ICOT Technical Memorandum: TM-0422

TM-0422

Introduction to CAL(Extended Abstract)

by
K. Sakai & A. Aiba

December, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Introduction to CAL (Extended Abstract)

KÔ SAKAI and AKIRA AIBA
Institute for New Generation Computer Technology
21F, Mita Kokusai Building,
4-18, Mita 1-Chome, Minato-ku, Tokyo 108, Japan

December 4, 1987

Abstract

Constraint logic programming (CLP) is an extension of logic programming by introducing the facility of writing and solving constraints in a certain domain. CAL (*Contrante avec Logique*) is a CLP language in which (possibly non-linear) polynomial equations can be written as constraints, while almost all the other CLP languages proposed so far accept only linear equations. The Buchberger algorithm to obtain Gröbner bases has been adopted as the constraint solver in the CAL interpreter.

1 Introduction

A number of extensions of logic programming languages have been proposed. One major trend is to try to introduce specific computational power in certain domains such as algebraic treatment of mathematical formulae. This kind of extension can be summarized by a common keyword "constraint".

With respect to constraint logic programming (CLP), which is an amalgamation of constraint and logic programming, there is an important language scheme called CLP(X), proposed by Jaffar and Lassez [6]. A system named CLP(R) was implemented in Monash University as its instance [4]. CLP(R) handles linear equations and inequations as constraints. There is another important CLP language with algebraic constraints: Prolog III of Colmerauer [2]. In Prolog III, the user can write not only linear constraints on rational numbers, but also Boolean and other constraints.

This paper describes a CLP language named CAL (*Contrante avec Logique*) and its future extensions. Since the current CAL interpreter employs the Buchberger algorithm to compute Gröbner bases of equations as its constraint solver, it can handle non-linear polynomial equations as well. This makes CAL very powerful and flexible.

At present, CAL can run in DEC 10 Prolog on DEC2060. The system is being transferred into ESP on the PSI machine now. We also have a modified version of CAL that can solve Boolean equations. This Boolean CAL has an algorithm to compute Boolean Gröbner bases of Boolean equations as its constraint solver instead of the original Buchberger algorithm.

As a future extension, we will introduce types into constraints to select the solver corresponding to the type of constraints automatically. Moreover, we will prepare a facility to incorporate user-defined constraint solvers.

2 Constraints in CAL

The meaning of the word "constraint" depends largely on the context. In CAL, it means an arbitrary algebraic equation in the form of a polynomial. The CAL constraint solver solves these constraints incrementally, that is, constraints in CAL are active according to the terminology of Dincbas [3]. If constraints contradict each other, the contradiction is detected as early as possible. This saves a lot of computing time if there is a failure. Moreover, even if information obtained from the given constraints is insufficient to determine the values of all the unknowns, relations between the unknowns can be obtained. Although constraints in the form of inequations are not allowed in the current system, they can be added easily by being treated as passive constraints [3].

In CAL, there are two kinds of terms; Prolog terms and the terms occurring in constraints (polynomials). The equations between Prolog terms are handled by unification as is usual in Prolog, and those between polynomials are treated only by the constraint solver.

3 Buchberger algorithm and Gröbner bases

Buchberger [1] introduced the notion of Gröbner bases and showed an algorithm to compute a Gröbner base of a given system of polynomial equations. His algorithm has found many applications in computer algebra over the last few years. The CAL interpreter regards a Gröbner base of a system of polynomial equations (constraints) as its solution. This section summarizes the theoretical background of Gröbner bases and the Buchberger algorithm.

We can assume that a polynomial equation has the form $p = 0$ without loss of generality. Let $E = \{p_1 = 0, \ldots, p_n = 0\}$ be a system of polynomial equations. Let I be the ideal generated by $\{p_1, \ldots, p_n\}$ in the ring of all polynomials. There is a close relation between elements of I and solutions of E, which is proved easily by the Hilbert zero point theorem [5].

Theorem 1 *Let p be a polynomial. Every solution of E is a solution of $p = 0$ if and only if some power, p^n, is in I.*

Moreover, the following corollary of the above theorem is very important to detect inconsistency of constraints.

Corollary 1 *E does not have any solution if and only if $1 \in I$.*

Now the problem of solving constraints is reduced to the membership problem of the generated ideal. Buchberger gave the following algorithm to determine whether a polynomial is a member of the ideal.

Suppose that there is a system of algebraic polynomial equations (constraints). Each equation can be viewed as a rewrite rule which rewrites the maximum monomial to the rest of the polynomial under a certain ordering between monomials. For example, under the lexicographic ordering between monomials, the polynomial equation $a + x - z = b$ can be viewed as the rewrite rule $z \rightarrow x - b + a$. When the left hand sides of two arbitrary rewrite rules are not mutually prime, these two rules are said to overlap. In such a case, the least common multiple (LCM) of their left hand sides can be rewritten in two ways, but the result, called a critical pair, might not converge by further rewriting. Namely, their irreducible forms may be different. In this case, this pair is added to the system of equations. Repeating this process eventually yields a confluent rewriting system. This confluent system is called Gröbner bases of the original system of equations. The following theorem [1] states the relation 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 I be the ideal generated by $\{p_1, \ldots, p_n\}$. Then a polynomial, p, is in I if and only if p is reduced to 0 by R.*

The following is a rigorous definition of the Buchberger algorithm. Let E be a set of equations and R be a set of rewrite rules. By the algorithm, the Gröbner base of E is obtained in R in the form of rewrite rules.

1. Set $R \leftarrow \phi$.
2. For each equation $l = r$ in E, simplify $l - r$ by rewrite rules in R and ordinary algebraic and arithmetic simplifications. Let e be the result. If $e \equiv 0$, then delete the equation $l = r$ form E. Otherwise, replace $l = r$ in E with the equation $e = 0$.
3. If $E = \phi$, then terminate.
4. Select an equation $e = 0$ from E.
5. Let l' be the maximum monomial in e under a certain ordering. Then solve $e = 0$ with respect to l', and obtain the equation $l' = r'$.
6. Add a rule $l' \rightarrow r'$ into R.
7. Create all the critical pairs among rules in R and add them as equations into E.
8. Goto 2.

4 Execution of a CAL program

The execution of a CAL program is illustrated by an example. As explained in the previous section, a feature of CAL can be seen when constraints are non-linear. The following is an example of proving a geometrical theorem; the four midpoints of the edges of a quadrangle form a parallelogram. The program is as follows:

```
mid(AX,AY,BX,BY,CX,CY) :-
    AX + CX = 2 * BX,
    AY + CY = 2 * BY.
para(AX,AY,BX,BY,CX,CY,DX,DY) :-
    (AX - BX) * (CY - DY) == (AY - BY) * (CX - DX).
```

The above clauses state the conditions for midpoint and parallel. The clause "`mid`" states that the point `(BX,BY)` is a midpoint of the segment `(AX,AY)-(CX,CY)`. The clause "`para`" checks whether the segment `(AX,AY)-(BX,BY)` and the segment `(CX,CY)-(DX,DY)` are parallel. The symbol `==` in the body of this clause denotes the predicate which checks the equality of its right and left hand sides under the current collection of constraints. This is obtained by transforming the following equation:

$$\frac{AY - BY}{AX - BX} == \frac{CY - DY}{CX - DX}$$

representing the equality of the tangents of the two segments.

To prove the above problem by this program, the following goal sequence should be evaluated:

```
?- mid(0,0,x4,y4,x1,y1),
   mid(x1,y1,x5,y5,x2,y2),
   mid(x2,y2,x6,y6,x3,0),
   mid(x3,0,x7,0,0,0),
   para(x4,y4,x5,y5,x7,0,x6,y6),
   para(x4,y4,x7,0,x5,y5,x6,y6).
```

Refer to the following figure for the coordinates.

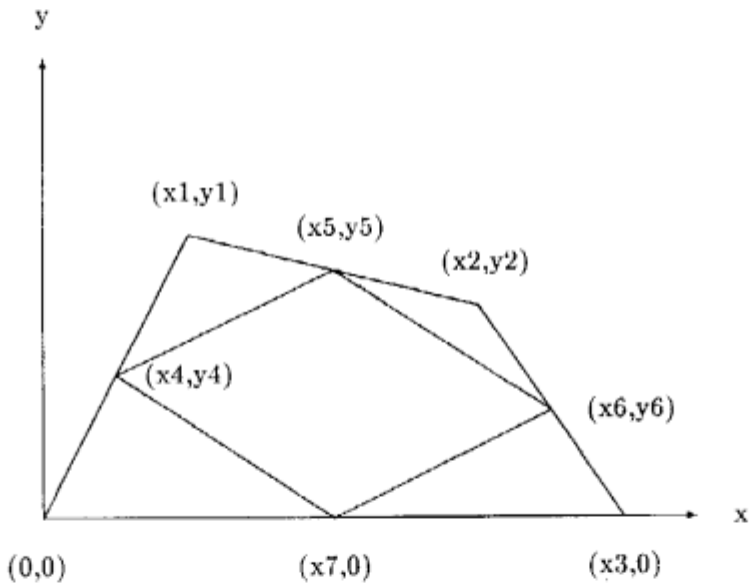

Figure 1 Coordinates for a Geometric Problem

The above goals are evaluated as follows:

1. A goal `mid(0,0,x4,y4,x1,y1)` is unified to the head of a clause for `mid`. Then the constraints `0+x1=2*x4`, and `0+y1=2*y4` are obtained.
2. A goal `mid(x1,y1,x5,y5,x2,y2)` is unified to the head of a clause for `mid`. Then the constraints `x1+x2=2*x5`, and `y1+y2=2*y5` are obtained.
3. A goal `mid(x2,y2,x6,y6,x3,0)` is unified to the head of a clause for `mid`. Then the constraints `x2+x3=2*x6`, and `y2+0=2*y6` are obtained.
4. A goal `mid(x3,0,x7,0,0,0)` is unified to the head of a clause for `mid`. Then the constraints `x3+0=2*x7`, and `0+0=2*0` are obtained.

5. At this moment, the obtaining constraints are as follows. Note that they are simplified.
   ```
   x1=2*x4
   y1=2*y4
   x1+x2=2*x5
   y1+y2=2*y5
   x2+x3=2*x6
   y2=2*y6
   x3=2*x7
   ```

6. A goal `para(x4,y4,x5,y5,x7,0,x6,y6)` is unified to the head of a clause for **para**. Then the equation `(x4-x5)*(0-y6) == (y4-y5)*(x7-x6)` is checked under the constraints obtained so far. Both sides of this equation are simplified to `x2*y2` by the current Gröbner base. This equation holds under the constraints.

7. A goal `para(x4,y4,x7,0,x5,y5,x6,y6)` is unified to the head of a clause for **para** and the equation `(x4-x7)*(y5-y6) == (y4-y5)*(x7-x6)` is checked. Both sides of this equation are simplified to `y1*(x1-x3)`, and therefore the equation holds.

During execution of CAL programs, each encounter with a constraint causes invocation of the constraint solver. If the new constraint is proved to be inconsistent with the previous ones, the execution fails and backtracks.

The above example uses Gröbner bases indirectly via the predicate `==`. The following is an example of using Gröbner bases of constraints directly.

```
horseandman(Horses,Men,Heads,Legs) :-
    Heads = Horses + Men,
    Legs = 2 * Men + 4 * Horses.
```

This program solves the Horse-and-Man problem. Let `horseandman(H,M,5,14)` be a goal. Then a Gröbner base of the constraints given in the body, i.e. {`Horses = 3,Man = 2`}, is computed and displayed.

5 Configuration of the CAL interpreter

The interpreter for CAL consists of three modules; preprocessor, inference engine, and constraint solver. These modules are related as shown in the following figure.

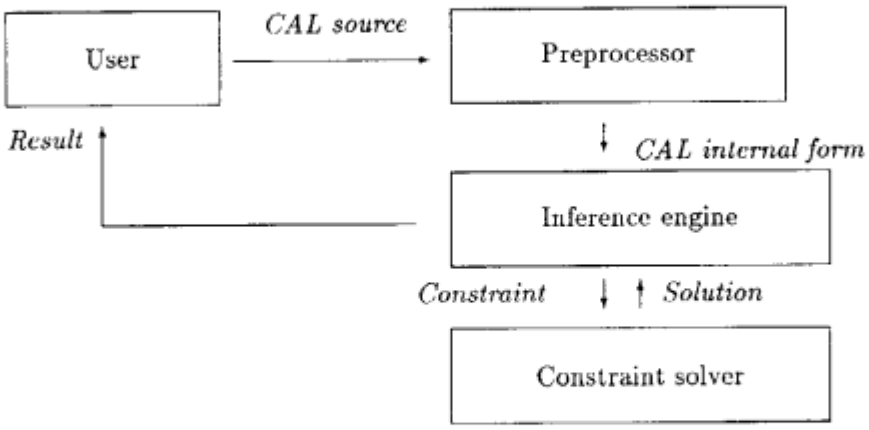

Figure 2 Configuration of the CAL interpreter

CAL source programs are received by the preprocessor and transformed to the corresponding internal forms. They are then passed to the inference engine and stored in the system.

CAL goals are received by the inference engine and translated into the internal forms. They are then executed immediately according to the CAL program currently stored in the system. When the execution is successfully terminated, the result is displayed on the terminal.

Whenever a constraint is obtained during the execution, the constraint solver is invoked to solve it. The constraint solver computes a new Gröbner base from the new constraint and the old Gröbner base for the previously obtained constraints.

6 CAL for Boolean expressions

The previous sections discussed the original CAL for algebraic polynomial equations in the domain of all rational numbers (or, more precisely, all algebraic numbers). We implement a modified version of CAL, which accepts Boolean expressions as constraints whose typical domain is the set of the truth values. Its constraint solver uses a somewhat different algorithm from Buchberger's original algorithm to obtain Boolean Gröbner bases. The precise definition of Boolean Gröbner bases and some theoretical considerations are in preparation for publication.

This version of CAL makes it very easy to write programs that require logical evaluation. For example, writing a program which verifies logical circuits will be easy.

7 Future extensions

At present, we have several plans to extend the facilities of CAL. In the current version of CAL, the (virtual) value of each variable in constraints can be any algebraic number, i.e. a complex number which can be a solution of a polynomial equation with integer coefficients. However, if it is known that the value of a certain variable can only be a real number, then a contradiction is immediately detected from constraints such as $x^2 + 1 = 0$. Thus, if there were a powerful constraint solver with knowledge about the smaller domain, the computation time might be reduced significantly.

One may want to write non-algebraic constraints such as $sin(X) = 1$ and $e^x = \pi$. In this case, we might have to widen the domain to the set of all complex numbers.

There must be a wide variety of these kinds of requirements in writing and solving constraints. The only thing we can do to satisfy these requirements is to make the constraint solver completely open and customizable. In this policy, the system is designed to allow users to redefine the constraint solver for their own purpose, for example, new types of constraints, other domains, and efficiency. In this sense, the two versions of CAL described above are different instances of the same language scheme. Because of this, the constraint solver can work almost independently of other modules in the system.

The first plan of the extension of CAL is to allow the user to write constraints for different domains in the same CAL program. For example, suppose that one constraint is for a Boolean value and the other is for an algebraic value. In this case, the system should pass the constraints to the corresponding solver. That is to say, a Boolean constraint should be solved in the boolean solver, and an algebraic constraint should be solved in the algebraic solver. To recognize the corresponding solver to each constraint, we are now planning to introduce types to constraints. In fact, types will be assigned to the parameters of each predicate. Suppose that the predicate p has the type *boole* $\times$ *algebraic* and the following clause is given:

```
p(¬ X,Y+1) :- ...
```

If the goal `p(a, b*c)` is given, it is matched with the head of the above clause, and the constraints `¬ X = a` and `Y+1=b*c` are passed to the boolean solver and the algebraic solver, according to the assigned types.

We can imagine a more complicated situation. For example, let X be an algebraic number, and let A and B be vectors of algebraic numbers. To solve constraints such as $X * A = A$ and $(A, B) = 0$, where (A, B) denotes the inner product of A and B, the vector solver and the scalar solver probably have to cooperate with each other. Therefore, there may be cases in which interface among constraint solvers is needed. However, we believe that this problem can be solved within user-defined constraint solvers, and will not cause a modification of the overall design of the CAL system itself.

References

[1] B. Buchberger. *Gröbner bases: An Algorithmic Method in Polynomial Ideal Theory.* Technical Report, CAMP-LINZ, 1983.

[2] A. Colmerauer. Opening the Prolog III universe: A new generation of Prolog promises some powerful capabilities. *BYTE*, 177–182, August 1987.

[3] M. Dincbas, H. Simonis, and P. Van Hentenryck. *Extending Equation Solving and Constraint Handling in Logic Programming.* Technical Report TR-LP-2203, ECRC, February 1987.

[4] N. Heintze, J. Jaffar, C. S. Lim, S. Michaylov, P. Stuckey, R. Yap, and C. N. Yee. *The CLP Programmer's Manual, Version 1.0.* Department of Computer Science, Monash University, 1986. Internal Memo.

[5] D. Hilbert. Über die Theorie der algebraischen Formen. *Math. Ann.*, 36:473–534, 1890.

[6] J. Jaffar and J-L. Lassez. *Constraint Logic Programming.* Technical Report, IBM Thomas J. Watson Research Center, 1986.