Strategies for Focusing Structure-Based Theorem Proving

Eyal Amir [1] Sheila McIlraith [2]

Stanford University, Computer Science Department, Gates Building 2A, Stanford, CA 94305-9020, USA

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 while minimizing the number of inferences done within each subtheory and the number of messages sent between subtheories. We do so 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 our algorithms.

1 Introduction

Theorem provers are becoming increasingly prevalent as query-answering machinery for reasoning over single or multiple large commonsense knowledge bases (KBs) [Amir and McIlraith, 2000]. Commonsense KBs, as exemplified by Cycorp's Cyc (e.g., [Lenat, 1995]) and the High Performance Knowledge Base (HPKB) systems developed by Stanford's Knowledge Systems Lab (KSL) (e.g., [Fikes and Farquhar, 1999]) and by SRI (e.g.,[Cohen et al., 1998]), often comprise tens/hundreds of thousands of logical axioms, embodying loosely

[1] E-mail: eyal.amir@cs.stanford.edu
[2] E-mail: sheila.mcilraith@cs.stanford.edu

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., [Jensen et al., 1990]), Constraint Satisfaction Problems (CSPs) (e.g., [Dechter and Pearl, 1988]) and most recently in logical reasoning ([Amir and McIlraith, 2000,Darwiche, 1996], and [Rish and Dechter, 2000] are examples). 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 [Amir and McIlraith, 2000,Amir, 2001].) 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.

This paper builds on the work presented in [Amir and McIlraith, 2000] and in [Amir and McIlraith, 2001]. In that work we introduced the notion of partition-based logical reasoning (PBLR), and proposed a set of associated message-passing algorithms for query answering and for satisfiability checking. We defined criteria for a partitioning that would optimize the efficiency of reasoning. Finally, we proposed a graph-based algorithm to decompose a theory into appropriate partitions, following the proposed criteria. In this paper, we pursue two aspects of PBLR in much greater depth. We tailor PBLR to first-order resolution-based theorem proving, comparing it to ordering strategies for resolution. Further, we develop on strategies that exploit structure inherent in theories to minimize and focus inference, leading to appreciable reductions in the number of inferences performed.

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 [Amir and McIlraith, 2000] to a larger class of local reasoning procedures, and to focused inference in a sublanguage dictated by the partitioning. 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 present algorithms that can use these compiled theories to either process partitions one after the other in batch, or to perform concurrent message passing. We 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.

2 Partition-Based Logical Reasoning

To perform theorem proving with structured theories or with multiple distributed theories, we adopt the basic framework of partition-based logical reasoning [Amir and McIlraith, 2001,Amir and McIlraith, 2000]. In this section, we review partition-based logical reasoning framework. We also extend it with new soundness and completeness results that enable us to minimize local inference.

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 [Amir and McIlraith, 2000].)

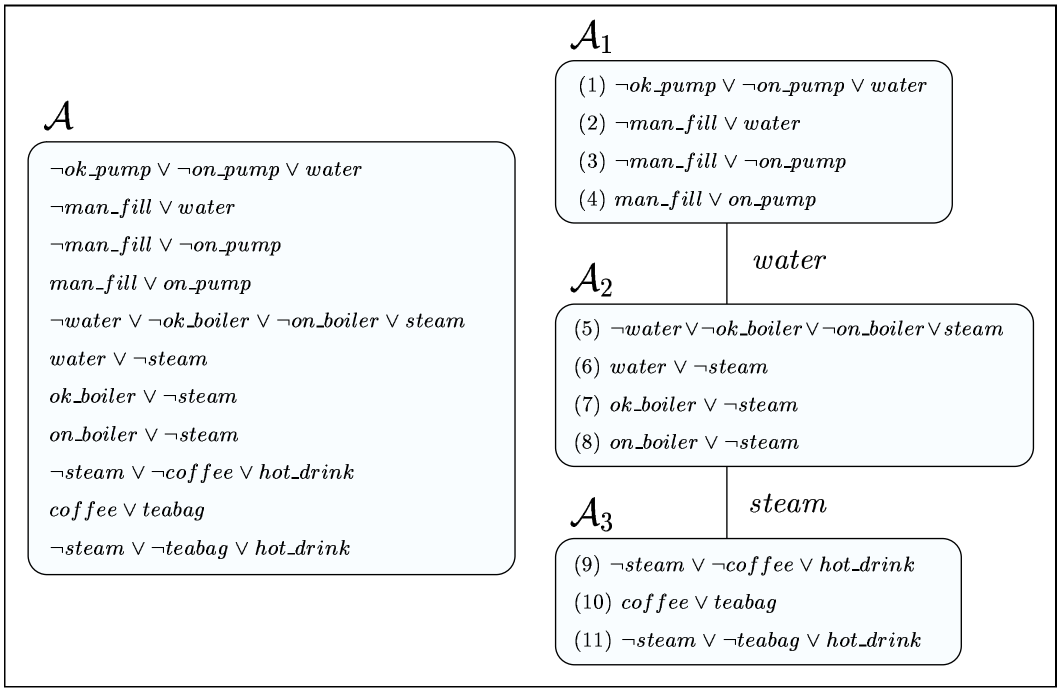
$\mathcal{A}$
$\neg ok\_pump \lor \neg on\_pump \lor water$
$\neg man\_fill \lor water$
$\neg man\_fill \lor \neg on\_pump$
$man\_fill \lor on\_pump$
$\neg water \lor \neg ok\_boiler \lor \neg on\_boiler \lor steam$
$water \lor \neg steam$
$ok\_boiler \lor \neg steam$
$on\_boiler \lor \neg steam$
$\neg steam \lor \neg coffee \lor hot\_drink$
$coffee \lor teabag$
$\neg steam \lor \neg teabag \lor hot\_drink$
$\mathcal{A}_1$
(1) $\neg ok\_pump \lor \neg on\_pump \lor water$
(2) $\neg man\_fill \lor water$
(3) $\neg man\_fill \lor \neg on\_pump$
(4) $man\_fill \lor on\_pump$
$water$
$\mathcal{A}_2$
(5) $\neg water \lor \neg ok\_boiler \lor \neg on\_boiler \lor steam$
(6) $water \lor \neg steam$
(7) $ok\_boiler \lor \neg steam$
(8) $on\_boiler \lor \neg steam$
$steam$
$\mathcal{A}_3$
(9) $\neg steam \lor \neg coffee \lor hot\_drink$
(10) $coffee \lor teabag$
(11) $\neg steam \lor \neg teabag \lor hot\_drink$

Fig. 1. A partitioning of $\mathcal{A}$ and its intersection graph, G.

Figure 2 displays FORWARD-M-P (MP), 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 [Lee, 1967] to be the problem of finding all nontautological logical consequences of a theory or sentences that subsume them. A prime implicate generator is a popular example of a consequence finder[3].

To determine the direction in which messages should be sent in the graph G,

[3] 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.

step 1 in MP 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 $dist(i,k)$ ($i,k \in V$) be the length of the shortest path between nodes i,k in G. Then $i \prec j$ iff $dist(i,k) < dist(j,k)$.*

```
PROCEDURE FORWARD-M-P (MP)($\{\mathcal{A}_i\}_{i \leq n}$, $G$, $Q$)
$\{\mathcal{A}_i\}_{i \leq n}$ 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 $\varphi$ of $\mathcal{A}_j$
          found (or $\varphi$ in $\mathcal{A}_j$), if $\varphi \in \mathcal{L}(l(i,j))$, then add $\varphi$ 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.

Fig. 2. A forward message-passing algorithm.

Figure 3 illustrates an execution of the MP 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, [Amir and McIlraith, 2000] showed that partition-based reasoning reduces the search space significantly, as a function of the size of the communication language between partitions.

MP 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. Also, [Amir and McIlraith, 2000] presented an algorithm called BREAK-CYCLES that transforms every intersection graph that is not tree into a properly labeled tree. 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*

Using MP to prove *hot_drink*

Partition	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)

Fig. 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 $\mathcal{A}_2$.

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 MP($\{\mathcal{A}_i\}_{i\leq n}$, G, Q) outputs YES.*

Proof See Appendix A.1. ■

The completeness in the last theorem relies on Craig's interpolation theorem.

Theorem 2.5 (Craig's Interpolation Theorem [Craig, 1957]) *If $\alpha \vdash \beta$, then there is a formula γ involving only symbols common to both α and β such that $\alpha \vdash \gamma$ and $\gamma \vdash \beta$.*

Our soundness and completeness result improves upon a soundness and completeness result in [Amir and McIlraith, 2000] by allowing consequence finders that focus on consequences in the communication language between partitions.

In certain cases, we can restrict consequence finding in MP even further by using reasoners that are complete for $\mathcal{L}$-*consequence finding.*

Definition 2.6 (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*[4] *φ.*

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 [del Val, 1999]. The notion of a consequence finder restricting consequence generation to consequences in a designated sublanguage was discussed by Inoue [Inoue, 1992], and further developed by del Val [del Val, 1999] 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 MP reasoners in Theorem 2.4 must be complete for $\mathcal{L}_i$-generation in arbitrary FOL languages, $\mathcal{L}_i$. Corollary 2.7 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 2.7 (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 MP($\{\mathcal{A}_i\}_{i \leq n}$, G, Q) outputs YES.*

In Sections 3, 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 Strategies

In this section, we specialize our message-passing algorithms with consequence finders that specifically employ *resolution* and several of its refinements. We focus on the first-order case of resolution and the restriction strategies of *linear resolution*, *set-of-support resolution* and some of their variants. Also, we relate MP to resolution strategies that use order on literals or symbols.

For background material on resolution and resolution strategies, the reader is

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

referred to [Chang and Lee, 1973,Loveland, 1978,Eisinger and Ohlbach, 1993] and [Genesereth and Nilsson, 1987].

3.1 Resolution Message-Passing

Resolution [Robinson, 1965] is one of the most widely used reasoning methods for automated deduction, and more specifically for consequence finding. The resolution rule is complete for *clausal* consequence finding. It requires the input formula to be in clausal form, i.e., a conjunction of disjunctions of unquantified literals. For general first-order formulae, a transformation to clausal form (e.g., [Lloyd and Topor, 1985]) includes *Skolemization*, which eliminates quantifiers and possibly introduces new constant symbols and new function symbols.

We present algorithm RESOLUTION-M-P (RES-MP), which 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 across partitions: the first implementation is for clausal propositional theories; the second is for clausal FOL theories, with associated graph G, which is a properly labeled tree and whose labels include all the function and constant symbols of the language; the third is also for clausal FOL theories, but it uses *unskolemization* and subsequent Skolemization to generate the messages to be passed across partitions; the fourth is a refinement of the third for the same class of theories that avoids unskolemization.

```
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
          $\varphi$ (as input or resolvent) and the predicates of $\varphi$ are in $\mathcal{L}(l(i, j))$, then
          perform RES-SEND($\varphi$, $j$, $i$).
      (c) If $Q$ is proven in $\mathcal{A}_k$, return YES.
```

Fig. 4. A resolution forward message-passing algorithm.

In the propositional case, subroutine RES-SEND(φ, j, i) (Implementation 1) simply adds φ to $\mathcal{A}_i$, as done in MP. If the resolution strategies being employed, $\{\mathfrak{R}_i\}_{i\leq n}$, are complete for $\mathcal{L}_i$-consequence finding (for $\mathcal{L}_i$ the clausal sublanguage of $\mathcal{L}(l(i,j))$, for $j \prec i$ and $(i,j) \in E$), then RES-MP is sound and complete. This is because completeness for $\mathcal{L}_i$-consequence finding for the clausal sublanguage $\mathcal{L}_i$ of $\mathcal{L}(l(i,j))$ implies completeness for $\mathcal{L}(l(i,j))$-consequence finding, if $l(i,j)$ is propositional.

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 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 [Slagle, 1970])). 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 [Craig, 1957] that applies for resolution [Slagle, 1970].

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 MP.

In large systems that consist of many partitions, the addition of so many constant and function symbols to each for 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 [Luckham and Nilsson, 1971,Bledsoe and Ballantyne, 1978] and [Cox and Pietrzykowski, 1984]. [Chadha and Plaisted, 1993] presents an algorithm U that is complete for our purposes and generalizes and simplifies an algorithm of [Cox and Pietrzykowski, 1984].

Theorem 3.1 ([Chadha and Plaisted, 1993]) *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 MP), 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($\varphi$, $j$, $i$)                                (Implementation 3)
$\varphi$ a formula, $j, i \leq n$.

  (1) Unskolemize $\varphi$ into a set of formulae, $\Phi$ 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 $\varphi_2$ is not subsumed by a clause that is in $\mathcal{A}_i$, then add
      the Skolemized version of $\varphi_2$ to the set of axioms of $\mathcal{A}_i$.
```

Fig. 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.

Steps (2) and (3) of procedure RES-SEND(φ, j, i) (Implementation 4) correspond to similar steps in procedure U presented in [Chadha and Plaisted, 1993], 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 (implementation 4) 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 $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned theory of propositional or first-order clauses, G a tree that is properly labeled with respect to $\mathcal{A}$, and $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leq n$, be a sentence that is the query. $\mathcal{A} \models Q$ iff applying RES-MP($\{\mathcal{A}_i\}_{i \leq n}$, G, Q) (with Implementation*

```
PROCEDURE RES-SEND($\varphi$, $j$, $i$)                                    (Implementation 4)
$\varphi$ 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 $\varphi$, if $f \in S$ and $t$ is not a
      subexpression of another term $t' = f'(t'_1, ..., t'_{k'})$ of $\varphi$ 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 $\varphi$, if $\alpha, \beta$ are unifiable, then unify all occurrences of $x, y$ (i.e., unify
      $\alpha_i, \beta_i$ for all markings $x \leftarrow \alpha_i$, $y \leftarrow \beta_i$).
  (4) For every marked argument "$x \leftarrow \alpha$" in $\varphi$,
      (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 $\varphi$, for a fresh function symbol $f$
          (if $r = 0$, $f$ is a fresh constant symbol).
  (5) Add $\varphi$ to $\mathcal{A}_i$.
```

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

Fig. 6. Subroutine RES-SEND without unskolemization.

4 of RES-SEND) outputs YES.

Proof See Appendix A.2.

3.2 Linear Resolution

In this section we show how linear resolution, a sound and complete resolution strategy for consequence finding, can be exploited for first-order MP.

For a set of axioms $\mathcal{A}$, *linear resolution* [Loveland, 1970,Yates et al., 1970], [Luckham, 1970,Anderson and Bledsoe, 1970] restricts the search space of resolution by allowing only resolutions in which one of the clauses is part of the initial set of clauses (the *input*) or those in which one of the clauses is an ancestor of the other in the resolution tree. Linear resolution is refutation-complete. It is also complete for consequence finding [Minicozzi and Reiter, 1972].

Figure 7 presents LIN-MP, our procedure for forward message passing using linear restriction in each of the partitions. Backward message passing with linear resolution is analogous. LIN-MP takes as input a partitioned theory $\mathcal{A}$, a graph G and a query Q, and returns YES if it can prove Q.

```
PROCEDURE LIN-MP($\{\mathcal{A}_i\}_{i\leq n}$, $G$, $Q$)
$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a graph, $Q$ a query
formula in the language of $\mathcal{L}(\mathcal{A}_k)$ $(k \leq n)$.
  (1) Let $dist(i, j)$ $(i, j \in V)$ be the length of the shortest path between $i, j$
      in $G$. Let $i \prec j$ iff $dist(i, k) < dist(j, k)$ ($\prec$ is a strict partial order).
  (2) Concurrently,
      (a) For every $(i, j) \in E$ such that $i \prec j$, perform linear resolution for
          $\mathcal{A}_i$.
      (b) If at any point $\mathcal{A}_j \models \varphi$ is proven and $\varphi$'s signature is in $\mathcal{L}(l(i, j))$,
          then add $\varphi$ to the axioms of $\mathcal{A}_i$, considering $\varphi$ as input for further
          linear resolutions.
      (c) If $Q$ is proven in $\mathcal{A}_k$, return YES.
```

Fig. 7. Linear resolution with forward message passing.

Theorem 3.3 (Soundness and Completeness) *Let $\mathcal{A} = \bigcup_{i\leq n} \mathcal{A}_i$ be a partitioned theory with a tree G that is properly labeled for $\mathcal{A}$. Let $k \leq n$ and φ be a sentence whose signature is in $\mathcal{L}(\mathcal{A}_k)$. $\mathcal{A} \models \varphi$ iff applying LIN-MP outputs YES.*

Proof Since linear resolution is complete for clausal consequence finding [Minicozzi and Reiter, 1972], it is enough to show that the way messages are handled does not jeopardize this completeness. In LIN-MP, messages are treated as *input clauses* in the receiving partition. Thus, both the original input clauses and the messages are considered *input clauses*. This means that valid consequences of the original set of clauses and the received messages are generated by linear resolution in each partition. This guarantees completeness for consequence generation in each partition, and the theorem follows from Theorem 3.2. ■

It is also possible to define a more restrictive version of message passing that exploits linear resolution. Instead of treating the messages as inputs in the receiving partition, we may treat them as resolved clauses (i.e., non-input clauses). It is not clear if this restriction can be made complete. It is more restrictive than simple linear resolution in that two clauses may not be resolved against each other even though one is a descendent of the other.

There are variants of linear resolution that include ordering strategies such as A-ordering (e.g., [Reiter, 1971]), C-ordering (e.g., [Reiter, 1971]) and Model Elimination [Loveland, 1969,Stickel, 1988] (more on ordering strategies in Section 3.4). All of these ordering-based linear resolution strategies turn out to be incomplete for consequence finding, despite being refutation-complete. Nevertheless, linear resolution with A-ordering can be made $\mathcal{L}$-generation complete for the necessary $\mathcal{L}$ in a way similar to that used in Section 3.4. Also, the rest of the strategies can be made complete for an incremental version of $\mathcal{L}$-generation [Inoue, 1992]. We use this observation in Section 5.

3.3 Set-of-Support and Semantic Resolution

In this section we examine the relevance of set-of-support and semantic resolution to our message-passing algorithms. In the general case, semantic resolution and set-of-support resolution are both incomplete for consequence finding. Nevertheless, we are able to propose an algorithm for sound and complete *propositional* forward message passing using semantic resolution or set-of-support.

For a set of axioms $\mathcal{A}$, *set-of-support resolution* [Wos et al., 1965] restricts the search space of resolution by distinguishing a set $S \subset \mathcal{A}$ as a *support* and disallowing resolutions between axioms in $\mathcal{A} \setminus S$. Resolvents (between clauses in S or between a clause in $\mathcal{A} \setminus S$ and a clause in S) are added to S. A *set-of-support refutation* is such a sequence of resolutions leading to the empty clause ($\{\}$). This algorithm is refutation-complete if $\mathcal{A} \setminus S$ is consistent. Typically, when trying to prove $T \vdash \varphi$, we assume that T is consistent and put $\neg\varphi$ in the set of support.

Semantic resolution [Slagle, 1967] is similar to set-of-support resolution in that it divides clauses into two distinguished sets, S_1, S_2. It differs from set-of-support in two main ways. First, it requires an interpretation I to be supplied. This interpretation is used to determine the division of axioms into the two sets. S_1 includes clauses that I satisfies, and S_2 includes clauses that I does not satisfy. Second, it allows resolution of two clauses only if each comes from a different set (set-of-support resolution allows resolutions between clauses in the support). Resolvents are added to the proper set (S_1 if I satisfies them, or S_2 otherwise). A *semantic I-refutation* (with an interpretation I) is such a sequence of resolutions, leading to the empty clause ($\{\}$).

The following theorem gives us some insight into the applicability of both semantic resolution and set-of-support resolution as reasoning procedures in our message-passing algorithms. From this theorem, we are able to propose an algorithm for sound and complete forward message passing using semantic resolution or set-of-support. The theorem holds for both FOL and propositional logic.

Theorem 3.4 ([Slagle et al., 1969]) *Let C be a clause, S be a finite set of clauses and I be an interpretation in $\mathcal{L}(S \cup \{C\})$. Assume C is a prime implicate of S. If $I \not\models C$, then there is a semantic I-deduction of C from S.*

This theorem implies a restricted form of completeness for consequence finding: Given an interpretation, I, semantic resolution will generate all the prime implicates that are *falsified* by I. Semantic resolution is a more restricted strategy than set-of-support (given the right interpretation, I), so this theorem applies for set-of-support as well.

Figure 8 presents SEM-MP, our procedure for *propositional* forward message-passing using semantic restriction in each of the partitions. Backward message-passing with semantic resolution is analogous. It takes as input a partitioned theory $\mathcal{A}$, a graph G, and a query, Q, and returns YES if it proves Q.

```
PROCEDURE MULTI-SEM($\mathcal{A}$, $M$, $L$)
$\mathcal{A}$ a theory, $M$ an interpretation, $L$ a set of symbols in $\mathcal{L}(\mathcal{A})$.
  (1) For every possible truth assignment, $a$, to $L$, let $M_a$ be an interpretation
      derived from $M$ by changing the truth assignment to the values
      mentioned in $a$. Let $A$ be the set of all those truth assignments.
  (2) For every interpretation in $\{M_a\}_{a \in A}$, let $S_a = \{\varphi \mid \varphi \in \mathcal{A},\ M_a \not\models \varphi\}$.
  (3) For every interpretation $a \in A$, perform semantic resolution with $M_a$.
```

```
PROCEDURE SEM-MP($\{\mathcal{A}_i\}_{i \leq n}$, $\{M_i\}_{i \leq n}$, $G$, $Q$)
$\{\mathcal{A}_i\}_{i \leq n}$ a partitioning of a theory $\mathcal{A}$, $\{M_i\}_{i \leq n}$ interpretations in languages
corresponding to the partitioning of $\mathcal{A}$, $G = (V, E, l)$ a graph, $Q$ a query
formula in the language of $\mathcal{L}(\mathcal{A}_k)$ $(k \leq n)$.
  (1) Let $dist(i,j)$ $(i,j \in V)$ be the length of the shortest path between $i,j$
      in $G$. Let $i \prec j$ iff $dist(i,k) < dist(j,k)$ ($\prec$ is a strict partial order).
  (2) Concurrently,
      (a) For every $(i,j) \in E$ such that $i \prec j$, perform MULTI-SEM($\mathcal{A}_i$,
          $M_i$, $l(i,j)$).
      (b) If at any point $\mathcal{A}_j \models \varphi$ is proven and $\varphi$'s signature is in $\mathcal{L}(l(i,j))$,
          then add $\varphi$ to $\mathcal{A}_i$ (putting it in the correct set for each concurrently
          running semantic resolution).
      (c) If $Q$ is proven in $\mathcal{A}_k$, return YES.
```

Fig. 8. Semantic resolution with forward message passing.

The reason this algorithm applies only to the propositional case is that Step (1) in procedure MULTI-SEM requires us to enumerate all interpretations over $l(i,j)$. Procedure SEM-MP applies semantic resolution in MP in a way that uses several complementary interpretations. For each message that may be sent, we keep an interpretation that does not satisfy it. For example, if an edge label includes l propositional symbols, we need to keep 2^l interpretations (each one does not satisfy a clause of size l in the language of the link). If an interpretation satisfies all the clauses, then we do not need to consider it in running our semantic resolution. This completes the procedures for consequence finding and makes SEM-MP complete.

Theorem 3.5 (Soundness and Completeness) *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned theory with the properly labeled tree G. Let $k \leq n$ and φ be a sentence whose signature is in $\mathcal{L}(\mathcal{A}_k)$. $\mathcal{A} \models \varphi$ iff applying SEM-MP outputs YES.*

Proof Procedure MULTI-SEM generates all the consequences in the language in which messages need to be sent, according to Theorem 3.4. By Theorem 3.2, this procedure is sound and complete. ■

It is not clear if there is an algorithm that corresponds more closely to set-of-support resolution that is also complete.

3.4 Directional Resolution, A-ordering and Lock Resolution

So far we have examined resolution strategies, demonstrating their applicability to our message-passing algorithms. The results in this section serve to place our message-passing algorithms in a broader context by relating them to some other resolution strategies. We examine three resolution strategies that are *incomplete* for consequence finding: directional resolution, A-ordering resolution and lock resolution. These strategies use orderings on symbols/literals to restrict the space of possible resolutions.

We present two interesting results relating these resolution strategies to our message-passing algorithms. First, we show that our forward message-passing algorithm, MP, can simulate the symbol ordering strategies of directional resolution, A-ordering and lock resolution if the orders correspond to orders on predicate/propositional symbols. Second, we show that in the propositional case directional resolution, A-ordering and lock resolution can likewise simulate MP, if MP uses unrestricted or directional resolution in each of the partitions. In the FOL case, the same result holds if all the function and constant symbols are on all the links between partitions.

Directional resolution[5] [Davis and Putnam, 1960,Dechter and Rish, 1994] is a procedure for concluding satisfiability of a propositional logic theory. For a set of propositional axioms $\mathcal{A}$ and an order on the propositional symbols $Q_1, ..., Q_n$ (Q_n is the *highest* symbol), directional resolution *resolves out* each of the propositions in order, if it can. Dechter and Rish [Dechter and Rish, 1994] present directional resolution using the framework of bucket elimination: Partition the clauses of $\mathcal{A}$ into $bucket_1, ..., bucket_n$, where $bucket_i$ contains all the clauses whose *highest* literal is Q_i. For $i = n$ to 1, perform unit resolution on $bucket_i$ and perform all resolutions possible on Q_i in $bucket_i$, putting the resolvents into the proper buckets. $\mathcal{A}$ is satisfiable iff the empty clause was not

[5] This name first appeared for this strategy in [Dechter and Rish, 1994].

generated in any bucket. Directional resolution is complete for satisfiability.

A-ordering (atom ordering) [Reynolds, 1965,Kowalski and Hayes, 1969] is a FOL version of directional resolution. It accepts a total order on literals and uses it as follows: Resolving clauses C_1, C_2 is allowed only if the resolved literal l is the *highest* literal in both[6] C_1 and C_2. Typically, the order is given over predicate symbols (instances of each predicate are equivalent for this order), but it can be given over all atoms if the order satisfies some natural properties (see [Loveland, 1978]). A-ordering is refutation-complete. It is sometimes combined with other strategies (e.g., linear resolution, semantic resolution, etc.), with modifications that allow the preservation of refutation-completeness.

Lock resolution [Boyer, 1971] is another resolution strategy for first-order theories. It generalizes both directional resolution and A-ordering. In a theory $\mathcal{A}$, every literal instance is given an index. The same literal appearing in two different clauses may receive a different index for each instance. Resolution is allowed between two clauses C_1, C_2 only upon their respectively *lowest* literals (literals with *lowest* index in the clause). The literals in the resolvent keep their original index (even if they actually changed due to unification). If two identical literals appear in the resolvent, only the one with lowest index is kept. Lock resolution is refutation-complete.

Note that none of these ordering-based resolution strategies is complete for consequence finding. They are not even $\mathcal{L}$-generation complete, in the general case. For example, $\{p \vee q, p \vee \neg q\} \vdash p$, but we will not conclude p if the order on propositional symbols requires our algorithm to resolve upon p first. Nevertheless, these strategies are closely related to message passing. For any given partitioned theory and graph, an order can be chosen that makes these *complete for $\mathcal{L}_i$-generation* (as introduced in Section 2), for the needed languages $\mathcal{L}_i$. Thus, such an order make these strategies suitable for inclusion in MP, for some or all the partitions. Also, for any order, both directional resolution and A-ordering can be simulated by MP, and for many indices, the lock resolution can also be simulated by MP. We first present the following useful lemma:

Lemma 3.6 *Let $\mathfrak{R}_P$ be a FOL resolution procedure that resolves two clauses C_1, C_2 only if each of them includes a literal containing P, the two respective literals are unifiable, and the resolution is done on these literals. Let $\mathcal{L}_P$ be the language including all the symbols of $\mathcal{A}$ besides P. Then, $\mathfrak{R}_P$ is $\mathcal{L}_P$-generation complete.*

Proof See Appendix A.3. ■

[6] In some places (e.g., [Chang and Lee, 1973]), the reference A-ordering is made to a modified procedure in which we are not required to resolve on the highest literal in C_2; this is not the case with the original procedure.

This lemma implies that the ordering-based resolution strategies we describe above can all be used as partition reasoners in MP (and its variants). To ensure completeness of MP, we make sure that the reasoner in $\mathcal{A}_j$ is $\mathcal{L}(l(i,j))$-generation complete for the right i (determined by the query and its partition). Lemma 3.6 says that we only need to make sure the order is such that the symbols of $l(i,j)$ are last in the order and have equal precedence.

In the following, we consider the resolutions allowed by each of the resolution strategies, assuming they are run indefinitely. We do not consider which proved the goal first, but rather we examine the respective search spaces.

Theorem 3.7 (MP Simulates Orderings) *The following relationships hold between the MP algorithm and the ordering strategies of directional resolution, A-ordering and lock resolution:*

(1) Let $\mathcal{A}$ be a propositional theory and $\leq_A$ a total order on its n propositional symbols. Then, there is a partitioning $\{\mathcal{A}_i\}_{i\leq n}$ of $\mathcal{A}$, a graph G and partition reasoners that are generation-set complete such that running MP does not perform more resolutions than directional resolution (alternatively, A-ordering) of $\mathcal{A}$ with order $\leq_A$.
(2) Let $\mathcal{A}$ be a FOL theory and $\leq_A$ a total order on its n predicate symbols. Then, there is a partitioning of $\mathcal{A}$ into $\{\mathcal{A}_i\}_{i\leq n}$, a graph G and partition reasoners that are generation-set complete such that running MP does not perform more resolutions than A-ordered resolution of $\mathcal{A}$ with order $\leq_A$.
(3) Let $\mathcal{A}$ be a FOL theory and I an indexing of its literal instances. Let $n = max_{l\ literal} I(l)$. Assume that $I(l_1) = I(l_2)$ if l_1, l_2 have the same predicate symbol. Then, there is a partitioning $\mathcal{A} = \bigcup_{i\leq n} \mathcal{A}_i$ and partition reasoners that are generation-set complete, such that running MP does not perform more resolutions than lock resolution of $\mathcal{A}$ with index I.

Proof See Appendix A.4.

Assuming we have a propositional theory, we use unrestricted resolution for each partition in MP. Then, directional resolution and lock resolution can be used to search a proof space that is no larger than that used by MP.

Theorem 3.8 (Orders Simulate MP) *The following relationships hold between the MP algorithm and the ordering-based resolution strategies of directional resolution and lock resolution:*

(1) Let $\mathcal{A} = \bigcup_{i\leq n} \mathcal{A}_i$ be a partitioned propositional theory and $G(V,E,l)$ be a tree that is properly labeled for $\mathcal{A}$. Then, there is a total order, $\leq_A$, on $\mathcal{A}$'s propositional symbols such that if a clause C is a consequence of directional resolution of $\mathcal{A}$ with order $\leq_A$, then C is a consequence of running MP on this partitioning using unrestricted resolution in each partition.

(2) *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned FOL theory and $G(V, E, l)$ be a tree that is properly labeled for $\mathcal{A}$. Then, there is a total order, $\leq_A$, on $\mathcal{A}$'s predicate symbols such that if a clause C is a consequence of A-ordered resolution of $\mathcal{A}$ with order $\leq_A$, then C is a consequence of running MP on this partitioning using unrestricted resolution in each partition.*

(3) *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned propositional theory and $G(V, E)$ be a tree that is properly labeled for $\mathcal{A}$. Then, there is an index, I, on $\mathcal{A}$'s literal instances such that if a clause C is a consequence of lock resolution of $\mathcal{A}$ with index I, then C is a consequence of running MP on this partitioning using unrestricted resolution in each partition.*

Proof See Appendix A.5.

Theorem 3.8 does not hold for neither A-ordered resolution or lock resolution in the FOL case, if we do not include all the function and constant symbols on the links in G. Let $C_1 = P(B, x), C_1' = P(C, y), C_2 = \neg P(z, D)$, have the partitions $\mathcal{A}_1 = \{C_1, C_1'\}$, $\mathcal{A}_2 = \{C_2\}$, and have $k = 2$ (i.e., messages go from $\mathcal{A}_1$ to $\mathcal{A}_2$). Lock resolution would always allow resolving C_1, C_2 and C_1', C_2. MP will have the intersection language include only P, so the only message sent to $\mathcal{A}_2$ from $\mathcal{A}_1$ is $\exists b P(b, x)$. This will make $\mathcal{A}_2$ resolve only one sentence against C_2 instead of two.

Finally, [Amir and McIlraith, 2000] discussed the relationship and limitations of MP, the length of interpolants required for proofs in MP. Our results above (variable ordering can be simulated by MP (Theorem 3.7)) combined with that discussion show some potential limitations to using variable ordering strategies for resolution in general. If the interpolants needed for the proof with MP are large, variable ordering (as in directional resolution) may lead us to spend significantly more time than we could have spent had we not used that order (or any other order). In fact, in the propositional case it is an open question whether or not the size of the smallest interpolant can be polynomially bounded by the size of the two formulae α, β. A positive answer to this question would imply an important consequence in complexity theory, namely that $NP \cap coNP \subseteq P/poly$ [Boppana and Sipser, 1990].

4 Minimizing Node Coupling Using Polarity

MP and RES-MP use the communication language to determine relevant inference steps between formulae in connected partitions. This section improves the efficiency of MP 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 [Lyndon, 1959], an extension to Craig's Theorem [Craig, 1957].

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 MP 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 MP 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 9) 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.

```
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 $G' = (V, E, l')$.
```

Fig. 9. Constraining the communication language of $\{\mathcal{A}_i\}_{i \leq n}$ by exploiting polarity.

Theorem 4.2 (Soundness and Completeness of POLARIZE + MP)

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 MP($\{\mathcal{A}_i\}_{i\leq n}$, G',Q) outputs YES.

Procedure POLARIZE can be combined with MP in a more dynamic fashion, yielding further restriction on the communication language. For example, step (4b) may determine $l'(i,j)$ such that we may need to send $P \vee Q$ from $\mathcal{A}_j$ to $\mathcal{A}_i$ when running MP on this theory and query. It may do so by adding P (positively) to $l'(i,j)$ because $\neg P$ appears in $\mathcal{A}_r$, for some $r \in V_2$. However, if we determine those polarities on the vocabulary of the links dynamically, and $r \not\prec j$ then r is not *on the path to the goal partition*, so we may still not have to send this message from $\mathcal{A}_j$ to $\mathcal{A}_i$. Only if later there is another partition, $\mathcal{A}_s$, that is on the path to the goal such that $\neg P$ appears in it (e.g., after $\mathcal{A}_r$ sent a message with $\neg P$ that arrived to $\mathcal{A}_s$), then we will allow this message to be sent from $\mathcal{A}_j$ to $\mathcal{A}_i$.

Darwiche [Darwiche, 1996] proposed a weaker 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, 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. Restricting reasoning to $\mathcal{L}$-consequence finding in the output communication language of a partition is not sufficient since $\mathcal{L}$-consequence finding can be achieved in some cases by general consequence finding followed by filtering for consequences in $\mathcal{L}$. We require algorithms that exploit $\mathcal{L}$ to minimize the number of deduction steps being performed.

In the propositional case, the most popular algorithms for performing focused consequence finding are certain $\mathcal{L}$-(prime) implicate finders. (See [Marquis, 2000] for an excellent survey.) SOL-resolution (skipping ordered linear resolution) [Inoue, 1992] and SFK-resolution (skip-filtered, kernel resolution) [del Val, 1999] are two first-order resolution-based $\mathcal{L}$-consequence finders. SFK-resolution is complete for first-order $\mathcal{L}$-consequence finding, and it reduces to Directional Resolution (see Section 3.4) in the propositional case. 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 2.6.

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 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. (A restricted class of first-order formulae for which we do not encounter this problem is monadic logic without function symbols [Marquis, 2000,Ayeb et al., 1993]). Consequently the proposed use of these particular algorithms is limited to the propositional case unless otherwise noted.

The first strategy is *compilation* of the theories in individual partitions into theories comprised solely of the minimal consequences of all the communication languages associated with an individual partition, $\mathcal{L}_{comm_i}$. Figure 10 presents COMPILE($\{\mathcal{A}_i\}_{i \leq n}$, G), an algorithm 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 $\mathcal{A}_i$. Since all messages received and sent by the partition are drawn from $\mathcal{L}_{comm_i}$, the compiled theory is adequate. Theorem 5.1 proves that MP is sound and complete with the compiled theory. Observe that this compilation is query independent, as long as the query can be expressed in the language of an existing partition. 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.

Theorem 5.1 (Soundness and Completeness of COMPILE + MP) *Let $\mathcal{A}$ be a partitioned theory $\{\mathcal{A}_i\}_{i \leq n}$ of arbitrary propositional 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*

$\mathcal{A}'$ be the output of COMPILE($\mathcal{A}, G$), with $\mathcal{A}_k$ substituted for $\mathcal{A}'_k$. Let $\{\Re_i\}_{i\leq n}$ be the $\mathcal{L}_i$-consequence finders associated with partitions $\{\mathcal{A}'_i\}_{i\leq n}$. If every $\Re_i$ is complete for $\mathcal{L}_i$-consequence finding then $\mathcal{A} \models Q$ iff MP($\{\mathcal{A}'_i\}_{i\leq n}$, G, Q) outputs YES.

PROOF See Appendix A.6. ■

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 $O(2^{|L(\mathcal{L}_{comm_i})|})$ clauses. Since our assumption is that partitions are produced to minimize communication between partitions, $|L(\mathcal{L}_{comm_i})|$ should be much smaller than $|L(\mathcal{A}_i)|$. As a consequence, we might expect the compiled theory to be smaller than the original theory, though this is not guaranteed (for example, see [Schrag and Crawford, 1996]). Under the further assumption that the theories in partitions are fairly static, the cost of compilation will be amortized over many queries. There are other compilation methods proposed by del Val and others that may provide a more parsimonious compilation than prime implicates. We leave this to future work.

```
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 $\mathcal{A}'_i$.
```

Fig. 10. A partition-based theory compilation algorithm.

Proposition 5.2 *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 11 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 [Dechter and Rish, 1994]. BATCH-MP preserves soundness and completeness of the global problem, while exploiting focused search within each partition.

```
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.

Fig. 11. A batch-style message-passing algorithm.

Theorem 5.3 (Soundness and Completeness of BATCH-MP) *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 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.*

Proof See Appendix A.7. ■

Our final algorithm, CONCURRENT-MP, (Figure 12), 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 5.4 (Soundness and Completeness of CONCURRENT-MP) *Let $\mathcal{A}$ be a set of clauses in propositional logic. Let $\{\mathfrak{R}_i\}_{i\leq n}$ be the $\mathcal{L}_i$-consequence*

```
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 con-
          sequences 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.

Fig. 12. A concurrent message-passing algorithm.

finders associated with partitions $\{\mathcal{A}_i\}_{i\leq n}$ in step 4 of CONCURRENT-MP 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.

Proof See Appendix A.8. ■

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., [Jensen et al., 1990]) our work is distinguished in that we 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., [Dechter and Pearl, 1988]) and more recently, logical reasoning (e.g., [Darwiche, 1996,Rish and Dechter, 2000]) 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 ([Bonacina and Hsiang, 1994,Denzinger and Dahn, 1998] are surveys of this literature) and to research on combining logical systems (e.g., [Nelson and Oppen, 1979,Baader and Schulz, 1998,Baader and Schulz, 1992], [Shostak, 1984,Ringeissen, 1996,Tinelli and Harandi, 1996]), Most parallel theorem prover implementations are guided by lookahead and subgoals to decompose the search space dynamically [Conry et al., 1990,Cowen and Wyatt, 1993], [Ertel, 1992,Sutcliffe, 1992,Bonacina and Hsiang, 1996,Suttner, 1997], or allow messages to be sent between different provers working in parallel, using heuristics to decide on which messages are relevant to each prover (e.g., the work of [Denzinger and Fuchs, 1999]). 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. Recent work introduced shared function symbols between two theories (e.g., [Ringeissen, 1996]), but no algorithm allows sharing of relation symbols. All approaches either nondeterministically instantiate the (newly created) variables connecting the theories (e.g., [Tinelli and Harandi, 1996]), 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 [Akman and Surav, 1996] 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 [Giunchiglia, 1994,Giunchiglia and Traverso, 1995]).

Finally, as noted previously, our use of focused consequence finding for theorem proving with structure theories is related to work by Inoue [Inoue, 1992] and more recently by del Val [del Val, 1999] on vocabulary-based focused consequence finding, and to work by Kautz and Selman [Kautz and Selman, 1996] on computing the Least Horn Upper Bound (LUB) of a theory using prime implicate generators. Lin [Lin, 2000] also reintroduces related algorithms in the context of computing strongest necessary and weakest sufficient conditions. Our ideas may also combine well with those of [Stickel, 1985] who proposed taking a subtheory and compiling it into rules that can be used with the rest of the theory.

7 Summary

In this paper we used graph-based techniques, together with Craig's and Lyndon's interpolation theorems, to improve the efficiency of theorem proving with structured theories. Our approach was to capture the structure inherent in a logical theory, by partitioning the theory into subtheories, minimally connected by the nonlogical symbols they share. We proposed sound and complete message-passing algorithms over these partitioned theories that focus and minimize logical inference. We specialized these algorithms to resolution theorem proving, comparing our results to other resolution strategies. Most of the algorithms we proposed are applicable to FOL and all are applicable to propositional logic.

Partitioning limits interaction between subtheories, reducing the number of possible inferences. Further, focused consequence finders limit inference within a partition to only those steps necessary for message-passing to adjacent partitions. We adapted message-passing algorithms to resolution and some of its restriction strategies for the case of first-order resolution, and for batch and concurrent theorem proving. We reduced the reasoning done 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 a subtheory in a reduced sublanguage.

We provided an algorithm that restricts the interaction between subtheories by exploiting the polarity of literals. We showed how to use polarity to limit the number of interactions that we allow between partitions and the number of inferences done within each partition separately. This restriction prevents interactions between sentences that may have been resolved together otherwise.

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.

A Proofs

A.1 Theorem 2.4: FORWARD-M-P (MP) is Sound and Complete

First, notice that soundness is immediate because the only rules used in deriving consequences are those used in our chosen consequence-finding procedure (of which rules are sound). In all that follows, we assume $\mathcal{A}$ is finite. The infinite case follows by the compactness of FOL.

Lemma A.1 *Let $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ be a partitioned theory. Let $\varphi \in \mathcal{L}(\mathcal{A}_2)$. If $\mathcal{A} \vdash \varphi$, then $\mathcal{A}_2 \vdash \varphi$ or there is a sentence $\psi \in \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ such that $\mathcal{A}_1 \vdash \psi$ and $\mathcal{A}_2 \vdash \psi \Rightarrow \varphi$.*

Proof of Lemma A.1. We use Craig's interpolation theorem (Theorem 2.5), taking $\alpha = \mathcal{A}_1$ and $\beta = \mathcal{A}_2 \Rightarrow \varphi$. Since $\alpha \vdash \beta$ (by the deduction theorem for FOL), there is a formula $\psi \in \mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$ such that $\alpha \vdash \psi$ and $\psi \vdash \beta$. By the deduction theorem for FOL, we get that $\mathcal{A}_1 \vdash \psi$ and $\psi \wedge \mathcal{A}_2 \vdash \varphi$. Since $\psi \in \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ by the way we constructed α, β, we are done. ■

Proof of Theorem 2.4. we prove the theorem by induction on the number of partitions in the logical theory. For $|V| = 1$ (a single partition), $\mathcal{A} = \mathcal{A}_1$ and the proof is immediate, as $\mathfrak{R}_1$ is refutation complete (every reasoner that is complete for $\mathcal{L}$-generation for some $\mathcal{L}$ is refutation complete) and we added $\neg Q$ to $\mathcal{A}_1$. Assume that we proved the theorem for $|V| \leq n-1$ and we prove the theorem for $|V| = n$.

In G, k has c neighbors, $i_1, ..., i_c$. $(k, i_1) \in E$ separates two parts of the tree G: G_1 (includes i_1) and G_2 (includes k). Let $\mathcal{B}_1, \mathcal{B}_2$ be the subtheories of $\mathcal{A}$ that correspond to G_1, G_2, respectively.

Notice that $Q \in \mathcal{L}(\mathcal{B}_2)$. By Lemma A.1, either $\mathcal{B}_2 \vdash Q$ or there is $\psi \in \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$ such that $\mathcal{B}_1 \vdash \psi$ and $\mathcal{B}_2 \vdash \psi \Rightarrow Q$. If $\mathcal{B}_2 \vdash Q$, then we are done, by the induction hypothesis applied to the partitioning $\{\mathcal{A}_i \mid i \in V_2\}$ (V_2 includes the vertices of G_2) and G_2 (notice that $\prec'$ used for G_2, Q agrees with $\prec$ used for G).

Otherwise, let ψ be a sentence as above. $\bigcup_{(i_1,j)\in E, j\neq k} l(i_1, j) \supseteq L(\mathcal{B}_2 \cup \mathcal{A}_{i_1}) \cap L(\mathcal{B}_1 \backslash \mathcal{A}_{i_1})$ because the set of edges (i_1, j) separates two subgraphs corresponding to the theories $\mathcal{B}_1 \setminus \mathcal{A}_{i_1}$ and $\mathcal{B}_2 \cup \mathcal{A}_{i_1}$, and G is properly labeled for our partitioning. Thus, since $\psi \in \mathcal{L}(\mathcal{B}_1)$ we get that $\psi \in \mathcal{L}(\mathcal{A}_{i_1} \cup \bigcup_{(i_1,j)\in E, j\neq k} l(i_1, j))$. By the induction hypothesis for $G_1, \mathcal{B}_1$, at some point there will be a set of consequences of $\mathcal{A}_{i_1}$ that entail ψ (after some formulae were sent to it from the other partitions in G_1, B_1). Since G is properly labeled for $\mathcal{A}, G$,

$\psi \in \mathcal{L}(l(k, i_1))$. Since $\mathfrak{R}_i$ is complete for $\mathcal{L}(l(k, i_1))$-generation, at some point the set C of consequences of $\mathcal{A}_{i_1}$ that are in $\mathcal{L}(l(k, i))$ will entail ψ.

At this point, our algorithm will have sent the sentences in C to $\mathcal{A}_k$ because $C \subset \mathcal{L}(l(k, i_1))$. Since $B_2 \vdash \psi \Rightarrow Q$, then $B_2 \vdash C \Rightarrow Q$. By the induction hypothesis applied to G_2, B_2 ($C \Rightarrow Q \in \mathcal{L}(\mathcal{A}_k \cup_{(k,i) \in E} l(k, i))$) at some point $C \Rightarrow Q$ will be entailed from consequences of $\mathcal{A}_k$ (after some message passing). Thus, at some point we will find consequences of $\mathcal{A}_k$ that entail Q. This completes the induction step. ■

A.2 Theorem 3.2: RESOLUTION-M-P (RES-MP) is Sound and Complete

Theorem A.2 ([Lee, 1967]) *For every non-tautologous clause D following from a given clause set $\mathcal{A}$, a clause C is derivable by the resolution rule such that D is obtained from C by instantiation and addition of further literals (i.e., C $\subset$-subsumes D).*

Proof of Theorem 3.2. 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 $\mathcal{A}_j$). ■

A.3 Lemma 3.6: Resolving on P is $\mathcal{L}_P$-Generation Complete

This proof uses the notion of *semantic trees.* For an exposition the reader is referred to [Chang and Lee, 1973]. We bring only the basic definitions needed for the proof here.

The *atom set* of a set of formulae, $\mathcal{A}$ is the set of all atoms in $\mathcal{A}$.

Definition A.3 *Given a set S of clauses, let A be the atom set of S. A* semantic tree *for S is a (downward) tree T, where each link is attached with a finite set of atoms or negations of atoms from A in such a way that*

(1) For each node N, there are only finitely many immediate links $L_1, ..., L_n$ from N. Let Q_i be the conjunction of all the literals in the set attached to $L_i, i = 1, ..., n$. Then, $Q_1 \vee ... \vee Q_n$ is a valid propositional formula.
(2) For each node N, let $I(N)$ be the union of all the sets attached to the links of the branch of T down to and including N. Then, $I(N)$ does not contain any complementary pair.

Definition A.4 *Let $A = \{A_1, A_2, ...\}$ be the atom set of a set S of clauses. A semantic tree for S is said to be* complete *if and only if for every leaf node N of the semantic tree, $I(N)$ contains either A_i or $\neg A_i$, for $i = 1, 2, ...$*

Roughly speaking, a complete semantic tree corresponds to all possible *Herbrand interpretations.*

Definition A.5 *A node N is a* failure node *if $I(N)$ falsifies some ground instance of a clause in S, but $I(N')$ does not falsify any ground instance of a clause in S for every ancestor node N' of N. A semantic tree is said to be* closed *iff every branch of T terminates at a failure node. A node N of a closed semantic tree is called an* inference node *if all the immediate descendant nodes of N are failure nodes.*

The following is a version of Herbrand's Theorem [Herbrand, 1930] stated using semantic trees.

Theorem A.6 (Herbrand's Theorem (ver.1)) *A set S of clauses is unsatisfiable if and only if corresponding to every complete semantic tree of S, there is a finite closed semantic tree.*

We will also make use of the classic mention of Herbrand's Theorem.

Theorem A.7 (Herbrand's Theorem (ver.2)) *A set S of clauses is unsatisfiable if and only if there is a finite unsatisfiable set S' of ground instances of clauses of S.*

Finally, we also make use of the following lifting lemma (see [Chang and Lee, 1973] p.84).

Lemma A.8 (Lifting Lemma) *If C_1', C_2' are instances of C_1, C_2, respectively, and if C' is a resolvent of C_1', C_2', then there is a resolvent C of C_1, C_2 such that C' is an instance of C.*

Proof of Lemma 3.6. Take a formula $\varphi \in \mathcal{L}_P$ such that $\mathcal{A} \models \varphi$. We show that for $\mathcal{A}' = C_{\mathfrak{R}}(\mathcal{A})$, $\mathcal{A}' \models \varphi$.

$\mathcal{A} \cup \{\neg\varphi\}$ is unsatisfiable, by our assumption. Let $\mathcal{A}_0$ be a finite unsatisfiable set of ground instances of $\mathcal{A} \cup \{\neg\varphi\}$, as guaranteed by Theorem A.7. Let T be a complete semantic tree of $\mathcal{A}_0$ such that the instances of P are the lowest steps in the tree. Since $\mathcal{A}_0$ is unsatisfiable, then by Theorem A.6 there is a finite closed semantic tree, T, corresponding to it. This tree is a semantic closed tree for $\mathcal{A} \cup \{\neg\varphi\}$ as well.

We show that this tree corresponds to a resolution deduction of $\{\}$ from $\mathcal{A} \cup \{\neg\varphi\}$ and modify the tree to achieve a resolution of $\{\}$ from $\mathcal{A}' \cup \{\neg\varphi\}$. First, notice that T has at least one inference node (a node under which both nodes are failure nodes), or otherwise we get an infinite branch in it.

Every inference node in T corresponds to a resolution step of two clauses that are falsified by this node's children. Let N be an inference node in T. Let N_1, N_2 be the failure nodes immediately below N. Since N_1, N_2 are failure nodes, there must be two ground instances C_1', C_2' of clauses C_1, C_2, respectively, such that C_1', C_2' are false in $I(N_1), I(N_2)$, respectively, but both C_1', C_2' are not falsified by $I(N)$. Let m be the ground atom that appears positively in $I(N_1)$ and negatively in $I(N_2)$. Then, without loss of generality, C_1' must contain $\neg m$, and C_2' must contain m. Resolving C_1', C_2' upon m we obtain the resolvent

$$C' = (C_1' \setminus \{\neg m\}) \cup (C_2' \setminus \{m\}).$$

C' must be false in $I(N)$ since both $(C_1' \setminus \{\neg m\}), (C_2' \setminus \{m\})$ are false in $I(N)$. By the lifting lemma there is a resolvent, C, of C_1, C_2 such that C' is a ground instance of C. Let T' be the closed semantic tree for $\mathcal{A} \cup \{\neg\varphi\} \cup \{C\}$ obtained from T by deleting any node or link that is below the first node where the resolvent C' is falsified. Clearly, $T' \subsetneq T$. If we continue this process iteratively, we eventually collapse the tree to a singleton, which is the empty set. This process then corresponds to a resolution proof of $\{\}$.

Coming back to our original semantic closed tree, T, we continue the process above iteratively until we remove from T all the inference nodes that resolve on P (we make sure to remove only those that resolve on P). This process results in a new closed semantic tree, T^*, for the theory including some of the resolvents of $\mathcal{A} \cup \{\neg\varphi\}$ on P together with $\mathcal{A}$. Since φ does not include P, φ did not participate in any of the resolutions. Also, T^* is closed before we get to nodes that are falsified by P. Thus, deleting the set of clauses that include a literal with P, we still have an inconsistent set of clauses. The new set of clauses is included in $\mathcal{A}' \cup \{\neg\varphi\}$. Thus, T^* corresponds to a proof of $\{\}$ from $\mathcal{A}' \cup \{\neg\varphi\}$, which concludes our proof. ■

A.4 Theorem 3.7: MP Can Simulate Variable Ordering (Most Times)

Proof of Theorem 3.7. 1. Let $\{p_i\}_{i \le n}$ be the set of propositional symbols of $\mathcal{A}$, enumerated by the order $\le_A$. For all $i \le n$ let $\mathcal{A}_i = \{\varphi \in \mathcal{A} \mid \mathit{highest\_literal}(\varphi) = p_i\}$. Let the graph G have vertices $V = \{1, ..., n\}$, edges $E = \{(i, i-1) \mid i \le n\}$ and labels $l(i, i-1) = \{p_1, ..., p_{i-1}\}$.

For partition $\mathcal{A}_i$ we take the reasoner $\mathfrak{R}_{p_i}$ that resolves sentences only on p_i. By Lemma 3.6 we know that $\mathfrak{R}_{p_i}$ is $\mathcal{L}_{p_i}$-generation complete. Also, for each $i \le n$, $\mathcal{L}_{p_i} = \mathcal{L}(l(i, i-1))$. This can be seen by induction on i. When $i = n$, $\mathcal{L}(\mathcal{A}_i)$ includes propositions in $\{p_1, ..., p_i\}$, and $\mathcal{A}_i$ does not change as it never receives any messages. For $i < n$, $\mathcal{A}_i$ initially includes propositions in $\{p_1, ..., p_i\}$, and the messages $\mathcal{A}_i$ receives from $\mathcal{A}_{i+1}$ are also in this language, by induction. This proves that $\mathcal{L}_{p_i} = \mathcal{L}(l(i, i-1))$ and the set of reasoners $\mathfrak{R}_{p_i}$ is generation-set complete for $\mathcal{A}, G$.

To see that MP does not perform more resolutions than directional resolution of $\mathcal{A}$ with order $\le_A$, we only need to notice that every partition corresponds to a bucket in the directional resolution algorithm. Thus, each partition performs exactly those resolutions done by the corresponding bucket in the directional resolution algorithm.

2. Let $\{P_i\}_{i \le n}$ be the set of predicate symbols of $\mathcal{A}$, enumerated by the order $\le_A$. For all $i \le n$ let $\mathcal{A}_i = \{\varphi \in \mathcal{A} \mid \mathit{highest\_predicate}(\varphi) = P_i\}$. Let the graph G have vertices $V = \{1, ..., n\}$, edges $E = \{(i, i-1) \mid i \le n\}$ and labels $l(i, i-1) = \{P_1, ..., P_{i-1}, f_1, ..., f_s, c_1, ..., c_t\}$. ($f_1, ..., f_s$ are the function symbols in $\mathcal{L}(\mathcal{A})$ and $c_1, ..., c_t$ are the constant symbols in $\mathcal{L}(\mathcal{A})$).

For partition $\mathcal{A}_i$ we take the reasoner $\mathfrak{R}_{P_i}$ that resolves sentences only on P_i. By Lemma 3.6 we know that $\mathfrak{R}_{P_i}$ is $\mathcal{L}_{P_i}$-generation complete. Also, for each $i \le n$, $\mathcal{L}_{P_i} = \mathcal{L}(l(i, i-1))$. This can be seen by induction on i. When $i = n$, $\mathcal{L}(\mathcal{A}_i)$ includes predicates in $\{P_1, ..., P_i\}$, and $\mathcal{A}_i$ does not change as it never receives any messages. For $i < n$, $\mathcal{A}_i$ initially includes propositions in $\{P_1, ..., P_i\}$, and the messages $\mathcal{A}_i$ receives from $\mathcal{A}_{i+1}$ are also in this language, by induction. This proves that $\mathcal{L}_{P_i} = \mathcal{L}(l(i, i-1))$ and the set of reasoners $\mathfrak{R}_{P_i}$ is generation-set complete for $\mathcal{A}, G$.

To see that MP does not perform more resolutions than A-ordered resolution of $\mathcal{A}$ with order $\leq_A$, we only need to notice that every partition resolves only on the top predicate in each clause. This is the case because in every clause in partition $\mathcal{A}_i$ the highest predicate possible is P_i, and the reasoner resolves only on instances of P_i. Thus, each partition performs only resolutions that are allowed by A-ordered resolution.

3. Let $\{\{P^i_j\}_{j \leq l_i}\}_{i \leq n}$ be the set of predicate symbols of $\mathcal{A}$, aggregated to sets of predicates according to the index I (i.e., $I(P^i_j) = i$). For all $i \leq n$ let $\mathcal{A}_i = \{\varphi \in \mathcal{A} \mid I(\mathit{highest\_predicate}(\varphi)) = i\}$. Let the graph G have vertices $V = \{1, ..., n\}$, edges $E = \{(i, i-1) \mid i \leq n\}$ and labels $l(i, i-1) = \{P^1_1, ..., P^{i-1}_{l_{i-1}}, f_1, ..., f_s, c_1, ..., c_t\}$. ($f_1, ..., f_s$ are the function symbols in $\mathcal{L}(\mathcal{A})$ and $c_1, ..., c_t$ are the constant symbols in $\mathcal{L}(\mathcal{A})$).

For partition $\mathcal{A}_i$ we take the reasoner $\mathfrak{R}_{P_i}$ that resolves sentences only on $P^i_1, ..., P^i_{l_i}$. Let $\mathbb{P}^i$ stand for $P^i_1, ..., P^i_{l_i}$. By Lemma 3.6 (applied recursively to the set of predicates $\mathbb{P}^i$), we know that $\mathfrak{R}_{\mathbb{P}^i}$ is $\mathcal{L}_{\mathbb{P}^i}$-generation complete. Also, for each $i \leq n$, $\mathcal{L}_{\mathbb{P}^i} = \mathcal{L}(l(i, i-1))$. This can be seen by induction on i. When $i = n$, $\mathcal{L}(\mathcal{A}_i)$ includes predicates in $\{\mathbb{P}^1, ..., \mathbb{P}^i\}$, and $\mathcal{A}_i$ does not change as it never receives any messages. For $i < n$, $\mathcal{A}_i$ initially includes propositions in $\{\mathbb{P}^1, ..., \mathbb{P}^i\}$, and the messages $\mathcal{A}_i$ receives from $\mathcal{A}_{i+1}$ are also in this language, by induction. This proves that $\mathcal{L}_{\mathbb{P}^i} = \mathcal{L}(l(i, i-1))$ and the set of reasoners $\mathfrak{R}_{\mathbb{P}^i}$ is generation-set complete for $\mathcal{A}, G$.

To see that MP does not perform more resolutions than lock resolution of $\mathcal{A}$ with index I, we only need to notice that every partition resolves only on predicates with highest index in each clause. This is the case because in every clause in partition $\mathcal{A}_i$ the highest predicates possible are in $\mathbb{P}^i$, and the reasoner resolves only on instances of $\mathbb{P}^i$. Thus, each partition performs only resolutions that are allowed by lock resolution. ■

A.5 Theorem 3.8: Variable Ordering Can Simulate MP (Propositional, Sometimes)

Proof of Theorem 3.8. 1. Let $\{\mathcal{A}_i\}_{i \leq n}$ be a partitioning of the theory $\mathcal{A}$, let $G = (V, E, l)$ be a graph and let $k \leq n$. Let $\prec$ be the partial order derived in step 1 of MP. Let $\tilde{L}(i) = \bigcup_{(i,j) \in E, i \prec j} l(i, j)$ (this is the vocabulary mentioned in the partition itself together with the vocabulary mentioned on the links to that partition).

For every $p \in L(\mathcal{A})$, let $P(p)$ be the $\prec$-smallest partition, i, such that $p \in \tilde{L}(i)$. For every $p \in L(\mathcal{A})$ there is exactly one such partition because otherwise there are at least two that are not connected via another node, contradicting the fact that G is properly labeled (proper labeling guarantees that if a symbol appears in two partitions then it appears on the edges on the path between those partitions).

Define $<_0$, an order between symbols, by $p <_0 q$ iff $P(p) \prec P(q)$. Define $<_A$ to be a total order that agrees with $<_0$. $<_A$ is clearly a partial order, as it obeys reflexivity, transitivity and is strict (thus asymmetric). We show that $<_A$ is an order as required. Let $\{p_i\}_{i \leq m}$ be an enumeration of the propositional symbols of $\mathcal{A}$ according to $<_A$. Notice that if directional resolution with $<_A$ resolved two clauses upon p_i, then p_i is the highest symbol in both clauses.

Let C_1, C_2 be two clauses resolved on p in the directional resolution using the order $<_A$. Let i_1, i_2 be such that C_1 in partition i_1 and C_2 in partition i_2.

Assume that MP cannot resolve the two clauses. That means that they are in different partitions and that at least one of the clauses, say C_2, should be sent to another partition, $\mathcal{A}_j$, with $j \prec i_2$, in order for the two clauses to be resolved (j is the $\prec$-largest partition such that $j \prec i_1$ and $j \prec i_2$).

Since C_2 was not sent to partition j, its vocabulary is not in the label of one of the edges on the path between i_2 and p in G (by the definition of MP, we would have sent that clause if it was expressed by the labels on each of the edges on that path).

Let q be a symbol that appears in C_2 but is not on one of the edges in the path between partitions i_2 and j. p is the $<_A$-largest symbol for both C_1 and C_2, meaning that p appears on the path between i_1, i_2 (thus it also appears on the path between i_2, j, which is a subset of the path between i_1, i_2) and that $q <_A p$ (because q does not appear on that path (thus, it is not p) and p is the $<_A$-largest in C_2).

Since q is not in $\tilde{L}(j)$ (otherwise it would be on the path between i_2, j) and $P(p) \preceq j$ (otherwise the path between j, i_2 would not include p) $P(p) \preceq j \prec P(q)$ implying that $p <_A q$. This contradicts the previous observation that $q <_A p$. Thus, q appears on the path between i_2, j and C_2 would have to be sent to partition j. The same holds for C_1. Thus, MP will resolve the two clauses.

2. The proof for A-ordering is identical to the one above, except that we treat the predicate symbols instead of the propositional symbols.

3. Let $\{\mathcal{A}_i\}_{i \leq n}$, $G = (V, E, l)$, k, $\prec$ and $\tilde{L}(i)$ be as for the proof of part 1.

For every predicate $p \in L(\mathcal{A})$, let $P(p)$ be the $\prec$-smallest partition, i, such that $p \in \tilde{L}(i)$. For every $p \in L(\mathcal{A})$ there is exactly one such partition because otherwise there are at least two that are not connected via another node, contradicting the fact that G is properly labeled (proper labeling guarantees that if a symbol appears in two partitions then it appears on the edges on the path between those partitions). For every atom a, $P(a) = P(p)$, if p is the predicate symbol of a.

Define $>_0$, a relation between atoms, by $a_1 >_0 a_2$ iff $P(a_1) \prec P(a_2)$. Define $>_A$ to be a total order that agrees with $>_0$. $>_A$ is clearly a partial order, as it obeys reflexivity, transitivity and is strict (thus asymmetric). Let $\{p_i\}_{i \leq m}$ be an enumeration of the predicate symbols of $\mathcal{A}$ according to $>_A$. Define I to be an index of the literals in $\mathcal{A}$ that agrees with this enumeration and has the same index for an atom and its negation.

We show that I is an index as required. Notice that if lock resolution with I resolved two clauses upon p_i, then p_i is the lowest literal in both clauses.

Let C_1, C_2 be two clauses that lock resolution can resolve together on predicate p. Let i_1, i_2 be such that C_1 in partition i_1 and C_2 in partition i_2.

Assume that MP cannot resolve the two clauses. That means that they are in different partitions and that at least one of the clauses, say C_2, should be sent to another partition, $\mathcal{A}_j$, with $j \prec i_2$, in order for the two clauses to be resolved (j is the $\prec$-largest partition such that $j \prec i_1$ and $j \prec i_2$).

Since C_2 was not sent to partition j, its vocabulary is not in the label of one of the edges on the path between i_2 and p in G (by the definition of MP, we would have sent that clause if it was expressed by the labels on each of the edges on that path).

Let q be a predicate symbol that appears in C_2 but is not on one of the edges in the path between partitions i_2 and j. p is the $>_A$-smallest symbol for both C_1 and C_2, meaning that p appears on the path between i_1, i_2 (thus it also appears on the path between i_2, j, which is a subset of the path between i_1, i_2) and that $q >_A p$ (because q does not appear on that path (thus, it is not p) and p is the $>_A$-largest in C_2).

Since q is not in $\tilde{L}(j)$ (otherwise it would be on the path between i_2, j) and $P(p) \preceq j$ (otherwise the path between j, i_2 would not include p) $P(p) \preceq j \prec P(q)$ implying that $p >_A q$. This contradicts the previous observation that $q >_A p$. Thus, q appears on the path between i_2, j and C_2 would have to be sent to partition j. The same holds for C_1. Thus, MP will resolve the two clauses. ■

A.6 Theorem 5.1: COMPILE with MP is Sound and Complete

Soundness is immediate. The only rules used to derive consequences are those in the sound consequence-finding procedure. The completeness follows from the completeness of MP in Theorem 2.4. $\mathcal{A} \models Q$ iff MP$(\{\mathcal{A}_i\}_{i \leq n}, G, Q)$ outputs yes. Thus, by completeness if MP$(\{\mathcal{A}_i\}_{i \leq n}, G, Q)$ outputs yes, then MP$(\{\mathcal{A}_i^{'}\}_{i \leq n}, G, Q)$ outputs yes. Assume COMPILE+MP is incomplete, then there are 2 cases.

Case 1: Let ψ be an arbitrary formula to be added by MP Step (2)(b) to $\mathcal{A}_i$, $\mathcal{A}_i^{'}$, respectively. I.e., $\psi \in \mathcal{L}_j = \mathcal{L}(l(i,j))$ for j such that $(i,j) \in E$ and $i \prec j$. Assuming incompleteness, then for some $\mathcal{A}_i, \exists \varphi \in \mathcal{L}(l(i,p)), (i,p) \in E.\varphi$ is a consequence of $\mathcal{A}_i \wedge \psi$, but φ is not an $\mathcal{L}_{comm_i}$-consequence of $\mathcal{A}_i \wedge \psi$. I.e., it is not a consequence of the compiled partition $\mathcal{A}_i^{'}$. Since $\psi, \varphi \in \mathcal{L}_{comm_i}$ and COMPILE uses a consequence finder that is complete for $\mathcal{L}$-consequence finding, computing all the consequences of $\mathcal{L}_{comm_i}$, we have a contradiction.

Case 2: Let ψ be an arbitrary formula to be added to $\mathcal{A}_k$, $\mathcal{A}_k^{'}$, respectively. I.e., $\psi \in \mathcal{L}_j = \mathcal{L}(l(k,j))$ for j such that $(k,j) \in E$ and $k \prec j$. Assuming incompleteness, then for some $\mathcal{A}_k, \exists \varphi \in \mathcal{L}(l(k,j)).\varphi$ is a consequence of $\mathcal{A}_k \wedge \psi$, but φ is not an $\mathcal{L}_{comm_k}$-consequence of $\mathcal{A}_k \wedge \psi$. But $\mathcal{A}_k = \mathcal{A}_k^{'}$, so we have a contradiction. ■

A.7 Theorem 5.3: BATCH-MP is Sound and Complete

Following from Theorem 5.1, the proof holds for both a compiled and uncompiled theory.

Soundness is immediate. The only rules used to derive consequences are those in the sound consequence-finding procedure. The completeness follows from the induction proof of the completeness of MP in Theorem 2.4. The induction is over the number of partitions in $\mathcal{A}$. It suffices to show that for every $\mathcal{A}_j, j \prec i$, that BATCH-MP adds the complete $\mathcal{L}_i$-consequences of $\mathcal{A}_j$ to $\mathcal{A}_i$. In the base case, when there is one partition, the procedure is trivially complete. In all other cases, since each of the reasoners, $\mathfrak{R}_i$ is complete for $\mathcal{L}_i$-consequence finding, and since consequence finding in $\mathcal{L}_i$ is exhaustive and all resulting consequences are passed to $\mathcal{A}_j$, $j \prec i$, then the BATCH-MP procedure is complete. ■

A.8 Theorem 5.4: CONCURRENT-MP is Sound and Complete

Following from Theorem 5.1, the proof holds for both a compiled and uncompiled theory.

Soundness is immediate. The only rules used to derive consequences are those in the sound consequence-finding procedures. The completeness follows from the induction proof of the completeness of MP in Theorem 2.4. The induction is over the number of partitions in $\mathcal{A}$. It suffices to show that for every $\mathcal{A}_j, j \prec i$, that CONCURRENT-MP terminates when it proves Q in Step (6)(b), or it adds the complete $\mathcal{L}_i$-consequences of $\mathcal{A}_i$ to $\mathcal{A}_j$ and fails to prove Q. In the base case, when there is one partition, the procedure is trivially complete. In all other cases, since each of the reasoners, $\mathfrak{R}_i$ is complete for $\mathcal{L}_i$-consequence finding, and by definition the $\mathcal{L}_i$-prime implicates entail all the $\mathcal{L}_i$-consequences, then all $\mathcal{L}_i$-consequences of $\mathcal{A}_i$ will be added to $\mathcal{A}_j$ in Steps (4) and (5). Further since every $\mathfrak{R}_i'$ is complete for incremental $\mathcal{L}_i$-consequence finding, then Step (6)(a) will add all the $\mathcal{L}_i$ consequences of $\mathcal{A}_i \wedge \varphi$ to $\mathcal{A}_j$, where φ is the complete $\mathcal{L}_h$-consequences of each $\mathcal{A}_h$, $j \prec h$. ■

References

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

[Amir, 2001] Amir, E. (2001). Efficient approximation for triangulation of minimum treewidth. In *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI '01)*, pages 7–15. Morgan Kaufmann.

[Amir and McIlraith, 2000] Amir, E. and McIlraith, S. (2000). Paritition-based logical reasoning. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conference (KR '2000)*, pages 389–400. Morgan Kaufmann.

[Amir and McIlraith, 2001] Amir, E. and McIlraith, S. (2001). Partition-based logical reasoning for first-order and propositional theories. *Submitted for publication.*

[Anderson and Bledsoe, 1970] Anderson, R. and Bledsoe, W. W. (1970). A linear format for resolution with merging and a new technique for establishing completeness. *Journal of the ACM*, 17:525–534.

[Ayeb et al., 1993] Ayeb, B. E., Marquis, P., and Rusinowitch, M. (1993). Preferring diagnoses by abduction. *IEEE Transactions on SMC*, 23(3):792–808.

[Baader and Schulz, 1992] Baader, F. and Schulz, K. U. (1992). Unification in the union of disjoint equational theories: Combining decision procedures. In

Proceedings of the 11th International conference on automated deduction, volume 607 of *LNAI*, pages 50–65. Springer-Verlag.

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

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

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

[Bonacina and Hsiang, 1996] Bonacina, M. P. and Hsiang, J. (1996). On the representation of dynamic search spaces in theorem proving. In Yang, C.-S., editor, *Proceedings of the International Computer Symposium*, pages 85–94.

[Boppana and Sipser, 1990] Boppana, R. B. and Sipser, M. (1990). The complexity of finite functions. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, volume 1: Algorithms and Complexity. Elsevier/MIT Press.

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

[Chadha and Plaisted, 1993] Chadha, R. and Plaisted, D. A. (1993). Finding logical consequences using unskolemization. In *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS'93)*, volume 689 of *Lecture Notes in AI*, pages 255–264. Springer-Verlag.

[Chang and Lee, 1973] Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving.* Academic Press.

[Cohen et al., 1998] Cohen, P., Schrag, R., Jones, E., Pease, A., Lin, A., Starr, B., Gunning, D., and Burke, M. (1998). The darpa high-performance knowledge bases project. *AI Magazine*, 19(4):25–49.

[Conry et al., 1990] Conry, S. E., McIntosh, D. J., and Meyer, R. A. (1990). DARES: a distributed automated reasoning system. In *Proc. National Conference on Artificial Intelligence (AAAI '90)*, pages 78–85. AAAI Press/MIT Press.

[Cowen and Wyatt, 1993] Cowen, R. and Wyatt, K. (1993). BREAKUP: A preprocessing algorithm for satisfiability testing of CNF formulas. *Notre Dame Journal of Formal Logic*, 34(4):602–606.

[Cox and Pietrzykowski, 1984] Cox, P. and Pietrzykowski, T. (1984). A complete nonredundant algorithm for reversed skolemization. *theoretical computer science*, 28:239–261.

[Craig, 1957] Craig, W. (1957). Linear reasoning. a new form of the herbrand-gentzen theorem. *Journal of Symbolic Logic*, 22:250–268.

[Darwiche, 1996] Darwiche, A. (1996). Utilizing knowledge-based semantics in graph-based algorithms. In *Proc. National Conference on Artificial Intelligence (AAAI '96)*, pages 607–613. Morgan Kaufmann.

[Davis and Putnam, 1960] Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215.

[Dechter and Pearl, 1988] Dechter, R. and Pearl, J. (1988). Tree Clustering Schemes for Constraint Processing. In *Proc. National Conference on Artificial Intelligence (AAAI '88)*.

[Dechter and Rish, 1994] Dechter, R. and Rish, I. (1994). Directional resolution: The davis-putnam procedure, revisited. In *Principles of Knowledge Representation and Reasoning: Proc. Fourth International Conference (KR '94)*, pages 134–145. Morgan Kaufmann.

[del Val, 1999] del Val, A. (1999). A new method for consequence finding and compilation in restricted language. In *Proc. National Conference on Artificial Intelligence (AAAI '99)*, pages 259–264. AAAI Press/MIT Press.

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

[Denzinger and Fuchs, 1999] Denzinger, J. and Fuchs, D. (1999). Cooperation of heterogeneous provers. In Thomas, D., editor, *Proc. Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99)*, pages 10–15. Morgan Kaufmann.

[Eisinger and Ohlbach, 1993] Eisinger, N. and Ohlbach, H. J. (1993). Deduction systems based on resolution. In Gabbay, D., Hogger, C., and J.A.Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 1: Logical Foundations.* Oxford University Press.

[Ertel, 1992] Ertel, W. (1992). OR-parallel theorem proving with random competition. In *Proc. LPAR'92*, volume 624 of *LNAI*, pages 226–237.

[Fikes and Farquhar, 1999] Fikes, R. and Farquhar, A. (1999). Large-scale repositories of highly expressive reusable knowledge. *IEEE Intelligent Systems*, 14(2).

[Genesereth and Nilsson, 1987] Genesereth, M. R. and Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers Inc.

[Giunchiglia and Traverso, 1995] Giunchiglia, E. and Traverso, P. (1995). A multi-context architecture for formalizing complex reasoning. *International Journal of Intelligent Systems*, 10:501–539. Also, IRST Tech. Report #9307-26.

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

[Herbrand, 1930] Herbrand, J. (1930). Recherches sur la théorie de la démonstration. *Travaux de la Société des Sciences et de Lettres de Varsovie, Classe III Sci. Math. Phys.*, 33.

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

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

[Kautz and Selman, 1996] Kautz, H. and Selman, B. (1996). Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224.

[Kowalski and Hayes, 1969] Kowalski, R. A. and Hayes, P. J. (1969). Semantic trees in automatic theorem-proving. *Machine Intelligence*, 4:87–101.

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

[Lenat, 1995] Lenat, D. B. (1995). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38.

[Lin, 2000] Lin, F. (2000). On strongest necessary and weakest sufficient conditions. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conference (KR '2000)*, pages 167–175.

[Lloyd and Topor, 1985] Lloyd, J. W. and Topor, R. W. (1985). A basis for deductive database systems. *Journal of Logic Programming*, 2:93–109.

[Loveland, 1969] Loveland, D. W. (1969). A simplified format for the model elimination procedure. *Journal of the ACM*, 16(3):349–363. Reprinted in: [Siekmann and Wrightson, 1983].

[Loveland, 1970] Loveland, D. W. (1970). A linear format for resolution. In *Proc. IRIA symposium on automatic demonstration*, volume 125 of *Lecture notes in mathematics*, pages 147–162. Springer-Verlag.

[Loveland, 1978] Loveland, D. W. (1978). *Automated theorem proving: a logical basis.* Fundamental studies in computer science. North-Holland.

[Luckham, 1970] Luckham, D. (1970). Refinement theorem in resolution theory. In *Proc. IRIA symposium on automatic demonstration*, volume 125 of *Lecture notes in mathematics*, pages 163–190. Springer-Verlag.

[Luckham and Nilsson, 1971] Luckham, D. and Nilsson, N. (1971). Extracting information from resolution proof trees. *Artificial Intelligence*, 2:27–54.

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

[Marquis, 2000] Marquis, P. (2000). Consequence finding algorithms. In Moral, S. and Kohlas, J., editors, *Algorithms for Defeasible and Uncertain Reasoning*, volume 5 of *Handbook on Deafeasible Reasoning and Uncertainty Management Systems*, pages 41–145. Kluwer Academic Publishers.

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

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

[Reiter, 1971] Reiter, R. (1971). Two results on ordering for resolution with merging and linear format. In *Journal of the ACM*, volume 18, pages 630–646.

[Reynolds, 1965] Reynolds, J. C. (1965). Unpublished seminar note, stanford university, stanford, ca. Referenced in [Slagle, 1967].

[Ringeissen, 1996] Ringeissen, C. (1996). Cooperation of decision procedures for the satisfiability problem. In Baader, F. and Schulz, K., editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 121–140. Kluwer Academic Publishers.

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

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

[Schrag and Crawford, 1996] Schrag, R. and Crawford, J. M. (1996). Implicates and prime implicates in random 3sat. *Artificial Intelligence*, 81:199–222.

[Shostak, 1984] Shostak, R. E. (1984). Deciding combinations of theories. *Journal of the ACM*, 31:1–12.

[Siekmann and Wrightson, 1983] Siekmann, J. and Wrightson, G., editors (1983). *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag.

[Slagle, 1967] Slagle, J. R. (1967). Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14(4):687–697.

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

[Slagle et al., 1969] Slagle, J. R., Chang, C.-L., and Lee, R. C. T. (1969). Completeness theorems for semantic resolution in consequence-finding. In *Proc. First International Joint Conference on Artificial Intelligence (IJCAI '69)*, pages 281–285.

[Stickel, 1985] Stickel, M. E. (1985). Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355.

[Stickel, 1988] Stickel, M. E. (1988). A Prolog technology theorem prover. In Lusk, E. and Overbeek, R., editors, *Proc. 9^{th} International Conference on Automated Deduction*, pages 752–753. Springer LNCS, New York.

[Sutcliffe, 1992] Sutcliffe, G. C. J. (1992). A heterogeneous parallel deduction system. FGCS'92 Workshop on Automated Deduction: Logic Programming and Parallel Computing Approaches.

[Suttner, 1997] Suttner, C. B. (1997). SPTHEO. *Journal of Automated Reasoning*, 18:253–258.

[Tinelli and Harandi, 1996] Tinelli, C. and Harandi, M. T. (1996). A new correctness proof of the Nelson–Oppen combination procedure. In Baader, F. and Schulz, K., editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 103–120. Kluwer Academic Publishers.

[Wos et al., 1965] Wos, L., Robinson, G. A., and Carson, D. F. (1965). Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM*, 12(4):536–541.

[Yates et al., 1970] Yates, R. A., Raphael, B., and Hart, T. P. (1970). Resolution graphs. *Artificial Intelligence*, 1:257–289.