Constraint Categorial Grammars

Luís Damas, Nelma Moreira
{luis,nam}@ncc.up.pt

LIACC, Universidade do Porto
Rua do Campo Alegre 823, 4150 Porto, Portugal

Abstract. Although unification can be used to implement a weak form of β-reduction, several linguistic phenomena are better handled by using some form of λ-calculus. In this paper we present a higher order feature description calculus based on a typed λ-calculus. We show how the techniques used in $\mathcal{CLG}$ for resolving complex feature constraints can be efficiently extended. $\mathcal{CCLG}$ is a simple formalism, based on categorial grammars, designed to test the practical feasibility of such a calculus.
Keywords: constraint satisfaction, computational semantics, high-order programming.

1 Introduction

Unification based formalisms show a clear inability to deal in a natural way with phenomena such as the semantics of coordination, quantification scoping ambiguity or bound anaphora. As a matter of fact, although unification can be used to implement a weak form of β-reduction, it seems that this kind of phenomena is better handled by using some form of λ-calculus [DSP91, Per90]. One possibility, which is at the heart of systems like λProlog[NM88], is to extend both the notion of term, to include λ-abstraction and application, and the definition of unification to deal with λ-terms. For this extension to be technically sound it is necessary to require λ-terms to be well typed. On the other hand, it turns out that if instead of using terms we use complex feature descriptions (where conjunction replaces unification), we still can follow the same plan to produce a higher-order calculus of feature descriptions. $\mathcal{CCLG}$ is a simple formalism, based on categorial grammars, designed to test the practical feasibility of such an approach. The main reason for selecting a categorial framework for this experiment was that, due to the simplicity of the categorial framework, it allowed us to concentrate on the constraint calculus itself. Another reason was also the close historical relationship between categorial grammars and semantic formalisms incorporating λ-abstraction. $\mathcal{CCLG}$ extends categorial grammar by associating not only a category but also a higher-order feature description with each well-formed part of speech. The type of these feature descriptions are determined by the associated category. Note also that a derivation leading to an unsatisfiable feature description is legal. When compared with other formalisms (for instance, [ZKC87]) one of the main distinguishing features of $\mathcal{CCLG}$ is the fact that it computes partial descriptions of feature structures and not the feature structures themselves.

It is important to notice that this calculus is easily modified to deal with constraints over finite or rational trees, instead of feature trees. Also, the advantages of this kind of calculus, namely its decidability, over the use of general high-order logic programming systems, for processing semantic representations in NLP systems should be obvious. The rest of the paper proceeds as follows. We start by defining a feature description calculus as an hybrid of λ-calculus and feature logics and we present its denotational semantics. In section 2.1 we describe a complete constraint solver for higher-order feature descriptions. In section 3 we define constraint categorial grammars and briefly present its implementation. Some final remarks are considered in section 4.

2 Feature Description Calculus

The feature description calculus $\Lambda_{\mathcal{FD}}$ at the heart of our formalism is inspired both on the λ-calculus and on feature logics [Smo89]. For technical reasons, namely that we want to ensure the existence of normal forms, it is a typed calculus. Our base types are **bool** for truth values and **fs** for feature structures.

Our types are described by

$$\tau ::= \mathbf{bool} \mid \mathbf{fs} \to \tau \mid \tau \to \tau'$$

Note that we exclude **fs** as the type of any feature description. This reflects our commitment to compute partial descriptions of feature structures rather than feature structures.

Now assume we are given a set of *atoms* a, b, ..., a set of *feature* symbols f, g, ..., a set of *feature structure variables* x, y, ..., and, for each type τ, a set of *variables of type* τ x_τ, y_τ, ... Then the set of *feature descriptions* of type τ is described by

$$\begin{array}{ll} e_\tau & ::= \mathbf{true} \mid \mathbf{false} \mid x_\tau \mid e_\tau \wedge e_\tau \mid e_\tau \vee e_\tau \mid \neg e_\tau \mid e_{fs\to\tau} x.p \mid e_{fs\to\tau} a \mid e_{\tau'\to\tau} e_{\tau'} \\ e_{bool} & ::= t.p \doteq s \mid t = s \\ e_{fs\to\tau} & ::= \lambda x.e_\tau \\ e_{\tau'\to\tau} & ::= \lambda x_{\tau'}.e_\tau \end{array}$$

where s and t denote either atoms or feature structure variables, and p is a, possibly empty, sequence of feature symbols denoting a path in a feature structure.

Note that the language thus defined includes both feature logics and a typed λ-calculus.

We import from both theories such notions as substitution, free and bounded occurrences of variables, $\alpha-$ and $\beta-$reductions and $\beta\alpha$-normal form. In particular, a *closed feature description* is a feature description with no free variables. Moreover, feature constraints of feature logics, widely used in unification grammars, correspond to a subset of *feature descriptions* of type **bool**, without abstractions or applications.

To define a semantics for the calculus of feature descriptions we adopt the standard model $\mathcal{RT}$ of rational trees for feature structures (see [DMV94]) and associate with each type τ a semantic domain D_τ as follows

$$\begin{aligned} D_{\mathbf{bool}} &= \{0,1\} \\ D_{\mathbf{fs}\to\tau} &= \mathcal{RT} \to D_\tau \\ D_{\tau'\to\tau} &= D_{\tau'} \to D_\tau \end{aligned}$$

From this point on a semantics for feature descriptions is defined in the same way as for feature logics and the typed λ-calculus by noting that the standard boolean operations can be extended to all the semantic domains involved in a component wise fashion, e.g.

$$(\lambda x.e) \vee (\lambda x.e') =_{def} (\lambda x.e \vee e').$$

More precisely, let an *assignment* ρ be a mapping defined on variables, such that $\rho(x) \in \mathcal{RT}$ and $\rho(x_\tau) \in D_\tau$, for each type τ. As usual, $\rho[d/\sigma]$ denotes the assignment obtained from ρ by mapping σ to d. Let $f^{\mathcal{RT}}$, $p^{\mathcal{RT}}$ and $a^{\mathcal{RT}}$ denote the interpretation of features, paths and atoms in $\mathcal{RT}$, respectively. Furthermore, let $t^{\mathcal{RT}}\rho$ be $\rho(t)$ if t is a variable and $t^{\mathcal{RT}}$ otherwise. Then, the semantics of feature descriptions $\Lambda_{\mathcal{FD}}$ given an assignment ρ is defined inductively, as follows:

$$\begin{aligned} [\![x_\tau]\!]\rho &= \rho(x_\tau) \\ [\![t.p \doteq s]\!]\rho &= \begin{cases} 1 & if\ \ p^{\mathcal{RT}}(t^{\mathcal{RT}}\rho) = s^{\mathcal{RT}}\rho \\ 0 & otherwise \end{cases} \\ [\![t = s]\!]\rho &= \begin{cases} 1 & if\ \ t^{\mathcal{RT}}\rho = s^{\mathcal{RT}}\rho \\ 0 & otherwise \end{cases} \end{aligned}$$

$$\begin{aligned} [\![\lambda x.e_\tau]\!]\rho &= \lambda\!\!\lambda v.[\![e_\tau]\!]\rho[v/x] \quad (v \in \mathcal{RT}) \\ [\![\lambda x_\tau.e'_\tau]\!]\rho &= \lambda\!\!\lambda v.[\![e'_\tau]\!]\rho[v/x_\tau] \quad (v \in \mathcal{D}_\tau) \\ [\![e_{fs\to\tau}x.p]\!]\rho &= ([\![e_{fs\to\tau}]\!]\rho)[\![x.p]\!]\rho \\ [\![e_{fs\to\tau}a]\!]\rho &= ([\![e_{fs\to\tau}]\!]\rho)a^{\mathcal{RT}} \\ [\![e_{\tau\to\tau'}e'_\tau]\!]\rho &= ([\![e_{\tau\to\tau'}]\!]\rho)[\![e'_\tau]\!]\rho \end{aligned}$$

where $\lambda\!\!\lambda$ denotes function "abstraction" in set theory and $(x \in D)$ means that D is the domain of x. For the conjunction operation, we define:

$$\begin{aligned} [\![e_{bool} \wedge e'_{bool}]\!]\rho &= \begin{cases} 1 & \text{if } [\![e_{bool}]\!]\rho = 1 \text{ and } [\![e'_{bool}]\!]\rho = 1 \\ 0 & \text{otherwise} \end{cases} \\ [\![e_{fs\to\tau} \wedge e'_{fs\to\tau}]\!]\rho &= \lambda\!\!\lambda v.d \wedge d' \text{ where } [\![e_{fs\to\tau}]\!]\rho = \lambda\!\!\lambda v.d \ \ (v \in \mathcal{RT}) \\ & \qquad\qquad\qquad\quad\ [\![e'_{fs\to\tau}]\!]\rho = \lambda\!\!\lambda v.d' \ (v \in \mathcal{RT}) \\ [\![e_{\tau'\to\tau} \wedge e'_{\tau'\to\tau}]\!]\rho &= \lambda\!\!\lambda v.d \wedge d' \text{ where } [\![e_{\tau'\to\tau}]\!]\rho = \lambda\!\!\lambda v.d \ \ (v \in \mathcal{D}_{\tau)'} \\ & \qquad\qquad\qquad\quad\ [\![e'_{\tau'\to\tau}]\!]\rho = \lambda\!\!\lambda v.d' \ (v \in \mathcal{D}_{\tau'}) \end{aligned}$$

and analogously for the other boolean operations. If $1_{bool} \equiv 1$, $1_{fs\to\tau} \equiv \lambda\!\!\lambda v.1_\tau$, and $1_{\tau'\to\tau} \equiv \lambda\!\!\lambda v.1_\tau$ then the semantics of **true**, for each type τ is defined by:

$$\begin{aligned} &[\![\mathbf{true}_{bool}]\!]\rho = 1_{bool}, \\ &[\![\mathbf{true}_{fs\to\tau}]\!]\rho = \lambda\!\!\lambda v.1_\tau \ (v \in \mathcal{RT}) \\ &[\![\mathbf{true}_{\tau'\to\tau}]\!]\rho = \lambda\!\!\lambda v.1_\tau \ (v \in \mathcal{D}_{\tau'}) \end{aligned}$$

and analogously for **false**.

An important property of the feature description calculus is the existence of normal form under β-reduction which is a simple consequence of well-typedness. Another important property is that for any closed feature description of type

τ we can decide if it is equivalent to **false**. This last property is essentially an extension of the satisfiability problem for a complete axiomatization of feature logics. For this reason we will say that a feature description of type τ is satisfiable iff its semantics is not that of **false**.

2.1 Constraint Solver

Our implementation of the feature description calculus is based on the reduction to normal form followed by the techniques used in $\mathcal{CLG}$ [DV92, DMV94] for resolving complex feature constraints. In order to face the NP-hardness of the satisfiability problem, our approach was based in factoring out, in polynomial time, deterministic information contained in a complex constraint and simplifying the remaining formula using that information. The deterministic information corresponds to a conjunction of (positive) atomic constraints in *solved form*[1], which we denote by $\mathcal{M}$. We say that $\mathcal{M}$ is a *partial model* of $\mathcal{C}$ if and only if every model of $\mathcal{C}$ is a model of $\mathcal{M}$. When every model of $\mathcal{M}$ is a model of $\mathcal{C}$, but no proper subset of $\mathcal{M}$ satisfies this condition, we will say that $\mathcal{M}$ is a *minimal* model of $\mathcal{C}$. By using disjunctive forms it can be proved that any set of feature constraints $\mathcal{C}$ admits at most a finite number of minimal models[2]. In [DV92, DMV94, Mor97] a rewrite system was presented that from a complex feature constraint $\mathcal{C}_0$ produces a pair $\langle\mathcal{M},\mathcal{C}\rangle$, where $\mathcal{M}$ is *solved form*, $\mathcal{C}$ is smaller than C_0 and such that $\mathcal{RT} \models \mathcal{C}_0 \leftrightarrow \mathcal{M} \wedge \mathcal{C}$ and any minimal model of $\mathcal{C}_0$ can be obtained by conjoining a minimal model of $\mathcal{C}$ with $\mathcal{M}$. Moreover the rewriting system is complete in the sense that $\mathcal{M} \wedge \mathcal{C}$ is satisfiable, unless it produces **false** as the final model.

We now extend that rewrite system to higher-order feature descriptions. First we give some more characterizations of feature descriptions. A *basic normal description* of type τ is described by:

$$
\begin{array}{ll}
e_\tau & ::= \mathbf{true} \mid \mathbf{false} \mid x_\tau \mid x_\tau \wedge e_\tau \mid x_\tau \vee e_\tau \mid \neg e_\tau \mid x_{fs\to\tau} x.p \mid x_{fs\to\tau} a \mid x_{\tau'\to\tau} e_{\tau'} \\
e_{bool} & ::= t.p \doteq s \mid t = s \\
e_{fs\to\tau} & ::= \lambda x.e_\tau \\
e_{\tau'\to\tau} & ::= \lambda x_{\tau'}.e_\tau
\end{array}
$$

Then, every closed feature description in basic normal form will be of the form $\lambda\bar{x}_\sigma.e_{bool}$ where $\bar{x}_\sigma$ denotes a sequence of bound variables of some types and e_{bool} is not an abstraction. Omitting the λ prefix, given a feature description of

[1] A conjunction of feature constraints $\mathcal{M}$ is a *solved form* if:

1. every constraint in $\mathcal{M}$ is of the form $x.f \doteq s$ or $x = s$
2. if $x = s$ is in $\mathcal{M}$ then x occurs exactly once in $\mathcal{M}$
3. if $x.f \doteq s$ and $x.f \doteq t$ are in $\mathcal{M}$ then $s = t$

Any conjunction of atomic constraints is satisfiable if and only if it can be reduced to a *solved form* [Smo89, Mah88, DMV94]

[2] Actually it is necessary to extend the notion of models to include negative atomic constraints, but that will not be addressed here.

type **bool**, e_{bool}, the solver will produce a partial model $\mathcal{M}$ and a smaller feature description e'_{bool} or **false**:

$$
\begin{array}{l}
\langle \mathcal{M}, e_{bool} \wedge \mathbf{false} \rangle \rightarrow \langle \perp, \mathbf{false} \rangle \\
\langle \mathcal{M}, e_{bool} \wedge \mathbf{true} \rangle \rightarrow \langle \mathcal{M}, e_{bool} \rangle \\
\langle \mathcal{M}, e_{bool} \wedge s = t \rangle \rightarrow \langle \mathcal{M} \wedge s = t, e_{bool} \rangle \\
\langle \mathcal{M}, e_{bool} \wedge t.p \doteq s \rangle \rightarrow \langle \mathcal{M} \wedge t.p \doteq s, e_{bool} \rangle \\
\langle \mathcal{M}, e_{bool} \rangle \rightarrow \langle \mathcal{M}, e'_{bool} \rangle \text{ if } e_{bool} \ \longrightarrow^{\star}_{\mathcal{M}} \ e'_{bool}
\end{array}
$$

with the convention that after each application of one of the rewrite rules the new partial model is reduced to solved form (or **false**). The complete rewrite system $\longrightarrow_{\mathcal{M}}$ is:

$$
\begin{array}{llllll}
(\lambda x.e_\tau)x.p & \longrightarrow_{\mathcal{M}} & e_\tau[x.p/x] & \neg(\lambda x.e) & \longrightarrow_{\mathcal{M}} & \lambda x.\neg e \\
(\lambda x.e_\tau)a & \longrightarrow_{\mathcal{M}} & e_\tau[a/x] & (\lambda x.e) \wedge (\lambda x.e') & \longrightarrow_{\mathcal{M}} & \lambda x.e \wedge e' \\
(\lambda x_{\tau'}.e_\tau)e'_{\tau'} & \longrightarrow_{\mathcal{M}} & e_\tau[e'_{\tau'}/x_{\tau'}] & (\lambda x.e) \vee (\lambda x.e') & \longrightarrow_{\mathcal{M}} & \lambda x.e \vee e
\end{array}
\tag{1}
$$

$$
\begin{array}{llll}
e_\tau \wedge x_\tau & \longrightarrow_{\mathcal{M}} & x_\tau \wedge e_\tau & \text{if } e_\tau \text{ is not a variable} \\
e_\tau \wedge (x_\tau \wedge e'_\tau) & \longrightarrow_{\mathcal{M}} & x_\tau \wedge (e_\tau \wedge e'_\tau) & \text{if } e_\tau \text{ is not a variable} \\
(e_\tau \wedge e'_\tau) \wedge e''_\tau & \longrightarrow_{\mathcal{M}} & e_\tau \wedge (e'_\tau \wedge e''_\tau) & \\
e_\tau \vee x_\tau & \longrightarrow_{\mathcal{M}} & x_\tau \vee e_\tau & \text{if } e_\tau \text{ is not a variable} \\
e_\tau \vee (x_\tau \vee e'_\tau) & \longrightarrow_{\mathcal{M}} & x_\tau \vee (e_\tau \vee e'_\tau) & \text{if } e_\tau \text{ is not a variable} \\
(e_\tau \vee e'_\tau) \vee e''_\tau & \longrightarrow_{\mathcal{M}} & e_\tau \vee (e'_\tau \vee e''_\tau) & \\
(x_{fs\to\tau} \wedge e_{fs\to\tau})x.p & \longrightarrow_{\mathcal{M}} & (x_{fs\to\tau})x.p \wedge (e_{fs\to\tau})x.p & \\
(x_{fs\to\tau} \wedge e_{fs\to\tau})a & \longrightarrow_{\mathcal{M}} & (x_{fs\to\tau})a \wedge (e_{fs\to\tau})a & \\
(x_{\tau'\to\tau} \wedge e'_{\tau'\to\tau})e_\tau & \longrightarrow_{\mathcal{M}} & (x_{\tau'\to\tau})e_\tau \wedge (e'_{\tau'\to\tau})e_\tau & \\
(x_{fs\to\tau} \vee e_{fs\to\tau})x.p & \longrightarrow_{\mathcal{M}} & (x_{fs\to\tau})x.p \vee (e_{fs\to\tau})x.p & \\
(x_{fs\to\tau} \vee e_{fs\to\tau})a & \longrightarrow_{\mathcal{M}} & (x_{fs\to\tau})a \vee (e_{fs\to\tau})a & \\
(x_{\tau'\to\tau} \vee e'_{\tau'\to\tau})e_\tau & \longrightarrow_{\mathcal{M}} & (x_{\tau'\to\tau})e_\tau \vee (e'_{\tau'\to\tau})e_\tau &
\end{array}
\tag{2}
$$

$$
\begin{array}{llll}
\lambda x.e_\tau & \longrightarrow_{\mathcal{M}} & \lambda x.e'_\tau & \text{if } e_\tau \longrightarrow^{\star}_{\mathcal{M}} e'_\tau \\
\lambda x_{\tau'}.e_\tau & \longrightarrow_{\mathcal{M}} & \lambda x_{\tau'}.e'_\tau & \text{if } e_\tau \longrightarrow^{\star}_{\mathcal{M}} e'_\tau \\
(x_{\tau'\to\tau}e_\tau) & \longrightarrow_{\mathcal{M}} & (x_{\tau'\to\tau}e'_\tau) & \text{if } e_\tau \longrightarrow^{\star}_{\mathcal{M}} e'_\tau
\end{array}
\tag{3}
$$

$$
\begin{array}{llllll}
false_{fs\to\tau}x.p & \longrightarrow_{\mathcal{M}} & false_\tau & true_\tau \wedge e_\tau & \longrightarrow_{\mathcal{M}} & e_\tau \\
false_{fs\to\tau}a & \longrightarrow_{\mathcal{M}} & false_\tau & e_\tau \wedge false_\tau & \longrightarrow_{\mathcal{M}} & false_\tau \\
false_{\tau'\to\tau}e_{\tau'} & \longrightarrow_{\mathcal{M}} & false_\tau & e_\tau \wedge true_\tau & \longrightarrow_{\mathcal{M}} & e_\tau \\
true_{fs\to\tau}x.p & \longrightarrow_{\mathcal{M}} & true_\tau & false_\tau \vee e_\tau & \longrightarrow_{\mathcal{M}} & e_\tau \\
true_{fs\to\tau}a & \longrightarrow_{\mathcal{M}} & true_\tau & true_\tau \wedge e_\tau & \longrightarrow_{\mathcal{M}} & true_\tau \\
true_{\tau'\to\tau}e_{\tau'} & \longrightarrow_{\mathcal{M}} & true_\tau & e_\tau \vee false_\tau & \longrightarrow_{\mathcal{M}} & e_\tau \\
false_\tau \wedge e_\tau & \longrightarrow_{\mathcal{M}} & false_\tau & e_\tau \wedge true_\tau & \longrightarrow_{\mathcal{M}} & true_\tau
\end{array}
\tag{4}
$$

$$
\begin{array}{llll}
e_\tau & \longrightarrow_\perp & false_\tau & \\
x & \longrightarrow_\mathcal{M} & t & \text{if } x = t \in \mathcal{M} \\
a.p \doteq s & \longrightarrow_\mathcal{M} & false & \\
x.p \doteq s & \longrightarrow_\mathcal{M} & t = s & \text{if } x.p \doteq t \in \mathcal{M} \\
a = b & \longrightarrow_\mathcal{M} & false & \\
t = t & \longrightarrow_\mathcal{M} & true & \\
x = t \wedge e_{bool} & \longrightarrow_\mathcal{M} & x = t \wedge e'_{bool} & \text{if } e_{bool} \longrightarrow^\star_{\mathcal{M} \wedge x = t} e'_{bool} \\
x = t \wedge e_{bool} & \longrightarrow_\mathcal{M} & false & \text{if } \mathcal{M} \wedge x = t \rightarrow \perp \\
x.p \doteq t \wedge e_{bool} & \longrightarrow_\mathcal{M} & x.p \doteq t \wedge e'_{bool} & \text{if } e_{bool} \longrightarrow^\star_{\mathcal{M} \wedge x.p \doteq t} e'_{bool} \\
x.p \doteq t \wedge e_{bool} & \longrightarrow_\mathcal{M} & false & \text{if } \mathcal{M} \wedge x.p = t \rightarrow \perp
\end{array}
\tag{5}
$$

$$
(e_\tau \vee e'_\tau) \wedge e''_\tau \longrightarrow_\mathcal{M} (e_\tau \wedge e''_\tau) \vee (e'_\tau \wedge e''_\tau) \text{ if both } e_\tau \text{ and } e'_\tau \text{ are } \mathcal{M}\text{-dependent with } e''_\tau \tag{6}
$$

We assume that α-reductions will be performed whenever necessary. For simplicity we omitted the rules concerning negation. The rewrite system is divided in six groups, each one dealing with: (1) β-reduction (where $e[d/x]$ denotes the substitution in e of x for d), abstraction and boolean operations for higher order types; this rules are applied before any other rule (2) application and boolean operations (3) rewrite inside abstractions and applications (4) **false** and **true**; (5) feature description of type **bool**, e_{bool}; this rules essentially correspond to the feature constraint rewrite system in [DMV94] (6) distributive law; this rule must apply only when both e_τ and e'_τ have variables in common with e''_τ, eventually through "bindings" in $\mathcal{M}$ [3]. If this last rule is omitted, the rewrite process becomes polynomial although incomplete.

Theorem 2.1 *Given a closed feature description e_τ the rewrite system is correct, terminating and complete in the sense that e_τ is satisfiable unless* **false** *is produced. Moreover the final feature description is in basic normal form.*

For a proof of the above results see [Mor97].

3 Constraint Categorial Grammar

In this section we show how the expressiveness of categorial grammars can be augmented using feature descriptions.

We will use a basic (rigid) categorial grammar (CG), consisting of a set of categories, a lexicon which assigns categories to words and a calculus which determines the set of admissible category combinations. Given a set of basic categories $\mathcal{C}at_0$ we define recursively the set of categories $\mathcal{C}at$ by: the elements of

[3] The notation of $\mathcal{M}$-dependence coincides with the one for complex feature constraints [DMV94], if $x \in c$ means x occurs free in c. Given two constraints c_1 and c_2 and a model $\mathcal{M}$, c_1 and c_2 are $\mathcal{M}$-*dependent* if and only if $Var_\mathcal{M}(c_1) \cap Var_\mathcal{M}(c_2) \neq \emptyset$, where $Var_\mathcal{M}(c)$ is the smallest set satisfying: if $x \in c$, then $x \in Var_\mathcal{M}(c)$; if $x \in Var_\mathcal{M}(c)$ and $x.f \doteq z \in \mathcal{M}$, then $z \in Var_\mathcal{M}(c)$.

$\mathcal{C}at_0$ are categories; if A and B are categories[4] then A/B and $A\backslash B$ are categories. Some unary (lexical) rules (lifting, division, etc) will be added to provide a flexible CG which can cope with discontinuity and other linguistic phenomena. Semantically these rules allow functional abstraction over displaced or missing elements.

A Constraint Categorial Grammar is a tuple $< \mathcal{C}at_0, \Upsilon, Lexicon, Rules >$ where

1. $\mathcal{C}at_0$ is a set of base categories
2. Υ is a map which associates with each category C a type $\Upsilon(C)$ and satisfies

$$\Upsilon(A/B) = \Upsilon(B\backslash A) = \Upsilon(B) \rightarrow \Upsilon(A)$$

3. $Lexicon$ is a set of triples $< w, A, c >$, where w is a word, A a category and c is a feature description of type $\Upsilon(A)$
4. $Rules$ is the set of inference rules to combine pairs $A - c$ of syntactic categories and feature descriptions (semantic representation).

The inference rules used in the current grammars are:

$$(app/)\ \frac{A/B - c_f \quad B - c_b}{A - (c_f c_b)} \text{ if } c_f c_b \text{ is satisfiable}$$
$$(app\backslash)\ \frac{B - c_b \quad B\backslash A - c_f}{A - (c_f c_b)} \text{ if } c_f c_b \text{ is satisfiable}$$

plus a set of unary rules.

3.1 A sample grammar

In figure 1. is given a fragment of an English grammar written in $\mathcal{CCLG}$. We use '\' for 'λ', '&' for '$\wedge$' and '|' for '$\vee$'. All variables are bound and can be any string of letters. The `let` constructor allows the use of macros in the writing of the lexicon. The `transformation` constructor implements unary rules for type raising. Type raising rules are just allowed for some categories and their application is controlled during execution. The `lex` constructor is used for each lexical entry. In this experiment we do not impose any type discipline (HPSG style) in the feature structures themselves[4]. If we assign to each part of speech a feature structure, then an associated feature description will be of type $fs \rightarrow \ldots \rightarrow \mathbf{bool}$. For instance, if we assign the type $fs \rightarrow \mathbf{bool}$ to "John", with semantics $\lambda s.s = john$, and assign the type $fs \rightarrow fs \rightarrow \mathbf{bool}$ to "runs", with semantics $\lambda x.\lambda s.s.reln = run \wedge s.arg1 = x$, the sentence "John runs" would have the type $fs \rightarrow \mathbf{bool}$ and semantics $\lambda s.s.reln = run \wedge s.arg1 = john$. Once more we note that the use of partial descriptions allows us to express directly, the relations between the several constituents. The semantic used is inspired in the ones in [PS87].

[4] Neither the distinction between "syntactic" and "semantic" features is made.

```
Base_Categories           % Define the set of base categories
    s = fs- >bool,        % and their types
    iv = fs- >fs- >bool,
    np = s/iv,
    tv = iv/np,
    dv = tv/np,
    n = fs- >fs- >bool,
    det = np/n,
    pp= fs- > bool;
transformation            % define a type raising rule
    np = (s/np)/(iv/np) : \S \Vt \C. S (Vt C);
%%%%%%%% some useful abbreviations
% agreement specifications
let 3RD_SG = \X. X.pers=p3 & X.nb=sg;
let NOT_3RD_SG = \X. X.pers\=p3 | X.nb\=sg;
let ANY = \X. X=X;
% proper nouns (generalized quantifier type)
let PN(W) = \P.\s. s.quant=exists_one & s.arg.reln=naming
    & s.arg.arg1=W & 3RD_SG(s.arg) & P s.arg s.pred ;
% common nouns (AGR is an agreement)
let CN(W,AGR) = \x\s. s.reln=W & s.arg1=x & AGR x;
%determiners
let DET(Q,AGR) = \N. \P. \s. s.quant=Q & AGR s.var &
    N s.var s.range & P s.var s.scope;
% intransitive verbs
let IV(W,AGR) = \s.\p. p.reln=W & p.arg1=s & AGR s;
% transitive verbs (Obj is the semantics of the object)
let TV(W,AGR) = \Obj. \su.\p. Obj (\o \q. q.reln=W & q.arg1=su & q.arg2=o) p;
let V_PP(W,AGR) = \SS. \su \ s. SS s.arg2 & s.reln=W & s.arg1=su & AGR su;
%ditransitive verbs
let DV(W,AGR) = \Ci. \Cs. \subj. \si. Cs ( \ind. \s. Ci ( \obj \p. p.reln=W &
    p.arg1=subj& p.arg2=obj & p.arg3=ind s) si & AGR subj;
%%%%%%%%%%%%% lexicon
lex a, det, DET(exists_one,3RD_SG); lex every, det, DET(all,ANY);
lex book, n, CN(book,3RD_SG); lex man, n, CN(book,3RD_SG);
lex john, np, PN(john); lex mary, np, PN(mary);
lex died, iv, IV(die,3RD_SG); lex loves, tv, TV(love,3RD_SG);
lex read, tv, TV(read,ANY); lex said, iv/pp, V_PP(say,ANY);
lex gave, dv, DV(give,ANY);
lex that, pp/s, \s.s;
% coordination lex and, s\(s/s), \S1\S2\s. s.type=coord & S1 s.arg1 & S2 s.arg2;
lex and, np\((tv\iv)/np), \NP1\NP2\VT. \subj\s. s.type=coord &
    VT NP1 subj s.arg1 & VT NP2 subj s.arg2;
lex and, iv\(iv/iv), \V1\V2. \subj.\s.
    s.type=coord & V1 subj s.arg1 & V2 subj s.arg2;
lex and, np\(np/np), \NP1\NP2\VT\s. s.type=coord &
    NP1 VT s.arg1 & NP2 VT s.arg2;
```

Fig. 1. Sample grammar

Processing $\mathcal{CCLG}$ is implemented in Prolog augmented with the constraint solver for feature descriptions[5]. In this section we briefly describe this implementation. Although the feature descriptions used in the grammar are untyped, a type inference algorithm is used to infer types for each expression. Moreover, for each lexical entry the type of the feature description is checked with that of the category and whenever possible the *normal form* of the feature description is computed. The inference rules are build-in in the grammar processor. Currently, we use a bottom-up chart parser that builds a context-free backbone. Each edge is a (Prolog) term $arc(Begin, End, Cat, Sref)$ where Cat is the category spanning from $Begin$ to End and $Sref$ is the information to be used to extract the semantic representation, and that reflects how this edge was formed: if it was a lexical entry $Sref$ is a reference to it; if it results from a left (right) application rule, it is a pair of references for its daughters; if it results from a unary rule, it is a pair of references to the initial category and to that rule. When the parse trees are successful built, the semantic representation is extracted and the constraint solver applied. These two components can be interleaved in order to prune, as soon as possible, inconsistent edges. As is apparent from the sample grammar (figure 1.) the semantic representations can become very cumbersome to write and visualize. So a graphical "workbench", based on a Tcl/Tk interface to Yap Prolog, was provided to edit grammars and lexicon, as well as to visualize the parse trees and semantic representations (as matrix boxes).

An Example As an example we analyze the parsing of the sentence "`a man said that john read a book and mary died`". There are two possible parse trees of this sentence, one with the coordination in the scope of the relative clause and other with a wider scope. The semantic representation of this sentence will be a feature description `\x_1.X1|X2` where `X1` and `X2` are partially represented in figures 2. and 3.. Figures 4. and 5. show the semantics of the sentences "`john read a book`" and "`mary died`", respectively. In the feature description `X2` (figure 3.) the former semantics is identified with the value of `x_1.arg1.scope.arg2` and the latter is identified with the value of `x_1.arg2`. In the feature description `X1` (figure 2.) the value of `x_1.scope.arg2` is the feature structure corresponding to the coordination of the these two sentences. As remarked in the previous section, the parsing process first builds a parse forest using only the categories of lexical items and the inference (and unary) rules for syntactic categories. The parse tree of sentence we are considering is too large to be considered here, so figure 6. shows only the parse forest of "`john read a book`". In the first row we have the syntactic categories of each lexical item (given in the lexicon or derived by a unary rule). In the following rows each entry corresponds to the possible ways of deriving a category spanning a portion of the input sentence. For instance, the category `iv` can be derived in the third row from `iv/(s/iv)` and `s/iv`, spanning "`read a book`". Then for each parse tree that spans the whole sentence with root category s, the semantic representation of the constituents

[5] So it can be seen as an instance of $\mathcal{CLP}(\Lambda_{\mathcal{FD}})$.

are combined and if the constraint solver does not produce **false**, a semantic representation is derived.

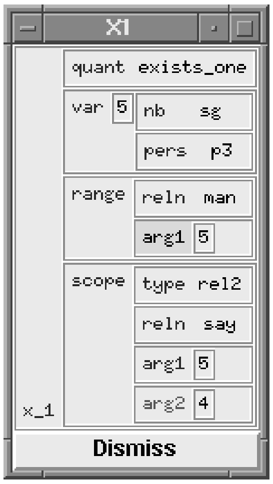

Fig. 2. Coordination inside relative

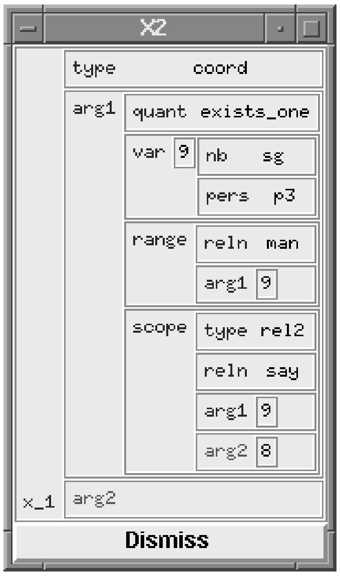

Fig. 3. Coordination wider scope

4 Final Remarks

The current implementation of $\mathcal{CCLG}$ shows the practical feasibility of using higher order feature structure descriptions as semantic representations. This reflects the fact that the complexity of the satisfiability problem for higher order feature descriptions is essentially the same as for feature logics. We should also point out that the good performance of the system results in part from its hybrid nature where a categorial grammar with atomic base categories is used to guide parsing. Some more toy English grammars where written that can handle some kinds of discontinuity, modifiers and quantifier scope. However, the introduction of a type discipline and more general treatment of recursive lexical rules ([BvN94]) must be considered, in future work. On the other hand, most recent developments of categorial grammars are based on the Lambek calculus [Lam58, Moo88, Mor94] (an intuitionist fragment of Linear Logic). Some implementations for the propositional fragment are based on chart parsers [Kön94, Hep92] and we conjecture that $\Lambda_{\mathcal{FD}}$calculus can be successfully used in such a systems, for process semantic representations. From an implementational

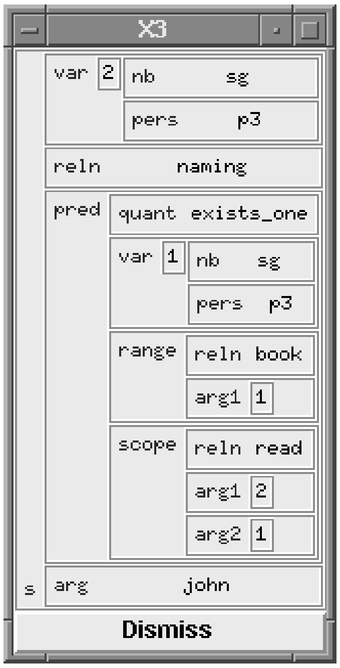

Fig. 4. Semantics of "john read a book".

Fig. 5. Semantics of "mary died".

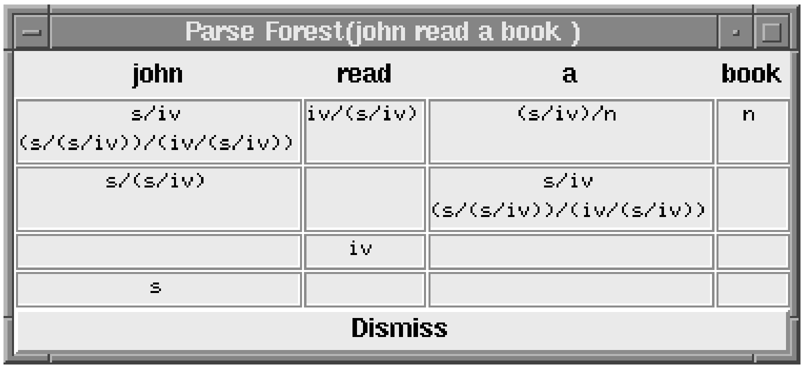

Fig. 6. A parse forest

perspective it would be helpful to study how current techniques employed in functional programming implementations, namely the use of combinators [BD95], can be imported for improve the computation of β-reductions.

Acknowledgments The authors would like to thank Sabine Broda and the anonymous reviewers for their valuable comments on an earlier draft of this paper.

References

[BD95] Sabine Broda and Luís Damas. A new translation algorithm from lambda calculus into combinatory logic. In Nuno Mamede and Carlos Pinto-Fereira, editors, *Proceedings of the 7th Portuguese Conference on Artificial Intelligence (EPIA 95)*, volume 990 of *Lecture Notes in Artificial Intelligence*, pages 359–370. Springer-Verlag, 1995.

[BvN94] Gosse Bouma and Gertjan van Noord. Constraint-based categorial grammars. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, 1994.

[DMV94] Luís Damas, Nelma Moreira, and Giovanni B. Varile. The formal and computational theory of complex constraint solution. In C. Rupp, M. A. Rosner, and R. L. Johnson, editors, *Constraints, Language, and Computation*, Computation in Cognitive Science, pages 149–166. Academic Press, London, 1994.

[DSP91] Mary Dalrymple, Stuart M. Shieber, and Fernando C. N. Pereira. Ellipsis and higher-order unification. *Linguistics and Philosophy*, 14:399–452, 1991.

[DV92] Luís Damas and Giovanni B. Varile. On the satisfiability of complex constraints. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, Nantes, France, 1992.

[Hep92] Mark Hepple. Chart parsing Lambek grammars. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, pages 134–140, Nantes,France, 1992.

[Kön94] Esther König. A hypothetical reasoning algorithm for linguistic analysis. *Journal of Logic and Computation*, 1994. to appear.

[Lam58] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958. Reprinted in: Buszkowski, W., W. Marciszewski, and J. van Benthem (eds): Categorial Grammar. Amsterdam, 1988.

[Mah88] Michael J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. Technical report, IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, U.S.A., 1988.

[Moo88] Michael Moortgat. *Categorial Investigations: Logical and Linguistic Aspects of Lambek Calculus.* Foris, Dordrecht, 1988.

[Mor94] Glyn Morrill. *Type Logical Grammar:Categorial Logic of Signs.* Kluwer Academic Publishers, Dordrecht, 1994.

[Mor97] Nelma Moreira. *Formalismos e tcnicas de implementao de gramticas lgicas com restrie.* PhD thesis, Faculdade de Cincias da Universidade do Porto, 1997. to appear.

[NM88] Gopalan Nadathur and Dale Miller. An overview of λProlog. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, volume 1, pages 810–827, Seattle, WA, 1988. MIT Press.

[Per90] Fernando C.N. Pereira. Categorial semantics and scoping. *Computational Linguistics*, 16(1), 1990.

[PS87] Fernando C.N. Pereira and Stuart M. Shieber. *Prolog and Natural Language Analysis.* Center for the Study of Language and Information Stanford, 1987.

[Smo89] Gert Smolka. Feature constraint logics for unification grammars. Technical report, IBM Wissenschafliches Zentrum, Institut für Wissensbasierte Systeme, 1989. IWBS Report 93.

[ZKC87] Henk Zeevat, Ewan Klein, and Jo Calder. Unification categorial grammar. In Nicholas Haddock, Ewan Klein, and Glyn Morrill, editors, *Categorial Grammar, Unification Grammar and Parsing*. Centre for Cognitive Science, University of Edinburgh, 1987. Volume 1 of Working Papers in Cognitive Science.

This article was processed using the LaTeX macro package with LLNCS style