Using Specialists to Accelerate General Reasoning

Stephanie A. Miller & Lenhart K. Schubert
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1

Abstract

Theorem provers are prone to combinatorial explosions, especially when the reasoning chains needed to establish a desired result are lengthy. To alleviate this problem, special purpose inference methods have been developed that exploit properties of certain domains to shorten chains of reasoning with types, temporal relations, colors, numeric relations, and sets, to name a few.

The problem investigated here is how to use these efficient, but limited methods in a more general environment. Although much research has been done on this problem, most of the resulting systems either restrict what they can represent and reason with, limit the types of special mechanisms that can be used, or are difficult to extend with other specialists.

We develop a uniform interface to specialists which allows them to assist a resolution-based theorem prover in function evaluation, literal evaluation, and generalized resolving and factoring. The specialists incorporated into this system include a temporal reasoner, a type hierarchy, a number specialist, a set specialist, and a simple color specialist. Each new specialist was found to make possible fast proofs of questions previously beyond the scope of the theorem prover. Examples from the fully operational hybrid system are included.

1. Introduction

General theorem provers all suffer from combinatorial explosions. However, for some frequently encountered subdomains, special purpose inference methods have been developed that reason faster than any general method can by exploiting the properties of those subdomains. These *specialists* may use completely different representations and methods to achieve their performance. For example, a type specialist can use a type hierarchy to short-cut the chain of reasoning involved in determining that a wolf is a living-thing (the sequence of inferences *wolf -> warm-blooded-quadruped -> larger-animal -> animal -> creature -> living-thing* can be reduced to *wolf -> living-thing*, using a preorder numbering scheme in the hierarchy). Similarly, a time specialist would compress reasoning about transitive temporal orderings, such as inferring that event1 is before event4, given that event1 is before event2, event2 is before event3, and event3 is before event4. Other specialists could deal with arithmetic relationships, sets, spatial relationships, colors, and so on.

The question then arises as to how to use these efficient, but limited methods in a more general environment. This is the problem central to this paper. Ideally, the specialists should be integrated with the general mechanism in such a way that the specialists will be used when appropriate, and the general method when no specialists apply. This would give us a system with a wider domain than all the specialists combined, while avoiding much of the combinatorial searching usually associated with a large domain.

We shall describe such a hybrid approach, in which a resolution-based theorem prover which operates on a semantic net is combined with several specialists (building on the ideas in [Schubert et al., 1987]). The specialists include a temporal inference specialist, a number/arithmetic specialist, a type hierarchy specialist, a set specialist and a very simple color specialist. Although hybrid approaches have been tried for these subdomains before, most use disjoint specialists, and do not systematically address the problem of communication among specialists.

The combined inference system developed here is intended for low level inferencing in a natural language understanding system (**ECoSystem** [de Haan and Schubert, 1986]) under development at the University of Alberta. The portion of **ECoSystem** presented here is called **ECoNet**. It accepts assertions in the form of first order predicate logic propositions, and answers questions phrased in the same form. The system is implemented in Lucid Common Lisp and runs on a Sun 3/75. A related paper [Miller and Schubert, 1988] contains details on the temporal specialist incorporated into this system.

2. Specialists

Before going any further, we should indicate what is meant by *specialists*, or special purpose inference mechanisms, and what can be gained by using these methods. A special inference method takes advantage of special properties of the predicates, terms and functions in the domain it works with, using efficient representations and methods for reasoning in that domain. Its reasoning steps may shortcut lengthy chains of standard inferences. For example, the temporal specialist uses a graph structure to represent times and temporal relations passed to it, and uses efficient graph algorithms to do its reasoning. Because lengthy chains of temporal connections may determine the relationship between two times, establishing such relationships by general methods can be computationally expensive.

Any inference made by a specialist must be sound, but there is no requirement that the specialist be complete, as the general method can fill in any inference gaps, albeit less efficiently. Also, since the specialists are to be used to accelerate the system, they must ALWAYS return an answer, and do so quickly (*unknown* is an acceptable answer).

Schubert et al. [Schubert et al., 1987] suggest several ways for a specialist to accelerate a theorem prover using derived rules of inference, including literal evaluation and generalized resolution and factoring. They discuss the relationship to Stickel's theory resolution [Stickel, 1983] in detail. In addition, a specialist can evaluate functional terms to simplify literals.

Literal evaluation uses a specialist's special representation to evaluate literals to *true* or *false*, and hence to simplify input clauses, and resolvents generated by the theorem prover. For example, if "*a* strictly before *b*" was represented in the temporal specialist's representation (the timegraph), it can be used to simplify literal *[a after b]* to *false*. Function evaluation simplifies a term by evaluating it (for example, *(max-of n1)* may be simplified to a number, say *3* by the number specialist).

Generalized resolving and factoring quickly determine incompatibility or subsumption of one literal by another. This allows resolution and factoring to be done where they usually cannot (in one step). For example, the type specialist can resolve *[d dog]* against ¬ *[x animal]*, to get the null clause, even though the two predicates are distinct. Similarly, the color specialist can resolve *[x blue]* against *[c red]* to get the null clause, even though the signs are the same. The type specialist can give us generalized factor *[d animal]* from literals *[d dog]* and *[x animal]*. Similarly, the temporal specialist can factor *[¬ [x during a] or [x during b]]* to *[a during b]* (if we have *[a during b]* represented in the timegraph). In addition, even though we may not have incompatibility, a specialist can determine the conditions for incompatibility and return these as a *residue*. A *residue* (from Stickel's partial theory resolution), is a literal or a set of literals whose negation would make the two literals being resolved incompatible (resolvable in one or more steps to the null clause). For example, if we resolve *[a before b]* against *[a after b]*, we get a residue of *[a equal b]*. If that residue is later determined to be false, we have the null clause.

As long as the operations a specialist is allowed to perform are equivalent to sets of standard deductive steps, the specialist is guaranteed to be logically sound. This restriction is satisfied by the operations we have implemented (literal evaluation, generalized resolving, etc).

3. Overview of the System

Having discussed how a specialist can assist a resolution-based theorem prover, we now need to consider the design of a general interface that will allow the theorem prover to invoke the specialists when appropriate. This interface should be general enough to handle any specialist we might imagine, and efficient enough that its cost is modest in comparison with the savings made possible by the specialists.

ECoNet's architecture is shown in Figure 1. The core of the system is a resolution-based theorem prover which has been under development at the University of Alberta for several years, most recently by de Haan [de Haan and Schubert, 1986]. The theorem prover uses a semantic net representation, and features automatic topical classification of entered clauses and organization in a topic hierarchy. The inference method used is resolution, enhanced by topical retrieval of clauses to resolve against the problem clauses. Inference is also accelerated by a concept hierarchy that enables type inheritance. Since the hierarchy specialist is used both for organizing knowledge in the theorem prover, and as a type specialist, it is shown inside the theorem prover box.

The most elaborate specialist in this system is the temporal specialist. The system is to be used for natural language understanding, which often deals with a number of temporally related events or episodes. Reasoning about these orderings, as well as the quantitative aspects of time (durations and dates) is required. As mentioned earlier, such inferences in a general theorem prover can be computationally expensive. Details of the temporal specialist can be found in [Miller and Schubert, 1988].

Temporal reasoning is certainly not the only domain requiring potentially explosive inferencing. Reasoning about numbers, and about set membership are also problems which we will have to deal with, even for understanding ordinary discourse. For example, if we are told that eleven of twelve jurors agreed on a guilty verdict, we should be able to figure out that exactly one juror demurred. Similarly, if we are told that John is a member of the curriculum committee, and all

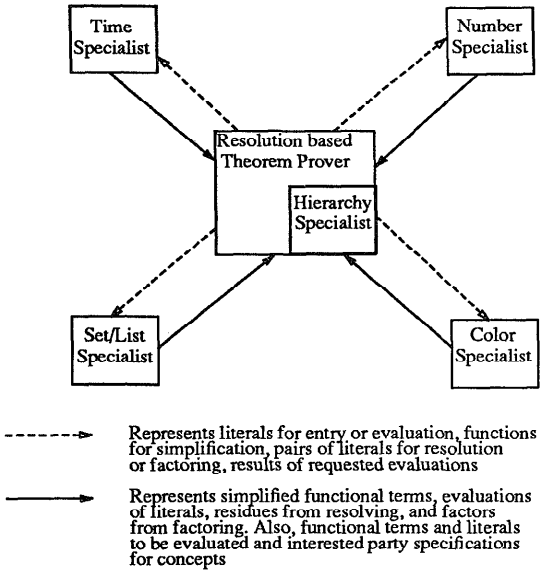

Figure 1. Architecture of ECoNet

members of the curriculum committee are members of the faculty council, we should be able to figure out that John is a member of the faculty council. To accelerate reasoning in these areas, the number/arithmetic specialist and the set specialist were incorporated. The number specialist uses a graph structure to represent and reason about orderings on integers and real numbers, and can evaluate numeric functions like *add*. The set specialist maintains the contents of sets, and can do simple operations on sets, like *union*, *intersection*, and testing set membership.

In addition, there is a very simple color specialist, which currently assumes that all color predicates are disjoint (e.g. *blue* and *red* are incompatible). A much better specialist based on a geometric three-dimensional color space has been designed [Schubert et al., 1987], which can handle subsumption of colors (e.g., *crimson* is subsumed by *red*), intermediate shades (e.g., *blue-green*), and some modified shades (e.g., *sort of brown*), and will eventually replace this one.

4. The Specialist Interface

There were several issues to consider in designing the interface. When and how does a specialist get invoked? How is the decision made that a particular specialist is likely to be helpful? How can useful information be transmitted between the specialists and the general theorem prover, and between specialists? To answer these questions, we first need to review how the general theorem prover works. Figure 2 shows a high level abstraction of the algorithm used by the general theorem prover, with notes in bold print showing where the various specialist operations described earlier fit.

An agenda is used to keep track of possible actions which can be used to carry out a proof - resolving actions and access actions. A successful resolving action causes the resolvent to be entered; a successful access action causes clauses to be retrieved which are likely to resolve against a particular clause. Agenda items are ordered so that the action most likely to result in success (i.e. the null clause) is at the top. When given a question, the theorem prover starts with the clauses corresponding to the question, and the clauses corresponding to the nega-

tion of the question, and enters both sets just like resolvents (and flags them appropriately). The classification phase of this entry will add the first access actions to the agenda. If a contradiction is eventually derived from the question clauses and the knowledge base, the answer will be NO (hence it is called the *disproof* attempt); with the negation of the question clause, the answer will be YES (the *proof* attempt). The *proof* and *disproof* attempts are carried out concurrently, using a common agenda.

Assertion
  Simplify asserted clause
    -> literal evaluation
    -> function evaluation
  Classify asserted clause
  Enter into Semantic net and concept and topic hierarchies
  -> entry into specialists' representations
Question
  Loop through agenda; take top item from agenda:
    If Resolving Action:
      Calculate resolvent
      Simplify resolvent
        -> literal evaluation
        -> function evaluation
      Classify resolvent and add relevant access actions to agenda
      Check for factoring possibilities
        -> generalized factoring
      Enter into semantic net and concept and topic hierarchies
    If Access action (involves *clause*, *topic*, and *concept*):
      Compare each clause retrieved under *topic* of *concept* with the given *clause*
        If resolvable, add a resolving action to agenda
        -> generalized resolving

Figure 2. Overview of Algorithm

During an access action where literals are being compared for resolvability, if the traditional test fails (same predicate, different signs), a specialist could potentially find a generalized resolving action. "Generalized resolutions" found by the specialists are added directly to the agenda. Similarly, if a factoring test fails (same predicate, same sign), a specialist might find a generalized factor.

Clause simplification involves both literal evaluation and function evaluation (a form of term simplification), both of which a specialist may assist with. Even if a literal or its negation have not been asserted before, a specialist may be able to detect its truth or falsity.

In all of these cases, a decision process must be invoked that can quickly decide which specialists, if any, apply, and invoke them.

4.1. Extensions to the Theorem Prover

To get maximum flexibility for the specialists and the specialist interface without sacrificing efficiency, some enhancements to the theorem prover were needed.

4.1.1. Expressiveness

First, some syntactic refinements were needed. To enable the specialist interface to decide when a specialist is appropriate for some problem, terms serving as predicate arguments should be sortally distinct, so that predications about physical objects, events, numbers, and so on, are easily distinguished from each other. (Predicates alone do not necessarily determine their argument sorts in our system. For example, the predicate *equal* may relate a number of different kinds of entities, including events or numbers.) This is accomplished by allowing a "sort tag" to accompany an argument, expressed by following the term with an underscore and the sort. Possible sorts include: *physical object*, *event/episode*, *time*, *number*, *symbolic expression* and *set*. Sorts are considered pairwise disjoint (in contrast with types, which in this system are unary predicates whose extensions may overlap).

Also, some entities, such as numbers, structured values, and symbolic expressions are not easily expressed as semantic net concepts, or are too numerous to represent that way. For example, we do not want an individual concept for every possible number! To avoid this problem, we allow *quoted expressions* as terms. For example, the functional expression *(date '1987 mm_number '1 '0 '0 '0)* evaluates to a quoted expression representing a structured value, *'(time 1987 mm 1 0 0 0)*, which is an absolute time representation recognized by the time specialist. A quoted atom is assumed to denote the string of characters making up the atom. For example, the denotation of *'Mary* is the string of alphabetic characters *"Mary"*, and the denotation of *'35* is the string of numerals *"35"*. By formally identifying the *natural numbers* with the strings of numerals normally used to represent them (in base 10), we ensure that quoted numerals are denotationally equivalent to unquoted ones[1]. The denotation of $'(t_1\ t_2\ \cdots\ t_n)$ is the tuple consisting of the denotation of t_1, followed by the denotation of t_2, ..., followed by the denotation of t_n. Thus, the above structured value is the 7-tuple whose first element is the string *"time"*, and whose remaining elements are numbers (the second one of these being whatever number is denoted by *mm*). Note that although this structured value looks very similar to the original functional term, if the terms for the function had been more complex (like *(add '1 '1986)* instead of *'1987*), the quoted expression would look considerably simpler than the original term. Term simplification is done bottom up, and guarantees to the specialists that when they are invoked with a literal or functional term, the terms have been simplified as much as possible. This allowed simpler and more elegant implementations of the interface and specialists.

4.1.2. Resolution

Because the "generalized resolutions" performed by specialists do not necessarily involve identical predicates, and may do the work of many ordinary resolution steps, the order of unification of arguments in two literals need not be the same. This is decided by the specialists, and details depend on the axiomatization (theory) assumed to underlie the specialist's domain. After the specialists have decided in what order the arguments should be unified when resolving or factoring, they invoke the unification process themselves. If a factoring or resolving action results, the specialist passes these substitutions back to the theorem prover.

Specialists may also return a residue (see example given earlier). The theorem prover has to incorporate this residue into the resolvent.

Also, when attempting to resolve, specialists may provide more than just substitutions and residues - they may also supply the evaluations of the two literals after substitution. Then, even if the two literals are not incompatible, we can use the evaluations to simplify the clauses. For example, when resolving $\neg$ *[x before a]* in *[$\neg$ [x before a] | [x during a]]* against a clause containing literal *[b after y]*, and if furthermore, we have "*a* strictly before *b*" represented in the timegraph, we can simplify the first clause to $\neg$ *[b before a]*. This simplification will then be added to the clauses to be considered.

[1] This enables us to confine numbers to quoted contexts, which turns out to be computationally convenient.

4.2. The Decision Process

This section briefly outlines how the specialists are selected for each operation they may perform. Details are in [Miller, 1988]. The mechanism for deciding which specialists should be called is just a cursory check to find specialists that are likely to be interested. Since much of the time no specialist will be involved, we do not want to waste too much time deciding which, if any, to call. To get quick and easy access to the specialists that may be useful, they are kept on the predicate's or function's property list.

4.2.1. Literal Entry and Evaluation

For literal entry and evaluation, the predicate involved, and its first argument are used to determine which specialist to call. The specialist itself is responsible for checking the other arguments to ensure that they are in its domain. Another possibility was to use pattern matching on the predicate-argument patterns, but this can be quite slow. On entry, all applicable specialists are called, as any one might later be required to use the information in inferencing. For literal evaluation, each specialist is called, one at a time, until an answer other than *unknown* is returned. For example, asserting *[n1_integer less-than r2_real]* would cause the number specialist to be invoked to enter it.

4.2.2. Generalized Resolution and Factoring

For generalized factoring and resolution, the two predicates alone are used to decide which specialists to call, using the intersection of the lists of interested specialists for each. No checking is done on arguments yet, as unification has not taken place (and cannot until the specialist decides how it is to be done), and substitutions may be involved. Final checks on whether the resulting literals (after unification and substitution) are in a specialist's domain must be done by the specialist. For example, when resolving

[t1_time before t2_time] vs [x equal t3_time],

both predicates have the temporal specialist on their property lists (although *equal* has other specialists as well), and so only that specialist would be called to try to resolve them. All applicable specialists are called for both generalized resolving and factoring, as each may find different resolving or factoring actions.

4.2.3. Function Evaluation

The function alone is used to determine which specialist(s) to call. All arguments are simplified before the function is evaluated, recursively. For example

(date (add '1987 '1) '4 '1 '12 '0 '0)

would first use the number specialist to calculate the year argument, and then the temporal specialist to calculate the absolute time with all the arguments.

5. Communication between Specialists

Sometimes a specialist may need additional information to complete its task, and it is possible that another specialist may be able to supply it. For example, assertion of

[e1_episode before (date '1987 mm_number '01 '12 '00 '00)]

would require that the time specialist ask about the bounds of *mm* (the territory of the number specialist). Similarly, a naive or qualitative physics specialist might need to know if the event of the robot hand going across the table happened before or after the vase of flowers was placed there in order to determine the current position of the vase. Also, when the information requested is not immediately available, or likely to be further constrained by clauses added later (for example, bounds on numbers), we want the specialist to be notified if and when it is.

The essential idea of our approach is to channel all communication between specialists through the interface. Thus specialists need not know which other specialists can help them. Two types of communication have been implemented for the specialists: *immediate evaluation*, where a specialist asks for the evaluation of a particular functional term or literal, and *delayed communication*, where the specialist is notified that something of interest in another domain has been asserted. Immediate evaluation may be useful during either assertion or question answering, while delayed communication can only be used to further enhance asserted information.

5.1. Immediate Evaluation

For immediate evaluation, the functional term or literal is sent out from the specialist to the specialist interface for evaluation. The interface decides which specialist(s) will be able to evaluate it as described earlier, and passes the item on. The result (a concept or quoted expression for a functional term; *yes*, *no*, or *unknown* for a literal), is sent back to the specialist that requested it. In the previous example, the temporal specialist could request the value of *(max-of mm)*[2], and use that value in the absolute time specification.

5.2. Delayed Communication

A specialist can communicate to the interface a particular concept (currently only constants) it wants "watched", and a literal to reassert when something involving that concept is asserted. This information is kept on an *interested party list* for the concept. Whenever anything new is asserted about that concept, each literal on the *interested party list* is re-asserted to the specialist that put it there. There is no point asserting the new literal to an interested specialists because it may not recognize the predicates and terms involved. In the previous example, the temporal specialist would add an entry to *mm*'s interested party list, consisting of the specialist (*time-specialist*) and the literal given to the temporal specialist (after term evaluation) *[e1 before '(time 1987 mm 1 12 0 0)]*. Later, if *[mm less-than '3]* were asserted, *[e1 before '(time 1987 mm 1 12 0 0)]* would be re-asserted to the temporal specialist.

6. Example

The system can answer questions which involve mixed reasoning - both specialist inference and ordinary resolution. The example in Figure 3 from the story of Little Red Riding Hood shows a few capabilities of the system. The example does not show either the number or set specialists, as examples involving them were too lengthy to include. Currently the number specialist is used mainly to support the temporal specialist by assisting it in maintaining the most constrained absolute times and durations. The set specialist is quite new and its capabilities have not been fully exploited in our Little Red Riding Hood knowledge base yet.

7. Comparison with Other Approaches

ECoNet's representational capabilities allow it to handle first order predicate calculus, minus equality. (The specialists provide partial handling of equality, but full equality reasoning should be added to the general theorem prover). The specialists, in principle, are used only to accelerate inference. This differentiates it from systems like KL-TWO [Vilain, 1985], where the core of the system is a computationally efficient subset of first order logic, and the specialist (a terminological inference mechanism) adds both to the overall expressiveness and reasoning power of the system.

[2] We always want the most complete, but correct information available. Since *mm* can never get bigger than its maximum, this is the safest bound to use on assertion. However, if we were trying to evaluate this literal, we would use the minimum, since if *e1* were less than the minimum of that time, we can be assured that it will be less than any other value that time might have.

```
; did a creature kill the wolf while he was in the cottage?
==> ?(E y E z_ep (y kill W z) & (y creature) & (z during wolf-in-cot))
Entering proof clauses:
((~ U-VAR-1 CREATURE) | (~ U-VAR-1 KILL W EPISODE-VAR-1)
 | (~ EPISODE-VAR-1 DURING WOLF-IN-COT))
   Using the temporal specialist:
  Time Specialist: Trying to resolve
    (~ EPISODE-VAR-1 DURING WOLF-IN-COT)
    against (WOLF-DEMISE DURING-1-0 WOLF-IN-COT)
  Time Specialist: Resolved with evaluation: NO with substitutions:
    ((EPISODE-VAR-1 by WOLF-DEMISE 2))
  Resolved (~ EPISODE-VAR-1 DURING WOLF-IN-COT) in the proof clause
    ((~ U-VAR-1 CREATURE) | (~ U-VAR-1 KILL W EPISODE-VAR-1) |
     (~ EPISODE-VAR-1 DURING WOLF-IN-COT))
    against (WOLF-DEMISE DURING-1-0 WOLF-IN-COT) in
        (WOLF-DEMISE DURING-1-0 WOLF-IN-COT)
    yielding ...
((~ U-VAR-1 CREATURE) | (~ U-VAR-1 KILL W WOLF-DEMISE))
   Ordinary resolution:
  Resolved (~ U-VAR-1 KILL W WOLF-DEMISE) in the proof clause
    ((~ U-VAR-1 CREATURE) | (~ U-VAR-1 KILL W WOLF-DEMISE))
    against (WOODCUTTER KILL W WOLF-DEMISE) in
        (WOODCUTTER KILL W WOLF-DEMISE)
    yielding ...
(~ WOODCUTTER CREATURE)
   Using the type hierarchy specialist:
  Resolved (~ WOODCUTTER CREATURE) in the proof clause
        (~ WOODCUTTER CREATURE)
    against (WOODCUTTER MAN) in (WOODCUTTER MAN)
    yielding the null clause.
YES
; Were all the capes blue?
==> ?(A x (x cape) => (x blue))
Entering disproof clauses:
((~ U-VAR-1 CAPE) | (U-VAR-1 BLUE))
   Using the color specialist:
  Resolved (U-VAR-1 BLUE) in the disproof clause
    ((~ U-VAR-1 CAPE) | (U-VAR-1 BLUE))
    against (SCON-340 RED) in (SCON-340 RED)
    yielding ...
(~ SCON-340 CAPE)
  (SCON-340 CAPE) evaluated to YES.
  The clause evaluated to NO.
NO
```

Figure 3. Example of ECoNet in operation[3]

Another hybrid, Krypton [Brachman et al., 1985] has a powerful core system, also enhanced with a terminological specialist. It is unclear whether the technique used to integrate this specialist could be used for other specialists as well (like time). Krypton is one of the few other systems that actually allow the specialists to do generalized resolving. However, the residues calculated by Krypton can be quite complex, involving quantification. Although ECoNet has no constraint on the complexity of residues that a specialist may create, the specialists implemented so far generate quite simple residues, further simplifying the resolving process.

The domain independent special mechanisms used by Bundy et al. [Bundy et al., 1982] allow for more than one specialist, but all must be in the form of specialized logical rules (precluding more efficient representations, like graphs). The only limitations on the specialists that may be incorporated into ECoNet are that they must make sound inferences.

Another thing that makes ECoNet unique is the communication among specialists. Although some earlier systems did have some forms of communication (for example, Nelson and Oppen [Nelson and Oppen, 1979], and Shostak [Shostak, 1984]), only equalities could be communicated. ECoNet is much more flexible, and unlike these systems, does not require that the specialists be disjoint (no overlapping functions or predicates).

[3] This example shows actual output of the system, edited for clarity and brevity. Bold print is user input; comments in italics; the rest, system output. The timegraph contained, among other relations, that *wolf-demise* (episode corresponding to the wolf being killed) is during *wolf-in-cot* (episode corresponding to the wolf being in the cottage). *[a during-1-0 b] => [(start-of a) > (start-of b)] & [(end-of a) = (end-of b)].*

8. Conclusions

Our intent was to combine efficient special methods with a general purpose theorem prover in such a way that the efficiency of the special methods averts the combinatorial explosions usually associated with a general theorem prover. Previous approaches limit the overall domain, or cannot easily accomodate a variety of specialists, or don't fully exploit the specialists' capabilities.

The interface we developed allows specialists to accelerate the resolution-based theorem prover in literal evaluation, function evaluation, and generalized resolution and factoring. In addition, assertions are given to the specialists to store in their own representations. Several specialists were incorporated into the system using the interface, including a type specialist, a temporal specialist. a number/arithmetic specialist, a set/list specialist, and a very simple color specialist. Communication between specialists is achieved by allowing specialists to request evaluation of functional terms or literals, and by maintaining *interested party lists* to notify specialists that something of interest to them has been derived. The temporal and number specialists communicate through absolute time and duration specifications.

Experience with the interface confirms that new specialists can be added with relative ease. This is because the specialists interact with the general theorem prover in a small, fixed set of ways. Each new specialist was found to make possible fast proofs of questions previously beyond the scope of the theorem prover.

References

[Brachman et al., 1985] Ronald J. Brachman, Victoria Pigman Gilbert and Hector J. Levesque, An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON, *Proceedings of IJCAI-85 I*, (1985), 532-539.

[Bundy et al., 1982] Alan Bundy, Lawrence Byrd and Chris Mellish, Special Purpose, but Domain Independent, Inference Mechanisms, *Proc. ECAI-82*, Orsay, France, 1982, 67-74.

[de Haan and Schubert, 1986] Johannes de Haan and Lenhart K. Schubert, Inference in a Topically Organized Semantic Net, *Proc. AAAI-86 I*, (1986), 334-338.

[Miller and Schubert, 1988] Stephanie A. Miller and Lenhart K. Schubert, Time Revisited, *CSCSI-88*, Edmonton, Alberta, 1988.

[Miller, 1988] Stephanie A. Miller, Time Revisited, M.Sc. Thesis, Department of Computing Science, University of Alberta, 1988.

[Nelson and Oppen, 1979] Greg Nelson and Derek C. Oppen, Simplification by Cooperating Decision Procedures, *ACM Transactions on Programming Languages and Systems 1*, 2 (1979), 245-257.

[Schubert et al., 1987] L.K. Schubert, M.A. Papalaskaris and J. Taugher, Accelerating Deductive Inference: Special Methods for Taxonomies, Colours and Times, in *The Knowledge Frontier*, N. Cercone and G. McCalla (ed.), 1987.

[Shostak, 1984] Robert E. Shostak, Deciding Combinations of Theories, *Journal of the ACM 31*, 1 (1984), 1-12.

[Stickel, 1983] Mark E. Stickel, Theory Resolution: Building in Nonequational Theories, *Proc. AAAI-83*, Washington, D.C., 1983, 391-397.

[Vilain, 1985] Marc Vilain, The Restricted Language Architecture of a Hybrid Representation System, *Proceedings of IJCAI-85 I*, (1985), 547-551.