CLG(n): Constraint Logic Grammars

Sergio BALARI
Eurotra Espana, 25 Avda. de Vallvidrera
E-08017 Barcelona
Giovanni B. VARILE
CEC, Jean Monnet Bldg., L-2920 Luxembourg

Luis DAMAS
Nelma MOREIRA
Universidade do Porto,Campo Alegre 823
P-4000 Porto

Abstract: CLG(2) is the latest member of a family of grammar formalisms centered around the notion of complex constraint expression for describing phrasal and lexical information and principles of language. Complex constrains can be expressed in a slightly restriced form of first order predicate logic, which makes CLG(2) well suited for expressing, amongst others, HPSG-style of grammars. A sound implementation of the formal semantics of CLG(2) is achieved by resorting to delayed evaluation of non equational constraints.

Introduction

Recently a number of formalisms for grammatical description have been proposed with the aim of overcoming the expressive deficiencies of simple unification based formalisms like PATR-II. Except for the more simple, although not unproblematic, extensions to PATR-II like the ones proposed by Karttunen (1984), most of these efforts have their root in the work by Rounds, Kasper and Moshier (Rounds & Kasper, 1986; Moshier & Rounds 1987), who give the proof of the existence of a sound, although non classical, logical interpretation for disjunctive and negative feature specifications.

Although Kasper (1987) has proposed an algorithm for handling grammars with disjunctive feature specifications, the computational tractability of complex constraints in unification based formalism remains an open problem (Pereira 1987).

Furthermore since the introduction of Head Phrase Structure Grammar (HPSG) (Pollard & Sag, 1987) the desirability of complex constraint expression has become clear. The attractiveness of HPSG-style grammatical description has made classical first order logic the candidate par excellence for constraint expression, modulo the problem of computational tractability.

Since 1988 we have been engaged in the design and implementation of a number of prototype formalisms sharing essentially the same constraint language, a slightly restricted form of first order predicate logic including explicit quantification (Damas & Varile 1989). In trying to achieve a logically sound and practical implementation, our work has been influenced by the CLP paradigm in logic programming (Jaffar & Lassez, 1988) especially our delayed evaluation scheme which amongst other things avoids systematic computations of normal forms of constraints. All CLG(n) prototypes have been implemented in Prolog using the YAP compiler developed at the University of Porto. The results to date have been encouraging. Two of the CLG(n) prototypes have undergone extensive testing with non-trivial grammars of several European languages as reported in (Damas & Varile 1989).

In this paper we present CLG(2), the latest prototype of the CLG(n) family, a formalism which was influenced by the HPSG grammar model. Although different members of the family differ with respect to a number of characteristics like the structure of grammatical description and the data structures defined by the formalism, they all share the same complex constraint language.

1. System Overview

In CLG(2) the data types defined are variables, constants, typed feature structures, list and sets of typed feature structures. Typed feature structure can be seen as directed graphs with labelled arcs, every node being indexed with its type name.

The main novel feature of CLG(2), and of the other members of the CLG(n) family, is its constraint language L, a slightly constrained form of first order predicate logic, including explicit quantification. Unification remains the sole building operation, under the control of complex constraints.

The logical symbols of the complex constraint language consist of variables, constants, the logical connectives & (conjunction), | (disjunction), ~ (negation), -> (material implication), <-> (logical equivalence), the binary predicate symbol "=" and non-logical function and predicate symbols.

The terms of the constraint language are variables, constants and path expressions. The atomic formulae are either equational constraints, i.e. formulae of the form t1=t2 for terms t1,t2, or r(t1,t2,...) for terms ti and relation symbols r. The complex constraints of CLG(2) are the non atomic well formed formulae of L, defined in the usual way: for well formed formulae (constraints) C1, C2 and variable X:

- ~ C1
- C1 -> C2
- C1 & C2
- C1 | C2
- forall(X,S) C1
- exist(X,S) C1

are also well formed formulae (constraints). The S in the quantified constraints are used to restrict the domain of the quantification and can be omitted. The interpretation of the constraint language L is the standard interpretation of first order predicate logic. In other words, we do not resort to intuitionistic or other non-standard interpretations, like for instance Moshier & Rounds (1987). Examples of constraints are:

```
S.syn.local.head ~= n
forall(C:compl_dtrs) C.syn.local.head.maj = n.
```

In order to facilitate the statement of constraints, a macro facility is available in all members of the CLG(n) family, which is a generalization of PATR-II templates in that it can take a list of formal parameters. In CLG(2) this facility has been extended in a fashion akin to UD (Johnson & Rosner, 1989) to include recursive user defined relations. An example of such a relation is:

```
discharge(E,E:L,L);
discharge(E,X:L1,X:L2) <- discharge(E,L1,L2);
```

In section 3 it will be shown how such definitions contribute to the statement of linguistic principles. We turn now to describe the components of a CLG(2) grammar.

Global type declarations: CLG(2) relies on a strong typing scheme similar to the concept of abstract data type. The following is a detail of the syntactic feature hierarchy used for one type of linguistic sign in one of the grammars implemented in CLG(2):

```
Sign = ( phon, syn, sem,dtrs);
   phon = word list;
   syn = ( local, bind);
      local = ( head, compls, funhead, select, lex );
         head = ( vform, inv, agr, tense);
           vform = {fin, bse, psp, prp, pas, inf, ger };
           inv = { -, + };
           agr = ( num, prs );
             num = { sg, pl };
             prs = { fst, snd, thrd };
           tense = { past, pre};
         compls = Sign list;
         lex = {+, -};
      bind = (slash, subj, wh);
         slash = Sign list;
         subj = Sign list;
         wh = {rel, que};
   dtrs = (head_dtr,compl_dtrs)
         head_dtr = Sign ;
         compl_dtrs = Sign list;
```

Other systems require typing information, including HPSG (Pollard & Sag 1987) and UCG (Moens et al. 1989). Type information is used in CLG(2) both to structure the grammatical information and to achieve a more efficient implementation.

Global constraints: these encode HPSG-type of linguistic principles. A principle is of the form: **partial-object-specification -> constraints.** For instance, HPSG's Head Feature Principle could be expressed as:

```
[head_dtr=[ _ ]]  ->
     syn.local.head =  head_dtr.syn.local.head
```

Partial descriptions of lexical signs. Lexical and phrasal descriptions have both the same format consisting of a pair <DAG,CS> whose first element is a DAG specified by a set of equations and whose second element is a set of complex constraints. Both lexical and phrasal constraints have a number of alternative shorthand formats to suit user requirements.

Partial descriptions of phrasal signs: these are the CLG(2) rules. A number of different equivalent rule formats are supported. For instance:

```
[comp=< >] -> head_dtr,coml_dtrs

M -> H,C+ where M.syn.local.comp=< >
```

are equivalent formulations.

2. Formal Semantics

We define in this section a denotational semantics for CLG(2) grammars in a similar way to what was done for CLG(0) grammars (Damas & Varile, 1989). For reasons of space, we present a slightly simplified version .

Starting from primitive sets Labels and Atoms of attribute names and atomic values we would like to define the domain of objects and the domain of values as follows

```
Objects = Labels --> Vals
Vals = [Atoms + Objects]*
```

Note that to simplify the semantics we are assuming that every label can have as value a list of sub-objects.

Given a set Vars of variable symbols and a set Preds of predicate symbols we define the following syntactic domains:

```
Path        ::= Label | Path.Label

Exp         ::= Var | Atom | Path | Var.Path
                     | Exp+Exp (list concatenation)
                     | Exp:Exp  (list cons )

Constraint  ::= Exp = Exp
                     | p(Exp,...,Exp)
                     | ~Constraint
                     | Constraint & Constraint
                     | Constraint | Constraint

Definition  ::=q(x1,x2,...,xn)  <-> Constraint
```

where we assume that every path which occurs in a definition is associated with a formal argument.

```
Grammar ::= Constraints x Path* x Definition*
```

The Constraints component in a Grammar denotes the conjunction of all principles with the disjunction of the descriptions of all lexical and phrasal signs. The Path* component specifies which paths are involved in the dominance relation for the grammar.

Given an object o and a path p we will extend o to paths by

o(p.l) = o(p)(l)

if o(p) has only one element and that element is not an atom, error otherwise

In what follows we will omit the handling of error values, which should produce error if any partial result leads to error.

To define our semantic functions we still need the following domains:

VEnv = Vars --> Objects*
PEnv = Preds --> U(n) Vals --> {T,F}

Now we define the following semantic functions

V: Exps --> VEnv --> Objects -> Vals
C: Constraint --> PEnv --> VEnv --> Objects
--> {T,F}
D: Definition* --> PEnv
G: Grammar -> Objects --> {T,F}

V, which assigns a value to every expression, is defined by

V[v] r o = r[v]
V[p] r o = o(p)
V[v.p] r o = cardinality(r[v])=1 -> r[v](p)
else bottom
V[e+e'] r o = concatenate(V[e] r o, V[e']r o)
V[e:e'] r o = cons(V[e]r o, V[e']r o)

C, which assigns a truth value to every constraint, is defined by

C[e=e'] d r o = V[e] r o = V[e'] r o
C[p(e1,...,en)] d r o =
d[p] (V[e1] r o , ..., V[en] r o)
C[c & c'] = C[c] & C[c']

D is defined by taking, for each sequence of definitions pi(x1,..,xn)<-> Di, the least fixed point of the function H: PEnv --> PEnv defined by:

H[pi] d (v1, ..., vn) = C[Di] d [vi/xi] o_nil

where o_nil is the empty object.We can now define G as follows:

G[< c, <p1, ...,pk>, Ds >] o = T

iff there is an environment r such that C[c] d r o= T and for every path pi such that o(pi) = <o1, ..., ol>:

G[< c, <p1, ...,pk>, <C1,...,Cn> >] oj = T

for j=1,...,l , where d = D[Ds].

3. Complementizer-Trace Effects in CLG(2)

We will illustrate the expressive power of CLG(2) with an analysis of those phenomena traditionally known as complementizer-trace effects (Perlmutter, 1971; Chomsky & Lasnik, 1977). It is inspired by the HPSG framework (Pollard & Sag, 1987), but it departs from it in some respects.

The most recent account of these phenomena within HPSG is that of Pollard (1985). There, he aims at showing that most of the GPSG insights (Gazdar, Klein, Pullum & Sag, 1985) can be preserved within a framework which does not express subcategorization directly in PS rules, and which does not make use of meta-rules.

In our revision of the analysis we will follow Pollard (1989) in separating subjects selection from complement selection. Our grammar incorporates, however, some radical differences, most of them concerned with the typing of features structures, and the typology of lexical and phrasal categories it induces.

In essence, our approach incorporates a much more articulated theory of minor categories which attributes them a more privileged role than it is generally assumed in PSG frameworks. We assume, then, that minor categories have a certain head-like status and, consequently, selectional properties (Chomsky, 1986; Warner, 1989).

Thus, the top of our hierarchy of signs is as follows:

Sign:	Minor:	Affixes
		Clitics
	Major:	Words
		Phrases

The main difference between major and minor signs is that the latter contain information of type syntactic category and semantics only, while major signs may contain also binding information.

Now consider, the following, schematic lexical entries for the English complementizers **that** and **for**, which are minor signs of type clitic:

that =
[syn.local.select <v[subj< >,compl< >,fin | bse]>]

for = [syn.local.select <v[subj< >,compl< >,inf]>]

Where subj and compl abbreviate subject and complements.

And the schematic entries for the following verbs:

think =
[syn.local.compl <v[compl< >,fin]>,

```
syn.bind.subj <np>]

want1 = [syn.local.compl <v[compl<>,comp.for] |
                    v[sb<np1>,cp<>,inf]>,
          syn.bind.subj <np1>]

want2 = [syn.local.compl<np1,v[sb<np1>,cp<>,inf]>,
               syn.bind.subj<np>]

complain = [syn.local.compl<v[comp.that]>,
              syn.bind.subj<np>]
```

Thus, subject extraction from a clausal complement of think or want is impossible if the complement has a complementizer, because it violates its selectional restrictions.

We predict then that, in English, subject extraction is only possible with bridge verbs (e.g. **think**), and that it is always impossible with non-bridge verbs (e.g. **want1**, **complain**), while complement extraction is always possible (e.g. object extraction in object control verbs like **want2**).

Note that the different syntactic properties of verbal complements (clauses, VPs) seems to have a direct semantic correlation in the property/proposition distinction which has been advocated in some recent analyses of control, e.g. Sag & Pollard (1988).

The CLG(2) grammar which accounts for the above facts contains four rules and four principles. Two rules are the well known Complementation and Topicalization rules of standard HPSG.

The other two are original: one, the Clitic Placement rule, licences those stuctures in which a minor head is attached to a major head; it requires that the selectional restrictions of the minor head be satisfied and marks the mother node with whatever features come from the minor head (e.g., comp=that, when the complementizer is attached to a clause). The other rule is like Topicalization, but for subject binding. As for the principles, we have a Head Feature Principle, a Complementation Principle, a Binding Principle, and a Control Principle.

As an example, we provide the CLG version of the Complementation Principle, which given its formulation has the direct consequence of performing gap introduction when some complement is not found:

```
   Complementation Principle

[head_dtr=[ _ ],compl_dtrs=[ _ ]] ->
                merge(dtrs.head_dtr.syn.loc.compls,
                dtrs.compl_dtrs,
                syn.bind.slash)
```

where merge is a user relation defined as follows:

```
merge(Z,[],Z);
merge(X:L1,X:L2,X.syn.bind.slash+R1)
                              <- merge(L1,L2,R1);
merge(X:L1,Y:L2,Y:R)  <- merge(L1,Y:L2,R);
```

The slash is computed by **merge** by concatenating the slashes of each of the complement daughters with those elements of the **compls** list for which there is no matching daughter.

4. Implementation

The CLG(2) parser has been implemented in Prolog. A CLG(2) grammar is compiled by successively compiling type declarations, partial descriptions of phrasal signs, principles, user defined relations and lexical information.

This implementation, uses a simple bottom-up parser with backtracking and handles constraints using an extension of the ideas described in Damas &.Varile (1989). The parser is implemented as a predicate of the form

```
derive(Tree,[Head|Input],Output) :-
    complete(Head,Input,Output,Tree).

complete(Tree,Input,Input,Tree).

complete(FirstDaughter,Input,Output,Tree) :-
    apply_rules(FirstDaughter,Input,Output1,Tree1),

complete(Tree1,Output1,Output,Tree).
```

where the apply_rules predicate is produced by compiling each grammar rule into a clause for this predicate, which attempts to apply the rule. These clauses also apply all the principles, which are partially evaluated at compile time. This technique usually results in verifying only those principles which are relevant for the particular rule. In the actual implementation the amount of backtracking involved is reduced by introducing other clauses for the complete predicate which handle rules known to have a fixed number of daughters.

Constraints are handled in a way similar to the one described in Damas & Varile (1989) by adding two extra arguments to each of the predicates mentioned above. These arguments contain a list of constraints at clause entry and exit, respectively. From time to time a rewriting process is applied to the list of constraints which may result into failure or new set of simpler constraints. Note that this rewriting process may also cause variable instantiation as a side effect.

Constraints imposed by principles are implemented by a call to a predicate add_constraint which first attempts to decide if the constraint holds or not. If not enough information is available at that time for that purpose the constraint is added to the list of unresolved constraints for latter re-evaluation.

However, the recursively defined constraints (e.g. the user defined relations) have a special treatment. Backtracking is allowed in its application, but some restrictions are imposed, namely they are applied only when sufficiently instantiated to insure that they finitely fail. In particular, for each recursively defined constraint, we must specify which are the minimum conditions of application (for instance which arguments may not be undefined).

Constraints on complex objects require some care on their interpretation and implementation. Consider, for instance, an object description such as

```
[syn.local.subj <NP1>
 syn.local.compls<v[compls< >,comp for] |
                    v[subj<NP1>,compls< >,inf]>]].
```

which is represented internally as a complex term containing only variables plus a constraint on those variables. Note that, if a variable that refers to a atomic value is envolved in a simple equality constraint (or conjunction of) that can be evaluated in compile time.

For the above example we could have (here in the user language, for simplicity) object(Spec,Const), and if in Spec we identify

```
    syn.local.subj = NP1
    syn.local.compls = CP1
    CP1.syn.local.head.maj =CM
    CP1.syn.local.compls= CP2
    CP1.syn.local.subj = NP2
    CP1.syn.local.head.form = F1
    CP1.syn.local.head.comp = CO
```

then

```
Const =( CM = v & CP2 = [] &
        ((NP2 = NP1 & F1 = inf) | CO = for)).
```

Final Remarks

It is clear that the highly structured nature of CLG(2) grammatical descriptions has a number of advantages with respect to more classical approaches, amongst which not least the possibility to express powerful generalization about languages in a highly structured way while maintaining the necessary capability for expressing exceptions.

A drawback of this approach is however that while it is possible to give a clean and simple formal semantics to each individual component, the formalization of the complete grammatical system is certainly more complex than desirable and, as a consequence, the possibility to achieve an efficient implementation is unnecessarily complicated.

We are currently investigating the possibility of making the type theory underlying the Global Declarations a first class citizen, namely being the unifying formal framework for all the grammar components (at least for all non lexical information).

By this we mean that a type declaration system in the form of an algebra of sorts can cover essentially the expressive requirements of our current formalism while providing a simple and uniform formal framework for the whole.

Acknowledgement

This work has been carried out within the framework of the Eurotra R&D programme for machine translation financed by the European Communities. We are especially grateful to a number of colleagues for their useful comments on earlier versions of CLG(n).

Bibliographical References

Chomsky, N. (1986). Barriers, MIT Press, Cambridge.

Chomsky, N. & H. Lasnik (1977). "Filters and control", Linguistic Inquiry, 8, 425-504.

Damas, L. & G.B. Varile (1989). CLG: A grammar formalism based on constraint resolution, in Proceedings of EPIA 1989, E.M. Morgado & J.P. Martins (eds.), Lecture Notes in Artificial Intelligence 390, Springer Verlag.

Gazdar, G., E. Klein, G.K. Pullum & I.A. Sag (1985). Generalized Phrase Structure Grammar, Basil Blackwell, Oxford.

Jaffar, J., J-L. Lassez (1988). From unification to constraints, in Logic Programming 1987, G. Goos & J. Hartmanis (eds.) Lecture Notes in Computer Science 315, Springer Verlag, 1-18.

Johnson, R. & M. Rosner (1989). A rich environment for experimentation with unification grammars, in Proceedings of the fourth conference of the European Chapter of the ACL, ACL, 182-189.

Karttunen L. (1984). Features and values in Proceeding of COLING-84, 28-33.

Kasper, R. (1987). A unification method for disjunctive feature description, in ACL Proceedings, 24th annual meeting, ACL, 235-242.

Moens, M. J. Calder, E. Klein, M. Reape, H. Zeevat (1989). Expressing generalizations in unification-based formalisms, in Proceedings of the fourth conference of the European Chapter of the ACL, ACL, 174-181.

Moshier M.D. & W.C. Rounds (1987). A logic for partially specified data structures in ACM symposium on the principles of programming languages, Association for Computing Machinery.

Pereira, F.C.N. (1987). Grammars and logics of partial information, Technical Note 420, SRI International, Menlo Park.

Perlmutter, D. (1971). Deep and Surface Constraints, in Syntax, Holt, Rinehart & Winston, New York.

Pollard, C.J. (1985). "Phrase structure grammar without metarules", in J. Goldberg, S. MacKaye & M.T. Wescoat, eds., Proceedings of the West Coast Conference on Formal Linguistics, 4, Stanford Linguistics Association, Stanford, 246-261.

Pollard, C.J. (1989). "The syntax-semantics interface in a unification-based phrase structure grammar", in S. Busemann, Ch. Hauenschild & C.Umbach, eds., Views of the Syntax/Semantics Interface, KIT Report 74, Technische Universitaet Berlin, Berlin.

Pollard, C.J. & I.A. Sag (1987). Information-Based Syntax and Semantics, CSLI Lecture Notes Series, CSLI, Stanford.

Rounds, W.C. & R. Kasper (1986). A complete logical calculus for record structures representing linguistic information, in Symposium on logic in computer science, IEEE Computer Society.

Sag, I. & C. Pollard (1988). A semantic theory of obligatory control. MS.

Warner, A.R. (1989). "Multiple heads and minor categories in generalized phrase structure grammar", Linguistics, 27, 179-205.