ICOT Technical Report: TR-701

TR-701

Structural Analysis of the Set of Constraints for Constraint Logic Programs

by
Y. Nagai (Toshiba)

October, 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

Structural Analysis of the Set of Constraints for Constraint Logic Programs

Yasuo Nagai
Information Systems Engineering Lab., Toshiba Corp.
70, Yanagi-cho, Saiwai-ku, Kawasaki, 210, Japan
nagai@aioh.ilab.toshiba.co.jp

Ryuzo Hasegawa
Research Center, Institute for New Generation Computer Technology
Mita Kokusai Building, 21F
1-4-28, Mita, Minato-ku, Tokyo, 108, Japan
hasegawa@icot.or.jp

Abstract

The introduction of the constraint concept into logic programming provides constraint logic programming (CLP) with more declarative expressiveness power. However, there are cases when CLP languages cannot be efficiently implemented because of the domains of constraint. In particular, since we use a CAL language, an instance of the CLP scheme, and this algebraic constraint solver is based on the Buchberger algorithm and computes the Gröbner base, it is necessary to process efficiently, by considering the order of constraint solving. Therefore, it is necessary to handle both the inference engine and the constraint solver to efficiently process the CLP. Furthermore, to efficiently process constraints using the CAL language on applications using the Buchberger algorithm, it is necessary to construct as simple a Gröbner base as possible.

This paper describes a program analysis method to improve the efficiency of CLP program execution by introducing this graph-theoretic approach, while considering the active researches of program analysis of logic programming. We consider program analysis methods for the CAL language and apply this method to some examples including non-linear constraints. The results show that the proposed method is effective for problems where the algebraic structure of the set of constraints is sparse.

Topics: Software
Subtopics: Constraint programming
Keywords: Constraint logic programming, constraint, program analysis, CAL, geometric theorem proving, graph-theoretic approach

1 Introduction

The introduction of the constraint concept into logic programming provides constraint logic programming (CLP) with more declarative expressiveness power in terms of: 1) relations that should be satisfied on objects or among their attributes, and 2) a control description about program execution [6] [5]. Most CLP language interpreters consist of three modules: an inference engine, constraint solver, and preprocessor or interface module. The constraint solver solves constraints which cannot be handled by the engine. In other words, it determines whether a set is solvable, and if it is, computes the solutions, given a set of constraints. To obtain solutions, it needs solution methods for the set of constraints, that is solution methods for simultaneous equations.

However, there are cases when CLP languages cannot be efficiently implemented because of the domains of constraint. In this case, it is important to control the execution mechanism of an inference engine and constraint solver. For example, in the inference engine that handles literal resolvent based on SLD-resolution, choice rules of literals and definite clauses unified with a selected literal effect the whole process [7] [8]. On the other hand, the processing of constraints effects order of constraint solving, because a constraint solver solves constraints accumulated by the inference engine. In short, the solvability of the set of constraints depends heavily on the exploration of the search tree during resolution.

In particular, since we use a CAL language [4], an instance of the CLP scheme, and this algebraic constraint solver is based on the Buchberger algorithm and computes the Gröbner base, it is necessary to process efficiently, by considering the order of constraint solving [3] [5] [15] [16]. The construction of a Gröbner base is time-consuming and it is known that the Buchberger algorithm is doubly exponential in worst-case complexity [13] [14]. Therefore, it is necessary to handle both the inference engine and the constraint solver to efficiently process the CLP. Furthermore, to efficiently process constraints using the CAL language on applications using the Buchberger algorithm, such as geometric theorem proving, it is necessary to construct as simple a Gröbner base as possible.

Meanwhile, in solving large sparse linear squares problems $A\ x \simeq b$, several numerical methods that compute the upper triangular matrix of a given A, i.e. the algebraic structure of A, are proposed as an efficient method [22] [10] [12]. These methods introduce a graph-theoretic approach.

This paper describes a program analysis method to improve the efficiency of CLP program execution by introducing this graph-theoretic approach, while considering the active researches of program analysis of logic programming [1] [2], such as functionality analysis, definite analysis, or mode analysis.

In section 2, we show CLP computation model and a search tree generated by SLD-resolution, as preliminaries.

In section 3, we describe the dataflow analysis of constraint logic programs and the collection of sets of constraints based on dataflow analysis [16]. We introduce the dataflow analysis approach and optimization based on information gathering using this analysis. We give the dataflow approach which uses the top-down analysis based on SLD-refutation. During this analysis, given programs and queries, flow information about variable bindings is propagated across each clause for that

predicate and substitutions are kept. Finally the substitution set and the set of constraints are collected, without executing the constraint solver.

In section 4, we describe the structural analysis of sets of constraints, using sparse orthogonal factorization, and its application as an efficient constraint solver [15]. We introduce a graph-theoretic approach to improve the efficiency of constraint processing by analyzing and using an algebraic structure of constraint representation. We describe constraints in terms of the graphical representation and represent the set of constraints as the structure of a matrix with a bipartite graph.

Section 5 describes program execution improvements based on the ordering of goals and the preference of variables, program application, and considerations of program analysis method effectiveness. Furthermore, the overview of the program analysis system under construction, is shown.

2 Preliminaries

In this section, we describe a computation model of CLP and a search tree generated by SLD-resolution.

2.1 Computation Model of CLP

[Def.1] A definite clause is a clause of the form $P \leftarrow P_1, P_2, \ldots, P_n; C_1, \ldots, C_m$, where $P_1, P_2, \ldots, P_n$ is called the literal part of the resolvent and $C_1, \ldots, C_m$ is called the constraint part of the resolvent.
[Def.2] A goal clause is a clause of the form $\leftarrow P_1, P_2, \ldots, P_n; C_1, \ldots, C_m$.
[Def.3] A resolvent is in the form R where $R =< RL; RC >$. RL is called the resolvent of a normal logic program and called the literal resolvent. RC is the resolvent for constraints and takes the canonical form of a set of constraints collected by resolution.
[Def.4] A substitution θ is a finite set $\{v_1/t_1, \cdots, v_n/t_n\}$, where each v_i is a variable, each t_i is a term distinct from v_i, and the variables $v_1, \cdots, v_n$ are distinct. Each element v_i/t_i is called as a binding of v_i.
[Def.5] Let P be a definite clause and let a resolvent R_n, calculated at one time, be $R_n =< L_1.L_2 \ldots L_m; RC >$. Let $P \leftarrow P_1, P_2 \ldots, P_n; C_1, C_2, \ldots, C_l$ be a definite clause of P, when there exists a most general unifier (mgu) θ of $P\theta$ and $L_1\theta$. Then a new resolvent R_{n+1} is derived from a resolvent R_n and P using θ such that the following conditions hold:

- Let $L_k$be an atom selected according to a computation rule $(1 \leq k \leq m)$.
- If $RC' = (RC \cup C_1 \cup C_2 \cup \ldots C_l)\theta$ is solvable, then the resolvent R_{n+1} derives $< (P_1.P_2 \ldots P_n.L_1.L_2 \ldots L_m)\theta; RC' >$ from P and R_n.

[Def.6] Let P be a definite program and G a goal. If successfully derived, then a sequence of this derivation whose last resolvent is $R_n =< RL; RC >$ such that literal part RL is empty $(=\phi)$. The last constraint part of the resolvent element RC of a successful derivation is called an answer constraint.

[Def.7] Let P be a definite program. The success set S_p of P is defined as follows:

$S_p = \{(\leftarrow P; C) \mid \text{goal} \leftarrow P; \phi$ has a successful derivation of SLD-resolution with an answer constraint C. $\}$

The computation model of constraint logic programs can be described using an interpreter as follows: given a CLP program P and a goal (query) G, it starts from an initial goal G and terminates one of two results: an answer constraint $C\theta$ or failure. If a computation succeeds, it outputs an answer constraint $C\theta$ that is an instance of G deduced from P, otherwise it outputs failure. The algorithm of the CLP computation model is given below.

[Algorithm]

Input: A CLP program P and a goal G
Output: Answer constraint $C\theta$, if this was an instance of G deduced from P ;
failure, then failure has occurred.

```
1  begin;
2    Initialize the resolvent R =< RL; RC >, composed of a literal part and constraint part, to be G, the input goal
3    and let R_n be < G; φ >;
4    while the literal resolvent RL is not empty do
5        Select a literal L_1 from the resolvent
6        and a definite clause P ← P_1, P_2, ..., P_n; C_1, C_2, ..., C_l from P
7        such that L_1 and P unify with mgu θ;
8        Remove L_1 from literal resolvents and add P_1, P_2 ..., P_n to the resolvent ;
9        if (RC ∪ C_1 ∪ C_2 ∪ ... C_l)θ is solvable
10         then let this be a new resolvent RC;
11       Apply θ to a new resolvent R_n(=< RL; RC >) and to G;
12   endwhile
13   if the literal resolvent is empty
14     then output an answer constraint Cθ
15     else output failure.
16 end;
```

2.2 Search tree generated by SLD-resolution

[Def.8] Let P be a program and G a goal. A search tree [7] [8] of a goal G with respect to a program P is defined as follows:

- G is a root node of a search tree.
- Nodes of the tree are goals with one goal selected.
- Let the resolvent $< L_1, L_2 \ldots L_m; C >$ be in a node of the search tree and suppose L_1 is the selected atom. Then, for each input clauses $P \leftarrow P_1, P_2, \ldots, P_q; C_1, \ldots, C_r$, the resolvent node has a successor resolvent $< (P_1, P_2 \ldots P_q, L_2 \ldots L_m)\theta; RC >$, where $RC = solve((C \cup C_1 \ldots \cup C_r)\theta)$ such that L_1 and P are unifiable with a mgu θ.
- Nodes that are empty clauses have no successor.

[Def.9] Leaves of the search tree are success nodes where the empty goal has been reached or failure nodes where the selected goal at the node cannot be reduced any more. Success nodes correspond to solutions of the root node of the tree.

3 Collection of the Set of Constraints

We introduce the dataflow analysis approach and the collection of the set of constraints using this analysis [16].

We give the dataflow approach which uses top-down analysis based on SLD-refutation. Given a program and queries, flow information about variable bindings is propagated across each clause for that predicate and substitutions are kept during this analysis. Finally a substitution set and constraint set are collected, without executing the constraint solver.

3.1 Dataflow Analysis

Dataflow analysis can be described as the process of ascertaining and collecting information about the behavior of a computer program prior to program execution. The results are used to guide various code optimizations in the compiler [2].

Here, when given a goal as an input, dataflow analysis ascertains and collects information about the behavior of a program, i.e. the set of constraints, by tracing computation paths on the search tree in a top down manner, based on the computation model of the logic program.

This analysis calculates definition relations and references for data in the program, i.e. dataflow information. This is done by tracing the search tree Tr corresponding to the computation path of the program. Given a program P and a goal G, analysis computes the set of constraints C and its substitution θ of a success path on the search tree, as output. Consequently, the instances of C by θ ($C\theta$) are obtained without executing a constraint solver directly. In other words, analysis does not execute constraint solving ($solve(C \cup \cdots \cup RC)$), but obtains the set of constraints ($C \cup \cdots \cup RC$).

The algorithm of dataflow analysis is given in Figure 1:

[Algorithm]

Input: Program P and goal $G(P \cup \{Q\})$

Output: Set of constraint C, substitution θ of all success paths on the search tree Tr, and all instances of sets of constraints $C\theta$

```
procedure main(Goal, Subst, Constr, Instance)
  begin
    Subst ← {};
    Constr ← {};
    analyze_goal(Goal, Subst, Constr);
    Apply_subst_to_constr(Subst, Constr, Instance);
  end;

procedure analyze_goal(Goal, Subst, Constr)
  begin
    for each clause Cl_i of Goal do;
    analyze_clause(Cl_i, Goal, Subst, Constr);
    endfor
  end;

procedure analyze_clause(Cl, Goal, Subst, Constr)
  begin
    let Cl be of the form Head :- Body;
    if unify(Head, Goal, Subst1);
      then Subst ← Subst ∪ Subst1;
```

```
            analyze_body(Body, Subst, Constr);
        else failed_unification(Head, Goal, Subst) ;
    end;

procedure analyze_body(Body, Subst, Constr)
  begin
    let Body be of the form Literal_Body;Constraint_Body ;
    if Literal_Body is empty
      then return Constr ← Constr ∪ Constraint_Body;
      else let Literal_Body be of the form p(X), LBody_Tail;
    generate_goal(p, Subst, Cp);
    analyze_goal(Cp, Subst, Constr1);
    analyze_body(LBody_Tail, Subst, Constr2);
    Constr ← Constr ∪ Constr1 ∪ Constr2;
  end;
```

Figure 1: Algorithm of dataflow analysis

`main/4` is a top-level of the dataflow algorithm. In this algorithm, `analyze_goal/3` propagates information about the variable bindings of all executable predicate calls across each corresponding clause, and it ascertains and collects the static flow information of each predicate. `analyze_clause/4` unifies a clause and a goal, propagates a substitution across the body part of the clause, and calls `analyze_body/3` using information about the variable sharing between the subgoal and the head.

3.2 Collection of the Set of Constraints

Example 1 is given as a typical instance of a CAL program including non-linear constraints. `example1` is a goal G and program P contains definitions of nine predicate symbols, `example1`, `f1`, `f2`, `t3`, `f4`, `f5`, `f6`, `f7`, and `f8`.

[Example]

```
?-   example1.

example1 :- f1(x2,x5), f2(x2,x5,x7), f3(x3,x6,x7),
            f4(x1,x7), f5(x3,x5,x8), f6(x1,x4,x7),
            f7(x6,x8), f8(x1,x4).
f1(X,Y) :- 2*X^2 + Y =2.
f2(X,Y,Z) :- X * Y^2 +Z =1.
f3(X,Y,Z) :- X + Y + 2*Z^3 =4.
f4(X,Y) :- X + Y =3.
f5(X,Y,Z) :- X^2 + 2*Y + Z =2.
f6(X,Y,Z) :- X^3 + Y + 3*Z =1.
f7(X,Y) :- X + Y^3 =2.
f8(X,Y) :- X * Y + X =4.
```

In Example 1, the collection process of the set of constraints reduces subgoals `f1(x2,x5)`, `f2(x2,x5,x7)`, `f3(x3,x6,x7)`, `f4(x1,x7)`, `f5(x3,x5,x8)`, `f6(x1,x4,x7)`, `f7(x6,x8)`, `f8(x1,x4)`, increments constraints and substitution, and finally obtains the set of constraints C and the set of substitution $\{\theta_1$(theta1), θ_2(theta2), ..., θ_8(theta8)$\}$. Consequently, the instance $C\theta = \{$ `2*`$x2^2$ `+ x5 =2, x2 *` $x5^2$ `+x7 =1, x3 + x6 + 2*`$x7^3$ `= 4, x1 + x7 =3,` $x3^2$ `+ 2*x5 + x8 = 1,` $x1^3$ `+ x4 + 3*x7=1, x6 +` $x8^3$ `= 2, x1*x4 + x1 =4` $\}$ of the set of constraint C by the substitution θ(theta) $= \theta_1 \circ \theta_2 \circ \cdots \circ \theta_8$ can be obtained, without executing the constraint solver,

using dataflow analysis. Figure 2 shows a collection process of a set of constraints based on dataflow analysis (Example 1)

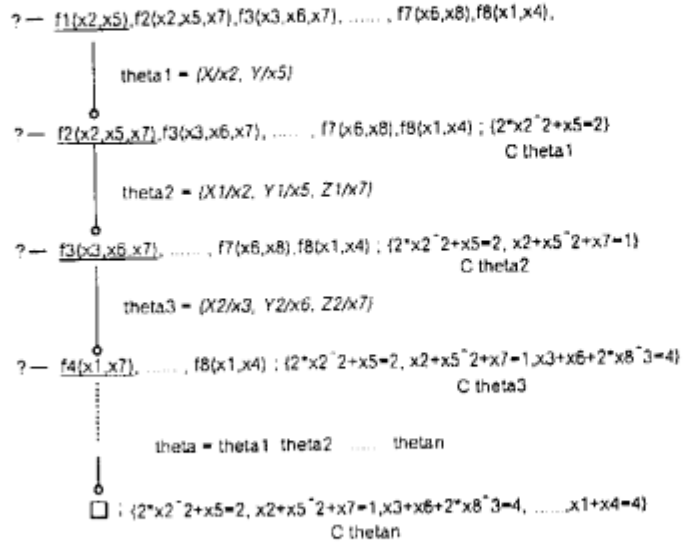

Figure 2: Collection of the constraint set based on dataflow analysis (Example 1)

4 Structural Analysis of the Collected Set of Constraints

We describe a structural decomposition method of the set of constraints [15]. In this method, the set of constraints can be represented in terms of a constraint graph i.e. a bipartite graph form. DM decomposition [10] is also performed, so that a block upper triangular matrix can be correctly computed by canonical reordering of a matrix which represents the constraint graph.

Next, we consider improving the efficiency of the constraint solver, through the structural analysis of the constraint graph, which decomposes this graph into subgraphs and checks their structural solvability.

4.1 Constraint and Constraint Graph

A constraint is defined as the relationship between a set of objects. Each constraint and each set of constraints are modeled from the viewpoint of graph-theoretic notation. Here, we introduce a bipartite graph representation in order to model the constraint graph, because a bipartite graph can usefully portray the structure of a set of constraint [15] [12].

A bipartite graph [9] is an undirected graph $G = (V; E)$ or $G = (V^+, V^-; E)$, in which V can be partitioned into two sets V^+ and V^- ($V = V^+ \cup V^-$) such that $(u, v) \in E$ implies either $u \in V^+$ and $v \in V^-$ or $u \in V^-$ and $v \in V^+$. That is, all edges go between the two sets V^+ and V^-.

The algebraic structure of a set of constraints, such as `f1(x2, x5) =0`, `f2(x2, x5, x7) =0`, ..., `f8(x1, x4) =0` are modeled with a bipartite graph $G = (V^+, V^-; E)$ (Figure 3). Constraints (a set of constraints) form the V^+ nodes while their variables (parameters) form the V^- nodes. For

example, a constraint `f2(x2, x5, x7) =0` forms `f2` in the V^+ nodes and `x2, x5, x7` in the V^- nodes.

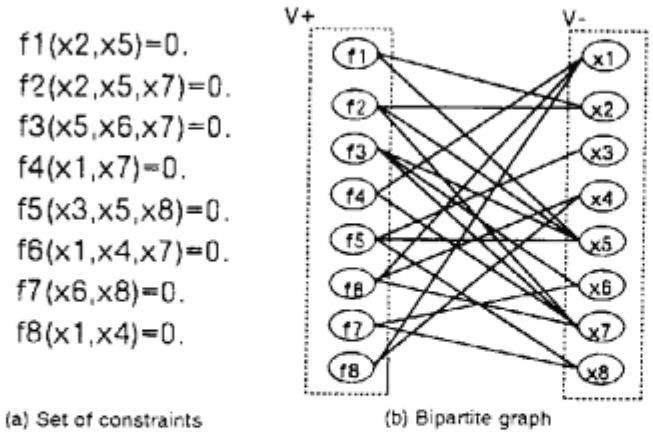

Figure 3: Bipartite graph of the set of constriants in Example1

4.2 Structural Decomposition of Constraint Graph

In order to find the algebraic structure of the constraint graph, we introduce the constraint graph structural decomposition method, DM (Dulmage and Mendelsohn) decomposition [10].

4.2.1 DM Decomposition

DM decomposition of a bipartite graph $G = (V^+, V^-; E)$ determines a finite cover. A finite cover is a pair of subsets U and V of V^+ and V^- respectively such that every edge (u, v) of E is either $u \in U$ or $v \in V$. Decomposition determines irreducible subgraphs $\{G_i\}_{i=1}^{n}$ that only have a minimum cover and, at most, two tails $\{G_-, G_+\}$. It has no tails if and only if G is a bipartite graph $G = (V^+, V^-; E)$, where $| V^+ | = | V^- | = n$, and has a transversal of order n.

Consequently, this method corresponds to finding a canonical reordering of a matrix which represents the constraint graph and correctly computing a block upper triangular matrix. Canonical reordering permutes the columns and rows of the matrix that leave square submatrices on the diagonal, and a rectangular submatrix in the lower right-hand corner. This is used to solve each square in turn, each square corresponds to a set of constraints.

An overview of the DM decomposition algorithm is shown in Figure 4.

```
1 procedure DM_decomposition(G:input, G':output, G'':output)
2   begin
3     find_maximum_matching(G, M);
4     make_auxiliary_graph(G, M, G_M);
5     induce_auxiliary_graph_and_handle_two_tails(G_M, G');
6     find_strongly_connected_component(G', G'');
7     permute_subgraph_merge(G'');
8   end;
```

Figure 4: Overview of DM decomposition algorithm

In the above algorithm, `find_maximum_matching/2` finds a maximum matching M in a bipartite graph $G = (V^+, V^-; E)$[9]. `make_auxiliary_graph/3` constructs an auxiliary graph $G_M = (V^+, V^-; \hat{E})$ such that $\hat{E} = E \cup \{(u, v) \mid (v, u) \in M\}$.

`induce_auxiliary_graph_and_handle_two_tails/2` determines the tails $\{G_-, G_+\}$. These tails are the subgraphs induced by $U_{(-)}$ and $U_{(+)}$, where $U_{(-)}$ is a set of the end nodes in G_M having start nodes in $V^+ - \partial^+ M$, and $U_{(+)}$ is a set of start nodes in G_M having end nodes in $V^- - \partial^- M$. `find_strongly_connected_component/2` finds the strongly connected components [11] $G_i = (W_i^+, W_i^-; E_i)$, $(i = 1, 2, \ldots, n)$ of $G_M - (U_{(-)} \cup U_{(+)})$. `permute_subgraph_merge/1` sorts $\{G_-, G_+\} \cup G'$ in topological order, where $G' = \{G_i\}_{i=1}^n$. Figure 5 shows the result of the DM decomposition in Example 1. In a subgraph $G_1 = (W_1^+, W_1^-; E_1)$, a set W_1^+ represents elements `f3`, `f5`, `f7`, and W_1^- represents elements `x3`, `x5`, `x8`. In other words, a subgraph G_1 corresponds to a set of constraints, `f1(x2,x5)=0`, `f5(x3,x6)=0`, `f7(x6,x8)=0`.

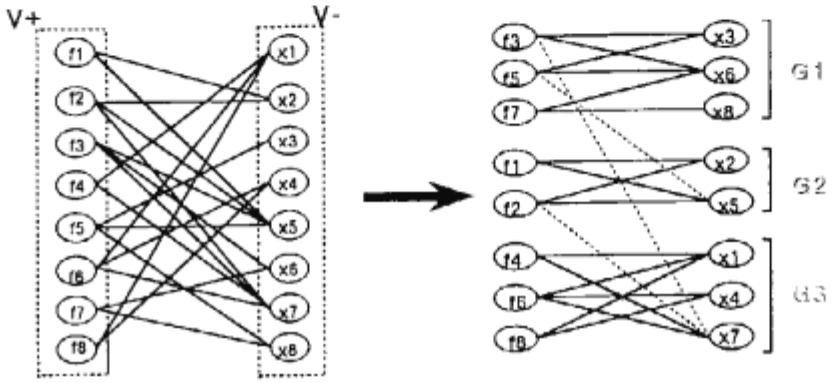

Figure 5. Bipartite graph of constraint graph and result of DM decomposition

4.2.2 Structural Analysis of the Set of Constraints

Structural analysis of the set of constraints decomposes the constraint graph into subgraphs and checks their structural solvability to improve the efficiency of the CLP constraint solver. Check of their structural solvability is effective to a real domain, where constraints are represented in terms of a linear equation, and based on the following theorem [10].

[Theorem]

In DM decomposition $G_i = (W_i^+, W_i^-; E_i)$ $(i = \{1, \ldots, n\} \cup \{-, +\})$ of bipartite graph $G = (V^+, V^-; E)$, the following items (a) to (c) hold;

(a) If $W_+^- \neq \phi$, then $|W_+^+| < |W_+^-|$. (G_+)

(b) If $W_-^+ \neq \phi$, then $|W_-^+| > |W_-^-|$. (G_-)

(c) If $|W_i^+| = |W_i^-|$ and G_i $(i = 1, 2, \ldots, n)$, then a perfect matching exists.

In this theorem, structural solvability is defined as the existence of perfect matching in a bipartite graph [15].

(a) shows that bipartite graph G is not structurally solvable and can be locally under-constrained. The number of constraint variables belonging to W_+^+ is larger than one of the constraints and there are infinite solutions.

(b) shows that bipartite graph G is not structurally solvable and can be locally over-constrained. The number of variables is fewer than one of the constraints and there is no solution.

(c) shows that bipartite graph G is structurally solvable, because each subgraph $G_i (i = 1, \ldots, n)$ has perfect matching.

According to this theorem, we can determine the degree of freedom in the system corresponding to the set of constraints and select the variables of constraints in executing the constraint solving on a real domain such that a under-constrained or over-constrained state can be handled.

However, the above structural solvability check method is not very effective to one of an algebraic constraint solver of a CAL language, because this algebraic constraint solver that is based on a Buchberger's Gröbner base computation algorithm takes a complex domain [4].

Here, since using a CAL language as CLP, we focus on not structural solvability, but dependency information determined in the structural analysis of the set of constraints. This dependency information between constraints can be determined by reordering $\mathcal{G} = \{G_i\}$ in a partial order $\prec$ and by representing a directed graph computed from $W_-^- = W_+^+ = \phi$ and $\mathcal{G}$ as a block upper triangular matrix form. As a result, it is expected that efficient constraint solving can be realized by solving each block corresponding to the subgraphs (W_n^+, W_n^-),(W_{n-1}^+, W_{n-1}^-), ..., (W_2^+, W_2^-), (W_1^+, W_1^-), block by block. This dependency information is utilized to optimize a source code of CLP languages for improvement of program execution.

5 Program Execution Improvement based on the Goal Ordering and the Variable Preference

We describe an improvement of CAL problem execution based on the ordering of goals and the preference of variables. An experimental result concerning the execution time and the number of critical pairs generated during program execution is shown on the geometric theorem proving problems. Finally, we consider main factors in obtaining good results with the efficiency method.

5.1 Determination of the Ordering of Goals and the Preference of Variables

The ordering of the goals and the preference of variables are determined using the dependency information obtained through the structural analysis of the set of constraints [1]. The source-level optimization of a CAL program, is performed based on this dependency information, such that they can be used as control information for the constraint solver. As a result, it is believed that this optimization avoids wasteful computation in constraint solving and improves program execution,

[1] In practice, the goal ordering and the variable preference are determined according to the number of elements of decomposed sets of constraints, the degree of the variables of elements, and the coefficient of constraints. In other words, a set of constraints including as simple constraints as possible from the above viewpoints (number of elements, degree of variables and coefficient of elements of constraints) is selected and ordered in advance.

when the optimized program is executed. This is why our optimization approach reduces redundant computation in critical pair generation when solving algebraic constraints, that constructs a Gröbner base as an answer constraint, and suppresses the growth of a coefficient to the solution.

Next, we explain the optimization result using example1.

The goals are ordered according to a sequence G_3, G_2, and G_1 that preserves the dependency information determined through DM decomposition, such that each set of constraints corresponding to the decomposed subproblem can be solved (Figure 6).

```
example1 :-
    f4(x1,x7),
    f6(x1,x4,x7),      } G3
    f8(x1,x4),
    f1(x2,x5),
    f2(x2,x5,x7),      } G2
    f3(x3,x6,x7),
    f5(X3,X5,X8),      } G1
    f7(x6,x8).
```

Figure 6: Execution ordering determined by changing the ordering of goals

The preference of the variables is determined using a pre predicate of CAL language, such that the preference of subproblem $G_3 \ll$ the preference of subproblem $G_2 \ll$ the preference of subproblem G_1 shown in Figure 7. The Gröbner base can be calculated efficiently because we can control a term rewriting operation, that is central processing of the algebraic constraint solving, by determining the preference using the pre predicate.

```
pre(x3,100).
pre(x6,100).
pre(x8,100).
pre(x2,99).
pre(x5,99).
pre(x1,98).
pre(x4,98).
pre(x7,98).
```

Figure 7: Ordering of the variables using predicate *pre/2*

Moreover, the program analysis system is under construction and consists of the following three processing components shown in Figure 8: a dataflow component, a structural analysis component, and a source-level optimization component. This system reads a CAL program to be analyzed (shown in Example 1) as an input and generates an optimized CAL program (shown in Figures 6 and 7) as an output.

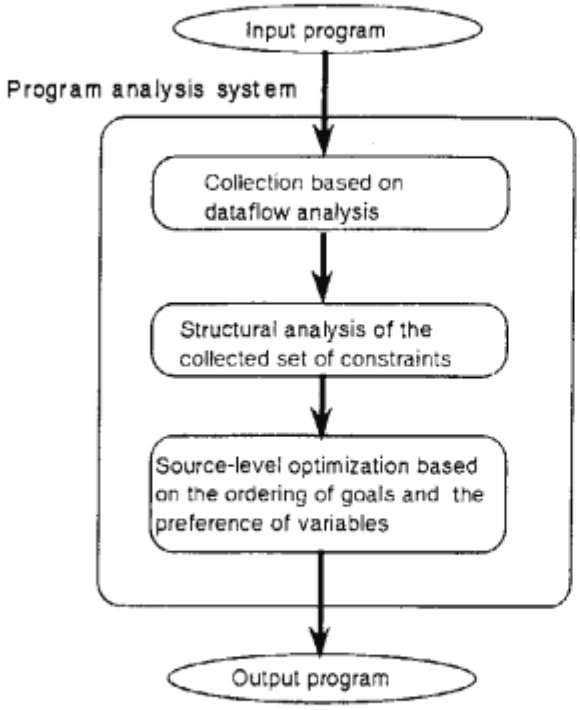

Figure 8: Overview of program analysis system for CAL

5.2 Applications and Experimental Results

In geometric theorem proving problems, when a geometric theorem is given, a geometric configuration and the conclusion are translated into algebraic formulas. This problem is considered as a problem of ideal membership whose answer determines whether the theorem holds. Though there are several approaches to geometric theorem proving [13] [21], the Gröbner base approach is focused on as an application and experiment with the efficiency method. In other words, we choose a geometric theorem proving problem as a typical example using Gröbner base, i.e. an application problem to an algebraic constraint solver in the CAL.

We apply the program analysis method to the geometric theorem proving problems, where the problems are described in terms of non-linear algebraic constraints, and evaluate this method from the viewpoint of the time of a program execution (Table 1). This experiment is carried out using the CAL language on a personal sequential inference machine (PSI-II [18]).

Example 2 is a geometric theorem proving problem of a triangle orthocenter described in terms of 5 hypotheses and 8 variables [21] . Example 3 is a geometric theorem proving problem of a ninepoint circle described in terms of 9 hypotheses and 12 variables [21].

In consequence, our efficiency method is effective to these examples, whose algebraic structure in terms of a graphical representation of the constraint set is sparse.

Problem	Pre-application	Post-application
Example 1 (8 constraints, 8 variables)	1731	121
Example 2 (Triangle orthocenter : 5 hypotheses, 8 variables)	7565	2371
Example 3 (Ninepointcircle : 9 hypotheses,12 variables)	3524098	12753

Table 1 : Result of an application of the efficiency method to examples including non-linear constraints (msec)

Table 2 shows the number of critical pairs generated in the Gröbner base construction of examples shown in Table 1.

Problem	Pre-application	Post-application
Example 1 (8 constraints, 8 variables)	12	4
Example 2 (Triangle orthocenter : 5 hypotheses, 8 variables)	85	75
Example 3 (Ninepointcircle : 9 hypotheses,12 variables)	3180	154

Table 2 : The number of critical pairs generated by an application to Examples

5.3 Considerations of Program Analysis Method Effectiveness

An algebraic constraint solver in CAL language [4] is implemented using the Buchberger algorithm [14] that determines the Gröbner base for testing the satisfiability of the set of non-linear constraints in the form of polynomial equations. The construction of a Gröbner base is a time-consuming process. It turns out that the worst case complexity of the Buchberger algorithm is doubly exponential [13] [14].

The following points should be considered as essential improving the performance of Gröbner base computation, which is central to the algebraic constraint solver in the CAL [13] [14]:

- The choice of an order for variables affects the performance of Gröbner base computation. For example, execution time differs drastically between basis computation using various ordering methods.
- The order in which critical pairs are generated in Gröbner base computation and the number of generated critical pairs considerably affect performance.
- Much of the computation time depends on the size of the generated rational numbers of the base. In other words, a coefficient can grow and influence execution time significantly, because the Gröbner base algorithm is implemented using rational arithmetic.

To improve the performance of the Gröbner base algorithm, our efficiency method decomposes the set of constraints into independent and dependent sets and extracts the algebraic syntactic structure of the set of constraints. The extracted algebraic structure is represented in the form of dependency information on the subset of constraints and on variables in the constraints. The structure is used to provide heuristics for control of algebraic constraint solvers.

In our method, the reduction of the number of generated critical pairs and the construction of the Gröbner base in as simple a form as possible are regarded as important for improving the performance of base computation (program execution time).

In the above section, an experiment on the execution time of the program and the number of generated critical pairs by application of an efficiency method is conducted to investigate the relationship between the improvement in program execution time and the reduction in the number of generated critical pairs and to clarify the above consideration.

The experimental results in Table 1 and Table 2 show that an application of our efficiency method to the program analysis of the CAL programs reduces the number of generated critical pairs, controls the growth of the coefficient, and leads to efficient algebraic constraint solving based on Buchberger algorithm.

6 Related Works and Conclusion

Gabbrielli *et al.* [19] proposes algebraic semantics for the success set $SS_3(P, \mathcal{R})$ in order to characterize the operational behavior of programs. They have defined a notion of constrained interpretation and models and given operational semantics. $SS_3(P, \mathcal{R})$ represents the computed answer constraint and is used for the abstract interpretation of CLP programs, that perform the abstraction of an algebraic operation on the particular structure $\mathcal{R}$ on which the computation is performed. A suitable notion of the $\mathcal{R}$-abstract solution is needed to perform the abstraction process. Our structural analysis of the set of constraints using a graph-theoretic approach will be applied to an abstraction of the set of constraints and an analysis of the $\mathcal{R}$-abstract solution using a success set $SS_3(P, \mathcal{R})$.

Serrano [20] presents a constraint-based environment for Mechanical Computer Aided Design, where a graph-theoretic approach is applied to constraint management. In this approach, the set of constraints are represented as a directed graph i.e. a bipartite graph, where nodes indicate variables and edges indicate relationships on constraint variables. Then, dependencies on the variables of constraints are generated, and the evaluation of the set of constraints and over-constrained or under-constrained systems in terms of the set of constraints are detected. The dependency analysis of the constrained systems of constraints is similar to our structural analysis of the constraint set in that both approaches use a graph-theoretic approach using a bipartite graph for representation of the set of constraints.

Kapur [13] describes a refutational approach to geometric theorem proving using the Gröbner base algorithm and experimentally observes that the pseudodivision and triangulation in Wu's method are strongly related to the the critical pair computation among polynomials which is central to the Gröbner base method. Critical pair computation is significantly reduced by obtaining a triangular form of Wu's method, because critical pairs are computed only among certain subset of polynomials after triangulation. We feel that our structural analysis of the constraint set, that decomposes the constraint graph (constraint set) into subgraphs (subsets of constraints), is similar to computation of triangular form of Wu's method from the point of reduction of the number of generated critical pairs. Therefore, we expect that our method is utilized as an efficiency one for geometric theorem proving using the Gröbner base algorithm

We have presented a program analysis method that focuses on the algebraic structure of the set of constraints in order to realize a program analysis system for CLP languages. Especially, we have

given a brief explanation of an application of this method to an algebraic constraint solver, based on the Buchberger algorithm, of the CLP language, CAL. Through an experiment using examples where the algebraic structure of the set of constraints is sparse, we have shown that our method is effective in the following points:

- The execution time of CAL programs is improved by application of our program analysis method.
- Our method reduces the number of generated critical pairs in the Buchberger algorithm that calculates the Gröbner base, and therefore the computation time for the Gröbner base construction is reduced.

We are now implementing a program analysis system based on structural analysis of the set of constraints, using the ESP language on the PSI-II machine. It seems we have little time to estimate our efficiency method for program analysis, because we have not yet completed an estimation for the implemented system. Therefore, we need to estimate our method by considering the analysis time of CAL programs to get a final result.

Acknowledgments

We would like to express thanks to Dr. Katsumi Nitta, Chief of the Seventh Research Laboratory, Dr. Akira Aiba, Subchief of the Fourth Research Laboratory, Dr. Koichi Furukawa, Deputy Director of the ICOT Research Laboratories, and the members of CLP Working Group (ICOT) for helpful comments and suggestions. Furthermore, we would like to express special thanks to Dr. Kazuhiro Fuchi, Director of ICOT Research Center, who has given us the opportunity to carry out research in the Fifth Generation Computer Systems Project.

References

[1] S.K. Debray and D.S. Warren, Automatic Mode Inference For Logic Programs, *J. Logic Programming 1988:5*, 1988

[2] M.S. Hecht, *Flow Analysis of Computer Programs*, North-Holland, 1977

[3] K. Marriott, H. Sondergaard, Analysis of Constraint Logic Programs, In *Proc. of NACLP '90*, 1990

[4] K. Sakai and A. Aiba, CAL: A Theoretical Background of Constraint Logic Programming and its Application, *J. Symbolic Computation 8*, 1989

[5] J. Cohen, Constraint Logic Programming Languages, *Comm. ACM*, Vol.33, No.7, 1990

[6] M. Dincbas, Constraint, Logic Programming and Deductive Database, In *Proc. of France-Japan Artificial Intelligence and Computer Science Symposium 86*, 1986

[7] J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 2nd Edition, 1987

[8] L. Sterling and E. Shapiro, *The Art of Prolog*, MIT Press, 1986

[9] J.E. Hopcroft and R.M. Karp, An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs, *SIAM J. Comput., Vol.2, No.4*, 1973

[10] A.L. Dulmage and N.S. Mendelsohn, Two Algorithms for Bipartite Graphs, *J. SIAM, Vol.11, No.1*, March, 1963

[11] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974

[12] E.J. Henry and R.A. Williams, *Graph Theory in Modern Engineering, Computer Aided Design, Control, Optimization, Reliability Analysis*, Academic Press, 1973

[13] K. Kapur and J.L. Mundy, Special Volume on Geometric Reasoning, *Artificial Intelligence Vol. 37, No.1-3*, December, 1988

[14] C.M. Hoffmann, Gröbner Bases Techniques, in Chapter 7, *Geometric & Solid Modeling. An Introduction*, Morgan Kaufmann Publishers, Inc., 1989

[15] Y. Nagai, Structural Analysis of Constraint Graph by means of Sparse Orthogonal Factorization and Its Application for Efficient Algebraic Constraint Solver (in Japanese), *Technical Report of Japanese Society for Artificial Intelligence*, SIG-F/H/K-9001 12, 1990

[16] Y. Nagai and R. Hasegawa, Dataflow Analysis of Constraint Logic Programs and Its Application to Source-Level Optimization for Program Improvement (in Japanese), In *Procs. of the 5th Annual Conf. of JSAI*, 1991

[17] T. Chikayama, Unique Features of ESP, In *Procs. of Int. Conf. FGCS 1984*, 1984

[18] H. Nakashima and K. Nakajima, Hardware architecture of the sequential inference machine: PSI-II, In *Procs. of 1987 Symposium on Logic Programming*, 1987

[19] M. Gabbrielli and G. Levi, Modeling answer constraints in Constraint Logic Programs, In *Proc. of ICLP 91*, 1991

[20] D. Serrano, Constraint Management in Conceptual Design, PhD. dissertation, MIT, 1987

[21] B. Kutzler, Algebraic Approaches to Automated Geometry Theorem Proving, PhD Thesis, RISC, Johannes Kepler University, 1988

[22] T.F. Coleman *et al.*, Predicting Fill for Sparse Orthogonal Factorization, *J. ACM*, Vol.33, No.3, 1986