Partition-Based Logical Reasoning

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

Abstract

We investigate the problem of reasoning with partitions of related logical axioms. Our motivation is two-fold. First, we are concerned with how to reason effectively with multiple knowledge bases that have overlap in content. Second, and more fundamentally, we are concerned with how to exploit structure inherent in a set of logical axioms to induce a partitioning of the axioms that will lead to an improvement in the efficiency of reasoning. To this end, we provide algorithms for reasoning with partitions of axioms in propositional and first-order logic. Craig's interpolation theorem serves as a key to proving completeness of these algorithms. We analyze the computational benefit of our algorithms and detect those parameters of a partitioning that influence the efficiency of computation. These parameters are the number of symbols shared by a pair of partitions, the size of each partition, and the topology of the partitioning. Finally, we provide a greedy algorithm that automatically decomposes a given theory into partitions, exploiting the parameters that influence the efficiency of computation.

1 Introduction

There is growing interest in building large knowledge bases (KBs) of everyday knowledge about the world, teamed with theorem provers to perform inference. Three such systems are Cycorp's Cyc, and the High Performance Knowledge Base (HPKB) systems developed by Stanford's Knowledge Systems Lab (KSL) [23] and by SRI (e.g., [12]). These KBs comprise tens/hundreds of thousands of logical axioms. One approach to dealing with the size and complexity of these KBs is to structure the content in some way, such as into multiple domain- or task-specific KBs, or into microtheories. In this paper, we investigate how to reason effectively with partitioned sets of logical axioms that have overlap in content, and that may even have different reasoning engines. More generally, we investigate the problem of how to exploit structure inherent in a set of logical axioms to induce a partitioning of the axioms that will improve the efficiency of reasoning.

To this end, we propose *partition-based* logical reasoning algorithms, for reasoning with logical theories[1] that are decomposed into related partitions of axioms. Given a partitioning of a logical theory, we use Craig's interpolation theorem [15] to prove the soundness and completeness of a forward message-passing algorithm and an algorithm for propositional satisfiability. The algorithms are designed so that, without loss of generality, reasoning within a partition can be realized by an arbitrary consequence-finding engine, in parallel with reasoning in other partitions. We investigate the impact of these algorithms on resolution-based inference, and analyze the computational complexity for our partition-based SAT.

A critical aspect of partition-based logical reasoning is the selection of a *good* partitioning of the theory. The computational analysis of our partition-based reasoning algorithms provides a metric for identifying parameters of partitionings that influence the computation of our algorithms: the *bandwidth* of communication between partitions, the size of each partition, and the topology of the partitions graph. These parameters guide us to propose a greedy algorithm for decomposing logical theories into partitions, trying to optimize these parameters.

Surprisingly, there has been little work on the specific problem of exploiting structure in theorem proving and SAT in the manner we propose. This can largely be attributed to the fact that theorem proving has traditionally examined mathematics domains, that do not necessarily have structure that supports decomposition. Nevertheless, there are

[*]Knowledge Systems Laboratory (KSL)

[1]In this paper, every set of axioms is a *theory* (and vice versa).

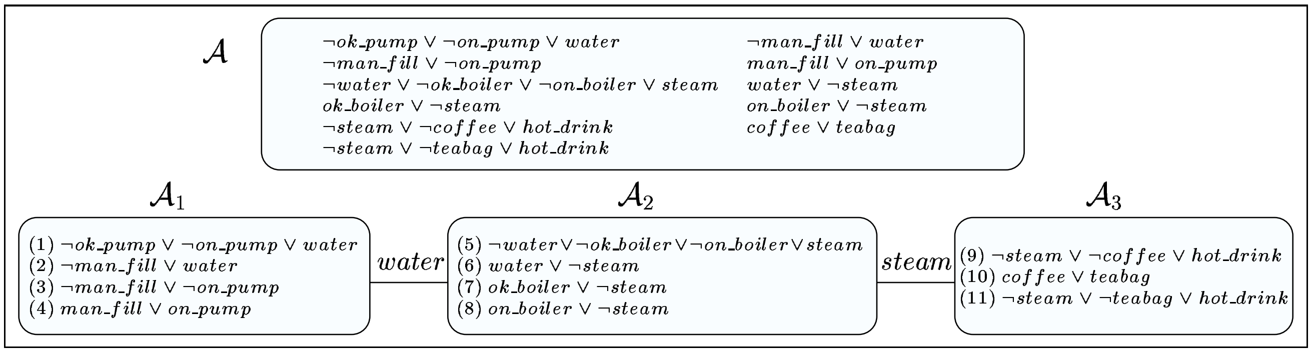
$\mathcal{A}$
$\neg ok\_pump \vee \neg on\_pump \vee water$
$\neg man\_fill \vee \neg on\_pump$
$\neg water \vee \neg ok\_boiler \vee \neg on\_boiler \vee steam$
$ok\_boiler \vee \neg steam$
$\neg steam \vee \neg coffee \vee hot\_drink$
$\neg steam \vee \neg teabag \vee hot\_drink$
$\neg man\_fill \vee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

Figure 1: A partitioning of $\mathcal{A}$ and its intersection graph.

many areas of related work, which we discuss at the end of this paper.

2 Partition-Based Theorem Proving

In this section we address the problem of how to reason with an already partitioned propositional or first-order logic (FOL) theory. In particular, we propose a forward message-passing algorithm, in the spirit of Pearl [40], and examine the effect of this algorithm on resolution-based inference.

$\{\mathcal{A}_i\}_{i\leq n}$ is a *partitioning* of a logical theory $\mathcal{A}$ if $\mathcal{A} = \bigcup_i \mathcal{A}_i$. Each individual $\mathcal{A}_i$ is called a *partition*, and $\mathcal{L}(\mathcal{A}_i)$ is its signature (the non-logical symbols). Each such partitioning defines a labeled graph $G = (V, E, l)$, which we call the *intersection graph*. In the intersection graph, each node 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 symbol in common ($E = \{(i, j) \mid \mathcal{L}(\mathcal{A}_i) \cap \mathcal{L}(\mathcal{A}_j) \neq \emptyset\}$), and the edges are labeled with the set of symbols that the associated partitions share ($l(i, j) = \mathcal{L}(\mathcal{A}_i) \cap \mathcal{L}(\mathcal{A}_j)$). 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, $l(i, j) = \emptyset$.

We illustrate the notion of a partitioning in terms of the simple propositional theory $\mathcal{A}$, depicted at the top of Figure 1. (This is the clausal form of the theory presented with material implication in Figure 2.) This set of axioms captures the functioning of aspects of an expresso machine. The top four axioms denote that if the machine pump is OK and the pump is on then the machine has a water supply. Alternately, the machine can be filled manually, but it is never the case that the machine is manually filling while the pump is on. The second four axioms denote that there is steam if and only if the boiler is OK and is on, and there is a supply of water. Finally, there is always either coffee or tea. Steam and coffee (or tea) result in a hot drink.

$ok\_pump \wedge on\_pump \Rightarrow water$	$man\_fill \Rightarrow water$
$man\_fill \Rightarrow \neg on\_pump$	$\neg man\_fill \Rightarrow on\_pump$
$water \wedge ok\_boiler \wedge on\_boiler \Rightarrow steam$	$\neg water \Rightarrow \neg steam$
$\neg ok\_boiler \Rightarrow \neg steam$	$\neg on\_boiler \Rightarrow \neg steam$
$steam \wedge coffee \Rightarrow hot\_drink$	$coffee \vee teabag$
$steam \wedge teabag \Rightarrow hot\_drink$	

Figure 2: Axiomatization of a simplified espresso machine.

The bottom of Figure 1 depicts a decomposition of $\mathcal{A}$ into three partitions $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and its intersection graph. The labels for the edges $(1, 2), (2, 3)$ are $\{water\}$ and $\{steam\}$, respectively.

2.1 Forward Message Passing

In this section we propose a forward message-passing algorithm for reasoning with partitions of logical axioms. Figure 3 describes our forward message-passing algorithm, FORWARD-M-P (MP) for finding the truth value of query formula Q whose signature is in $\mathcal{L}(\mathcal{A}_k)$, given partitioned theory $\mathcal{A}$ and graph $G = (V, E, l)$, possibly the intersection graph of $\mathcal{A}$, but not always so.

```
PROCEDURE FORWARD-M-P($\{\mathcal{A}_i\}_{i\leq n}, G, Q$)
$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a graph describing the connections between the partitions, $Q$ a query 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 perform consequence finding for each of the partitions $\mathcal{A}_i$, $i \leq n$.
    3. For every $(i, j) \in E$ such that $i \prec j$, if we prove $\mathcal{A}_j \models \varphi$ and $\varphi$'s signature is in $\mathcal{L}(l(i, j))$, then add $\varphi$ to the set of axioms of $\mathcal{A}_i$.
    4. If we proved $Q$ in $\mathcal{A}_k$, return YES.
```

Figure 3: A forward message-passing algorithm.

This algorithm exploits consequence finding (step 2) to perform reasoning in the individual partitions. Consequence finding was defined by Lee [33] to be the problem of finding all the logical consequences of a theory or sentences that subsume them. In MP, we can use any sound and complete consequence-finding algorithm. The *resolution rule* is complete for consequence finding (e.g., [33, 46]) and the same is true for *semantic resolution* [47] (and *set-of-support resolution* [24]), and *linear resolution* variants (e.g., [30]). Such consequence finders are used for prime implicate generation in applications such as diagnosis. Inoue [30] provides an algorithm for selectively generating consequences or *characteristic clauses* in a given sub-vocabulary. We can exploit this algorithm to focus consequence finding on axioms whose signature is in the communication language of the partition.

Figure 4 illustrates an execution of MP using resolution.

Using FORWARD-M-P to prove hot_drink

Part.	Resolve	Generating	
$\mathcal{A}_1$	(2) , (4)	$on\_pump \vee water$	(m1)
$\mathcal{A}_1$	$(m1)$, (1)	$ok\_pump \vee water$	(m2)
$\mathcal{A}_1$	$(m2)$, (12)	$water$	(m3)
		clause $water$ passed from $\mathcal{A}_1$ to $\mathcal{A}_2$	
$\mathcal{A}_2$	(m3) , (5)	$ok\_boiler \wedge on\_boiler \supset steam$	(m4)
$\mathcal{A}_2$	(m4) , (13)	$\neg on\_boiler \vee steam$	(m5)
$\mathcal{A}_2$	(m5) , (14)	$steam$	(m6)
		clause $steam$ passed from $\mathcal{A}_2$ to $\mathcal{A}_3$	
$\mathcal{A}_3$	(9) , (10)	$\neg steam \vee teabag \vee hot\_drink$	(m7)
$\mathcal{A}_3$	(m7) , (11)	$\neg steam \vee hot\_drink$	(m8)
$\mathcal{A}_3$	(m8) , $(m6)$	hot_drink	(m9)

Figure 4: 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 $\mathcal{A}_2$.

Given a partitioning whose intersection graph forms an *undirected tree,* our MP algorithm is a sound and complete proof procedure. The completeness relies on Craig's Interpolation Theorem.

Theorem 2.1 (Craig's Interpolation Theorem [15]) *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 $\gamma \vdash \beta$.*

Craig's interpolation theorem is true even if we take α, β to be infinite sets of sentences [46], and use resolution theorem proving [46, 29], with or without equality [15, 16] (all after proper reformulation of the theorem). Lyndon's version of the interpolation theorem [35] adds sensitivity to the polarity of relation symbols (γ includes P positively (negatively) only if P shows positively (negatively) in both α and β).

Theorem 2.2 (Soundness and Completeness) *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned theory with the intersection graph G being a tree (i.e., no cycles). Let $k \leq n$ and φ a sentence whose signature is in $\mathcal{L}(\mathcal{A}_k)$. Then, $\mathcal{A} \models \varphi$ iff MP outputs YES.*

PROOF See Appendix A.1.

If the intersection graph of $\mathcal{A}$ is not a tree, and MP uses it as input, then MP may fail to be a complete proof procedure. Figure 5 illustrates the problem. The left-hand side of Figure 5 illustrates the intersection graph of partitioning $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$ of a theory $\mathcal{A}$. If we try to prove s (which follows from $\mathcal{A}$) from this partitioning and graph using MP, nothing will be transmitted between the partitions. For example, we cannot send $p \Rightarrow s$ from $\mathcal{A}_2$ to $\mathcal{A}_4$ because the graph only allows transmission of sentences containing s.

Thus, using MP with the left-hand side graph will fail to prove s. In such a case, we must first syntactically transform the intersection graph into a tree with enlarged labels, (i.e. an enlarged communication language) and apply MP to the resultant tree. Algorithm BREAK-CYCLES, shown in Figure 6, performs the appropriate transformation ($|X|$ is the cardinality of a set X).

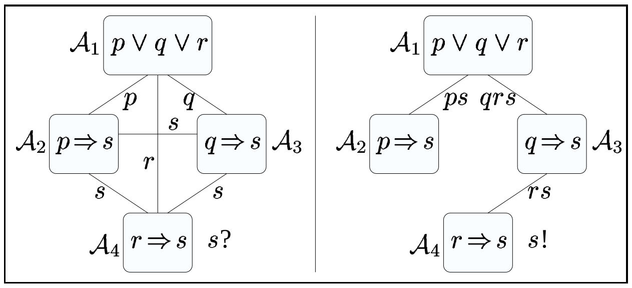
$\mathcal{A}_1$ $p \vee q \vee r$
p
q
s
r
$\mathcal{A}_2$ $p \Rightarrow s$
$q \Rightarrow s$ $\mathcal{A}_3$
s
s
$\mathcal{A}_4$ $r \Rightarrow s$ $s?$
$\mathcal{A}_1$ $p \vee q \vee r$
ps qrs
$\mathcal{A}_2$ $p \Rightarrow s$
$q \Rightarrow s$ $\mathcal{A}_3$
rs
$\mathcal{A}_4$ $r \Rightarrow s$ $s!$

Figure 5: An intersection graph before (left) and after (right) applying BREAK-CYCLES.

```
PROCEDURE BREAK-CYCLES($G = (V, E, l)$)
    1. Find a minimal-length cycle of nodes $i_1, ..., i_c$ in $G$. If
       there are no cycles, return $G$.
    2. Select index $a$ s.t. $a < c$ and $\sum_{a \neq j < c} |l(i_j, i_{j+1}) \cup l(i_a, i_{a+1})|$ is minimal (the label of $(i_a, i_{a+1})$ adds a min-
       imal number of symbols to the rest of the cycle).
    3. For all $j < c$, $j \neq a$, set $l(i_j, i_{j+1}) \leftarrow l(i_j, i_{j+1}) \cup$
       $l(i_a, i_{a+1})$.
    4. Set $E \leftarrow E \setminus \{(i_a, i_{a+1})\}$, $l(i_a, i_{a+1}) \leftarrow \emptyset$ and go to 1.
```

Figure 6: An algorithm to transform an intersection graph G into a tree.

Using BREAK-CYCLES, we can transform the graph depicted on the left-hand side of Figure 5, into the tree on its right. First, we identify the minimal cycle $\langle(1,3),(3,4),(4,1)\rangle$, remove $(4,1)$ from E and add r to the labels of $(1,3),(3,4)$. Then, we find the minimal cycle $\langle(2,3),(3,4),(4,2)\rangle$ and remove $(2,3)$ from E (s already appears in the labels of $(4,2),(3,4)$). Finally, we identify the minimal cycle $\langle(1,3),(3,4),(4,2),(2,1)\rangle$, remove $(4,2)$ and add s to the rest of the cycle. The proof of s by MP now follows by sending $p \Rightarrow s$ from $\mathcal{A}_2$ to $\mathcal{A}_1$, sending $q \vee r \vee s$ from $\mathcal{A}_1$ to $\mathcal{A}_3$, sending $r \vee s$ from $\mathcal{A}_3$ to $\mathcal{A}_4$ and concluding s in $\mathcal{A}_4$.

Notice that when executing BREAK-CYCLES, we may remove an edge that participates in more than one minimal cycle (as is the case of removing the edge $(4,1)$), but its removal influences the labels of only one cycle.

Theorem 2.3 (Soundness and Completeness) *Let $\mathcal{A} = \bigcup_{i\leq n}\mathcal{A}_i$ be a partitioned theory with the intersection graph G. Let $k \leq n$ and φ a sentence whose signature is in $\mathcal{L}(\mathcal{A}_k)$. $\mathcal{A} \models \varphi$ iff applying BREAK-CYCLES and then MP outputs YES.*

PROOF See Appendix A.2.

BREAK-CYCLES is a greedy algorithm that has a worst-case complexity of $O(|E|^2 * m)$ (where m is the number of symbols in $\mathcal{L}(\mathcal{A})$). An algorithm for finding a minimum spanning tree (e.g., [22]) can be used to compute the optimal subgraph for our purpose (minimizing $\sum_{(i,j)\in E}|l(i,j)|$) in time $O(n^2 + m * n)$, if we assume that the labels are mutually disjoint. It is not clear whether the algorithm can be generalized to provide an optimal solution to the case of arbitrary labels.

Note that MP requires that query Q be in the language of a single partition, $\mathcal{L}(\mathcal{A}_k)$. One way to answer a query Q that comprises symbols drawn from multiple partitions is to add a new partition $\mathcal{A}_Q$, with $\mathcal{L}(\mathcal{A}_Q) = \mathcal{L}(Q)$, the signature the query. $\mathcal{A}_Q$ may contain $\neg Q$ or no axioms. Following addition of this new partition, BREAK-CYCLES must be run on the new intersection graph. To prove Q in $\mathcal{A}_Q$, we run MP on the resulting graph.

Our MP algorithm uses the query Q to induce an ordering on the partitions, which in turn may guide selective consequence finding for *reasoning forward*. Many theorem proving strategies exploit the query more aggressively by *reasoning backwards* from the query. Such strategies have proven effective for a variety of reasoning problems, such as planning. Indeed, many theorem provers (e.g., PTTP [49]) are built as backward reasoners and must have a query or *goal* in order to run.

One way to use MP for an analogous backward message-passing scheme is to assert $\neg Q$ in $\mathcal{A}_k$, choose a partition $\mathcal{A}_j$ that is most distant from $\mathcal{A}_k$ in G, and try to prove $\{\}$ in $\mathcal{A}_j$. If we wish to follow the spirit of backward-reasoning more closely, we can connect all the nodes of G, and run BREAK-CYCLES with the stipulation that it must produce a *chain* graph with $\mathcal{A}_k$ at one end. The resultant chain graph may then be used for query-driven backward message-passing, from $\mathcal{A}_k$. With resolution, the goal of a partition at a given time can be taken to be the disjunction of the negation of the messages it has received. Note that for the algorithm to be complete, each partition's reasoner must be complete for consequence finding.

2.2 Resolution-Based Inference

We now analyze the effect of forward message-passing (MP) on the computational efficiency of resolution-based inference, and identify some of the parameters of influence. Current measures for comparing automated deduction strategies are insufficient for our purposes. Proof length (e.g., [28]) is only marginally relevant. More relevant is comparing the sizes of search spaces of different strategies (e.g., [41]). These measures do not precisely address our needs, but we use them here, leaving better comparison for future work.

In a *resolution search space*, each node includes a set of clauses, and properties relevant to the utilized resolution strategy (e.g., clause parenthood information). Each arc is a resolution step allowed by the strategy. In contrast, in an *MP resolution search space* the nodes also include partition membership information. Further, each arc is a resolution step allowed by the utilized resolution strategy that satisfies either of: (1) the two axioms are in the same partition, or (2) one of the axioms is in partition $\mathcal{A}_j$, the second axiom is drawn from its communication language $l(i,j)$, and the query-based ordering allows the second axiom to be sent from $\mathcal{A}_i$ to $\mathcal{A}_j$. Legal sequence of resolutions correspond to paths in these spaces.

Proposition 2.4 *Let $\mathcal{A} = \bigcup_{i\leq n}\mathcal{A}_i$ be a partitioned theory. Any path in the MP resolution search space of $\{\mathcal{A}_i\}_{i\leq n}$ is also a path in the resolution search space of the unpartitioned theory $\mathcal{A}$.*

A similar proposition is true for linear resolution (each partition $\mathcal{A}_i$ uses linear resolution in which messages to $\mathcal{A}_i$ are treated as regular (non-input) sentences in $\mathcal{A}_i$).

From the point of view of proof length, it follows that the longest proof without using MP is as long or longer than the longest MP proof. Unfortunately, the shortest MP proof may be longer than the shortest possible proof without MP. This observation can be quantified most easily in the simple case of only two partitions $\mathcal{A}_1, \mathcal{A}_2$. The set of messages that need to be sent from $\mathcal{A}_1$ to $\mathcal{A}_2$ to prove Q is exactly the interpolant γ promised by Theorem 2.1 for $\alpha = \mathcal{A}_1$, $\beta = \mathcal{A}_2 \Rightarrow Q$. The MP proof has to prove $\alpha \vdash \gamma$ and $\gamma \vdash \beta$. Carbone [9] showed that, if γ is a minimal interpolant, then for many important cases the proof length of $\alpha \vdash \gamma$ together with the proof length of $\gamma \vdash \beta$ is in $O(k^2)$ (for sequent calculus with cuts), where k is the length of the minimal proof of $\alpha \vdash \beta$.

In general, the size of γ itself may be large. 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 $NP \cap coNP \subseteq P/poly$ [7]. Nevertheless, there is a good upper bound on the length of the interpolation formula as a function of the length of the minimal proof [32] : If α, β share l symbols, and the resolution proof of $\alpha \vdash \beta$ is of length k, then there is an interpolant γ of length $min(kl^{O(1)}, 2^l)$.

Thus, we can guarantee a small interpolant, if we make sure the communication language is minimal. Unfortunately, we do not always have control over the communication language, as in the case of multiple KBs that have extensive overlap. In such cases, the communication language between KBs may be large, possibly resulting in a large interpolant. In Section 4 we provide an algorithm for partitioning theories that attempts to minimize the communication language between partitions.

3 Propositional Satisfiability

In this section we propose an algorithm for partition-based logical reasoning based on propositional satisfiability (SAT) search. We show that the complexity of computation is directly related to the size of the labels in the intersection graph.

3.1 A Partition-Based SAT Procedure

The algorithm we propose uses a SAT procedure as a subroutine and is back-track free. We describe the algorithm using database notation [51]. $\pi_{p_1,...,p_k} T$ is the *projection* operation on a relation T. It produces a relation that includes all the rows of T, but only the columns named $p_1, ..., p_k$ (suppressing duplicate rows). $S \bowtie R$ is the *natural join* operation on the relations S and R. It produces the cross product of S, R, selecting only those entries that are equal between identically named fields (checking $S.A = R.A$), and discarding those columns that are now duplicated (e.g., $R.A$ will be discarded).

The proposed algorithm shares some intuition with prime implicate generation (e.g., [36, 30]). Briefly, we first compute all the models of each of the partitions (akin to computing the implicates of each partition). We then use $\bowtie$ to combine the partition models into models for $\mathcal{A}$. The algorithm is presented in Figure 7.

```
PROCEDURE LINEAR-PART-SAT($\{\mathcal{A}_i\}_{i \leq n}$)
$\{\mathcal{A}_i\}_{i \leq n}$ a partitioning of the theory $\mathcal{A}$,
  1. $G_0 \leftarrow$ the intersection graph of $\{\mathcal{A}_i\}_{i \leq n}$. $G \leftarrow BREAK\text{-}CYCLES(G_0)$.
  2. For each $i \leq n$, let $L(i) = \bigcup_{(i,j) \in E} l(i,j)$.
  3. For each $i \leq n$, for every truth assignment $A$ to $L(i)$,
     perform SAT-search on $\mathcal{A}_i \cup A$, storing the result in a table $T_i(A)$.
  4. 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 $dist(i,1) < dist(j,1)$ ($\prec$ is a strict partial order).
  5. Iterate over $i \leq n$ in reverse $\prec$-order (the last $i$ is 1).
     For each $j \leq n$ that satisfies $(i,j) \in E$ and $i \prec j$, perform:
       • $T_i \leftarrow T_i \bowtie (\pi_{L(i)} T_j)$ (Join $T_i$ with those columns of $T_j$ that correspond to $L(i)$).
         If $T_i = \emptyset$, return FALSE.
  6. Return TRUE.
```

Figure 7: An algorithm for SAT of a partitioned propositional theory.

The iterated join that we perform takes time proportional to the size of the tables involved. We keep table sizes below $2^{|L(i)|}$ ($L(i)$ computed in step 2), by *projecting* every table before *joining* it with another. Soundness and completeness follow by an argument similar to that given for MP.

Theorem 3.1 (Soundness and Completeness) *Given a sound and complete SAT-search procedure, LINEAR-PART-SAT is sound and complete for SAT of partitioned propositional theories.*

PROOF See Appendix A.3.

3.2 Analyzing Satisfiability in LINEAR-PART-SAT

Let $\mathcal{A}$ be a partitioned propositional theory with n partitions. Let $m = |\mathcal{L}(\mathcal{A})|$, $L(i)$ the set of propositional symbols calculated in step 2 of LINEAR-PART-SAT, and $m_i = |\mathcal{L}(\mathcal{A}_i) \setminus L(i)|$ ($i \leq n$). Let $a = |\mathcal{A}|$ and k be the length of each axiom.

Lemma 3.2 *The time taken by LINEAR-PART-SAT to compute SAT for $\mathcal{A}$ is*

$$Time(n, m, m_1, ..., m_n, a, k, |L(1)|, ..., |L(n)|) = O(a * k^2 + n^4 * m + \sum_{i=1}^{n} (2^{|L(i)|} * f_{SAT}(m_i))),$$

where f_{SAT} is the time to compute SAT. If the intersection graph G_0 is a tree, the second argument in the summation can be reduced from $n^4 * m$ to $n * m$.

PROOF See Appendix A.4.

Corollary 3.3 *Let $\mathcal{A}$ be a partitioned propositional theory with n partitions, m propositional symbols, and intersection graph $G = (V, E, l)$. Let $d = max_{v \in V} d(v)$, where $d(v)$ is the degree of node v, and let $l = max_{i,j \leq n} |l(i,j)|$. Assume $P \neq NP$. If intersection graph G of $\mathcal{A}$ is a tree and all the partitions $\mathcal{A}_i$ have the same number of propositional symbols, then the time taken by the LINEAR-PART-SAT procedure to compute SAT for $\mathcal{A}$ is*

$$Time(m, n, l, d) = O(n * 2^{d*l} * f_{SAT}(\frac{m}{n})).$$

For example, if we partition a given theory $\mathcal{A}$ into only two partitions ($n = 2$), sharing l propositional symbols, the algorithm will take time $O(2^l * f_{SAT}(\frac{m}{2}))$. Assuming $P \neq NP$, this is a significant improvement over a simple SAT procedure, for every l that is small enough ($l < \frac{\alpha m}{2}$, and $\alpha \leq 0.582$ [42, 13]).

4 Decomposing a Logical Theory

The algorithms presented in previous sections assumed a given partitioning. In this section we address the critical problem of automatically decomposing a set of propositional or FOL clauses into a partitioned theory. Guided by the results of previous sections, we propose guidelines for achieving a good partitioning, and present a greedy algorithm that decomposes a theory following these guidelines.

4.1 A Good Partitioning

Given a theory, we wish to find a partitioning that minimizes the formula derived in Lemma 3.2. To that end, assuming $P \neq NP$, we want to minimize the following parameters in roughly the following order. For all $i \leq n$:

1. $|L(i)|$ - the total number of symbols contained in all links to/from node i. If G_0 is already a tree, this is the number of symbols shared between the partition $\mathcal{A}_i$ and the rest of the theory $\mathcal{A} \setminus \mathcal{A}_i$.

2. m_i - the number of symbols in a partition, less those in the links, i.e., in $\mathcal{A}_i \setminus L(i)$. This number is mostly influenced by the size of the original partition $\mathcal{A}_i$, which in turn is influenced by the number of partitions of $\mathcal{A}$, namely, n. Having more partitions will cause m_i to become smaller.

3. n - the number of partitions.

Also, a simple analysis shows that given *fixed* values for l, d in Corollary 3.3, the maximal n that maintains l, d such that also $n \leq ln2 * \alpha * m$ ($\alpha = 0.582$ [42, 13]) yields an optimal bound for LINEAR-PART-SAT. In Section 2.2 we saw that the same parameters influence the number of derivations we can perform in MP: $|L(i)|$ influences the interpolant size and thus the proof length, and m_i influences the number of deductions/resolutions we can perform. Thus, we would like to minimize the number of symbols shared between partitions and the number of symbols in each partition less those in the links.

The question is, how often do we get large n (many partitions), small m_i's (small partitions) and small $|L(i)|$'s (weak interactions) in practice. We believe that in domains that deal with engineered physical systems, many of the domain axiomatizations have these structural properties. Indeed, design of engineering artifacts encourages modularization, with minimal interconnectivity (see [2, 34, 12]). More generally, we believe axiomatizers of large corpora of real-world knowledge tend to try to provide structured representations following some of these principles.

4.2 Vertex Min-Cut in the Graph of Symbols

To exploit the partitioning guidelines proposed in the previous subsection, we represent our theory $\mathcal{A}$ using a *symbols graph* that captures the features we wish to minimize. $G = (V, E)$ is a symbols graph for theory $\mathcal{A}$ such that each vertex $v \in V$ is a symbol in $\mathcal{L}(\mathcal{A})$, and there is an edge between two vertices if their associated symbols occur in the same axiom of $\mathcal{A}$, i.e., $E = \{(a, b) \mid \exists \alpha \in \mathcal{A}$ s.t. a, b appear in $\alpha\}$.

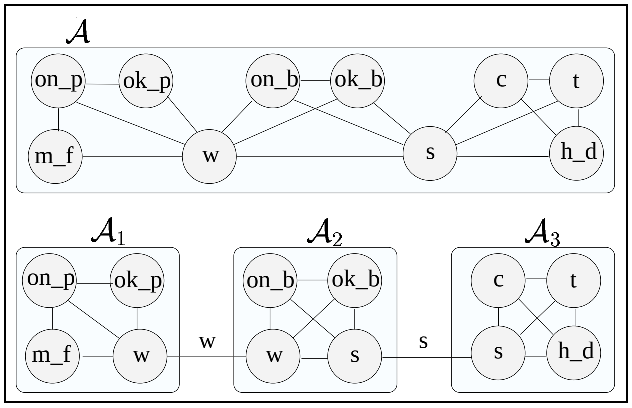

Figure 8: Decomposing $\mathcal{A}$'s symbols graph.

Figure 8 (top) illustrates the symbols graph of theory $\mathcal{A}$ from Figure 1 and the connected symbols graphs (bottom) of the individual partitions $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$. The symbols *ok_p*, *on_p*, *m_f*, *w*, *ok_b*, *on_b*, *s*, *c*, *t*, *h_d* are short for ok_pump, on_pump, man_fill, $water$, ok_boiler, on_boiler, $steam$, $coffee$, $teabag$, hot_drink, respectively. Notice that each axiom creates a clique among its constituent symbols. To minimize the number of symbols shared between partitions (i.e., $|L(i)|$), we must find partitions whose symbols have minimal *vertex separators* in the symbols graph.

We briefly describe the notion of a vertex separator. Let $G = (V, E)$ be an undirected graph. A set S of vertices is called an (a, b) *vertex separator* if $\{a, b\} \subset V \setminus S$ and every path connecting a and b in G passes through at least one vertex contained in S. Thus, the vertices in S split the path from a to b. Let $N(a, b)$ be the least cardinality of an (a, b) vertex separator. The *connectivity* of the graph G is the minimal $N(a, b)$ for any $a, b \in V$ that are not connected by an edge.

Even [22] described algorithms for finding minimum vertex separators, building on work of Dantzig and Fulkerson. We briefly review one of these algorithms before we use it to decompose our theories. It is shown in Figure 9. The algorithm is given two vertices, a, b, and an undirected graph, G. In this algorithm, we transform G into $\bar{G}$ and run a max-flow algorithm on it (lines 1–3). The produced flow, f, has a throughput of $N(a, b)$. To extract a minimal separator, we produce a *layered network* (see [22] p.97) from $\bar{G}, f$ in line 5. The layered network includes a subset of the vertices of $\bar{G}$. The set of edges between this set of vertices and the rest of $\bar{G}$ corresponds to the separator.

```
PROCEDURE MIN-V-SEP-A-B($G = (V, E), a, b$)

  1. Construct a digraph $\bar{G}(\bar{V}, \bar{E})$ as follows. For every $v \in V$
     put two vertices $v', v''$ in $\bar{V}$ with an edge $e_v = \overrightarrow{(v', v'')}$
     (internal edge). For every edge $e = (u, v)$ in $G$, put two
     edges $e' = \overrightarrow{(u'', v')}$ and $e'' = \overrightarrow{(u', v'')}$ in $\bar{G}$ (external
     edges).

  2. Define a network, with digraph $\bar{G}$, source $a''$, sink $b'$ and
     unit capacities for all the edges.

  3. Compute the maximum flow $f$ in the network. $N(a, b)$ is
     the throughput of $f$.

  4. Set the capacities of all the external edges in $\bar{G}$ to infinity.

  5. Construct the layered network $\{V_i\}_{i \leq l}$ from $\bar{G}$ using $f$.
     Let $S = \bigcup_{i \leq l} V_i$.

  6. Let $R = \{v \in V \mid v' \in S,\ v'' \notin S\}$. $R$ is a minimum
     $(a, b)$ vertex-separator in $G$.
```

Figure 9: An algorithm for finding a minimal separator between a and b in G.

Algorithms for finding maximal flow are abundant in the graph algorithms literature. Prominent algorithms for max-flow include the Simplex method, Ford and Fulkerson's [31], the push-relabel method of Goldberg and Tarjan [27] (time bound of $O(|V| * |E| * lg\frac{|V|^2}{|E|})$ and several implementations [11]), and Dinitz's algorithm [21]. When Dinitz's algorithm is used to solve the network problem, algorithm MIN-V-SEP-A-B is of time complexity $O(|V|^{\frac{1}{2}} * |E|)$ [22].

Finally, to compute the vertex connectivity of a graph and a minimum separator, without being given a pair (a, b), we check the connectivity of any c vertices (c being the connectivity of the graph) to all other vertices. When Dinitz's algorithm is used as above, this procedure takes time $O(c * |V|^{\frac{3}{2}} * |E|)$, where $c \geq 1$ is the connectivity of G. For the cases of $c = 0, 1$ there are well known linear time algorithms.

Figure 10 presents a greedy recursive algorithm that uses Even's algorithm to find *sets of vertices* that together separate a graph into partitions. The algorithm returns a set of symbols sets that determine the separate subgraphs. Different variants of the algorithm yield different structures

```
PROCEDURE SPLIT($G, M, l, a, b$)
$G = (V, E)$ is an undirected graph. $M$ is the limit on the num-
ber of symbols in a partition. $l$ is the limit on the size of links
between partitions. $a, b$ are in $V$ or are nil.

  1. If $|V| < M$ then return the graph with the single symbol
     set $V$.

  2. (a) If $a$ and $b$ are both nil, find a minimum vertex separator
     $R$ in $G$. (b) Otherwise, if $b$ is nil, find a minimum vertex
     separator $R$ in $G$ that does not include $a$. (c) Otherwise,
     find a minimum vertex separator $R$ in $G$ that separates $a$
     and $b$.
     If $R > l$ then return the graph with the single symbol set
     $V$.

  3. Let $G_1, G_2$ be the two subgraphs of $G$ separated by $R$,
     with $R$ included in both subgraphs.

  4. Create $G'_1, G'_2$ from $G_1, G_2$, respectively, by aggregating
     the vertices in $R$ into a single vertex $r$, removing all self
     edges and connecting $r$ with edges to all the vertices con-
     nected by edges to some vertices in $R$.

  5. Set $V^1 = SPLIT(G'_1, M, l, r, a)$ and $V^2 =$
     $SPLIT(G'_2, M, l, r, b)$.

  6. Replace $r$ in $V^1, V^2$ by the members of $R$. Return
     $V^1, V^2$.
```

Figure 10: An algorithm for generating symbol sets that define partitions.

for the intersection graph of the resulting partitioning. As is, SPLIT returns sets of symbols that result in a chain of partitions. We obtain arbitrary *trees*, if we change step 2(c) to find a minimum separator that does not include a, b (not required to separate a, b). We obtain arbitrary *graphs*, if in addition we do not aggregate R into r in step 4.

Proposition 4.1
Procedure SPLIT takes time $O(|V|^{\frac{5}{2}} * |E|)$.

PROOF See Appendix A.5.

Finally, to partition a theory $\mathcal{A}$, create its symbols graph G

and run SPLIT(G, M, l, nil, nil). For each set of symbols returned, define a partition $\mathcal{A}_i$ that includes all the axioms of $\mathcal{A}$ in the language defined by the returned set of symbols.

We know of no easy way to find an optimal selection of l (the limit on the size of the links) and M (the limit on the number of symbols in a partition) without having prior knowledge of the dependency between the number (and size) of partitions and l. However, we can find out when a partitioning no longer helps computation (compared to the best time bound known for SAT procedures [42, 13]). Our time bound for the procedure is lower than $\Theta(2^{\alpha m})$ when $l \leq \frac{\alpha m - \alpha m_i - lgn}{d}$ ($i = argmax_j m_j$). In particular, if $l > \frac{m}{2}$, a standard deterministic SAT procedure will be better. Hence, l and M are perhaps best determined experimentally.

In contrast to the SPLIT procedure, there are other approaches that can be taken. One approach that we have also implemented is a normalized cut algorithm [44], using the dual graph of the theory. The dual graph represents each axiom, rather than each symbol, as a node in the graph to be split. The advantage of this approach is that it preserves the distinction of an axiom. Also, since the min-cut algorithm is normalized, it helps preclude the creation of small isolated partitions by both maximize the similarity within partitions and minimizing the similarity between partitions. Finally, our reasoning algorithms and our computational analysis suggested a syntactic approach to decomposition. Semantic approaches are also possible along lines similar to [10] (Ch. on semantic resolution) or [48]. Such decomposition approaches may require different reasoning algorithms to be useful.

5 Related Work

There has been little work on the specific problem of exploiting structure in theorem proving and SAT search in the manner we propose, and little work on automatically partitioning logical theories for this purpose. Nevertheless, there are many areas of related work.

Work on formalizing and reasoning with *context* (e.g., [37, 1]) can be related to partition-based logical reasoning 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 [26]), There have been few reported successes with automated reasoning; [6] presents one example.

Many AI researchers have exploited some type of structure to improve the efficiency of reasoning (e.g., Bayes Nets [40], Markov decision processes [8], CSPs [19, 18], and model-based diagnosis [17]). There is also a vast literature in both clustering and decomposition techniques. Most relevant to our work, are CSP decomposition techniques that look for a *separation vertex* [18]. Also related is cutset conditioning [40]. In contrast, our work focuses on the possibility of using vertex separator *sets* as a generalization of the separation-vertex concept. Partly due to this generality, our work is the first to address the problem of defining guidelines and parameters for good decompositions of sets of axioms for the purpose of logical reasoning.

Decomposition has not been exploited in theorem proving until recently (see [4, 5]). We believe that part of the reason for this lack of interest has been that theorem proving has focused on mathematical domains that do not necessarily have structure that supports decomposition. Work on theorem proving has focused on decomposition for parallel implementations [5, 14, 50] and has followed decomposition methods guided by lookahead and subgoals, neglecting the types of structural properties we used here. Another related line of work focuses on combining logical systems (e.g., [38, 45, 3]). Contrasted with this work, we focus on interactions between theories with overlapping signatures, the efficiency of reasoning, and automatic decomposition.

Decomposition for propositional SAT has followed different tracks. Some work focused on heuristics for clause weighting or symbol ordering (e.g., [43, 20]). [39] suggested a decomposition procedure that represents the theory as a hypergraph of clauses and divides the propositional theory into two partitions (heuristically minimizing the number of hyperedges) modifying ideas described in [25]. [14] developed an algorithm that partitions a propositional theory into connected components. Both [14, 39] performed experiments that demonstrated a decrease in the time required to prove test sets of axioms unsatisfiable.

6 Conclusions

We have shown that decomposing theories into partitions and reasoning over those partitions has computational advantages for theorem provers and SAT solvers. Theorem proving strategies, such as resolution, can use such decompositions to constrain search. Partition-based reasoning will improve the efficiency of propositional SAT solvers if the theory is decomposable into partitions that share only small numbers of symbols. We have provided sound and complete algorithms for reasoning with partitions of related logical axioms, both in propositional and FOL. Further, we analyzed the effect of partition-based logical reasoning on resolution-based inference, both with respect to proof search space size, and with respect to the length of a proof. We also analyzed the performance of our SAT algorithm and showed that it takes time proportional to SAT solutions on individual partitions and an exponent in the size of the links between partitions. Both algorithms can gain further time efficiency through parallel processing.

Guided by the analysis of our SAT algorithm, we suggested guidelines for achieving a good partitioning and proposed an algorithm for the automatic decomposition of theories that tries to minimize identified parameters. This algorithm generalizes previous algorithms used to decompose CSPs by finding single-vertex separators.

Our work was motivated in part by the problem of reasoning with large multiple KBs that have overlap in content. The results in this paper address some of the theoretical principles that underly such partition-based reasoning. In future work, we plan an experimental analysis of Stanford KSL's and SRI's KBs to analyze structure in these KBs, to test the effectiveness of our automatic partitioning algorithms, and to investigate the effectiveness of proposed partition-based reasoning algorithms. We are also involved in further theoretical investigation.

A Proofs

A.1 Proof of Theorem 2.2

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 2.1), 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. ■

Definition A.2 *For a partitioning $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$, we say that a tree $G = (V, E, l)$ is* properly labeled, *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 $\mathcal{L}(l(i,j)) \supseteq \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$.*

We will show that all intersection graphs are properly labeled. First, the following lemma provides the main argument behind all of the completeness proofs in this paper.

Lemma A.3 *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned theory and assume that the graph G is a tree that is properly labeled for the partitioning $\{\mathcal{A}_i\}_{i \leq n}$. Let $k \leq n$ and let $Q \in \mathcal{L}(\mathcal{A}_k \cup \bigcup_{(k,i) \in E} l(k,i))$ be a sentence. If $\mathcal{A} \models Q$, then MP will output YES.*

Proof of Lemma A.3. We prove the lemma 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. Assume that we proved the lemma for $|V| \leq n-1$ and we prove the lemma for $|V| = n$.

In G, k has c neighbors, $i_1, ..., i_c$. (k, i_1) 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 Lemmma 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. $\bigcup_{(i_1,j) \in E, j \neq k} l(i_1, j) \supseteq \mathcal{L}(\mathcal{B}_2 \cup \mathcal{A}_{i_1}) \cap \mathcal{L}(\mathcal{B}_1 \setminus \mathcal{A}_{i_1})$ 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 ψ will be proved in $\mathcal{A}_i$ (after some formulae were sent to it from the other partitions in G_1, B_1).

At this point, our algorithm will send ψ to $\mathcal{A}_k$ because $\psi \in \mathcal{L}(l(k, i_1))$ because G is properly labeled for $\mathcal{A}, G$. Since $B_2 \vdash \psi \Rightarrow Q$, then by the induction hypothesis applied to G_2, B_2 ($\psi \Rightarrow Q \in \mathcal{L}(\mathcal{A}_k \cup_{k,i) \in E} l(k,i))$ at some point $\psi \Rightarrow Q$ will be proved in $\mathcal{A}_k$ (after some message passing). Thus, at some point $\mathcal{A}_k$ will prove Q. ■

Proof of Theorem 2.2. All we are left to prove is that the intersection graph G is properly labeled. But if G is the intersection graph of the partitioning $\{\mathcal{A}_i\}_{i \leq n}$ then $l(i,j) = \mathcal{L}(\mathcal{A}_i) \cap \mathcal{L}(\mathcal{A}_j)$. If for $(i,j) \in E$ $\mathcal{L}(l(i,j)) \not\supseteq \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$, with $\mathcal{B}_1, \mathcal{B}_2$ the theories on the two sides of (i,j) in the tree G, then there are $\mathcal{A}_x, \mathcal{A}_y$ in $\mathcal{B}_1, \mathcal{B}_2$, respectively, such that $(x,y) \in E$ and $x \neq i$ or $y \neq j$. Since G is connected (it is a single tree), this means there is a cycle in G, contradicting G being a tree. Thus $\mathcal{L}(l(i,j)) \supseteq \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$ and G is properly labeled. The proof follows from Lemma A.3. ■

A.2 Proof of Theorem 2.3

Soundness is immediate, using the same argument as for Theorem 2.2. For completeness, first notice that the graph output by BREAK-CYCLES is always a tree, because

BREAK-CYCLES will not terminate if there is still a cycle in G. Now, we need the following lemma.

Lemma A.4 *Let $G' = (V, E', l')$ be a tree resulting from applying BREAK-CYCLES to $G = (V, E, l)$ and $\{\mathcal{A}_i\}_{i \leq n}$. Then G' is properly labeled for this partitioning.*

Proof of Lemma A.4. Assume there is a symbol p in $\mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$ that is not in $l(i, j)$, and let $\mathcal{A}_x, \mathcal{A}_y$ be partitions on the two sides of (i, j) that include sentences with the symbol p. We will prove that throughout the run of the BREAK-CYCLES algorithm there is always a path in G' (we start with $G' = G$) between $\mathcal{A}_x, \mathcal{A}_y$ that has p showing on all the edge labels. Call such a path a *good path*.

Obviously we have a good path in G, because we have $(x, y) \in E$ and $p \in l(x, y)$ (because G is the intersection graph of $\mathcal{A}_1, ..., \mathcal{A}_n$). Let us stop the algorithm at the first step in which G' does not have a good path (assuming there is no such path, or otherwise we are done). In the last step we must have removed an arc (a, b) (which was on a good path) to cause G' to not have a good path. Since $p \in l(a, b)$ and (a, b) is in a cycle $\langle (b, a_1), (a_1, a_2), ..., (a_c, a), (a, b) \rangle$ (this is the only reason we removed (a, b)), we added $l(a, b)$ to the labels of the rest of this cycle. In particular, now the labels of $(b, a_1), (a_1, a_2), ..., (a_c, a)$ include p. Replacing (a, b) in the previous good path by this sequence, we find a path in the new G' that satisfies our required property. This is a contradiction to having assumed that there is no such path at this step. Thus, there is no such p as mentioned above and $\mathcal{L}(l(i, j)) \supseteq \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$. ■

Proof of Theorem 2.3. The proof of Theorem 2.3 follows immediately from Lemma A.3 and Lemma A.4. ■

A.3 Proof of Theorem 3.1

Proof of Theorem 3.1. For each partition $\mathcal{A}_i, i \leq n$, lines 1 – 3 perform the equivalent of finding all the models of $\mathcal{A}_i$ and storing their truth assignments to the symbols of $\mathcal{L}(L(i))$ in T_i. ($L(i)$ specifies the columns, thus each row corresponds to a truth assignment.) This is equivalent to finding the implicates of the theory $\mathcal{A}_i$ in the sublanguage $\mathcal{L}(L(i))$. Thus, if A_i is the DNF of the set of implicates $(\alpha_1(p_{j_1}, ..., p_{j_{l_i}}) \vee ... \vee \alpha_{a_i}(p_{j_1}, ..., p_{j_{l_i}}))$, then T_i initially includes the set of models of A_i in the sublanguage $\mathcal{L}(L(i))$, namely, $[\![A_i]\!]_{\mathcal{L}(L(i))}$.

The *natural join* operation ($\bowtie$) then creates all the consistent combinations of models from $[\![A_i]\!]_{\mathcal{L}(L(i))}$ and $[\![A_j]\!]_{\mathcal{L}(L(j))}$. This set of consistent combinations is the set of models of $A_i \cup A_j$. Thus, $T_i \bowtie T_j \equiv [\![(A_i \cup A_j)]\!]_{\mathcal{L}(L(i) \cup L(j))}$.

Finally, the *projection* operation restricts the models to the sublanguage $\mathcal{L}(L(i))$, getting rid of duplicates in the sublanguage. This is equivalent to finding all the implicates of $A_i \wedge A_j$ in the sublanguage $\mathcal{L}(L(i))$. Thus, $\pi_{L(i)}(T_i \bowtie T_j) \equiv [\![\{\varphi \in \mathcal{L}(L(i)) \mid A_i \cup A_j \models \varphi\}]\!]_{\mathcal{L}(L(i))}$.

To see that the algorithm is sound and complete, notice that the it does the analogous operations to our forward message-passing algorithm MP (Figure 3). We break the cycles in G_0 (creating G) and perform forward reasoning as in MP, using the set of implicates instead of online reasoning in each partition: instruction 3 in MP is our projection ("$\mathcal{A}_i \models \varphi$ and $\varphi \in \mathcal{L}(l(i, j))$") and then join ("add φ to the set of axioms of $\mathcal{A}_j$"). Since $T_i \bowtie T_j \equiv [\![(A_i \cup A_j)]\!]_{\mathcal{L}(L(i) \cup L(j))}$, joining corresponds to sending all the messages together. Since $\pi_{L(i)}(T_i \bowtie T_j) \equiv [\![\{\varphi \in \mathcal{L}(L(i)) \mid A_i \cup A_j \models \varphi\}]\!]_{\mathcal{L}(L(i))}$, projection corresponds to sending only those sentences that are allowed by the labels.

By Theorem 2.3, LINEAR-PART-SAT is sound and complete for satisfiability of $\mathcal{A}$. ■

A.4 Proof of Lemma 3.2

Proof of Lemma 3.2. Let $\mathcal{A}$ be a partitioned propositional theory with n partitions. Let m be the total number of propositional symbols in $\mathcal{A}$, $L(i)$ the set of propositional symbols calculated in step 2 of LINEAR-PART-SAT, and m_i the number of propositional symbols mentioned in $\mathcal{A}_i \setminus L(i)$ ($i \leq n$). Let us examine procedure LINEAR-PART-SAT (Figure 7) step by step, computing the time needed for computation.

Computing the intersection graph takes $O(a * k^2)$ time, where k is the number of propositional symbol in each axiom (for 3SAT, that is 3), because we check and add k^2 edges to G_0 for each axiom.

BREAK-CYCLES' loop starts by finding a minimal-length cycle, which takes time $O(n)$ (BFS traversal of n vertices). Finding the optimal a in line 2 takes time $O((c * m) * c)$, where c is the length of the cycle found (union of two labels takes at most $O(m)$ time). Finally, since a tree always satisfies $|E| = |V| - 1$, breaking all the cycles will require us to remove $|E| - (|V| - 1)$ edges. Thus, the loop will run $|E| - (|V| - 1)$ times (assuming the graph G_0 is connected). An upper bound on this algorithm's performance is then $O(n^2 * (n^2 * m)) = O(n^4 * m)$ (because $c \leq n$ and and $|E| \leq |V|^2 = n^2$).

Step 2 of LINEAR-PART-SAT takes time $O(n * m)$, since there are a total of $n - 1$ edges in the graph G (G is a tree with n vertices) and every label is of length at most m.

Checking the truth assignments in step 3 takes time $\sum_{i=1}^{n} 2^{|L(i)|}$ per satisfiability check of $\mathcal{A}_i \cup A$, because there are $2^{|L(i)|}$ truth assignments for each $i \leq n$. Since

$\mathcal{A}_i \cup A$ has only m_i free propositional variables, (A is an assignment of truth values to $|L(i)|$ variables), $\mathcal{A}_i \cup A$ is reducible (in time $O(|A|)$) to a theory of smaller size with only m_i propositional variables. If the time needed for a satisfiability check of a theory with m variables is $O(f_{SAT}(m))$, then the time for step 3 is

$$O(\sum_{i=1}^{n}(2^{|L(i)|} * f_{SAT}(m_i)))$$

Finding the relation $\prec$ takes $O(n)$ as it is easily generated by a BFS through the tree.

Instruction 5 performs a *projection* and *join*, which takes time $O(2^{|L(i)|})$ (the maximal size of the table). Since the number of iterations over $i \leq n$ and j being a child of i is $n-1$ (there are only $n-1$ edges), we get that the total time for this step is $O(\sum_{i=1}^{n} 2^{|L(i)|})$.

Summing up, the worst-case time used by the algorithm is

$$\begin{aligned}&Time(n, m, m_1, ..., m_n, a, k, L(1), ..., L(n)) = \\ &\quad O(a * k^2 \ + \ n^4 * m \ + \ n * m \ + \\ &\quad\quad \textstyle\sum_{i=1}^{n}(2^{|L(i)|} * f_{SAT}(m_i)) \ + \ n \ + \ \sum_{i=1}^{n} 2^{|L(i)|}) = \\ &\quad O(a * k^2 \ + \ n^4 * m \ + \ \textstyle\sum_{i=1}^{n}(2^{|L(i)|} * f_{SAT}(m_i))).\end{aligned}$$

We can reduce the second argument (in the last formula) from $n^4 * m$ to $n * m$, if the intersection graph G_0 is already a tree. ■

A.5 Proof of Proposition 4.1

Proof of Proposition 4.1. Finding a minimum vertex separator R in G takes time $O(c * |V|^{\frac{3}{2}} * |E|)$. Finding a minimum separator that does not include s is equivalent to having s be the only source with which we check connectivity (in Even's algorithm). Thus, this can be done in time $O(|V|^{\frac{3}{2}} * |E|)$. Finding a minimum separator that separates s from t takes time $O(|V|^{\frac{1}{2}} * |E|)$. In the worst case, each time we look for a minimum s-separator ($t = nil$), we get a very small partition, and a very large one. Thus, we can apply this procedure $O(|V|)$ times. Summing up the time taken for each application of the procedure yields $O(|V| * |V|^{\frac{3}{2}} * |E| + c * |V|^{\frac{3}{2}} * |E|) = O(|V|^{\frac{5}{2}} * |E|)$. ■

Acknowledgements

We wish to thank Rada Chirkova and Tom Costello for helpful comments on the contents of this paper. We would also like to thank Mike Genesereth, Nils Nilsson and John McCarthy for many interesting discussions on more general topics related to this work. Finally, we thank the KR2000 reviewers for their helpful reviews. This research was supported in part by DARPA grant N66001-97-C-8554-P00004 and by AFOSR grant AF F49620-97-1-0207.

References

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

[2] E. Amir. Object-Oriented First-Order Logic. *Linköping Electronic Articles in Computer and Information Science (http://www.ida.liu.se/ext/etai)*, 4, 1999. Under review in the RAC publication area. A preliminary version appeared in NRAC'99, an IJCAI-99 workshop.

[3] F. Baader and K. U. Schulz. Unifi cation in the union of disjoint equational theories: Combining decision procedures. In *11th Intl. conf. on automated deduction*, volume 607 of *LNAI*, pages 50–65. Springer-Verlag, 1992.

[4] M. P. Bonacina. A taxonomy of theorem-proving strategies. In *Artificial Intelligence Today – Recent Trends and Developments*, volume 1600 of *LNAI*, pages 43–84. Springer, 1999.

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

[6] P.E. Bonzon. A refective proof system for reasoning in contexts. In *Proc. Nat'l Conf. on Artificial Intelligence (AAAI '97)*, pages 398–403, 1997.

[7] R. Boppana and M. Sipser. The complexity of fi nite functions. In *Handbook of Theoretical Computer Science*, volume 1. Elsevier and MIT Press, 1990.

[8] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *14th Intl. Joint Conf. on Artificial Intelligence (IJCAI '95)*, pages 1104–1111, 1995.

[9] A. Carbone. Interpolants, cut elimination and fbw graphs for the propositional calculus. *Annals of Pure and Applied Logic*, 83(3):249–299, 1997.

[10] Chin-Liang Chang and R. Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[11] B. V. Chekassky and A. V. Goldberg. On implementing the push-relabel method for the maximum fbw problem. *Algorithmica*, 19(4):390–410, 1997.

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

[13] S. A. Cook and D. G. Mitchell. Finding hard instances of the satisfi ability problem: a survey. In *Dimacs Series in Discrete Mathamatics and Theoretical Computer Science*, volume 35. AMS, 1997.

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

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

[16] W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. of Symbolic Logic*, 22:269–285, 1957.

[17] A. Darwiche. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research*, 8:165–222, 1998.

[18] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

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

[20] R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *Intl. Conf. on Knowledge Representation and Reasoning (KR '94)*, pages 134–145. Morgan Kaufmann, 1994.

[21] E. A. Dinitz. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.

[22] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

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

[24] J. J. Finger. Exploiting constraints in design synthesis. Technical Report STAN-CS-88-1204, Department of Computer Science, Stanford University, Stanford, CA, 1987.

[25] G. Gallo and G. Urbani. Algorithms for testing the satisfiability of propositional formulae. *J. of Logic Programming*, 7:45–61, 1989.

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

[27] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. of the ACM*, 35(4):921–940, 1988.

[28] A. Haken. The intractability of resolution. *theoretical computer science*, 39:297–308, 1985.

[29] G. Huang. Constructing Craig interpolation formulas. In *Conference on computing and combinatorics*, pages 181–190, 1995.

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

[31] L. R. Ford Jr. and D. R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

[32] J. Krajiček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. of Symbolic Logic*, 62(2):457–486, 1997.

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

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

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

[36] P. Marquis. Knowledge compilation using theory prime implicates. In *14th Intl. Joint Conf. on Artificial Intelligence (IJCAI '95)*, pages 837–843, 1995.

[37] J. McCarthy and S. Buvač. Formalizing Context (Expanded Notes). In A. Aliseda, R.J. van Glabbeek, and D. Westerståhl, editors, *Computing Natural Language*, volume 81 of *CSLI Lecture Notes*, pages 13–50. Center for the Study of Language and Information, Stanford U., 1998.

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

[39] T. J. Park and A. Van Gelder. Partitioning methods for satisfiability testing on large formulas. In *Proc. Intl. Conf. on Automated Deduction (CADE-13)*, pages 748–762. Springer-Verlag, 1996.

[40] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[41] D. A. Plaisted. The search efficiency of theorem proving strategies. In *Proc. Intl. Conf. on Automated Deduction (CADE-12)*, pages 57–71, 1994.

[42] I. Schiermeyer. Pure literal look ahead: an $O(1,497^n)$ 3-satisfiability algorithm (extended abstract). Technical report, University of Köln, 1996. Workshop on the Satisfiability Problem, Siena April 29-May 3.

[43] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *13th Intl. Joint Conf. on Artificial Intelligence (IJCAI '93)*, 1993.

[44] J. Shi and J. Malik. Normalized cuts and image segmentation. In *Proceedings of IEEE CVPR*, pages 731–737, 1997.

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

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

[47] J. R. Slagle, C.-L. Chang, and R. C. T. Lee. Completeness theorems for semantic resolution in consequence-finding. In *1st Intl. Joint Conf. on Artificial Intelligence (IJCAI '69)*, pages 281–285, 1969.

[48] J. Slaney and T.J. Surendonk. Combining finite model generation with theorem proving: Problems and prospects. In *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 141–156. Kluwer, 1996.

[49] M. E. Stickel. A Prolog Technology Theorem Prover: a new exposition and implementation in Prolog. *Theoretical Computer Science*, 104:109–128, 1992.

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

[51] J. D. Ullman. *Principles of Database and knowledge-base systems*, volume 1. Computer Science Press, 1988.