March 1990

Extending Definite Clause Grammars With Scoping Constructs

Remo Pareschi
ECRC

Dale Miller
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-20.

Extending Definite Clause Grammars With Scoping Constructs

Abstract

Definite Clause Grammars (DCGs) have proved valuable to computational linguists since they can be used to specify phrase structured grammars. It is well known how to encode DCGs in Horn clauses. Some linguistic phenomena, such as *filler-gap dependencies*, are difficult to account for in a completely satisfactory way using simple phrase structured grammar. In the literature of logic grammars there have been several attempts to tackle this problem by making use of special arguments added to the DCG predicates corresponding to the grammatical symbols. In this paper we take a different line, in that we account for filler-gap dependencies by encoding DCGs within *hereditary Harrop formulas*, an extension of Horn clauses (proposed elsewhere as a foundation for logic programming) where implicational goals and universally quantified goals are permitted. Under this approach, filler-gap dependencies can be accounted for in terms of the operational semantics underlying hereditary Harrop formulas, in a way reminiscent of the treatment of such phenomena in Generalized Phrase Structure Grammar (GPSG). The main features involved in this new formulation of DCGs are mechanisms for providing scope to constants and program clauses along with a mild use of λ-terms and λ-conversion.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-20.

Extending Definite Clause Grammars With Scoping Constructs

MS-CIS-90-20
LINC LAB 167

Remo Pareschi
Dale Miller

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

March 1990

Proceedings of the Seventh International Conference on Logic Programming edited by David D.H. Warren and Peter Szeredi

Extending Definite Clause Grammars with Scoping Constructs

Remo Pareschi
ECRC, Arabellastrasse 17
D-8000 Munich 81, West Germany
remo@ecrc.de

Dale Miller
Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104–6389 USA
dale@cis.upenn.edu

Abstract

Definite Clause Grammars (DCGs) have proved valuable to computational linguists since they can be used to specify phrase structured grammars. It is well known how to encode DCGs in Horn clauses. Some linguistic phenomena, such as *filler-gap dependencies*, are difficult to account for in a completely satisfactory way using simple phrase structured grammar. In the literature of logic grammars there have been several attempts to tackle this problem by making use of special arguments added to the DCG predicates corresponding to the grammatical symbols. In this paper we take a different line, in that we account for filler-gap dependencies by encoding DCGs within *hereditary Harrop formulas*, an extension of Horn clauses (proposed elsewhere as a foundation for logic programming) where implicational goals and universally quantified goals are permitted. Under this approach, filler-gap dependencies can be accounted for in terms of the operational semantics underlying hereditary Harrop formulas, in a way reminiscent of the treatment of such phenomena in Generalized Phrase Structure Grammar (GPSG). The main features involved in this new formulation of DCGs are mechanisms for providing scope to constants and program clauses along with a mild use of λ-terms and λ-conversion.

1 Introduction

Logic programming and natural language processing have in the past been fruitfully indebted to each other. Indeed, Prolog was born in the early seventies as a result of Alain Colmerauer's efforts to create a programming environment suitable both for natural language processing and deductive question-answering. In the early eighties Pereira and Warren [26] gave a

rigorous definition of the framework of Definite Clause Grammars (DCGs), which is directly motivated by the possibility of encoding phrase-structure grammars as Prolog programs. DCGs represent a fundamental contribution to the formalization of linguistic theories for computational purposes, and to the idea that grammatical formalisms can be viewed as programming languages.

In the meanwhile, the development of linguistic theories has been steadily evolving and new notations have been devised that aim to ease the task of providing natural representations for complex linguistic phenomena. From this point of view, there has been a substantial rift between computational linguists, who have striven to account for complex natural language phenomena via the clever exploitation of the Horn-based formulation of DCGs, and theoretical linguists, who have endeavored to refine phrase-structure grammar notation to meet more fully the needs of natural language analysis. Quite revealing of this situation is the case of *filler-gap dependencies* (characterizing such crucial natural language constructs as questions and relative clauses): in the tradition of logic grammars, this has led to the use of special arguments occurring inside DCG predicates [4, 24, 25]; in theoretical linguistics, it has produced, among other developments, the novel notion of *slashed non-terminal* in the framework of Generalized Phrase Structure Grammar (GPSG) [7, 8] — a notion which is hard to directly account for within the expressive boundaries of Horn-based DCGs.

We aim to show that the treatment of filler-gap dependencies in GPSG can be reconciled with the tradition of logic grammars if DCGs are based on a extension of Horn clause logic that contains a mechanism to handle *local information*; for such is the kind of information typically involved in the GPSG notion of slashed non-terminal, where a certain phrasal node is marked (i.e. slashed) with respect to an internally missing subnode (a gap). We carry out our intent by recasting DCGs within an extension of Horn clauses, know as *hereditary Harrop formulas*, where implicational goals and universally quantified goals are permitted. These formulas have been proposed elsewhere as a foundation for logic programming [19] and have found applications in other areas of logic programming where local information is involved, such as modules and information hiding [15, 16]. Under our approach, filler-gap dependencies can be directly handled in terms of the operational semantics underlying hereditary Harrop formulas in a way reminiscent of the GPSG treatment of such phenomena. The main features involved in this new formulation of DCGs are mechanisms for providing scope to constants and program clauses along with a mild use of λ-terms and λ-conversion.

This paper is organized as follows. In Section 2, *hereditary Harrop formulas* are briefly described. Section 3 compares the formalisms of *Definite Clause Grammars (DCG)* and *Generalized Phrase Structured Grammars (GPSG)*. Section 4 argues that if the logic programming language is based on hereditary Harrop formulas then a GPSG-like treatment of filler-gap dependencies can be naturally implemented. Section 5 hints at how

richer linguistic behavior than those of the examples in Section 4 may be accounted. Finally, Section 6 briefly reviews related work.

2 Hereditary Harrop Formulas

The class of *hereditary Harrop formulas* has been proposed as the foundation of a logic programming language that contains positive Horn clauses as a sublanguage and for which a natural, goal-directed proof procedure exists [16, 19]. Other very similar proposals to extending Horn clause logic have also been proposed: in particular, N-Prolog [6] and the intuitionistic clausal system of [2, 13, 14]. Below we develop the central aspects of our extension in two steps. First, we add implications and universal quantifiers to the body of program clauses, and then we add some weak notions of function variable quantification and λ-terms. The logic programming language described below is a subset of the λProlog language [21].

2.1 Implications and Universal Quantifiers in the Body of Clauses

Positive Horn clauses can be described using three syntactic categories: A for atomic formulas, G for goal formulas, and D for Horn clauses (definite clauses). The latter two are defined as follows:

$$\begin{array}{lll} G & ::= & A \mid G_1 \wedge G_2 \\ D & ::= & A \mid G \supset A \mid D_1 \wedge D_2 \mid \forall x\ D. \end{array}$$

If disjunctions and existential quantifiers are permitted within goal formulas, the resulting class of formulas has properties very similar to those of Horn clauses. It is also possible to restrict D-formulas so that the only conjunctions in the scope of a universal quantifier are within goal formulas (the body of clauses). We now extend the definitions for G and D to the point where they are mutually recursive:

$$\begin{array}{lll} G & ::= & A \mid G_1 \wedge G_2 \mid D \supset G \mid \forall y\ G \\ D & ::= & A \mid G \supset A \mid D_1 \wedge D_2 \mid \forall x\ D. \end{array}$$

We have the following symmetry: the negative subformulas of D- (resp. G-) formulas are G- (D-) formulas, and the positive subformulas of D- (resp. G-) formulas are D- (G-) formulas. D-formulas are also called *hereditary Harrop* formulas (the definition in [19] permits the mild extension of allowing disjuncts and existentials in goals).

To describe the operational (proof-theoretic) semantics of implications and universal quantifiers in goals, let Σ be a set of (non-logical) constants (such sets are called *signatures*) and let $\mathcal{P}$ be a set of closed D-formulas all of whose non-logical constants are members of Σ (in which case we say that $\mathcal{P}$ is *over* Σ). The goal $D \supset G$ (assumed to be over Σ) follows from the pair

Σ and $\mathcal{P}$ if the goal G follows from the pair Σ and $\mathcal{P} \cup \{D\}$. The goal $\forall y\ G$ follows from the pair Σ and $\mathcal{P}$ if the goal $[y := c]G$ follows from the pair $\Sigma \cup \{c\}$ and $\mathcal{P}$, where c is a "new" constant, that is, one that is not a member of Σ. An atom A follows from Σ and $\mathcal{P}$ if there is some clause in $\mathcal{P}$ that has an instance using terms only over Σ of the form A or of the form $G \supset A$ where G follows from Σ and $\mathcal{P}$. Such an interpretation of implication and universal quantification is sound and complete with respect to intuitionistic logic, but not with respect to classical logic.

A universally quantified goal formula is interpreted *intensionally*: that is, to prove the goal $\forall x\ G$, an attempt to prove the generic instance $[x := c]G$ is required. Notice that if a universal goal is proved with respect to the signature Σ in this way, then $[x := t]G$ will be provable for all terms t over Σ: the intensional interpretation of the universal quantifier implies the *extensional* interpretation. The converse is, however, not the case. For example, from the signature $\{p, a, b\}$ and program $\{p(a), p(b)\}$, the goal $\forall x\ p(x)$ follows extensionally but not intensionally. Scoped constants are often called *eigenvariables* in the context of natural deduction and sequential proof calculi.

To implement hereditary Harrop formulas using the standard techniques of free (logic) variables and unification, a few changes to the standard techniques are required. First, free variables may appear within programs, as is the case if the goal $D \supset G$ contains a variable that is free in D. When D is added to the program, that variable must remain free. Subsequent substitutions for that variable would need to be applied to both the goal *and* the program. Second, the intensional interpretation of unification requires that unification be modified. Let Σ be a signature and let $\mathcal{P}(\bar{x})$ be a program and let $\forall y\ G(y, \bar{x})$ be a goal formula both of which are over Σ and whose free variables are members of the list of variables $\bar{x}$. Consider trying to find a substitution for the variables $\bar{x}$ so that $\forall y\ G(y, \bar{x})$ follows from Σ and $\mathcal{P}(\bar{x})$. Here, the substitution instances of $\bar{x}$ must be terms over Σ. This goal is provable if the goal $G(c, \bar{x})$ (where c is a constant not in Σ) is provable form $\Sigma \cup \{c\}$ and $\mathcal{P}(\bar{x})$. It might be possible for unification to suggest that a variable in $\bar{x}$ be instantiated with a term containing c, as in the case $\forall y\ (X = y)$ (assuming the usual clause for equality). In that case, unification should generate a failure. Free variables arising from backchaining over program clauses that appear later in the construction of a proof of $G(c, \bar{x})$ may, however, be instantiated with terms that contain c. Thus, different free variables can be replaced with terms over different signatures. There are several ways to modify unification so that these different restrictions on free variables are obeyed. See [16, 21] for two different approaches.

In presenting D-formulas, we shall use the usual Prolog convention of writing `:-` to denote the converse of $\supset$ and of not writing outermost universal quantification at all. In presenting G-formulas, we shall use a comma for conjunction, `=>` for implication, and `all x,y,z\` to denote the universal quantification of the variables `x,y,z`. Free variables are denoted by tokens with an initial capital letter.

2.2 Discharging a Constant from a Term

Assume that `append/3` is axiomatized in the usual way and let Σ be a signature containing at least the constants `append`, `[_|_]`, `[]`, `a`, `b`. Consider the problem of finding a substitution term over Σ for the free variable `X` so that the goal formula `all y\ append([a,b],y,X)` is provable. Proving this goal first reduces to proving `append([a,b],k,X)` (`k` not a member of Σ). This goal is provable if `X` can be unified with `[a,b|k]`. This will fail, however, since `X` can be instantiated with terms over Σ but not over $\Sigma \cup \{\mathtt{k}\}$. Unification failure here is quite sensible since the value of `X` should be independent of the choice of the constant used to instantiate `y`. It might be very desirable, however, to have this computation succeed if we could, in some sense, abstract away this particular choice of constant. This is possible if we are willing to admit some forms of λ-abstraction into our logic.

Consider, for example, proving the goal `all y\ append([a,b],y,H(y))` where `H` is a functional variable that may be instantiate with a λ-term whose constants are again from the set Σ. Assume that we instantiate `y` again with the constant `k`. The important unification is now `H(k)` with `[a,b|k]`. There are two λ-terms (up to λ-conversion) that when substituted for `H` into `H(k)` and then λ-normalized yield `[a,b|k]`, namely the terms `w\[a,b|k]` and `w\[a,b|w]` (we shall use `\` as an infix symbol to denote λ-abstraction). Since `H` cannot contain `k`, only the second of these possible substitutions will succeed in being a legal solution for this goal. Notice that the choice of constant to instantiate the universal quantifier in this goal is not reflected in this answer substitution. In a sense, the λ-term `w\[a,b|w]` is the result of *discharging* the constant `k` from the term `[a,b|k]`. Notice, however, that discharging a first-order constant from a first-order term is now a "second-order" term: it can be used to instantiate a function variable.

We shall now permit hereditary Harrop formulas to contain universal quantifiers over functional variables of second-order, that is, variables that can be applied to first-order terms. We shall also permit explict λ-abstractions to be parts of terms.

2.3 Is this a Higher-Order Extension?

The term *higher-order* can be used in many situations. A proof theorist would consider a logic higher-order only if it contained quantification over predicate variables. In this sense, the extension motivated above is not higher-order since it requires the presence of function and not predicate variables. On the other hand, first-order unification is not enough to implement the logic we have outlined. The unification of simply typed λ-terms (sometimes called *higher-order unification*) will, however, be adequate to implement this logic correctly. In this sense, all the features that are needed in this paper are available in the logic programming language λProlog [21] since it contains unification of λ-terms as a computational mechanism.

The unification of simply typed λ-terms in its full generality is a very

rich and costly operation. The examples in this paper will, however, make very little use of the full power of such unification. In fact, in the examples below, unification will generally yield unique answer substitutions since most non-first-order unification problems arise when discharging constants from terms. As the above example shows, the possibility of multiple unifiers, which exists when unifying λ-terms, can be restricted sufficiently if used in conjunction with scoped constants so that only single, most general unifiers exist. For an analysis of a non-first-order subset of λProlog that always has most general unifiers and contains most of the examples of this paper, see [17]. Although this logic requires second-order variable quantification and λ-conversion, it is only a mild extension to first-order unification: much of the richness and costs of the higher-order logic programming language λProlog are not present here. We shall not attempt to outline precisely the subset of λProlog or *higher-order hereditary Harrop formulas* [19] we shall need in this paper. The examples here are particularly simple but they lead into richer examples when additional linguistic phenomena are addressed. See [5] for applications of roughly this same extension of logic programming to theorem proving and see [18, 27] for discussions on the role of λ-terms and function quantification in logic programming implementations of natural language programs.

3 Generalized Phrase Structure Grammars and Definite Clause Grammars

Definite Clause Grammars (DCGs) were introduced by Pereira and Warren [26], their direct ancestry being traceable to Colmerauer's more complex framework of Metamorphosis Grammars [3]. The basic insight behind DCGs is that grammatical formalisms encoded as rewrite systems can be translated into sets of definite clauses. Each non-terminal symbol in the original grammar corresponds in the DCG notation to a predicate taking as arguments a pair of *string positions*, plus other optional arguments. (In practice, implementations of DCGs adopt a sugared notation, which we shall not follow here, where the string positions arguments are omitted.) Parsing can then be viewed as that restricted kind of theorem proving that takes place within logic programming systems.

3.1 Definite Clause Grammars and Phrase Structure Grammars

An immediate and well-known application of DCGs is in translating phrase structure grammars into logical notation. This is also an application that is of particular interest to us here since Generalized Phrase Structure Grammar (GPSG) is itself a variation of phrase structure grammar, and the purpose of this paper is in showing how to extend the DCG framework in terms

S	→	*NP VP*	*VP*	→	*TV NP*
VP	→	*STV S_BAR*	*S_BAR*	→	**that** *S*
NP	→	*PN*	*PN*	→	**Kay**
PN	→	**Fred**	*PN*	→	**Paul**
TV	→	**loves**	*TV*	→	**married**
STV	→	**believes**			

Figure 1: Example of phrase structure grammar

```
s(P1, P2) :- np(P1, P0), vp(P0, P2).
vp(P1, P2) :- tv(P1, P0), np(P0, P2).
vp(P1, P2) :- stv(P1, P0), sbar(P0, P2).
sbar([that|P1],P2) :- s(P1,P2).
np(P1, P2) :- pn(P1, P2).
pn([kay|L], L).
pn([fred|L], L).
pn([paul|L], L).
tv([loves|L], L).
tv([married|L], L).
stv([believes|L], L).
```

Figure 2: A DCG encoding the phrase structure grammar in Figure 1

of hereditary Harrop formulas so as to accomodate some of the features of GPSG.

As an example of translation of a simple phrase structure grammar into a set of definite clauses, consider the grammar in Figure 1 [1]. We can translate this grammar into Horn clauses as in Figure 2 by mapping its non-terminals into two-place predicates taking as arguments string positions. Strings are encoded as lists and string positions are represented in terms of the portion of the list they identify and of the substring which follows it, according to the familiar "difference-list" notation.

3.2 Syntactic Categories in GPSG

GPSG extends simple phrase-structure grammar by providing a more sophisticated notion of syntactic category. For our purposes, we shall make

[1] Throughout this paper, we shall adopt the following more or less standard conventions for labels of syntactic categories: *S* stands for the category of sentences, *S_BAR* for that of *complement* clauses, e.g. sentences prefixed by the complementizer **that**, and *REL* for that of relative clauses; *NP* stands for the category of noun phrases and *PN* for that of proper names; *VP* stands for the category of verb phrases, *TV* for that of transitive verbs, and *STV* for the category of verbs taking as arguments sentence complements.

use of the early version of GPSG assumed, for instance in [7], rather than the more complex, "principle-based" one of [8]. We can summarize the main aspects of GPSG as follows:

(*i*) A GPSG category augments the bare non-terminals of simple phrase structure grammars with *morpho-syntactic* information, i.e., information concerning parts of speech, inflection, case, and agreement. Such information is encoded in terms of *features*, i.e. attribute-value pairs of the form **[number sg]**.

(*ii*) GPSG categories are allowed in slashed form, that is, in the form X/Y. Such categories denote a category X with an internal gap (i.e., a missing constituent) of category Y. In this way GPSG can elegantly account for *filler-gap dependencies*[2]. Examples of filler-gap dependencies are given by sentences such as

Fred loves the woman whom [Paul married $\uparrow$]
Fred loves the woman whom [Kay believes that Paul married $\uparrow$]

In these examples, the relative pronoun *whom* acts as a filler for the sentence fragments *Paul married* and *Kay believes that Paul married*, characterized by a missing noun-phrase. (We have indicated with an upward-looking arrow the position where the gap occurs.) A possible rule to account for the relative clause in the examples above can be stated as

$$REL \rightarrow \textbf{whom} \quad S/NP$$

That is, one form of a relative clause is the word "whom" followed by a sentence with an NP gap.

(*iii*) GPSG states explicitly how to build the logical form for a given string via rules of semantic interpretation which come in pairs with the syntactic rules. Such semantic rules are inspired by Montague's principle of compositionality [20] and view the interpretation of a sentence as obtained from the combination of the interpretations of its subconstituents, where the method of combination is given by functional application and λ-reduction. Thus, the rule in (*ii*) can be paired with a rule of semantic interpretation as follows:

$$REL \rightarrow \textbf{whom} \quad S/NP \qquad S/NP'$$

(The prime notation " $'$ " refers here to the semantic counterpart of a given syntactic category.) This pairing provides the information that the semantic interpretation of a relative clause is given by the semantic interpretation of the sentence where the gap occurs.

[2] *Filler-gap dependencies* are also known in the GPSG literature as *unbounded dependencies* because they can involve unbounded portions of a phrase-structure tree.

Now, it is well-known that an augmentation of phrase structure grammars of the kind described in (*i*) can be implemented in DCGs by adding to predicates extra-arguments corresponding to morpho-syntactic features. For this reason, we shall not be further concerned with it here. On the other hand, categories of the kind described in (*ii*) do not have a natural translation within Horn clauses. We show here, however, that they can be naturally translated into hereditary Harrop formulas by viewing a given slashed category as an implication, with the category on the left and the one on the right as, respectively, the consequent and the antecedent in the implication. Therefore, parsing with this kind of grammar can be implemented as theorem proving with hereditary Harrop formulas. Moreover, we provide a natural implementation of (*iii*) by embedding the rules of semantic interpretation into their syntactic counterparts by passing logical forms as extra arguments of non-terminal predicates and exploiting the mechanism of β-reduction. Under our approach, (*ii*) and (*iii*) will subtly interact in the fact that the semantic representations associated with gaps are going to be *scoped constants.*

4 Gap Introduction as Hypothesis Introduction

By using rules such as

$$REL \rightarrow \textbf{whom} \quad S/NP \qquad S/NP'$$

GPSG can analyze a sentence like

Fred loves the woman whom [Kay believes that Paul married $\uparrow$]

in terms of the phrase-structure tree in Figure 3. (The tree refers to the part of the string corresponding to the relative clause.) We show here how one can use hereditary Harrop formulas to define a program to obtain a corresponding proof. In the first place, we augment the rules of the DCG in Figure 2 with semantic arguments encoded as λ-terms. We obtain in this way the set of rules in Figure 4. We then add to these rules a definite clause version of the GPSG rule above. This is obtained as in the following formula simply by interpreting the slash as implication. The semantic representation of the assumed noun phrase will be a scoped constant (introduced, of course, by a universal quantifier), and the semantic representation of the target category will be an abstraction that is the result of discharging that scoped constant.

```
rel([whom|X], Y, REL) :-
   all gap\ (np(Z, Z, gap) => s(X, Y, REL(gap))).
```

Let us refer to this set of rules as $\mathcal{G}$ and let us observe what happens when parsing the relative clause *whom Kay believes that Paul married* by calling the goal

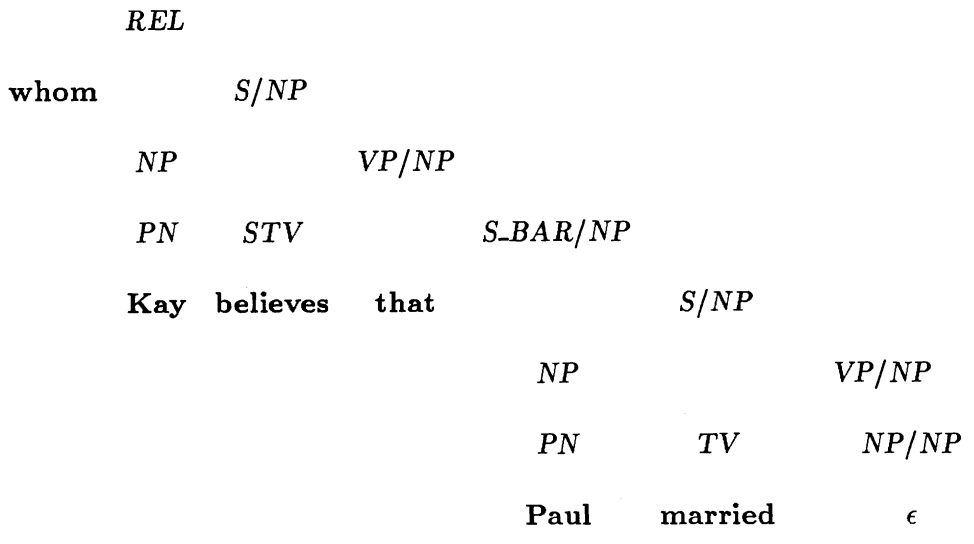

Figure 3: GPSG analysis for **whom Kay believes that Paul married**

```
s(P1, P2, VP(NP)) :- np(P1, P0, NP), vp(P0, P2, VP).
vp(P1, P2, TV(NP)) :- tv(P1, P0, TV), np(P0, P1, NP).
vp(P1, P2, STV(SBAR)) :- stv(P1, P0, STV), sbar(P0, P2, SBAR).
sbar([that|P1], P2, S) :- s(P1, P2, S).
np(P1, P2, PN) :- pn(P1, P2, PN).
pn([kay|L], L, kay).
pn([fred|L], L, fred).
pn([paul|L], L, paul).
tv([loves|L], L, x\y\love(x, y)).
tv([married|L], L, x\y\married(x, y)).
stv([believes|L], L, s\y\believe(y, s)).
```

Figure 4: Augmenting definite-clause rules with semantic arguments

```
?- rel([whom, kay, believes, that, paul, married], [], REL).
```

Backchaining on the rule above yields the goal

```
?- all gap\(np(Z, Z, gap) =>
        s([kay, believes, that, paul, married], [], REL(gap))).
```

Given the intensional reading of universal quantification, we proceed by selecting a new constant, say c, and restrict REL and Z so that they cannot be instantiated with a term containing c. We now attempt to prove the goal

```
?- np(Z, Z, c) =>
     s([kay, believes, that, paul, married], [], REL(c)).
```

This goal succeeds if the goal

```
?- s([kay, believes, that, paul, married], [], REL(c)).
```

follows from the augmented program (grammar) $\mathcal{G} \cup \{$`np(Z, Z, c)`$\}$. It is easy to see that this indeed follows via backchaining and the solving of conjunctive goals. In the course of the proof, the logic variable Z corresponding to the string position of the gap gets instantiated to the empty list [] (signifying that the gap occurred at the end of the given phrase), while the logic variable REL corresponding to the semantic representation of the relative clause gets instantiated to the ground term

```
x\believe(marry(paul, x), kay)
```

by solving the unification problem

```
REL(c) = believe(marry(paul, c), kay).
```

This instantiation provides us with the desired semantic representation for the relative clause. As mentioned in 2.2, the unification problem above admits a second solution, namely, that of instantiating REL to the vacuous λ-term

```
x\believe(marry(paul, c), kay).
```

However, this possibility is here automatically ruled out by the fact that it would violate the restriction on scoped constants.

After we have succeeded in parsing the phrase *Kay believes that Paul married*, the clause `np(Z, Z, c)` (which has now been instantiated to `np([], [], c)`) is removed from the current program and the constant c is similarly removed from the current signature. Thus, the rule

```
rel([whom|X], Y, REL) :-
    all gap\ (np(Z, Z, gap) => s(X, Y, REL(gap))).
```

introduces a noun-phrase gap that can *only* occur within the sentence coming after the relative pronoun whom; all other possibilities are ruled out by the fact that after parsing such a phrase the gap is discharged. Moreover, even within such a phase, the rule above will never be able to locate more than one gap, since the variable corresponding to the string position where the gap occurs is a logic variable. The reader may want to contrast this situation with the one that would be obtained by using the definite clause

```
rel([whom|X], Y, REL) :-
    all gap\ (all z\(np(z, z, gap)) => s(X, Y, REL(gap))).
```

In this case the string position corresponding to the gap would be a universal variable having as scope the atomic definite clause encoding the gap. Clearly, this would imply that more than one gap position could be located by adding such a definite clause to the program. Thus, the contrast between this "too liberal" gap-introducing rule and the former one can be captured in terms of different quantifier scopings.

To summarize the contents of this section, we have provided a very simple and direct logic programming encoding of GPSG-rules for filler-gap dependencies. Under such an encoding, sentences containing gaps are interpreted as normal sentences, with the only difference that their semantic interpretation is characterized by the occurrence of a scoped constant in correspondence of the gap. We are from this point of view on the same line of the GPSG development presented in [7] where sentences containing gaps are also treated in the same way as other sentences, and gaps within sentences correspond to occurrences of *designated variables* in the semantic interpretation. We could in fact say that our use of scoped constants provides a formal characterization of this notion of "designated variable." In the later GPSG development contained in [8] this view of sentences containing gaps was abandoned in favour of a more complicated approach which imposes upon them a special grammatical status. One of the reasons for which the earlier approach was abandoned was indeed a lack of clear understanding of the formal status of the notion of "designated variable" [12].

5 Unused Hypotheses and Constraints over the Distributions of Gaps

We briefly discuss here how to enrich the simple encodings of GPSG-style rules presented in the section above to cope with problems having to do with the distribution of gaps.

We would like to be able to capture in a natural way the following situations:

(i) Natural languages are characterized by many constraints on the distribution of gaps. For instance, gaps occurring in the subject position of embedded sentences are in general forbidden in English; furthermore,

a relative pronoun like *whom* can never act as the filler for a subject gap. Thus, we would like to have a natural way to account for the ungrammaticality of relative clauses such as

*whom [Kay believes that ↑ married Paul]
*whom [Kay believes ↑ married Paul]

(*ii*) We would also like to be able to account for the fact that for every filler there *must* be a corresponding gap. This is not captured by the gap-introducing rule of the section above, since the definite clause encoding the gap may be introduced without ever being used; therefore, we may accept ungrammatical relative clauses such as

*whom Paul married Kay

where there is no gap corresponding to the relative pronoun.

An interesting way to deal with point (*i*) is to introduce at run time (non-atomic) clauses identifying phrase-structure trees "legally" characterized by an empty element in the branch corresponding to the gap, rather than by adding an explicit (atomic) clause encoding the gap itself, as we have done so far. For instance, suppose that we want the gap introduced by *whom* to occur only after transitive verbs; this could be achieved via the following modified rule for relative clauses:

```
rel([whom|X], Y, REL) :-
    all gap\ ((vp(Z, Z1, TV(gap)) :- tv(Z, Z1, TV)) =>
                s(X, Y, REL(gap))).
```

In the course of parsing the sentence containing a noun-phrase gap, this rule permits parsing a verb-phrase whose only children is a transitive verb; that is, we create a verb-phrase node that differs from normal verb-phrase nodes obtained from transitive verbs in the fact that there is no noun-phrase after the verb. Again, this solution is strictly on the same line of the one adopted by GPSG to deal with the problem, where special "gap-locating" rules are used to "terminate" the gap; for instance, the object position after transitive verbs can be made a legal gap site via a GPSG rule such as

$$VP \rightarrow TV \quad NP/NP$$

where NP/NP defines an empty noun-phrase. We could in fact effectively introduce at run-time both a gap and a gap-locating rule as with the following clause:

```
rel([whom|X], Y, REL) :-
   all gap\ (np(Z1, Z1, gap) =>
             (vp(Z, Z1, TV(gap)) :-
                 tv(Z, Z1, TV), np(Z1, Z1, gap)) =>
                                  s(X, Y, REL(gap))).
```

This formula logically implies the one above. Of course, in real applications, we would like to have rules of the form

```
rel([whom|X], Y, REL) :-
    all gap\ (gap_rules[gap] => s(X, Y, REL(gap))).
```

where `gap_rules` refers to a *module* of gap-locating rules containing all the possible legal gap sites for a gap whose filler is the relative pronoun *whom*, and `gap` acts as a *local parameter* in the module. This nicely connects the use we have made of hereditary Harrop formulas in this paper to their application in the field of modular logic programming described in [15, 16].

Point (*ii*) is related to viewing formulas as "limited" resources during computation. It arises from the fact that in Section 2 we have provided the proof-theoretic semantics of hereditary Harrop formulas in terms of the framework of Intuitionistic Logic, where premises can be used an unbounded number of times or not used at all. It is however possible to recast hereditary Harrop formulas in terms of a finer-grained operational semantics, like the one provided by Linear Logic [9]. Under this approach, one can distinguish between formulas which, in the course of the proof, can be used from 0 to many times, and formulas which *must* be used *exactly once*; of course, corresponding syntactic facilities are needed to distinguish between the two cases. Thus, by encoding the premise introduced at run-time to account for the gap as a "must-be-used" formula, cases of "vacuous extraction" like the one mentioned in point (*ii*) are ruled out[3]. Further developments in Linear Logic [10] permit also the possibility of distinguishing the intermediate case of formulas which can be used from 0 to n times, for n fixed; this could turn out to be quite handy for dealing with *parasitic gaps* constructions, characterized by the occurrence of two gaps, one of which is obligatory and the other optional — that is, the clause corresponding to the optional gap would be marked as usable from 0 to 1 times.

6 Related Work

The problem of filler-gap dependencies has received much attention in earlier work in logic grammars, like, for instance, [4, 24]; a good overview of the techniques developed in this tradition can be found in [25]. These attempts rely on Horn clause-based adaptations of the DCG formalism obtained by cleverly exploiting logic programming features; while computationally they serve well their purposes, they are far from possessing the elegance and intelligibility of the GPSG approach. Our own effort on the subject aims at showing that such two requirements — computational implementability

[3]It is interesting to notice that a similar strategy can be applied to the use of hereditary Harrop formulas to model object-oriented programming as in [11], where object states can also be encodable as "restricted-use" formulas; in a rather different operational setting, Linear Logic is for related reasons explicitly assumed as the theoretical background for the object-oriented logic programming language Linear Objects [1].

within a logic programming environment and formal perspicuity — have some reasonable chances to lead a harmonious life together.

Additional discussion on the use of hereditary Harrop formulas to extend DCGs in the direction of theoretical linguistics can be found in [23] (see also [22]). On the use of λ-terms for natural language processing, see [18, 27].

Acknowledgements

We are grateful to Ewan Klein, Mark Steedman, and Jean-Marc Andreoli for helpful discussions, and to the ICLP referees for many helpful comments on the contents of this paper. Most of the research reported here was done while Pareschi was visiting the University of Pennsylvania for the academic year 1987-88. During this visit he was supported in part by a Sloan foundation grant to the Cognitive Science Program, Univ. of Pennsylvania, NSF grants MCS-8219196-CER, IRI-10413 AO2, ARO grants DAA29-84-K-0061, DAA29-84-9-0027 and DARPA grant N00014-85-K0018 to Computer and Information Science. Miller is supported in part by grants ONR N00014-88-K-0633, NSF CCR-87-05596, and DARPA N00014-85-K-0018.

References

[1] Andreoli, J.M., and R. Pareschi. 1990. *Linear Objects: logical processes with built-in inheritance.* In Proceedings of the Seventh International Conference of Logic Programming, Jerusalem, Israel.

[2] Bonner, A. J., L. T. McCarty, and K. Vadaparty. 1989. *Expressing Database Queries with Intuitionistic Logic.* Proceedings of the North American Logic Programming Conference, Cleveland, Ohio. Eds. E. Lusk and R. Overbeek. pp. 831 – 850.

[3] Colmerauer, Alan 1978. *Metamorphosis Grammars.* In Leonard Bolc ed., *Natural Language Communication with Computers.* Springer-Verlag, Berlin, Germany. Reidel, Dordrecht/Boston.

[4] Dahl, V. and H. Abramson. 1984. *On Gapping Grammars.* In Proceedings of the Second International Conference on Logic Programming.

[5] Felty, Amy and Dale Miller 1988. *Specifying Theorem Provers in a Higher-order Logic Programming Language.* In Proceedings of the ninth International Conference on Automated Deduction, Argonne, Illinois

[6] Gabbay, D. M., and U. Reyle. 1984. *N-Prolog: An Extension of Prolog with Hypothetical Implications. I* The Journal of Logic Programming. 1. pp. 319-355.

[7] Gazdar, Gerald. 1981. *Unbounded Dependencies and Coordinate Structure.* Linguistic Inquiry. 12. pp. 155-184

[8] Gazdar, Gerald, E. Klein, G. Pullum, I. Sag. 1985. *Generalized Phrase Structure Grammar.* Blackwell's, Oxford, and Harvard University Press, Cambridge, Mass.

[9] Girard, J. Y. 1987. *Linear Logic.* Theoretical Computer Science. 50:1. pp1-102.

[10] Girard, J. Y., A. Scedrov and P. J. Scott. 1990. *Bounded Linear Logic.* In Proceedings of the Feasible Mathematics Workshop. Cornell University.

[11] Hodas, J., and D. Miller. 1990. *Representing Objects in a Logic Programming Language with Scoping Constructs.* In Proceedings of the Seventh International Conference of Logic Programming, Jerusalem, Israel.

[12] Klein, Ewan 1989. Personal Communication.

[13] McCarty, L. T. 1988. *Clausal Intuitionistic Logic I. Fixed Point Semantics.* The Journal of Logic Programming. 5. pp. 1-31.

[14] McCarty, L. T. 1988. *Clausal Intuitionistic Logic II. Tableau Proof Procedure.* The Journal of Logic Programming. 5. pp. 93-132.

[15] Miller, Dale. 1989. *A Logical Analysis of Modules in Logic Programming.* The Journal of Logic Programming. 6. pp. 79-108.

[16] Miller, Dale. 1989. *Lexical Scoping as Universal Quantification.* In Proceedings of the Sixth International Logic Programming Conference, Lisboa, Portugal. Eds. G. Levi and M. Martelli. pp. 268 – 283.

[17] Miller, Dale. 1989. *A Logic Programming Language with λ-Abstraction, Function Variables, and Simple Unification: Extended Abstract.* Unpublished, September 1989.

[18] Miller, Dale and Gopalan Nadathur. 1986. *Some Uses of Higher-order Logic in Computational Linguistics.* In Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University.

[19] Miller, Dale, Gopalan Nadathur, Frank Pfenning and Andre Scedrov. 1989. *Uniform Proof Systems as a Foundation for Logic Programming* To appear in the Annals of Pure and Applied Logic.

[20] Montague, Richard. 1974. *Formal Philosophy* Yale University Press, New Haven, Connecticut.

[21] Nadathur, Gopalan and Dale Miller. 1988. *An Overview of λ-Prolog.* In Proceedings of the Fifth International Logic Programming Conference, Seattle.

[22] Pareschi, Remo. 1988. *A Definite Clause Version of Categorial Grammar.* In Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics, University of Buffalo.

[23] Pareschi, Remo. 1989. *Type-driven Natural Language Analysis.* PhD Dissertation. University of Edinburgh.

[24] Pereira, Fernando. 1981. *Extraposition Grammars.* Computational Linguistics. 7.

[25] Pereira, Fernando C. N. and Stuart M. Shieber. 1987. *Prolog and Natural Language Analysis.* CSLI Lectures Notes No. 10. CSLI, Stanford University.

[26] Pereira, Fernando C. N. and David H. D. Warren. 1980. *Definite Clauses for Language Analysis.* Artificial Intelligence. 13. pp. 231-278.

[27] Warren, David S. 1983. *Using λ-Calculus to Represent Meaning in Logic Grammars.* Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics, pp. 51 – 56.