DESIGNING EXAMPLES FOR SEMANTICALLY GUIDED HIERARCHICAL DEDUCTION

Tie Cheng Wang
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

ABSTRACT

Semantically guided hierarchical deduction prover is a resolution-based theorem-proving procedure which is capable of using the domain dependent knowledge presented in well designed examples. This paper gives an overview of the basic deduction components of the prover, investigates some rules for human design of examples, and demonstrates their usage in proving several non-trivial theorems.

1. INTRODUCTION

A method for using domain dependent knowledge in mechanical theorem proving is to present the hypotheses of a theorem—i.e. the axiomatic system involved as a model (or say, an example*)—which is then used in guiding a mechanical prover to search a proof along with semantically provable paths. This method has been investigated by several researchers[2,3,5,6], where it was used mainly in some natural deduction proving procedures.

Though the basic idea of this method is simple and plausible, its potential usage has been limited by some problems or difficulties exposed in earlier implementations.

One problem is that this method often produces incompleteness for the traditional goal-oriented theorem provers, such as a linear resolution procedure or some natural deduction procedures. For example, consider a unsatisfiable set 5 of ground clauses, *S* — {A V -C V B, ->*A* V *B,C,-iB}*. Suppose the first three clauses of *S* are hypotheses. We use *M* = *{A,B,C}* as a model of these hypotheses to guide a linear resolution proof (refutation) of *S*. Let the last clause *-B* of 5 be the the first goal. The first step of deduction has two different branches: one produces a resolvent *-A* by resolving *-B* against -A V *B,* another produces a resolvent *Av -C* by resolving ->*B* against the first clause of *S.* Note that, for a semantically guided refutation proof, only the clauses that are false in the given model are treated as semantically provable, so can be used as goals during deduction. Because -^*A* is false in M, but *A* V *-C* is not, only the first branch can be developed next. Then, from *-A,* only a resolvent -C V *B* is produced by resolving ->*A* against the first clause of *S.* Now, because *-C* V *B* is true in *M* and there is no other deduction path left, the deduction has to terminate with failure. Some previous approaches tried to avoid this source of incompleteness by changing the model during deduction [5,6). For example, according to Example 1 of [5], a secondary model M' — *{->A, B,C}* should be used later to finally prove *S* (The set *S* of the above example is just a simplified version of the problem given in Example 1 of [5]). But, generally speaking, this strategy is not feasible to implement. One reason for this is that to design a "suitable" example for helping prove a hard theorem is not a easy matter. This is another problem encountered in using this method.

In fact, the design or selection of a suitable example for a theorem to prove is an essential problem with using this method. Of course, this task is difficult for a computer to do automatically because domain dependent knowledge is usually needed and also because it is hard to give a precise description of what is a suitable example. As for the human design, the difficulty lies in dealing with the interpretation of skolem functions (here, we are concerned with first-order logic only), because a casually-chosen interpretation of a skolem function may cause proof failure.

This paper tries to give a solution of the above problems. We will use examples to guide a resolution-based theorem prover, semantically guided hierarchical deduction prover (SHD-PROVER). Different from the usual goal-oriented deduction procedure, this procedure has a particular mechanism in producing and retaining all "legal" resolvents producible from a goal clause in each deduction step. In light of this feature, many failure cases caused by using models in a traditional goal-oriented prover can be avoided. For example, this procedure will retain both resolvents *-A* and *A* V ->C producible from the goal ->*B* at the first step of deduction of the above example. But the resolvent *A* V ->C which is true in *M*

This work was supported in part by NSF grant MCS-8313499.

* In this paper, the word "example" means a model that contains real semantics.

will be allowed to be used only as rule clause. Then a resolvent -C can be produced in the next step by resolving *-A* against this new rule. Because -C is false in *M*, it can be used as next goal to finally deduce a refutation of *S.*

The SHD-PROVER is based on a complete deductive component, hierarchical deduction procedure. The efficiency of this procedure is enhanced by several completeness-preserving refinements. The procedure can prove automatically a series of non-trivial theorems without the help of examples (see [8]). So, our approach emphasizes also the importance of a compact deductive component.

This paper mainly describes our method of designing and using examples for SHD-PROVER. we will investigate some rules for the human design of examples, especially, for the interpretation of skolem functions. More considerations about completeness and effectiveness problems will be included in these rules. As an application, this method will be used in designing examples to help our prover prove several non-trivial theorems, AM8, IMV, GCD, LCM and Schubert's statement. In order to include type information and other useful knowledge into examples, a three valued modeling and the corresponding interpretation method will be proposed.

The paper concludes with a summary of the results of computer runs in proving the example theorems.

II. OUTLINE OF THE HIERARCHICAL DEDUCTION

(See [8] for details. Those familiar with [8] should proceed to next section.)

The hierarchical deduction procedure proves a theorem by traversing a tree of nodes. Each node contains a different set of clauses and other information. All candidate goal clauses are contained in a goal-list. Each literal of each goal clause is indexed by a node name, through which a set of nodes can be located to obtain rule clauses for the resolution upon that literal.

At the beginning of a deduction, there is only one node (node l) which contains all input clauses, while the goal-list contains only the input goal clauses. The literals of the input goal clauses are all indexed by 1, which corresponds to the name of the first node. The general proof process is as follows:

1. The first goal clause G of the goal-list is taken. Along with the index of the left-most literal of G, a set of rule clauses is obtained by retrieving the node indicated by this index and ail parent nodes of this node. This index will be the parent name of the new node being produced in this round of deduction.
2. All "legal" resolvents are produced by resolving the goal clause against each clause of the set of rules upon the left-most literal of the goal clause. For each of the resolvents produced, the indices and the order of the literals inherited from the goal clause are retained, but the indices of the literals inherited from the rule clause are replaced by a larger integer which is just the name of a node being produced; these literals are placed to the left in the resolvent.
3. If CD (the empty clause) is obtained, then the procedure returns "proved". Otherwise, the resolvents will be stored into the new node and also be inserted into the goal-list according to their priorities. Then the above process is repeated.

The "legal" resolvents produced by the prover are called H-resolvents (hierarchical resolvents). They are produced under a number of constraints which will be briefly described next.

Local Subsumption Test: The literals previously resolved upon are retained by the resolvent as "framed" literals. A resolvent is said to be *locally subsumed* if it has more than one literal including its framed literals, sharing a common atom. The locally subsumed resolvent will be discarded.

Constraints on Common Tails: We call any index of a literal in a clause C which is less than the index of the left-most literal of C, a *tail index* of C. An index that is a tail index of two different clauses is called a *common tail indtx* of these clauses. We require that a legal resolvent of a goal clause must not be produced by resolving the goal upon the literal of a rule clause whose index is a common tail index of these clauses. We require also that the common tails (consisting of the literals with common tail indices) of the clauses being resolved be mergeable (unifiable).

Proper Reduction: An H-resolvent H is called a *proper reduction* of an ancestral goal clause G, iff H is a variant (including indices) of a proper part of G. Once a proper reduction is obtained, all other descendant resolvents produced from G (including G itself) will be discarded.

Global Subsumption Test: The hierarchical deduction has a feature that the resolvents grouped in the same node are similar to each other in the sense that the tails of these resolvents all are instances of the same part of a goal clause. Based on this property, the procedure can use an effective subsumption test method, so that only a few relevant resolvents need to be compared to determine whether a new resolvent is redundant.

Resolvent Evaluation and Reordering: The prover uses a best-first search strategy by which the candidate goal clause with largest priority is to be resolved first. The priority of each resolvent is assigned by an evaluation routine based on certain ideas, such as "twin symbol". These ideas are also used by a *level subgoal reordering* subroutine, which is responsible for selecting an "important literal" from a goal clause to be the left-most literal of that clause.

A Partial Set of Support Strategy and Others: This strategy is another use method of the model methodology (or say, set of support strategy) to reduce the search space. With this strategy, some literals of input clauses will be disallowed to resolve upon according to a model or "setting". The prover includes a finite forward chaining subroutine to produce some useful information for the evaluation routine. A special factoring algorithm, hierarchical factoring, is used by the procedure, which allows production of fewer factoring resolvents without causing incompleteness.

III. SEMANTICALLY GUIDED HIERARCHICAL DEDUCTION

Now, suppose *M* is a model of the input set of rule clauses. We require that only the resolvents that are false in *M* (that is, semantically refutable) can be used as goals by the hierarchical deduction procedure. Then this procedure becomes a so-called *semantically guided hierarchical deduction* (SHD-PROVER). Note, the newly produced resolvents that are true in *M* will still be retained in some node, so they may be used later as rules.

In order to incorporate type information and other useful knowledge into a model, the interpretation method to be discussed in this paper will be slightly different from the traditional one. In particular, we will allow the domain *D* of an interpretation to include a special element, *"unknown"*. Thus $D = Do \cup \{unknown\}$, where Do is an ordinary nonempty domain not containing the element *"unknown"*. This element will be used for indicating the interpreted value of semantically meaningless terms, atoms, or formulas. We restrict the use of our interpretation method to a subset of the first-order formulas, namely, the *simplified first order formulas.*

Definition. A simplified first-order formula is a quantifier free first-order formula containing no logical symbol other than -, A, V; and each negation symbol -i occurring in this formula must be applied to an atomic subformula (ie. atom).

All atomic formulas, clauses, and sets of clauses (treated as conjunctions of disjuncts) are simplified first-order formulas. The formulas in the *interpretation normal form,* which is to be introduced next, are also simplified first-order formulas.

Interpretation. An interpretation I for a simplified first-order formula *W* consists of a domain *D* and a definition for each non-variable symbol occurring in *W',* such that

- o Each n place predicate symbol *P* is interpreted in I as a function, $P : D^n \longrightarrow \{true, false, unknown\}$;
- o Each n place function symbol *F* is interpreted in I as a function, $F : D^n \to D$;
- o The logical symbols are interpreted in / traditionally, except $-unknown = unknown$, $unknown \wedge X = X$, and $unknown \vee X = unknown$, where $X \in \{true, false, unknown\}$;
- o For any term or simplified first-order formula W, the term or formula W', obtained from W by substituting all variables occurring in W with some elements of D, is called an interpretation instance of W over D. Let $[W']_I$ denote the interpreted value obtained by evaluating W' according to the interpretations of all non-variable symbols occurring in W';
- o Let C be any simplified first-order formula, then $[C]_I = unknown$ iff, for each interpretation instance C' of C over $D_0, [C']_I = unknown$;
  $[C]_I = false$ iff there is an interpretation instance C' of C over $D_0, [C']_I = false$;
  otherwise, $[C]_I = true$.

Domain. SHD-PROVER requires that the domain *D(Do)* of an interpretation be finite. This finite domain is obtained by the following method. For a theorem *W* to be proved, we first interpret it by using a natural domain (it may be infinite). Then the domain $D(D_0)$ used by the prover is produced mechanically by the following procedure:

```
PROCEDURE DOMAIN-GENERATOR {domain -limit);
  count := 1;
  D :— { e : t is the interpreted value of a constant (0 place function symbol) of W };
  WHILE count < domainJimit DO
    count := count + 1;
    D :— D U { e : c is the interpreted value of an interpretation instance of a term in W over D };
  Do := D — {unknown}.
```

The *domainJimit* can be given by the user. In practice, we prefer to use a small *domainJimit* first, such as 1, to generate a small size domain. If the prover fails, then repeatedly increase *dominJimit* and call the prover until a proof is obtained. We will give several examples of this procedure being used.

For SHD-PROVER, the resolvent that is interpreted as "unknown" will be discarded. It is noticed that the expense of obtaining the interpreted value of a literal or a term increases exponentially along with the increase of the number of different variables it contained. We have used a more efficient method (not given here) to implement this semantic test for avoiding the expensive evaluation.

IV. EXAMPLE DESIGN METHOD

Semantically guided hierarchical deduction, by retaining the resolvents which are true in the model as rules, has acquired a valuable property: if the prover is guided by a particular model, such that at most one semantically refutable and useful resolvent is produced in each step of a deduction, then the procedure will not lose a proof of a theorem. Our task in designing examples and interpreting skolem functions is to obtain this kind of model*.

According to this property and others, a model M with the following characteristics is preferable:

1. The model M should not represent a trivial case of the theorem.
2. Each rule clause should be interpreted to true in M, and the goal clause should be false in M.
3. Among a group of resolvents produced in each step of deduction, at most one useful resolvent is false in M (A useful resolvent is one which can be used by the hierarchical deduction procedure in continuing the current path to O).
4. A literal or a term should be interpreted to "unknown" for arguments that lead to semantically meaningless situations.

The above characteristic 3 is important to reduce the possibility that a semantic guide causes proof failure (it is also important to efficiency). Let us consider a set of clauses, $S_1 = \{Q \vee R, P \vee \neg Q, P \vee \neg R, \neg P\}$, with $\neg P$ as goal. Suppose we use $\{P, Q, \neg R\}$ as a model of the hypothesis clauses, then there is a successful deduction path in which only one useful resolvent false in the model can be produced in each step of deduction. Now, suppose we use $\{P, Q, R\}$ as a model, then the proof will fail, because the two useful resolvents $\neg Q, \neg R$ produced in the first round of deduction are both false in this model.

Certainly, it is usually not easy to construct a model that has such a characteristic. In the following, we will investigate a way toward a partial solution of this problem. Notice that the clause $P \vee \neg Q$ and $P \vee \neg R$ in the above example can be put together to form a formula, $P \vee [\neg Q \wedge \neg R]$. We will call this form of a formula *interpretation normal form.*

Definition. An interpretation norm form (IPN form) is a simplified first-order formula in the following form:

$$L_1 \vee \ldots \vee L_k \vee [C_1 \wedge \ldots \wedge C_h],$$

where $L_1 \ldots L_k$ are literals and $C_1 \ldots C_h$ are clauses.

For consistency, a formula with a form $L_1 \vee \ldots \vee L_k$ (a clause) is treated as a degenerate case of IPN form, where $h = 1$ and $C_1 = \Box$ (so is deleted); a formula with a form $C_1 \wedge \ldots \wedge C_h$ is treated as another degenerate case of IPN form, where $k = 1$ and $L_1 = \Box$ (so is deleted).

Note that, corresponding to each IPN form, there is a set of clauses, each of which contains a subclause C_j for some $j, 1 \leq j \leq h$.

Interpretation rules for a formula in IPN form: For each interpretation instance of an IPN form,

$$L'_1 \vee \ldots \vee L'_k \vee [C'_1 \wedge \ldots \wedge C'_h],$$

1. if $[L'_1 \vee \ldots \vee L'_k]_I = false$, then for each $j, 1 \leq j \leq h, [C'_j]_I$ should not be false, except when C_j is identical to a part of the goal clause
2. if $[L'_1 \vee \ldots \vee L'_k]_I = true$, then there should be at most one $j, 1 \leq j \leq h$, such that $[C'_j]_I = false$.

As a simple application of these rules, let us consider again to design a model for proving the set S_1 by the SHD-PROVER. First, we combine the clauses in S_1 into 3 IPN forms, $\{Q \vee R, P \vee [\neg Q \wedge \neg R], \neg P\}$. Because $\neg P$ will be the goal clause, P must be interpreted to true. Applying the above rule 2 to the interpretation of the IPN form $P \vee [\neg Q \wedge \neg R]$, we notice that at most one of $\neg Q$ and $\neg R$ can be interpreted to false. Thus the model $\{P, \neg Q, R\}$ $\{P, Q, \neg R\}$ is acceptable, but the model $\{P, Q, R\}$ is not.

In this paper, we will mainly consider the problem of designing examples for theorems that have real semantics. In this case, as it will be shown in what follows, the interpretation rules of IPN form will be mainly applied to interpret some skolem functions.

An example design method. Let *T* be a theorem of first-order theory, and S be the set of clauses obtained by skolemizing the negation of *T.* We design an example of *T* for SHD-PROVER by follows:

1. Design a real example *E* for T;
2. Interpret naturally each predicate of *S* according to its meaning in E;
3. Interpret naturally each function symbol (including constant) that has a corresponding instance in *E* (this kind of symbol is called interpreted symbol);
4. For each of the remaining function symbols occurring in *S* (the uninterpreted symbols), combine all clauses of *S* containing an occurrence of this symbol into an IPN form, then interpret this symbol, according to its meaning and following the interpretation rules for IPN form.

V. APPLICATION OF THE EXAMPLE DESIGN METHOD

1. AM8: The attaining minimum (or maximum) value theorem in analysis: a continuous function f in a closed real interval [a,b] attains its minimum (or maximum) in this interval.

To prove this theorem, we use a continuity axiom, and the least upper bound axiom instantiated by this particular problem. The theorem is formalized as follows (We did not present the needed inequality axioms):

$$\begin{aligned}
&[\ a \leq l \leq b \\
&\wedge \forall t(a \leq t \leq l \rightarrow f(l) \leq f(t)) \\
&\wedge \forall x(a \leq x \leq b \wedge \forall t[a \leq t \leq x \rightarrow f(x) \leq f(t)] \\
&\quad \rightarrow x \leq l) \\
&\wedge \forall w \exists g\{(a \leq w \leq b \rightarrow a \leq g \leq b \wedge f(g) \leq f(w)) \\
&\qquad \wedge \forall x([a \leq w \leq b \wedge a \leq x \leq b \wedge f(x) \leq f(w)] \\
&\qquad\quad \rightarrow g \leq x))\}] \\
&\rightarrow \exists u[a \leq u \leq b \wedge \forall t(a \leq t \leq b \rightarrow f(u) \leq f(t))].
\end{aligned}$$

We first transform the negation of this theorem into a simplified first-order formula (the predicate p stands for

* Not all theorems have such models, but most of the practical theorems seem to have, at least the only counter-example we found is very unnatural and complicated.

symbol $\leq$) F1:

$$\begin{aligned}&p(a,l)\\ &\wedge p(l,b)\\ &\wedge\{\neg p(a,t)\vee\neg p(t,l)\vee p(f(l),f(t))\}\\ &\wedge\{\neg p(a,x)\vee\neg p(x,b)\vee p(x,l)\vee[p(a,q(x))\wedge p(q(x),x)\wedge \neg p(f(x),f(q(x)))]\}\\ &\wedge\{\neg p(a,w)\vee\neg p(w,b)\vee[p(a,h(w))\wedge p(h(w),b)\wedge p(f(h(w)),f(w))\wedge(\neg p(a,x)\vee\neg p(x,b)\vee\neg p(f(x),f(w))\vee p(h(w),x))]\}\\ &\wedge\{\neg p(a,u)\vee\neg p(u,b)\vee[p(a,k(u))\wedge p(k(u),b)\wedge\neg p(f(u),f(k(u)))]\}.\end{aligned}$$

In the formula F1, we have forced all occurrences of every uninterpreted function symbol, such as q, h, k, to be contained in an IPN form. So the interpretation rules for IPN form can be used in interpreting these skolem functions.

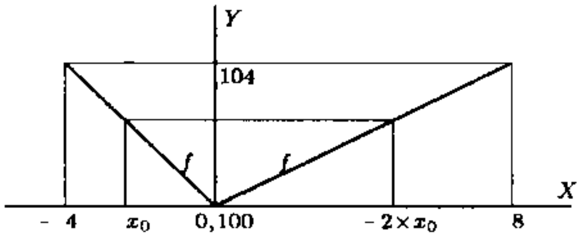

Figure 1. An example of theorem AM8

According to the meaning of the theorem, we designed an example, shown in Figure 1. The interpretation for the interpreted functions or predicates of the theorem are natural:

$$a=-4,\quad b=8,\quad l=0,$$

$$f(x)=\begin{cases}100-x, & x\in[-4,0];\\ 100+0.5\times x, & x\in(0,8];\\ \textit{unknown}, & \text{otherwise},\end{cases}$$

$$p(x,y)=\begin{cases}\textit{true}, & x\leq y\wedge x,y\in[-4,8];\\ \textit{false}, & y<x\wedge x,y\in[-4,8];\\ \textit{true}, & x\leq y\wedge x,y\in[100,104];\\ \textit{false}, & y<x\wedge x,y\in[100,104];\\ \textit{unknown}, & \text{otherwise}.\end{cases}$$

Notice that, by the above interpretation, an atom, such as $p(f(a),a)$, will be interpreted to "unknown", because it would be "semantically meaningless" to write $[f(a)]_I\leq[a]_I$ according to the example shown as Figure 1, since the two arguments of $\leq$ belong to different types: the point $[f(a)]_I=104$ is on the Y-axis and the point $[a]_I=-4$ is on the X-axis. Notice also that, we defined the function f only in the interval $[-4,8]$ of X-axis. This restriction allows the procedure to prevent some redundant resolvents. For example, a resolvent containing a term $f(f(a))$ or $f(f(l))$ must be discarded, because f is undefined in $[f(a)]_I$ or $[f(l)]_I$.

Now we interpret the skolem function q, which is contained in the fourth conjunct of F1:

$$\neg p(a,x)\vee\neg p(x,b)\vee p(x,l)\vee[p(a,q(x))\wedge p(q(x),x)\wedge\neg p(f(x),f(q(x)))]$$

For the meaningful interpretation, we need only to consider two cases:

1. $x\in[-4,0]$. Then $[p(x,l)]_I=\textit{true}$. By interpretation rule 2 for IPN form, at most one among the literals $p(a,q(x)), p(q(x),x)$ and $\neg p(f(x),f(q(x)))$ (each corresponds a C_j in the definition of IPN form) can be interpreted to false. So we can interpret q to be a function, which maps the point $x, x\in[-4,0]$, to a point $x', x'\in[-4,-2\times x)$ (see Figure 1). In contrast, if q were interpreted to map the point x to a point point $x'', x''\in[-2\times x,8]$, then there would be two literals $P(q(x)),x)$ and $\neg P(f(x),f(q(x)))$ interpreted to false. We tested some different interpretations, such as $q(x)=-4, q(x)=x$ (AM8^a of Table 1), $q(x)=-2\times x$ (AM8^b of table 1) and $q(x)=8$, for $q(x), x\in[-4,0]$, by running the prover with identical other conditions. The prover succeeded with the use of the first two interpretations, which satisfy the interpretation rules, and failed with the use of the last two interpretations, which did not satisfy the interpretation rules.

2. $x\in(0,8]$. Then $[p(x,l)]_I, [\neg p(a,x)]_I$ and $[\neg p(x,b)]_I$ are all false. By interpretation rule 1 for IPN form, we should interpret all of $p(a,q(x)), p(q(x),x)$ and $\neg p(f(x),f(q(x)))$ to true. Thus the following relations should be satisfied:

$$0<x\leq 8\wedge -4\leq q(x)\leq x\wedge f(q(x))<f(x)).$$

Consulting Figure 1, we define $q(x)=0.5\times x$ for $x\in(0,8]$. The final interpretation for q can be

$$q(x)=\begin{cases}x, & x\in[-4,0],\\ 0.5\times x, & x\in(0,8],\\ \textit{unknown}, & \text{otherwise}.\end{cases}$$

By a similar process, we interpret the skolem function h, contained in the fifth conjunct of F1, as follows:

$$h(x)=\begin{cases}x, & x\in[-4,0],\\ -0.5\times x, & x\in(0,8],\\ \textit{unknown}, & \text{otherwise}.\end{cases}$$

The skolem function k occurs in the last IPN form of the above formula F1:

$$\neg p(a,u)\vee\neg p(u,b)\vee[p(a,k(u))\wedge p(k(u),b)\wedge\neg p(f(u),f(k(u)))].$$

Because the last conjunct, $\neg p(f(u),f(k(u)))$, together with the first two disjuncts will form the goal clause of the input set, which should be interpreted to false. According to the meaning of the theorem, the function k(u) is meaningful only for u=0 in this example, so we interpret k as follows:

$$k(u)=\begin{cases}4, & u=0,\\ \textit{unknown}, & \text{otherwise}.\end{cases}$$

The domain D_0 is obtained by the subroutine `DOMAIN_GENERATOR`, where we set *domain_limit* = 3,

$$D_0 = \{-4.0, -2.0, 0.0, 4.0, 8.0, 100.0, 102.0, 104.0\}.$$

By the help of this example, the prover obtained a quite efficient proof (see Table 1): only 68 resolvents accepted, among which 30 were useful. 4 useful resolvents were produced by resolving goals against some resolvents that were true in the model. So the hierarchical deduction structure, which retained these resolvents as rules, was important for this proof.

2. IMV: The mean value theorem in analysis: If function f is continuous in a real closed interval $[a, b]$, where $f(a) \leq 0$ and $0 \leq f(b)$, then $\exists x f(x) = 0$.

Several interesting interpretations of uninterpreted skolem functions, similar to the interpretation of skolem function q in AM8, were encountered in designing examples for this theorem. One can find the set of clauses for this theorem in [8]. This problem has been considered in [2]. A different example was designed by us according to our interpretation rules.

3. GCD: If $gcd(a, b)$ is the greatest common divisor of two positive integers a and b, then for any positive integer e, $gcd(a \times e, b \times e) = gcd(a, b) \times e$.

Let $g(x, y, u)$ denote $u = gcd(x, y)$, and $d(x, y)$ denote x properly divides y. Then the theorem is represented as the follows:

$$\begin{aligned}&[\ g(a,b,c)\\ &\wedge \forall x \forall y \forall u \{g(x,y,u) \leftrightarrow d(u,x) \wedge d(u,y) \wedge \forall v[d(v,x) \wedge d(v,y) \rightarrow d(v,u)]\}\\ &\wedge \forall x \forall y \exists u (d(u,x) \wedge d(u,y) \wedge \forall v[d(v,x) \wedge d(v,y) \rightarrow d(v,u)])]\\ &\quad \rightarrow g(a \times e, b \times e, c \times e).\end{aligned}$$

The second conjunct of the above formula is the definition of positive integers gcd. The third conjunct is the assertion of the existence of the gcd. Number theory is a typical example for this theorem. We interpreted a, b, c, d, e, g naturally:

$$a = 6, \quad b = 10, \quad c = 2, \quad e = 7.$$

$$g(x,y,u) = \begin{cases} true, & u = gcd(x,y), \\ unknown, & \text{otherwise,} \end{cases}$$

$$d(x,y) = \begin{cases} true, & x \text{ properly divide } y, \\ unknown, & \text{otherwise.} \end{cases}$$

Skolemizing the definitions of gcd and the third conjunct introduced skolem functions i and k, each of which was forced to be contained in an IPN form:

$$\neg d(u,x) \vee \neg d(u,y) \vee g(x,y,u) \vee [d(i(x,y,u),x) \wedge d(i(x,y,u),y) \wedge \neg d(i(x,y,u),u)],$$

$$d(k(x,y),x) \wedge d(k(x,y),y) \wedge [\neg d(v,x) \vee \neg d(v,y) \vee d(v,k(x,y))].$$

The following interpretations for i and k satisfy interpretation rules for IPN forms, which were used by the prover:

$$i(x,y,u) = 1,$$
$$k(x,y) = gcd(x,y).$$

A subset of axioms about the product operator $\times$, and the proper division operator $/$ was used in proving this theorem. These operators were interpreted according to their natural meanings. The domain is produced by `DOMAIN_GENERATOR` with *domain_limit* $= 1$. An commutative unification algorithm and a sorting ground commutative terms subroutine were used by the prover to avoid explicitly including commutative laws into input set of clauses.

4. LCM: If $lcm(a, b)$ is the least common multiple of two positive integers, a and b, then

$$lcm(a,b) = (a \times b)/gcd(a,b).$$

The above theorem GCD, was used as a lemma in proving this theorem. Let $l(x, y, u)$ mean $u = lcm(x, y)$, and let the predicates g and d be the same as that of GCD, then the theorem is represented as follows:

$$\begin{aligned}&[\ g(a,b,c)\\ &\wedge \forall x \forall y \forall u \{g(x,y,u) \rightarrow \forall w[g(x \times w, y \times w, u \times w)]\}\\ &\wedge \forall x \forall y \forall u \{g(x,y,u) \leftrightarrow d(u,x) \wedge d(u,y) \wedge \forall v[d(v,x) \wedge d(v,y) \rightarrow d(v,u)]\}\\ &\wedge \forall x \forall y \forall u \{l(x,y,u) \leftrightarrow d(x,u) \wedge d(y,u) \wedge \forall v[d(x,v) \wedge d(y,v) \rightarrow d(u,v)]\}]\\ &\quad \rightarrow l(a,b,(a \times b)/c)\end{aligned}$$

The same subset of axioms for the operators $\times$ and $/$, and a similar strategy in dealing with commutative laws as that for proving GCD were used in proving this theorem. The example designed for this theorem is similar to GCD, but is augmented by interpretations for predicate l and a skolem function j obtained by skolemizing the definition of LCM. We do not go into much detail.

5. SST: In AAAI 1984, C. Walther reported his success of using many-sorted resolution to prove Schubert's statement (SST). It was said that this problem was so hard, that no automatic prover could ever obtain a proof, except the many sorted resolution prover. After the meeting, we submitted this problem, together with an example designed according to the meaning of the statement, to our SHD-PROVER. Without any difficulty, the prover succeeded.

To understand our design of the example for this problem, the reader can refer to [7] for finding the original statement of the problem and the resulted set of clauses.

Several skolem constants occur in the set of clauses, each of which corresponds to a type of entities, where

w_0: a wolf	b_0: a bird	s_0: a snail
f_0: a fox	c_0: a caterpillars	g_0: grain.

To facilitate implementation, we interpret each entity as a integer:

$w_0 = 13, f_0 = 11, b_0 = 7, c_0 = 5, s_0 = 3, g_0 = 2.$

With the above interpretations of entities and according to the meaning of each sentence of the statement, we interpret the predicates occurring in the set of clauses as follows:

$$\begin{array}{lll}
W(x) = true, & \text{if } x = 13, & \text{otherwise } unknow; \\
F(x) = true, & \text{if } x = 11, & \text{otherwise } unknow; \\
B(x) = true, & \text{if } x = 7, & \text{otherwise } unknow; \\
C(x) = true, & \text{if } x = 5, & \text{otherwise } unknow; \\
S(x) = true, & \text{if } x = 3, & \text{otherwise } unknow; \\
G(x) = true, & \text{if } x = 2, & \text{otherwise } unknow; \\
P(x) = true, & \text{if } x = 2, & \text{otherwise } unknow;
\end{array}$$

$$A(x) = \begin{cases} true, & x \in \{3,5,7,11,13\}, \\ unknown, & \text{otherwise}; \end{cases}$$

$$M(x,y) = \begin{cases} true, & x < y \wedge x, y \neq 2, \\ false, & y < x \wedge x, y \neq 2, \\ unknown, & \text{otherwise}; \end{cases}$$

$$E(x,y) = \begin{cases} true, & x = 13 \wedge y \in \{7,5,3\}, \\ false, & x = 13 \wedge y \in \{2,11,13\}, \\ true, & x = 11 \wedge y \in \{7,5,3\}, \\ false, & x = 11 \wedge y \in \{2,11,13\}, \\ true, & x = 7 \wedge y \in \{5,2\}, \\ false, & x = 7 \wedge y \in \{3,7,11,13\}, \\ true, & x = 5 \wedge y = 2, \\ false, & x = 5 \wedge y \neq 2, \\ true, & x = 3 \wedge y = 2, \\ unknown, & \text{otherwise}. \end{cases}$$

Three skolem functions occurred in the set of clauses, each of which could be forced to be contained in an IPN form:

$h : \neg C(x) \vee [P(h(x)) \wedge E(x, h(x))],$
$i : \neg S(x) \vee [P(i(x)) \wedge E(x, i(x))],$
$j : \neg A(x) \vee \neg A(y) \vee [G(j(x,y)) \wedge (\neg E(x,y) \vee \neg E(y, j(x,y)))]$

Then they were interpreted as follows:

$h(x) = 2, \quad \text{if } x = 5, \quad \text{otherwise } unknow;$
$i(x) = 2, \quad \text{if } x = 3, \quad \text{otherwise } unknow;$

$$j(x,y) = \begin{cases} 2, & \text{if } x, y \in \{3,5,7,11,13\}, \\ unknown, & \text{otherwise}. \end{cases}$$

The domain D_0 consists of a finite set of integers:

$D_0 = \{2,3,5,7,11,13\}.$

We should emphasize that, different from the many-sorted resolution, our use of this example includes both type information and the knowledge for guiding the prover to select semantically provable paths.

Table 1 is a summary of the computer runs by SHD-PROVER in proving (without interaction) these theorems with the help of the above examples. The prover was run on the Symbolics 3600. The code has not been well optimized. The heuristic search strategy (it was described in [8]) based on syntactic features was important to these proofs, but no particular heuristics were added to help prove different theorem, except some control parameters, such as the limit of search depth, the limit of function nesting depth, etc. may be different.

Table 1. A summary of computer run

Theorem	AM8[a]	AM8[b]	IMV	GCD	LCM	SST
Semantic test*	9	80	13	10	34	69
Locally subsumed	23	313	17	19	54	8
Tail unmergeable	12	256	4	0	4	4
Proper reduction	12	21	6	0	2	1
Globally subsumed	0	30	0	23	41	3
CPU second	14	125	14	41	82	16
Useful resolvents	30	Fail	38	23	22	58
Accepted resolvents	68	256	69	70	83	82

ACKNOWLEDGMENTS

The author is grateful to W. W. Bledsoe who contributed valuable suggestions and helped with phrasing for this manuscript, also to L. Wos, L. J. Henschen and A. M. Ballantyne for their help related to this topic.

REFERENCES

[1] Bledsoe, W. W. Non-resolution theorem proving. Artificial Intelligence 9 (1977), 1-35.

[2] Ballantyne, A.M. and Bledsoe, W. W. On generating and using examples in proof discovery. Machine Intelligence 10(1982).

[3] Gelernter, H. Realization of a geometry theorem-proving machine. Proc. IF IP congr.(1959).

[4] Kowalski, R., and Kuehner D. Linear resolution with selection function. Artificial Intelligence 2 (1971), 227-260.

[5] Plaisted D. A. Using examples, case analysis, and dependency graphs in theorem proving. 7th International Conference on automatic deduction. (1984)

[6] Reiter, R. A semantically guided deductive system for automatic theorem proving, Proc. 3rd IJCAI(1973)41-46.

[7] Waither, C. A mechanical solution of Schubert's steamroller by many-sorted resolution. Proc. 8rd AAAI(1984)330-334.

[8] Wang, T. C. Hierarchical Deduction. Tech. Report ATP-78, Univ. of Texas at Austin, March, 1984.

* It is counted as the number of resolvents which are not false in the model.