, CPD se remet dans le contexte d'interaction englobant.

`w` pour "write" : réaffiche l'arc courant.

`=` affiche l'état actuel de CPD.

4.5.3 Le contexte d'interaction sur les structures d'attributs

Le contexte d'interaction sur les structures d'attributs permet à l'utilisateur d'obtenir des informations de détail sur le contenu de l'arbre des dépendances syntaxiques. Un exemple typique d'un tel contexte est le suivant :

```
... 0-4: np#2(A, <B agr(fem,sing,C)>), <++> T
np<1>
 |-- determiner()<2>
 `-- cnp<3>
      |-- noun(utilisation)<4>
      |-- pp<5>
      |    |-- prep(de)<6>
      |    `-- np<7>
      |         |-- determiner(le)<8>
      |         `-- cnp<9>
      |              `-- noun(carte)<10>
      `-- pp()<11>
           `-- _36817(par)<12>
<FS>
```

La caractéristique visible de ce contexte est le nouveau prompt `"<FS>"`. Les commandes disponibles sont les suivantes :

`h` pour "help", encore et toujours la seule commande à mémoriser, qui affiche l'aide-mémoire suivant :

```
Help:
<i><RET> content of selected node
   T     redisplay the tree structure
   a     abort
   h     help
   s     skip to a spypoint
```

i`<RET>` Lorsque `<RET>` est précédé du nombre entier *i*, CPD affiche le contenu détaillé du nœud ayant ce numéro dans l'arbre. Les attributs `head` et `daughters` lorsqu'ils sont unifiés à d'autres noeuds de l'arbre, sont affichés comme des références numériques à ces noeuds, comme dans l'exemple suivant :

```
cnp<1>
 |-- noun(utilisation)<2>
 |-- pp<3>
 |     |-- prep(de)<4>
 |     `-- np<5>
 |           |-- determiner(le)<6>
 |           `-- cnp<7>
 |                 `-- noun(carte)<8>
 `-- pp()<9>
       `-- _28885(par)<10>
<FS> 3
FS values display: <3>

cat-----:pp
agr-----:A
head----:<8>
daught.-:[<4>, <5>]
<FS>
```

`T` pour "tree structure" : réaffiche l'arbre des dépendances syntaxiques.

`a` pour "abort" : interrompt irrévocablement le processus d'analyse après avoir demandé une confirmation.

`s` pour "skip" : quitte le contexte d'interaction sur les structures d'attributs et réaffiche le message du contexte englobant.

5 Conclusion

Le système CPD a été utilisé pour l'analyse de textes français dans le cadre du projet de recherche *Specification and Prototyping of a System for the Intelligent Management of Information*. La version actuelle sera modifiée dans un avenir proche. En particulier, nous comptons introduire un mécanisme d'élimination sélective des arcs. Cette technique devrait améliorer très sensiblement les performances de l'analyseur en ne conservant que les résultats jugés intéressants. Une version expérimentale introduisant cette technique existe déjà mais elle ne traite, pour l'instant, qu'une partie des cas. D'autres adaptations plus cosmétiques devraient aussi être réalisées concernant, par exemple, la syntaxe des grammaires linguistiques et la dénomination des arcs.

Comme tout ce système est implanté en SICStus-Prolog, il est aussi envisageable de le transporter sur un autre matériel que SUN. En particulier, une version sur PC en Quintus-Prolog peut être mise en place très rapidement.

Dans son organisation actuelle, CPD est intimement lié aux autres composantes développées dans le cadre du projet de recherche, en particulier la morphologie et les prédicats d'accès de la structure d'attributs. Il ne s'agit donc pas d'un module indépendant qu'on rattache à un système existant pour augmenter la fonctionalité de l'ensemble. Une telle décomposition est souhaitable. Elle est, du reste, à l'étude et devrait voir le jour très prochainement.

6 Résumé des commandes

`parse(+`*Initial*`, -`*Final*`, +`*Texte*`)` (cf. page 10)
: Effectue une analyse de *Texte* en générant un chart dont le sommet initial a la valeur *Initial* et le sommet final a la valeur *Final*.

`<>trace` (cf. page 15)
: Active le mode de trace exhaustive de CPD.

`<>notrace` (cf. page 15)
: Désactive le mode de trace exhaustive de CPD.

`<>debug` (cf. page 15)
: Active le mode de trace sélective de CPD.

`<>nodebug` (cf. page 15)
: Désactive le mode de trace sélective de CPD.

`<>spy(+`*Spec*`)` (cf. page 16)
: Place un point d'arrêt ou un point de contrôle sur une règle ou une instance de règle déterminée par *Spec*.

`<>nospy(+`*Spec*`)` (cf. page 16)
: Enlève un point d'arrêt ou un point de contrôle sur une règle ou une instance de règle déterminée par *Spec*.

`<>nospyall` (cf. page 17)
: Enlève tous les points d'arrêt ou de contrôle placés antérieurement.

`<>debugging` (cf. page 17)
: Affiche l'état courant de CPD.

Références

[1] Mats Carlsson and Johan Widén. *SICStus Prolog User's Manual.* Swedish Institute of Computer Science, PO Box 1263, S-164 28 KISTA, Sweden, July 1990.

[2] J. Earley. An efficient context-free parsing algorithm. *Communications of the Association for Computer Machinery*, 13(2):94–102, 1970.

[3] M. Kay. The MIND system. In R. Rustin, editor, *Natural Language Processing.* Prentice Hall, 1973.

[4] Gerard Mellish and Chris Gazdar. *Natural Language Processing in Prolog.* Addison-Wesley, 1989.

[5] Peter Ross. *Advanced Prolog, Techniques and Examples.* Addison-Wesley, 1989.