The Formal and Processing Models of CLG

Luis DAMAS
Nelma MOREIRA
University of Porto, Campo Alegre 823
P-4000 Porto
luis@nccup.ctt.pt

Giovanni B. VARILE
CEC
Jean Monnet Bldg. B4/001
L-2920 Luxembourg
nino@eurokom.ic

Abstract: We present the formal processing model of CLG, a logic grammar formalism based on complex constraint resolution. In particular, we show how to monotonically extend terms and their unification to constrained terms and their resolution. The simple CLG constraint rewrite scheme is presented and its consequence for CLG's multiple delay model explained.

Keywords: Grammatical formalisms, Complex constraint resolution.

Introduction

CLG is a family of grammar formalisms based on complex constraint resolution designed, implemented and tested over the last three years. CLG grammars consist of the description of global and local constraints of linguistic objects as described in [1] and [2].

For the more recent members of the CLG family, global constraints consist of sort declarations and the definition of relation between sorts, while local constraints consist of partial lexical and phrasal descriptions. The sorts definable in CLG are closed, in a way akin to the ones used by UCG [3]. Relations over sorts represent the statement of linguistic principles in the spirit of HPSG [4].

The constraint language is a classical first order language with the usual unary and binary logical connectives, i.e. negation (~), conjunction (&), disjunction (|), material implication ($\rightarrow$), equivalence ($\leftrightarrow$) and a restricted form of quantification ($\forall$ and $\exists$) over finitely instantiatable domains. The interpretation of these connectives in CLG is strictly classical as in Smolka's FL [6] and Johnson's AVL [5], unlike the intuitionistic interpretation of negation of Moshier and Rounds [7]. A more detailed description of CLG including its denotational semantics can be found in [2].

In this paper we present the formal processing model of CLG, which has been influenced by the Constraint Logic Programming paradigm [8] [9]. We show in what way it extends pure unification based formalisms and how it achieves a sound implementation of classically interpreted first order logic while maintaining practical computational behaviour by resorting to a simple set of constraint rewrite rules and a lazy evaluation model for constraints satisfaction thus avoiding the problem mentioned in [10] concerning the non-monotonic properties of negation and implication interpreted in the Herbrand universe.

The paper is organized as follows: in the first part we show how we extend term unification to accommodate complex constraint resolution. We then explain what rewrites are involved in CLG constraint resolution, proceeding to show what the benefits of the delayed evaluation model of CLG are. We conclude by discussing some of the issues involved in our approach and compare it to other approaches based on standard first order logics.

From Unification to Constraint Solving

We will first show how to extend a unification based parsing algorithm for a grammar formalism based on an equational theory, to an algorithm for a formalism with complex constraints attached to rules.

Assume a countable set V of variables x, y, z, ... and a countable set F of function symbols f, g, h, ... each one equipped with an arity expressed as f^n. Let T be the term algebra over F and V, and T_0 be the corresponding set of ground terms.

Assume furthermore that rules are of the form:

$$t \rightarrow t_1 \ldots t_n$$

$$\text{for } t, t_1, \ldots, t_n \text{ are in } T$$

and that the parsing algorithm relies solely on the unification algorithm for its operation, applying it to terms and either computing a unifier of those terms or failing.

Associating with a term t its usual denotation

$$[\![t]\!] = \{ St \in T_0 \}$$

(where S denotes a substitution of terms for variables) the unifier t of two terms t' and t" has the following important property

$$[\![t]\!] = [\![t']\!] \cap [\![t'']\!]$$

Next we introduce constraints over terms in T. For the moment we will assume that constraints c include at least atomic equality constraints between terms and formulas built from the atomic constraints using the standard logic operators, namely disjunction, conjunction and negation, and that a notion of validity can be defined for closed formulas (see however [2] for an extended constraint language).

We will extend terms to constrained terms t:c, where c is a constraint involving only variables occurring in t, and take

$$[\![t:c]\!] = \{ St \in T_0 \mid \vdash Sc \}$$

as its denotation.

Now, given constrained terms t:c, t':c' and t":c" we say that t:c is a unifier of t':c' and t":c" iff

$$[\![t:c]\!] = [\![t':c']\!] \cap [\![t'':c'']\!].$$

It is easy to see that there is at least one algorithm which given two constrained terms either fails, if they do not admit a unifier, or else returns one unifier of the given terms. As a matter of fact it is enough to apply the unification algorithm to t' and t" to obtain an unifying substitution S and to return S(t':c'&c").

We can then annotate the rules of our formalism with constraints and use any algorithm for computing the unifier of the constrained terms to obtain a new parsing algorithm for the extended formalism. It is interesting to note that, if we used the trivial algorithm described above for computing the unifier of constrained terms, we would obtain exactly the same terms as in the equational case but annotated with the conjunction of all the constraints attached to the instances of the rules involved in the derivation.

One of the obvious drawbacks of using such a strategy for computing unifiers is that there is no guarantee that the denotation of S(t':c'&c") is not empty since S(c'&c") may be unsatisfiable. We will now give two properties of unifiers which can be used to derive more interesting algorithms.

Assume t:c is an unifier of t':c' and t":c" and c is logically equivalent to d, then t:d is also a unifier. Similarly if, for some variable x and term r, we can derive x=r from c, then [r/x](t:c) is also a unifier for t':c' and t":c", where [r/x] denotes substitution of r for x.

It is obvious that by using an algorithm similar to the one used by Jonhson [5] for reducing the constraint c to normal form, it is possible to find all the equalities of the form x=r which can be derived from c, and also decide if c is satisfiable. This strategy, however, suffers from the inherent NP hardness, and, for practical implementations we prefer to use, at most unification steps, an incomplete algorithm reserving the complete algorithm for special points in the computation process which include necessarily the final step.

Rewriting and Delaying Constraints

In this section we present a slightly simplified version of the constraint rewriting system which is at the core of the CLG model. As will be apparent from these rules they attempt a partial rewrite to conjunctive rather than to the more common disjunctive normal form. Some of the reasons for this choice will be explained below.

Another point worthwhile mentioning here is that linguistic descriptions and linguistic representations are pairs consisting of a partial equational description of an object and constraints (cf. [2]) in contrast to [12,14] where constraints are kept within linguistic objects.

The CLG constraint language includes expressions involving paths which allow reference to a specific argument of a complex term in order to avoid the need for introducing existential quantifiers and extraneous variables when specifying constraints on arguments of terms.

We define paths p, values v and constraints c as follows (quantification is omitted for reasons of simplicity):

$p ::=$ <empty>
p. $f^n.\pi_i$

$v ::=$ t
t.p
$\perp$

$c ::=$ $t.p.f^n$
$v = v$
$\sim c$
$c \ \& \ c$
$c \mid c$

In the above definitions π_i denotes the i -th projection while the superscript in f^n indicates the arity of f as before. As an example, if t denotes

$$f(a,g(c,d))$$

the following constraints are satisfied:

$t.f^2$ $\quad$ $t.f^2.\pi_2.g^2$

$t.f^2.\pi_1 = a$ $\quad$ $t.f^2.\pi_2.g^2.\pi_2 = d$

We can now state the CLG rewriting rules for values:

Rewriting Values

$f(t_1,...,t_n).f^n.\pi_i.p \rightarrow t_i.p$

$f(t_1,...,t_n).g^k.\pi_i \rightarrow \perp$ $\quad$ if $f^n \neq g^k$

and for constraints (keeping in mind that implication and equivalence are just shorthands):

Rewriting Constraints

true & c $\rightarrow$ c

false | c $\rightarrow$ c

$\sim\sim c \rightarrow c$

$\sim$false $\rightarrow$ true

$\sim$true $\rightarrow$ false

true | c $\rightarrow$ true

false & c $\rightarrow$ false

$\sim(c \mid c') \rightarrow \sim c \ \& \sim c'$

$\perp.f^k \rightarrow$ false

$f(t_1,...,t_n).f^n \rightarrow$ true

$g(t_1,...,t_n).f^k \rightarrow$ false $\quad$ if $f^k \neq g^n$

$v = v' \rightarrow$ false $\quad$ if either v or v' is $\perp$

$v = v' \rightarrow$ true $\quad$ if v and v' are the same value

$v = v' \rightarrow$ false if v and v' are atomic and $v \neq v'$

$f(t_1,...,t_n) = f(u_1,...,u_n) \rightarrow$
$t_1=u_1 \ \& \ ... \ \& \ t_n=u_n$

$f(t_1,...,t_n) = g(u_1,...,u_n) \rightarrow$ false

We will use set notation to denote a conjunction of the constraints in the set. Using this notation we can state the following rules for rewriting constrained terms:

Rewriting Constrained Terms

$t:\{..., false,...\} \rightarrow FAIL$

$t:\{..., true,...\} \rightarrow t:\{...,...\}$

$t:\{..., c_1 \& c_2,...\} \rightarrow t:\{...,c_1,c_2...\}$

$t:\{...,x.p = t',...\} \rightarrow [p(t') / x \mid t:\{...,...\}$

$t:\{...,x.p = y.q,...\} \rightarrow$
$[p(z) / x, q(z) / y] t:\{...,...\}$

$t:\{...,x.p.f^k ,...\} \rightarrow$
$[p(f(z_1,...,z_k)) / x \mid t:\{...,...\}$

where $z, z_1,...,z_n$ are new variables and p(...) which can be defined is by:

<empty> (x) = x

$f^n.\pi_i.p(x) = f^n(z_1,...,z_{i-1}, p(x),...,z_n)$

returns a new generic term t such that the constraint t.p = x is satisfied.

The above is a slight simplification: constraints associated with terms come in fact in pairs, the second element of which is omitted here for the sake of simplicity and contains essentially negated literals and inequations. The reason for this is that we want to give the system a certain inferencing capability without having to resort to expensive exhaustive pairwise search through the constraint set.

It should also be mentioned that after one constraint in a set is rewritten it will only be rewritten again if some variable occurring in it is instantiated.

Completing Rewrites

As already mentioned the set of rewrite rules given above is not complete in the sense that it is not sufficient to reduce all constraints to conjunctive normal form, although CLG has a complete set of rewrite rules available to be used whenever needed. At least at the end of processing, representations are reduced to conjunctive form.

Sets of rules for rewriting first order logic formulae to conjunctive normal form can be found in the literature [11]. The specific set of complete rewrites currently used in CLG includes e.g.:

(1) $c \mid (c' \,\&\, c'') \rightarrow (c \mid c') \,\&\, (c \mid c'')$

(2) $\sim (c \,\&\, c') \rightarrow \sim c \mid \sim c'$

(3) $(c \mid c') \,\&\, (\sim c \mid c'') \rightarrow c' \mid c''$

There are various reasons for not using them at every unification step. The application of the distributive law (1) is avoided since it contributes to the P-Space completeness of the reduction to normal form: in general we avoid using rules which are input length increasing.

As for the de Morgan law (2), we do not use it because by itself it does neither help to detect failure nor does it contribute to add positive equational information.

Lastly, the cut rule (3) is just too expensive to be used in a systematic way.

Our current experience shows that the number of constraints which need the complete set of rewrite rules to be solved is usually nil or extremely small even for non-trivial grammars [1].

Discussion

The three main characteristics of the CLG processing model are the use of constrained terms to represent partial descriptions, the lack of systematic rewriting of constraints to normal form and the lazy evaluation of complex constraints.

The choice of constrained terms instead of the more common sets of constraints is motivated by methodological rather than theoretical reasons. The two representations are logically equivalent but CLG's commitment to naturally extend unification to constraint resolution makes the latter better suited if, as in the present case, we want to use existing algorithms where they have shown successful.

The alternative, to develop new algorithms and data structures for complex constraint resolution (including equation solving) [12,13,14] is less attractive. It is preferable to split the problem into its well understood equational subpart and the more speculative complex constraint resolution.

It is also worthwhile noting that terms constitute a very compact representation for sets of equations and naturally suggest the use of conjunctive forms, another distinguishing characteristics of CLG. Furthermore, conjunctive forms constitute a compact way of representing partial objects in that they localise ambiguity.

We already have discussed the reasons for avoiding systematic rewrites of constraints to normal form. This in no way affects the soundness of the system although it may prevent early failure. Even so it is computationally more effective than resorting to normal form reduction

Note that CLG is not a priori committed to check whether newly added constraints will lead to inconsistency. However it is often possible to check such inconsistencies at little cost without full reduction to normal form. A solvability check is only performed for a limited number of easily testable situations, mainly for the case of negated literals, of which a separate list is kept as mentioned above.

It has to be pointed out though, that in order to guarantee the global completeness of the rewrites, as opposed to potential local incompleteness, CLG completes the rewrite to normalized form at the latest at the very end of processing. Nevertheless this decision is not a commitment. Rather, a rewrite to normal form could be carried out with the frequency deemed necessary. Our present experience however shows that a full rewrite at the end is sufficient.

Finally, the way constraint resolution is delayed is a direct consequence of the rewrites available at run-time. Every constraint which cannot at a given point in time be reduced with one of the above rules is just left untouched in that cycle of constraint evaluation, awaiting for further instantiations to make it a candidate for reduction.

A last note on some consequences these properties have for the user: as with other complex constraint based systems, in CLG there is no guarantee that all constraints will always be solved, not even after the last rewrite to normal form. As a result (a) the system does not fail because all constraints have not been resolved and (b) the intermediate and final data structure are also partial descriptions, being potentially annotated with unresolved constraints, and denote not a single, but a class of representations.

The first consequence is clearly a desirable property, for it is unreasonable to think that grammatical descriptions will ever be complete to the point where all and only the constraints which are needed will be expressed in a grammar and all and only the information which is needed to satisfy these constraints will be available at the appropriate moment.

As for the second consequence, we have found unresolved constraints to be the best possible source of information about the state of the computation and the incompleteness of grammatical description.

Relation to Other Work

Although in this paper we have presented a specific (subset of) constraint language and a specific incomplete set of rewrite rules, neither is integral part of CLG's theoretical framework.

In fact the basic ideas behind the CLG processing model can be carried over to other frameworks, such as the feature logic of Smolka [6,15], by replacing the unification of terms with the unification of the set of equational constraints and by either redefining the constraint language in a suitable way (e.g. redefining the notion of path) or else by translating the non-atomic formulae of the feature logic.

Finally, note that the processing model described in this paper can, and eventually should, be complemented with techniques from constraint logic programming [16] to handle cases such as constraints on finite domain variables where the completeness of the constraint handling is computationally tractable.

Conclusions

We have shown how, starting from a purely unification based framework, it is possible to extend its expressive power by introducing a constraint language for restricting the ways in which partial objects can be instantiated, and have provided a general strategy for processing in the extended framework.

We have also presented and justified the use of partial rewrite rules which, while maintaining the essential formal properties, are computationally effective with available technologies.

We justified the use of conjunctive forms as a better option than their disjunctive counterparts as a means for providing amongst other things a compact representation of partial objects.

Finally we have emphasized the importance of lazy evaluation of complex constraints in order to ensure computational tractability.

Acknowledgement

The work reported herein has been carried out within the framework of the Eurotra R&D programme financed by the European Communities. The opinions exposed are the sole responsibility of the authors.

References

[1] Damas, Luis and Giovanni B. Varile, 1989. "CLG: A grammar formalism based on constraint resolution", in EPIA '89, E.M. Morgado and J.P. Martins (eds.), Lecture

Notes in Artificial Intelligence 390, Springer, Berlin.

[2] Balari, Sergio, Luis Damas, Nelma Moreira and Giovanni B. Varile, 1990. "CLG: Constraint Logic Grammars", Proceedings of the 13th International Conference on Computational Linguistics, H. Karlgren (ed.), Helsinki.

[3] Moens, M., J. Calder, E. Klein, M. Reape and H. Zeevat, 1989. "Expressing generalizations in unification-based formalisms", in Proceedings of the fourth conference of the European Chapter of the ACL, ACL.

[4] Pollard, Carl J. and Ivan A. Sag, 1987. "Information-Based Syntax and Semantics 1: Fundamentals", Center for the Study of Language and Information, Stanford, CA.

[5] Johnson, Mark, 1988. "Attribute-Value Logic and the Theory of Grammar", Center for the Study of Language and Information, Stanford, CA.

[6] Smolka, G. 1989. "Feature Constraint Logics for Unification Grammars", LILOG Report 93, IWBS, IBM Deutschland.

[7] Moshier, M. Drew and William C. Rounds, 1986. "A logic for partially specified data structures", manuscript, Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI.

[8] 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, Berlin.

[9] Cohen, Jacques, 1990. "Constraint Logic Programming Languages", in CACM, July 1990,volume 33, No. 7.

[10] Doerre, Jochen, Andreas Eisele, 1990. "Feature Logic with Disjunctive Unification", Proceedings of the 13th International Conference on Computational Linguistics, H. Karlgren (ed.), Helsinki.

[11] Hilbert, D., P. Bernays, 1934 & 1968. "Grundlagen der Mathematik I & II", Springer, Berlin.

[12] Carpenter, B., C. Pollard, A. Franz (to appear). "The Specification and Implementation of Constraint-Based Unfication Grammars".

[13] Kasper, Robert, 1987. "A Unification Method for Disjunctive Feature Description", Proceedings of the 25th Annual Meeting of the ACL, ACL.

[14] Carpenter, Bob, 1990. "The Logic of Typed Feature Structures: Inheritance, (In)equations and Extensionality", unpublished Ms.

[15] Smolka, Gert, 1988. "A Feature Logic with Subsorts", LILOG Report 33, IWBS, IBM Deutschland.

[16] Van Hentenryck, P., M. Dincbas, 1986. "Domains in Logic Programming", Proceedings of the AAAI, Philadelphia, PA.