The Formal and Computational Theory of Complex Constraint Solution

Luis Damas
Nelma Moreira
University of Porto

Giovanni B. Varile
Commission of the European Communities

The framework presented in this chapter assumes that any grammar is a particular first order theory with equality. We will turn our attention to those first order theories admitting complete models and extend them to recursively defined relations.

Logic programming is an ideal paradigm for such a framework in that it supports a direct mapping between grammars, seen as first order logic theories, and the first order theory of their implementation and at the same time provide a formally sound and efficient computational scheme.

It is also particularly well suited to simultaneously satisfy the main requirements put on *computational* grammar formalisms, namely expressivity, formal soundness and computational tractability.

As we will see, the framework presented in this chapter is not only compatible with the unification grammar tradition but it also constitutes a simple framework for extending the notion of unification to complex constraint resolution. At the same time a high degree of declarativeness is achieved by avoiding any reference to an *operation* like *unification.*

The motivations for the choice of the particular family of first order theories will become apparent in the rest of this chapter.

The main reasons derives from the fact that restricting the formal analysis to the *static* properties of formalisms does not do justice to the *computational* complexity of modern linguistic frameworks. A finer grained analysis of the formal *and computational* properties of formalisms than decidability, formal complexity and model theoretic properties, sheds a different light on the problem motivating choices which would otherwise appear to be arbitrary.

It is not sufficient to design formalisms with a *simple* and sound denotational semantics and appropriate formal complexity characteristics. Rather, it is necessary to provide sound and adequate formal processing schemes for such formalisms, lacking which the main challenges facing modern grammatical formalism design are not addressed.

Taking this point further, we claim that the theory of a grammar formalism and its formal processing model constitute a homogeneous and integrated whole and that the practice of relegating processing issues to low level implementation decisions had, and has, nefast consequences, not least preventing the right questions to be addressed.

The deductive process by which a fact is proven or an object computed must be the subject of theoretical inquiry just as, and together with, the fact or object and their descriptions.

The analogy with logic programming is paradigmatic: defining the syntax and (static) semantics of a logic programming scheme is an essential first step. But it also constitutes a relatively trivial task compared with definition of a formal processing scheme with the necessary computational characteristics.

Our goal is to show how and under what circumstances we can ensure a simple and natural relation between grammars with complex constraint expression seen as first order theories and the logic programming paradigm, in particular a version of Constraint Logic Programming over the domain of rational trees.

This chapter is organized as follows: in Section 1 we will present the family of first order theories upon which *C*onstraint *L*ogic *G*rammars, or CLGs, are based, namely those having the complete algebra $\mathcal{RT}$ of rational trees as their canonical model.

After having justified this choice, we will elaborate on the basic characteristics of these first order theories, extend them with recursive relations and relate them to $CLP(\mathcal{RT})$[1].

In Section 2 we will show that these theories relate in a natural way to feature logics and in Section 3 how to extend term unification to constraint resolution in a simple way.

In Section 4 we will present CLGs complex constraint solver and its complete constraint rewriting system.

[1]*C*onstraint *L*ogic *P*rogramming over $\mathcal{RT}$ [JL88].

The description of the formal processing model of CLGs will be completed in Section 5 where we define the CLG model seen as an instantiation of the constraint logic programming scheme for the symbolic domain of rational trees.

1 The First Order Theory of CLG

CLG grammars are first order theories, i.e. object of first order logic consisting of alphabets of logical and non logical symbols including the equality, the usual logical axioms and inference rules, and a number of non-logical axioms [2].

The role of the non logical axioms is twofold: for one, a number of these axioms will restrict the possible models for our theories in a convenient way as explained below.

The second role of these axioms is to represent grammar rules, lexical objects and linguistic principles. In other words they characterize the interpretations which are well formed with respect to a given grammar.

All the theories considered here will be extensions of theories having as canonical models the complete algebra $\mathcal{RT}$of rational trees. The extension are needed to account for recursive definitions within the theory, as explained below. No generality is lost by making this choice as we will see in Section 2.

We will first proceed to define the denoting objects of a CLG theory, i.e. the terms. We will then characterize the constraint language and impose restrictions on the possible interpretations of the theory. Lastly we will extend the first order theory obtained in this way in order to account for recursive relations needed to express grammar rules and principles.

1.1 Terms

Given countable sets F and V of function symbols and variables respectively, we define the terms T of our theory in the usual way as being the least set such that:

1. elements of V are in T
2. for f^n in F and $t_1, \ldots, t_n$ in T, $f^n t_1, \ldots, t_n$ is in T

Function symbols come equipped with their arity n written as f^n.

[2]See [Sho67] for an introduction to first order theories.

We will see below how we will restrict the interpretation of the elements of T to decidable algebrae.

1.2 Constraint Language

We define the constraint language $L_{\mathcal{C}}$ over T as being the usual first order language with the equality as the only predicate symbol and the logical connectives:

$$\wedge, \vee, \neg, \rightarrow, \exists, \forall$$

The constraints $\mathcal{C}$ are the well formed formulae of this language. The atomic constraints are expressions of the form $t_1 = t_2$ for terms t_1 and t_2, *true* and *false*. The set $\mathcal{C}$[3] of constraints is defined as follows:

1. atomic constraints are constraints in $\mathcal{C}$
2. for constraints c_1, c_2 in $\mathcal{C}$, $\neg c_1$, $c_1 \wedge c_2$, $c_1 \vee c_2$, $c_1 \rightarrow c_2$, $\exists x c_1$ and $\forall x c_1$ are also in $\mathcal{C}$

In Section 4.1 we will slightly extend the notion of constraints.

1.3 Constrained Terms

Constrained terms take a special role in *CLG* as will become clear in the reminder of this chapter. They consist of a term t and a constraint constraint c of $\mathcal{C}$ constraining the values which the variables in t can take. Constrained terms will be written as $t : c$.

1.4 Admissible Models of *CLG*

We will restrict the class of admissible models of the first order theory with equality of *CLG* to the complete algebra of rational trees $\mathcal{RT}$ as axiomatized by Maher [Mah88a, Mah88b]. As a consequence all the theories considered here will be extensions of theories having as canonical model $\mathcal{RT}$.

The first order theory of $\mathcal{RT}$ contains no predicate symbols other than equality and is obtained by introducing as non-logical axioms the axiom schemata required to make the algebra of rational trees the canonical model for the theory [Mah88a]:

[3]Note that for the case of simple equational theories like the one underlying pure PATR-II, the constraint language is a special case of $L_{\mathcal{C}}$ with $\wedge$ as the only logical connective.

1. for every term t,

$$t = t$$

2. for all terms t,s,

$$t = s \to s = t$$

3. for all terms t, s, u,

$$t = s \wedge s = u \to t = u$$

4. for every n, f^n,

$$f(x_1, ..., x_n) = f(y_1, ..., y_n) \leftrightarrow x_1 = y_1, ..., x_n = y_n$$

5. for every $f \not\equiv g$,

$$f(x_1, ..., x_n) \neq g(y_1, ..., y_n)$$

6. for every rational solved form $\bar{x} = t(\bar{x}, \bar{y})$[4],

$$\forall \bar{y}\ \exists! \bar{x}.\ \bar{x} = t(\bar{x}, \bar{y})$$

where a set of equations is in rational solved form [Mah88a] if it is of the form:

$$x_1 = t_1(\bar{x}, \bar{y}), ..., x_n = t_n(\bar{x}, \bar{y})$$

[4] $\bar{x}$, resp. $\bar{y}$ stand for sets of variables $x_1, ..., x_n$ resp. $y_1, ..., y_m$.

with disjoint set of variables $\bar{x}$, $\bar{y}$ and such that the set of equations contains no circular subset [Mah88a], i.e. a subset of the form:

$$x_1 = x_2, ..., x_{n-1} = x_n, x_n = x_1$$

There are several motivations for the choice of this model:

1. In absence of predicate symbols other than equality, $\mathcal{RT}$ is decidable [Mah88a].

2. The domain of rational trees $\mathcal{RT}$ has been used as a model in many computation domains and a wealth of formal and computational results are available on this domain.

3. Under certain assumptions, there is a relationship between Feature Algebra [Smo88] and the domain $\mathcal{RT}$ of rational trees, as we will see in Section 2

1.5 The First Order Theory of CLG and $CLP(\mathcal{RT})$

Although $\mathcal{RT}$ is adequate as a language to describe classes of objects, grammar rules and principles, except for the simplest cases, require some form of recursive definition.

One way to account for this would be to regard grammar rules as inference rules. This has the disadvantage that any general statement about the logical theory would be difficult to make.

The other possibility is to introduce *n-ary predicate symbols*[5] to express classes of objects in the grammar and grammar rules as axioms defining these predicates.

We will restrict the grammar axioms to take the form of constrained clauses:

$$A_1 \wedge ... \wedge A_n \wedge C \rightarrow A$$

where A and the A_i denote atomic formulae of the form $p(t_1, ..., t_k)$ with p a predicate symbol of arity k and $t_1, ..., t_k$ terms, and C denotes a constraint, i.e. a logic formula involving only the equality predicate.

Any logic theory of the type just defined has a clear and well understood semantics. It can be seen as a constraint logic programming program in $CLP(\mathcal{RT})$ [JL87]. It can also be seen as a generalization of Prolog II [Col82].

[5]For instance by using the Höhfeld-Smolka construction [HS88].

One benefit of this choice is that we can benefit from the extensive theoretical work and implementation techniques which were developed for Constraint Logic Programming.

In Section 5 we will expand on this issue by giving the specific formal processing model of CLG seen as a constraint logic programming language over $\mathcal{RT}$.

Note that with the extension just performed, our theory CLG is only semi-decidable. Further constraints could be imposed on the form of the defining axioms for the predicates which would ensure decidability. We refrain from doing so, leaving to the user the responsibility to ensure that grammars, as opposed to their supporting formal theory, are decidable.

2 Term Logics and Feature Logics

In recent years it has become common practice to reason about the properties of formalisms for linguistic description within the framework of formal logic. A number of first order theories have been proposed to serve the purpose, ranging from classical first order predicate calculus to numerous variants of non-classical logics [RK86, KR86] elsewhere dubbed *designer logics* [Joh90].

These latter approaches have been criticized for cutting the field off the wealth of results in classical logic without real benefits since it has never been shown that any of the non-classical devices proposed was indeed necessary [Joh90].

The same applies when it comes to consider formal processing models: the many results obtained in the logic programming and constraint logic programming fields bear no *direct* relation to non-classical logical frameworks, other than providing high-level implementation tools.

In our opinion there is a lot to be gained, from the theoretical and the formal processing point of view, in keeping as close a relation as possible to these fields.

It is not our intention to further feed this discussion, but since we are restricting our attention to classical first order logics with the algebra of rational trees as canonical model as explained in Section 1.4, it will be useful to make explicit the relation between the logic of our choice and feature logics, at least in their classical versions.

2.1 The Relation Between the Algebra of Rational Trees $\mathcal{RT}$ and the Feature Algebra $\mathcal{F}$

The purpose of this section is to establish a relation between the logic adopted for the *CLG* representation theory, namely the algebra of $\mathcal{RT}$ rational trees as axiomatized by Maher [Mah88a, Mah88b], and the axiomatization of Feature Logics of Smolka [Smo88].

As already mentioned, a number of non-classical logics have been proposed in order to formalize feature based grammar formalisms over the past few years.

Smolka has given an axiomatization of the models of feature logics in first order predicate logic [Smo89], and he proposes to model features by representing them as binary predicates $p(x, y)$ satisfying the following axiom schemata:

$$\begin{array}{ll} \vdash \neg a = b & \text{for all distinct atoms } a \text{ and } b \\ \vdash \forall y. \neg p(a, y) & \text{for all atom } a \\ \vdash \forall x \forall y \forall z. p(x, y) \wedge p(x, z) \rightarrow y = z & \end{array}$$

The first of the axioms above expresses that distinct atoms denote different entities, the second that atoms do not have any features and the third the functional character of features.

In what follows we will assume that every feature logic contains an infinite number of atoms. If Π is a set of two-place predicate symbols, we will use $F(\Pi)$ to denote the first order equality theory over Π consisting of the standard axioms for equality (axioms 1-3 of Section 1.4) and the above axioms.

Although these axioms captures most of the essential properties of features, they are too weak in the sense that they do not guarantee the existence of non-trivial objects. As a matter of fact they admits models in which predicates are always false thus making the logic trivial in some sense.

More recently Smolka has proposed another axiom which asserts the existence of solutions of a set of recursive feature equations in rational solved form. He has also shown that, in the case of an infinite number of features, the axiomatization is complete and admits a canonical model a domain similar to rational trees.

Let's assume a countable infinite number of atoms.

We start by recalling that the complete axiomatization of Feature Logic [Smo88] assumes an infinite number of features. This assumption is essential for the completeness of Smolka's axiomatization since if there was a finite number of features $f_1, f_2, ..., f_n$ then a formula like:

$$xf_1z_1 \wedge yf_1z_1 \wedge ... \wedge xf_nz_n \wedge yf_nz_n \rightarrow x = y$$

would be valid in some models and false in others.

The formula above is asserting the fact that if two objects have exactly the same features then they are the same object. It can be generalized, in the case of feature logic with a finite number of features, to a formula stating that if two objects are undefined for the same set of features and take identical values for the remaining features, then the objects are equal.

If we take such a formula as axiom it is tantamount to accepting the "completeness" of feature descriptions, i.e. that objects are completely defined by their features.

When we have a finite number of features, the axiomatization obtained by adding such an axiom to the axioms for the infinite case, is complete and admits as canonical model the above-mentioned algebra of rational trees over a finite number of features.

Our first result is that a formula A is valid in the infinite model $\mathcal{F}_\infty$ iff it is valid in a model $\mathcal{F}$ for a finite number of features. The details of the proof are given in Section 2.2 below.

One advantage of this equivalence is that we can work, in any practical situation, with a finite number of features. It is interesting to note that we were able to reduce proofs in a logic without "completeness" of feature descriptions to proofs in a logic were that "completeness" holds.

This is not surprising since we can introduce an extra feature f_{n+1} describing the extra information necessary to make the object unique. This has been realized in practice in term-based implementations of unification formalisms.

The second main result concerns the inter-operability of feature structures and feature terms. It is in fact formally irrelevant whether we use the former or the latter since there exists an effective translation between feature structures and terms as is shown in Section 2.3.

2.2 Finite and Infinite Models

In the following we show that a formula A is valid in the infinite model $\mathcal{F}_\infty$ iff it is valid in a model $\mathcal{F}$ for a finite number of features.

More precisely assume A involves only the features $f_1, f_2, ..., f_n$, then A is valid in Smolka's model iff it is valid in the model for the features $f_1, f_2, ..., f_n, f_{n+1}$.

The proof of this result relies on the existence of injective mappings θ_0 and θ_1, from $\mathcal{F}$ to $\mathcal{F}_\infty$ and vice-versa, such that for rational trees t and t' and for $i = 1, ..., n$, if $t f_i t'$ then $\theta_k(t) f_i \theta_k(t')$. For θ_0 we can take the inclusion map from $\mathcal{F}$ to $\mathcal{F}_\infty$.

θ_1 can be constructed by encoding all the extra features (i.e. f_{n+1}, f_{n+2}, ...) under f_{n+1}. To do this we first select a distinct atom $a_1, a_2, ...$ for each one of $f_{n+1}, f_{n+2}, \ldots$.

Now we note that each rational tree can be uniquely represented by a feature structure of the form

$$\begin{bmatrix} f_1 : s_1 \\ \vdots \\ f_n : s_n \\ f_{n+1} : s_{n+1} \\ f_{n+2} : s_{n+2} \\ \vdots \end{bmatrix}$$

the s_i denote either feature structures or indices (i.e. variables).

To such a structure we associate the following feature structure, involving only the features $f_1, f_2, ..., f_n, f_{n+1}$

$$\begin{bmatrix} f_1 : \tilde{s}_1 \\ \vdots \\ f_n : \tilde{s}_n \\ f_{n+1} : \begin{bmatrix} f_1 : a_1 \\ f_2 : \tilde{s}_{n+1} \\ f_{n+1} : \begin{bmatrix} f_1 : a_2 \\ f_2 : \tilde{s}_{n+2} \\ f_{n+1} : \begin{bmatrix} \vdots \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

where $\tilde{s}$ denotes the transformed version of s. For atoms and indices this is the identity transformation.

Now a simple proof by induction in A, using θ_0 and θ_1 to map between valuations in the two models, shows that A is valid in $\mathcal{F}$ iff it is valid in $\mathcal{F}_\infty$.

Finally, since the models are canonical models, a formula is a theorem in one logic iff its a theorem in the other logic.

2.3 Feature Terms and Feature Structures

In this section we will show that using terms, or more precisely rational trees, is justified in formal grounds by defining an explicit translation between feature structures and terms.

Assume we have a finite number of features $f_1, f_2, ..., f_n$ with the complete axiomatization above. We want to show this logic is equivalent to the Maher's complete axiomatization of rational trees.

To do this we first define a finite set of function symbols F_I where I denotes a non-empty subset of $\{1, ..., n\}$. Each F_I is assumed to be of arity equal to the cardinality of I.

Now, to each formula of feature logics we associate a formula in Maher's logic by replacing each sub-formula xf_iy with

$$\bigoplus_{I \subset \{1,...,n\}, i \in I} \exists z_1 ... z_{\#I} . x = F_I(z_1, ... z_{\#I}) \wedge y = z_{i\#I}$$

where $\bigoplus$ denotes exclusive or (i.e. it is true iff one and only one of the arguments is true), $\#I$ denotes the cardinality of I and $i\#I$ denotes the index of i in I, i.e. the number of elements in I which are less or equal to i.

Now using the above mapping we can derive the feature logic axioms form Maher's, i.e. the mapped version of a feature logic axiom is a theorem in Maher's Logic.

There is also an inverse mapping which consists of replacing an equality $x = F_I(y_1, ..., y_k)$ with the conjunction of

$$xf_{m_i}y_i$$

for each m_i in $I = \{m_1, ..., m_k\}$ and

$$\forall z. \neg xf_jz$$

for each j not in I. Note this is enough since any formula involving terms can be reduced (using only the standard equality axioms) to an equivalent one where terms only occur in the above form.

Again, from the feature logic axioms we can derive the Maher's axioms .

Now from the completeness of Maher's axiomatization if follows the completeness of the axiomatization for Feature Logic with a finite number of features and a countable number of atoms.

3 From Terms and Unification to Constrained Terms and Resolution

We assume again the countable set V of variables $x, y, z, ...$, the countable set F of function symbols $f, g, h, ...$ and the set of terms T (cf. Section 1). Let T_0 denote the corresponding set of ground terms.

Associated with a term t is its usual denotation

$$[\![\, t \,]\!] = \{ \sigma t \, \in \, T_0 \, \} \tag{1}$$

where σ denotes a substitution of terms for variables. The unifier t of two terms t' and t" has the following important property:

$$[\![\, t \,]\!] \; = \; [\![\, t' \,]\!] \, \cap \, [\![\, t" \,]\!]$$

We will now extend terms $t \in \; T$ to constrained terms $t : c$ where c is an arbitrary well formed formula of the first order theory of CLG involving only variables occurring in t and take:

$$[\![\, t : c \,]\!] \; = \; \{ \sigma t \in \; T_0 \mid \; \vdash \; \sigma c \; \} \tag{2}$$

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 a unifying substitution σ and to return $\sigma(t' : c' \& c")$. In this case we would obtain exactly the same terms as in the equational case but annotated with the conjunction of all the constraints attached to the resulting unifier

Two interesting properties of unifiers can be used to derive more interesting algorithms. Assume for instance that $t : c$ is a unifier and c is logically equivalent to c', then $t : c'$ is also an unifier. This fact is at the heart of every constraint rewriting system.

Similarly if, for some variable x and term v, we can derive $x = v$ from c, then $[v/x](t : c)$, where $[v/x]$ denotes substitution of v for x, is also an unifier. It is obvious that by reducing c to normal form, it is possible to find all the equalities of the form $x = v$ which can be derived from c, and also decide if c is satisfiable. This strategy, however, suffers from the inherent NP hardness. In the next section we will analyze this problem and see how it can be overcome.

4 The CLG Formal Processing Model

This section presents the CLG formal processing model for constraint resolution. As already said in Section 1 constraints in CLG consist simply of any formula of $\mathcal{RT}$ the first order theory of rational trees as axiomatized by Maher.

Although such formulae are well understood they are, due to the presence of quantifiers, unsuitable for computational purposes. To overcome this problem we introduce a quantifier free constraint language, similar to the one described in [DMV91] and show that any formula of $\mathcal{RT}$ is equivalent to a formula in the constraint language.

We will next turn our attention to the problem of computing all the models of a particular constraint. However, we will diverge from the treatment presented in [DMV91] by showing that it is enough, for this purpose, to consider a restricted form of the constraint language which bears a close resemblance to Feature Logics [Smo89].

We will then present a complete rewriting system for the restricted constraint language. However, since closing a formula under that system is an NP-complete problem, we will consider an incomplete subset of that system, for which the corresponding problem becomes polynomial, and study some of its formal properties.

4.1 The Constraint Language

Let $Vars$ be a countable set of variables x, y, z, ... and F be a countable set of n-ary function symbols f, g, h, ..., $n \geq 0$. A function symbol f with arity n will be denoted by f^n, whenever necessary. A functional symbol of arity 0 will be called an atom and denoted by a, b, ...

We define three classes of objects, *paths*, *values* and *constraints* in the following way:

$$\begin{array}{lcll} p & ::= & \epsilon & \\ & | & p.f_i^n & 1 \leq i \leq n \\ v & ::= & x.p & \\ & | & a & \\ c & ::= & v.f^n & \\ & | & v \dot{=} v & \\ & | & false & \\ & | & true & \\ & | & \neg c & \\ & | & c \wedge c & \\ & | & c \vee c & \end{array}$$

Rather than introducing an independent axiomatization or semantics for the above language we prefer to regard each constraint c as an abbreviation of a formula of $\mathcal{RT}$, e.g.:

$$\begin{aligned} x.f_1^2.g^1 &\longrightarrow \exists z_1 z_2.x = f(g(z_1), z_2) \\ x.f_2^2.g_1^2 \dot{=} y &\longrightarrow \exists z_1 z_2.x = f(z_1, g(y, z_2)) \\ x.f_1^2 \dot{=} y.g_1^2.h_1^1 &\longrightarrow \exists z_1 z_2 z_3.x = f(z_1, z_2) \wedge y = g(h(z_1), z_3) \end{aligned}$$

An important property of this constraint language is that for any formula of $\mathcal{RT}$ we can find an equivalent constraint.

To see this we recall that Maher proves in [Mah88a] that any formula of $\mathcal{RT}$ is equivalent to a boolean combination of *rational basic* formulae which are formulae of the form:

$$\exists u_1, \ldots, u_m.x_1 = t_1 \wedge \ldots \wedge x_n = t_n \wedge u_1 = r_1 \wedge \ldots \wedge u_p = r_p$$

where $p \leq m$ and we can assume that the only variables which occur in the terms t_i and r_j are the u_k. Now it is easy to see that each of the equalities in a basic formula can be replaced by a conjunction of path equalities such that the right hand side of each of the equalities is either an atom or one of the u_k. Then, using these equalities and assuming the original set of equations to be non-*circular* (cf. Section 1.4) it is possible to eliminate the u_k from all the other equalities in a way that in the end each u_k appears in at most one equality. It is easy to see that now the existential quantifiers can be eliminated.

Note that the language defined above differs from the one in [DMV91] by avoiding the presence of complex terms.

4.2 The Restricted Constraint Language

Although the language of the previous section could be directly used, as in [DMV91], for studying the formal properties related to the satisfiablity of constraints, it is convenient and sufficient to restrict ourselves to a subset of it, with the following new definitions for *values* and *constraints*:

$$\begin{array}{lcll} v & ::= & x & \\ & | & a & \\ c & ::= & v.f^n & \\ & | & v = v & \\ & | & v.f_i^n \doteq v & 1 \leq i \leq n \\ & | & false & \\ & | & true & \\ & | & \neg c & \\ & | & c \wedge c & \\ & | & c \vee c & \end{array}$$

Again we can regard any c as a formula of $\mathcal{RT}$ by introducing the abbreviations:

$$\begin{array}{rcl} v.f^n & \longrightarrow & \exists z_1 \ldots z_n . v = f(z_1, \ldots, z_n) \\ v.f_i^n \doteq v' & \longrightarrow & \exists z_1 \ldots z_n . v = f(z_1, \ldots, z_n) \wedge z_i = v' \end{array}$$

As a matter of fact for any constraint c of our original constraint language we can build, by introducing extra variables, a constraint c' of the restricted language such that c is satisfiable iff c' is satisfiable. To build such a formula we first push negation inside so that it only occurs applied to atomic constraints. Since, by introducing new (existentially quantified) variables it is possible to reduce any non-negated atomic constraint to a conjunction of constraints of the restricted language, the only problem remaining is posed by negated atomic constraints which have to be handled as in the following example, where:

$$\neg x.f_1^2.g_1^1 \dot{=} y$$

is replaced by:

$$\neg x.f^2 \vee (x.f_1^2 = z \wedge \neg z.g_1^1 = y)$$

and z is a new variable which does not occur elsewhere.

It is interesting to notice the similarity of this constraint language with Smolka's Feature Logics with f_i^n playing the role of features and $x.f^n$ something akin to divergence constraints. It is thus not surprising that the following definitions bear a strong resemblance to those in [Smo89].

A set of constraints $\mathcal{C}$ is in *solved form* iff

1. every constraint in $\mathcal{C}$ is of one of the forms $x.f^n$, $x.f_i^n \dot{=} v$, $x = v$ or $x \neq v$
2. if $x = v$ is in $\mathcal{C}$ then x occurs exactly once in $\mathcal{C}$
3. if $x.f_i^n \dot{=} v$ and $x.f_i^n = v'$ are in $\mathcal{C}$ then v and v' are identical
4. if $x.f_i^n \dot{=} v$ is in $\mathcal{C}$ the $x.f^n$ is also in $\mathcal{C}$
5. if $x.f^n$ is in $\mathcal{C}$ then there is no constraint in $\mathcal{C}$ of the form $x.g^m$
6. if $x \neq v$ is in $\mathcal{C}$ then v is not identical to x.
7. if for some x, y and v, both $x.f_i^n \dot{=} v$ and $y.f_i^n \dot{=} v$ are in $\mathcal{C}$ then for some j between 1 and n, there is no v' such that $x.f_j^n \dot{=} v'$ and $y.f_j^n \dot{=} v'$ are both in $\mathcal{C}$.

The purpose of the last clause in the previous definition is to force solved forms to contain $x = y$ if for some f^n there is a v_i for each i between 1 and n such that $x.f_i^n = v_i$ and $y.f_i^n = v_i$ hold.

Given a set $\mathcal{C}$ of constraints of the form in 1. above it can be proved that $\mathcal{C}$ is satisfiable iff it can be reduced to solved form using the following set of simplification rules:

1. $\{x = v\} \cup \mathcal{C} \rightarrow \{x = v\} \cup [v/x]\mathcal{C}$ if x is distinct from v and x occurs in $\mathcal{C}$
2. $\{a = x\} \cup \mathcal{C} \rightarrow \{x = a\} \cup \mathcal{C}$
3. $\{x.f_i^n \dot{=} v, x.f_i^n \dot{=} v'\} \cup \mathcal{C} \rightarrow \{x.f_i^n \dot{=} v, v = v'\} \cup \mathcal{C}$

4. $\{x.f_i^n \doteq v\} \cup \mathcal{C} \to \{x.f^n, x.f_i^n \doteq v\} \cup \mathcal{C}$ if $x.f^n$ is not in $\mathcal{C}$

5. if for all $1 \leq i \leq n$ exists v such that $x.f_i^n \doteq v \in \mathcal{C}$ and $y.f_i^n \doteq v \in \mathcal{C}$ then $\mathcal{C} \to \{x = y\} \cup \mathcal{C}$

6. $\{v = v\} \cup \mathcal{C} \to \mathcal{C}$

7. $\{a \neq x\} \cup \mathcal{C} \to \{x \neq a\} \cup \mathcal{C}$

8. $\{a \neq b\} \cup \mathcal{C} \to \mathcal{C}$ if a and b are distinct atoms.

Given a set $\mathcal{M}$ of non-negated atomic constraints in solved form and a set of constraints $\mathcal{C}$ we will say that $\mathcal{M}$ is a *partial model* of $\mathcal{C}$ iff every model of $\mathcal{C}$ is a model of $\mathcal{M}$. We will say that $\mathcal{M}$ is a *minimal* partial model of $\mathcal{C}$ iff it is a partial model of $\mathcal{C}$ and no proper subset of $\mathcal{M}$ in solved form is a partial model of $\mathcal{C}$. By using disjunctive forms it can be proved that any set of constraints $\mathcal{C}$ admits at most a finite number of minimal models.

4.3 Rewriting Constraints

Constraint processing in *CLG* is based on the use of a rewriting system to produce from a set of constraints $\mathcal{C}_0$ a partial model $\mathcal{M}$ and a smaller set of constraints $\mathcal{C}$ such that any minimal model of $\mathcal{C}_0$ can be obtained by conjoining (i.e. "unifying") a minimal model of $\mathcal{C}$ with $\mathcal{M}$ and moreover for any minimal model of $\mathcal{C}$ the reunion $\mathcal{M} \cup \mathcal{C}$ is satisfiable.

We start by defining a set of rewriting rules $\longrightarrow_{\mathcal{M}}$ for values and constraints

$$\begin{array}{ll}
x \longrightarrow_{\mathcal{M}} v & \text{if } x = v \text{ is in } \mathcal{M} \text{ and } x \text{ occurs in } \mathcal{C} \\
x.f_i^n \longrightarrow_{\mathcal{M}} v & \text{if } x.f_i^n \doteq v \text{ is in } \mathcal{M} \text{ and } x.f_i^n \text{ occurs in } \mathcal{C} \\
c \longrightarrow_{\perp} false & \\
\neg true \longrightarrow_{\mathcal{M}} false & \\
\neg false \longrightarrow_{\mathcal{M}} true & \\
\neg\neg c \longrightarrow_{\mathcal{M}} c & \\
\neg(c_1 \wedge c_2) \longrightarrow_{\mathcal{M}} \neg c_1 \vee \neg c_2 & \\
\neg(c_1 \vee c_2) \longrightarrow_{\mathcal{M}} \neg c_1 \wedge \neg c_2 & \\
true \wedge c \longrightarrow_{\mathcal{M}} c & \\
false \wedge c \longrightarrow_{\mathcal{M}} false & \\
c \wedge true \longrightarrow_{\mathcal{M}} c & \\
c \wedge false \longrightarrow_{\mathcal{M}} false & \\
true \vee c \longrightarrow_{\mathcal{M}} true &
\end{array}$$

$false \vee c \longrightarrow_{\mathcal{M}} c$	
$c \vee true \longrightarrow_{\mathcal{M}} true$	
$c \vee false \longrightarrow_{\mathcal{M}} c$	
$(c_1 \wedge c_2) \wedge c_3 \longrightarrow_{\mathcal{M}} c_1 \wedge (c_2 \wedge c_3)$	
$\neg c_1 \wedge c_2 \longrightarrow_{\mathcal{M}} c_2 \wedge \neg c_1$	if c_1 is an atomic constraint and c_2 is not a conjunction of negated atomic constraints
$c_1 \wedge c_2 \longrightarrow_{\mathcal{M}} c_2 \wedge c_1$	if c_2 is atomic and c_1 is not
$c_1 \wedge (c_2 \wedge c_3) \longrightarrow_{\mathcal{M}} c_2 \wedge (c_1 \wedge c_3)$	if c_2 is atomic and c_1 is not
$a = b \longrightarrow_{\mathcal{M}} false$	if a and b are distinct atoms
$a = x \longrightarrow_{\mathcal{M}} x = a$	
$v = v \longrightarrow_{\mathcal{M}} true$	
$x.f^n \longrightarrow_{\mathcal{M}} true$	if $x.f^n$ is in $\mathcal{M}$
$x.f^n \longrightarrow_{\mathcal{M}} false$	if $x.g^m$ is in $\mathcal{M}$
$x = a \longrightarrow_{\mathcal{M}} false$	if $x.g^m$ is in $\mathcal{M}$
$a.f^n \longrightarrow_{\mathcal{M}} false$	
$a.f_i^n \doteq v \longrightarrow_{\mathcal{M}} false$	
$x.f_i^n \doteq v \longrightarrow_{\mathcal{M}} false$	if $x.g^m$ is in $\mathcal{M}$
$x = y \longrightarrow_{\mathcal{M}} false$	if $\mathcal{M} \cup \{x = y\} \rightarrow \bot$
$x = v \wedge c \longrightarrow_{\mathcal{M}} x = v \wedge c'$	if $c \longrightarrow^{\star}_{\mathcal{M}\cup\{x=v\}} c'$
$x.f_i^n \doteq v \wedge c \longrightarrow_{\mathcal{M}} x.f_i^n \doteq v \wedge c'$	if $c \longrightarrow^{\star}_{\mathcal{M}\cup\{x.f_i^n \doteq v\}} c'$
$x.f^n \wedge c \longrightarrow_{\mathcal{M}} x.f^n \wedge c'$	if $c \longrightarrow^{\star}_{\mathcal{M}\cup\{x.f^n\}} c'$
$(c_1 \vee c_2) \wedge c_3 \longrightarrow_{\mathcal{M}} (c_1 \wedge c_3) \vee (c_2 \wedge c_3)$	if both c_1 and c_2 are not $\mathcal{M}$-independent with c_3.

Note that in the rules above $\mathcal{M} \cup \mathcal{C}$ denotes the solved form of the union of $\mathcal{M}$ and $\mathcal{C}$ if one exists or $\bot$ if that union is not satisfiable. The last rule must apply only when both c_1 and c_2 have variables in common with c_3, eventually through "bindings" in $\mathcal{M}$. In order to formalize this notion we need the following definitions. Given two variables x and y, *x depends on y under $\mathcal{M}$* , $x \lhd_{\mathcal{M}} y$, iff

1. $x \lhd_{\mathcal{M}} x$
2. if $y.f_i^n \doteq x \in \mathcal{M}$ then $x \lhd_{\mathcal{M}} y$
3. if $x \lhd_{\mathcal{M}} z$ and $z \lhd_{\mathcal{M}} y$ then $x \lhd_{\mathcal{M}} y$

A *slot* s is a variable or an expression of the form $x.f^n$ or $x.f_i^n$ for any variable x, function symbol f^n and i, $1 \leq i \leq n$. Given a constraint $\mathcal{C}$, a variable x occurs *properly in* $\mathcal{C}$, $x\sharp\mathcal{C}$, if it occurs in an atomic constraint $x = v$, $y = x$ or $s \doteq x$. This notion can be extend to *slots*. If a constraint $\mathcal{C}$ is closed under $\longrightarrow_{\mathcal{M}}$, any slot that occurs in $\mathcal{C}$ can only occur in $\mathcal{M}$ in the right hand side of a equation (and so it must be a variable). On the other hand if $x.f_i^n$ or $x.f^n$ occur in $\mathcal{C}$, x can occur in $\mathcal{M}$ in the right hand side of an equation or in an expression $x.f_j^n$, $j \neq i$ or $x.f^n$, and in any case the values of these *slots* are not bounded to each other. So we are left with the "dependences" under $\mathcal{M}$ of variables that properly occur in $\mathcal{C}$. Let $Vars_{\mathcal{M}}(\mathcal{C}) = \{y \mid y \lhd_{\mathcal{M}} x, x\sharp\mathcal{C}\}$. Then, two constraints $\mathcal{C}$ and $\mathcal{C}'$ are $\mathcal{M}$-*independent* iff

1. $\mathcal{C}$ and $\mathcal{C}'$ are closed under $\longrightarrow_{\mathcal{M}}$
2. any variable x which occurs in $Vars_{\mathcal{M}}(\mathcal{C})$ does not occur in $Vars(\mathcal{C}') \cup Vars_{\mathcal{M}}(\mathcal{C}')$ and vice-versa
3. for every variable x which only occurs in $\mathcal{C}$ and $\mathcal{C}'$ in atomic constraints of the form $x.f^n$ or $x.f_i^n \doteq v$ for some f^n, if $x.f_i^n$ occurs in $\mathcal{C}$, $x.f_i^n$ does not occur in $\mathcal{C}'$, $1 \leq i \leq n$, and vice-versa

We now define a rewriting system for pairs $\langle \mathcal{M}, \mathcal{C} \rangle$ by first closing $\mathcal{C}$ under $\longrightarrow_{\mathcal{M}}$ and then using the following rules:

$$\begin{array}{l}
\langle \mathcal{M}, \mathcal{C} \cup \{false\} \rangle \rightarrow \langle \bot, \emptyset \rangle \\
\langle \mathcal{M}, \mathcal{C} \cup \{true\} \rangle \rightarrow \langle \mathcal{M}, \mathcal{C} \rangle \\
\langle \mathcal{M}, \mathcal{C} \cup \{x = v\} \rangle \rightarrow \langle \mathcal{M} \cup \{x = v\}, \mathcal{C} \rangle \\
\langle \mathcal{M}, \mathcal{C} \cup \{x.f^n\} \rangle \rightarrow \langle \mathcal{M} \cup \{x.f^n\}, \mathcal{C} \rangle \\
\langle \mathcal{M}, \mathcal{C} \cup \{x.f_i^n \doteq v\} \rangle \rightarrow \langle \mathcal{M} \cup \{x.f_i^n \doteq v\}, \mathcal{C} \rangle
\end{array}$$

with the convention that after each application of one of the rewrite rules the new partial model is reduced to solved form and the resulting set of constraints is closed under $\longrightarrow_{\mathcal{M}}$.

We will now sketch the proof of the claims made above about the rewriting system.

We will first argue that if given an initial set of constraints $\mathcal{C}_0$ we apply the rewriting system to $\langle \emptyset, \mathcal{C}_0 \rangle$ to obtain $\langle \mathcal{M}, \mathcal{C} \rangle$ then $\mathcal{C}_0$ (more precisely the conjunct of all the constraints in $\mathcal{C}_0$) is equivalent to $\mathcal{M} \cup \mathcal{C}$. As a matter of fact this follows from the fact that each rewrite rule is associated with a similar meta-theorem of First Order Logic and/or the axioms of $\mathcal{RT}$.

As for the other property of the rewriting system, namely that the minimal models of $\mathcal{C}_0$ are obtained by conjoining $\mathcal{M}$ with those of $\mathcal{C}$, we will only sketch the argument of the proof which follows from the fact that if some minimal model $\mathcal{C}'$ of $\mathcal{C}$ was inconsistent with $\mathcal{M}$ then, after reducing $\mathcal{C}$ to disjunctive form , at least one of the disjunctions should subsume $\mathcal{C}'$ and thus should be inconsistent with $\mathcal{M}$ while admitting itself a model.

Now, since every atomic formula occurring in the disjunction already occurred in $\mathcal{C}$ it is possible to derive a contradiction with the hypothesis that $\mathcal{C}$ was closed under $\longrightarrow_{\mathcal{M}}$. To see this we notice that each disjunct in the disjunctive form of $\mathcal{C}$ can be regarded as a set of (possibly negated) atomic constraints $\mathcal{C}''$ of the form above. Now if $\mathcal{M} \cup \mathcal{C}''$ was unsatisfiable then some sequence of simplification rules should lead to a clash. Now, noticing that any atomic formula in $\mathcal{C}''$ must already be present in $\mathcal{C}$, it is easy to check that any sequence of simplification rules would involve only formulae from $\mathcal{C}''$ and a clash with a formula in $\mathcal{M}$ would thus be impossible.

The other interesting property of the rewriting system above is that it is complete in the sense that unless it produces $\perp$ as the final model then $\langle \mathcal{M}, \mathcal{C} \rangle$ is satisfiable. This is achieved mainly by the last rule above for $\longrightarrow_{\mathcal{M}}$. However, even if this rule attempts to limit the number of cases where it applies to an essential minimum, it causes NP-completeness of the rewriting process since it can lead to an exponential growth of the constraints. If we omit this rule then the rewrite process becomes polynomial, although incomplete. The *CLG* systems uses the incomplete rewriting system most of the time and the rule causing the NP-completeness is only used at critical points in the process. In the next subsection we will study a class of constraints for which we can prove a weak completeness result for the polynomial rewriting system.

An example of the application of this rewrite system to a concrete case can be found in [DV92].

4.4 Weak Completeness

Since *CLG* systems rely heavily on incomplete rewrites it is convenient to address the problem of characterizing classes of constraints they are able to solve in a complete way. This characterization can then be used to transform input constraints into equivalent constraints more adapted to the incomplete rewrite system.

We present here a class of constraints that, when rewritten by the incomplete system, produce a constraint belonging to the same class and which is satisfiable unless it reduces to $false$. This class is general, in the sense that for any constraint c we can find an equivalent constraint belonging to the class. Unfortunately, although the class is closed under disjunction, it is not so under conjunction. Nevertheless it is still interesting from a practical point of view since it provides for a higher degree of constraint resolution.

The class D can be described by

$$\begin{array}{lll} D & ::= & F \\ & \mid & D \vee F \\ F & ::= & NF \\ & \mid & A \wedge D \\ NF & ::= & true \\ & \mid & \neg A \\ & \mid & \neg A \wedge NF \end{array}$$

where A denotes any atomic formula. This class excludes formulae involving conjunctions of disjunctions and allows negation to apply only to atomic formulae, and in this case at the inner- most and right-most levels of an expression.

5 A Processing Model for CLG as $CLP(\mathcal{RT})$

In this section we present the formal processing model for CLG seen as a $CLP(\mathcal{RT})$ program, which can be seen as a generalization of the constraint rewriting system and also as an extension of the Andorra Model [War88, SWY91].

Assume a $CLP(\mathcal{RT})$ consists of clauses of the form

$$h \leftarrow c \wedge b$$

where c is a constraint, h is the head of the clause and b is a body consisting of a, possibly empty, conjunction of goals. In the sequel we will consider atoms (in the logic connotation of the word) as a particular case of terms.

To define the semantics of a $CLP(\mathcal{RT})$ program, we will use a non-deterministic rewriting system on triples of the form

$$\langle M, C, G \rangle$$

where M is a partial model, C is a set of constraints and G is a *goal expression* of the following form:

$$(c_1^1 \wedge b_1^1 \parallel c_2^1 \wedge b_2^1 \parallel \ldots) \wedge (c_1^2 \wedge b_1^2 \parallel c_2^2 \wedge b_2^2 \parallel \ldots) \wedge \ldots$$

where, as before, the c_j^i denote constraints and the b_j^i bodies.

Note that $\parallel$ is used to denote alternative ways of building a model and is, from a logical point of view, just $\vee$. The idea behind this notation is that each $\parallel$ connected group originates from the clause bodies of a single predicate.

The purpose of the rewrite system is to build specifications of models of an initial goal g by reducing an initial triple $\langle \emptyset, \emptyset, g\rangle$ to a triple $\langle M, C, true\rangle$.

The rewriting system for triples is defined by first closing every constraint in the triple using $\rightarrow_M$, then using the analogous of the rules for pairs $\langle M, C\rangle$, and then one of the following rules:

$$\begin{array}{l}
\langle M, C, (false \wedge b \parallel B) \wedge G\rangle \rightarrow \langle M, C, B \wedge G\rangle \\
\langle M, C, (c \wedge g_1 \wedge \ldots \wedge g_n) \wedge G\rangle \rightarrow \\
\qquad \langle M, C \cup \{c\},\ \ (g_1 = h_1^1 \wedge c_1^1 \wedge b_1^1 \parallel \ldots \parallel g_1 = h_{k_1}^1 \wedge c_{k_1}^1 \wedge b_{k_1}^1) \wedge \\
\qquad\qquad\qquad\quad \ldots \\
\qquad\qquad\qquad\quad (g_n = h_1^n \wedge c_1^n \wedge b_1^n \parallel \ldots \parallel g_n = h_{k_n}^n \wedge c_{k_n}^n \wedge b_{k_n}^n) \wedge G\rangle
\end{array}$$

assuming

$$h_j^i \leftarrow c_j^i \wedge b_j^i$$

are new instances of all the clauses for predicate g_i.

Note that the rewriting system thus defined, while not necessarily terminating, is convergent, and, as in the Andorra Model, essentially performs all the deterministic goal expansions while collecting constraints.

However, to enable reduction of the third component to *true* we need, in the general case, the following non-deterministic rewrite rule:

$$\langle M, C, (c \wedge b \parallel As) \wedge G\rangle \rightarrow \langle M, C, (c \wedge b) \wedge G\rangle$$

This completes the description of the formal processing model of *CLG*. Some implementation considerations can be found in [DV89, BDMV90, DMV91]. In a forthcoming article we elaborate on the practical importance of program transformation techniques for *CLG* systems.

References

[BDMV90] 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.

[Col82] Colmerauer, Alain, 1982. Prolog and infinite trees. In: Logic Programming, S. A. Tarnlund (ed.), Academic Press, New York, 231-251.

[DMV91] Damas, Luis, Nelma Moreira and Giovanni B. Varile, 1991. The formal and processing models of CLG. In: Fifth Conference of the European Chapter of the Association for Computational Linguistics, Berlin, 173-178.

[DV89] 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, .

[DV92] Damas, Luis and Giovanni B. Varile, 1992. On the Satisfiability of Complex Constraints. To appear in: Proceedings of COLING92, Nantes, France.

[HS88] M. Höhfeld and G. Smolka,1988. Definite relations over constraint languages, LILOG-Report 53, IBM Deutschland GmbH, Stuttgart.

[JL87] Jaffar, J., J-L. Lassez, 1987. Constraint logic programming. In: Symposium on Principles of Programming Languages, Munich.

[JL88] 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.

[Joh90] Johnson, Mark, 1990. Expressing Disjunctive and Negative Feature Constraints with Classical First-Order Logic. In: ACL Proceedings, 28th Annual Meeting, 173–179.

[KR86] Kasper, Robert and William Rounds. 1986. A logical semantics for feature structure. In : ACL Proceedings, 24th Annual Meeting, 257–266.

[Mah88a] Maher, Michael J., 1988. Complete axiomatization of the algebras of finite, rational and infinite trees. Technical Report, IBM Thomas Watson Research Center, Yorktown Heights, New York.

[Mah88b] Maher, Michael J., 1988. Complete axiomatization of the algebras of finite, rational and infinite trees. In: Proceedings of the 3rd Annual Symposium on Logic in Computer Science, 348-457.

[RK86] Rounds, William C. and Robert Kasper. 1986. A complete logical calculus for record structures representing linguistic information. In : Symposium on Logic in Computer Science, IEEE Computer Society

[Sho67] Shoenfield, Joseph R., 1967. Mathematical Logic. Addison-Wesley.

[Smo88] Smolka, Gert, 1988. A Feature Logic with Subsorts, LILOG Report 33, IWBS, IBM Deutschland.

[Smo89] Smolka, G. 1989. Feature Constraint Logics for Unification Grammars, LILOG Report 93, IWBS, IBM Deutschland.

[SWY91] Santos, Vitor Costa, David H.D. Warren and Rong Yang, 1991. The Andorra-I Preprocessor: Supporting Full Prolog on the Basic Andorra Model. Proceedings of the eight International Conference on Logic Programming, M.I.T., Cambridge, Massachusetts, 443-456.

[War88] Warren, D.H.D., 1988. The Andorra Model. Gigalips Workshop, University of Manchester, Manchester, England.