知識工学と人工知能　52－8
（1987．5．7）

ガード付節集合による知識表現とその処理システムの構成

淡　誠一郎[†]　青木　直明[†]　北橋　忠宏[††]　手塚　慶一[†]
[†]大阪大学工学部　[††]大阪大学産業科学研究所

現実システム上の知識表現としてホーン節集合が有用であるという事実は否めないが，対象領域の知識がホーン節集合では表現できない場合や，ホーン節だけで記述すると不自然になる場合が存在するのもまた事実である．そこで本研究では，一般の一階述語論理式が表現可能であり，かつホーン節論理の有用性も活かせる表現としてガード付節形式を提案する．
ガード付節は形式的にはガード付ホーン節の拡張であり，また，推論方法から見れば，多類論理の一般化である．多類論理計算においては，個体－集合あるいは集合－集合間の包含関係に注目し，これらの関係を他の関係とは区別して扱うが，ガード付節の計算はこの別処理される関係をホーン節で定義された関係へと一般化する．
以上の提案に基づいた推論システム（定理証明器）を構成し，定理証明の難問として知られるシューベルトのスティームローラー問題の証明を通してガード付節計算の有効性を示す．

KNOWLEDGE REPRESENTATION AND INFERENCE SYSTEM BASED ON GUARDED CLAUSE SET

Seiichiro DAN*, Naoaki AOKI*, Tadahiro KITAHASHI**, and Yoshikazu TEZUKA*

*Faculty of Engineering
**The Institute of Scientific and Industrial Research
Osaka University

The set of Horn clauses is one of the most available knowledge representation methods on practical knowledge based systems. However it is also true that the domain knowledge is not always naturally represented as a set of Horn clauses. In this paper, we propose an extended clause form called guarded clause form (GCF) to represent full first-order logical expressions and make use of the availability of Horn clauses at the same time.

GCF is an extended form of the guarded Horn clause for representing non-Horn clause as well as Horn clause, and its calculus is a generalized version of many-sorted calculus. In many-sorted calculus, the sort relations (instance-set, subset-superset) are dealt with in different way from other relations. Instead of these relations, the relations which are defined by only Horn clauses are specially processed in guarded clause calculus.

We demonstrate the availability of GCF and its calculus through the proof of Schubert's steamroller by an inference system implemented based on this proposal.

1. Introduction

This research is concerned with the knowledge representation and the reasoning system based on logic. Logic has been used to explain the structure of human thinking. Specially, the first-order predicate logic is one of the most powerful knowledge representation methods in the artificial intelligence field.

While the theoretical reseaches have been made for full first-order predicate logic, almost all practical logic based reasoning systems adopt mere Horn clause logic or its extensions. This is caused by so much complexity of full first-order calculus and relatively rich representability of Horn clauses. However, it is also true that Horn clauses do not always provide natural representation for the domain knowledge. Sometimes the knowledge cannot be represented without non-Horn clauses, and sometimes it can be represented without them but in unnatural way.

In this paper, we investigate the availability of an extended clause form called GCF: guarded clause form, which we propose as a knowledge representation method to express full first-order predicate logical formulas.

GCF is an extended form of the guarded Horn clause so as to be able to represent non-Horn clauses. A guarded clause has a guard part which is the precondition for the clause to be used. To make a resolvent from two parent clauses, it must be made sure that the conjunctive condition of the guard parts of both parents is satisfiable. By pruning the pairs whose conjunctive guard cannot be shown to be satisfiable, the search space is effectively limited. If the pruning cost could be kept small, the calculus of guarded clauses would be performed efficiently.

In order to keep the overhead of guard check relatively small, we define the well-defined guarded clause set (WGCS). A guard predicate is well-defined when it is defined by a set of Horn clauses. Under this condition, a guard check becomes a proof on a set of Horn clauses, so its cost becomes relatively small.

Specially, when we use only sort predicates as the guard part predicates in WGCS, the calculus is like a many-sorted calculus. Namely, it can be said that the calculus on WGCS is a generalized version of many-sorted calculus. In calculus on WGCS, the well-defined relations are specially processed instead of the sort relations in many-sorted calculus.

As the resolution methods for WGCS, GL-resolution and OGL-resolution are proposed. The GL-resolution is a sort of linear resolution for the set of guarded clauses and OGL-resolution is its ordered version. Both resolution methods are shown to be complete for WGCS.

Finally, a theorem proving system for WGCS is implemented based on the above proposals. With this system, the availability of the guarded clause calculus is demonstrated in proving Schubert's steamroller problem compared with the many-sorted calculus. The problem is a challenge problem for the mechanical theorem proving systems and it has been reported that the many-sorted calculus successfully proves this problem.

2. Guarded Clause Form

Any first-order formula can be expressed as a set of clauses which are of the form

{ H1,H2,...,Hm, ~B1, ~B2,...,~ Bn } (m,n>=0). (1)

Here, ~P means the negation of P. This is sometimes described as the following implication form,

H1,H2,...,Hm <- B1,B2,...,Bn (m,n>=0), (2)

which means "if B1 & B2 &...& Bn then H1 or H2 or...or Hm".

This standard form has made a great contribution to theoretical discussions and easy construction of theorem provers or logic programming environments. However it will be true that there is some information dropped in transforming into the standard form. The dropped information is not what we call semantics of formulas in the model theory but higher level information such as what roles the predicates play in the formula or how we use the rules in problem solving. Such information is so-called the meta-level knowledge.

The **guarded clause form** (abbreviated to **GCF**) is an extended clause form which reflects such information. The form is

H1,..,Hm<- B1,..,Bn | G1,..,Gk (m,n,k>=0), (3)

which means "when G1 & ... & Gk is true, the clause, H1 & ... & Hm <- B1 or ... or Bn, is available". The part H1,..,Hm , B1,..,Bn and G1,..,Gk are called **head, body** and **guard** respectively. The left side part of the clause divided by '|' is called **non-guard** part and the literals in this side are called **non-guard literals,** while the literals in the opposite side are called **guard literals.**

Expression (3) can be transformed into standard form,

H1,...,Hm <- B1,...,Bn,G1,...,Gk. (4)

Expressions (3) and (4) have the same semantics, namely, the one is true (false) in some interpretation M iff the other is true (false) in M. However they are different from each other in their way of use. Now, we consider linear resolution. As for standard form, any clause may be used if a literal in the clause is unifiable with a literal of center clause. While for GCF, it is required for a clause to be used that the guard part of the resolvent from this clause and the center clause is holding besides the unifiability.

A guarded clause whose guard part holds is said to be **meaningful,** otherwise **meaningless.** The guard part of a clause holds if every literals in it are logical consequence of axioms. When it contains variables, they are interpreted as being existentially quantified.

The resolvent of two parent guarded clauses is made in the same way as that of standard clauses except the additional condition that the guard literals of both parents must be placed at the guard part of their resolvent.

3. Linear Resolution for Guarded Clause Set

As checking a clause being meaningful is also just a proof, so it must be easy enough in comparison with the whole proof. In this chapter, we define some concepts for the calculus of guarded clauses to be available, and propose linear resolution methods with completeness for the guarded clause set.

It is known that Horn clause set has good property to deal with. But it is also known that knowledge for problem solving is not always described naturally as a Horn clause set. Here, for the guarded clause set to have richer ability of representation than Horn clause set and to make use of its efficiency at the same time, we define the following notions.

Definition 1 (well-defined)

Let S be a set of clauses, P be a predicate appearing in S, and S_P be a set of all clauses each of whose head contains at least one literal having predicate symbol P.

Predicate P is **well-defined** in S if the following conditions hold. And then the set S_P is called the **definition clause set** of P on S.

1) All clauses in S_P are Horn clauses.

Horn clause is defined as the clause whose head part contains at most one literal.

2) All clauses in S_P satisfy one of the following conditions.

2-1) The clause contains no literal having predicate symbol except P.

2-2) Otherwise, let S' be the set of clauses made by deleting all literals that have predicate symbol P from every clause in the set S-S_P. After this operation, every predicate except P in the clause is well-defined in S'.

Definition 2 (WGCS)

Let S and S' be a set of guarded clauses and its corresponding set of standard clauses made by replacing '|' with ','.

The set S of guarded clauses is said to be a **well-defined guarded clause set (WGCS)** if every guard predicate of every guarded clause in S is well-defined in S'.

Now, we are ready to discuss the linear resolution methods for the guarded clause set, specially for WGCS. The linear resolution and its variations such as input resolution, OL-resolution, SNL-resolution, etc. are widely used in theorem proving and logic programming

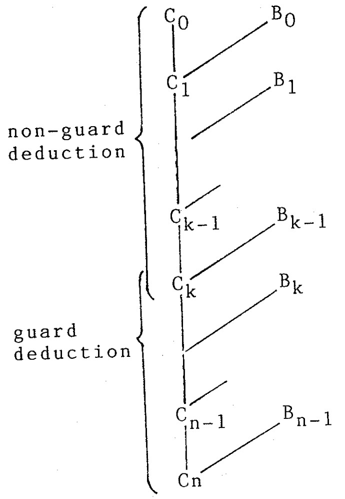


Fig. 1 Guarded linear resolution.

fields because they are relatively efficient and easy to implement on machine.

Definition 3 (GL-resolution)

Given a set S of guarded clauses and a guarded clause C0 which is not a guard definition in S. A **guarded linear** (abbreviated to **GL** below) **deduction** of Cn from S with top clause C0 is a deduction of the form shown in Fig. 1 where

1) for i=0,1,...,n-1, Ci+1 is a resolvent of Ci (called a center clause) and Bi (called a side clause),

2) each Bi is either in S, or is a Cj for some j, j < i,

3) for i=0,1,..,n-1, if the non-guard part of Ci is not empty, then the literal resolved upon in Ci for deduction of Ci+1 is a non-guard literal, only otherwise it is a guard literal, and

4) the guarded clauses in the deduction tree are always meaningful.

A GL-deduction of the empty guarded clause (<-|) is called a **GL-refutation.**

A GL-refutation tree T is composed of upper subtree and lower subtree. The upper one is a deduction tree of the clause Ck which is the first center clause having no non-guard literal in T, namely,

Ck: <-| G1,G2,...,Gn (n>=0). (5)

The lower subtree is a refutation tree from the top clause Ck, where all clauses are Horn clauses. When the fourth condition is kept holding in the course of a resolution, we get success in refutation as soon as a center clause formed (5) is deduced. Because the fact that the clause formed (5) is meaningful is exactly the fact that there is at least one refutation from it.

Just like OL-resolution to linear resolution, by introducing the notion of the order of literals in a guarded clause into GL-resolution, we present the guarded linear resolution for ordered guarded clause set, **OGL-resolution** as follows.

Definition 4 (the largest literal)

Given a guarded clause C. A literal L in C is **the largest literal** of C when

1) L is the left most literal in non-guard part of C, or

2) C has no literal in non-guard part and L is the left most literal in guard part of C.

Definition 5 (OGL-resolution)

An **OGL-deduction** is such a restricted GL-deduction that the third condition in Definition 3 is replaced by

3') for each i=0,1,...,n-1, the literal resolved upon in Ci for the deduction of Ci+1 is the largest literal in Ci.

An OGL-deduction of the empty guarded clause is also called an **OGL-refutation.**

Specially, if a WGCS S is composed of only guard definitions, then the OGL-deduction on S is just like a SNL-deduction, and if all clauses in S have empty guard then it is an OL-deduction.

In definition 4 and 5, the concepts, factoring or framed literal are omitted for simplicity.

Finally in this chapter, we refer to completeness. Completeness is an important property of proof method. The GL- and OGL-resolution have completeness for WGCS. Here,

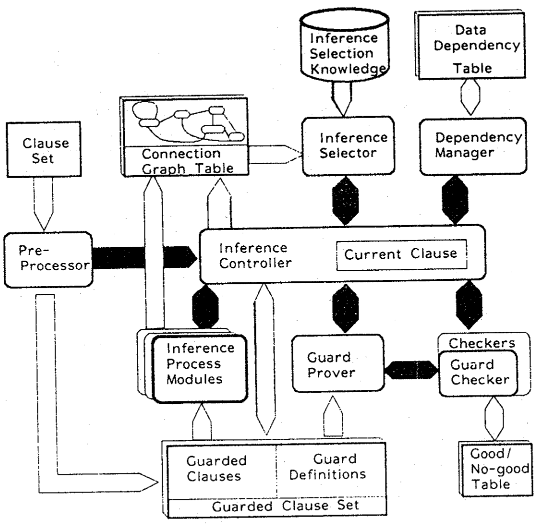


Fig. 2 System module structure.

the proofs are omitted because of the page limitation, but on the insight over the discussions of completeness of linear resolution and some other restricted linear resolutions, it will be easy to guess their completeness for WGCS.

4. Implementation

This chapter presents a brief description of a prototype system for mechanical theorem proving based on GCF. The module structure of this system is shown in Fig. 2. The software of this system is written in Prolog language on Toshiba AS3000 (Sun-3).

The user puts a problem described as a set of standard formed clauses to the system. The **preprocessor** transforms it into a **guarded clause set(GCS)**, which is divided into a set of the **guard definitions(GD)** and the rest set simply called the **guarded clauses(GC).** In this transformation, the user can select one of the following three modes:

1) **PURE:** no predicate is used as guard, namely each clause of the form (1) is transformed into a guarded clause of the form

H1,...,Hm <- B1,...,Bn | (m,n>=0), (6)

2) **MSL:** many-sorted logic type, every unary well-defined predicate is used as guard, and

3) **GCF:** every well-defined predicate is used as guard.

After this preprocess, the control of the proof is handed to the inference controller. It selects a guarded clause among the subset of guarded clauses which represents the negation of the theorem to be proved. Below, the term 'clause' will be used for guarded clause unless otherwise mentioned. This selected clause is the first current clause, namely, the top clause of a linear resolution. The **current clause** is the center clause to be processed at each time.

The inference controller performs the following processes until the non-guard part of the current clause becomes empty.

Step1: let each **inference process module** propose all possible deductions for current clause with their resulting clauses. These are recorded in the **connection graph table.**

One inference process module is provided for one independent primitive inference such as resolution and factoring. Each possible deduction is characterized by the primitive and the literals which are used in the deduction.

Step 2: for each proposed deduction, consult the **guard checker** about whether its resulting clause is meaningful or not. If not, delete the deduction from the connection graph table.

The guard check is performed by invoking the guard prover based on SNL-resolution. To check the guard efficiently, the literals in the guard part are clustered into some independent groups and each group is proved in SNL-resolution. In this process, if a group of literals is made sure to be provable, the group is recorded as good in the **good/no-good table,** otherwise as no-good. This information is used to cut the proof of the same group of literals.

Step 3: at this point, while there is no possible deduction left for the current clause, perform the following processes: by using the **dependency manager,** decide where to backtrack among center clauses, let the decided clause be new current clause and delete the deduction which made old current clause from the connection graph table.

If there are some deductions left, proceed to the following steps.

Step 4: determine one deduction to be performed among the possible deductions by invoking the inference selector. The current clause is updated by replacing the old one with the resulting clause of the selected deduction.

Step 5: perform some checks for new current clause: subsumption check, tautology check and so on. If it is shown that new current clause has bad properties, then abandon this clause with deletion of its deduction from the connection graph table, reset the current clause to the last center clause and back to step 3.

When no problem is found with new current clause, remove all redundant literals and all such guard literals that have no common variable with non-guard part from this clause.

As a special case heuristics, if a guard literal has only one set of solutions for its variables then substitute them over the current clause and remove that literal from this clause.

After these processes, new current clause is recorded in GC and the information of data dependency made in performing the deduction is recorded in the data dependency table.

To make determination of next deduction at step 4, the inference selector refers to the inference selection knowledge written in Prolog code provided by the user. If no selection knowledge is given, then the first deduction is selected as default. It is possible to make the knowledge so as for each selection to be made by the user. Then the prover can be considered as an interactive prover.

5. Experiments and Results

In order to demonstrate the availability of the guarded clause calculus, we tried to prove Schubert's steamroller problem by the system illustrated in the previous chapter.

Schubert's steamroller is a challenge problem for mechanical theorem proving(1). The problem of standard clause form and of guarded clause form (GCF mode) are shown in Fig. 3 and Fig. 4 respectively. This problem is hard to prove because of the rapidly spreading search space. Some solution methods have been reported. At present, the many-sorted resolution

<<< Schubert's Steamroller >>>

Wolves, foxes, birds, caterpillars, and snails are animals (7-11), and there are some of each of them (1-5). Also there are some grains, and grains are plants (6,12). Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants (13). Caterpillars and snails are much smaller than birds (14,15), which are smaller than foxes (16), which in turn are much smaller than wolves (17). Wolves do not like to eat foxes and grains (18,19), while birds like to eat caterpillars (20), but not snails (21). Caterpillars and snails like to eat some plants (22-25).

Therefore there is an animal that likes to eat a grain-eating animal (26,27).

```
<< Predicate Abbreviations >>
w(X): X is a wolf          f(X): X is a fox
b(X): X is a bird          c(X): X is a caterpillar
s(X): X is a snail         g(X): X is a grain
a(X): X is an animal       p(X): X is a plant
m(X,Y): X is much smaller than Y
e(X,Y): X likes to eat Y
h(X),i(X),j(X,Y): Skolem functions
==============================================
 (1) w(w) <-.     (2) f(f) <-.      (3) b(b) <-.
 (4) c(c) <-.     (5) s(s) <-.      (6) g(g) <-.
 (7) a(X) <- w(X).               (8) a(X) <- f(X).
 (9) a(X) <- b(X).              (10) a(X) <- c(X).
(11) a(X) <- s(X).              (12) p(X) <- g(X).
(13) e(X,Y),e(X,Z) <-
            a(X),p(Y),a(Z),p(W),m(Z,X),e(Z,W).
(14) m(X,Y) <- c(X),b(Y).
(15) m(X,Y) <- s(X),b(Y).
(16) m(X,Y) <- b(X),f(Y).
(17) m(X,Y) <- f(X),w(Y).
(18) <- f(X),w(Y),e(Y,X).
(19) <- g(X),w(Y),e(Y,X).
(20) e(X,Y) <- b(X),c(Y).
(21) <- b(X),s(Y),e(X,Y).
(22) p(h(X)) <- c(X).        (23) e(X,h(X)) <- c(X).
(24) p(i(X)) <- s(X).        (25) e(X,i(X)) <- s(X).
(26) g(j(X,Y)) <- a(X),a(Y).
(27) <- a(X),a(Y),e(X,Y),e(Y,j(X,Y)).
```

Fig. 3 Schubert's steamroller of standard clause form.

```
-------------<< Guard Definitions >>------------
 (1) w(w) <-|.     (2) f(f) <-|.     (3) b(b) <-|.
 (4) c(c) <-|.     (5) s(s) <-|.     (6) g(g) <-|.
 (7) a(X) <-|w(X).                (8) a(X) <-|f(X).
 (9) a(X) <-|b(X).               (10) a(X) <-|c(X).
(11) a(X) <-|s(X).               (12) p(X) <-|g(X).
(14) m(X,Y) <-|c(X),b(Y).
(15) m(X,Y) <-|s(X),b(Y).
(16) m(X,Y) <-|b(X),f(Y).
(17) m(X,Y) <-|f(X),w(Y).
(22) p(h(X)) <-|c(X).        (24) p(i(X)) <-|s(X).
(26) g(j(X,Y)) <-|a(X),a(Y).

--------------<< Guarded Clauses >>--------------
(13) e(X,Y),e(X,Z) <- e(Z,W)
                   | a(X),p(Y),a(Z),p(W),m(Z,X).
(18) <- e(Y,X)|f(X),w(Y).
(19) <- e(Y,X)|g(X),w(Y).
(20) e(X,Y) <-|b(X),c(Y).
(21) <- e(X,Y)|b(X),s(Y).
(23) e(X,h(X)) <-|c(X).   (25) e(X,i(X)) <-|s(X).
(27) <- e(X,Y),e(Y,j(X,Y))|a(X),a(Y).
-------------------------------------------------
```

Fig. 4 Schubert's steamroller of guarded clause form.

Table 1 Results of proving Schubert's steamroller.

	INTERACTIVE*			AUTOMATIC			
MODE	PURE	MSL	GCF	MSL		GCF	
Control	USER CONTROL			DEPTH FIRST	META SEARCH	DEPTH FIRST	META SEARCH
Generated Clauses	25	10	7	failed	23	9	9
Proof Depth	25	10	7		10	8	7
CPU Time (sec.)					202	247	94

* *The results of* INTERACTIVE *are for the optimum selection.*

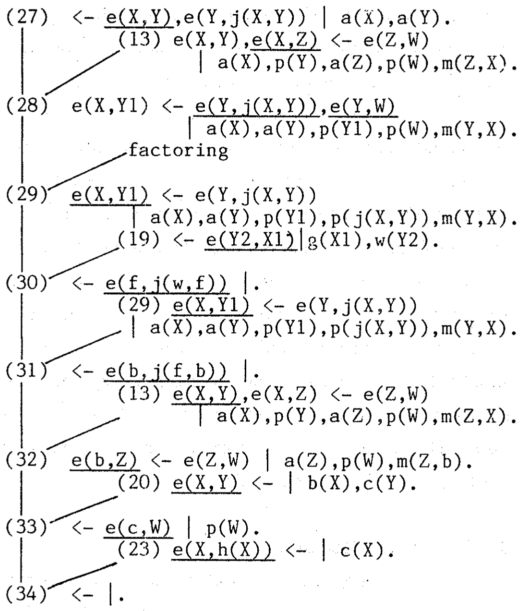
is the best solution among them.

The fundamental results of the proof of Schubert's steamroller are given in Table 1. Here, PURE, MSL and GCF are those described in the previous chapter. For each of them, the number of generated clauses, the depth of deduction for the refutation and CPU time are shown with the three different inference selection knowledges.

The first one, **INTERACTIVE**, is for the interactive prover mentioned above. As for this, the numbers shown in the Table 1 are of the case of the optimum selection. So the proof depth equals to the number of generated clauses. In this case, the differences among the three modes are merely caused by the fact that the steps of the proof for the guard part are not taken into account. However, the facts should be noted that each number reflects how many times the user must be answered during the proof and that the result 25 for PURE could be gained only after the user had learned enough by the proof of MSL or GCF.

As an instance, the refutation tree of GCF mode is shown in Fig. 5, which is also the tree of GCF with inference selection knowledge METASEARCH discussed below.

The other selection knowledges are for the automatic prover. **DEPTHFIRST** is the default selection, namely, simple left-to-right depth-first search and **METASEARCH** is the following selection. If both resolution and factor-



```
(27)  <- e(X,Y),e(Y,j(X,Y)) | a(X),a(Y).
         (13) e(X,Y),e(X,Z) <- e(Z,W)
                   | a(X),p(Y),a(Z),p(W),m(Z,X).
(28)  e(X,Y1) <- e(Y,j(X,Y)),e(Y,W)
                  | a(X),a(Y),p(Y1),p(W),m(Y,X).
         factoring
(29)  e(X,Y1) <- e(Y,j(X,Y))
              | a(X),a(Y),p(Y1),p(j(X,Y)),m(Y,X).
         (19) <- e(Y2,X1)|g(X1),w(Y2).
(30)  <- e(f,j(w,f)) |.
         (29) e(X,Y1) <- e(Y,j(X,Y))
              | a(X),a(Y),p(Y1),p(j(X,Y)),m(Y,X).
(31)  <- e(b,j(f,b)) |.
         (13) e(X,Y),e(X,Z) <- e(Z,W)
                   | a(X),p(Y),a(Z),p(W),m(Z,X).
(32)  e(b,Z) <- e(Z,W) | a(Z),p(W),m(Z,b).
         (20) e(X,Y) <- | b(X),c(Y).
(33)  <- e(c,W) | p(W).
         (23) e(X,h(X)) <- | c(X).
(34)  <- |.
```

Fig. 5 A refutation tree of Schubert's steamroller.

ing are applicable to the current clause, then factoring is selected and the first candidate among the possible factoring is tried. In case of resolution, the following three selections are sequentially performed: the literal resolved upon in the center clause, the side clause and the literal resolved upon in it. First, as the literal in the center clause, the literal which has the fewest number of variables is selected. Second, the clause which has the fewest number of literals is preferred as the side clause. Finally, if there are many candidates for the literal resolved upon in the selected side clause, the first one is tried first.

As mentioned above, the many-sorted calculus have been reported as the best solution method for Schubert's steamroller. However, it seems from the results of DEPTHFIRST that linear resolution with no (simple) control cannot reach the empty clause in case of many-sorted calculus (MSL), while the refutation is completed in case of full guard (GCF). This difference is caused by whether the predicate m(X,Y) (X is much smaller than Y) is used as guard or not. This fact shows that the calculus in GCF is not mere generalization of the many-sorted calculus but has higher ability of proof under appropriate selection of the guard predicates.

From the results of METASEARCH, it is concluded that the selection of guard predicates influences the efficiency of the proof, and meta-level knowledge is available in solving hard problem such as Schubert's steamroller.

Appropriate selection of guard predicates will be a difficult task if knowledge is already transformed into standard form. However it should be noted that the difficulty is not due to GCF. Knowledge represented in GCF with appropriate guard predicates seems rather natural than in standard form.

6. Concluding Remarks

In this paper, an extended clause form, guarded clause form was presented. This presentation is intended to provide both more powerful knowledge representation and efficiency to practical logic based reasoning system.

The guarded clause is an extension of guarded Horn clause from the view of clause form and its calculus is an extension of many-sorted calculus. The guarded clause form has richer representability than Horn clause form and at least same ability as the many-sorted form or the standard form.

For the guarded clause form to be available, the concepts 'WGCS' and 'meaningful guarded clause' were defined. GL-resolution and OGL-resolution were presented as resolution methods with completeness for WGCS. GL-resolution is such restricted linear resolution in whose deduction tree every clause is meaningful and non-guard literals precede guard literals as literals resolved upon. OGL-resolution is GL-resolution with the order of literals in a clause.

To verify the availability of guarded clause form and its resolution methods, a proving system was implemented. From the proof results of Schubert's steamroller on this system, the following facts were obtained,

1) the guarded clause form may be used as a method of task partition between human and machine in interactive proving environment, and

2) the guarded clause calculus has more possibility and could gain higher efficiency than the many-sorted calculus in practical use. The success depends on appropriate selection of guard predicates.

The idea of literal separation in a clause according to their roles may be also used for extended logical reasoning such as non-monotonic reasoning and default reasoning. Because beliefs, assumptions or defaults should be treated in different way from ordinary true-or-false predicates. We are going to make investigation in this direction.

Reference

(1) C. Walther : " A Mechanical Solution of Schubert's Steamroller by Many-Sorted Resolution", Artificial Intelligence, Vol. 26, pp. 217-224 (1985).

(2) C. Chang and R.C. Lee : " Symbolic Logic and Mechanical Theorem Proving ", Academic (1973).

(3) B. Silver : " Meta-Level inference ", North Holland (1986).

(4) J. de Kleer : " An Assumption-based TMS ", Artificial Intelligence, Vol. 28, pp. 127-162 (1986).

(5) H.J. Levesque : " Foundations of a Functional Approach to Knowledge Representation ", Artificial Intelligence, Vol.23, pp.155-212 (1984).