Available online at www.sciencedirect.com
SCIENCE DIRECT®
Artificial Intelligence 162 (2005) 49–88
Artificial Intelligence
www.elsevier.com/locate/artint

Partition-based logical reasoning for first-order and propositional theories

Eyal Amir [a], Sheila McIlraith [b,*]

[a] *Computer Science Department, University of Illinois, Urbana-Champaign, Urbana, IL 61801, USA*
[b] *Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3H5, Canada*

Received 20 November 2001; accepted 5 November 2004

Available online 15 December 2004

Abstract

In this paper we show how tree decomposition can be applied to reasoning with first-order and propositional logic theories. Our motivation is two-fold. First, we are concerned with how to reason effectively with multiple knowledge bases that have overlap in content. Second, we are concerned with improving the efficiency of reasoning over a set of logical axioms by partitioning the set with respect to some detectable structure, and reasoning over individual partitions either locally or in a distributed fashion. To this end, we provide algorithms for partitioning and reasoning with related logical axioms in propositional and first-order logic.

Many of the reasoning algorithms we present are based on the idea of passing messages between partitions. We present algorithms for both forward (data-driven) and backward (query-driven) message passing. Different partitions may have different associated reasoning procedures. We characterize a class of reasoning procedures that ensures completeness and soundness of our message-passing algorithms. We further provide a specialized algorithm for propositional satisfiability checking with partitions. Craig's interpolation theorem serves as a key to proving soundness and completeness of all of these algorithms. An analysis of these algorithms emphasizes parameters of the partitionings that influence the efficiency of computation. We provide a greedy algorithm that automatically decomposes a set of logical axioms into partitions, following this analysis.

* Corresponding author.
E-mail addresses: eyal@cs.uiuc.edu (E. Amir), sheila@cs.toronto.edu (S. McIlraith).

0004-3702/$ – see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.artint.2004.11.004

Keywords: Reasoning with structure; Theorem proving; First-order logic; SAT; Tree decomposition; Graphical models; Parallel computation; Distributed computation

1. Introduction

There is growing interest in building large knowledge bases (KBs) of everyday knowledge about the world, teamed with theorem provers or other reasoners 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) (e.g., [51]) and by SRI (e.g., [24]). 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. Our algorithms exploit the idea of tree decomposition (e.g., [7]), extending it to propositional and first-order logic (FOL) theorem proving. We provide forward (data-driven) and backward (query-driven) message-passing algorithms over theories that are partitioned into a structure very similar to a join-tree [66], specializing them for resolution theorem proving. We also provide an algorithm for partition-based propositional satisfiability (SAT). Our message-passing algorithms are designed so that, without loss of generality, reasoning within a partition can be realized by an arbitrary consequence-finding engine [67]. We characterize a class of reasoning procedures that ensures completeness and soundness of our algorithms. We use Craig's interpolation theorem [30] to prove the soundness and completeness of all our message-passing algorithms with respect to this class of procedures. It is also used to prove the soundness and completeness of our propositional satisfiability algorithm. We investigate the impact of these algorithms on resolution-based inference, and analyze the computational complexity for our partition-based SAT algorithm.

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 suggests parameters of partitionings that influence the computation of our algorithms: the number of nonlogical symbols included in the communication between partitions, the size of each partition, and the topology of the partitions graph. This observation guides us to propose a generic algorithm for decomposing logical theories into partitions, and a greedy algorithm that tries to optimize these parameters.

Surprisingly, there has been little work on the specific problem of exploiting structure in FOL theorem proving in the manner we propose. We speculate that this might be attributed

[1] In this paper, every set of axioms is a *theory* (and vice versa). Also, unless stated otherwise, theories, axioms and KBs are in first-order logic.

to the fact that FOL theorem proving has traditionally examined mathematics domains, that do not necessarily have structure that supports decomposition. Nevertheless, tree decomposition methods similar to those we apply here have been used successfully in reasoning with Bayes networks (e.g., [66,83]), constraint satisfaction problems (e.g., [37]), propositional reasoning (e.g., [38,86]) and originally in dynamic programming [12]. The common insight is that when knowledge can be partitioned into clusters that interact in a tree-like manner, reasoning can be accomplished in time that is exponential in a graph parameter known as *tree-width* that captures the size of the clusters relative to the original problem graph [7]. Where possible, we adopt this common terminology in our paper, to relate our work to previous contributions.

The rest of the paper is organized as follows. Section 2 describes our message-passing algorithms and sufficient conditions for their soundness and completeness. In Section 3 we specialize these algorithms to theorem proving using resolution and discuss the efficiency of message-passing. Section 4 offers an algorithm for propositional satisfiability and analyzes its computational complexity. Section 5 presents an algorithm for decomposing a logical theory. Finally, Section 6 discusses some related work. Some of the results in this paper appeared previously in [5,78].

2. Partition-based theorem proving

In this section we address the problem of how to reason with an already partitioned propositional or FOL theory using theorem proving. In particular, we propose forward and backwards message-passing algorithms, in the spirit of Pearl [83]. We further identify conditions underwhich partition-specific theorem proving results in sound and complete partition-based logical reasoning.

We define the following terminology. $\{\mathcal{A}_i\}_{i\leqslant n}$ 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*, $L(\mathcal{A}_i)$ is its signature (the set of non-logical symbols), and $\mathcal{L}(\mathcal{A}_i)$ is its language (the set of formulae built with $L(\mathcal{A}_i)$). Each partitioning defines a labeled graph $G = (V, E, l)$, which we call the *intersection graph*. In the intersection graph, each node i corresponds to an individual partition $\mathcal{A}_i$, $(V = \{1, \ldots, n\})$, two nodes i, j are linked by an edge if $L(\mathcal{A}_i)$ and $L(\mathcal{A}_j)$ have a symbol in common $(E = \{(i, j) \mid L(\mathcal{A}_i) \cap L(\mathcal{A}_j) \neq \emptyset\})$. 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, $l(i, j) = \emptyset$.

We illustrate the notion of a partitioning in terms of the simple propositional theory $\mathcal{A}$, depicted on the left of Fig. 1 (this is the clausal form of the theory presented with material implication in Fig. 2). These axioms capture the functioning of aspects of an espresso machine. The first 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 filled manually while the pump is on. The next four axioms denote that there is steam if and only if the boiler is OK and is on, and there is a

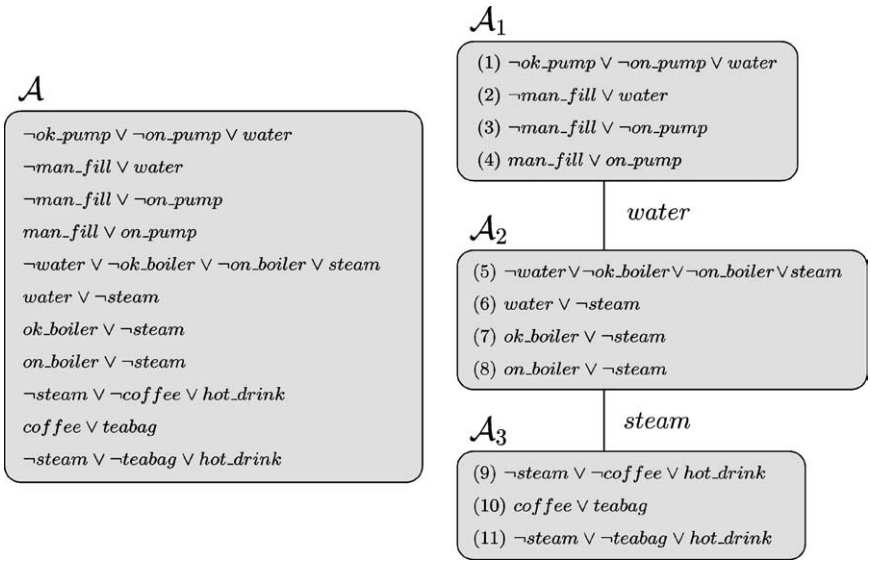

Fig. 1. A partitioning of $\mathcal{A}$ and its intersection graph.

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

Fig. 2. Axiomatization of a simplified espresso machine.

supply of water. The final three axioms denote that there is always either coffee or tea, and that steam and coffee (or tea) result in a hot drink.

The right-hand side of Fig. 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 $(1, 2)$ and $(2, 3)$ 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. Fig. 3 describes our forward message-passing algorithm, FORWARD-M-P (MP), for finding the truth value of query formula $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leqslant n$, given partitioned theory $\mathcal{A}$ and graph $G = (V, E, l)$. G may be the intersection graph of $\mathcal{A}$, but is not always so.

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

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

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

Fig. 3. A forward message-passing algorithm.

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

This algorithm exploits consequence finding (step (2a)) to perform reasoning in the individual partitions. Consequence finding was defined by Lee [67] to be the problem of finding all the logical consequences of a theory or sentences that subsume them. Recall, in *clausal* FOL, φ subsumes ψ if there is a substitution θ such that $\varphi\theta\subset\psi$. φ strictly subsumes ψ if φ subsumes ψ and ψ does not subsume φ.

Theorem 2.4 proves the soundness and completeness of our MP algorithm. It requires each of the reasoners in step (2) to be sound and complete.

Definition 2.2 (*Completeness for consequence finding*). Given a set of formulae $\mathcal{A}$ and a reasoning procedure $\mathfrak{R}$, $\mathfrak{R}$ is complete for consequence finding iff for every clause φ, that is a non-tautologous logical consequence of $\mathcal{A}$, $\mathfrak{R}$ derives a clause ψ from $\mathcal{A}$ such that ψ subsumes φ.

Furthermore, we say that $\mathfrak{R}$ is *complete for consequence finding in FOL* (as opposed to *clausal* FOL) iff for every non-tautologous logical consequence φ of $\mathcal{A}$, $\mathfrak{R}$ derives a logical consequence ψ of $\mathcal{A}$ such that $\psi\models\varphi$ and $\psi\in\mathcal{L}(\varphi)$.

In Section 3.1 we show that every reasoning procedure that is complete for consequence finding in clausal FOL can be converted to a reasoning procedure that is complete for consequence finding in FOL. In propositional logic the two conditions are identical.

Consequently, we can use any sound and complete consequence-finding algorithm for reasoning within an individual partition in MP. This is not restricted to variants of resolution. Nevertheless, the *resolution rule* is complete for clausal consequence finding (e.g., [67,98]) and the same is true for several variants of *linear resolution* such as the ones that are described by Inoue [61] and Minicozzi and Reiter [80]. A weaker version of complete-

Using FORWARD-M-P to prove *hot_drink*

Partition	Resolve	Generating	
$\mathcal{A}_1$	(2), (4)	*on_pump* $\vee$ *water*	(m1)
$\mathcal{A}_1$	(m1), (1)	$\neg$*ok_pump* $\vee$ *water*	(m2)
$\mathcal{A}_1$	(m2), (12)	*water*	(m3)
	$\Longrightarrow$	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)
	$\Longrightarrow$	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. 4. A proof of *hot_drink* from $\mathcal{A}$ in Fig. 1 after asserting *ok_pump* (12) in $\mathcal{A}_1$ and *ok_boiler* (13), *on_boiler* (14) in $\mathcal{A}_2$.

ness for consequence finding is also true for *semantic resolution* [99] and *set-of-support resolution*. We discuss the case of using resolution further in Section 3.

In addition, there are reasoning methods that focus on a given sublanguage as discussed in [18,52,61], and also [38,39,64,70,75]. An example of such restricted consequence finders is a prime implicate generator over a sublanguage. (Recall, a clause, φ, is a *prime implicate* of a theory T if $T \models \varphi$ and no formula that strictly subsumes φ is entailed from T.) Such consequence finders are commonly used for prime implicate generation in applications such as diagnosis and abduction [77]. Consequence finders that focus on a sublanguage can be directly used in MP for reasoning within partitions. Alternatively, they can be used in a batch mode to generate select consequences in the sublanguage and then send the messages in batch. In Fig. 4 we illustrate an execution of MP using resolution.

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 2.3 (Craig's interpolation theorem [30]). *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 [98] and use resolution theorem proving [60,98] with or without equality [30,31] (all after proper reformulation of the theorem).

Theorem 2.4 (Soundness and completeness). *Let $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$ be a partitioned theory with the intersection graph G being a tree* (*i.e., no cycles*)*. Let $k \leqslant n$ and φ a sentence in $\mathcal{L}(\mathcal{A}_k)$. If the reasoning procedure in each partition is sound and complete for consequence finding in FOL* (*as defined in Definition* 2.2)*, then $\mathcal{A} \models \varphi$ iff MP outputs YES.*

Proof. See Appendix A.1.

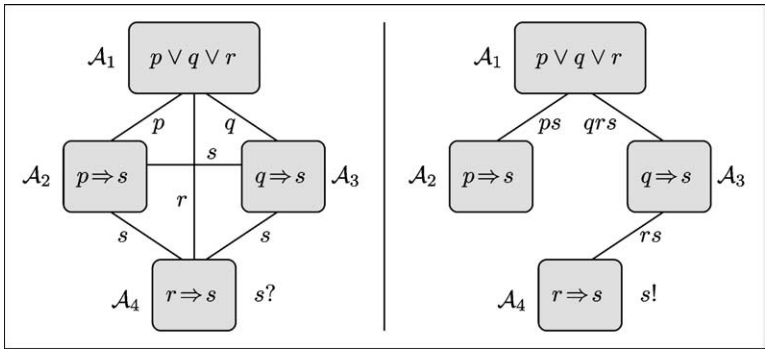

Fig. 5. An intersection graph before (left) and after (right) applying BREAK-CYCLES.

Note that Theorem 2.4 requires the intersection graph of $\mathcal{A}$ to be a tree. 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. Fig. 5 illustrates the problem. The left-hand side of Fig. 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 can 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. In particular, we would like the resultant tree to have a *proper labeling* for the given partitioning.

Definition 2.5 (*Proper labeling*). For a partitioning $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$, we say that associated tree $G = (V, E, l)$ has 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 $l(i, j) \supseteq L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$.

Note that this property is analogous to the *running intersection property* used in join-tree algorithms for inference in Bayes networks (e.g., [11,33,96]) and constraint satisfaction problems (e.g., [37,58]). The running intersection property is defined with respect to partitions or cluster whose contents are individual logical proposition, rather than a set of logical axioms formed from such logical propositions. In our context, the running intersection property requires that if a symbol s appears in A_i and A_j, then s appears in all the *partitions* on the tree-path between A_i and A_j. In contrast, the proper labeling property is defined with respect to the *language* $L(\mathcal{A}_i)$ of a partition $\mathcal{A}_i$, the partition itself being a set of logical axioms. As such, proper labeling is a condition that is applied only to the *links* on a path and not the partitions themselves.

The following lemma provides the main argument behind most of the completeness proofs in this paper.

Lemma 2.6. *Let $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$ be a partitioned theory and assume that the associated graph G is a tree that has a proper labeling for the partitioning $\{\mathcal{A}_i\}_{i \leqslant n}$. Also assume*

PROCEDURE BREAK-CYCLES($G = (V, E, l)$)

(1) Find a minimal-length cycle of nodes $v_1, \ldots, v_c$ ($v_1 = v_c$) in G. If there are no cycles, return G.
(2) Select index a s.t. $a < c$ and $\sum_{a \neq j < c} |l(v_j, v_{j+1}) \cup l(v_a, v_{a+1})|$ is minimal (the label of (v_a, v_{a+1}) adds a minimal number of symbols to the rest of the cycle).
(3) For all $j < c$, $j \neq a$, set $l(v_j, v_{j+1}) \leftarrow l(v_j, v_{j+1}) \cup l(v_a, v_{a+1})$.
(4) Set $E \leftarrow E \setminus \{(v_a, v_{a+1})\}$, $l(v_a, v_{a+1}) \leftarrow \emptyset$ and go to (1).

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

that each of the reasoning procedures used in MP is complete for consequence finding (*as defined in Definition* 2.2). *Let* $k \leqslant n$ *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 outputs YES.*

Proof. See Appendix A.1.

Observe that Theorem 2.4 is overly narrow. Even when the intersection graph is not a tree, MP can be sound and complete.

Algorithm BREAK-CYCLES, shown in Fig. 6, performs a transformation that produces a tree with a proper labeling from any labeled graph. ($|X|$ denotes the cardinality of a set X.) It is of particular importance when we are given a set of KBs that we cannot merge or restructure. Note that Section 5 gives a more general algorithm and treats the case where such restructuring is possible.

Using BREAK-CYCLES, we can transform the graph depicted on the left-hand side of Fig. 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 when removing the edge $(4, 1)$), but its removal influences the labels of only one cycle.

Theorem 2.7 (Soundness and completeness). *Let* $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$ *be a partitioned theory with intersection graph* G. *Let* $k \leqslant n$ *and* φ *a sentence in* $\mathcal{L}(\mathcal{A}_k)$. *If the reasoning procedure in each partition is sound and complete for consequence finding* (*as defined in Definition* 2.2), *then* $\mathcal{A} \models \varphi$ *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 $O(|E|^2 \cdot m)$ (where m is the number of symbols in $L(\mathcal{A})$). The rationale is roughly as follows: There are at most $|E|$ cycles that can be *broken*, step (1) takes $O(|E|)$ time, step (3) takes $O(|E| \cdot m)$ time, and step (2) can be implemented to take $O(|E| \cdot m)$ time using dynamic programming.

Other algorithms that we may use in this context are variants on the *cutset* method for reasoning with graphs [9,10]. Darwiche [33] used an algorithm that is similar to BREAK-CYCLES for the problem of creating a join tree. Our algorithm differs from Darwiche's in treating an already formed partition and creating the tree in a greedy way (Darwiche's method randomly selects a tree).

Finally, from Theorems 2.4 and 2.7 we observe that if partitioned theory $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$ is a tree decomposition as defined by Arnborg and others (e.g., [7]), then MP is sound and complete.

2.2. Backward message passing

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 [101]) 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 (where the distance between 2 nodes in graph G is the number of nodes comprising the shortest path between the two nodes), and try to prove $\{\}$ in $\mathcal{A}_j$ using MP. If we wish to follow the spirit of backward-reasoning more closely, we can transform G into a *chain* in a similar way to our transformation of G into a tree using BREAK-CYCLES. The resultant chain graph may then be used for query-driven backward message passing, from $\mathcal{A}_k$. We present such an algorithm, called BACKWARD-M-P (BMP), in Fig. 7. BMP takes as input a partitioned theory $\mathcal{A}$, a graph G_0, and a query, Q, and returns YES if it can prove Q.

Procedure CHAINIFY is outlined in Fig. 8. It accepts a labeled graph and returns a transformation of the graph into a chain (changing the labels appropriately). Alternately, we can create a chain directly from the partitions and a total order over them. CHAINIFY ensures that the resulting graph has a proper labeling. BMP is sound and complete if the reasoning procedure used in every partition is complete for consequence finding.

Theorem 2.8 (Soundness and completeness). *Let $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$ be a partitioned theory. Let $k \leqslant n$ and $\varphi \in \mathcal{L}(\mathcal{A}_k)$ a sentence. If the reasoning procedure used in each partition is sound and complete for consequence finding, then $\mathcal{A} \models \varphi$ iff applying BMP outputs YES.*

Proof. See Appendix A.3.

Algorithm BMP is presented for the case of *subgoal-disjunctive* systems, i.e., a proof of any subgoal yields a proof of the entire query. This is the case with resolution and its

PROCEDURE BACKWARD-M-P(BMP)($\{\mathcal{A}_i\}_{i\leqslant n}$, G_0, Q)
$\{\mathcal{A}_i\}_{i\leqslant n}$ a partitioned theory, $G_0 = (V, E, l)$ a graph, Q a query in $\mathcal{L}(\mathcal{A}_k)$ ($k \leqslant n$).

(1) $G \leftarrow$ CHAINIFY(G_0, k).
(2) For all $i \leqslant n$, $i \neq k$ set $goal_i \leftarrow$ *FALSE* (the goal of $\mathcal{A}_i$ is to prove FALSE). Set $goal_k \leftarrow Q$.
(3) Concurrently,
  (a) For each partition $\mathcal{A}_i$, $i \leqslant n$, attempt to prove $goal_i$.
  (b) For every $(i, j) \in E$ such that $i \prec j$, if we generate a subgoal[a] φ in $\mathcal{A}_j$ and $\varphi \in \mathcal{L}(l(i, j))$, then set[b] $goal_i \leftarrow goal_i \vee \varphi$.
  (c) If $goal_i$ is proved in any $\mathcal{A}_i$, return YES.

[a] In resolution every generated clause can be considered the negation of a subgoal.
[b] In resolution refutation the goal is negated, so this step essentially adds $\neg\varphi$ to $\mathcal{A}_i$.

Fig. 7. A backward message-passing algorithm.

PROCEDURE CHAINIFY(G, k)
$G = (V, E, l)$ a graph describing connections between partitions, $k \leqslant |V|$.

(1) Let $dist(i, j)$ ($i, j \in V$) be the length of the shortest path between i, j in G. Let $i \prec_0 j$ iff $dist(i, k) < dist(j, k)$ ($\prec_0$ is a strict partial order).
(2) Impose a total order $\prec$ on V that agree with $\prec_0$ (i.e., $i \prec_0 j \Rightarrow i \prec j$).
(3) Let $\{v_a\}_{a\leqslant n} = V$ such that $v_1 = k, \forall a \leqslant n\ v_a \prec v_{a+1}$.
(4) Let $E' = \{(v_a, v_{a+1})\}_{i<n}$.
(5) Set $l'(i, j) \leftarrow \emptyset$ for all $i, j \in V$.
(6) For all $(i, j) \in E$, for all $a < n$, if $i \preceq v_a \prec j$ (i.e., v_a is between i and j), then set $l'(v_a, v_{a+1}) \leftarrow l'(v_a, v_{a+1}) \cup l(i, j)$.
(7) Return $G' = (V, E', l')$.

Fig. 8. A procedure that transforms a graph G into a chain G'.

variants. The intuition behind the algorithm is that when a partition is supplied a subgoal sentence φ from another partition, φ is added to (*OR*-ed with) the partition's goal.

We make G a chain because otherwise subgoals may have to split between partitions. Splitting subgoals requires accounting for different preconditions (as in natural deduction), which we wish to avoid here, for simplicity of inference.

2.3. Queries drawn from multiple partitions

MP and its variants require that query Q be in the language of a single partition, $\mathcal{L}(\mathcal{A}_k)$, for some $k \leqslant n$. One way to answer a query Q that comprises symbols drawn from multiple partitions is to add a new partition $\mathcal{A}_Q$, with language $\mathcal{L}(\mathcal{A}_Q) = \mathcal{L}(Q)$, the language of 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 ensure a proper labeling

of G for the partitioned theory (as discussed in Section 2.1). To prove Q in $\mathcal{A}_Q$, we run MP on the resulting graph.

Alternately, we can decompose the query into the appropriate partitions, following the methods of [81] or [97]. Since the issue of decomposing a query is not simple, we describe only the simple case of a propositional query and leave the first-order case (with literals that contain symbols from multiple partitions) for future work.

Given a propositional query Q, we transform it into the form $(Q_1^1 \vee \cdots \vee Q_{r_1}^1) \wedge \cdots \wedge (Q_1^l \vee \cdots \vee Q_{r_l}^l)$, where each Q_j^i is a formula in the language of a single partition $\mathcal{L}(\mathcal{A}_{k_{ij}})$ (k_{ij} is the index of a partition that includes the vocabulary of Q_j^i). For example, if Q is in CNF, it is already in this form. We check a disjunct $Q_1^i \vee \cdots \vee Q_{r_i}^i$ by asserting $\neg Q_j^i$ in $\mathcal{A}_{k_{ij}}$ for all $j \leqslant r_i$, and proving FALSE in one of the partitions. To prove Q we check each of the disjunct in its transformed form. It is a valid consequence of $\mathcal{A}$ iff all the disjuncts are valid consequences of $\mathcal{A}$. We discuss this special topic no further here, and assume Q is drawn from $\mathcal{L}(\mathcal{A}_k)$, for some $k \leqslant n$.

3. Resolution and message-passing

The previous section presented message-passing algorithms with an arbitrary sound and complete consequence finder. In this section, we specialize our message-passing algorithms with consequence finders that specifically employ *resolution*. We focus on the first-order case of resolution. We also analyze the effect message passing has on the computational efficiency of resolution-based inference.

The presentation in this section makes explicit reference to the forward message-passing algorithm, MP, but we wish to stress that the results in this section are equally applicable to other message-passing algorithms introduced in the previous section. For background material on resolution, the reader is referred to [45,54] as well as to [22,72].

3.1. Resolution message-passing

Resolution [88] is one of the most widely used reasoning methods for automated deduction, and more specifically for consequence finding. As noted in Section 2, 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., [71]) 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 Fig. 9. 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 subse-

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

(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 in each of the partitions $\mathcal{A}_i$, $i \leqslant n$.
   (b) For every $(i, j) \in E$ such that $i \prec j$, if partition $\mathcal{A}_j$ includes the clause φ (as input or resolvent) and the predicates of φ are in $\mathcal{L}(l(i, j))$, then perform RES-SEND(φ, j, i).
   (c) If Q is proven in $\mathcal{A}_k$, return YES.

Fig. 9. A resolution forward message-passing algorithm.

quent 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.

In the propositional case, subroutine RES-SEND(φ, j, i) (Implementation 1) simply adds φ to $\mathcal{A}_i$, as done in MP. MP is then sound and complete.

In the FOL case, implementing RES-SEND requires more care. To illustrate, consider the case where resolution generates the clause $P(B, x)$ (B a constant symbol and x a variable). It also implicitly proves that $\exists b\ P(b, x)$. RES-MP 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 [98])). 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 consequence finding for the corresponding full FOL language. This problem is also reflected in a slightly different statement of Craig's interpolation theorem [30] that applies for resolution [98].

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

The process of conservatively replacing function and constant symbols by existentially quantified variables is called *unskolemization* or *reverse Skolemization* and is discussed

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

(1) Unskolemize φ into a set of formulae, Φ in $\mathcal{L}(l(i, j))$, treating every symbol of $L(\varphi) \setminus l(i, j)$ as a Skolem symbol.
(2) Let Φ_2 be the Skolemized version of Φ.
(3) For every term t that was unskolemized in step 1, if t appeared in a previously sent message, ψ, and s was t's Skolemization there, then replace the Skolemization of t in Φ_2 with s.
(4) For every $\varphi_2 \in \Phi_2$, if φ_2 is not subsumed by a clause that is in $\mathcal{A}_i$, then add φ_2 to the set of axioms of $\mathcal{A}_i$.

Fig. 10. Subroutine RES-SEND using unskolemization.

in [13,29], as well as [21]. Chadha and Plaisted in [21] present an algorithm U that is complete for our purposes and generalizes and simplifies an algorithm of [29].

Theorem 3.1 [21]. *Let V be a vocabulary and φ, ψ be formulae such that $\psi \in \mathcal{L}(V)$ and $\varphi \models \psi$. There exists $F \in \mathcal{L}(V)$ that is generated by algorithm U such that $F \models \psi$.*

Thus, for every reasoning procedure that is complete for clausal 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 FOL consequence finding. This procedure can then be used readily in RES-MP (or in MP), upholding the soundness and completeness to that supplied by Lemma 2.6. The subroutine RES-SEND(φ, j, i) (Implementation 3) that implements this approach is presented in Fig. 10. It replaces φ with a set of formulae in $\mathcal{L}(l(i, j))$ that follows from φ. It then Skolemizes the resulting formulae for inclusion in $\mathcal{A}_i$. The procedure makes sure that terms and functions that appear in more than one message are replaced by the same Skolem constants and functions in all those instances (we discuss the reason for this at the end of this subsection).

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 Fig. 11.

Steps (3) and (4) of procedure RES-SEND(φ, j, i) (Implementation 4) correspond to similar steps in procedure U presented in [21], simplifying where appropriate for our setup. Our procedure differs from unskolemizing procedures in step (5), 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 (4) is used to add *all* the possible combinations of unified terms, which is required to ensure completeness.

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

(1) T is a *static* table (i.e., keeps its value between invocations of RES-SEND) that is initialized to $\emptyset$ when RES-MP is called.
(2) Set $S \leftarrow L(\varphi) \setminus l(i, j)$ (S is the set of symbols of φ that we cannot send).
(3) For every term instance, $t = f(t_1, \ldots, t_k)$, in φ, if $f \in S$ and t is not a subexpression of another term $t' = f'(t'_1, \ldots, t'_{k'})$ of φ with $f' \in S$, then replace t with "$x \leftarrow t$" for some new variable, x (if $k = 0$, t is a constant symbol).
(4) Nondeterministically[a], for every pair of marked arguments "$x \leftarrow \alpha$", "$y \leftarrow \beta$" in φ, if α, β are unifiable, then unify all occurrences of x, y (i.e., unify α_i, β_i for all markings $x \leftarrow \alpha_i$, $y \leftarrow \beta_i$).
(5) For every marked argument "$x \leftarrow \alpha$" in φ,
    (a) Collect all marked arguments with the same variable on the left-hand side of the "$\leftarrow$" sign. Suppose these are $x \leftarrow \alpha_1, \ldots, x \leftarrow \alpha_l$.
    (b) Let $y_1, \ldots, y_r$ be all the variables occurring in $\alpha_1, \ldots, \alpha_l$. For every $i \leqslant l$, replace "$x \leftarrow \alpha_i$" with $f(y_1, \ldots, y_r)$ in φ, for a function symbol f (if $r = 0$, f is a constant symbol) such that
    - If α_i appears in table T, then f is the symbols that is in the α_i entry in T. Else, f is a fresh symbol; add the entry $\langle \alpha_i, f \rangle$ to T.
(6) Add φ to $\mathcal{A}_i$, if it is not subsumed by a clause in $\mathcal{A}_i$.

[a] Nondeterministically select the set of pairs for which to unify all occurrences of x, y. From here forth we continue with one such set of unifications. The end result is the set of clauses that includes all these possibilities.

Fig. 11. Subroutine RES-SEND without unskolemization.

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

Proof. See Appendix A.4.

In Implementations 3 and 4 of RES-SEND we carefully chose and repeated Skolem constants in the sent clauses. The reason for this is the following. Craig's interpolation theorem guarantees that we need to send only a single sentence that includes only symbols that are in the link language. However, this guaranteed sentence may include several clauses,

and the Skolem constants that we need to apply to those must be the same. For example, consider the two partitions $\mathcal{A}_1, \mathcal{A}_2$ with $\mathcal{A}_1 = \{P(s), Q(s)\}$ for a constant symbol s, and $\mathcal{A}_2 = \{\neg Q(x) \vee \neg P(x)\}$, for a variable x, and $l(1,2) = \{P, Q\}$. In two separate messages from $\mathcal{A}_1$ to $\mathcal{A}_2$ we need to send $P(s)$ and $Q(s)$. If we unskolemize and Skolemize $P(s)$ into $P(Skolem1)$ and $Q(s)$ into $Q(Skolem2)$, then we will not be able to reach the empty clause in $\mathcal{A}_2$, a conclusion that does follow if we send $P(Skolem1)$, $Q(Skolem1)$.

We suspect that the procedure that we outlined in Implementation 4 of RES-SEND for deciding which symbols should be repeated can be significantly improved. Consider the example from the previous paragraph, with the change that $\mathcal{A}_1 = \{P(s), Q(s), P(r), Q(r)\}$. There are four messages that should be sent from $\mathcal{A}_1$ to $\mathcal{A}_2$, but in fact we have grounds to restrict ourselves and send only two messages, namely, the messages that contain s (or the unskolemization and Skolemization of s). Since the only messages that mention r have isomorphic messages that mention s, then the messages containing r can be dropped. This observation promises to cut the number of constant symbols in a partition significantly, but its efficient and optimal application in the general case raises a set of graph-theoretical problems that are outside the scope of this article and we leave it for future work.

3.2. Analysis and comparison of resolution-based inference

In this final subsection relating to resolution, we analyze the effect of 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., [59,105,107]) (and see the survey article [26]) is only marginally relevant. More relevant is comparing the sizes of search spaces induced by different strategies (e.g., resolution of propositional Horn clauses [84], and contraction rules for FOL [15]). These measures do not precisely address our needs, but we use them here, leaving the development of better measures for comparison to future work.

In a *resolution search space*, each node in the search space includes a set of clauses, and properties relevant to the utilized resolution strategy (e.g., clause parenthood information). Each arc in the search space 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 3.3. *Let $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$ be a partitioned theory. Any path in the MP resolution search space of $\{\mathcal{A}_i\}_{i \leqslant n}$ is also a path in the resolution search space of the unpartitioned theory $\mathcal{A}$.*

Evaluating MP with respect to 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 2.3 for $\alpha = A_1$, $\beta = \mathcal{A}_2 \Rightarrow Q$. The MP proof has to prove $\alpha \vdash \gamma$ and $\gamma \vdash \beta$.

For the propositional case there are several results relating shortest proofs and proofs using the interpolant. Carbone [20] 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 (with cuts (the case that is the most similar to resolution) the bound is k)), where k is the length of the minimal proof of $\alpha \vdash \beta$. In some of these cases, the minimal interpolant is shown to be of size $O(a^2)$, where a is the sum of lengths of α, β.

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 [17]. Nevertheless, there is a good upper bound on the length of the interpolation formula as a function of the length of the minimal proof [65]: 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 $\min(kl^{O(1)}, 2^l)$.

[86] presented an analysis of *ordered resolution* in the propositional case. They used the concepts of *induced width* and *treewidth* to analyze the performance of this algorithm. We bring their analysis here as its results generalize to ours as well.

Definition 3.4 [87]. A *tree-decomposition* of a graph $G(V, E)$ is a pair $D = (S, T)$ with $S = \{X_i \mid i \in I\}$ a collection of subsets of vertices of G and $T = (I, F)$ a tree, with one node for each subset of S, such that the following three conditions are satisfied: (1) $\bigcup_{i \in I} X_i = V$. (2) For all edges $(v, w) \in E$ there is a subset $X_i \in S$ such that both v, w are contained in X_i. (3) For each vertex x, the set of nodes $\{i \mid x \in X_i\}$ forms a subtree of T.

The *width* of a *tree-decomposition* $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I}(|X_i| - 1)$. The *treewidth* of a graph G equals the minimum width over all tree-decompositions of G. Every ordering on symbols induces a tree decomposition (we do not present details here; they can be found in [63] and others). The width of that tree decomposition is sometimes called the *induced width* of that ordering.

In our context, the *width* of the decomposition $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$ is the largest number (minus 1) of nonlogical symbols appearing in a single partition (including the symbols on its links to other partitions). The *treewidth* is the smallest width achievable for a theory (i.e., the best decomposition that is possible while still maintaining the proper-labeling property). The results of [86] show that ordered resolution cannot resolve more than $2^k n$ clause pairs when k is the width of the decomposition. The same result can be shown for any instantiation of MP for propositional theories.

Theorem 3.5. *Let $\mathcal{A}$ be a partitioned propositional theory with n, and let $G = (V, E, l)$, $\bigcup_{i \leqslant n} \mathcal{A}_i$ be a tree decomposition of width k of $\mathcal{A}$. Then, the time taken by RES-MP* (*with any sound resolution strategy for in-partition computation*) *to compute SAT for $\mathcal{A}$ is* $O(2^k n)$. *The space needed for this computation is* $O(2^k)$.

For the first-order case there are very few results. The best known is by Meyer [79] who showed that for the first-order predicate calculus with equality there is no recursive bound on the length of the smallest interpolant as a function of the length of the input axioms. However, there is no result relating the size of the interpolant with the length of the minimal proof (in resolution or any other proof system).

The results above suggest that 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. Take, for example, 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 5 we provide an algorithm for partitioning theories that attempts to minimize the communication language between partitions.

Finally, we bring a pair of results that appear in [6] for completion of our current exposition. These results relate reasoning with partitioned theories and different ordering strategies of resolution (look at [19,22,38] for more information on these strategies).

Theorem 3.6 (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 $\leqslant_A$ a total order on its n propositional symbols. Then, there is a partitioning $\{\mathcal{A}_i\}_{i\leqslant n}$ of $\mathcal{A}$, a graph G and partition reasoners that are based on ordered resolution such that running MP does not perform more resolutions than directional resolution* (*alternatively, A-ordering*) *of $\mathcal{A}$ with order $\leqslant_A$.*
(2) *Let $\mathcal{A}$ be a FOL theory and $\leqslant_A$ a total order on its n predicate symbols. Then, there is a partitioning of $\mathcal{A}$ into $\{\mathcal{A}_i\}_{i\leqslant n}$, a graph G and partition reasoners that are based on ordered resolution such that running MP does not perform more resolutions than A-ordered resolution of $\mathcal{A}$ with order $\leqslant_A$.*
(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 $\mathcal{A} = \bigcup_{i\leqslant n} \mathcal{A}_i$ 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.*

Theorem 3.7 (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 $\mathcal{A} = \bigcup_{i\leqslant n} \mathcal{A}_i$ 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, $\leqslant_A$, on $\mathcal{A}$'s propositional symbols such that if a clause C is a consequence of directional resolution of $\mathcal{A}$ with order $\leqslant_A$, then C is a consequence of running MP on this partitioning using unrestricted resolution in each partition.
(2) *Let $\mathcal{A} = \bigcup_{i\leqslant n} \mathcal{A}_i$ 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, $\leqslant_A$, on $\mathcal{A}$'s predicate symbols such that if a clause C is a consequence of A-ordered resolution of $\mathcal{A}$ with order $\leqslant_A$, then C is a consequence of running MP on this partitioning using unrestricted resolution in each partition.

(3) *Let $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$ 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.

4. Partition-based propositional satisfiability

In this section we present an algorithm for partition-based logical reasoning that takes advantage of propositional satisfiability (SAT) search subroutines (e.g., DPLL [34], GSAT [92] and WALKSAT [91]).

This algorithm is very similar to the algorithm of [36,37] for constraint satisfaction problems and we contrast it here with our MP algorithms. We also bring a correctness proof that follows from our soundness and completeness proof for MP. The algorithm also allows us to examine the complexity of computation and show that here too the complexity is directly related to the size of the labels in the intersection graph, i.e., the width and link size of the graph.

4.1. A partition-based SAT procedure

The algorithm we propose is presented in Fig. 12. It uses a SAT procedure as a subroutine and is backtrack-free. We describe the algorithm using database notation [106]. $\pi_{p_1,\ldots,p_k} T$ is the *projection* operation on a relation T. It produces a relation that includes all the rows of T, but only the columns named $p_1, \ldots, p_k$ (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

PROCEDURE LINEAR-PART-SAT($\{\mathcal{A}_i\}_{i \leqslant n}$)
$\{\mathcal{A}_i\}_{i \leqslant n}$ a partitioning of the theory $\mathcal{A}$.

(1) $G_0 \leftarrow$ the intersection graph of $\{\mathcal{A}_i\}_{i \leqslant n}$. $G \leftarrow$ *BREAK-CYCLES*(G_0).
(2) For each $i \leqslant n$, let $L(i) = \bigcup_{(i,j) \in E} l(i, j)$.
(3) For each $i \leqslant n$, 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) Determine $\prec$ as in Definition 2.1.
(5) Iterate over $i \leqslant n$ in reverse $\prec$-order (the last i is 1). For each $j \leqslant n$ 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) If FALSE has not be returned, return TRUE.

Fig. 12. An algorithm for SAT of a partitioned propositional theory.

(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., [61,74]). Step (1) of the algorithm converts the intersection graph of $\mathcal{A}$ into a tree. Step (2) computes $L(i)$, the set of symbols on all of partition $\mathcal{A}_i$'s links, i.e., the union of all the communication languages connected to partition $\mathcal{A}_i$. Step (3) determines which truth values of $L(i)$ are satisfiable (akin to computing the implicates of each partition in the language $\mathcal{L}(L(i))$. Finally, the algorithm uses $\bowtie$ to combine those values to find out if there are any models for $\mathcal{A}$.

This algorithm resembles finding all the models of each partition and then joining the consistent interpretation fragments into models for $\mathcal{A}$ (as done in [37]). The iterated join that we perform takes time proportional to the size of the tables involved. Furthermore, we keep table sizes below $2^{|L(i)|}$ by keeping only the consistent truth assignments for $L(i)$ and *projecting* every table before *joining* it with another table. This is similar to an approach that was presented in [36,94] that trades space for time in CSPs and Bayes Network. The computation is done via search in each partition, yielding a method that takes time exponential in the partition size and space exponential in the separator (label) size.

Fig. 13(a) displays the result of applying LINEAR-PART-SAT up to step (3) to the partitioned theory and input of Fig. 4. Fig. 13(b) and 13(c) show the progression of step (5) of LINEAR-PART-SAT.

Soundness and completeness follow by an argument similar to that given for MP.

Theorem 4.1 (Soundness and completeness). *Given a sound and complete SAT-search procedure, LINEAR-PART-SAT is sound and complete for SAT of partitioned propositional theories.*

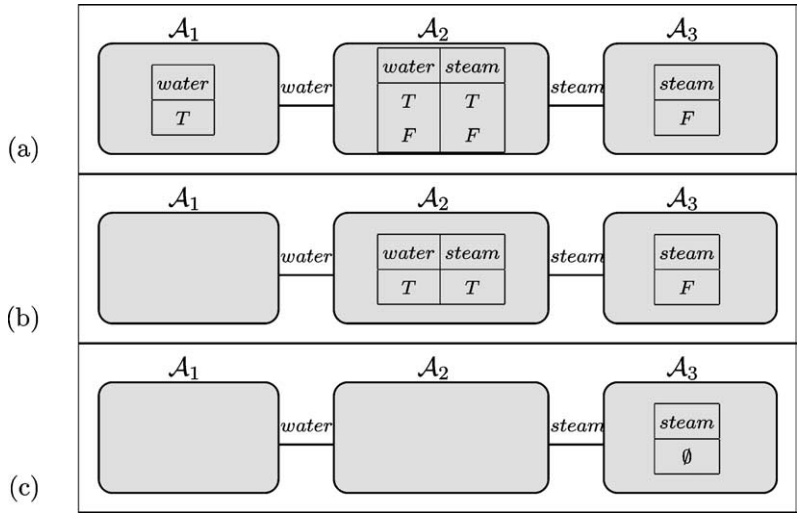

Fig. 13. Iteratively projecting and joining tables to check satisfiability.

Proof. See Appendix A.5.

4.2. Analyzing satisfiability in LINEAR-PART-SAT

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

Lemma 4.2. *The time taken by LINEAR-PART-SAT to compute SAT for $\mathcal{A}$ is*

$$\begin{aligned} &\mathit{Time}\big(n, m, m_1, \ldots, m_n, a, k, \big|L(1)\big|, \ldots, \big|L(n)\big|\big) \\ &\quad = \mathrm{O}\left(a \cdot k^2 \;+\; n^4 \cdot m + \sum_{i=1}^{n} \big(2^{|L(i)|} \cdot f_{SAT}(m_i)\big)\right), \end{aligned}$$

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 $n^4 \cdot m$ to $n \cdot m$.

Proof. See Appendix A.6.

Corollary 4.3. *Let $\mathcal{A}$ be a partitioned propositional theory with n partitions, m propositional symbols and intersection graph $G = (V, E, l)$. Let $d(v)$ be the degree of node v in the graph $G(V, E, l)$, let $d = \max_{v \in V} d(v)$ and let $l = \max_{i,j \leqslant n} |l(i, j)|$. 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*

$$\mathit{Time}(m, n, l, d) = \mathrm{O}\big(n \cdot 2^{d \cdot l} \cdot f_{SAT}(m/n)\big).$$

The space taken for this computation is $\mathrm{O}(2^{d \cdot l})$.

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 $\mathrm{O}(2^l \cdot f_{SAT}(m/2))$. Assuming P $\neq$ NP, this is a significant improvement over a simple SAT procedure for every l that is small enough ($l < \alpha m/2$, and $\alpha \leqslant 0.582$ [26,89]).

Corollary 4.4. *Let $\mathcal{A}$ be a partitioned propositional theory with n partitions, m propositional symbols and intersection graph $G = (V, E, l)$ of width k. Then, the time taken by the LINEAR-PART-SAT procedure to compute SAT for $\mathcal{A}$ is* $\mathrm{O}(2^k n)$. *Taking $ld = \max_{i \leqslant n} |\bigcup_{j \leqslant n} l(i, j)|$, the space needed for this computation is* $\mathrm{O}(2^{ld})$.

5. Decomposing a logical theory

The algorithms presented in previous sections assumed a given partitioning of theory $\mathcal{A}$. 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.

5.1. What is a good partitioning?

The analysis done in Section 4.2 does not assume any particular time complexity for $f_{SAT}(m)$ (aside from P $\neq$ NP in the corollary). If we assume that $f_{SAT}(m) = \Theta(2^m)$, then we can conclude that the time for our reasoning algorithm is dominated by the largest partition (including its links). If the largest partition is of size s (i.e., it has s propositional symbols in its language, link languages included), then the time for the algorithm is $O(n \cdot 2^s)$.

This analysis is exactly the one that is done for CSPs and Bayes networks (e.g., [37,38, 86], and [11]), where the utilized algorithms do in fact use time $\Theta(2^m)$ for a problem with m variables. For satisfiability the situation is slightly different. There are known stochastic algorithms (e.g., [91,92]) that perform much better than this pessimistic forecast. These algorithms are not complete, but they can be used in our algorithm, if we are willing to give up completeness. Efficient complete algorithms also typically exhibit better than worst-case behavior (see the analysis of [93]).

In the general FOL case we have no clear-cut measure of computational complexity, but the results for propositional logic suggest similar relationship between partitioning and computational behavior. All of this suggests that emphasizing link sizes together with partition sizes is more accurate for the satisfiability problem.

Thus, given a theory, we wish to find a partitioning that minimizes the formula derived in Lemma 4.2. To that end, assuming P $\neq$ NP, we want to minimize the following parameters in roughly the following order. For all $i \leqslant n$:

(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 $\mathcal{A}_i$ but not in $L(i)$. 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 4.3, the maximal n that maintains l, d such that also $n \leqslant \ln 2 \cdot \alpha \cdot m$ ($\alpha = 0.582$ [26,89]) yields an optimal bound for LINEAR-PART-SAT. In Section 3.2 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 per-

form. 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 [3,24,69]). More generally, we believe axiomatizers of large corpora of real-world knowledge tend to try to provide structured representations following some of these principles. Recent experiments with the HPKB knowledge base of SRI and a part of the Cyc knowledge base support this belief (those experiments are reported elsewhere).

5.2. An approach to partitioning logical theories

To exploit the partitioning guidelines proposed in Section 5.1, 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 $L(\mathcal{A})$, 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 $\alpha\}$.

Fig. 14 (top) illustrates the symbols graph of theory $\mathcal{A}$ from Fig. 1 and the connected symbols graphs (bottom) of the individual partitions $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$. 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.

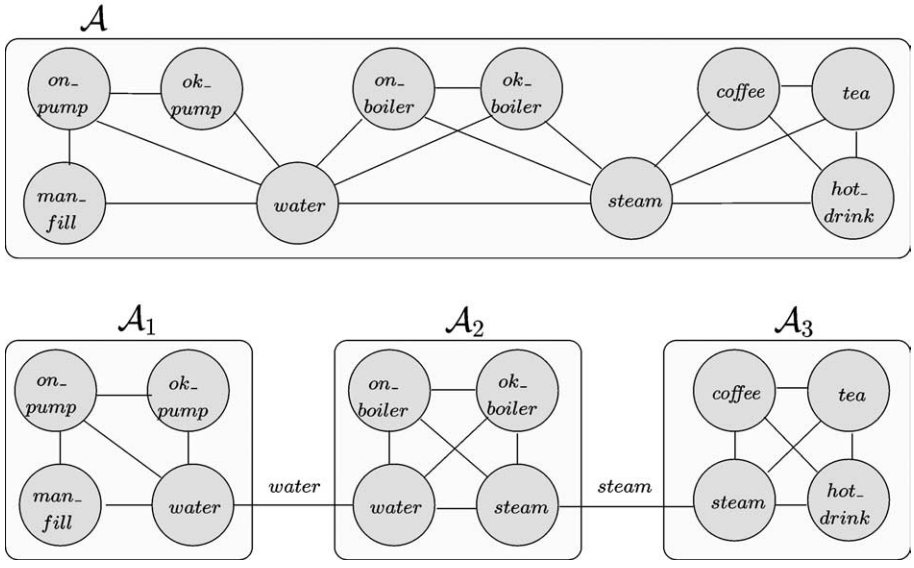

Fig. 14. Decomposing $\mathcal{A}$'s symbols graph.

Generally speaking, we decompose theory $\mathcal{A}$ by first creating the symbols graph of $\mathcal{A}$, then partition this graph into partitions (similar to the bottom part of Fig. 1), and finally use the partitioning of the graph to define a partitioning of the axioms. The first part of Fig. 16 (Procedure Split-Thy) is a generic algorithm that does just that. G_{str} is manipulated by the subroutine of this procedure such that at the end G_{str} is a tree decomposition (see Definition 3.4) of G. Section 5.3 describes one complete instantiation of this algorithm for this task. Examples of the recursive procedure that we are going to use are presented in Fig. 17.

The relationship between computing minimum vertex separators and computing tree decompositions is well known and has been studied in the literature for some time (e.g., [7]). Here we present a particular greedy algorithm for partitioning propositional and first-order logical theories that is also based on the computation of minimum vertex separators. In more recent work, we use this algorithm as one of two algorithms for evaluating the performance of our partition-based logical reasoning work [73].

5.3. *Split: greedy vertex min-cut in the graph of symbols*

5.3.1. *Minimum vertex separators*

In this section, 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.

Let $N(a, b)$ be the least cardinality of an (a, b)-vertex-separator. The *connectivity* of the graph G is the minimum $N(a, b)$ for any $a, b \in V$ that are not connected by an edge. An (a, b)-vertex-separator of minimum cardinality is said to be a *minimum* (a, b)-*vertex-separator*. The weaker property of a vertex separator being *minimal* requires that no subset of the (a, b)-vertex-separator is an (a, b)-vertex-separator.

We briefly review an algorithm by Even and Tarjan for finding minimum vertex separators [48,49]. This algorithm builds on [32]. It is shown in Fig. 15. The algorithm is given two vertices, a, b, and an undirected graph, G. It transforms G into a directed graph, $\overline{G}$, that has two vertices (corresponding to *input* and *output*) for each original vertex of G, directed edges connecting the corresponding input and output vertices, and edges corresponding to those of G, but only from output to input vertices. It then runs a max-flow algorithm on $\overline{G}$ (steps (1)–(3)). The produced flow, f, has a throughput of $N(a, b)$. To extract a minimum separator, it produces a *layered network* (see [48, p. 97]) from $\overline{G}$ and the flow found, f, in step (5). The layered network includes a subset of the vertices of $\overline{G}$. The set of edges between this set of vertices and the rest of $\overline{G}$ corresponds to the separator.

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 [53], the push-relabel method of Goldberg and Tarjan [57] (time bound of $O(|V| \cdot |E| \cdot \log(|V|^2/|E|))$ and several implementations [23]), and Dinitz's algorithm [44]. When Dinitz's algorithm is used to solve the network problem, Even and Tarjan's algorithm has time complexity $O(|V|^{1/2}|E|)$ [48]. The unit-capacity network-flow algorithm of [1] can also be used here, giving Even and Tarjan's algorithm time complexity of $O(|V|^{1/2}|E|)$ as well.

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

(1) Construct a digraph $\overline{G}(\overline{V}, \overline{E})$ as follows. For every $v \in V$ put two vertices v', v'' (*input/output vertices*) in $\overline{V}$ with an edge $e_v = (v', v'')$ (*internal edge*). For every edge $e = (u, v)$ in G, put two edges $e' = (u'', v')$ and $e'' = (v'', u')$ in $\overline{G}$ (*external edges*).
(2) Define a network, with digraph $\overline{G}$, source a'', sink b' and unit capacities for all the edges.
(3) Compute the *maximum flow* f in the network.
(4) Set the capacities of all the external edges in $\overline{G}$ to infinity.
(5) Construct the *layered network* $\{V_i\}_{i \leqslant l}$ from $\overline{G}$ using f. Let $S = \bigcup_{i \leqslant l} V_i$.
(6) Let $R = \{v \in V \mid v' \in S,\ v'' \notin S\}$. R is a minimum (a, b) vertex-separator in G.

Fig. 15. An algorithm for finding a minimum separator between a and b in G.

Another possibility is to use the Ford–Fulkerson flow algorithm as described in [53] (alternatively, see [27]), for computing maximum flow. For an original graph of tree-width[2] $< k$ this involves finding at most k augmenting paths of capacity 1. Thus, the combined algorithm using the Ford–Fulkerson maximum flow algorithm finds a minimum (a, b)-vertex-separator in time $O(k(|V| + |E|))$.

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 $O(c \cdot |V|^{3/2} \cdot |E|)$, where $c \geqslant 1$ is the connectivity of G. When we use Ford–Fulkerson's algorithm for a graph of tree-width k, this procedure takes time $O(c \cdot k \cdot |V| \cdot (|V| + |E|))$, where $c \geqslant 1$ is the connectivity of G. For the cases of $c = 0, 1$ there are well-known linear time algorithms. [47] also showed a way to test for k connectivity of a graph using only $n + k^2$ pairs of vertices.

5.3.2. *Procedure* Split

Procedure Split-Thy, presented in Fig. 16, uses procedure Split to decompose a theory into a tree of partitions. It is given a theory, $\mathcal{A}$, and limitations on the partition size (a lower limit, M) and the separators between partitions (an upper limit, l). Split initially considers the theory as one big partition, and at every recursive iteration it breaks one of the partitions in two. It represents the tree structure of the partitions in a global variable, G_{str}. This tree structure and the set of partitions, $\{\mathcal{A}_i\}_{i \leqslant p}$, is returned as the result of Split-Thy. An example of the input and the output is shown in Fig. 1.

Split partitions the theory $\mathcal{A}$ by taking as input its symbols graph, $G = (V, E)$, the two limiting parameters, M and l, and nodes $a, b \in V$ that are initially set to nil. Split updates the global variable G_{str} to represent the progressing decomposition. In each recursive call, Split finds a minimum vertex separator of a, b in G (i.e., a minimum-size set of vertices

[2] The tree-width of a graph plus one is the minimum, over all triangulations of this graph, of the size of the largest clique in the triangulation (see [63]).

PROCEDURE Split-Thy($\mathcal{A}$, M, l)
$\mathcal{A}$ is a theory. M limits the number of symbols in a partition from below. l limits the number of symbols shared between partitions.

(1) Let $G(V, E)$ be an undirected graph with $V = L(\mathcal{A})$ and $E = \{(l_1, l_2) \mid \exists C \in \mathcal{A}\ l_1, l_2 \in L(C)\}$.
(2) Let $G_{str}(V_{str}, E_{str})$ be an undirected graph with $V_{str} = \{\{V\}\}$ and $E_{str} = \emptyset$.
(3) Run Split(G, M, l, nil, nil).
(4) For every $v \in V_{str}$, let $\mathcal{A}_v = \{C \in \mathcal{A} \mid L(C) \subset v\}$. Return $\{\mathcal{A}_v\}_{v \in V_{str}}$ and G_{str}.

PROCEDURE Split(G, M, l, a, b)
$G = (V, E)$ is an undirected graph. M, l as above. a, b are in V or are nil.

(1) If $|V| < M$, then return V.
(2) (a) If $a, b = nil$, find a minimum vertex separator, R, in G. (b) Otherwise, if $b = nil$, find a minimum vertex separator, R, of a in G. (c) Otherwise, find minimum vertex separators, R_a of a in G, and R_b of b in G. Let R be the smaller of R_a, R_b.
(3) If $R = V$ or $|R| > l$ then return V.
(4) Let G_1, G_2 be the two subgraphs of G separated by R, with R included in both subgraphs.
(5) Let $V_{str} \leftarrow V_{str} \setminus \{\{V\}\} \cup \{\{V_1\}, \{V_2\}\}$ and $E_{str} \leftarrow E_{str} \cup \{(\{V_1\}, \{V_2\})\}$. Change the edges that connected to $\{V\}$ to connect to one of $\{V_1\}, \{V_2\}$.
(6) 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 connected by edges to some vertices in R.
(7) Run $Split2(G'_1, M, l, r, a)$, $Split2(G'_2, M, l, r, b)$. Replace r in the nodes of V_{str} by the members of R.

Fig. 16. An algorithm for generating partitions of axioms.

that crosses every path between a, b). If one of a, b or both are nil, it finds the overall minimum vertex separator between all vertices and the non-nil vertex (or all other vertices). This separator splits G into two graphs, G_1, G_2, and the process continues recursively. An example of the progress made on the input graph G is shown in Fig. 17.

Different variants of the algorithm yield different structures 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 3(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 6.

Proposition 5.1. *Procedure Split takes time* $O(|V|^{5/2} \cdot |E|)$.

Proof. See Appendix A.7.

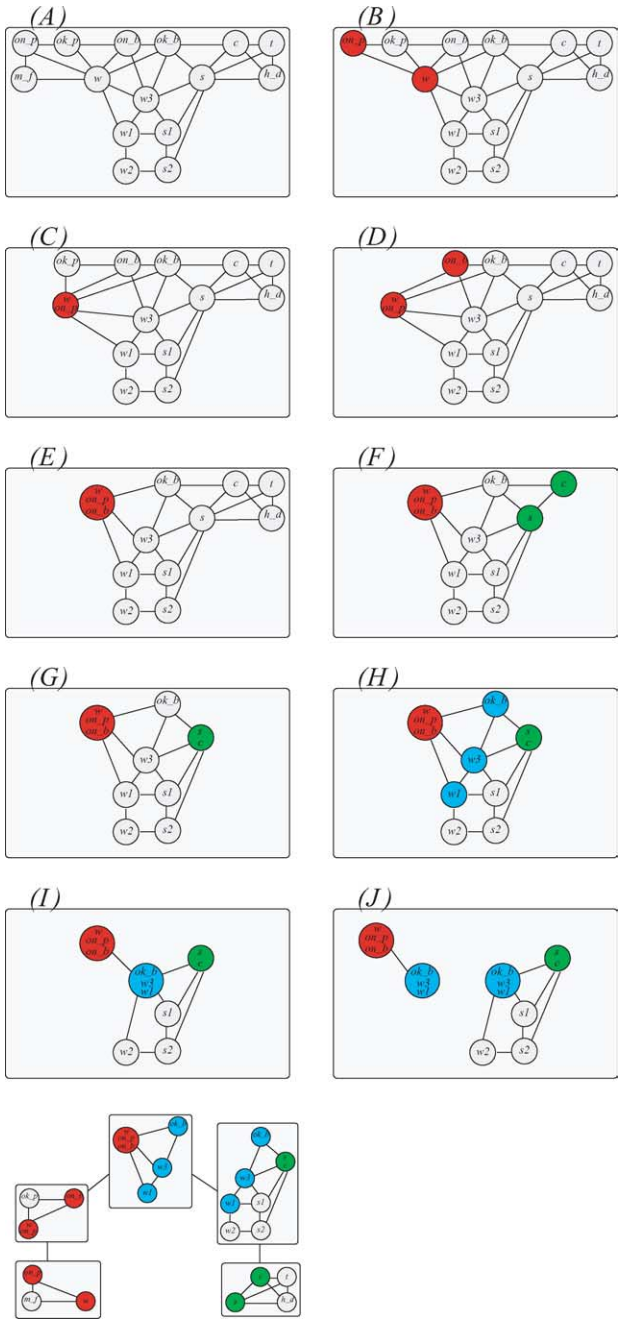

Fig. 17. Recursive use of *Split* by aggregating minimal separators into single nodes. Only the larger side of the leftover graph is shown after each split.

5.3.3. Fine-tuning Split

Since $f_{SAT}(m)$ is not known, and the time for reasoning with FOL theories in MP is not bounded, it is not clear what is an *optimal* decomposition. Nevertheless, the analyses done throughout this paper suggests minimizing the parameters mentioned in the last section. If we assume that $f_{SAT}(m) = \Theta(2^{\alpha m})$, then the problem of finding an optimal partition for LINEAR-PART-SAT is equivalent to finding triangulations of minimum clique number (a.k.a. finding *treewidth*). With this assumption, M should be chosen to be 1, l should be chosen to be m, and the algorithm will stop the recursive decomposition only when reaching a graph that is a clique. This is justified by the observation that any further decomposition can only decrease the size of the maximum partition (including the links). Thus, Assuming $f_{SAT}(m) = \Theta(2^{\alpha m})$, further decompositions only decrease the asymptotic time function of LINEAR-PART-SAT.

For reasoning with FOL theories it may sometimes be useful to choose M (the limit on the number of symbols in a partition) to be large, so that sentences are aggregated more closely to *topics*. This can be useful for managing large axiom sets as well as for applying specialized reasoning algorithms for each partition. This can be combined with replacing our vertex separator algorithm with a *balanced separator* algorithm. A balanced separator is a vertex separator that separates the graph such that the separated subgraphs are of comparable sizes (typically, they are chosen to be no larger than a constant times the size of the original graph). The problem of finding balanced separators is NP-hard, but several approximations exist (e.g., [50,62,68]).

Our time bound for Split is lower than $\Theta(2^{\alpha m})$ when $l \leqslant (\alpha m - \alpha m_i - \lg n)/d$ ($i = \text{argmax}_j m_j$). In particular, if $l > m/2$, a standard deterministic SAT procedure will be better (compared to the best time bound known for SAT procedures [26,89]).

All the observations above are predicated on the assumption the $\mathcal{A}$ is propositional and that $f_{SAT}(n) = \mathrm{O}(2^{\alpha n})$, for some $\alpha > 0$ constant. If our theory is in FOL, or we drop the assumption on f_{SAT}, then there is no clear good way to choose M, l. In those circumstances, l and M are perhaps best determined experimentally.

5.3.4. Other decomposition approaches

There are many possible approaches to decomposing a set of logical axioms. One complementary approach that we have briefly experimented with is a normalized cut algorithm [95], 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 possible 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 maximizing the similarity within partitions and minimizing the similarity between partitions.

A different decomposition is conceived from a semantic approach. Our reasoning algorithms and our computational analysis suggested a syntactic approach to decomposition. Semantic approaches are also possible along lines similar to [100] or to [22] (chapter on semantic resolution). Such decomposition approaches may require different reasoning algorithms to be computationally useful.

6. Related work

The work related to ours is vast. We divide it into three parts. First is the work on automated decomposition; second is the use of decompositions for propositional reasoning; and third is the work that relates to FOL theorem proving.

6.1. Automated decomposition

Decomposition techniques for CSPs, Bayes nets and other NP-hard problems are most relevant to our work on automated decomposition. These typically look for a *separation vertex* [35], use various heuristics to order symbols (an ordering that translates to a decomposition of the graph) [37,38,90], and use approximations for *tree decomposition of minimum width* (equivalent to finding *triangulations* of minimum clique number, computing *treewidth*, and finding optimal *clique trees*) [4,11,63,87,96].

The last approach is applicable to our setup, if we assume that $f_{SAT}(m) = \Theta(2^{\alpha m})$. In contrast to our SPLIT, these algorithms find weak approximations (factor $O(\log OPT)$) to the optimal in polynomial time and constant-factor approximations or optimal results in quasi-polynomial time (polynomial time, assuming the treewidth is bounded by a constant, and exponential time otherwise). Furthermore, work on implementing SAT and automated deduction strongly suggests that the assumption of $f_{SAT}(m) = \Theta(2^{\alpha m})$ is overly pessimistic. For this reason we prefer to minimize the links first, and then look at minimizing the partitions, leading to our proposed algorithm.

Cut-set conditioning (e.g., [9,10,35,83]) and hypertree decompositions of CSPs (such as the work of [58]) are other methods for using decompositions, that are fairly different from the one we use in this paper.

6.2. Use of decompositions in propositional SAT

With respect to propositional SAT problems, perhaps the most relevant previous work is that of Dechter and Pearl [37], which presented algorithms for reasoning with decomposed CSPs. These can be used for SAT under a given decomposition. In comparison, the algorithm we presented for partition-based SAT does not produce all the models possible in each partition, as proposed in [37]. Instead, it finds the truth values for propositions on the links that are extendible to a satisfying truth assignment for the whole partition. This reduces our computation time and makes it more dependent on the links' sizes rather than on partition sizes.

Other SAT uses of SAT decomposition include [82] which proposed 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). It finds the set of possible truth-value assignments to the propositions associated with the hyper-edges and tests them recursively for the two partitions. Cowen and Wyatt [28] developed an algorithm that partitions a propositional CNF theory into connected components that can be tested for satisfiability individually. Their partitioning algorithm is an adaptation of a best first search as used to find components in a graph, or strongly connected components in a digraph. The authors tested their decomposition algorithm together with

a SAT solver, demonstrating a dramatic decrease in the runtime of the SAT solver on the decomposed theories described in the paper.

Prior to [5] there has been no work on using decompositions for automated deduction in propositional logic in the manner we propose. Concurrently to this work, Rish and Dechter [86] proposed an algorithm similar to our MP algorithm for the case of propositional ordered resolution. However, their work looks at a limited case (ordered resolution, propositional logic), and they allow excessive computation by performing all possible resolutions in each partition, twice. Our MP algorithm can be used with different reasoning procedures for every partition, maintaining completeness as long as those algorithms satisfy some natural conditions. It is opportunistic in the sense that it does not wait for each partition to perform all of the possible resolutions. (In FOL waiting is not even a possibility.) Thus, Rish and Dechter's algorithm may use exponential amounts of space and time over and above MP in the same settings. Also, MP can be considered a generalization of theirs in the propositional case.

6.3. Use of decompositions in FOL theorem provers

Surprisingly, there has been little work on the specific problem of exploiting structure in theorem proving in the manner we propose in this paper. We conjecture that 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 is related work both in the parallel theorem proving community, and in the work on combining logical systems.

The majority of work on parallel theorem proving implementations followed decomposition of the search space [15,25,28,46,102,103], or allowed messages to be sent between the different provers working in parallel, using heuristics to decide on what messages are relevant to each prover [40,41,43] (surveys can be found in [14,42]). Both approaches typically look at decompositions into very few sub-problems (typically less than ten). 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. The second approach is more similar to ours, but there are some major differences still. First, there is no clear methodology for deciding what messages should be sent from one partition to another, or which partitions should receive messages from which other partitions. Second, there are no clear criteria for decomposing a theory into sub-problems.

Another related line of work focuses on combining logical systems, including the work of [8,81,85,97,104]. Here, the computational focus has been on treating combinations of signature-disjoint theories (allowing the queries to include symbols from all signatures), e.g., [8]. Recent work introduced sharing function symbols between two theories (e.g., [85]), but no algorithm allowed any sharing of relation symbols. All approaches either nondeterministically instantiate the (newly created) variables connecting the theories (e.g., [104]), 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, theorem provers and SAT procedures. We do not have any restrictions on the language besides finiteness.

Finally, work on formalizing and reasoning with *context*, including the work of [2,76]), 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 work on reasoning has focused on proof checking. Examples include GETFOL [55,56]. Automated reasoning has few successes; [16] presents one example.

7. Conclusions

In this paper we have shown that decomposing theories into partitions and reasoning over those partitions has potential 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 logic and in FOL. Different reasoning algorithms can be plugged-in for different partitions in these algorithms. We gave conditions on those reasoners that ensure that the combined reasoning procedure is sound and complete. Specialized versions of these algorithms for resolution strategies in FOL were provided. We showed that some of these algorithms simulate some order-based resolution strategies, while order-based strategies may simulate some of our algorithms in restricted cases. All our reasoning algorithms are suited for parallel and distributed processing.

We examined the efficiency of our theorem-proving algorithms and our SAT algorithm. Guided by both analyses, we suggested guidelines for achieving a good partitioning and proposed an algorithm for the automatic decomposition of theories that tries to minimize identified parameters.

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. More recently, we have integrated our partition-based reasoning algorithms into SRI's SNARK theorem prover and tested the effectiveness of our automated partitioning reasoning algorithms. Our experimental results [73] indicate that decomposing FOL theories automatically and using MP with resolution to answer queries reduces the number of resolution steps significantly, sometimes by orders of magnitude.

Acknowledgements

We particularly wish to thank Rina Dechter for many helpful and clarifying comments on a draft of this paper. We also wish to thank Mike Genesereth, John McCarthy, Leora Morgenstern, Nils Nilsson, and Mark Stickel for their comments on this work and for interesting discussions on general topics related to this work. Finally, we thank Rada Chirkova,

Tom Costello and Jörg Denzinger for comments on earlier papers describing this work. This research was supported in part by DARPA grant N66001-97-C-8554-P00004 and by AFOSR grant AF F49620-97-1-0207.

Appendix A. Proofs

A.1. 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 2.3), 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 (*Definition* 2.5). For a partitioning $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$, we say that a tree $G = (V, E, l)$ has 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 $\mathcal{L}(l(i, j)) \supseteq \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$.

We will show that all intersection graphs have a proper labeling. First, the following lemma provides the main argument behind all of the completeness proofs in this paper.

Lemma A.3 (Lemma 2.6). *Let $\mathcal{A} = \bigcup_{i \leqslant n} \mathcal{A}_i$ be a partitioned theory and assume that the graph G is a tree that has a proper labeling for the partitioning $\{\mathcal{A}_i\}_{i \leqslant n}$. Let $k \leqslant n$ 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 find a consequence of $\mathcal{A}_k$ that subsumes Q.*

Proof of Lemma 2.6. 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, as the reasoning procedure for $\mathcal{A}_1$ is complete for consequence finding. Assume that we proved the lemma for $|V| \leqslant n - 1$ and we prove the lemma for $|V| = n$.

In G, k has c neighbors, $i_1, \ldots, i_c$. (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 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. $\bigcup_{(i_1,j)\in E, j\neq k} l(i_1, j) \supseteq \mathcal{L}(\mathcal{B}_2 \cup \mathcal{A}_{i_1}) \cap \mathcal{L}(\mathcal{B}_1 \setminus \mathcal{A}_{i_1})$ 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 has a proper labeling 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 a sentence ψ' that subsumes ψ will be proved in $\mathcal{A}_{i_1}$ (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 has a proper labeling 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 a sentence subsuming $\psi' \Rightarrow Q$ will be generated in $\mathcal{A}_k$ (after some message passing). Thus, at some point a sentence subsuming Q will be generated in $\mathcal{A}_k$. □

Proof of Theorem 2.4. All we are left to prove is that the intersection graph G has a proper labeling. But if G is the intersection graph of the partitioning $\{\mathcal{A}_i\}_{i\leqslant n}$ then $l(i, j) = L(\mathcal{A}_i) \cap L(\mathcal{A}_j)$. If for $(i, j) \in E$ $L(l(i, j)) \not\supseteq L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$, 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 $L(l(i, j)) \supseteq L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$ and G has a proper labeling. The proof follows from Lemma 2.6. □

A.2. FORWARD-M-P with BREAK-CYCLES is sound and complete

Soundness is immediate, using the same argument as for Theorem 2.4. 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 $\{\mathcal{A}_i\}_{i\leqslant n}$. Then G' has a proper labeling for this partitioning.*

Proof of Lemma A.4. Assume there is a symbol p in $L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$ 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 $\mathcal{A}_1, \ldots, \mathcal{A}_n$). 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 $\langle (b, a_1), (a_1, a_2), \ldots, (a_c, a), (a, b)\rangle$ (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 $(b, a_1), (a_1, a_2), \ldots, (a_c, a)$ 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 con-

tradiction to having assumed that there is no such path at this step. Thus, there is no such p as mentioned above and $L(l(i,j)) \supseteq L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$. □

Proof of Theorem 2.7. The proof of Theorem 2.7 follows immediately from Lemmas 2.6 and A.4. □

A.3. BACKWARD-M-P (BMP) is sound and complete

Proof of Theorem 2.8. Notice that for a prover i to have a goal Q_i means that it needs to prove that the theory $\mathcal{A}_i \cup \{\neg Q_i\}$ is inconsistent. φ is a *subgoal* in a subgoal-disjunctive system if $\{\varphi\} \cup \mathcal{A}_i \vdash Q_i$. For a series of subgoals $\psi_1, \ldots, \psi_r$ in partition $\mathcal{A}_i$, $\{\psi_1 \vee \cdots \vee \psi_r\} \cup \mathcal{A}_i \vdash Q_i$. Also, if $\mathcal{A}_j$ is the partition to whom $\mathcal{A}_i$ sends its subgoals, then Q_j, the goal of partition $\mathcal{A}_j$, is $\psi_1 \vee \cdots \vee \psi_r$ at this point in time.

Let φ be a subgoal of $\mathcal{A}_i$. This means that $\mathcal{A}_i \cup \{\neg Q_i\} \vdash \neg\varphi$. Thus, our BMP algorithm readily reduces to MP, as $\neg\varphi$ would be sent from $\mathcal{A}_i$ to $\mathcal{A}_j$ in MP while φ would be disjoined with the goal of $\mathcal{A}_j$ in BMP, and both need to prove inconsistency of $\mathcal{A}_j \cup \{\neg Q_j \wedge \neg\varphi\}$, when Q_j is the goal of $\mathcal{A}_j$ before the arrival of φ. From the soundness and completeness of MP for graphs that are trees, we get soundness and completeness for BMP. □

A.4. Theorem 3.2: RESOLUTION-M-P (RES-MP) is sound and complete

Theorem A.5 [67]. *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 MP, 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 MP 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 4 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 $y_1, \ldots, y_r$. 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 5 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 5 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 gener-

ated 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.5. LINEAR-PART-SAT is sound and complete

Proof of Theorem 4.1. For each partition $\mathcal{A}_i, i \leqslant n$, 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 $(\alpha_1(p_{j_1},\ldots,p_{j_{l_i}}) \vee \cdots \vee \alpha_{a_i}(p_{j_1},\ldots,p_{j_{l_i}}))$, 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 it does the analogous operations to our forward message-passing algorithm MP (Fig. 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 2b 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 2.7, LINEAR-PART-SAT is sound and complete for satisfiability of $\mathcal{A}$. □

A.6. Time complexity of LINEAR-PART-SAT

Proof of Lemma 4.2. 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 4 of LINEAR-PART-SAT, and m_i the number of propositional symbols mentioned in $\mathcal{A}_i \setminus L(i)$ ($i \leqslant n$). Let us examine procedure LINEAR-PART-SAT (Fig. 12) step by step, computing the time needed for computation.

Computing the intersection graph takes $O(a \cdot k^2)$ 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 $O((c \cdot m) \cdot c)$, 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 $O(n^2 \cdot (n^2 \cdot m)) = O(n^4 \cdot m)$ (because $c \leqslant n$ and $|E| \leqslant |V|^2 = n^2$).

Step 2 of LINEAR-PART-SAT takes time $O(n \cdot m)$, 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 $i \leqslant n$. 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

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

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 $i \leqslant n$ 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

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

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

A.7. Time complexity for SPLIT

Proof of Proposition 5.1. Finding a minimum vertex separator R in G takes time $O(c \cdot |V|^{3/2} \cdot |E|)$. 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 $O(|V|^{3/2} \cdot |E|)$. Finding a minimum separator that separates s from t takes time $O(|V|^{1/2} \cdot |E|)$. In the worst case, each time we look for a minimum s-separator ($t = \mathit{nil}$), 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 $O(|V| \cdot |V|^{3/2} \cdot |E| + c \cdot |V|^{3/2} \cdot |E|) = O(|V|^{5/2} \cdot |E|)$. □

References

[1] R.K. Ahuja, J.B. Orlin, Distance directed augmenting path algorithms for maximum flow and parametric maximum flow problems, Naval Research Logistics Quarterly 38 (1991) 413–430.

[2] V. Akman, M. Surav, Steps toward formalizing context, AI Magazine 17 (3) (1996) 55–72.
[3] E. Amir, (De)Composition of situation calculus theories, in: Proc. National Conference on Artificial Intelligence (AAAI '00), Austin, TX, AAAI Press/MIT Press, 2000, pp. 456–463.
[4] E. Amir, Efficient approximation for triangulation of minimum treewidth, in: Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI '01), Morgan Kaufmann, San Mateo, CA, 2001, pp. 7–15.
[5] E. Amir, S. McIlraith, Partition-based logical reasoning, in: Proc. Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR '2000), Morgan Kaufmann, San Mateo, CA, 2000, pp. 389–400.
[6] E. Amir, S. McIlraith, Strategies for focusing structure-based theorem proving, Artificial Intelligence, in press.
[7] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability—a survey, BIT 25 (1985) 2–23.
[8] F. Baader, K.U. Schulz, Unification in the union of disjoint equational theories: combining decision procedures, in: Proceedings of the Eleventh International Conference on Automated Deduction, in: Lecture Notes in Artificial Intelligence, vol. 607, Springer, Berlin, 1992, pp. 50–65.
[9] A. Becker, R. Bar-Yehuda, D. Geiger, Randomized algorithms for the loop cutset problem, J. Artificial Intelligence Res. 12 (2000) 219–234.
[10] A. Becker, D. Geiger, Approximate algorithms for the loop cutset problem, in: Proc. Tenth Conference on Uncertainty in Artificial Intelligence (UAI '94), Morgan Kaufmann, San Mateo, CA, 1994, pp. 60–68.
[11] A. Becker, D. Geiger, A sufficiently fast algorithm for finding close to optimal junction trees, in: Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96), Morgan Kaufmann, San Mateo, CA, 1996, pp. 81–89.
[12] U. Bertele, F. Brioschi, Nonserial Dynamic Programming, Academic Press, New York, 1972.
[13] W.W. Bledsoe, A.M. Ballantyne, Unskolemizing, Technical Report Memo ATP-41, Mathematics Department, University of Texas, Austin, 1978.
[14] M.P. Bonacina, J. Hsiang, Parallelization of deduction strategies: an analytical study, J. Automat. Reason. 13 (1994) 1–33.
[15] M.P. Bonacina, J. Hsiang, On the representation of dynamic search spaces in theorem proving, in: C.-S. Yang (Ed.), Proceedings of the International Computer Symposium, 1996, pp. 85–94.
[16] P.E. Bonzon, A reflective proof system for reasoning in contexts, in: Proc. National Conference on Artificial Intelligence (AAAI '97) Providence, RI, 1997, pp. 398–403.
[17] R.B. Boppana, M. Sipser, The complexity of finite functions, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. 1, in: Algorithms and Complexity, Elsevier/MIT Press, Amsterdam/Cambridge, MA, 1990, pp. 757–804.
[18] G. Bossu, P. Siegel, Saturation, nonmonotonic reasoning and the closed world assumption, Artificial Intelligence 25 (1) (1985) 13–65.
[19] R.S. Boyer, Locking: a restriction of resolution, PhD thesis, Mathematics Department, University of Texas, Austin, 1971.
[20] A. Carbone, Interpolants, cut elimination and flow graphs for the propositional calculus, Ann. Pure Appl. Logic 83 (3) (1997) 249–299.
[21] R. Chadha, D.A. Plaisted, Finding logical consequences using unskolemization, in: Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS '93), in: Lecture Notes in Artificial Intelligence, vol. 689, Springer, Berlin, 1993, pp. 255–264.
[22] C.-L. Chang, R.C.-T. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.
[23] B.V. Chekassky, A.V. Goldberg, On implementing the push-relabel method for the maximum flow problem, Algorithmica 19 (4) (1997) 390–410.
[24] P. Cohen, R. Schrag, E. Jones, A. Pease, A. Lin, B. Starr, D. Gunning, M. Burke, The DARPA high-performance knowledge bases project, AI Magazine 19 (4) (1998) 25–49.
[25] S.E. Conry, D.J. McIntosh, R.A. Meyer, DARES: a distributed automated reasoning system, in: Proc. National Conference on Artificial Intelligence (AAAI '90), Boston, MA, AAAI Press/MIT Press, 1990, pp. 78–85.
[26] S.A. Cook, D.G. Mitchell, Finding hard instances of the satisfiability problem: a survey, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 35, American Mathematical Society, Providence, RI, 1997.

[27] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, McGraw Hill, New York, 1989.
[28] R. Cowen, K. Wyatt, BREAKUP: a preprocessing algorithm for satisfiability testing of CNF formulas, Notre Dame J. Formal Logic 34 (4) (1993) 602–606.
[29] P. Cox, T. Pietrzykowski, A complete nonredundant algorithm for reversed skolemization, Theoret. Comput. Sci. 28 (1984) 239–261.
[30] W. Craig, Linear reasoning. A new form of the Herbrand–Gentzen theorem, J. Symbolic Logic 22 (1957) 250–268.
[31] W. Craig, Three uses of the Herbrand–Gentzen theorem in relating model theory and proof theory, J. Symbolic Logic 22 (1957) 269–285.
[32] G.B. Dantzig, D.R. Fulkerson, On the max-flow min-cut theorem of networks, in: H.W. Kuhn, A.W. Tucker (Eds.), Linear Inequalities and Related Systems, in: Annals of Mathematics Study, vol. 38, Princeton Univ. Press, Princeton, NJ, 1956, pp. 215–221.
[33] A. Darwiche, Utilizing knowledge-based semantics in graph-based algorithms, in: Proc. National Conference on Artificial Intelligence (AAAI '96), Portland, OR, Morgan Kaufmann, San Mateo, CA, 1996, pp. 607–613.
[34] M. Davis, G. Logemann, D. Loveland, A machine program for theorem proving, Comm. ACM 5 (1962) 394–397.
[35] R. Dechter, Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition, Artificial Intelligence 41 (3) (1990) 273–312.
[36] R. Dechter, Y. El Fattah, Topological parameters for time-space tradeoff, Artificial Intelligence 125 (1) (2001) 93.
[37] R. Dechter, J. Pearl, Tree clustering for constraint networks, Artificial Intelligence 38 (1989) 353–366.
[38] R. Dechter, I. Rish, Directional resolution: the Davis–Putnam procedure, revisited, in: Proc. Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR '94), Morgan Kaufmann, San Mateo, CA, 1994, pp. 134–145.
[39] A. del Val, A new method for consequence finding and compilation in restricted language, in: Proc. National Conference on Artificial Intelligence (AAAI '99), Orlando, FL, AAAI Press/MIT Press, 1999, pp. 259–264.
[40] J. Denzinger, D. Fuchs, Cooperation of heterogeneous provers, in: D. Thomas (Ed.), Proc. Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99), Stockholm, Sweden, Morgan Kaufmann, San Mateo, CA, 1999, pp. 10–15.
[41] J. Denzinger, Knowledge-based distributed search using teamwork, in: V. Lesser (Ed.), Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, CA, MIT Press, Cambridge, MA, 1995, pp. 81–88.
[42] J. Denzinger, I. Dahn, Cooperating theorem provers, in: W. Bibel, P.H. Schmitt (Eds.), in: Automated Deduction. A Basis for Applications, vol. 2, Kluwer, Dordrecht, 1998, pp. 383–416.
[43] J. Denzinger, M. Fuchs, M. Fuchs, High performance ATP systems by combining several AI methods, in: Proc. Fifteenth International Joint Conference on Artificial Intelligence (IJCAI '97), Nagoya, Japan, Morgan Kaufmann, San Mateo, CA, 1997, pp. 102–107.
[44] E.A. Dinic, Algorithm for solution of a problem of maximum flow in networks with power estimation, Soviet Math. Dokl. 11 (1970) 1277–1280.
[45] N. Eisinger, H.J. Ohlbach, Deduction systems based on resolution, in: D.M. Gabbay, C.J. Hogger, J.A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, vol 1: Logical Foundations, Oxford University Press, Oxford, 1993.
[46] W. Ertel, OR-parallel theorem proving with random competition, in: Proc. International Conference on Logic Programming and Automated Reasoning (LPAR '92), in: Lecture Notes in Artificial Intelligence, vol. 624, 1992, pp. 226–237.
[47] S. Even, An algorithm for determining whether the connectivity of a graph is at least k, SIAM J. Comput. 4 (3) (1975) 393–396.
[48] S. Even, Graph Algorithms, Computer Science Press, 1979.
[49] S. Even, R. Endre Tarjan, Network flow and testing graph connectivity, SIAM J. Comput. 4 (4) (1975) 507–518.
[50] U. Feige, R. Krauthgamer, A polylogarithmic approximation of the minimum bisection, in: Proc. 41st IEEE Symp. on Foundations of Computer Science (FOCS '00), IEEE Press, 2000.

[51] R. Fikes, A. Farquhar, Large-scale repositories of highly expressive reusable knowledge, IEEE Intelligent Systems 14 (2) (1999).
[52] J.J. Finger, M.R. Genesereth, Residue: a deductive approach to design synthesis, Technical Report STAN-CS-85-1035, Stanford University, Computer Science Department, Stanford, CA, 1985.
[53] L.R. Ford Jr., D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, NJ, 1962.
[54] M.R. Genesereth, N.J. Nilsson, Logical Foundations of Artificial Intelligence, Morgan Kaufmann, San Mateo, CA, 1987.
[55] E. Giunchiglia, P. Traverso, A multi-context architecture for formalizing complex reasoning, Internat. J. Intelligent Systems 10 (1995) 501–539. Also, IRST Tech. Report #9307-26.
[56] F. Giunchiglia, GETFOL manual—GETFOL version 2.0, Technical Report DIST-TR-92-0010, DIST—University of Genoa, 1994. Available at http://ftp.mrg.dist.unige.it/pub/mrg-ftp/92-0010.ps.gz, and website at http://www-formal.stanford.edu/clt/ARS/Entries/getfol.
[57] A.V. Goldberg, R.E. Tarjan, A new approach to the maximum-flow problem, J. ACM 35 (4) (1988) 921–940.
[58] G. Gottlob, N. Leone, F. Scarcello, A comparison of structural CSP decomposition methods, in: Proc. Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99), Orlando, FL, Morgan Kaufmann, San Mateo, CA, 1999, pp. 394–399.
[59] A. Haken, The intractability of resolution, Theoret. Comput. Sci. 39 (1985) 297–308.
[60] G. Huang, Constructing Craig interpolation formulas, in: First Annual International Conference on Computing and Combinatorics (COCOON '95), 1995, pp. 181–190.
[61] K. Inoue, Linear resolution for consequence finding, Artificial Intelligence 56 (2–3) (1992) 301–353.
[62] P. Klein, S.A. Plotkin, S. Rao, Excluded minors, network decomposition, and multicommodity flow, in: Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing, 1993, pp. 682–690.
[63] T. Kloks, Treewidth: Computations and Approximations, Lecture Notes in Computer Science, vol. 842, Springer, Berlin, 1994.
[64] J. Kohlas, R. Haenni, S. Moral, Propositional information systems, J. Logic Comput. 9 (5) (1999) 651–681.
[65] J. Krajíček, Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic, J. Symbolic Logic 62 (2) (1997) 457–486.
[66] S.L. Lauritzen, D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, J. Roy. Statist. Soc. B 50 (2) (1988) 157–224.
[67] R.C.-T. Lee, A completeness theorem and a computer program for finding theorems derivable from given axioms, PhD Thesis, University of California, Berkeley, 1967.
[68] T. Leighton, S. Rao, An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms, in: Proc. 29th IEEE Symp. on Foundations of Computer Science (FOCS'88), 1988, pp. 422–431.
[69] D.B. Lenat, Cyc: a large-scale investment in knowledge infrastructure, Comm. ACM 38 (11) (1995) 33–38.
[70] F. Lin, On strongest necessary and weakest sufficient conditions, in: Proc. Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR '2000), 2000, pp. 167–175.
[71] J.W. Lloyd, R.W. Topor, A basis for deductive database systems, J. Logic Programming 2 (1985) 93–109.
[72] D.W. Loveland, Automated Theorem Proving: A Logical Basis. Fundamental Studies in Computer Science, North-Holland, Amsterdam, 1978.
[73] B. MacCartney, Sh. McIlraith, E. Amir, T. Uribe, Practical partition-based theorem proving for large knowledge bases, in: Proc. Eighteenth International Joint Conference on Artificial Intelligence (IJCAI '03), Acapulco, Mexico, Morgan Kaufmann, San Mateo, CA, 2003, pp. 89–96.
[74] P. Marquis, Knowledge compilation using theory prime implicates, in: Proc. Fourteenth International Joint Conference on Artificial Intelligence (IJCAI '95), Montreal, Quebec, 1995, pp. 837–843.
[75] P. Marquis, Consequence finding algorithms, in: D. Gabbay, Ph. Smets (Eds.), Handbook of Defeasible Reasoning and Uncertainty Management Systems, vol 5: Algorithms for Defeasible and Uncertain Reasoning, Kluwer, Dordrecht, 2000.
[76] J. McCarthy, S. Buvač, Formalizing context (expanded notes), in: A. Aliseda, R.J. van Glabbeek, D. Westerståhl (Eds.), Computing Natural Language, in: CSLI Lecture Notes, vol. 81, Center for the Study of Language and Information, Stanford University, Stanford, CA, 1998, pp. 13–50.

[77] S. McIlraith, Logic-based abductive inference, Technical Report KSL-98-19, Knowledge Systems Lab, Department of Computer Science, Stanford University, 1998.
[78] S. McIlraith, E. Amir, Theorem proving with structured theories, in: Proc. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI '01), Seattle, WA, Morgan Kaufmann, San Mateo, CA, 2001, pp. 624–631.
[79] A.R. Meyer, A note on the length of Craig's interpolants Technical Memo MIT/LCS/TM-183, Massachusetts Institute of Technology, Laboratory for Computer Science, 1980.
[80] E. Minicozzi, R. Reiter, A note on linear resolution strategies in consequence-finding, Artificial Intelligence 3 (1972) 175–180.
[81] G. Nelson, D.C. Oppen, Simplification by cooperating decision procedures, ACM Trans. Programming Lang. Syst. 1 (2) (1979) 245–257.
[82] T.J. Park, A. Van Gelder, Partitioning methods for satisfiability testing on large formulas, in: Proceedings of the Thirteenth International Conference on Automated Deduction (CADE-13), Springer, Berlin, 1996, pp. 748–762.
[83] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, San Mateo, CA, 1988.
[84] D.A. Plaisted, The search efficiency of theorem proving strategies, in: Proceedings of the Twelfth International Conference on Automated Deduction (CADE-12), 1994, pp. 57–71.
[85] C. Ringeissen, Cooperation of decision procedures for the satisfiability problem, in: F. Baader, K.U. Schulz (Eds.), Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany), Applied Logic, Kluwer Academic, Dordrecht, 1996, pp. 121–140.
[86] I. Rish, R. Dechter, Resolution versus search: two strategies for SAT, J. Automat. Reason. 24 (1–2) (2000) 225–275.
[87] N. Robertson, P.D. Seymour, Graph minors. II: algorithmic aspects of treewidth, J. Algorithms 7 (1986) 309–322.
[88] J.A. Robinson, A machine-oriented logic based on the resolution principle, J. ACM 12 (1) (1965) 23–41.
[89] 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.
[90] B. Selman, H. Kautz, Domain-independent extensions to GSAT: solving large structured satisfiability problems, in: Proc. Eighteenth International Joint Conference on Artificial Intelligence (IJCAI '03), Acapulco, Mexico, 1993.
[91] B. Selman, H.A. Kautz, B. Cohen, Noise strategies for local search, in: Proc. National Conference on Artificial Intelligence (AAAI '94), Seattle, WA, 1994, pp. 337–343.
[92] B. Selman, H.J. Levesque, D. Mitchell, A new method for solving hard satisfiability problems, in: P. Rosenbloom, P. Szolovits (Eds.), Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, CA, American Association for Artificial Intelligence, AAAI Press, Menlo Park, CA, 1992, pp. 440–446.
[93] B. Selman, D. Mitchell, H. Levesque, Generating hard satisfiability problems, in: B. Selman, D. Mitchell, H. Levesque (Eds.), Artificial Intelligence, 1997, pp. 17–29.
[94] G. Shafer, P. Shenoy, Probability propagation, Ann. Math. Artificial Intelligence 2 (1990) 327–352.
[95] J. Shi, J. Malik, Normalized cuts and image segmentation, in: Proc. of IEEE Conference on Computer Vision and Pattern Recognition, 1997, pp. 731–737.
[96] K. Shoikhet, D. Geiger, A practical algorithm for finding optimal triangulations, in: Proc. National Conference on Artificial Intelligence (AAAI '97), Providence, RI, Morgan Kaufmann, San Mateo, CA, 1997, pp. 185–190.
[97] R.E. Shostak, Deciding combinations of theories, J. ACM 31 (1984) 1–12.
[98] J.R. Slagle, Interpolation theorems for resolution in lower predicate calculus, J. ACM 17 (3) (1970) 535–542.
[99] J.R. Slagle, C.-L. Chang, R.C.T. Lee, Completeness theorems for semantic resolution in consequence-finding, in: Proc. First International Joint Conference on Artificial Intelligence (IJCAI '69), Washington, DC, 1969, pp. 281–285.
[100] J. Slaney, T.J. Surendonk, Combining finite model generation with theorem proving: problems and prospects, in: F. Baader, K.U. Schulz (Eds.), Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany), Applied Logic, Kluwer Academic, Dordrecht, 1996, pp. 141–156.

[101] M.E. Stickel, A Prolog technology theorem prover: a new exposition and implementation in Prolog, Theoret. Comput. Sci. 104 (1992) 109–128.
[102] G.C.J. Sutcliffe, A heterogeneous parallel deduction system, in: FGCS'92 Workshop on Automated Deduction: Logic Programming and Parallel Computing Approaches, 1992.
[103] C.B. Suttner, SPTHEO, J. Automat. Reason. 18 (1997) 253–258.
[104] C. Tinelli, M.T. Harandi, A new correctness proof of the Nelson–Oppen combination procedure, in: F. Baader, K.U. Schulz (Eds.), Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany), Applied Logic, Kluwer Academic, Dordrecht, 1996, pp. 103–120.
[105] G.S. Tseitin, On the complexity of proofs in propositional logics, Seminars in Mathematics 8 (1970).
[106] J.D. Ullman, Principles of Database and Knowledge-base Systems, vol. 1, Computer Science Press, 1988.
[107] A. Urquhart, Hard examples for resolution, J. ACM 34 (1) (1987) 209–219.