Reasoning at Multiple Levels of Abstraction

Robert Veroff

University of New Mexico, U.S.A.
veroff@cs.unm.edu

Abstract. In order to be successful in complex domains involving hierarchies of defined terms, an automated reasoning program must be able to reason effectively at multiple levels of abstraction. At a minimum, this requires appropriate problem representations and good search strategies. Ideally, the reasoning program reasons at high levels of abstraction when possible and appeals to arguments at lower levels of abstraction only as necessary. In this article, we describe our early experiences developing representations and search strategies for an application of automated reasoning to a problem domain from theoretical computer science. We then summarize some of the approaches we are considering to permit the automated reasoning program to reason effectively at multiple levels of abstraction.

1 Introduction

Automated reasoning has been applied successfully to a number of problems from mathematics and formal logic—including open questions—and to a variety of design and verification problems. In this article, we consider some practical issues concerning the *effective* use of automated reasoning programs. Although there is much interest in developing systems based on more expressive higher-order logics, especially for applications to hardware and software verification, we restrict ourselves in this article to first-order refutation systems—specifically, to resolution-style systems using the language of clauses.

Most of our experience with automated reasoning programs has been with relatively small axiom systems having few distinct predicate and function symbols. As we challenge our systems to work on more complex problems, we will necessarily have to develop more sophisticated techniques and features than those currently available. For example, as axiom systems get larger and more complex, there often will be a natural hierarchy of defined terms. Many proofs will contain distinct subproofs at the various levels of abstraction defined by this hierarchy. Automated reasoning programs typically have not been good at pursuing multiple lines of reasoning simultaneously; mixing arguments at different levels of abstraction—similar to mixing cases of a case analysis—is not conducive to effective reasoning [25]. Until our automated reasoning programs can reason effectively at multiple levels of abstraction, we will be severely limited in the kinds of problems we can seriously consider as applications [12]. In this article, we consider this aspect of effective reasoning and describe a few first steps towards a possible solution.

Many people have studied and written about abstraction and related themes such as multi-level reasoning, meta reasoning, and analogy. In the area of automated reasoning, the work includes formal theories of abstraction, the use of abstraction in the formal development of theories, and the use of abstraction for proof planning. Plaisted [17], for example, considers various abstraction mappings for sets of clauses and shows how a resolution proof of an abstracted problem relates to a resolution proof of the original. Giunchiglia and Walsh [8] present an extensive formal theory of abstraction. Similar to Plaisted's work, the primary application to automated reasoning is to transform a problem with an abstraction mapping, solve the transformed problem, and then use the solution to the transformed problem to help solve the original problem.

Barker-Plummer [1] uses abstraction as part of a planning process. In this work, an inference rule called *gazing* is used to apply a sequence of rewrite steps to a goal sequent. The sequence is determined by the *gazing planner*, which systematically refines a candidate sequence at successively lower levels of abstraction. At each level of abstraction, the candidate sequence is evaluated and refined according to information available at the current level. In earlier work, Sacerdoti [20] uses a similar approach—refining a plan through an abstraction hierarchy—for the general planning problem.

In all of the cited work, abstraction refers to a mapping from an original problem space to an abstraction space. Proof planning is done by considering the (simpler) transformed problem. In this article, abstraction refers solely to relationships between the terms, literals, and clauses appearing in an input clause set—for example, the relationship between primitives and defined terms. Our objective is to adapt resolution-style theorem provers to take advantage of these abstraction relationships in order to restrict and direct the applications of inference rules (that is, without applying abstraction transformations). As in [1], one of the specific objectives is to manage a potentially large space of definitions and previously proved theorems.

In the remainder of the paper, we explore the issue of effective reasoning in more detail, emphasizing the need for reasoning at multiple levels of abstraction. In Section 2, we briefly describe an application area from theoretical computer science. In addition to having a rich hierarchy of defined terms, some aspects of the theory are set theoretic in nature, which further illustrates the importance of abstraction hierarchies. In Sections 3, 4, and 5, we describe approaches, currently being explored, for giving the automated reasoning program the ability to reason effectively at multiple levels of abstraction. These include a relevance strategy, the use of linked inference rules, and the use of distributed reasoning. In Section 6, we summarize our plans for future work.

2 An Application from Theoretical Computer Science

In this section, we describe our first experiences with the application of automated reasoning to a problem area coming from theoretical computer science. In [10], Helman presents an algebraic model for studying combinatorial optimization algorithm paradigms such as dynamic programming and branch and bound. Our short-term objective is to use an automated reasoning program to prove some of the basic properties of Helman's theory. Our longer-term objective, which may be quite ambitious, is to develop a methodology for using automated reasoning to establish complexity results, including lower bounds.

The formal model includes the following key elements.

- A set $ATOMS$ of *atoms* representing the base objects of interest for a particular problem type. For example, the atoms may represent vertices of a graph for a path problem or matrices for a matrix multiplication problem.
- A set $CONJ$ of *conjuncts* representing possible solutions and partial solutions for a problem type. Conjuncts are defined as expressions in terms of atoms and applications of a binary operator $\odot$. An application of $\odot$ represents the concatenation of two partial solutions. For example, a conjunct may represent a path in a graph or a specific association for a matrix product $A_1 \times A_2 \times A_3 \times \ldots \times A_n$.
- A congruence relation $\cong$ on $CONJ$ that partitions the possible solutions and partial solutions into sets considered to be indistinguishable for a given problem type. For example, the associative variants of a $CONJ$ expression are considered to be indistinguishable for many path problems. In contrast, associative variants are necessarily distinguishable for a matrix multiplication problem in which the objective is to find the association that minimizes the total number of scalar multiplications.
- An extension relation EXT on $CONJ$ that indicates when one partial solution can be "grown" into another by applications of $\odot$.[1] The extension relation is strictly syntactic in that it does not take congruence into account. For example, the path $(a \odot b) \odot c$ is an extension of the path $a \odot b$, but $a \odot (b \odot c)$ is not, even though the two longer paths may be congruent to each other. Similarly, congruent conjuncts do not, in general, have the same extensions.
- A distinguished subset Q of $CONJ$ of complete solutions. For example, if the problem type is to find a least-cost tour in a complete graph, Q may consist of all possible permutations of the vertices of a graph.
- A total order α on the congruence classes of Q. Two conjuncts in Q are *not* ordered by α if and only if they are congruent.

If the total order α is determined by a cost model for the relevant problem type, then the α-minimum congruence class of Q represents the optimal solution to the original problem.

A key feature of Helman's theory is the extension of the α relation to a partial order on *all* of $CONJ$ as follows.

$conj_1 <_\alpha conj_2$ if and only if there exists an extension of $conj_1$ to Q that is $<_\alpha$ every extension of $conj_2$ to Q.

[1] In the clause representation, a conjunct is a subterm of its extensions.

Given this ordering, we can consider the following general procedure for finding the α-minimum congruence class of Q.

```
while ( more than one congruence class remains in Q )
{
    choose c1 and c2 in CONJ                                  (1)
    if (c1 <α c2)                                             (2)
    {
        eliminate from Q any congruence classes               (3)
           containing an extension of c2
    }
}
```

As a practical consideration, it may be prohibitively expensive to determine the α-ordering of two conjuncts. Formally, this is accounted for by defining a "computationally feasible" subset of α. Informally, it suffices to simply consider restrictions on the choices of c_1 and c_2. The difference between specific algorithms then is the implementation of computations (1), (2) and (3) in the general algorithm, and the difference between various algorithm paradigms, such as branch-and-bound and dynamic programming, can be described in terms of general properties of these implementations. A branch-and-bound algorithm, for example, compares a full solution c_1 to a partial solution c_2. If the cost of c_1 is determined to be less than the cost of c_2, then c_2 and its extensions are eliminated from future consideration. A dynamic programming algorithm, on the other hand, compares a partial solution—considered to be a full solution to a subproblem of the original—to another full solution to the *same* subproblem. Dynamic programming algorithms are appropriate for problems to which Bellman's principle of optimality [2] applies.

Automated Reasoning Considerations

We initially used the automated reasoning program OTTER [15] to prove elementary properties of $CONJ$, $\cong$, EXT, and α relations using a simple and limited representation for the required axioms. As we started to move on to some of the first significant results of the theory, we quickly "outgrew" the representation we were using and needed to consider a representation with more generality (for example, accounting for multiple relations rather than a single relation). Because several of the results of interest appear to rely on set theoretic arguments, there is at least some motivation to recast the entire representation in terms of set theory—similar to what was done for group theory in [4]. Although there is theoretical interest in this approach, we do not consider this to be realistic for effective reasoning. Instead, we have adopted, at least for now, the evolutionary approach of adapting the representation as necessary as the theory evolves. The following example illustrates our current representation strategy.

Example 1. A Representation Strategy for α Relations

Following the presentation of [10], we define α relations and their extension to all of $CONJ$ in terms of three basic properties,

$$ALPHA(x) \leftrightarrow (PROPi(x) \wedge PROPii(x) \wedge PROPiii(x)),$$

where $ALPHA(x)$ denotes that x is an α relation. Our current representation relies in addition on the following predicate symbols.

$CONJ(x)$	– x is a conjunct.
$REL(x,y,z)$	– y and z are related by relation x.
$CONGR(x,y)$	– x is congruent to y.
$EXT(x,y)$	– x is an extension of y.
$Q(x)$	– x is in set Q.

The third property for α relations, for example, is expressed with the formula

```
(∀ xa (   PROPiii(xa)
          ↔
          (∀ x, y (   (CONJ(x) ∧ CONJ(y))
                      →
                      (   REL(xa,x,y)
                          ↔
                          (∃ xq1 (   Q(xq1)
                                     ∧ EXT(xq1,x)
                                     ∧ (∀ xq2 (   (Q(xq2) ∧ EXT(xq2,y))
                                                  →
                                                  REL(xa,xq1,xq2)))
                          ))
                      )
          ))
)).
```

Mapping this formula to the language of clauses [5] results in fourteen clauses with six new Skolem functions. See [11] for a more complete description.

Although the current representation strategy suffices for the problems considered to date, we will soon “outgrow” this representation as well. We will need to represent new concepts as they get introduced into the theory; we will need to refine some of the concepts already present; and we will frequently need to appeal to set-theoretic arguments to complete proofs. It is our contention that rather than making the great leap all the way to set theory, the best solution will be to develop a representation strategy and appropriate enhancements to the automated reasoning program to support reasoning at multiple levels of abstraction. In this case, the objective is to permit the automated reasoning program to reason about higher-level constructs when possible, and to appeal to lower-level constructs as needed.

A Diversion into Set Theory

In order to be able to reason effectively at multiple levels of abstractions, it will be necessary at least to be able to reason effectively solely at the lower levels of abstraction. Since set theory can be considered to be the foundation of all problems of interest, and many proofs make use of set-theoretic arguments of some form, it seems reasonable to consider briefly the problem of reasoning effectively about sets.

In [4], Boyer, et al present a finite set of clauses for the von Neumann-Bernays-Gödel version of set theory.[2] In addition to describing the underlying axiom system and the representation choices made for the resulting set of clauses, the authors illustrate the expressive power of the system by providing numerous examples of problem statements and proofs. They also provide a set of challenge problems and discuss the difficulty of proving theorems in the theory using an automated reasoning program.

Proving theorems in this system is especially difficult for an automated reasoning program. The large number of predicate and function symbols—73 distinct symbols in the basic axiom set alone—and the very large number of potential resolution and paramodulation steps between the clauses quickly leads to a proliferation of inferred clauses, even with careful use of the set of support strategy.

3 A Relevance Strategy

Working a number of problems both in the algebraic model and in the underlying set theory, it becomes apparent that several of the function and predicate symbols often have very little to do with a given problem. In set theory, for example, it is unlikely—but not impossible—that the symbols directly associated with the concept of a function will be relevant to proving a general property about a simple operation such as set intersection. This observation motivates a type of "relevance" strategy.

Consider the graph that represents the potential resolution and paramodulation steps between the clauses in the input set for a given problem. The vertices of the graph are clauses; there is a *type 1* edge between two clauses if and only if the two clauses can resolve, and there exists a *type 2* edge between two clauses if and only if one of the two clauses can be used to paramodulate into the other. We then can define a number of relevance measures for symbols by considering the paths in this graph. For each symbol S,

$any\_dist(S)$ = the length of the shortest path from a clause containing the symbol S to a clause in the set of support;

$res\_dist(S)$ = the length of the shortest path, consisting of type 1 edges only, from a clause containing the symbol S to a clause in the set of support;

[2] See also [18].

$para\_dist(S)$ = the length of the shortest path, consisting of type 2 edges only, from a clause containing the symbol S to a clause in the set of support.

Finally, we can define a weighting function [14] in terms of these relevance measures.

$$weight(S) = w_1 * any\_dist(S) + w_2 * res\_dist(S) + w_3 * para\_dist(S)$$

Increasing the values of the w_i coefficients tends to bias the search to prefer symbols that are more likely to be relevant to the problem at hand. [3]

Using this relevance strategy, we have used OTTER to prove several theorems about Helman's algebraic model as well as in the underlying set theory.

Example 2. A Property of α Relations

Theorem. Let $<$ be an α relation. Then for all x, y, u, and v,

$$((\mathrm{x} \cong \mathrm{u}) \wedge (\mathrm{y} \cong \mathrm{v}) \wedge (\mathrm{x} < \mathrm{y})) \Longrightarrow (\mathrm{u} < \mathrm{v}).$$

Using the relevance strategy, OTTER finds a proof of this theorem consisting of 28 hyperresolution and UR-resolution steps.

Example 3. Composition of Homomorphisms is a Homomorphism

Theorem. $(\forall x_i((HOM(x_0, x_2, x_3, x_4, x_5) \wedge HOM(x_1, x_4, x_5, x_6, x_7)) \rightarrow HOM(composition(x_1, x_0), x_2, x_3, x_6, x_7)))$

This is one of the challenge problems given in [4]. Although the relevance strategy may be a good starting point for developing a strategy for a given problem, and it does indeed suffice for several problems, it is not realistic to expect it, by itself, to suffice for more difficult problems. Two special considerations allow OTTER to find a proof of the composition of homomorphisms theorem.

1. It is apparent from the first experiments that the terms
   `composition(cx1,cx0)`,
   `f32(composition(cx1,cx0),cx2,cx3,cx6,cx7)`, and
   `f33(composition(cx1,cx0),cx2,cx3,cx6,cx7)`
   are especially relevant to this problem. We promote the use of these three terms in the search by using OTTER weighting patterns to assign each of them a low weight.
2. Terms in the predicate `M` quickly proliferate and dominate the search. We eliminate this distraction by assigning predicate `M` a high weight.

The proof consists of 24 hyper-resolution, UR-resolution, and paramodulation steps.

[3] A similar notion is described in [6]. In this case, the complexity of a sequent is defined as a function of the rewrite distance between the predicate symbols in the hypothesis and conclusion of the sequent.

4 Linked Inference Rules

In this section, we consider the application of linked inference rules to the problem of reasoning at multiple levels of abstraction. The basic idea is to use linking to permit the automated reasoning program to focus on higher-level terms by avoiding the explicit consideration of lower-level terms.

Inference rules are used to deduce new information from a database of facts. One objective of the study of inference rules is to develop new rules that take "larger" steps that are better directed. Inference rules such as hyperresolution [19] and UR-resolution [14] take larger steps than binary resolution, but even these steps are still relatively small, and they are defined solely in terms of syntactic criteria. Our research has focused on a class of inference rules, called *linked inference rules* [23, 24], that can take arbitrarily large steps and allow steps to be chosen based on semantic rather than syntactic criteria.

Our work in linking has been motivated, in part, by the observation that traditional resolution-style reasoning programs do not distinguish, in a strategic way, different applications of inference rules. Even when there is a good ordering of the clauses to which the inference rules apply, the individual applications of inference rules tend to be governed by a single uniform strategy. Our contention is that many low-level deductions, although technically required in a proof, are a significant distraction to the overall proof strategy. Linking provides a way to combine several low-level steps into a single higher-level step, avoiding the explicit consideration of uninteresting intermediate results, and allowing the proof search strategy to focus on the most interesting and relevant information.

Linked inference rules are naturally applied to the problem of reasoning at multiple levels of abstraction. The basic idea is to restrict low-level deductions to the implicit intermediate steps that are not kept, permitting the main line of reasoning to remain at the higher levels of abstraction. Linked inference rules begin to address, for example, the difficult problem of reasoning about defined objects—reasoning about the defined object when possible, but expanding the definition in terms of lower-level properties as necessary.

Example 4. Consider the following example from naive set theory. The theorem to prove is $(\forall x, y(x \cup (x \cap y) = x))$.

```
% relevant axioms for union/intersection
-EL(x,Union(y,z)) | EL(x,y) | EL(x,z).
-EL(x,Inter(y,z)) | EL(x,y).

% define subset in terms of element
-SUBSET(x,y) | -EL(z,x) | EL(z,y).
SUBSET(x,y) | EL(f(x,y),x).
SUBSET(x,y) | -EL(f(x,y),y).

% selected lemmas
-SUBSET(x,y) | -SUBSET(y,x) | EQSET(x,y).
SUBSET(x,Union(x,y)).
```

```
% negation of theorem
-EQSET(Union(A,Inter(A,B)),A).
```

Using ordinary UR-resolution, we can obtain the following proof.

```
1 [] SUBSET(x,Union(x,y)).
2 [] -EL(x,Union(y,z)) | EL(x,y) | EL(x,z).
3 [] -EL(x,Inter(y,z)) | EL(x,y).
4 [] SUBSET(x,y) | EL(f(x,y),x).
5 [] SUBSET(x,y) | -EL(f(x,y),y).
6 [] -SUBSET(x,y) | -SUBSET(y,x)| EQSET(x,y).
7 [] -EQSET(Union(A,Inter(A,B)),A).
8 [ur,7,6,1] -SUBSET(Union(A,Inter(A,B)),A).
9 [ur,8,5] -EL(f(Union(A,Inter(A,B)),A),A).
10 [ur,8,4] EL(f(Union(A,Inter(A,B)),A),Union(A,Inter(A,B))).
11 [ur,9,3] -EL(f(Union(A,Inter(A,B)),A),Inter(A,x)).
15 [ur,11,2,9] -EL(f(Union(A,Inter(A,B)),A),Union(A,inter(A,x))).
16 [binary,15.1,10.1] $F.
```

Alternatively, we can use linked UR-resolution to take advantage of the abstraction relationships in the problem. In the following proof, the main line of reasoning is in terms of the SUBSET and EQSET relations; reasoning about the lower-level EL relation is done implicitly.

```
1 [] SUBSET(x,Union(x,y)).
2 [] -EL(x,Union(y,z)) | EL(x,y) | EL(x,z).
3 [] -EL(x,Inter(y,z)) | EL(x,y).
4 [] SUBSET(x,y) | EL(f(x,y),x).
5 [] SUBSET(x,y) | -EL(f(x,y),y).
6 [] -SUBSET(x,y) | -SUBSET(y,x) | EQSET(x,y).
7 [] -EQSET(Union(A,Inter(A,B)),A).
8 [linked_ur,7,6,1]
        -SUBSET(Union(A,Inter(A,B)),A).
9 [linked_ur,8,5,3,2,4,5,8]
         SUBSET(Union(A,Inter(A,B)),A).
10 [binary,9.1,8.1] $F.
```

Clause 8 happens to be an application of ordinary UR-resolution. In Clause 9, the resulting literal is an instance of the SUBSET literal in Clause 4. Clauses 2 and 3 and both occurrences of Clause 5 are linking clauses. Both occurrences of Clause 8 are unit satellites.

Example 5. In Example 2, we presented a theorem for which OTTER finds a proof, using the relevance strategy, consisting of 28 hyperresolution and UR-resolution steps. The steps of this proof are unit clauses in the predicates CONJ, REL, CONGR, EXT, and Q. For the sake of illustration, we can consider CONJ and CONGR to represent relatively low-level concepts and REL, EXT, and Q to represent higher-level concepts. Using the target strategy to designate these levels of abstraction, OTTER finds a proof strictly in terms of the higher-level predicates; the lower-level predicates appear in intermediate steps only.

Our current work extends the linked inference principle to include both standard and linked versions of inference rules for equality-oriented reasoning. Inference rules based on the extended theory permit chains of equality substitutions into a term and its descendants to be performed as a single reasoning step. In addition, the extension captures the type of reasoning that combines that found in resolution-oriented rules with that found in equality-oriented rules. One especially appealing consequence of this combination is that it provides a way to put conditions on the application of equality substitutions.

One problem with linking and effective reasoning is that the application of linked inference rules is computationally expensive. For example, consider a single linking clause,

```
-L1 | -L2 | L3,
```

n positive unit clauses `U1`, `U2`, ... `Un`, and the application of linked resolution to a single literal `L`. The number of possible derivations for this simple application is defined by the recurrence

$$Count(0) = n$$
$$Count(h) = Count(h-1)^2 + n,$$

where h denotes the height of the corresponding derivation tree. The cost of applying linked inference rules can be prohibitive for large h.[4] The effectiveness of linking, then, may be limited to fairly "shallow" levels of abstraction with fairly simple theories at the lower levels of abstraction. We continue to develop and experiment with ways to limit the intermediate processing of applications of linked inference rules without sacrificing the desirable properties.

5 Distributed Theorem Proving

In this section, we briefly consider a distributed approach to reasoning at multiple levels of abstraction. The basic idea is to use multiple, communicating reasoning components, each restricted to reason at specified levels of abstraction, thereby avoiding the confusion and distraction of mixing levels of reasoning.

The idea of taking advantage of multiple processors for theorem proving is not new, of course. Approaches range from parallelizing low-level procedures such as unification (as in [21], for example), to dedicating processors to logical operations such as subsumption or demodulation (as in [9] and [13], for example), to having a network of independent reasoning components that communicate with each other (as in [7] and [22], for example). While parallelizing specific procedures and logical operations can be thought of as addressing execution-time performance, distributing reasoning components can be thought of as addressing search performance. For example, separation of clauses between processors

[4] For this reason, there is a user settable limit on h, the `max_ur_depth` parameter, in our current implementation of linked UR-resolution.

prevents their interaction during the application of inference rules and other other logical operations such as subsumption and demodulation. In this sense, distributed reasoning implements a type of restriction strategy. Since different reasoning components may use different inference rules and may employ different search, demodulation, and deletion criteria, distributed reasoning also implements a type of search strategy.

We currently are developing AR-NET, a network of reasoning components that can collaborate on a single problem. There are at least two different models for collaboration between AR-NET reasoning components. Under the first model, all of the components work on the same problem (or a predetermined part of the same problem) and tell others about particularly interesting intermediate results. Under the second model, one component asks another component to focus on a specific subproblem that the first process has identified as being significant. The subproblem could be, for example, a lemma to prove or one case of a case analysis.

We have an operational prototype of AR-NET that works under the first model for collaboration. This first prototype, based on an extended version of OTTER, is homogeneous in the sense that it consists of multiple copies of *the same* automated reasoning program, each working with different processing parameters and/or input clauses.[5] The communications architecture is a user-specified tree of AR-NET components; each component communicates with adjacent components using messages. The current criteria for sending out "interesting" results is based on OTTER's weighting mechanism and a few special-purpose OTTER flags and parameters. The AR-NET components can run either synchronously or asynchronously.

When an AR-NET component derives the empty clause, it displays a proof in terms of the deductions it has made and the lemmas it has received from other components. The other components are sent messages requesting them to display the derivations of the lemmas they have contributed to the proof.

Initial experimental results with the first prototype are promising; we have solved a number of problems in mathematics and logic, including, for example, problems from associative rings, cancellative semigroups, and some of the early results about Robbins algebra [16]. In many cases, the primary proof is presented in terms of lemmas that are meaningful (to us). In just as many cases, however, the split between primary proof and lemmas appears more arbitrary. This isn't necessarily bad, but it is something that we would like to explore further.

We are planning, and have started implementing, a heterogeneous version of AR-NET that will include more than one type of reasoning program. Work on the desired heterogeneous system will require us to support a common language for exchanging information. We also are developing support for the second collaboration model (subtasking).

A distributed approach is well suited to the problem of reasoning at multiple levels of abstraction. Each component is designed to work at some specified level and to communicate with the other components as desired or necessary.

[5] This type of architecture is called "pseudo-heterogeneous" in [22].

The level of abstraction of a component is determined by its input clauses, by the weighting strategy it employs, and by the types of information it receives from the other components. The intention is that whenever low-level processes deduce "interesting" clauses in terms of higher-level concepts, the clauses are sent to the appropriate higher-level components. Similarly, higher-level components send information to the lower-level components for more detailed consideration.

The levels of abstraction of the terms, literals, and clauses of a problem can be derived formally for some theories—consider, for example, the Boyer-Moore logic [3], and see some of the discussion about abstraction hierarchies in [1])—or can be defined according to user preferences.

In our first implementation of AR-NET, the search and communication functions of a component relied on the same weighting mechanism. That is, the communication model was based on sending *what is interesting to me*. Although this model is suitable for our more general notion of distributed reasoning, it turns out to be problematic from the perspective of reasoning at multiple levels of abstraction. In particular, there was nothing to prevent the intended lower-level component from deducing and reasoning about higher-level constructs. Again, this isn't necessarily bad, but it may not lead to a clean separation of arguments at the different levels of abstraction. In the limiting cases, the entire proof may be derived by a single component, or the primary proof may consist entirely of two conflicting lemmas, with both lemmas proved by another reasoning component.

In our current implementation of AR-NET, a distinction is made between the search and communication aspects of the reasoning components. The user can choose the original communication model, a model based on sending *what is interesting to you*, or something in between. The user now has the flexibility to prevent the mixing of the levels of abstraction, for example, by preventing a process from sending a clause unless it is at the appropriate level for the receiving process.

Example 6. Consider the use of AR-NET with two components to prove the theorem from Example 4. The first component is configured to reason about the higher-level concepts `SUBSET` and `EQSET`, while the second component is configured to reason about the lower-level concept `EL`. In particular, the first component is instructed to consider only clauses in the higher-level predicates. The second process is permitted to consider clauses in all predicates; it must be able to map information to lower levels of abstraction, reason at the lower levels of abstraction, and infer higher-level concepts from lower-level concepts. In addition, the first component is instructed to periodically send clauses to the second component for expansion and consideration at lower levels of abstraction. The second component is restricted to send only deduced clauses that are solely in terms of the higher-level concepts. The following is a UR-resolution proof.

```
% proof steps on the second AR-NET process
1 [] -EL(x,Union(y,z)) | EL(x,y) | EL(x,z).
2 [] -EL(x,Inter(y,z)) | EL(x,y).
5 [] SUBSET(x,y) | EL(f(x,y),x).
6 [] SUBSET(x,y) | -EL(f(x,y),y).
```

```
10 [] -SUBSET(Union(A,Inter(A,B)),A)
                # network(0)   # cl_id(6).
12 [ur,10,6] -EL(f(Union(A,Inter(A,B)),A),A).
13 [ur,10,5] EL(f(Union(A,Inter(A,B)),A),Union(A,Inter(A,B))).
15 [ur,12,2] -EL(f(Union(A,Inter(A,B)),A),Inter(A,x)).
27 [ur,15,1,12] -EL(f(Union(A,Inter(A,B)),A),Union(A,Inter(A,x))).
28 [binary,27.1,13.1] $F.

% derivation information from the first AR-NET process
1 [] SUBSET(x,Union(x,y)).
2 [] -SUBSET(x,y) | -SUBSET(y,x) | EQSET(x,y).
5 [] -EQSET(Union(A,Inter(A,B)),A).
6 [ur,5,2,1] -SUBSET(Union(A,Inter(A,B)),A).
```

The empty clause is derived on the second (lower-level) AR-NET component. Clause `10` in the proof is Clause `6` from the first component; it is received from the network. When displaying the proof, the second component makes a request to the first to display the derivation of its Clause `6`. We might expect the primary proof to be completed and presented by the higher-level component. The fact that it isn't in this example is a simple consequence of doing proofs by contradiction and relying on the set of support strategy. The critical lower-level steps depend on a negated clause which is derived on the first component.

Example 7. We once again consider the theorem from Example 2. For the sake of illustration, we can, as in Example 5, consider `CONJ` and `CONGR` to represent relatively low-level concepts and `REL`, `EXT`, and `Q` to represent higher-level concepts. Using a configuration similar to the previous example, AR-NET finds several proofs. Variations in the processing parameters lead to different distributions of the primary proof steps and lemma derivations between the two reasoning components.

6 Future Work

In this article, we motivated the need for automated reasoning programs that can reason effectively at multiple levels of abstraction and described some first steps toward a possible solution. These include a relevance strategy, the use of linked inference rules, and the use of distributed reasoning. There are three categories of future work: the application area described in Section 2, the continued development of linked inference rules, and the continued development of AR-NET.

Application area. In this article, we focused primarily on elementary properties of the general theory. We currently are extending this work to express and reason about computations [11]. In future work, we will account for cost models and complexity issues associated with the execution of algorithms. A long-term objective is to be able to prove general complexity results, including lower bounds. In addition to the issue of levels of abstraction discussed in this article, we face at least two notable automated reasoning challenges: reasoning about finite sets and proving theorems that require existence arguments (for example, the existence of a relation with a certain property).

Linked inference rules. It is essential that we support equality in our linked inference rules and develop even better strategies for restricting the application of all linked inference rules. We have implemented and experimented with prototypes that are suitable for simulating aspects of linking and equality; it is clear that the need for effective restriction strategies is especially important when equality plays a role.

AR-NET. What is the best way to configure AR-NET for a given application? Through extensive experimentation, we are attempting to develop expertise in and establish guidelines for choosing a communications architecture (that is, the number of reasoning components and their connectivity), the distribution of input clauses between the reasoning components, communications guidelines for sharing information between reasoning components, and the processing parameters used by each reasoning component. In addition, we will continue to develop the second model of collaboration (subtasking) and will develop support for a heterogeneous architecture. The latter problem requires us to identify and support a language for sharing information between disparate reasoning components.

Acknowledgments

I thank Paul Helman for introducing me to his theory and for reviewing my first attempts at a first-order representation of the early concepts. Work on this application of automated reasoning continues as a joint effort.

References

1. Barker-Plummer, D., "Gazing: An Approach to the Problem of Definition and Lemma Use," *J. Automated Reasoning* **8**(3), 1992, 311–344.
2. Bellman, R., and Dreyfus, S., *Applied Dynamic Programming*, Princeton, NJ: Princeton University Press, 1962.
3. Boyer, R., and Moore, J, *A Computational Logic* New York: Academic Press, 1979.
4. Boyer, R., Lusk, E., McCune, W., Overbeek, R, Stickel, M., and Wos, L., "Set Theory in First-order Logic: Clauses for Gödel's Axioms," *J. Automated Reasoning* **2**(3), 1986, 287–327.
5. Chang, C., and Lee, R., *Symbolic Logic and Mechanical Theorem Proving*, New York: Academic Press, 1973.
6. Cvetkovic, D., and Pevac, I., "Man-machine Theorem Proving in Graph Theory," *Artificial Intelligence* **35**, 1988, 1–23.
7. Denzinger, J., Kronenburg, M., and Schulz, S., "DISCOUNT: A Distributed and Learning Equational Prover," *J. Automated Reasoning* **18**(2), 1997, 189–198.
8. Giunchiglia, F., and Walsh T., "A Theory of Abstraction," *Artificial Intelligence* **57**(2,3), 1992, 323–389.
9. Jindal, A., Overbeek, R., and Kabat, W., "Exploitation of Parallel Processing for Implementing High-Performance Deduction Systems," *J. Automated Reasoning* **8**(1), 1992, 23–38.

10. Helman, P., "A Common Schema for Dynamic Programming and Branch and Bound Algorithms," *J. ACM* **36**(1), 1989, 97–128.
11. Helman, P., and Veroff, R., "The application of automated reasoning to formal models of combinatorial optimization," Applied Mathematics and Computation, to appear.
12. Henschen, L., "Metalevel Reasoning for Controlling Automated Reasoning Programs," R. Veroff (ed.), *Automated Reasoning and Its Applications*, MIT Press, Cambridge, 1997, 211–230.
13. Lusk, E., McCune, W., and Slaney J., "ROO: A parallel theorem prover." D. Kapur (ed.), *Proceedings of the 11th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, Vol. 607, Springer-Verlag, 1992, 731–734. System abstract.
14. McCharen, J., Overbeek, R., and Wos, L., "Complexity and Related Enhancements for Automated Theorem-proving Programs," *Computers and Mathematics with Applications* **2**, 1976, 1–16.
15. McCune, W., OTTER 3.0 Reference Manual and Guide, Technical Report ANL-94/6, Argonne National Laboratory, Argonne, Illinois, 1994.
16. McCune, W., "Solution of the Robbins Problem," *J. Automated Reasoning* **19**(3), 1997, 263–276.
17. Plaisted, D., "Abstraction Mappings in Mechanical Theorem Proving," W. Bibel and R. Kowalski, (eds.), *Proceedings of the 5th International Conference on Automated Deduction*, Lecture Notes in Computer Science, Vol. 87, Springer-Verlag, 1980, 264–280.
18. Quaife, A., *Automated Development of Fundamental Mathematical Theories*, Dordrecht: Kluwer Academic Publishers, 1992.
19. Robinson, J., "Automatic Deduction with Hyper-resolution," *International Journal of Computer Mathematics* **1**, 1965, 227–234.
20. Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence* **5**, 1974, 115–135.
21. Sibai, F., "Parallel Unification: Theory and Implementations." B. Fronöfer and G. Wrightson (eds.), *Proceedings of the International Worskshop on Parallelization in Inference Systems*, Lecture Notes in Artificial Intelligence, Vol. 590, Springer-Verlag, 1990, 51–81.
22. Sutcliffe, G., "Heterogeneous Parallel Deduction Systems," *Fifth Generation Computer Systems*, Tokyo, 1992, `http://www.cs.jcu.edu.au/~geoff/Papers/HPDS.ps.gz`. Workshop paper.
23. Veroff, R., and Wos, L., "The Linked Inference Principle, I: The Formal Treatment," *J. Automated Reasoning* **8**(2), 1992, 213–274.
24. Wos, L., Veroff, R., Smith, B., and McCune, W., "The Linked Inference Principle, II: The User's Viewpoint." R. Shostak (ed.), *Proceedings of the 7th International Conference on Automated Deduction*, Lecture Notes in Computer Science, Vol. 170, Springer-Verlag, 1984, 316–332.
25. Wos, L., "The Problem of Definition Expansion and Contraction," *J. Automated Reasoning* **3**(4), 1987, 433–435.