Appears in Proceedings of 17th Int'l Joint Conference on Artificial Intelligence (IJCAI '01).

Theorem Proving with Structured Theories

Sheila McIlraith[*] and **Eyal Amir**
Department of Computer Science,
Gates Building, Wing 2A
Stanford University, Stanford, CA 94305-9020, USA
{sheila.mcilraith,eyal.amir}@cs.stanford.edu

Abstract

Motivated by the problem of query answering over multiple structured commonsense theories, we exploit graph-based techniques to improve the efficiency of theorem proving for structured theories. Theories are organized into subtheories that are minimally connected by the literals they share. We present message-passing algorithms that reason over these theories using consequence finding, specializing our algorithms for the case of first-order resolution, and for batch and concurrent theorem proving. We provide an algorithm that restricts the interaction between subtheories by exploiting the polarity of literals. We attempt to minimize the reasoning within each individual partition by exploiting existing algorithms for focused incremental and general consequence finding. Finally, we propose an algorithm that compiles each subtheory into one in a reduced sublanguage. We have proven the soundness and completeness of all of these algorithms.

1 Introduction

Theorem provers are becoming increasingly prevalent as query-answering machinery for reasoning over single or multiple large commonsense knowledge bases (KBs) [3]. Commonsense KBs, as exemplified by Cycorp's Cyc and the High Performance Knowledge Base (HPKB) systems developed by Stanford's Knowledge Systems Lab and by SRI often comprise tens/hundreds of thousands of logical axioms, embodying loosely coupled content in a variety of different subject domains. Unlike mathematical theories (the original domain of automated theorem provers), commonsense theories are often highly structured and with large signatures, lending themselves to graph-based techniques for improving the efficiency of reasoning.

Graph-based algorithms are commonly used as a means of exploiting structure to improve the efficiency of reasoning in Bayes Nets (e.g., [18]), Constraint Satisfaction Problems (CSPs) (e.g., [12]) and most recently in logical reasoning (e.g., [11; 3; 25]). In all cases, the basic approach is to convert a graphical representation of the problem into a tree-structured representation, where each node in the tree represents a tightly-connected subproblem, and the arcs represent the loose coupling between subproblems. Inference is done locally at each node and the necessary information is propagated between nodes to provide a global solution. Inference thus proves to be linear in the tree structure, and often worst-case exponential within the individual nodes.

We leverage these ideas to perform more efficient sound and complete theorem proving over theories in first-order logic (FOL) and propositional logic. In this paper we assume that we are given a first-order or propositional theory that is partitioned into subtheories that are minimally coupled, sharing minimal vocabulary. Sometimes this partitioning is provided by the user because the problem requires reasoning over multiple KBs. Other times, a partitioning is induced automatically to improve the efficiency of reasoning. (Some automated techniques for performing this partitioning are discussed in [3; 2].) This partitioning can be depicted as a graph in which each node represents a particular partition or subtheory and each arc represents shared vocabulary between subtheories. Theorem proving is performed locally in each subtheory, and relevant information propagated to ensure sound and complete entailment in the global theory. To maximize the effectiveness of structure-based theorem proving we must 1) minimize the coupling between nodes of the tree to reduce information being passed, and 2) minimize local inference within each node, while, in both cases, preserving global soundness and completeness.

In this paper we present message-passing algorithms that reason over partitioned theories, minimizing the number of messages sent between partitions and the local inference within partitions. We first extend the applicability of a message-passing algorithm presented in [3] to a larger class of local reasoning procedures. In Section 3 we modify this algorithm to use first-order *resolution* as the local reasoning procedure. In Section 4 we exploit Lyndon's Interpolation Theorem to provide an algorithm that reduces the size of the communication languages connecting partitions by considering the polarity of literals. Finally, in Section 5 we attempt to minimize the reasoning within each partition using algorithms for focused and incremental consequence finding. We also provide an algorithm for compiling partitioned propositional theories into theories in a reduced sublanguage. We

[*] Knowledge Systems Laboratory (KSL)

have proven the soundness and completeness of all of these algorithms with respect to reasoning procedures that are complete for consequence finding in a specified sublanguage. Proofs omitted from this paper can be found at [22].

2 Partition-Based Logical Reasoning

In this section we describe the basic framework adopted in this paper. We extend it with new soundness and completeness results that will enable us to minimize local inference.

Following [3], we say that $\{\mathcal{A}_i\}_{i\leq n}$ is a *partitioning* of a logical theory $\mathcal{A}$ if $\mathcal{A} = \bigcup_i \mathcal{A}_i$. Each individual $\mathcal{A}_i$ is a set of axioms called a *partition*, $L(\mathcal{A}_i)$ is its signature (the set of non-logical symbols), and $\mathcal{L}(\mathcal{A}_i)$ is its language (the set of formulae built with $L(\mathcal{A}_i)$). The partitions may share literals and axioms. A partitioning of a theory induces a graphical representation, $G = (V, E, l)$, which we call the theory's *intersection graph*. Each node of the intersection graph, i, represents an individual partition, $\mathcal{A}_i$, ($V = \{1, ..., n\}$), two nodes i, j are linked by an edge if $\mathcal{L}(\mathcal{A}_i)$ and $\mathcal{L}(\mathcal{A}_j)$ have a non-logical symbol in common ($E = \{(i, j) \mid L(\mathcal{A}_i) \cap L(\mathcal{A}_j) \neq \emptyset\}$), and the edges are labeled with the set of symbols that the associated partitions share ($l(i, j) = L(\mathcal{A}_i) \cap L(\mathcal{A}_j)$). We refer to $l(i, j)$ as the *communication language* between partitions $\mathcal{A}_i$ and $\mathcal{A}_j$. We ensure that the intersection graph is connected by adding a minimal number of edges to E with empty labels, $l(i, j) = \emptyset$. Figure 1 illustrates a propositional theory $\mathcal{A}$ in clausal form (left-hand side) and its partitioning displayed as an intersection graph (right-hand side). (Figures 1, 2 and 3 first appeared in [3].)

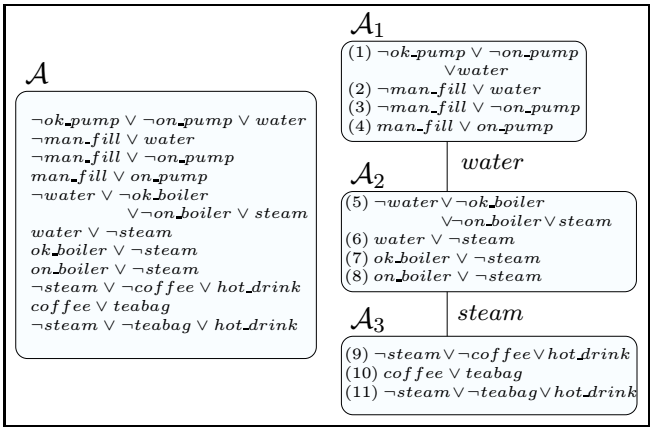

Figure 1: Partitioned theory $\mathcal{A}$ intersection graph G.

Figure 2 displays FORWARD-M-P (FMP), a message-passing algorithm for partition-based logical reasoning. It takes as input a partitioned theory, $\mathcal{A}$, an associated graph structure $G = (V, E, l)$, and a query formula Q in $\mathcal{L}(\mathcal{A}_k)$, and returns YES if the query was entailed by $\mathcal{A}$. The algorithm uses procedures that generate consequences (consequence finders) as the local reasoning mechanism within each partition or graphical node. It passes a concluded formula to an adjacent node if the formula's signature is in the communication language l of the adjacent node, and that node is on the path to the node containing the query.

Recall, consequence finding (as opposed to proof finding) was defined by Lee [19] to be the problem of finding all non-tautological logical consequences of a theory or sentences that subsume them. A prime implicate generator is a popular example of a consequence finder [1].

To determine the direction in which messages should be sent in the graph G, step 1 in FMP computes a strict partial order over nodes in the graph using the partitioning together with a query, Q.

Definition 2.1 ($\prec$) *Given partitioned theory $\mathcal{A} = \bigcup_{i\leq n} \mathcal{A}_i$, associated graph $G = (V, E, l)$ and query $Q \in \mathcal{L}(\mathcal{A}_k)$, let $dist(i, j)$ ($i, j \in V$) be the length of the shortest path between nodes i, j in G. Then $i \prec j$ iff $dist(i, k) < dist(j, k)$.*

PROCEDURE FORWARD-M-P (FMP)($\{\mathcal{A}_i\}_{i\leq n}$, G, Q)

$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a graph describing the connections between the partitions, Q a query in $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. Determine $\prec$ as in Definition 2.1.
2. Concurrently,
   (a) Perform consequence finding for each of the partitions $\mathcal{A}_i$, $i \leq n$.
   (b) For every $(i, j) \in E$ such that $i \prec j$, for every consequence φ of $\mathcal{A}_j$ found (or φ in $\mathcal{A}_j$), if $\varphi \in \mathcal{L}(l(i, j))$, then add φ to the set of axioms of $\mathcal{A}_i$.
   (c) If Q is proven[a] in $\mathcal{A}_k$, return YES.

[a] Derive a subsuming formula or initially add $\neg Q$ to $\mathcal{A}_k$ and derive inconsistency.

Figure 2: A forward message-passing algorithm.

Figure 3 illustrates an execution of the FMP algorithm using resolution as the consequence finder within a partition. As can be seen from the example, the partitioning reduces the number of possible inference steps by precluding the direct resolution of axioms residing in different partitions. Indeed, [3] showed that partition-based reasoning reduces the search space significantly, as a function of the size of the communication language between partitions.

FMP is sound and complete if we guarantee some properties of the graph G and the consequence finders used for each partition. The graph G is required to be a tree that is *properly labeled* for $\mathcal{A}$.

Definition 2.2 (Proper Labeling) *A tree-structured representation, $G = (V, E, l)$, of a partitioned theory $\mathcal{A} = \{\mathcal{A}_i\}_{i\leq n}$ is said to have a* proper labeling, *if for all $(i, j) \in E$ and $\mathcal{B}_1, \mathcal{B}_2$, the two subtheories of $\mathcal{A}$ on the two sides of the edge (i, j) in G, it is true that $l(i, j) \supseteq L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$.*

For example, every intersection graph that is a tree is properly labeled. A simple algorithm called BREAK-CYCLES

[1] Recall, an implicate is a clause entailed by a theory. It is prime if it is minimal in some way. Definitions of prime vary including the use of subsumption, syntactic minimality, or entailment.

Using FMP to prove hot_drink			
Part.	Resolve	Generating	
$\mathcal{A}_1$	(2) , (4)	$on_pump \vee water$	(m1)
$\mathcal{A}_1$	$(m1)$, (1)	$ok_pump \vee water$	(m2)
$\mathcal{A}_1$	$(m2)$, (12)	$water$	(m3)
		clause $water$ passed from $\mathcal{A}_1$ to $\mathcal{A}_2$	
$\mathcal{A}_2$	(m3) , (5)	$ok_boiler \wedge on_boiler \supset steam$	(m4)
$\mathcal{A}_2$	(m4) , (13)	$\neg on_boiler \vee steam$	(m5)
$\mathcal{A}_2$	(m5) , (14)	$steam$	(m6)
		clause $steam$ passed from $\mathcal{A}_2$ to $\mathcal{A}_3$	
$\mathcal{A}_3$	(9) , (10)	$\neg steam \vee teabag \vee hot_drink$	(m7)
$\mathcal{A}_3$	(m7) , (11)	$\neg steam \vee hot_drink$	(m8)
$\mathcal{A}_3$	(m8) , $(m6)$	hot_drink	(m9)

Figure 3: A proof of hot_drink from $\mathcal{A}$ in Figure 1 after asserting ok_pump (12) in $\mathcal{A}_1$ and ok_boiler (13), on_boiler (14) in $\mathcal{A}_2$.

that transforms an intersection graph that is not tree into a properly labeled tree was presented in [3]. Note that the notion of proper labeling is equivalent, in this context, to the running intersection property used in Bayes Nets.

The consequence finders applied to each partition i are required to be *complete for $\mathcal{L}_i$-generation* for a sublanguage $\mathcal{L}_i \subset \mathcal{L}(\mathcal{A}_i)$ that depends on the graph G and the query Q.

Definition 2.3 (Completeness for $\mathcal{L}$-Generation) *Let $\mathcal{A}$ be a set of axioms, $\mathcal{L} \subseteq \mathcal{L}(\mathcal{A})$ a language, and $\mathfrak{R}$ a consequence finder. Let $C_{\mathfrak{R},\mathcal{L}}(\mathcal{A})$ be the consequences of $\mathcal{A}$ generated by $\mathfrak{R}$ that are in $\mathcal{L}$. $\mathfrak{R}$ is* complete for $\mathcal{L}$-generation *if for all $\varphi \in \mathcal{L}$, if $\mathcal{A} \models \varphi$, then $C_{\mathfrak{R},\mathcal{L}}(\mathcal{A}) \models \varphi$.*

Theorem 2.4 (Soundness and Completeness) *Let $\mathcal{A}$ be a partitioned theory $\{\mathcal{A}_i\}_{i\leq n}$ of arbitrary propositional or first-order formulae, G a tree that is properly labeled with respect to $\mathcal{A}$, and $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leq n$, a query. For all $i \leq n$, let $\mathcal{L}_i = \mathcal{L}(l(i,j))$ for j such that $(i,j) \in E$ and $j \prec i$ (there is only one such j), and let $\{\mathfrak{R}_i\}_{i\leq n}$ be reasoning procedures associated with partitions $\{\mathcal{A}_i\}_{i\leq n}$. If every $\mathfrak{R}_i$ is complete for $\mathcal{L}_i$-generation then $\mathcal{A} \models Q$ iff FMP($\{\mathcal{A}_i\}_{i\leq n}$, G, Q) outputs YES.*

This soundness and completeness result improves upon a soundness and completeness result in [3] by allowing consequence finders that focus on consequences in the communication language between partitions. In certain cases, we can restrict consequence finding in FMP even further by using reasoners that are complete for *$\mathcal{L}$-consequence finding*.

Definition 2.5 (Completeness for $\mathcal{L}$-Consequence Finding) *Let $\mathcal{A}$ be a set of axioms, $\mathcal{L} \subseteq \mathcal{L}(\mathcal{A})$ a language, and $\mathfrak{R}$ a consequence finder. $\mathfrak{R}$ is* complete for $\mathcal{L}$-consequence finding *iff for every $\varphi \in \mathcal{L}$ that is not a tautology, $\mathcal{A} \models \varphi$ iff there exists $\psi \in \mathcal{L}$ such that $\mathcal{A} \vdash_{\mathfrak{R}} \psi$ and ψ subsumes[2] φ.*

[2]For clausal theories, we say that clause ψ subsumes φ if there is a substitution θ such that $\psi\theta \subset \varphi$.

Observe that every reasoner that is complete for $\mathcal{L}$-consequence finding is also complete for $\mathcal{L}$-generation, for any language $\mathcal{L}$ that is closed under subsumption [14]. The notion of a consequence finder restricting consequence generation to consequences in a designated sublanguage was discussed by Inoue [17], and further developed by del Val [14] and others. Most results on the completeness of consequence finding exploit resolution-based reasoners, where completeness results for $\mathcal{L}$-consequence finding are generally restricted to a clausal language $\mathcal{L}$. The FMP reasoners in Theorem 2.4 must be complete for $\mathcal{L}_i$-generation in arbitrary FOL languages, $\mathcal{L}_i$. Corollary 2.6 refines Theorem 2.4 by restricting $\mathcal{A}_i$ and $\mathcal{L}_i$ to propositional clausal languages and requiring reasoners to be complete for $\mathcal{L}_i$-consequence finding rather than $\mathcal{L}_i$-generation.

Corollary 2.6 (Soundness and Completeness) *Let $\mathcal{A}$ be a partitioned theory $\{\mathcal{A}_i\}_{i\leq n}$ of propositional clauses, G a tree that is properly labeled with respect to $\mathcal{A}$, and $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leq n$, a query. Let $\mathcal{L}_i = \mathcal{L}(l(i,j))$ for j such that $(i,j) \in E$ and $j \prec i$ (there is only one such j), and let $\{\mathfrak{R}_i\}_{i\leq n}$ be reasoning procedures associated with partitions $\{\mathcal{A}_i\}_{i\leq n}$. If every $\mathfrak{R}_i$ is complete for $\mathcal{L}_i$-consequence finding then $\mathcal{A} \models Q$ iff FMP($\{\mathcal{A}_i\}_{i\leq n}$, G, Q) outputs YES.*

In Section 5 we provide examples of reasoners that are complete for $\mathcal{L}$-consequence finding and show how to exploit them to focus reasoning within a partition.

3 Local Inference Using Resolution

In this section, we specialize FMP to *resolution-based* consequence finders. *Resolution* [26] is one of the most widely used reasoning methods for automated deduction, and more specifically for consequence finding. It requires the input formula to be in clausal form, i.e., a conjunction of disjunctions of unquantified literals. The *resolution rule* is complete for consequence finding (e.g., [19; 27]) and so is *linear resolution* and some of its variants (e.g., [23]).

We present algorithm RESOLUTION-M-P (RES-MP), that uses resolution (or resolution strategies), in Figure 4. The rest of this section is devoted to explaining four different implementations for subroutine RES-SEND(φ, j, i), used by this procedure to send appropriate messages between partitions: the first implementation is for clausal propositional theories; the second is for clausal FOL theories, with associated graph G, that is a properly labeled trees and whose labels include all the function and constant symbols of the language; the third is also for clausal FOL theories, however it uses *unskolemization* and subsequent Skolemization to generate the messages to be passed between partitions; the fourth is a refinement of the third for the same class of theories that avoids unskolemization.

In the propositional case, subroutine RES-SEND(φ, j, i) (Implementation 1) simply adds φ to $\mathcal{A}_i$, as done in FMP. If the resolution strategies being employed satisfy the conditions of Corollary 2.6, then RES-MP is sound and complete.

In the FOL case, implementing RES-SEND requires more care. To illustrate, consider the case where resolution generates the clause $P(B,x)$ (B a constant symbol and x a variable). It also implicitly proves that $\exists b\ P(b,x)$. RES-MP

PROCEDURE RESOLUTION-M-P(RES-MP)($\{\mathcal{A}_i\}_{i\leq n}$, G, Q)
$\{\mathcal{A}_i\}_{i\leq n}$ a partitioned theory, $G = (V, E, l)$ a graph, Q a query formula in the language of $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. Determine $\prec$ as in Definition 2.1.
2. Add the clausal form of $\neg Q$ to $\mathcal{A}_k$.
3. Concurrently,
   (a) Perform resolution for each of the partitions $\mathcal{A}_i$, $i \leq n$.
   (b) For every $(i, j) \in E$ such that $i \prec j$, if partition $\mathcal{A}_j$ includes the clause φ (as input or resolvent) and the predicates of φ are in $\mathcal{L}(l(i, j))$, then perform RES-SEND(φ, j, i).
   (c) If Q is proven in $\mathcal{A}_k$, return YES.

Figure 4: A resolution forward message-passing algorithm.

may need to send $\exists b\ P(b, x)$ from one partition to another, but it cannot send $P(B, x)$ if B is not in the communication language between partitions (for ground theories there is no such problem (see [27])). In the first-order case, completeness for consequence finding for a clausal first-order logic language (e.g., Lee's result for resolution) does not guarantee completeness for $\mathcal{L}$-generation for the corresponding full FOL language, $\mathcal{L}$. This problem is also reflected in a slightly different statement of Craig's interpolation theorem [10] that applies for resolution [27].

A simple way of addressing this problem is to add all constant and function symbols to the communication language between every connected set of partitions. This has the advantage of preserving soundness and completeness, and is simple to implement. In this case, subroutine RES-SEND(φ, j, i) (Implementation 2) simply adds φ to $\mathcal{A}_i$, as done in FMP.

In large systems that consist of many partitions, the addition of so many constant and function symbols to each of the other partitions has the potential to be computationally inefficient, leading to many unnecessary and irrelevant deduction steps. Arguably, a more compelling way of addressing the problems associated with resolution for first-order theories is to infer the existential formula $\exists b\ P(b, x)$ from $P(B, x)$, send this formula to the proper partition and Skolemize it there. For example, if $\varphi = P(f(g(B)), x)$ is the clause that RES-SEND gets, replacing it with $\exists b\ P(b, x)$ eliminates unnecessary work of the receiving partition.

The process of conservatively replacing function and constant symbols by existentially quantified variables is called *unskolemization* or *reverse Skolemization* and is discussed in [5; 9; 8]. [8] presents an algorithm *U* that is complete for our purposes and generalizes and simplifies an algorithm of [9]. (Space precludes inclusion of the algorithm.)

Theorem 3.1 ([8]) *Let V be a vocabulary and φ, ψ be formulae such that $\psi \in \mathcal{L}(V)$ and $\varphi \Rightarrow \psi$. There exists $F \in \mathcal{L}(V)$ that is generated by algorithm* U *such that $F \Rightarrow \psi$.*

Thus, for every resolution strategy that is complete for $\mathcal{L}$-consequence finding, unskolemizing φ using procedure *U* for $V = l(i, j)$ and then Skolemizing the result gives us a combined procedure for message generation that is complete for $\mathcal{L}_j$-generation. This procedure can then be used readily in RES-MP (or in FMP), upholding the soundness and completeness to that supplied by Theorem 2.4. The subroutine RES-SEND(φ, j, i) that implements this approach is presented in Figure 5. It replaces φ with a a set of formulae in $\mathcal{L}(l(i, j))$ that follows from φ. It then Skolemizes the resulting formulae for inclusion in $\mathcal{A}_i$.

PROCEDURE RES-SEND(φ, j, i) (Implementation 3)
φ a formula, $j, i \leq n$.

1. Unskolemize φ into a set of formulae, Φ in $\mathcal{L}(l(i, j))$, treating every symbol of $L(\varphi) \setminus l(i, j)$ as a Skolem symbol.
2. For every $\varphi_2 \in \Phi$, if φ_2 is not subsumed by a clause that is in $\mathcal{A}_i$, then add the Skolemized version of φ_2 to the set of axioms of $\mathcal{A}_i$.

Figure 5: Subroutine RES-SEND using unskolemization.

Procedure *U* may generate more than one formula for any given clause φ. For example, if $\varphi = P(x, f(x), u, g(u))$, for $l(i, j) = \{P\}$, then we must generate both $\forall x \exists y \forall u \exists v P(x, y, u, v)$ and $\forall u \exists v \forall x \exists y P(x, y, u, v)$ (φ entails both quantified formulae, and there is no one quantified formula that entails both of them). In our case we can avoid some of these quantified formulae by replacing the *unskolemize and then Skolemize* process of RES-SEND (Implementation 3) with a procedure that produces a set of formulae directly (Implementation 4). It is presented in Figure 6.

PROCEDURE RES-SEND(φ, j, i) (Implementation 4)
φ a formula, $j, i \leq n$.

1. Set $S \leftarrow L(\varphi) \setminus l(i, j)$.
2. For every term instance, $t = f(t_1, ..., t_k)$, in φ, if $f \in S$ and t is not a subexpression of another term $t' = f'(t'_1, ..., t'_{k'})$ of φ with $f' \in S$, then replace t with "$x \leftarrow t$" for some new variable, x (if $k = 0$, t is a constant symbol).
3. Nondeterministically[a], for every pair of marked arguments "$x \leftarrow \alpha$", "$y \leftarrow \beta$", in φ, if α, β are unifiable, then unify all occurrences of x, y (i.e., unify α_i, β_i for all markings $x \leftarrow \alpha_i$, $y \leftarrow \beta_i$).
4. For every marked argument "$x \leftarrow \alpha$" in φ, do
   (a) Collect all marked arguments with the same variable on the left-hand side of the "$\leftarrow$" sign. Suppose these are $x \leftarrow \alpha_1, ..., x \leftarrow \alpha_l$.
   (b) Let $y_1, ..., y_r$ be all the variables occurring in $\alpha_1, ..., \alpha_l$. For every $i \leq l$, replace "$x \leftarrow \alpha_i$" with $f(y_1, ..., y_r)$ in φ, for a fresh function symbol f (if $r = 0$, f is a fresh constant symbol).
5. Add φ to $\mathcal{A}_i$.

[a] Nondeterministically select the set of pairs for which to unify all occurrences of x, y.

Figure 6: Subroutine RES-SEND without unskolemization.

Steps 2–3 of procedure RES-SEND(φ, j, i) (Implementation 4) correspond to similar steps in procedure *U* presented

in [8], simplifying where appropriate for our setup. Our procedure differs from unskolemizing procedures in step 4, where it stops short of replacing the Skolem functions and constants with new existentially quantified variables. Instead, it replaces them with new functions and constant symbols. The nondeterminism of step 3 is used to add *all* the possible combinations of unified terms, which is required to ensure completeness.

For example, if $\varphi = P(f(g(B)), x)$ and $l(i,j) = \{P\}$, then RES-SEND adds $P(A, x)$ to $\mathcal{A}_i$, for a new constant symbol, A. If $\varphi = P(x, f(x), u, g(u))$, for $l(i,j) = \{P\}$, then RES-SEND adds $P(x, h_1(x), u, h_2(u))$ to $\mathcal{A}_i$, for new function symbols h_1, h_2. Finally, if $\varphi = P(x, f(x), u, f(g(u)))$, then RES-SEND adds $P(x, f(x), u, h(u))$ and $P(h_1(u), h_2(u), u, h_2(u))$ to $\mathcal{A}_i$, for h, h_1, h_2 new function symbols.

Theorem 3.2 (Soundness & Completeness of RES-MP) *Let $\mathcal{A}$ be the partitioned theory $\bigcup_{i \leq n} \mathcal{A}_i$ of propositional or first-order clauses, G a tree that is properly labeled with respect to $\mathcal{A}$, and $Q \in \mathcal{L}(\mathcal{A}_k)$ $k \leq n$, a sentence that is the query. $\mathcal{A} \models Q$ iff applying RES-MP($\{\mathcal{A}_i\}_{i \leq n}$, G, Q) (with Implementation 4 of RES-SEND) outputs YES.*

PROOF SKETCH Soundness and completeness of the algorithm follow from that of FMP, if we show that RES-SEND (Implementation 4) adds enough sentences (implying completeness) to $\mathcal{A}_i$ that are implied by φ (thus sound) in the restricted language $\mathcal{L}(l(i,j))$.

If we add all sentences φ that are submitted to RES-SEND to $\mathcal{A}_i$ without any translation, then our soundness and completeness result for FMP applies (this is the case where we add all the constant and function symbols to all $l(i,j)$).

We use Theorem 3.1 to prove that we add enough sentences to $\mathcal{A}_i$. Let φ_2 be a quantified formula that is the result of applying algorithm *U* to φ. Then, φ_2 results from a clause C generated in step 4 of algorithm *U* (respectively, Step 3 in RES-SEND). In algorithm *U*, for each variable x, the markings "$x \leftarrow \alpha_i$" in C are converted to a new variable that is existentially quantified immediately to the right of the quantification of the variables $y_1, ..., y_r$. φ_2 is a result of ordering the quantifiers in a consistent manner to this rule (this process is done in steps 5–6 of algorithm *U*).

Step 4 of RES-SEND performs the same kind of replacement that algorithm *U* performs, but uses new function symbols instead of new quantified variables. Since each new quantified variable in φ_2 is to the right of the variables on which it depends, and our new function uses exactly those variables as arguments, then Step 4 generates a clause C' from C that entails φ_2. Thus, the clauses added to $\mathcal{A}_i$ by RES-SEND entail all the clauses generated by unskolemizing φ using *U*. From Theorem 3.1, these clauses entail all the sentences in $\mathcal{L}(l(i,j))$ that are implied by φ.

To see that the result is still sound, notice that the set of clauses that we add to $\mathcal{A}_i$ has the same consequences as φ in $\mathcal{L}(l(i,j))$ (i.e., if we add those clauses to $\mathcal{A}_j$ we get a conservative extension of $\mathcal{A}_j$). ■

Resolution strategies that can be readily used in RES-MP, while preserving completeness, include linear resolution [23], *directional resolution* [25] and *lock resolution* [7]. Strategies such as set-of-support and semantic resolution can be used with somewhat different treatments.

4 Minimizing Node Coupling Using Polarity

FMP and RES-MP use the communication language to determine relevant inference steps between formulae in connected partitions. This section improves the efficiency of FMP and RES-MP by exploiting the polarity of predicates in our partitions to further constrain the communication language between partitions. This leads to a reduction in the number of messages that are passed between adjacent partitions, and thus a reduction in the search space size of the global reasoning problem. Our results are predicated on Lyndon's Interpolation Theorem [20], an extension to Craig's Theorem [10].

Theorem 4.1 (Lyndon's Interpolation Theorem) *Let α, β be sentences such that $\alpha \vdash \beta$. Then there exists a sentence γ such that $\alpha \vdash \gamma$ and $\gamma \vdash \beta$, and that every relation symbol that appears positively [negatively] in γ appears positively [negatively] in both α and β. γ is referred to as the* interpolant *of α and β.*

This theorem guarantees that FMP need only send clauses with literals that may be used in subsequent inference steps. For example, let $\{\mathcal{A}_1, \mathcal{A}_2\}$ be a partitioned theory, $G = (V = \{1,2\},\ E = \{(1,2)\},\ l)$ be a graph, and $Q \in \mathcal{L}(\mathcal{A}_2)$, be a query. If FMP concluded P from $\mathcal{A}_1$, and P does not show positively in $\mathcal{A}_2 \Rightarrow Q$ (i.e., P does not show negatively in $\mathcal{A}_2$ and does not show positively in Q), then there is no need to send the message P from $\mathcal{A}_1$ to $\mathcal{A}_2$.

Procedure POLARIZE (Figure 7) takes as input a partitioned theory, associated tree $G = (V, E, l)$, and a query Q. It returns a new graph $G' = (V, E, l')$ that is minimal with respect to our interpretation of Lyndon's Interpolation Theorem. The labels of the graph now include predicate/propositional symbols with associated polarities (the same symbol may appear both positively and negatively on an edge label). All function and object symbols that appeared in l also appear in l' for the respective edges.

Theorem 4.2 (Soundness and Completeness) *Let $\mathcal{A}$ be a partitioned theory $\{\mathcal{A}_i\}_{i \leq n}$ of arbitrary propositional or first-order formulae, G a tree that is properly labeled with respect to $\mathcal{A}$, and $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leq n$, a query. Let G' be the result of running POLARIZE($\{\mathcal{A}_i\}_{i \leq n}$, G, Q). Let $\mathcal{L}_i = \mathcal{L}(l(i,j))$ for j such that $(i,j) \in E$ and $j \prec i$ (there is only one such j), and let $\{\mathfrak{R}_i\}_{i \leq n}$ be reasoning procedures associated with partitions $\{\mathcal{A}_i\}_{i \leq n}$. If every $\mathfrak{R}_i$ is complete for $\mathcal{L}_i$-generation then $\mathcal{A} \models \varphi$ iff FMP($\{\mathcal{A}_i\}_{i \leq n}$, G',Q) outputs YES.*

Darwiche [11] proposed a more restricted use of polarity in graph-based algorithms for propositional SAT-search. His proposal is equivalent to first finding those propositional symbols that appear with a unique polarity throughout the theory and then assigning them the appropriate truth value. In contrast, our proposed exploitation of polarity is useful for both propositional and first-order theories, it is more effective in constraining inference steps, and is applicable to a broader class of message-passing algorithms problems. In particular,

PROCEDURE POLARIZE($\{\mathcal{A}_i\}_{i\leq n}$, G, Q)

$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a tree and Q a query formula in $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. For every $i, j \in V$, set $l'(i, j)$ to be the set of object and function symbols that appear in $l(i, j)$, if there are any.
2. Rewrite $\{\mathcal{A}_i\}_{i\leq n}$ such that all negations appear in front of literals (i.e., in negation normal form).
3. Determine $\prec$ as in Definition 2.1.
4. For all $(i, j) \in E$ such that $i \prec j$, for every predicate symbol $P \in l(i, j)$,
   (a) Let V_1, V_2 be the two sets of vertices in V separated by i in G, with $j \in V_1$.
   (b) If $[\neg]P$ appears in V_1 then, if $[\neg]P$ appears in Q or $\neg[\neg]P$ appears in $\mathcal{A}_m$, for some $m \in V_2$, then add $[\neg]P$ to $l'(i, j)$.
5. Return $G' = (V, E, l')$.

Figure 7: Constraining the communication language of $\{\mathcal{A}_i\}_{i\leq n}$ by exploiting polarity.

our method is useful in cases where symbols appear with different polarities in different partitions.

5 Minimizing Local Inference

To maximize the effectiveness of structure-based theorem proving, we must minimize local inference within each node of our tree-structured problem representation, while preserving global soundness and completeness. First-order and propositional consequence finding algorithms have been developed that limit deduction steps to those leading to *interesting* consequences, *skipping* deduction steps that do not.

In the propositional case, the most popular algorithms are certain $\mathcal{L}$-(prime) implicate finders. (See [21] for an excellent survey.) SOL-resolution (skipping ordered linear resolution) [17] and SFK-resolution (skip-filtered, kernel resolution) [14] are two first-order resolution-based $\mathcal{L}$-consequence finders. SFK-resolution is complete for first-order $\mathcal{L}$-consequence finding, reducing to Directional Resolution in the propositional case [13]. In contrast, SOL-resolution is not complete for first-order $\mathcal{L}$-consequence finding, but is complete for first-order *incremental* $\mathcal{L}$-consequence finding. Given new input Φ, an *incremental $\mathcal{L}$-consequence finder* finds the consequences of $\mathcal{A} \cup \Phi$ that were not entailed by Φ alone. Defining *completeness for incremental $\mathcal{L}$-consequence finding* is analogous to Definition 2.5.

In the rest of this section, we propose strategies that exploit our graphical models and specialized consequence finding algorithms to improve the efficiency of reasoning. Following the results in previous sections, using SFK-resolution as a reasoner within partitions will preserve the soundness and completeness of the global problem while significantly reducing the number of inference steps. SFK-resolution can be used by all of the procedures below. Unless otherwise noted, the algorithms we describe are limited to propositional theories because first-order consequence finders may fail to terminate, even for decidable cases of FOL.

The first strategy is compilation. Figure 8 provides an algorithm, COMPILE($\{\mathcal{A}_i\}_{i\leq n}$, G), that takes as input a partitioned theory $\{\mathcal{A}_i\}_{i\leq n}$ and associated tree G, that is properly labeled, and outputs a compiled partitioned theory $\{\mathcal{A}_i^{'}\}_{i\leq n}$. Each new partition is composed of the logical consequences of partition $\mathcal{A}_i$ that are in the language $\mathcal{L}_{comm_i}$, all the communication languages associated with $\mathcal{A}_i$. Prime implicate finders have commonly been used for knowledge compilation, particularly in propositional cases. SFK-resolution can be used as the sound and complete $\mathcal{L}$-consequence finder in Step 2 of COMPILE.

Knowledge compilation can often create a large theory. Each partition produced by COMPILE($\{\mathcal{A}_i\}_{i\leq n}$, G) will be of worst case size of $\mathcal{O}(2^{|L(\mathcal{L}_{comm_i})|})$ clauses. Since our assumption is that partitions are produced to minimize communication between partitions, $\mid L(\mathcal{L}_{comm_i}) \mid$ should be much smaller than $\mid L(\mathcal{A}_i) \mid$. As a consequence, we might expect the compiled theory to be smaller than the original theory, though this is not guaranteed. Under the further assumption that the theories in partitions are fairly static, the cost of compilation will be amortized over many queries. We discuss further options for compilation, including the use of partial compilation, in a longer paper.

PROCEDURE COMPILE($\{\mathcal{A}_i\}_{i\leq n}$, G)

$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a tree with proper labeling for $\mathcal{A}$. For each partition $\mathcal{A}_i$, For $i = 1, \ldots, n$,

1. Let $\mathcal{L}_{comm_i} = \mathcal{L}(\bigcup_{(i,j)\in E} l(i, j))$
2. Using a sound and complete $\mathcal{L}$-consequence finder, perform $\mathcal{L}_{comm_i}$-consequence finding on each partition $\mathcal{A}_i$, placing the output in a new partition $\mathcal{A}_i^{'}$.

Figure 8: A partition-based theory compilation algorithm.

Proposition 5.1 *Let $\mathcal{A} = \bigcup_{i\leq n} \mathcal{A}_i$ be a partitioned theory with associated tree G that is properly labeled for $\mathcal{A}$. Let $\mathcal{L}_{comm_i} = \mathcal{L}(\bigcup_{(i,j)\in E} l(i, j)))$. For all $\varphi \in \mathcal{L}_i \subseteq \mathcal{L}_{comm_i} \subseteq \mathcal{L}(\mathcal{A}_i)$, $\mathcal{A}_i \models \varphi$ iff $\mathcal{A}_i^{'} \models \varphi$, where $\{\mathcal{A}_i^{'}\}_{i\leq n}$ are the compiled partitions output by COMPILE($\{\mathcal{A}_i\}_{i\leq n}$, G).*

We may use our compiled theories in several different strategies for batch-style and concurrent theorem proving, as well as in our previous message-passing algorithms. Figure 9 presents an algorithm for batch-style structure-based theorem proving. BATCH-MP takes as input a (possibly compiled) partitioned theory, associated tree G that is properly labeled, and query Q. For each partition in order, it exploits focused $\mathcal{L}$-consequence finding to compute all the relevant consequences of that theory. It passes the conclusions towards the partition with the query. This algorithm is very similar to the bucket elimination algorithm of [13]. BATCH-MP preserves soundness and completeness of the global problem, while exploiting focused search within each partition.

Theorem 5.2 (Soundness and Completeness) *Let $\mathcal{A}$ be a set of clauses in propositional logic. Let $\{\mathfrak{R}_i\}_{i\leq n}$ be the $\mathcal{L}_i$-consequence finders associated with partitions $\{\mathcal{A}_i\}_{i\leq n}$*

PROCEDURE BATCH-MP ($\{\mathcal{A}_i\}_{i\leq n}$, G, Q)

$\{\mathcal{A}_i\}_{i\leq n}$ a (compiled) partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a properly labeled tree describing the connections between the partitions, Q a query in $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. If $\{\mathcal{A}_i\}_{i\leq n}$, is a compiled theory, replace partition A_k with the partition A_k from the uncompiled theory.
2. Determine $\prec$ as in Definition 2.1.
3. Let $\mathcal{L}_i = \mathcal{L}(l(i, j))$ for j such that $(i, j) \in E$ and $j \prec i^a$.
4. Following $\prec$ in a decreasing order, for every $(i, j) \in E$ such that $j \prec i^a$,
   Run the $\mathcal{L}_i$-consequence finder on $\mathcal{A}_i$ until it has exhausted its consequences, and add the consequences in $\mathcal{L}_i$ to $\mathcal{A}_j$.
5. If Q is proven[b] in $\mathcal{A}_k$, return YES.

[a] There is only one such j.

[b] Derive a subsuming formula or initially add $\neg Q$ to $\mathcal{A}_k$ and derive inconsistency.

Figure 9: A batch-style message-passing algorithm.

in step 4 of BATCH-MP ($\{\mathcal{A}_i\}_{i\leq n}$,G,Q). If every $\mathfrak{R}_i$ is complete for $\mathcal{L}_i$-consequence finding then $\mathcal{A} \models Q$ iff applying BATCH-MP($\{\mathcal{A}_i\}_{i\leq n}$,G,Q) outputs YES.

Our final algorithm, CONCURRENT-MP, (Figure 10), takes as input a (possibly compiled) partitioned theory, associated tree G that is properly labeled, and query Q. It exploits incremental $\mathcal{L}$-consequence finding in the output communication language of each partition to compute the relevant incremental consequences of that theory, and then passes them towards the partition with the query. Once again, SFK-resolution can be used as the sound and complete $\mathcal{L}$-consequence generator for the preprocessing (Step 4). In the case where the theory is compiled into propositional prime implicates, the consequences in $\mathcal{L}_i$ may simply be picked out of the existing consequences in $\mathcal{A}_i$. SOL-resolution can be used as the sound and complete incremental $\mathcal{L}$-consequence finder (Step 6a). CONCURRENT-MP preserves soundness and completeness of the global problem in the propositional case, while exploiting focused search within each partition.

Theorem 5.3 (Soundness and Completeness) *Let $\mathcal{A}$ be a set of clauses in propositional logic. Let $\{\mathfrak{R}_i\}_{i\leq n}$ be the $\mathcal{L}_i$-consequence finders associated with partitions $\{\mathcal{A}_i\}_{i\leq n}$ in step 4 of CONCURRENT-MP($\{\mathcal{A}_i\}_{i\leq n}$,G,Q) and let $\{\mathfrak{R}'_i\}_{i\leq n}$ be the incremental $\mathcal{L}_i$-consequence finders associated with partitions $\{\mathcal{A}_i\}_{i\leq n}$ in step 6 of CONCURRENT-MP($\{\mathcal{A}_i\}_{i\leq n}$,G,Q). If every $\mathfrak{R}_i$ is complete for $\mathcal{L}_i$-consequence finding, and every $\mathfrak{R}'_i$ is complete for incremental $\mathcal{L}_i$-consequence finding then $\mathcal{A} \models Q$ iff applying CONCURRENT-MP($\{\mathcal{A}_i\}_{i\leq n}$,G,Q) outputs YES.*

6 Related Work

A number of AI reasoning systems exploit some type of structure to improve the efficiency of reasoning. While our exploitation of graph-based techniques is similar to that used in Bayes Nets (e.g., [18]) our work is distinguished in that we

PROCEDURE CONCURRENT-MP ($\{\mathcal{A}_i\}_{i\leq n}$, G, Q)

$\{\mathcal{A}_i\}_{i\leq n}$ a (compiled) partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a properly labeled tree describing the connections between the partitions, Q a query in $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. Determine $\prec$ as in Definition 2.1.
2. Let $\mathcal{L}_i = \mathcal{L}(l(i, j))$ for j such that $(i, j) \in E$ and $j \prec i^a$.
3. If $\{\mathcal{A}_i\}_{i\leq n}$, is a compiled theory, then replace partition A_k with the partition A_k from the uncompiled theory.
4. For every $i \leq n$, run the $\mathcal{L}_i$-consequence finder on partition $\mathcal{A}_i$ until it has exhausted its consequences.
5. For every $(i, j) \in E$ such that $j \prec i^a$, add the $\mathcal{L}_i$-prime implicates to partition $\mathcal{A}_j$.
6. Concurrently,
   (a) For every $(i, j) \in E$ such that $j \prec i^a$, perform incremental $\mathcal{L}_i$-consequence finding for each of the partition $\mathcal{A}_i$ and add the the consequences in $\mathcal{L}_i$ to $\mathcal{A}_j$.
   (b) If Q is proven[b] in $\mathcal{A}_k$, return YES.

[a] There is only one such j.

[b] Derive a subsuming formula or initially add $\neg Q$ to $\mathcal{A}_k$ and derive inconsistency.

Figure 10: A concurrent message-passing algorithm.

reason with logical rather than probabilistic theories, where notions of structure and independence take on different roles in reasoning. Our work is most significantly distinguished from work on CSPs (e.g., [12]) and more recently, logical reasoning (e.g., [11; 25]) in that we reason with explicitly partitioned theories using message passing algorithms and our algorithms apply to FOL as well as propositional theories.

In the area of FOL theorem proving, our work is related to research on parallel theorem proving (see surveys in [6; 15]) and on combining logical systems (e.g., [24; 4]). Most parallel theorem prover implementations are guided by lookahead and subgoals to decompose the search space dynamically or allow messages to be sent between different provers working in parallel, using heuristics to decide on which messages are relevant to each prover. These approaches typically look at decompositions into very few sub-problems. In addition, the first approach typically requires complete independence of the sub-spaces or the search is repeated on much of the space by several reasoners. In the second approach there is no clear methodology for deciding what messages should be sent and from which partition to which.

The work on combining logical systems focuses on combinations of signature-disjoint theories (allowing the queries to include symbols from all signatures) and decision procedures suitable for those theories. All approaches either nondeterministically instantiate the (newly created) variables connecting the theories or restrict the theories to be convex (disjunctions are intuitionistic) and have information flowing back and forth between the theories. In contrast, we focus on the structure of interactions between theories with signatures that share symbols and the efficiency of reasoning with consequence finders and theorem provers. We do not have any

restrictions on the language besides finiteness.

Work on formalizing and reasoning with *context* (see [1] for a survey) can be related to theorem proving with structured theories by viewing the contextual theories as interacting sets of theories. Unfortunately, to introduce explicit contexts, a language that is more expressive than FOL is needed. Consequently, a number of researchers have focused on context for propositional logic, while much of the reasoning work has focused on proof checking (e.g., GETFOL [16]).

7 Summary

In this paper we exploited graph-based techniques to improve the efficiency of theorem proving for structured theories. Theories were organized into subtheories that were minimally connected by the literals they share. We presented message-passing algorithms that reason over these theories using consequence finding, specializing our algorithms for the case of first-order resolution, and for batch and concurrent theorem proving. We provided an algorithm that restricts the interaction between subtheories by exploiting the polarity of literals. We attempted to minimize the reasoning within each individual partition by exploiting existing algorithms for focused incremental and general consequence finding. Finally, we proposed an algorithm that compiles each subtheory into one in a reduced sublanguage. We have proven the soundness and completeness of all of these algorithms. The results presented in this paper contribute towards addressing the problem of reasoning efficiently with large or multiple structured commonsense theories.

Acknowledgements

We wish to thank the anonymous IJCAI reviewers for their thorough review of this paper, and Alvaro del Val and Pierre Marquis for helpful comments on the relationship between our work and previous work on consequence finding. This research was supported in part by DARPA grant N66001-97-C-8554-P00004, NAVY grant N66001-00-C-8027, and by DARPA grant N66001-00-C-8018 (RKF program).

References

[1] V. Akman and M. Surav. Steps toward formalizing context. *AI Magazine*, 17(3):55–72, 1996.

[2] E. Amir. Efficient approximation for triangulation of minimum treewidth. Manuscript submitted for publication. Available at http://www-formal.stanford.edu/eyal/papers/decomp-uai2001.ps, 2001.

[3] E. Amir and S. McIlraith. Paritition-based logical reasoning. In *Proc. KR '2000*, pages 389–400. Morgan Kaufmann, 2000.

[4] F. Baader and K. U. Schulz. Combination of constraint solvers for free and quasi-free structures. *Theoretical Computer Science*, 192(1):107–161, 1998.

[5] W. W. Bledsoe and A. M. Ballantyne. Unskolemizing. Technical Report Memo ATP-41, Mathematics Department, University of Texas, Austin, 1978.

[6] M. P. Bonacina and J. Hsiang. Parallelization of deduction strategies: an analytical study. *Journal of Automated Reasoning*, 13:1–33, 1994.

[7] R. S. Boyer. *Locking: a restriction of resolution*. PhD thesis, Mathematics Department, University of Texas, Austin, 1971.

[8] R. Chadha and D. A. Plaisted. Finding logical consequences using unskolemization. In *Proceedings of ISMIS'93*, volume 689 of *LNAI*, pages 255–264. Springer-Verlag, 1993.

[9] P. Cox and T. Pietrzykowski. A complete nonredundant algorithm for reversed skolemization. *Theoretical Computer Science*, 28:239–261, 1984.

[10] W. Craig. Linear reasoning. a new form of the Herbrand-Gentzen theorem. *J. of Symbolic Logic*, 22:250–268, 1957.

[11] A. Darwiche. Utilizing knowledge-based semantics in graph-based algorithms. In *Proc. AAAI '96*, pages 607–613, 1996.

[12] R. Dechter and J. Pearl. Tree Clustering Schemes for Constraint Processing. In *Proc. AAAI '88*, 1988.

[13] R. Dechter and I. Rish. Directional resolution: The Davis-Putnam procedure, revisited. In *Proc. KR '94*, pages 134–145. Morgan Kaufmann, 1994.

[14] A. del Val. A new method for consequence finding and compilation in restricted language. In *Proc. AAAI '99*, pages 259–264. AAAI Press/MIT Press, 1999.

[15] J. Denzinger and I. Dahn. Cooperating theorem provers. In W. Bibel and P. Schmitt, editors, *Automated Deduction. A basis for applications.*, volume 2, chapter 14, pages 383–416. Kluwer, 1998.

[16] F. Giunchiglia. GETFOL manual - GETFOL version 2.0. Technical Report DIST-TR-92-0010, DIST - University of Genoa, 1994. Available at http://ftp.mrg.dist.unige.it/pub/mrg-ftp/92-0010.ps.gz.

[17] K. Inoue. Linear resolution for consequence finding. *Artificial Intelligence*, 56(2-3):301–353, Aug. 1992.

[18] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

[19] R. C.-T. Lee. *A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms*. PhD thesis, University of California, Berkeley, 1967.

[20] R. C. Lyndon. An interpolation theorem in the predicate calculus. *Pacific Journal of Mathematics*, 9(1):129–142, 1959.

[21] P. Marquis. Consequence finding algorithms. In *Algorithms for Defeasible and Uncertain Reasoning*, volume 5 of *Handbook on Deafeasible Reasoning and Uncertainty Management Systems*, pages 41–145. Kluwer, 2000.

[22] S. McIlraith and E. Amir. Theorem proving with structured theories (full report). Technical Report KSL-01-04, KSL, Computer Science Dept., Stanford U., Apr. 2001.

[23] E. Minicozzi and R. Reiter. A note on linear resolution strategies in consequence-finding. *Artificial Intelligence*, 3:175–180, 1972.

[24] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, Oct. 1979.

[25] I. Rish and R. Dechter. Resolution versus search: two strategies for SAT. *Journal of Automated Reasoning*, 24(1-2):225–275, 2000.

[26] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. of the ACM*, 12(1):23–41, 1965.

[27] J. R. Slagle. Interpolation theorems for resolution in lower predicate calculus. *J. of the ACM*, 17(3):535–542, July 1970.