Model-Theoretic Semantics and Tractable Algorithm for CNF-BCP*

Rahul Roy-Chowdhury and **Mukesh Dalal**
Columbia University
Department of Computer Science
New York, NY 10027
{royr,dalal}@cs.columbia.edu

Abstract

CNF-BCP is a well-known propositional reasoner that extends clausal Boolean Constraint Propagation (BCP) to non-clausal theories. Although BCP has efficient linear-time implementations, CNF-BCP requires clausal form transformation that sometimes leads to an exponential increase in the size of a theory. We present a new quadratic-time reasoner, RFP, that infers exactly the same literals as CNF-BCP. Although CNF-BCP has been specified only syntactically, we present a simple model-theoretic *semantics* for RFP. We also present a convergent term-rewriting system for RFP that is suitable for reasoning with knowledge bases that are built incrementally. Potential applications of RFP include logical truth-maintenance systems and general-purpose knowledge representation systems.

Introduction

Given a propositional theory, it is important in many AI systems to be able to determine the formulae that it logically entails. Since this reasoning problem is intractable, many systems settle for sound, incomplete, and tractable reasoners (Crawford 1992). One such reasoner, Clausal Boolean Constraint Propagation (BCP) (McAllester 1990), is widely used for incomplete linear-time reasoning with clausal propositional theories. However, all extensions of BCP to non-clausal theories that have been proposed so far are intractable (de Kleer 1990). One well-known extension of BCP is CNF-BCP, in which the non-clausal theories are first transformed into logically equivalent clausal theories. Although there are two approaches to this transformation, the standard one (Mendelson 1964) causes an exponential increase in the size of the theories in some cases, and the other (Cook 1971) requires adding new atoms that strongly inhibits reasoning by BCP.

*Copyright ©1997, American Association for Artificial Intelligence (www.aaai.org). All rights reserved. This work is partially supported by NSF Grant No. IRI-94-10117, NSF infrastructure grant CDA-9625374, and DARPA/ARL Contract No. DAAL01-94-K-0119.

For example, consider the non-clausal theory $\Gamma_1 = \{P \vee (P \wedge Q)\}$. The standard, potentially exponential CNF transformation of Γ_1 distributes $\vee$ over $\wedge$ to produce the clausal theory $\{P, P \vee Q\}$, from which BCP trivially infers P. The second method of CNF transformation of Γ_1 introduces a new atom, say R, for the subformula $P \wedge Q$ to produce the clausal theory $\{P \vee R, \neg R \vee P, \neg R \vee Q, \neg P \vee \neg Q \vee R\}$, from which BCP cannot infer any new formula. In general, no literals can be inferred from a disjunctive formula after using the second transformation. Thus, CNF-BCP reasoners that use the standard transformation infer more literals than those using the second transformation. For this reason, the standard transformation is the one used as the first step of CNF-BCP.

In this paper, we present a new quadratic-time propositional reasoner that is sound, incomplete, and does not require theories to be in clausal form. Since our reasoner is a restricted version of fact propagation (FP) (Dalal 1992; 1995), we call it *restricted fact propagation* (RFP). For clausal theories, RFP runs in linear time. We will prove that for any given propositional theory, not necessarily in clausal form, RFP and CNF-BCP infer exactly the same set of literals. Since we specify RFP using a model-theoretic semantics, we obtain the first model-theoretic semantics of inferences made using CNF-BCP. Rather than assigning *true* and *false* to atoms, we assign a real number between 0 and 1 to each atom. By extending such valuations to formulas and theories, we obtain notions of models and counter-models that are used to define RFP. Our model theory not only provides an independent characterization of reasoning, but is also conceptually simpler than the corresponing proof theoretic characterization. The importance of model-theoretic semantics for reasoners has been convincingly argued in several papers (c.f. (Levesque 1984)).

Apart from a quadratic time algorithm and a model-theoretic semantics, we also provide a rewrite system (Plaisted 1993) for RFP. A rewrite system is a collection of rewrite rules that indicate all the ways in which subformulas of a theory can be systematically replaced by possibly simpler formulas. We prove that the rewrite system for RFP is *convergent*; that is, any rewrite sequence from any theory terminates after a finite number of steps producing a unique irreducible form for that theory. We also prove RFP to be *modular*; that is, parts of a theory can be independently rewritten before rewriting the entire theory. We also present empirical results comparing the performance of RFP with some standard CNF-BCP reasoners.

Syntax and Semantics

In this section we define the syntax of our language, and specify a semantics for it. We extend the notion of the boolean-valued interpretations of classical logic, to real-valued *valuations*. We then specify the notion of models and counter-models of a valuation, and use this to define a notion of entailment.

We consider finite formulas and theories built using $\wedge$ and $\vee$ over a denumerable set $\mathcal{A} = \{P, Q, \ldots\}$ of atoms and their complements. In formal notation, $\wedge$ and $\vee$ are used as unary functors over finite bags of formulas, and the unary functor $\odot$ is used to construct theories. For readability, we will adopt the standard notation of propositional calculus (PC) for formulas in *negation normal form* (Mendelson 1964).

For example, the set $\{P, (\neg P \wedge Q) \vee \mathbf{f}\}$ is the informal notation for the theory $\Gamma_2 = \odot(P, \vee(\wedge(\neg P, Q), \vee()))$. The formulas $\wedge()$ and $\vee()$ are represented by the *logical constants* $\mathbf{t}$ and $\mathbf{f}$, respectively. We normally use variables p, q etc. to denote atoms, variables α, β etc. to denote literals, variables ψ, ϕ etc. to denote formulas, and variables Γ, Δ etc. to denote theories.

With each formula, we associate a set of sets of literals, called the support set, that captures the logical content of the formula (we normally use variables A, B, etc. to denote sets of literals, and variables X, Y, etc. to denote support sets):

Definition: The *support set*, $S(\psi)$, of a formula ψ is defined inductively as follows:

1. $S(\alpha) = \{\{\alpha\}\}$, for any literal α;
2. $S(\mathbf{t}) = \{\{\mathbf{t}\}\}$ and $S(\mathbf{f}) = \{\{\mathbf{f}\}\}$;
3. $S(\psi_1 \wedge \ldots \wedge \psi_n) = S(\psi_1) \cup \ldots \cup S(\psi_n)$;
4. $S(\psi_1 \vee \ldots \vee \psi_n) = \{A_1 \cup \ldots \cup A_n \mid A_i \in S(\psi_i)\}$;

for $n > 0$.

The support set, $S(\Gamma)$, of a theory Γ is defined to be the set $\cup\{S(\psi) \mid \psi \in \Gamma\}$. ■

The support set of a formula or theory is similar to its clausal normal form, except that duplicates are removed by using the set operations. For example, if $\Gamma_1 = \{P \vee (P \wedge Q)\}$, and $\Gamma_2 = \{P, (\neg P \wedge Q) \vee \mathbf{f}\}$ then $S(\Gamma_1) = \{\{P\}, \{P, Q\}\}$, and $S(\Gamma_2) = \{\{P\}, \{\neg P, \mathbf{f}\}, \{Q, \mathbf{f}\}\}$. Since support sets are part of the semantics and are never actually computed, the potentially exponential growth in their size does not cause computational problems.

Support sets are used for extending valuations on the set of atoms to the set of formulas. Intuitively, a valuation maps atoms to real numbers in the closed interval [0,1]. It is our counterpart of the notion of truth assignments in PC that map atoms to either 0 or 1.

Definition: A *valuation* is a function $\mathcal{A} \rightarrow [0, 1]$. That is, a valuation maps each atom to a real number in the interval $[0, 1]$. A valuation V is extended to the set of formulas as follows:

1. $V(\neg p) = 1 - V(p)$, for any atom p;
2. $V(\mathbf{t}) = 1$, and $V(\mathbf{f}) = 0$;
3. $V(\psi) = Min\{\sum\{V(\alpha) \mid \alpha \in A\} \mid A \in S(\psi)\}$, otherwise. ■

Intuitively, the valuation of a disjunction is the sum of the valuations of its disjuncts, while the valuation of a conjunction is the minimum valuation of its conjuncts, after removing all the duplicates. Although valuations of atoms and literals are real numbers in [0,1], valuations of other formulas may exceed 1. For example, a valuation that maps both P and Q to 1 maps $P \vee Q$ to 2, since $S(P \vee Q) = \{\{P, Q\}\}$. The same valuation maps $P \vee P$ to 1.

Definition: A valuation V is a *model* of a formula ψ iff $V(\psi) \geq 1$. A valuation V is a *counter-model* of a formula ψ iff $V(\psi) = 0$. A model of a theory is a model of each formula in the theory. ■

Unlike PC, where a truth assignment is either a model or a counter-model of a formula, it is possible that, given a valuation V and a formula ψ, V is neither a model nor a counter-model of ψ. This happens when $0 < V(\psi) < 1$, for example, in a valuation that maps P to 0.2 for the formula $\neg P$. This notion of models leads to an obvious notion of entailment:

Definition: For a theory Γ and a formula ψ: $\Gamma \models\!\!\approx \psi$ iff each model of Γ is a model of ψ. ■

Once again, consider the theories $\Gamma_1 = \{P \vee (P \wedge Q)\}$, and $\Gamma_2 = \{P, (\neg P \wedge Q) \vee \mathbf{f}\}$. Since any model of Γ_1 maps P to 1, it follows that $\Gamma_1 \models\!\!\approx P$. Since Γ_2 does not have any model (because Γ_2 requires mapping both P and $\neg P$ to 1), $\Gamma_2 \models\!\!\approx \psi$ for any formula ψ. Each truth assignment of PC corresponds to a valuation. If $\Gamma \models\!\!\approx \psi$ then we know that for any model V of Γ, $V(\psi) \geq 1$. In PC, which uses boolean valuations, this implies that $V(\psi) = 1$. Thus $\models\!\!\approx$ is sound with respect to $\models$.

RFP: A rewrite system

In this section we present a rewrite system (Plaisted 1993), RFP, that precisely captures the notion of entailment described above. In particular, we show that RFP is sound and complete with respect to $\models\!\!\approx$. We also prove that RFP possesses several desirable properties, namely termination, confluence and modularity.

A *rewrite system* is a collection of *rewrite rules*, each of which is a directed pair $l \Rightarrow r$ of terms that are either both formulas or both theories. Intuitively, a rewrite rule $l \Rightarrow r$ *rewrites* a term s by replacing its subterm l by r to obtain t; this is denoted by $s \Rightarrow_{\text{RFP}} t$. The reflexive-transitive closure of $\Rightarrow_{\text{RFP}}$ is denoted by

Simplification Rules:

$S1_\wedge$	$\wedge(\mathbf{f}, B) \Rightarrow \wedge(\mathbf{f})$	$S2_\wedge$	$\wedge(\mathbf{t}, B) \Rightarrow \wedge(B)$
$S1_\vee$	$\vee(\mathbf{t}, B) \Rightarrow \vee(\mathbf{t})$	$S2_\vee$	$\vee(\mathbf{f}, B) \Rightarrow \vee(B)$
$S1_\odot$	$\odot(\mathbf{f}, B) \Rightarrow \odot(\mathbf{f})$	$S2_\odot$	$\odot(\mathbf{t}, B) \Rightarrow \odot(B)$
$S3_\wedge$	$\wedge(\psi) \Rightarrow \psi$	$S3_\vee$	$\vee(\psi) \Rightarrow \psi$

Propagation Rules:

$$P1_\vee \quad \vee(\alpha, B) \Rightarrow \vee(\alpha, B[\mathbf{f} \overset{*\sim}{\leftarrow} \alpha])$$

$$P1_\odot \quad \odot(\alpha, B) \Rightarrow \odot(\alpha, B[\mathbf{t} \overset{*\sim}{\leftarrow} \alpha])$$

Lifting Rules:

$$L1_\wedge \quad \wedge(\wedge(\alpha, B_1), B_2) \Rightarrow \wedge(\alpha, \wedge(B_1), B_2)$$

$$L1_\odot \quad \odot(\wedge(\alpha, B_1), B_2) \Rightarrow \odot(\alpha, \wedge(B_1), B_2)$$

$$L2_\wedge \quad \wedge(\alpha_1, \ldots, \alpha_n, \wedge(B)) \Rightarrow \wedge(\alpha_1, \ldots, \alpha_n, B)$$

Factoring Rule $F1_\vee$:

$$\vee(\alpha_1, \ldots, \alpha_n, \wedge(\alpha, B_0), \ldots, \wedge(\alpha, B_m)) \Rightarrow \vee(\alpha_1, \ldots, \alpha_n, \wedge(\alpha, \vee(\wedge(B_0), \ldots, \wedge(B_m))))$$

Figure 1: Rewrite system RFP

$\Rightarrow^*_{\mathrm{RFP}}$ and the equivalence closure of $\Rightarrow_{\mathrm{RFP}}$ is denoted by $\Leftrightarrow^*_{RFP}$. If a term can not be rewritten by any rule in RFP, then it is said to be *irreducible*. A term s *reduces* to a term t, denoted by $s \Rightarrow^!_{\mathrm{RFP}} t$, if s rewrites to t (using 0 or more rewrite steps) and t is irreducible.

We will represent groups of rewrite rules by rule schemas that use variables to denote formulas. The rules of RFP are given by the rule schemas in Figure 1, where $n, m \in \mathcal{N}$, α's are literals, ψ is a formula, B's are bags of formulas, and $B[\mathbf{t} \overset{*\sim}{\leftarrow} \alpha]$ is obtained from the bag B of formulas by replacing each occurrence of the literal α by $\mathbf{t}$ and $\sim\alpha$ by $\mathbf{f}$. To ensure termination, that is, each sequence of rewrite steps is finite, B can't be an empty bag in $S1$ rules, the atom of α must be a subterm of B in $P1$ rules, and $m \geq 1$ in $F1$ rule.

The simplification rules simplify terms containing $\mathbf{t}$, $\mathbf{f}$, and redundant connectives. The propagation rules propagate a literal α by replacing each occurrence of α and $\sim\alpha$ in B by some logical constant. The lifting rules remove a redundant nesting of connectives. The factoring rule identifies common literals in a disjunction and factors them out.

For example, the following rewrite sequence terminates after producing the irreducible theory $\{\mathbf{f}\}$ from the theory Γ_2:

$\{P, ((\neg P \wedge Q) \vee \mathbf{f})\}$	
$\Rightarrow_{\mathrm{RFP}} \{P, ((\neg P \wedge Q))\}$	(Rule $S2_\vee$)
$\Rightarrow_{\mathrm{RFP}} \{P, (\neg P \wedge Q)\}$	(Rule $S3_\vee$)
$\Rightarrow_{\mathrm{RFP}} \{P, (\mathbf{f} \wedge Q)\}$	(Rule $P1_\odot$)
$\Rightarrow_{\mathrm{RFP}} \{P, (\mathbf{f})\}$	(Rule $S1_\wedge$)
$\Rightarrow_{\mathrm{RFP}} \{P, \mathbf{f}\}$	(Rule $S3_\wedge$)
$\Rightarrow_{\mathrm{RFP}} \{\mathbf{f}\}$	(Rule $S1_\odot$)

Theorem 1 shows some important properties of the rewrite system RFP.

Theorem 1 *For any terms s and t, and any bags B, B_1, B_2 of terms:*

1. *(Termination) there is no infinite chain $t_1 \Rightarrow_{\mathrm{RFP}} t_2 \Rightarrow_{\mathrm{RFP}} \ldots$ of terms;*
2. *(Confluence) if $s \Leftrightarrow^*_{RFP} t$ then there is a term v for which $s \Rightarrow^*_{\mathrm{RFP}} v$ and $t \Rightarrow^*_{\mathrm{RFP}} v$;*
3. *(Modularity) if $\odot(B_1) \Leftrightarrow^*_{RFP} \odot(B_2)$ then $\odot(B, B_1) \Leftrightarrow^*_{RFP} \odot(B, B_2)$;*

The proof can be found in (Dalal 1995). The termination of RFP guarantees that any rewrite sequence is finite. It then follows from confluence that RFP is convergent; that is, each rewrite sequence starting from a theory produces a unique irreducible form for that theory. The modularity of RFP allows independently rewriting parts of a theory before rewriting the entire theory.

The rest of this section is devoted to proving the soundness and completeness of RFP with respect to the semantics we defined earlier. We will first show that *irreducible* theories have some important characteristics. In particular, theorem 2 shows that a literal α can be in the support set of an irreducible theory Δ iff it occurs as a formula in the theory. It follows that all models of Δ are models of α, which implies that $\Delta \models\!\approx \alpha$. We then look at the effect that rewrite rule applications have on the support set of a theory. It can be shown (Lemma 3) that in fact applying rewrite rules to a theory has *no* effect on the models of that theory. If we had a theory Γ such that $\Gamma \Rightarrow^*_{\mathrm{RFP}} \Delta$, then Γ and Δ have exactly the same models. In particular, if Δ is irreducible and $\Delta \models\!\approx \alpha$, it follows that $\Gamma \models\!\approx \alpha$.

Theorem 2 shows that a literal can be inferred from an irreducible theory (different from $\{\mathbf{f}\}$) using $\models\!\approx$ iff the literal is a formula in the theory iff the singleton set containing the literal is in the support set of the theory:

Theorem 2 *For any irreducible theory Γ and any literal α, the following are equivalent:*

1. *α is a formula in Γ;*
2. *$\{\alpha\} \in S(\Gamma)$;*
3. *$\Gamma \neq \{\mathbf{f}\}$ and $\Gamma \models\!\approx \alpha$.*

The first two statements are proved equivalent by first showing that for any literal α and any irreducible formula ψ, $\{\alpha\} \in S(\psi)$ iff $\psi = \alpha \wedge \phi$, for some formula ϕ. The last two statements are proved equivalent by showing that $S(\Gamma)$ for any irreducible Γ different from $\{\mathbf{f}\}$ can be split into two parts, S_1 containing singleton sets with disjoint atoms, and S_2 containing the rest, such that they contain disjoint atoms. Each model of Γ must map each literal in S_1 to 1, and the valuation that also maps each atom in S_2 to 0.5 is a model of Γ.

Lemma 3 *If Γ, Δ are theories such that $\Gamma \Rightarrow_{\mathrm{RFP}} \Delta$ then Γ and Δ have identical models.*

This result can be proved by considering the effect of each rewrite rule on support sets and showing that no rule application changes the models of the theory.

The following corollary of Lemma 3, which follows directly from Theorem 2, shows that RFP is sound and complete for inferring literals using $\approx\!\!|$:

Corollary 4 *For any theory* Γ *and any literal* α, $\Gamma \models_{\approx} \alpha$ *iff either* $\Delta = \{\mathbf{f}\}$ *or* α *is a formula in* Δ, *where* $\Gamma \Rightarrow^{!}_{\text{RFP}} \Delta$.

Corollary 5 relates the rewrite system RFP to the the notion of models in our semantics:

Corollary 5 *A theory has a model iff it is not reducible to* $\{\mathbf{f}\}$.

Comparing RFP and CNF-BCP

In this section, we will show that RFP and CNF-BCP infer exactly the same literals from any given theory. We first define CNF-BCP and prove some properties. Since CNF-BCP operates by applying BCP to the CNF transformation of a non-clausal theory, we define CNF transformation and present a rewrite system for BCP.

Using support sets, we first present a simple definition of CNF transformation:

Definition: For any theory Γ, its CNF transformation CNF(Γ) is defined to be the theory $\{\vee(B) \mid B \in S(\Gamma)\}$.

For example, CNF($\{P\vee(\neg P\wedge Q)\}$) = $\{(P\vee\neg P), (P\vee Q)\}$.

BCP is a variant of unit resolution (Chang & Lee 1973). Given any clausal theory Γ, BCP monotonically expands it by adding facts as follows: in each step, if any *single* clause in Γ and *all the facts* in Γ taken together *logically entail* any other fact, then the new fact is added to the theory Γ. This step is repeated until no new fact can be so obtained. For example, starting with the theory $\{(\neg P), (P\vee\neg Q), (P\vee\neg R), (Q\vee R)\}$, BCP adds the following sequence of facts: $(\neg P), (\neg Q), (\neg R), \mathbf{f}$, thereby determining that the theory is unsatisfiable.

The rewrite system, CBCP, given below provides a characterization of BCP:

$$\begin{array}{rrcl} 1. & \odot(\mathbf{f}, B) & \Rightarrow & \odot(\mathbf{f}) \\ 2. & \odot(\vee(\alpha), \vee(\sim\alpha, B_1), B_2) & \Rightarrow & \odot(\vee(\alpha), \vee(B_1), B_2) \end{array}$$

where α is a literal and B's are bags of formulas; B must be non-empty. Rule 1 ensures that the theory reduces to $\mathbf{f}$ after any empty clause is obtained. In addition to encoding the unit resolution step, Rule 2 also explicitly specifies that the input clause, which is subsumed by the output, should be removed from the theory. It turns out that the rewrite system CBCP is convergent, content preserving, monotonic, modular, and tractable. In particular, the linear time algorithm presented in (McAllester 1990) reduces any theory with respect to CBCP.

Lemma 6 shows that rewriting using CBCP does not change models of a clausal theory.

Lemma 6 *If* $\Gamma \Rightarrow_{\text{CBCP}} \Delta$, *then* Γ *and* Δ *have identical models.*

Proof: We consider each of the rewrite rules in CBCP: If rule 1 is used, then $S(CNF(\Gamma)) = \{\{f\}\} \cup S(B)$ and $S(\Delta) = \{\{f\}\}$. Since neither of them have models, the claim is vacuously true.

If rule 2 is used, then $S(CNF(\Gamma)) = \{\{\alpha\}\}\cup\{\alpha, \neg\alpha \mid \alpha \in S(B_1)\}\cup S(B_2)$ and $S(\Delta) = \{\{\alpha\}\}\cup S(B_1)\cup S(B_2)$. The result follows from the fact that if V is a model of $CNF(\Gamma)$ or Δ, then $V(\alpha) = 1$, which means that $V(\neg\alpha) = 0$. ■

Theorem 7 is an analogue of Theorem 2 for CNF-BCP.

Theorem 7 *If a theory* Γ *is irreducible with respect to CNF-BCP, and* α *is any literal, then the following are equivalent:*

1. α *is a formula in* Γ
2. $\{\alpha\} \in S(\Gamma)$
3. $\Gamma \neq \{f\}$ *and* $\Gamma \models_{\approx} \alpha$.

Proof:

($1 \Rightarrow 2$) Follows from the definition of $S(\Gamma)$.

($2 \Rightarrow 1$) Since $\{\alpha\} \in S(\Gamma)$, $\alpha \in S(\psi)$ for some $\psi \in \Gamma$. Since ψ is a clause, it follows that $\psi = \alpha$; hence $\alpha \in \Gamma$.

($2 \Rightarrow 3$) If an interpretation V is a model of Γ, then $\Sigma_{\alpha\in X} V(\alpha) \geq 1$ for every $X \in S(\Gamma)$. In particular, $V(\alpha) = 1$ and so $\Gamma \models_{\approx} \alpha$.

($3 \Rightarrow 2$) Let all the singleton sets of $S(\Gamma)$ be grouped together in the set S_1. Let V be a valuation that maps each literal in S_1 to 1, and all other literals to 0.5. It can be shown that V is then a model of Γ. Since $\Gamma \models_{\approx} \alpha$, it follows that $V(\alpha) = 1$. Thus, $\{\alpha\} \in S_1$ and so $\{\alpha\} \in S(\Gamma)$. ■

The following corollary of theorem 7, which is based on the equivalence of models established by Lemma 6, shows that CNF-BCP is sound and complete for inferring literals using $\models_{\approx}$.

Corollary 8 *For any theory* Γ *and any literal* α, $\Gamma \models_{\approx} \alpha$ *iff either* $\Delta = \{f\}$ *or* α *is a formula in* Δ, *where* $\Gamma \Rightarrow^{!}_{\text{CBCP}} \Delta$.

Corollary 9, which follows from corollary 4 and corollary 8, shows that RFP and CNF-BCP infer exactly the same set of literals from any theory Γ.

Corollary 9 *For any theories* Γ, Δ, *and* Δ', *if* $\Gamma \Rightarrow^{!}_{\text{RFP}} \Delta$ *and* CNF(Γ) $\Rightarrow^{!}_{\text{CBCP}} \Delta'$, *then*

1. $\Delta = \{\mathbf{f}\}$ *iff* $\Delta' = \{\mathbf{f}\}$, *and*
2. *for any literal* α, $\alpha \in \Delta$ *iff* $\alpha \in \Delta'$.

A Tractable Algorithm

We sketch an algorithm, ARFP, for computing the irreducible form of a theory using the rewrite system RFP. It takes a non-clausal theory as input and returns the irreducible form of the theory as output. It works by repeatedly rewriting the theory using some rewrite rule

```
Procedure ARFP:
  1. Root := ReadArgs(⊙); /*read theory,build tree*/
  2. InitPropagate; /* set Occurs and PQ */
  3. L1Q := L2Q : = Nil;
  4. InitFactor; /* set FQ */
  5. loop {
  6. if PQ ≠ Nil then { /* possible P rule */
  7.    Propagate(Head(PQ));
  8.    AddQ(Pop(PQ),L1Q);
  9. } elseif L1Q ≠ Nil then { /* possible L1 rule */
 10.    Lift1(Head(L1Q));
 11.    AddQ(Pop(L1Q),L2Q);
 12. } elseif L2Q ≠ Nil then { /* possible L2 rule */
 13.    Lift2(Head(L2Q));
 14.    Pop(L2Q);
 15. } elseif FQ ≠ Nil then { /* possible F1 rule */
 16.    Factor(Head(FQ));
 17.    Pop(FQ);
 18. } else exit; /* got an irreducible form */
 19. }
end (ARFP).
```

Figure 2: The ARFP Algorithm

that is applicable to it. Rather than naively searching for an applicable rule in each step, which does not give an efficient algorithm, even for clausal theories, ARFP uses several refinements like lazy evaluation, queues, and pre-processing.

Algorithm ARFP uses the tree representation of a theory, where leaves are labeled by literals and internal nodes, which represent subformulas, are labeled by connectives. For each internal node N, each element N.Occurs[P] of an array called N.Occurs, indexed by literals, keeps a list of those nodes that provide access to all leaves in the subtree rooted at N which are labeled by the literal P. *Occurs* arrays are used in efficiently applying propagation rules.

ARFP reads the input theory while applying all possible simplification rules and constructing its tree representation (ReadArgs), initializes the data structures for propagation and lifting rules (InitPropagate), initializes the data structures for factoring (InitFactor), and then applies all possible rules that become applicable. Most of the rewriting of the input theory is done within the main loop of the algorithm by inspecting the various queues of possible rule applications. In each iteration, if PQ (the queue of possible propagation rule applications) is non-empty then propagation rules are attempted (Propagate) at its first node; otherwise, if L1Q is non-empty then L1 rules are attempted (Lift1) at its first node; otherwise, if L2Q is non-empty then the L2 rule is attempted (Lift2) at its first node; otherwise, if FQ is non-empty then the F1 rule is attempted (Factor) at its first node. The algorithm terminates when either all the queues are empty or the theory reduces to $\{\mathbf{f}\}$ or $\{\ \}$.

Theorem 10 *For any theory Γ, any execution of ARFP(Γ) takes $O(n^2)$ time, where n is the size of Γ, and returns a theory Δ such that $\Gamma \Rightarrow^{!}_{\mathrm{RFP}} \Delta$. If Γ is clausal, any execution of ARFP(Γ) takes time $O(n)$.*

```
Procedure RandomFormula(V,AW,OW,Depth,P)
  1. $P = P^{Depth}$ /*To taper off the nesting depth*/
  2. X = Random(0,1)
  3. if ($X < P$) then { /*The formula has a connective*/
  4. Connective = Randomly pick one of {∨, ∧}
  5. if (Connective=∧) then NumArgs=Random(1,AW)
  6. else NumArgs = Random(1,OW)
  7. for i = 1 to NumArgs
  8.    RandomFormula(V,AW,OW,Depth+1,P)
  9. } else /* The current formula is a literal */
 10.    Formula = pick variable at random
end (RandomFormula).
```

Figure 3: Generating a random formula

Empirical Results

We ran experiments using six different non-clausal reasoners: FP, RFP, and four variants of CNF-BCP. In this section we first discuss the method we used for random theory generation, and then look at the results obtained by running these algorithms on randomly generated theories.

Random Theory Generation

We considered five parameters in generating random theories:

1. *NF:* number of formulas in the theory;
2. *V:* set of variables to choose from;
3. *AW:* max. number of arguments in a conjunction;
4. *OW:* max. number of arguments in a disjunction;
5. *P:* probability that formula will have depth > 1.

The formula generation procedure is given in figure 3. Initially, each formula in the theory is generated by a call to $RandomFormula(V, AW, OW, 1, P)$. The initial call to $RandomFormula$ generates formulas at $Depth = 1$. Each recursive call increases $Depth$ by 1 and recalculates $Prob$ as $Prob = Prob^{Depth}$. This ensures that as we get deeper nesting, the probability of generating atomic formulas increases.

Our technique provides a systematic way to generate random theories with uniformly varying structures. For example, a high value for $Prob$ will result in a more deeply nested theory while a large value for $AndWidth$ or $OrWidth$ will result in wider conjuncts and disjuncts, respectively. This method also results in total coverage of the space of all non-clausal theories, and thus we feel it is a useful generation mechanism.

The Algorithms

We ran experiments to verify that non-clausal reasoning with either Fact Propagation (FP) or Restricted Fact Propagation (RFP) is indeed more efficient than doing a clausal transformation and then using BCP.

We considered four different clausal transformations, followed by BCP. In addition to FP and RFP, these algorithms are:

CNF-BCP Usual exponential transformation into clausal form (Mendelson 1964) followed by BCP.

DUP-BCP This is similar to CNF-BCP, except that duplicate literals in a clause are removed before applying BCP.

CLAUSE-BCP This is similar to DUP-BCP except that duplicate clauses are also removed.

POLY-BCP Polynomial transformation into clausal form followed by BCP (Cook 1971). This requires the addition of new literals to the theory.

We considered three methods to measure the efficiency of the reasoning algorithms. The first of these is simply the execution time of the algorithms. The second, which can be considered the *deductive power* of a reasoner, is the number of literals inferred. The third, the *reductive power* of a reasoner is the size of the irreducible output theory as a proportion of the original input theory. In order to measure these values, we performed experiments on 7435 random theories (using a wide range of input parameters). The results of the experiments are summarized in the table below.

algorithm	avg(time in msecs)	avg(literals inferred)	avg(out_size/ in_size)
RFP	56	107.678	0.128
FP	60	107.679	0.124
CNF-BCP	1905	107.678	25.779
DUP-BCP	2472	107.678	13.957
CLAUSE-BCP	2381	106.315	4.128
POLY-BCP	1193	59.737	1.915

As the table indicates, both FP and RFP are significantly faster than any of the clausal transformations followed by BCP. Further, although FP was shown to be stronger than RFP, it appears that on average they infer almost the name number of literals. Since RFP and CNF-BCP have equal deductive power, they infer exactly the same number of literals.

The average number of literals inferred by POLY-BCP was actually 106.353. However, this figure includes literals that were introduced by the algorithm itself and hence not truly indicative of deductive power. To compensate for this, we projected the number of literals inferred onto the set of original literals.

Further, if we consider the size of the resultant theory as a proportion of the original theory, RFP and FP again performed well. Both algorithms reduced theories to approximately 10% of their original size. All the clausal transformations resulted in theories that were an order of magnitude larger; they were even larger than the original theories. This is a consequence of the large size of the clausal form transformation of the theories. Somewhat surprisingly, the polynomial transformation resulted in sizes that were about half the sizes produced by the shortest exponential transformation.

Related Work and Conclusions

For clausal theories, BCP (McAllester 1980) is the hyper-resolution variant of unit resolution (Chang & Lee 1973), that has been variously called unit-resulting resolution, forward pruning, and DPL-oracle. Several efficient implementations of BCP have been proposed (McAllester 1990). Several extensions of BCP to non-clausal theories have been defined syntactically in (de Kleer 1990): Formula BCP, Prime BCP, and CNF-BCP. While the first two are intractable (unless P = NP), no tractable algorithm for CNF-BCP was previously known. We have proved that our quadratic-time reasoner RFP, specified using a model-theoretic semantics as well as a convergent and modular rewrite system, infers exactly the same literals that are inferred by CNF-BCP.

Although (Dalal 1992; 1995) uses a rewrite system to specify a reasoner called FP, no model-theoretic semantics is known for FP. Moreover, although FP sometimes infers *more* literals than either CNF-BCP or RFP, the empirical results indicate that the difference is negligible for randomly generated theories.

References

Chang, C., and Lee, R. 1973. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, London.

Cook, S. 1971. The complexity of theorem proving procedures. In *Proceedings Third Annual ACM Symposium on the Theory of Computing*, 151–158.

Crawford, J., ed. 1992. *Proceedings of the AAAI Workshop on Tractable Reasoning.* San Jose, California: American Association for Artificial Intelligence.

Dalal, M. 1992. Efficient propositional constraint propagation. In *Proceedings Tenth National Conference on Artificial Intelligence (AAAI-92)*, 409–414. San Jose, California: American Association for Artificial Intelligence.

Dalal, M. 1995. Tractable reasoning in knowledge representation systems. Technical Report CUCS-017-95, Department of Computer Science, Columbia University, New York, NY.

de Kleer, J. 1990. Exploiting locality in a TMS. In *Proceedings Eighth National Conference on Artificial Intelligence (AAAI-90)*, 264–271.

Levesque, H. 1984. A logic of implicit and explicit belief. *Proceedings National Conference on Artificial Intelligence (AAAI-84)* 198–202.

McAllester, D. 1980. An outlook on truth maintenance. Memo 551, MIT AI Lab.

McAllester, D. 1990. Truth maintenance. In *Proceedings Eighth National Conference on Artificial Intelligence (AAAI-90)*, 1109–1116.

Mendelson, E. 1964. *Introduction to Mathematical Logic.* Princeton, N.J.: Van Nostrand.

Plaisted, D. 1993. Equational reasoning and term rewriting systems. In Gabbay, D., and Siekmann, J., eds., *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1. Oxford University Press.