TR-592

The Hierarchical Constraint Logic Language CHAL

by
K. Satoh & A. Aiba

September, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

The Hierarchical Constraint Logic Language: CHAL

Ken Satoh and Akira Aiba
Institute for New Generation Computer Technology
4-28 Mita 1-Chome Minato-ku, Tokyo 108 Japan
phone: +81-3-456-2514
email: ksatoh@icot.or.jp

Abstract

This paper presents the hierarchical constraint logic programming language CHAL(Contrainte Hierarchique avec Logique) which can handle non-required constraints.

Constraint logic programming(CLP) is a paradigm which extends logic programming by introducing constraint solving mechanism to increase expressiveness. We so far developed a CLP language called CAL(Contrainte avec Logique) [11].

However, as many CLP language, CAL only considers required constraints which every solution must satisfy. However, in the area of scheduling and planning, there are non-required constraints(soft constraints) which provide preference of possible solutions. We have proposed a logical foundation of soft constraints [13] by using a model-theoretical meta-language.

In this paper, we extend CAL to handle a restricted class of soft constraints defined in [13] and propose CHAL.

1 Introduction

Recently, a paradigm called *Constraint Logic Programming* (CLP) has been studied by many researchers [4, 9, 5]. This paradigm extends logic programming by including constraint solving mechanism to increase expressiveness.

Whereas ordinary logic programming languages such as Prolog are executed by unification, CLP is based on constraint solving mechanism. In this sense, constraint solving can be viewed as a generalization of unification.

The idea of programming with constraints has been proposed in the literature [7, 15]. However, a combination of logic programming and constraints gives fully declarative semantics of constraint programming because logic programming has not only operational semantics but also declarative semantics and it can preserve a declarative nature of constraint programming. This declarative semantics helps to understand program easily.

We so far developed a constraint logic programming language (CLP language) called CAL(Contrainte avec Logique) [11]. CAL is based on constraint solvers which can handle algebraic constraints represented in multi-variate polynomial equations and Boolean constraints expressed in Boolean equations using Gröbner bases [2]. Unlike other constraint solvers, we have a canonical form of constraints which gives a simplified form of constraints.

However, as many CLP languages, CAL only considers *hard constraints* which every solution must satisfy. However, in the area of *synthesis* problem such as job shop scheduling, circuit design and planning, there is another kind of constraints, that is, *soft constraints* which provides preferences over solutions [6, 8, 14].

We have proposed a logical foundation of soft constraints [13] by using a meta-language [12] which expresses an interpretation ordering. The idea of formalizing soft constraints is as follows. Let hard constraints be represented in the first-order formulas. Then an interpretation which satisfies all of those first-order formulas can be regarded as a solution. Then soft constraints can be regarded as an order over those interpretations because soft constraints represent criteria over solutions to choose the most preferable ones. We use a meta-language which presents a preference order directly. This meta-language can be translated into the second-order formula to provide a syntactical definition of the most preferred solutions.

Although this framework is rigorous and declarative, it is not computable in general because it is defined by second-order formula. Therefore, we have to restrict a class of constraints so that these constraints are computable.

In this paper, we propose an extension of CAL which handles a restricted class of soft constraints. We call this language CHAL(Contrainte Hierarchique avec Logique).

In section 2, we introduce CAL and its basic mechanism, Buchberger Algorithm. In section 3, we discuss soft constraints. In section 4, we show CHAL and gives relationship between a solution in CHAL and the most preferable solution defined in soft constraints defined in second order language. In section 5, we show examples of CHAL. In section 6, we give a conclusion.

2 CAL(Contrainte avec Logique)

This section briefly reviews the constraint logic programming language CAL [11] developed in ICOT. Unlike almost all the other CLP languages proposed so far, CAL has a constraint solver which handles multi-variate polynomial equations and boolean equations based on Gröbner bases [2] and computes a canonical form of constraints.

2.1 Language and domain

CAL program consists of rules of the from:

$$h(\mathbf{t}) :- b_1(\mathbf{t}_1), ..., b_n(\mathbf{t}_n).$$

where $h, b_1, ..., b_n$ are predicate symbols or constraint symbols and $\mathbf{t}$, $\mathbf{t}_1, ..., \mathbf{t}_n$ are lists of terms. If c is a constraint symbol and $\mathbf{t}$ is a list of terms, $c(\mathbf{t})$ is called an (atomic) constraint. Constraint symbols and terms used in constraints vary with respect to constraint solvers.

In algebraic CAL which handles algebraic constraints represented in multi-variate polynomial equations, the following constraint symbols and terms can be used.

1. A set of constraint symbols = $\{=\}$
2. A set of terms = a set of composite terms composed by function symbols $*$, $+$, variable symbols, constant symbols and fractions.

The meaning of those symbols is as follows.

1. $X = Y$ means that X is equal to Y.
2. $X * Y$ is the rational number of the result of multiplication of X and Y.
3. $X + Y$ is the rational number of the result of addition of X and Y.
4. fraction is the rational number it denotes.

For example, the program which computes the area S of a triangle from its height H and baseline length A is expressed as the following program.

```
sur(H,A,S) :- A*H=2*S.
```

In Boolean CAL which handles Boolean constraints expressed in Boolean equations, the following constraint symbols and terms can be used.

1. A set of constraint symbols = $\{=\}$
2. A set of terms = a set of composite terms composed by function symbols $\wedge$, $\oplus$, variable symbols, constant symbols and 1 and 0.

The meaning of those symbols is as follows.

1. $X = Y$ means that X is equal to Y.

2. $X \wedge Y$ is the result of conjunction of X and Y.

3. $X \oplus Y$ is the result of exclusive disjunction of X and Y.

4. 1 is true and 0 is false.

In the actual system, other logical connectives such as $\vee$(disjunction), $\supset$(implication), $\equiv$(equivalence) and $\neg$(negation) are also included as function symbols. However, since it is well known that they can be defined from the above symbols, we have omitted them for simplicity.

For example, the program which computes disjunction C of A and B is expressed as the following program.

```
or(A,B,C) :- A⊕NA=0, B⊕NB=0, C⊕(NA∧NB)=0.
```

2.2 Buchberger Algorithm and Gröbner Bases

CAL is based on Buchberger Algorithm [2] to solve constraints. This algorithm has been widely used in the field of computer algebra over the past few years. In this subsection, we describe the theoretical background of Gröbner bases and the Buchberger algorithm for algebraic CAL. Boolean CAL uses a modified version of Buchberger algorithm for Boolean equations.

Without loss of generality, we can assume that all polynomial equations are in the form of $p = 0$. Let $E = \{p_1 = 0, \ldots, p_n = 0\}$ be a system of polynomial equations, and I the ideal in the ring of all the polynomials generated by $\{p_1, \ldots, p_n\}$. The following close relation between the elements of I and the solutions of E is well known as the Hilbert zero point theorem.

Theorem 1 *Let p be a polynomial. Every solution of E is also a solution of $p = 0$, if and only if there exists a natural number n such that p^n is an element of I.*

Moreover, the following corollary is important to determine the satisfiability of constraints.

Corollary 1 *E has no solution if and only if $1 \in I$.*

Thus, the problem of checking satisfiability of constraints is reduced to the problem of determining whether 1 belongs to the generated ideal. Buchberger gave an algorithm to determine whether a polynomial belongs to the ideal. A rough sketch of the algorithm is as follows (see [2] for a precise definition).

Let there be a certain ordering among monomials and let a system of polynomial equations be given. An equation can be considered a rewrite rule which rewrites the greatest monomial in the equation to the polynomial consisting of the remaining monomials. For example, if the ordering is lexicographic, a polynomial equation, $Z - X + B = A$, can be considered as a rewrite rule, $Z \rightarrow X - B + A$. Two rewrite rules whose left hand sides are not mutually prime are said to *overlap*. In this case,

the least common multiple (LCM) of their left hand sides can be rewritten in two ways by these two rules, which may produce different results. The resulting pair is called a *critical pair*. If further rewriting does not succeed in converging a critical pair, the pair is said to be *divergent* and is added to the system of equations. By repeating this procedure, we can eventually obtain a confluent rewriting system. The confluent rewriting system thus obtained is called a *Gröbner base* of the original system of equations. The following theorem establishes the relationship between ideals and Gröbner bases.

Theorem 2 *Let R be a Gröbner base of a system of equations $\{p_1 = 0, \ldots, p_n = 0\}$, and let I be an ideal generated by $\{p_1, \ldots, p_n\}$. A polynomial, p, belongs to I if and only if p is rewritten to 0 by R.*

From the above corollary and the above theorem, if R contains rewrite rule $1 \rightarrow 0$, then E has no solution, that is, constraints are not satisfiable.

Moreover, the following theorem guarantees the validity of considering the reduced Gröbner bases as the canonical forms of constraints. A Gröbner base is said to be *reduced* if it has no two rules, one of which rewrites the other.

Theorem 3 *Suppose that the ordering among monomials is fixed. Let E and F be systems of equations. Then if the ideal generated from E is the same as that from F, then the reduced Gröbner base of E is same as that of F.*

3 Soft Constraints

Although CAL extends logic programming to include constraints, it only includes required constraints. However, in the domain of scheduling or planning, there is another kind of constraints which we call *soft constraints* and expresses preference over possible solutions which are confined by required constraints.

For example, consider the following meeting scheduling problem for the president, the vice president and the manager in the company.

1. The president *must* attend the meeting.
2. The vice president should *preferably* attend the meeting.
3. The manager also should *preferably* attend the meeting. However, the schedule of the vice president is *prioritized* to the schedule of the manager.

The first condition is a required constraint and a solution must satisfy the condition. However, the second and the third conditions are soft constraints and are regarded as criteria to choose the most preferable models among the solutions of hard constraints. In this case, if there are solutions both of which satisfy the hard constraint, we choose solutions which also satisfy the soft constraints. However, if there is no solution which satisfies soft constraints, we just ignore those soft constraints. Soft constraints, therefore, may not always be satisfied.

Looking into the third conditions, there is a note that the second condition is prioritized to the third condition. If there are two solutions in one of which the second condition is satisfied and the third is not satisfied, and in the other of which

the third condition is satisfied and the second is not, then we choose the former because of the priority.

Consider another situation where a value in a part of a solution should be as large as possible. In this case, we can regard those conditions as a criterion which chooses a solution which has a larger value for that part.

To summarize, there are three kinds of soft constraints stated below. Let σ, θ be possible solutions which satisfy hard constraints and C_σ, C_θ be a set of soft constraints which are satisfied by σ and θ respectively.

1. **Soft constraints without priorities:**
   In this case, if $C_\sigma \subset C_\theta$ (C_θ is a strict super set of C_σ), then θ is better than σ. However, if $C_\sigma \not\subset C_\theta$, we cannot say which solution is better.
2. **Soft constraints with priorities:**
   In this case, we may be able to distinguish solutions in the case of $C_\sigma \not\subset C_\theta$. Let the most preferable constraints which are satisfied by σ, θ be C_σ^1, C_θ^1 respectively, and the second most preferable constraints which are satisfied by σ, θ be C_σ^2, C_θ^2 respectively,..., and the k-th most preferable constraints which are satisfied by σ, θ be C_σ^k, C_θ^k respectively. Then, θ is better than σ if one of the following conditions is satisfied.
   (a) $C_\sigma^1 \subset C_\theta^1$
   (b) There exists j such that for all $i = 1, ..., j-1$, $C_\sigma^i = C_\theta^i$ and $C_\sigma^j \subset C_\theta^j$.
3. **General soft constraints:**
   Any order over solutions can be regarded as a general case of soft constraints. If we can define an order of solutions, we choose the most preferable solutions in the order. For example, if we would like to choose the solution a part of which is bigger, then we can express the order as follows. Let values of the part of σ, θ be V_σ, V_θ. If $V_\sigma < V_\theta$, θ is better than σ.

Let S_0 be a set of the solutions satisfying hard constraints and $<$ be an order over solutions. Then, the set of most preferable solutions can be defined as follows:

$$\{\sigma | \sigma \in S_0 \text{ and there exists no } \theta \in S_0 \text{ such that } \theta < \sigma\}.$$

where smaller solutions are preferable solutions.

We can paraphrase the above definition into first-order logic as follows. Hard constraints can be regarded as the first-order axiom set which solutions must satisfy. Then a solution to those hard constraints becomes an interpretation which satisfies the axiom set, and soft constraints can be regarded as providing an order over those interpretations, and the most preferred solutions are the most preferred interpretations in that order. Then, we can define a set of the most preferable solutions as follows. Let C be a formula which represents a conjunction of hard constraints, and M, M' be logical interpretations, and $\prec$ be an order over interpretations.

$$\{M | M \models C \text{ and } M \text{ is comparable with } M' \text{ and there exists no } M' \text{ such that } M' \models C \text{ and } M' \prec M\}.$$

where smaller interpretations are preferable.

Then, we can see that the logical interpretation in the above set is minimal model with respect to the order $\prec$.

In [13], we use a meta-language [12] which presents a preference order directly. This meta-language can be translated into the second-order formula to provide a syntactical definition of the most preferred solutions.

4 CHAL(Contrainte Hierarchique avec Logique)

Although the logical definition of the soft constraints proposed in [13] is very rigorous, it is not computable in general because the definition is based on the second-order logic.

We restrict the domain so that constraint solver in CAL can be used to calculate the most preferable solutions. And also we restrict forms of constraints to only atomic constraints.

CHAL program consists of rules of the form:

$$h: -b_1, ..., b_n$$

where $h, b_1, ..., b_n$ are predicates or constraints or labeled constraints. Labeled constraints is of the form:

$$\textbf{label}\ C$$

where C is a complex constraint in which only domain-dependent functional symbols can be allowed as functional symbols and **label** is a label which expresses strength of the complex constraint C.

A complex constraint is a disjunction of conjunctions of domain-dependent constraints of the form:

$$((c_{11}, c_{12}, ..., c_{1n_1}); (c_{21}, c_{22}, ..., c_{2n_2}); ...; (c_{m1}, c_{m2}, ..., c_{mn_m}))$$

where $1 \leq m$ and $0 \leq n_i$ and ';' expresses a disjunction and ',' expresses a conjunction and c_{ij} is an atomic constraint whose constraint symbol is domain-dependent.

The operational semantics for CHAL is similar to CAL except manipulating constraint hierarchy. In CHAL, we accumulate non-required constraints to form constraint hierarchy while executing CAL until CAL gives an answer with substitution of variables and the canonical form of required constraints. Then, we solve constraint hierarchy with required constraints.

At the point of obtaining constraint hierarchy, there is a relationship between constraint hierarchy and the most preferable solutions in the previous section.

We regard constraint hierarchy as soft constraints with priority. Let constraints which should be satisfied in the first place be

$$C^1_1, ..., C^1_{m_1},$$

and constraints which should be satisfied in the second place be

$$C^2_1, ..., C^2_{m_2}, ...$$

and constraints which should be satisfied in the k-th place be

$$C^k_1, ..., C^k_{m_k}, ...$$

Let M' and M be an interpretation with considered domain. Then, order over interpretations defined by the above constraint hierarchy is defined by the meta-language as follows:

$$(M' < M) \stackrel{\text{def}}{=} (M' \leq M) \wedge \neg(M \leq M').$$

where $M' \leq M$ is an abbreviation of $(M' \leq^1 M) \wedge ... \wedge (M' \leq^k M)$ and $M' \leq^i M$ $(i \geq 2)$ is an abbreviation of the following expression:

$$(\bigwedge_{j=1}^{i-1} \bigwedge_{l=1}^{m_j}((M' \models C_l^j) \equiv (M \models C_l^j))) \supset (\bigwedge_{l=1}^{m_i}((M' \models C_l^i) \supset (M \models C_l^i))),$$

and $M' \leq^1 M$ is an abbreviation of the following expression:

$$(\bigwedge_{l=1}^{m_1}((M' \models C_l^1) \supset (M \models C_l^1))).$$

This relation means that interpretations which satisfy $C_1^1, ..., C_{m_1}^1$ as much as possible is preferable and if there are interpretations which satisfy the same formulas in the first place, then interpretations which satisfy $C_1^2, ..., C_{m_2}^2$ as much as possible are preferable and, ... if there are interpretations which satisfy the same formulas in the $(k-1)$-th place, then interpretations which satisfy $C_1^k, ..., C_{m_k}^k$ as much as possible are preferable.

Then most preferable solutions with respect to the above ordering is a model M with considered domain which satisfies required constraints and there is no model M' such that M' satisfies the required constraints and M' is preferred to M in the above relation, that is, $M' < M$.

We can give a syntactic definition of the most preferable solutions from the result in [13]. Let A be the axioms of the considered domain and $\mathbf{x}$ be a tuple of all free variables contained in required constraints and soft constraints and $RC(\mathbf{x})$ be a conjunction of required constraints and $C_1^i(\mathbf{x}), ..., C_{m_i}^i(\mathbf{x})$ $(i = 1, ..., k)$ be soft constraints. Then, the following is a syntactic definition of the most preferable solutions.

$$A \wedge RC(\mathbf{x}) \wedge \neg\exists\mathbf{y}(RC(\mathbf{y}) \wedge (\mathbf{y} \leq \mathbf{x}) \wedge \neg(\mathbf{x} \leq \mathbf{y})) \qquad (\mathrm{P})$$

where $\mathbf{y} \leq \mathbf{x}$ is an abbreviation of $(\mathbf{y} \leq^1 \mathbf{x}) \wedge ... \wedge (\mathbf{y} \leq^k \mathbf{x})$ and $\mathbf{y} \leq^i \mathbf{x}$ is an abbreviation of the following formula:

$$(\bigwedge_{j=1}^{i-1} \bigwedge_{l=1}^{m_j}(C_l^j(\mathbf{x}) \equiv C_l^j(\mathbf{y}))) \supset (\bigwedge_{l=1}^{m_i}(C_l^i(\mathbf{x}) \supset C_l^i(\mathbf{y}))).$$

Adapted from the result in [13], we can show the following theorem.

Theorem 4 *M is a most preferable solution w.r.t $RC(\mathbf{x})$ and the order $<$ if and only if M is a model of the formula (P).*

We show an algorithm for solving constraint hierarchy in Appendix. Inputs of the algorithm are required constraints and constraint hierarchy and its output is all maximal consistent set of constraints simplified by the constraint solver which correspond with all the most preferable solutions.

The algorithm has already been implemented on PSI(Personal Sequential Inference) Machine developed by ICOT. CHAL program is complied into ESP language which is a dialect of PROLOG augmented by object-oriented feature and run based on Buchberger algorithm of calculating Gröbner bases.

5 Examples

In this section, we show how the meeting scheduling problem in Section 3 can be solved by using Boolean CHAL.

In a Boolean CHAL program, we express constraints as Boolean equations and in Boolean equations, we can use the only constraint symbol, = and the function symbols such as /\ (conjunction), \/ (disjunction), -> (implication), <-> (equivalence), ~ (negation) and the constants such as 1 as truth and 0 as falsity and propositional variables.

Firstly, we regard the following terms as propositional variables. $c(x)$ represents that the meeting will be held on day x and $p(x), v(x), m(x)$ represent that the president, the vice president and the manager attend the meeting on day x.

Suppose that we consider a meeting schedule for day 1, 2 and 3 and the following hard constraints represented in Boolean equations exist.

1. The meeting must be held:
   $(c(1)\backslash/c(2)\backslash/c(3)) = 1$.
2. The president must attend the meeting:
   $(c(x)\text{->}p(x)) = 1$ for all $x = 1, 2, 3$.
3. Since $p(x)$ expresses that the president attends the meeting on day x, if it is true, $c(x)$ (the meeting is held on day x) is also true:
   $(p(x)\text{->}c(x)) = 1$ for all $x = 1, 2, 3$.
4. The same thing holds if the vice president or the manager attends the meeting. We can expand these constraints as follows:
   $(v(x)\text{->}c(x)) = 1$ for all $x = 1, 2, 3$ and $(m(x)\text{->}c(x)) = 1$ for all $x = 1, 2, 3$.
5. The president cannot attend the meeting on day 1 and the manager cannot attend the meeting on day 2:
   $p(1) = 0$ and $m(2) = 0$

And we consider the following soft constraints.

1. The vice president should *preferably* attend the meeting. This soft constraint means that $(c(x)\text{->}v(x)) = 1$ should be satisfied as much as possible for all $x = 1, 2, 3$.

2. The manager also should *preferably* attend the meeting. This soft constraint means that $(c(x)\text{->}m(x)) = 1$ should be satisfied as much as possible for all $x = 1, 2, 3$

3. The schedule of the vice president is *prioritized* to the schedule of the manager. This priority means that $(c(x)\text{->}v(x)) = 1$ is stronger than $(c(y)\text{->}m(y)) = 1$ for every $x = 1, 2, 3$ and $y = 1, 2, 3$. To do so, we attach the stronger label to $(c(x)\text{->}v(x)) = 1$ than to $(c(y)\text{->}m(y)) = 1$ for every $x = 1, 2, 3$ and $y = 1, 2, 3$.

Then, a Boolean CHAL program which builds constraint hierarchy of the above example is shown as follows.

```
meeting1 :-
    bool:(c(1)\/c(2)\/c(3))=1,hard([1,2,3]),soft([1,2,3]),
```

```
    bool:p(1)=0,bool:m(2)=0.
meeting2 :- meeting1,bool:v(3)=0.
hard([]).
hard([X|Y]):-
    bool:(c(X)<->p(X))=1,bool:(v(X)->c(X))=1,bool:(m(X)->c(X))=1,
    hard(Y).
soft([]).
soft([X|Y]):-
    chal:soft(bool:(c(X)->v(X))=1,0),chal:soft(bool:(c(X)->m(X))=1,1),
    soft(Y).
```

In the above program,

`chal:soft(bool:(c(X)->v(X))=1,0)` and `chal:soft(bool:(c(X)->m(X))=1,1)` express soft constraints. The first argument is a constraint and the second argument expresses the strength of the constraint. In CHAL, we use a natural number to express the strength. A soft constraint with the number 0 is the strongest and a constraint becomes weaker as the associated number becomes bigger.

If we ask `?-meeting1`, then Boolean CHAL firstly calculates a set of reduced constraints from required constraints and then computes all simplified maximal consistent sets of constraints by solving constraint hierarchy. In this case, Boolean CHAL returns only one maximal consistent set which includes $c(1) = 0, c(2) = 0, c(3) = 1$. Since on day 3, all of three can attend the meeting, day 3 is selected for the most preferable date for the meeting.

The conclusion may be withdrawn by adding another constraint. For example, suppose a new constraint that the vice president cannot attend the meeting on day 3 is added. That is, the following constraint is added:

$c(3) = 0$.

This is done by asking `?-meeting2`, and Boolean CHAL returns only one maximal consistent set which includes $c(1) = 0, c(2) = 1, c(3) = 0$. This means that day 2 is the most preferable meeting date in this new situation because the schedule of the vice president has the priority to the schedule of the manager. This expresses nonmonotonic character of soft constraints.

6 Conclusion

Firstly, we discuss the related work.

1. Borning et al. [3] were the first to propose the HCLP scheme. However, in [3], there is no logical formalization of the most preferable solutions. In this paper, we provide a logical formalization by a variant of prioritized circumscription.

   In [3], they discuss a relation of HCLP to nonmonotonic reasoning and claim that HCLP can handle the multiple extension problems of nonmonotonic logic. However, our result shows that a constraint hierarchy defined by HCLP is no more than a variant of prioritized circumscription. This means that HCLP

can handle only multiple extension problems that can be solved by prioritized circumscription.

2. Baker et al. [1] give a theorem prover of prioritized circumscription. Since they use the finite domain closure axioms, they impose that their considered domain be finite.

   On the other hand, if we use algebraic CHAL, our domain is a complex number. So, semantic restriction does not always impose that the considered domain be finite.

We think that the following future work will be needed.

1. In this paper, we only considered a disjunction of atomic non-required constraints. However, to capture wider class of soft constraints, we must go further so that any form of non-required constraints can be expressed.

2. The operational semantics of CHAL considered in this paper can be regarded as batch-type solving of constraint hierarchy. However, in some cases, it is better to evaluate constraint hierarchy incrementally. For example, if non-required constraint $x = 5$ is found during the execution, then if we can apply this constraint in the earlier time of the execution, constraint solving might become much more easy.

   However, incremental evaluation of constraint hierarchy must support backtracking mechanism so that preferred constraint can be deleted.

Appendix A: An algorithm for solving constraint hierarchy

solve_constraint_hierarchy(CH, RRC)
% Solve constraint hierarchy CH with a set of reduced required constraints RRC.
begin
  $PA := \{(\emptyset, RRC)\}$
  % PA is a set of pairs of (Combined Constraints, Reduced Constraints).
  for every level L in CH from the strongest to the weakest **do**
  begin
    if $L \neq \emptyset$ **then**
    begin
      for every pair $\langle Cs, RC \rangle$ in PA **do**
        $NewPA$:=maximal_constraints(L, Cs, RC, PA)
      $PA := NewPA$
    end
  end
  Take every RC of $\langle Cs, RC \rangle$ in PA to form a set, SC.
  return SC
end (solve_constraint_hierarchy)

maximal_constraints(L, Cs, RC, PA)
% Find all maximal subsets in L which is consistent with RC.
begin
  $QL := \{(Cs, \emptyset, RC, L)\}$, $NGs := \emptyset$.
  do
    QL, NGs, PA :=maximal_constraints1(QL, NGs, PA)
  until $QL = \emptyset$
  return PA
end (maximal_constraints)

maximal_constraints1(QL, NGs, PA)
% Produce all extended consistent sets of constraints from QL.
% QL: a list of quadruple of the following sets of constraints:
% (Combined Constraints, Used Constraints, Reduced Constraints, Rest)
% NGs: a set of contradictory combinations of constraints with RC.
begin
  $NewQL := \emptyset$
  for every element $(Cs, UC, RC, Rest)$ in QL **do**
  begin
    while ($Rest \neq \emptyset$) **do**
    begin
      Take one constraint C from $Rest$ and delete C from $Rest$.
      % Note that $Rest$ is decreased by one element for each while loop
      % so that every combination of constraints is checked only once.

```
        Add C to Cs to get NewCs
        % We extend Cs by adding C.
        if C is a disjunction then
          for every disjunct D in C do
            NewQL, NGs, PA :=
              maximal_constraints2(D, NewCs, UC, RC, Rest, NewQL, NGs, PA)
        else
          NewQL, NGs, PA :=
            maximal_constraints2(C, NewCs, UC, RC, Rest, NewQL, NGs, PA)
      end
    end
    return NewQL and NGs and PA
end (maximal_constraints1)

maximal_constraints2(C, NewCs, UC, RC, Rest, QL, NGs, PA)
% This is the main procedure of calculating maximal consistent sets of constraints.
begin
    Add C to UC to get NewUC.
    if there exists NG ∈ NGs such that NG ⊆ NewUC then
      return QL and NGs and PA
      % If we see that a subset of NewUC is contradictory then
      % we do not invoke solve and no longer extend NewUC.
    NewRC :=solve(C, RC)
    % If C and RC is consistent then
    % solve(C, RC) returns a new set of reduced constraints
    % otherwise it returns inconsistent information.
    if NewRC = inconsistent then
    begin
      Add NewUC to NGs.
      return QL and NGs and PA
      % If we see that NewRC is contradictory then we register it as nogoods
      % and use it for further contradiction detecting and no longer extend NewUC.
    end
    if Rest ≠ ∅ then
      Add ⟨NewCs, NewUC, NewRC, Rest⟩ to QL
    if there exists ⟨Cs', RC'⟩ ∈ PA such that NewCs ⊂ Cs' then
      return QL and NGs and PA
      % If NewCs is a strict subset of another combined constraints in PA
      % then it is not a maximal consistent set.
    Delete any ⟨Cs', RC'⟩ ∈ PA s.t. Cs' ⊂ NewCs.
    % We delete every non-maximal consistent set from PA.
    Add ⟨NewCs, NewRC⟩ to PA.
    return QL and NGs and PA
end (maximal_constraints2)
```

References

[1] Baker, A. B. and Ginsberg, M. L.: *A Theorem Prover for Prioritized Circumscription, Proc. of IJCAI'89*, pp. 463 - 467 (1989).

[2] Buchberger, B.: *Gröbner bases: An Algorithmic Method in Polynomial Ideal Theory*, In N. Bose, ed., *Multidimentional Systems Theory*, pp.184 - 232, D. Reidel, Dordecht (1985).

[3] Borning, A., Maher, M., Martindale, A. and Wilson, M.: *Constraint Hierarchies and Logic Programming, Proc. of ICLP89*, pp.149 - 164 (1989).

[4] Colmerauer, A.: *Opening the Prolog III Universe: A New Generation of Prolog promises some powerful capabilities, BYTE*, pp.177 - 182 (1987).

[5] Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T. and Berthier, F.: *The Constraint Logic Programming Language CHIP, Proc. of the International Conference on Fifth Generation Computer Systems 1988*, pp.693 - 702 (1988).

[6] Descotte, Y. and Latombe, J.: *Making Compromises among Antagonist Constraints in a Planner, Artif. Intell.*, Vol. 27, pp.183 - 217 (1985).

[7] Fikes, R. E.: *REF-ARF: A System for Solving Problems Stated as Procedures, Artif. Intell.*, Vol. 1, pp.27 - 120 (1970).

[8] Fox, M. S., Allen, B. P., Smith, S. F. and Strohm, G. A.: *ISIS: A Constraint-Directed Reasoning Approach to Job Shop Scheduling*, CMU-RI-TR-83-8, Carnegie Mellon University (1983).

[9] Jaffar, J. and Lassez, J-L.: *Constraint Logic Programming, Proc. of the 14th ACM Principles of Programming Languages Conference*, pp.111 - 119 (1987).

[10] McCarthy, J.: *Applications of Circumscription to Formalizing Commonsense Knowledge, Artif. Intell.*, Vol. 28, pp.89 - 116 (1986).

[11] Sakai, K. and Aiba, A.: *CAL: A Theoretical Background of Constraint Logic Programming and its Applications, J. Symbolic Computation*, Vol. 8, pp.589 - 603 (1989).

[12] Satoh, K.: *Formalizing Nonmonotonic Reasoning by Preference Order*, ICOT-TR 440, ICOT, Japan (1988), also to be presented at InfoJapan'90 (1990).

[13] Satoh, K.: *Formalizing Soft Constraints by Interpretation Ordering, Proc. of ECAI90*, pp.585 - 590 (1990).

[14] Smith, S. F., Fox, M. S. and Ow, P. S.: *Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems, AI Magazine*, Vol. 7, pp.45 - 61 (Fall 1986).

[15] Sussman, G. J. and Steel, G. L.: *CONSTRAINTS - A Language for Expressing Almost-Hierarchical Descriptions, Artif. Intell.*, Vol. 14, pp.1 - 39 (1980).