Compact Propositionalizations of First-Order Theories

Deepak Ramachandran and Eyal Amir
Computer Science Department
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{dramacha,eyal}@cs.uiuc.edu

Abstract. We present new insights and algorithms for converting reasoning problems in monadic First-Order Logic (includes only 1-place predicates) into equivalent problems in propositional logic. Our algorithms improve over earlier approaches in two ways. First, they are applicable even without the unique-names and domain-closure assumptions, and for possibly infinite domains. Therefore, they apply for many problems that are outside the scope of previous techniques. Secondly, our algorithms produce propositional representations that are significantly more compact than earlier approaches, provided that some structure is available in the problem. We examined our approach on an example application and discovered that the number of propositional symbols that we produced is smaller by a factor of $f \approxeq 50$ than traditional techniques, when those techniques can be applied. This translates to a factor of about 2^f increase in the speed of reasoning for such structured problems.

1 Introduction

It is often advantageous to perform reasoning with a First-Order Logic (FOL) theory by first transforming it into an equivalent propositional theory (*propositionalization*) and then using propositional inference methods on it [8]. There are a number of good reasons for this: Propositional Inference is decidable and the last 20 years of research have resulted in relatively efficient and successful algorithms (e.g. [16, 15]).

The simplest propositionalization of a First-Order theory is obtained by creating a proposition for every ground atom of the theory. Quantified variables are systematically replaced with constants from the language in the theory. This leads to $O(|P||C|^k)$ propositions, where $|P|$ and $|C|$ are the number of predicates and constants in the FOL theory and k is the maximum arity of any predicate. The completeness of this propositionalization requires the *Domain Closure Assumption*(DCA[1]; every object in the universe is referenced by a constant symbol), and the *Unique Names Assumption* (UNA; every constant symbol refers to a unique object).

Propositionalization is used in a number of applications involving First-Order representations, such as Planning [9] and Relational Data Mining [12]. Many specialized propositionalization algorithms exist for such domains that use prior knowledge to construct efficient (small) propositionalizations. ILP systems such as LINUS [7] use the training data to guide construction of the propositionalizations. Bottom-Up Propositionalization [11] is tailored for biochemical databases and chooses propositions by constructing frequently occuring fragments of linearly connected atoms.

While the DCA holds for a number of important domains (e.g., Planning) there are a number of applications where it is not reasonable, notably in problems with potentially unbounded or infinite number of objects. Also, the number of propositions that are generated for many problems (e.g., planning problems) is still prohibitively large. We address these problems in what follows.

In this paper we present algorithms that propositionalize a general monadic function-free First-Order Theory. Our algorithms do not make the DCA or the UNA, and they yield a set of propositional symbols that is significantly smaller than previous algorithms, if some structural assumptions hold. We detail these results below.

First, we show that every monadic FOL theory T (with or without DCA,UNA) can be reformulated into a propositional theory, T_p, with at most $3^P + PC$ propositional symbols, if P, C are the number of predicates and constant symbols in T, respectively. Consequently, we achieve a better complexity bound for the decidable class of monadic FOL than was known before: To the best of our knowledge, the best earlier result about inference (e.g., satisfiability) in monadic FOL is $O(2^{E^{2 \cdot U^{2^P}}})$ for a given FOL formula φ with E existentially quantified variables or constant symbols, U universally quantified variables, and P predicates [6] (this follows from the need to check all Herbrand structures of size at most $E^{2 \cdot U^{2^P}}$). In comparison, our result is independent of the number of quantifiers in φ and is at most $O(2^{3^P+PC})$ (significantly smaller in practice, using current propositional SAT solvers).

Secondly, we use structure (in the manner of *Partition-based Reasoning* [3, 2]) to obtain a propositionalization that has an exponential-factor fewer propositions than the one above. Our method is quite general and does not require special knowledge of the underlying domain or any semantic restrictions. Precisely, in many real-world cases we can partition a given theory T into n loosely dependent partitions that are arranged in a tree structure. Each partition includes axioms that are restricted to a fraction of the signature of T. If each partition has $2P/n$ predicates and C/n constant symbols, then the number of propositions that we need is at most $\frac{2PC}{n} + 3^{2P/n}n$. For example, if $P = 200$, $C = 2500$, $n = 200$, then we need at most $26,200$ propositional symbols (compared with $500,000+3^{200}$ when no structure is used). For further comparison, consider the case when we can make the DCA,UNA. Current techniques yield $1,000,000$ propositions for the same problem. This is a factor of about 40 times more propositions (translating to computation that is slower by a factor of about 2^{40}).

The rest of this paper is organized as follows Section 2.1 gives some preliminary definitions and Section 2.2 gives a motivating ex-

[1] This is also called the *Closed-World Assumption* (CWA)

ample. Section 3 introduces the problem of Propositionalization, and presents our results and algorithm for propositionalizing without DCA,UNA. Section 4 presents our algorithm for compact propositionalization using structure and an analysis of the number of propositions it creates.

2 Preliminaries

2.1 Definitions

We make some definitions here that we will use later. We assume familiarity with the standard definitions of FOL. Recall that an *atomic formula* of a language is of the form $P(t_1 \ldots t_k)$, where P is a k-place predicate and $t_1 \ldots t_k$ are terms. Atomic formulas not containing variables are called *atoms*. A *signed atomic formula* or *literal* is either an atomic formula or a negated atomic formula. The *Matrix* of a formula F, denoted $Matrix(F)$, is the formula obtained by deleting each occurrence of a quantifier as well as the occurrence of the variable immediately to its right. In this paper we will mainly be concerned with monadic predicates, i.e., predicates of one variable. A *factor* is a monadic First-Order formula that is a monadic atom or is of the form $[\neg]\exists x(L_1 \wedge L_2 \ldots :_n)$ where each L_i is a monadic literal with argument x and each predicate occurs at most once in some L_i.

For a logical theory τ, $L(\tau)$ is its signature (the set of non-logical symbols) and $\mathcal{L}(\tau)$ is its language (the set of formulas built with $L(\tau)$). $L_{pred}(\tau)$ and $L_{const}(\tau)$ are the set of predicate symbols and constant symbols respectively of τ. For the rest of the paper, we assume that all logical theories are function-free. We will use the convention that A, B, C stand for constants in a logical theory, x, y, z are variables and a, b, c are objects in a universe.

2.2 Motivating Example

We will now present a machine-scheduling problem involving constraint satisfaction for which current propositionalization methods do not scale very well. Later we will develop a partitioned propositionalization algorithm, which performs two orders of magnitude better when applied to this problem.

A factory has two machines Q^1 and Q^2 that can process items in incremental time steps. Every item is in one of n states at any point in time. We use $P_i(j)$ to denote that our item is in state $j \leq n$ at time i. The machine is either available or not at any point in time to process items in state j (it may process more than one item at a time, though, if they are in different states (e.g., consider machines that works with a pipeline)). We write $Q_i^r(j)$ to say that the machine r is available at time i to process an item in state j.

The function of the machine is to preserve the state of the item. Thus, the following relations hold for every time step i.

$$\forall j\; Q_i^1(j) \Rightarrow (P_i(j) \Rightarrow P_{i+1}(j)) \quad (1)$$
$$\forall j\; Q_i^2(j) \Rightarrow (P_i(j) \Rightarrow P_{i+1}(j)) \quad (2)$$

At certain predefined time step, an item in a certain state can be moved to another state if neither machine is scheduled to work on it. For example in time step 2314, an item in state 42 can be moved to state 29. We represent this information as follows.

$$\neg Q_i^1(42) \wedge \neg Q_i^2(42) \Rightarrow (P_{77}(42) \Rightarrow P_{78}(29)) \quad (3)$$

If neither machine is available to process the item and it is not allowed to change state then it is lost.

In addition, there are a number of constraints on the scheduling of the machines. For example,

$$\forall i\; Q_{12}^1(i) \Rightarrow (\neg Q_{13}^1(i) \vee \neg Q_{14}^1(i))$$
$$\neg Q_{313}^1(56)\ \vee\ \neg Q_{314}^2(56)$$
$$\vdots$$

Assume that we know such relationships among the first 125 time steps, but have no knowledge about steps beyond 125 (e.g., because our scheduling personnel do not look beyond 125 steps).

Finally, suppose that we know that an item is in state 1 initially (time 1), and we wish to know if it is possible for the system to reach a state n after say, 125 steps. Call the axioms above T. Then our task is to determine if $T \wedge P_1(1) \models P_{125}(n)$.

Assume that we try to solve this problem by making the DCA and a naive propositionalization in the manner of [9]. Then the number of propositional symbols is the number of constants times the number of predicates, i.e., $125 \cdot 3n$ (there are n states, 3 predicates per state, and 125 time steps). This number is impractical for more than a small number of states n. For example, for $n = 2500$ states we get $937,500$ propositional variables, a number that is way beyond the capabilities of current SAT solvers. In the next two sections we will see an approach that leads to a solution for this problem without DCA and with $\approxeq 20,000$ variables (a reduction by a factor of 50), which is a solvable size with state-of-the-art SAT technology.

3 Propositionalizing First-Order Theories

First-Order Theory	Propositionalization
$\neg(P(A) \wedge Q(B))$	$\neg(P_A \wedge Q_B)$
$\forall x P(x) \Rightarrow \forall y \neg Q(y)$	$(P_A \wedge P_B \wedge P_C \Rightarrow (\neg Q_A \wedge \neg Q_B \wedge \neg Q_C))$
$\exists z(R(z, C) \wedge \neg Q(x))$	$(R(A, C) \wedge \neg Q(A)) \vee (R(B, C) \wedge \neg Q(B)) \vee (R(C, C) \wedge \neg Q(C))$

Table 1. Propositionalization with the DCA

Table 1 gives examples of propositionalizations of First-Order theories, created in the manner of [8] by replacing ground atoms with the corresponding subscripted proposition (e.g $P(A)$ with P_A), Universally quantified formulas with the conjunction of their instantiations ($\forall x(P(x) \vee Q(x))$ with $(P_A \vee Q_A) \wedge (P_B \vee Q_B) \ldots$) and existentially quantified formulas with the disjunction of their instantiations (e.g $\exists x P(x, C)$ with $P_{\langle A,C\rangle} \vee P_{\langle B,C\rangle} \ldots$). We will refer to this as the *Naive Propositionalization*.

Converting a First-Order theory into a propositional satisfiability problem as shown, is neither sound nor complete unless the DCA is made. The intuition is that there may be a model M with some object a in its universe such that $P^M(a)$ is true. There, $M \models \exists P(x)$, but there need not exist some constant A in the theory such that $P(A)$ is true, unless the DCA holds.

The DCA is reasonable in a Planning scenario, since one expects the world to be more or less completely specified by the initial assumptions and operator definitions. Several techniques have been employed in the planning literature to obtain optimized propositional encodings of planning problems stated in a Situation Calculus [13] formalism. Some use Lifted Causal Encodings, an idea borrowed from the Theorem Proving community and others reduce the number of variables by compiling away state variables and fluents. A good introduction is [9].

3.1 Propositionalization Without the DCA

We present a technique for constructing a propositionalization of a monadic function-free FOL theory with open domain semantics. It is common to try to do this by creating a new constant for every existentially quantified variable. This does not work for universally quantified formulas. For example, the formula $\forall x P(x) (\equiv \neg\exists x \neg P(x))$ would become $P(c)$ for some new constant c, but this does not mean the predicate P is true for *every* argument. In general, it is impossible to describe an algorithm to convert an arbitrary FOL formula into a propositionalization since that would imply the existence of a decision procedure for FOL. Hence, we choose to concentrate on *decidable fragments* [6] of FOL, specifically monadic logic.

Informally our idea is as follows: Given a function-free monadic FOL formula τ, we convert it into a form called the *standard propostional-ready form*(SPR). We next create a propositional theory $\mathcal{P}(\tau)$ defined by purely syntactic operations on the SPR of τ. $P(\tau)$ contains two kinds of propositional symbols - symbols of the form P_a which represent atoms and symbols of the form $E_{\langle P,Q\ldots\rangle}$ which are the propositional equivalent of $\exists x(P(x) \wedge Q(x)\ldots)$. We then define a set of consistency axioms $\mathcal{E}(\tau)$ which preserve the semantic meaning of these symbols. We show that the propositional theory $\mathcal{P}(\tau) \wedge \mathcal{E}(\tau)$ is implicationally equivalent to τ. Reasoning with it is therefore sound and complete.

Definition 1 *A monadic First-Order formula τ in prenex form is in* proposition-ready form *iff $Matrix(\tau)$ is a conjunction of disjunctions of factors.*

Theorem 1 *Algorithm Make-SPR (Figure 1) converts every function-free monadic First-Order formula τ to a logically equivalent formula τ'' in proposition-ready form.*

In algorithm Make-SPR, we have used the notation $var(l)$, where l is a literal, to mean the (unique) free variable ,if any, of l.

For example, the conversion of $\forall x\exists y(P(x) \wedge Q(y))$ to proposition-ready form by Algorithm Make-SPR is given below:

$$\begin{aligned}
\forall x\exists y(P(x) \wedge Q(y)) &\equiv \exists y\forall x(P(x) \wedge Q(y)) \\
&\equiv \exists y(\forall x P(x) \wedge Q(y)) \\
&\equiv \exists y(\neg\exists x\neg P(x) \wedge Q(y)) \\
&\equiv (\neg\exists x\neg P(x)) \wedge \exists y Q(y)
\end{aligned}$$

Note that in the first step we have used the fact that the relative order of the existential and universal operators in an FOL formula is irrelevant when all the predicates are monadic [6].

For any monadic FOL formula τ, let the result of Make-SPR be the *standard proposition-ready form* of τ, $SPR(\tau)$. We now describe a propositionalization of τ created by purely syntactic operations on $SPR(\tau)$. First we define the set of propositional symbols that will appear in this propositionalization.

If L is the language of a monadic first order Formula τ then,

$$Prop(L) \triangleq \{P_c | P \in L_{pred}(L), c \in L_{const}(L)\} \cup \{E_{\langle[\neg]P_1,[\neg]P_2,\ldots[\neg]P_n\rangle} | P_1 \ldots P_n \in L_{pred}(L)\}$$

Definition 2 $\mathcal{P} : \mathcal{L}(L) \rightarrow \mathcal{L}(Prop(L))$ *is defined as follows.*

If τ is in proposition-ready form,

1. *If $\tau = P(a)$ then $\mathcal{P}(\tau) \triangleq P_a$*
2. *If $\tau = \exists x([\neg]P_1(x) \wedge [\neg]P_2(x), \ldots [\neg]P_n(x))$ then $\mathcal{P}(\tau) \triangleq E_{\langle[\neg]P_1,[\neg]P_2\ldots[\neg]P_n\rangle}$*
3. *If $\tau = \neg\tau'$ then $\mathcal{P}(\tau) \triangleq \neg\mathcal{P}(\tau')$*
4. *If $\tau = \tau_1 \wedge \tau_2 \ldots \tau_n$, then $\mathcal{P}(\tau) \triangleq \mathcal{P}(\tau_1) \wedge \mathcal{P}(\tau_2) \ldots \mathcal{P}(\tau_n)$*
5. *If $\tau = \tau_1 \vee \tau_2 \ldots \tau_n$, then $\mathcal{P}(\tau) \triangleq \mathcal{P}(\tau_1) \vee \mathcal{P}(\tau_2) \ldots \mathcal{P}(\tau_n)$*

Finally, if τ is not in proposition-ready form, then $Prop(\tau) = Prop(SPR(\tau))$.

Make-SPR(Monadic FOL formula τ)

1. Rearrange $Matrix(\tau)$ into Conjunctive Normal Form F
2. Move the existential quantifiers in the Prefix of τ to the head of the formula, to give $\tau = \exists x_1 \ldots \exists x_m \forall y_1 \ldots \forall y_n F$
3. **For** each y_i
   (a) **For** each conjunct $C_j = (l_1 \vee \ldots l_m)$ of F
      i. $C'_j := (\neg\exists y_i \bigwedge_{var(l_k)=y_i,k\leq m}(\neg l_k) \vee \bigvee_{var(l_k)\neq y_i,k\leq m}(l_k))$
   Call the resulting formula $\tau' := \exists x_1 \ldots \exists x_m F'$ where $F' = \bigwedge C'_j$
4. Convert F' to Disjunctive Normal Form
5. **For** each x_i
   (a) **For** each disjunct $D_j = (l_1 \wedge \ldots l_m)$ of F
      i. $D'_j := (\exists x_i \bigwedge_{var(l_k)=x_i,k\leq m}(l_k) \wedge \bigwedge_{var(l_k)\neq x_i,k\leq m}(l_k))$
   Let $\tau'' = \bigvee D'_j$
6. Rearrange τ'' into CNF
7. **Return** τ''

Figure 1. Conversion to Standard Proposition Ready Form

By replacing each factor by a propositional symbol, we have created a propositionalization $\mathcal{P}(\tau)$ "consistent" with the FOL theory τ. However the meaning of factors like $\exists x(P(x) \wedge Q(x))$ are lost. To ensure that each Propositional symbol $E_{\langle P,Q\ldots\rangle}$ retains the semantics of its First-Order counterpart,we assert a set of axioms $\mathcal{E}(\tau)$ than ensure the consistency of the propositionalization.

Let $P = \{P_1, P_2, \ldots P_n\}$ be a set of monadic FOL predicates and C be a set of constants. Then,

$$\begin{aligned}
\mathcal{E}(P,C) \triangleq & \bigwedge_{\substack{c \in C \\ k \leq n \\ i_1\ldots i_k \in P}} ([\neg]P_{i_1 c} \wedge \ldots [\neg]P_{i_k c} \Rightarrow E_{\langle[\neg]P_1,\ldots[\neg]P_k\rangle}) \wedge \\
& \bigwedge_{\substack{k \leq n \\ l < k \\ i_1\ldots i_k \in P}} (E_{\langle[\neg]P_{i_1},\ldots[\neg]P_{i_k}\rangle} \Rightarrow E_{\langle[\neg]P_{i_1},\ldots[\neg]P_{i_l}\rangle} \wedge E_{\langle[\neg]P_{i_{l+1}},\ldots[\neg]P_{i_k}\rangle})
\end{aligned}$$

The first set of axioms in $\mathcal{E}(\tau)$ ensures that the existence of a constant c for which a conjunction of literals instantiated with c can be deduced, implies that the corresponding existential proposition is true. The second set asserts that if any conjunction of literals instantiated with the same variable is true, then all subsets of that conjunction is true as well. We sometimes use $\mathcal{E}(\tau)$ to mean $\mathcal{E}(L_{pred}(\tau), L_{const}(\tau))$.

For example, consider the formula τ in Table 2. The development of its propositionalization is shown.

Our main result follows:

Theorem 2 (Consistency and Completeness) *If α and β are monadic FOL theories, $\alpha \models \beta$ iff $\mathcal{P}(\alpha) \wedge \mathcal{E}(L_{pred}(\alpha), L_{const}(\alpha)) \models \mathcal{P}(\beta) \wedge \mathcal{E}(L_{pred}(\beta), L_{const}(\beta))$*

τ	$\forall x \exists y[(P(x) \vee Q(x) \vee R(y)) \wedge \neg S(y)]$
SPR(τ)	$(\neg\exists x(\neg P(x) \wedge \neg Q(x)) \vee \exists y(R(y) \wedge \neg S(y))) \quad \wedge$ $(\exists y(\neg S(y)) \vee \exists y(R(y) \wedge \neg S(y)))$
$\mathcal{P}(\tau)$	$(\neg E_{\langle \neg P, \neg Q \rangle} \vee E_{\langle R, \neg S \rangle}) \quad \wedge$ $(E_{\langle \neg S \rangle} \vee E_{\langle R, \neg S \rangle})$
$\mathcal{E}(\tau) =$ $\mathcal{E}(L_{pred}(\tau), \{A\})$	$P_A \Rightarrow E_{\langle P \rangle} \quad \wedge \quad P_A \wedge \neg Q_A \Rightarrow E_{\langle P, \neg Q \rangle} \quad \wedge$ $E_{\langle P, \neg Q \rangle} \Rightarrow E_{\langle P \rangle} \wedge E_{\langle \neg Q \rangle} \dots$

Table 2. $\mathcal{P}(\tau) \wedge \mathcal{E}(\tau)$ is the propositionalization of τ using Make-SPR.

PART-PROP($\{\mathcal{A}_i\}_{i \le n}, G$)

1. $\{\mathcal{A}_i\}_{i \le n}$ a partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a graph describing the connections between the partitions.
2. **For** $i := 1 \rightarrow n$ **do**
   (a) $\mathcal{A}_i{}' := \mathcal{P}(\mathcal{A}_i) \cup \mathcal{E}(\mathcal{A}_i)$
   (b) **For** $j := 1 \rightarrow n$ *do*
      i. $l'(i, j) := Prop(l(i, j))$
3. $G' := (V, E, l')$
4. $Q' := \mathcal{P}(Q)$
5. **Return** $(\{\mathcal{A}_i{}'\}_{i \le n}, G', Q')$

Figure 2. Compact Propositionalizing algorithm

Theorem 2 formalizes the notion that reasoning in $\mathcal{P}(\tau) \wedge \mathcal{E}(\tau)$ is equivalent to reasoning in τ. Thus $\mathcal{P}(\tau) \wedge \mathcal{E}(\tau)$ is the *propositionalization* of τ.

This approach creates $|P| \cdot |C| + 3^{|P|}$ propositional symbols in $PROP(\tau)$ which can be unacceptably large. We describe a method to reduce this number significantly in the next section.

4 Structure and Compact Propositionalization

Section 3.1 describes a propositionalization which can require an excessively large number of propositions. An analysis of most domains shows that many of these propositions are unnecessary. For the purpose of soundness and completeness it is clearly not required to instantiate a literal with every possible constant as an argument, but only those from which useful inferences can be made. Deciding which propositions to retain should therefore be an important aspect of an efficient propositionalization algorithm. One popular strategy has been to use typed predicates or *Many-sorted Logics* [14] to restrict the set of objects that can substitute for an argument in a predicate.

We are interested in a more general setting where an efficient propositionalization can be derived purely from the syntactic features of the theory independent of its intended semantics. Specifically, our intention is to determine which predicates need to be instantiated with which constants by analyzing the global properties of the theory. Our idea is to use the principles of *Partition-based Reasoning* [3, 2] to do so.

The next section describes an algorithm that finds a more compact propositionalization using partitioning. We present this algorithm, its analysis and application to the machine scheduling problem in Section 2.2 in the following.

FORWARD-MP($\{\mathcal{A}_i\}_{i \le n}, G, Q$)

1. $\{\mathcal{A}_i\}_{i \le n}$ a partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a graph describing the connections between the partitions, Q a query in $\mathcal{L}(A_k)(k \le n)$
2. Determine $\prec$ as in Definition 4
3. Concurrently
   (a) Perform consequence finding for each partition $\mathcal{A}_i, i \le n$.
   (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 proved in $\mathcal{A}_k$ return YES

Figure 3. Message Passing

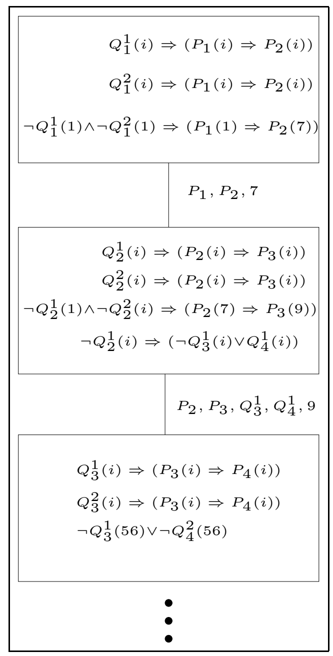

Figure 4. A partitioning of theory T (Section 2.2), which describes machines and item state in a factory.

4.1 Factored Propositionalization

We will now describe our factored propositionalized algorithm.

Briefly, the First-Order Theory τ is divided into sub-domains corresponding to subsets of the predicates and constants of τ. This is done by hand or automatically (e.g., [3, 1]) such that the predicates and the constants are divided somewhat evenly among the sub-domains (which we call *partitions*). Figure 4 shows this applied to our example from Section 2.2. Then, each partition is individually propositionalized. After this we may choose to do either of two things. The partitioning of the theory can be retained and reasoning can be done using the (sound and complete) Message-Passing algorithms described in [3, 2] (Henceforth, we call this *Method 1*). Alternatively, the domains can be merged together, creating a single propositionalized theory, to which any propositional SAT solver or theorem prover can be applied (Henceforth, we call this *Method 2*).

Our method uses the theory of *partition-based reasoning* to determine which propositions can be safely left out, while maintaining completeness. It does so by exploiting Craig's Interpolation Theorem [4].

Formally, $\{\mathcal{A}_i\}_{i \leq n}$ is a *partitioning* of a logical theory $\mathcal{A}$ if $A = \cup_i \mathcal{A}_i$. Each individual $\mathcal{A}_i$ is called a *partition*. We associate a tree $G = (V, E, l)$ with this partitioning, such that each node i represents an individual partition $\mathcal{A}_i, (V = \{1, \ldots, n\})$, and we require the graph to be *properly labeled* for $A = \cup_i \mathcal{A}_i$:

Definition 3 (Proper Labeling) *For a partitioning $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$, 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[2] of $\mathcal{A}$ on the two sides of the edge (i, j) in G, it is true that $l(i, j) \supseteq L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$.*

For every partitioning there are many such trees, and it is important to find such a tree with optimal computational properties (more on that below). A very similar property, the *running intersection property*, is used in literature on probabilistic graphical models (e.g., [10]) and CSPs (e.g., [5]).

Figure 2 presents algorithm PART-PROP. The input to PART-PROP is a partitioning of the monadic FOL theory τ and a graph G. PART-PROP propositionalizes each partition $\mathcal{A}_i$ and the link languages $l(i, j)$ in the manner of Section 3.1. The link language $l(i, j)$ is the language in which messages will be passed from partition i to j. PART-PROP returns the partitioning for the propositionalized theory.

Recall our example from Section 2.2. Procedure PART-PROP (Figure 2) applies to it by examining a partitioning of the set of axioms as presented diagrammatically in Figure 4. Here, every partition includes the set of axioms that describe the effects of the machine being ready and not ready, as well as knowledge about the availability of the machine in different times for different states of our item. The edges between the partitions are labeled with the set of nonlogical (predicate and constant) symbols that are shared between partitions.

Figure 3 reproduces the Message Passing algorithm FORWARD-MP from [3, 2]. Given a partitioned theory, its intersection graph and query Q in the language of one of the partitions $\mathcal{A}_k$, FORWARD-MP will try to prove Q. (If the query is not contained in any partion $\mathcal{A}_k$, then a new partition can be created which contains just the nonlogical symbols of the query Q, and the graph G must be extended with the new partition.)

FORWARD-MP defines a partial ordering on the partitions based on distance to the query partition $\mathcal{A}_k$ as follows:

Definition 4 ($\prec$) *Given partitioned theory $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$, intersection 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 $dist(i, k) < dist(j, k)$.*

Consequence-Finding is performed within each partition independently and concurrently with the other partitions. Those consequences that are in the link language of the partition $\mathcal{A}_i$ and its parent $\mathcal{A}_j$ (ie. in the language $l(i, j)$) are transmitted as *messages* to $\mathcal{A}_j$. Partition $\mathcal{A}_j$ then asserts the message as an axiom of its theory, performs Consequence Finding and so on. When the algorithm reaches A_k it attempts to prove the query Q and returns the result. The following recounts a soundness and completeness result of [3, 2] for partitioned reasoning with Message-Passing on trees. We use this result to show that COMPACT-PROP is correct.

[2] If we remove (i, j) from the graph, then each of $\mathcal{B}_1, B_2$ is the union of the partitions in connected a connected component of the graph.

Definition 5 (Completeness for Consequence Finding) *Given a set of formulae $\mathcal{A}$ and a reasoning procedure $\mathcal{R}$, $\mathcal{R}$ is complete for consequence finding iff for every clause φ, that is a non-tautologous logical consequence of $\mathcal{A}$, $\mathcal{R}$ derives a clause ψ from $\mathcal{A}$ such that ψ subsumes φ.*

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

Theorem 3 ([3]) *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned theory and assume that the graph G is a tree that has a proper labeling for the partitioning $\{\mathcal{A}_i\}_{i \leq n}$. Also assume that each of the reasoning procedures used in FORWARD-MP is complete for consequence finding (as defined in Definition 5). Let $k \leq n$ and let $Q \in \mathcal{L}(\mathcal{A}_k \cup \bigcup_{(k,i) \in E} l(k, i))$ be a sentence. If $\mathcal{A} \models Q$, then FORWARD-MP outputs YES.*

The proof of this theorem uses Craig's Interpolation theorem [4] (stated below), which guarantees that FORWARD-MP transmits between partitions exactly those messages that are necessary for completeness.

Theorem 4 (Craig's Interpolation Theorem) *If $\alpha \vdash \beta$, then there is a formula $\gamma \in \mathcal{L}(L(\alpha) \cap L(\beta))$ such that $\alpha \vdash \gamma$ and $\gamma \vdash \beta$.*

Method 1 corresponds to running FORWARD-MP on the partitioned theory returned by PART-PROP($\{\mathcal{A}_i\}$). Method 2 is running a SAT solver on $\bigcup_{i=1}^{n}$ PART-PROP($\{\mathcal{A}_i\}$). The next theorem proves the soundness and completeness of these methods:

Theorem 5 *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned monadic FOL theory with G a properly labeled tree. Let $k \leq n$ and Q a sentence in $\mathcal{L}(\mathcal{A}_k)$. Then, $\mathcal{A} \models Q$ iff PART-PROP($\mathcal{A} \cup \{\neg Q\}) \models FALSE$.*

The quality of the propositionalization obtained depends on how balanced the partitions are, that is how evenly the predicates and constants are divided among the partitions (as will be shown in the next section). Finding a balanced partitioning can be done with human guidance or automatically. Sometimes, we can reduce this problem to finding graph decompositions with minimum treewidth of the intersection graph $G(V, E, l)$. A good reference is [1].

The algorithms above give sound and complete propositionalizations without the DCA by using the $\mathcal{E}$-sets. Even if we are allowed to make the DCA for the entire problem, a partitioned propositionalization would still need the $\mathcal{E}$-sets for completeness. The reason is that, even though the theory as a whole is closed, each individual partition is not, as the constant representing an object could be in a different partition.

4.2 Analysis

We now compare briefly the efficiency of our methods to the standard techniques. We do so with two metrics: the number of propositional symbols created by propositionalization, and the running time of the resulting SAT procedure.

First, the number of propositions created by method 1 and 2 on a theory τ are exponentially less than the number of propositions created by a propositionalization of τ without the DCA. When the number of constants is large, we get an improvement even when compared to naive propositionalization with the DCA.

Theorem 6 *Let τ be a monadic FOL theory with $L_{pred}(\tau) = P, L_{const}(\tau) = C$. Let $\mathcal{A}$ be a partitioning of τ into $\mathcal{A}_1, \mathcal{A}_3 \ldots \mathcal{A}_n$ such that $L_{pred}(\mathcal{A}_i) = O(|P|/n)$ and $L_{const}(\mathcal{A}_i) = O(|C|/n)$. If the number of propositions created in $\mathcal{P}(\tau) \cup \mathcal{E}(\tau)$ is $N(\tau)$, created by a naive propositionalization of τ with the DCA is $N_{DCA}(\tau)$, and by Method 2 applied to $\mathcal{A}$ is $N(\mathcal{A})$, then*

1. $$\frac{N(\tau)}{N(\mathcal{A})} = \Omega(1/n \cdot 3^{(1-1/n)|P|})$$

2. *If* $|C| \geq \frac{n^2 3^{|P|/n}}{|P|}$,
$$\frac{N_{DCA}(\tau)}{N(\mathcal{A})} = \Omega(n)$$

Now we examine the running time of our inference procedure compared to a naive propositionalization without the DCA.

Theorem 7 *Let τ be a monadic FOL theory with $L_{pred}(\tau) = P, L_{const}(\tau) = C$. Let $\mathcal{A}$ be a partitioning of τ into $\mathcal{A}_1, \mathcal{A}_2 \ldots \mathcal{A}_n$ (with intersection graph $G = (V, E, l)$) such that $L_{pred}(\mathcal{A}_i) = O(|P|/n)$ and $L_{const}(\mathcal{A}_i) = O(|C|/n)$. Let $d(v)$ be the degree of node v in the graph G, let $d = max_{v \in V} d(v)$ and let $l = max_{i,j \leq n} |l(i.j)|$. Finally, assume $|C| \geq \frac{n^2 3^{|P|/n}}{|P|}$. If $T(\tau)$ is the running time complexity of performing inference on τ with the DCA, and $T(\mathcal{A})$ is the running time of FORWARD-MP on a propositionalization of $\mathcal{A}$ created by PART-PROP, then*

$$\frac{T(\mathcal{A})}{T(\tau)} = O\left(\frac{n \cdot 2^{2dl} \cdot f_{SAT}(\frac{|P||C|}{n^2})}{f_{SAT}(|P||C|)}\right)$$

where $f_{SAT}(n)$ is the time taken to solve SAT problems over n variables.

The proof of Theorem 7 relies on the running-time analysis of FORWARD-MP given in [3]. Since f_{SAT} is typically exponential in the number of propositional symbols, the fraction $\frac{T(\mathcal{A})}{T(\tau)}$ will be small.

4.3 The Factory Example Revisited

Recall the machine scheduling problem from Section 2.2. Here we show an efficient solution of this problem by our methods.

The partioned theory is given in Figure 4.

Essentially, we create a partition $\mathcal{A}_i$ for every time step i, and place the following (propositional) axioms in it (this is the propositionalization of Equations 1 and 2):

$$\neg E_{\langle Q_i^1, P_i, \neg P_{i+1} \rangle}$$
$$\neg E_{\langle Q_i^2, P_i, \neg P_{i+1} \rangle}$$

We use the fact that $\forall x P(x) \equiv \neg \exists \neg P(x) \equiv \neg E_{\langle \neg P \rangle}$. Also in our example, if the system allows the item to move from state j to k at time step i, then we add

$$P_{ij} \Rightarrow P_{i+1_k}$$

to the propositionalization of Equation 3.

Similarly we propositionalize any constraints on the schedules of Q^1 and Q^2 that may exist and add them to the appropriate partition. Figure 4 gives an example of the final partitioned theory (before propositionalization).

Each partition has 4 predicates $P_i, P_{i+1}, Q_i^1, Q_i^2$ (sometimes we may have more predicates per partition because of additional constraints on the Q's, but for now we assume that such exceptions are not significant). The structural assumption we make with this partitioning is that only a limited set of constants (i.e., states), say at most 20, appears in each partition,

Using Method 2 for this propositionalization, we get a propositionalization with $n(3^4 + 4 \cdot 20)$ propositional symbols. Substituting the number of time steps to be ,say, 125 (from the example in Section 2.2) gives us 20,125 propositions, which is solvable using state-of-the-art SAT solvers. For comparison, Section 2.2 shows that a conventional propositionalization creates nearly *1 million* propositional symbols, which is beyond the capabilities of current SAT solvers.

5 Conclusions and Future Work

We have described algorithms that construct compact propositionalizations of function-free monadic First Order Logic Theories by exploiting structure in them. Our methods are quite general and result in significant savings in the number of propositional symbols required. They have applications to a number of domains that use logical reasoning such as Program Verification, Deductive Databases, Planning and Commonsense Reasoning.

In the near future we expect to extend this approach to reasoning with equality and binary predicates. We eventually intend to explore applications of our methods to Planning and Probabilistic Relational Models.

REFERENCES

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

[2] Eyal Amir and Sheila McIlraith, 'Partition-based logical reasoning', in *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conference (KR '2000)*, pp. 389–400. Morgan Kaufmann, (2000).

[3] Eyal Amir and Sheila McIlraith, 'Partition-based logical reasoning for first-order and propositional theories', *Artificial Intelligence*, (2004). Accepted for publication.

[4] William Craig, 'Linear reasoning. A new form of the Herbrand-Gentzen theorem', *Journal of Symbolic Logic*, **22**, 250–268, (1957).

[5] Rina Dechter and Judea Pearl, 'Tree clustering for constraint networks', *Artificial Intelligence*, **38**, 353–366, (1989).

[6] Burton Dreben and Warren D. Goldfarb, *The decision problem; solvable classes of quantificational formulas*, Addison-Wesley, 1979.

[7] S. Dzeroski and N. Lavrac, 'Learning relations from noisy examples: An empirical comparison of linus and foil', in *Proceedings of the 8th International Workshop on Machine Learning*, eds., L. Birnbaum and G. Collins, pp. 399–402. Morgan Kaufmann, (1991).

[8] P.C. Gilmore, 'A proof method for quantification theory: It's justification and realization', *IBM Journal of Research and Development*, **4**(1), 28–35, (January 1960).

[9] Henry Kautz and Bart Selman, 'Pushing the envelope: Planning, propositional logic, and stochastic search', in *Proc. National Conference on Artificial Intelligence (AAAI '96)*, (1996).

[10] Uffe Kjaerulff, *Aspects of efficiency imporvement in bayesian networks*, Ph.D. dissertation, Aalborg University, Department of Mathematics and Computer Science, Fredrik Bajers Vej 7E, DK-9220 Aalborg, Denmark, 1993.

[11] Stefan Kramer and Eibe Frank, 'Bottom-up propositionalization', in *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, eds., J. Cussens and A. Frisch, pp. 156–162, (2000).

[12] Nada Lavrac Mark-A. Krogel and Stefan Wrobel, 'Comparative evaluation of approaches to propositionalization', in *ILP*, pp. 197–214, (2003).

[13] John McCarthy and Patrick J. Hayes, 'Some philosophical problems from the standpoint of artificial intelligence', in *Machine Intelligence 4*, eds., B. Meltzer and D. Michie, 463–502, Edinburgh University Press, (1969).
[14] Karl Meinke and John V. Tucker, *Many Sorted Logic and its Applications*, Wiley Proffesional Computing, John Wiley & Sons, Chichester, New York, 1993.
[15] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik, 'Chaff: Engineering an Efficient SAT Solver', in *Proceedings of the 38th Design Automation Conference (DAC'01)*, (2001).
[16] Bart Selman, Henry A. Kautz, and Bram Cohen, 'Noise strategies for local search', in *Proc. 12th National Conference on Artificial Intelligence, AAAI'94, Seattle/WA, USA*, pp. 337–343, (1994).