Journal of Automated Reasoning **28:** 371–396, 2002.
© 2002 Kluwer Academic Publishers. Printed in the Netherlands.

Partial Instantiation Methods for Inference in First-Order Logic

J. N. HOOKER[*]
Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.

G. RAGO
Computer Science Department, University of Pisa, Corso Italia 40, 156125 Pisa, Italy

V. CHANDRU
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India 560 012

A. SHRIVASTAVA
Sanchez Computer Associates, 37/2, 4th Main, Malleswaram, Bangalore, India 560 003

(Received: 21 January 1997; in final form: 3 September 2001)

Abstract. Satisfiability algorithms for propositional logic have improved enormously in recently years. This improvement increases the attractiveness of satisfiability methods for first-order logic that reduce the problem to a series of ground-level satisfiability problems. R. Jeroslow introduced a partial instantiation method of this kind that differs radically from the standard resolution-based methods. This paper lays the theoretical groundwork for an extension of his method that is general enough and efficient enough for general logic programming with indefinite clauses. In particular we improve Jeroslow's approach by (1) extending it to logic with functions, (2) accelerating it through the use of satisfiers, as introduced by Gallo and Rago, and (3) simplifying it to obtain further speedup. We provide a similar development for a "dual" partial instantiation approach defined by Hooker and suggest a primal–dual strategy. We prove correctness of the primal and dual algorithms for full first-order logic with functions, as well as termination on unsatisfiable formulas. We also report some preliminary computational results.

Key words: satisfiability algorithms, primal–dual strategy.

1. Introduction

The past several years have seen remarkable improvements in the computational performance of satisfiability algorithms for propositional logic. They lend new attractiveness to methods that solve satisfiability problems in first-order logic by first reducing them to propositional satisfiability problems.

A naive approach is to try to generate all complete instantiations of a first-order formula and to use a fast satisfiability algorithm to find a truth valuation that

[*] Partial support by U.S. Office of Naval Research Grant N00014-95-1-0517 is gratefully acknowledged.

satisfies the resulting ground-level formulas. But the number of instantiations can be astronomically large or even infinite.

R. Jeroslow [20, 21] addressed this problem with a *partial instantiation* (PI) approach. It solves a series of propositional satisfiability problems, each obtained by instantiating one or more of the variables in the last. With luck, the first-order satisfiability question is resolved when only a few of the possible instantiations have been generated. Gallo and Rago [12, 13] and Hooker [16] proposed a "dual" version of PI.

Both Jeroslow's "primal" approach and the dual approach apply to the ∃∀ or Schönfinkel–Bernays fragment of first-order logic without function symbols. Jeroslow indicates that PI can be extended to undecidable fragments, but it may run indefinitely because the depth of Skolem function nesting is unbounded.

Our purpose in this paper is twofold:

– To extend the PI method to full first-order logic with function symbols, so as to make it more useful for theorem proving and logic programming (with both definite and indefinite clauses).
– To simplify the method by removing Jeroslow's mechanism of "direct covers" and accelerate it by using a restricted form of "blockage" between satisfiers, as defined by Gallo and Rago [13].

Our eventual goal is to develop a practical inference algorithm for non-Horn first-order logic with functions. In this paper we lay some of the theoretical foundations by stating basic primal and dual PI algorithms for inference and proving correctness, and termination when applied to unsatisfiable formulas. We also report some preliminary computational results in order to address implementation issues.

The motivating idea of a primal PI algorithm is similar to that of "row generation" techniques in optimization. Suppose one wants to find a solution that satisfies an impracticably large set of constraints. One can first solve the problem using a small subset of the constraints and check whether the solution satisfies the other constraints. If not, one or more of the violated constraints are added to the problem, which is then re-solved. It is not unusual to find a solution when only a small fraction of the constraints have been explicitly considered.

In the first-order satisfiability problem, the "constraints" are the formula's various instantiations that the truth valuation must satisfy. In each iteration of a PI method, a truth valuation is found to satisfy the formulas that result from instantiating only a few universally quantified variables. If the valuation can be extended in a natural way to one that satisfies all partial and complete instantiations of the formula, the problem is solved. Otherwise the valuation is said to be "blocked," and the algorithm instantiates a few more variables to generate formulas that are not satisfied. This presents a new propositional satisfiability problem, and the process is repeated. The algorithm terminates if the current set of partially instantiated formulas is unsatisfiable (in which case the original formula is unsatisfiable), or if it has an unblocked satisfying valuation exists (in which case the formula is satisfiable).

The "dual" approach to PI (as defined by [16]) anticipates blockage by temporarily augmenting the formula with clauses that no blocked valuation can jointly satisfy. The additional clauses not only rule out blocked valuations but typically go too far and rule out all valuations. In an attempt to relieve this unsatisfiability, a few more variables are instantiated in the original formula, and the process is repeated. The algorithm terminates when the augmented formula is satisfiable or the partial instantiations of the original formula are unsatisfiable.

The name "partial instantiation" is somewhat misleading because PI is also used in other methods. Resolution methods, for instance, may only partially instantiate predicates when they unify atoms in the parents of a resolvent [28]. The difference in PI methods, as defined here, is that a propositional satisfiability problem is solved to completion after each stage of partial instantiation. This approach permits one to use the fastest available algorithm for propositional satisfiability. In particular, it permits one to avoid using resolution, which has been demonstrated to be orders of magnitude slower than other methods when used to check for propositional satisfiability [15]. Furthermore, when a new instantiation is added in each stage, the information obtained in solving previous satisfiability problems can be put to good use [3, 17]. This makes the fast satisfiability algorithms even faster.

2. Previous Work

After Jeroslow's original 1988 paper [20], Gallo and Rago [12, 13, 4] described a hypergraph-based satisfiability algorithm for "datalog" (universally quantified Horn) formulas that in effect uses PI. Hooker [16] showed that Jeroslow's method is actually a "primal" version of PI whereas the Gallo–Rago algorithm is a special case of a "dual" form. He also described the dual algorithm for general function-free first-order formulas. Kagan, Nerode, and Subrahmanian [22] described an algorithm similar to Jeroslow's primal algorithm in which his mechanism of direct and indirect covers is implemented in a tree of partial instantiations. Specifically, an instantiation is directly covered by its parent node in the tree and indirectly covered by other ancestors.

Another line of research that is related to PI and that has proceeded more or less independently began in 1960 with the resolution method of Davis and Putnam [11]. As already mentioned, resolution-based methods use a form of PI but are fundamentally different from those presented here. Davis [10] introduced in 1963 a clause linking technique that has influenced much subsequent work and is related to the idea of blockage defined below. Yet the purpose of clause linking in Davis's method is to avoid generating some resolvents that can play no role in the proof. The resulting algorithm remains a resolution-based method.

The hyperlinking method of Lee and Plaisted [24], on the other hand, bears some resemblance to the primal PI method described below. It repeatedly solves a propositional satisfiability problem to completion after a round of PI based on the detection of clause links. Our primal method similarly solves a satisfiability problem after partially instantiating clauses based on blockages. Our instantiations, however, are guided by semantic information, specifically by the truth values in the solution of the previous satisfiability problem. So they are more in the spirit of constraint generation methods in optimization.

The hyperlinking method can be enhanced by using semantic information, and Chu and Plaisted [7] do just this in their semantic hyperlinking method. It conducts a depth-first search of a tree that branches by assigning truth and falsehood to elements of the Herbrand base, with more deeply nested functions in the deeper parts of the tree. It therefore does not solve a sequence of satisfiability problems to completion, as our methods do. A similar observation may be made of the ordered semantic hyperlinking method of Plaisted and Zhu [27].

Baumgartner's method [2] "lifts" the classical Davis–Putnam–Logeman–Loveland branching method by branching on partially instantiated formulas rather than ground formulas. It resembles our method in that a partially instantiated formula stands for all of its complete instantiations, unless contradicted by a more specific partially instantiated formula. It does not, however, follow our approach of solving a series of ground-level satisfiability problems in which semantic information from the last solution guides the next. It generates a single branching tree in which the branching is designed to resolve what we call "blockage." In addition, it does not accommodate function symbols, nor does it use the concept of satisfiers.

A large literature exists on algorithms for the propositional satisfiability problem, much of which is surveyed in [5, 29]. More relevant for present purposes, however, are incremental algorithms: those that update a solution rapidly after a clause is added to the problem. Ausiello and Italiano [1] originally studied this problem for the case of Horn (definite) clauses. Hooker [17] presented a discrete branching algorithm for the general case, and it is used here. Bennaceur et al. [3] recently proposed what appears to be a faster algorithm based on integer programming and Lagrangian relaxation.

3. Preliminaries

We begin by defining our notation and stating the form of Herbrand's theorem we will use later.

DEFINITION 3.1. A *term* is recursively defined in the following way:

(1) Variables $x_1, x_2, \ldots$ and constants $a_1, a_2, \ldots$ are terms of (nesting) *depth* 0.
(2) If f is a function symbol and $t_1, t_2, \ldots, t_n$ are terms, then $f(t_1, t_2, \ldots, t_n)$ is a term of *depth* $1 + \max_i\{\mathit{depth}(t_i)\}$.

We assume that a first-order formula F is given in prenex clausal form,

$$F = \bigwedge_{i=1}^{m} \forall x_i C_i,$$

where x_i is a vector of all the variables appearing in clause C_i. The clauses are standardized apart, meaning that no two contain a common variable. Each clause C_i is quantifier free and can be written

$$\neg P_{i1}(t_{i1}) \vee \cdots \vee \neg P_{iq}(t_{iq}) \vee P_{i,q+1}(t_{i,q+1}) \vee \cdots \vee P_{ik}(t_{ik}) \tag{1}$$

or, in the equivalent form,

$$P_{i1}(t_{i1}) \wedge \cdots \wedge P_{iq}(t_{iq}) \rightarrow P_{i,q+1}(t_{i,q+1}) \vee \cdots \vee P_{ik}(t_{ik}),$$

where, for $h = 1, \dots, k$, $P_{ih}(t_{ih})$ is an atom and t_{ih} a vector of terms. It is well known that any first-order formula can be written in prenex clausal form (see, for instance, [26]).

A formula is said to be *ground* if it contains no variables. A *substitution* σ replaces one or more variables of a formula F with terms, in such a way that each occurrence of a given variable is replaced by the same term. The result of the substitution, written $F\sigma$, is an *instantiation* of F. It is a *ground instance* (*or complete instantiation*) *of* F if it is ground; otherwise it is a *partial instantiation.* A substitution is written $\sigma = \{x_1/t_1, \dots, x_k/t_k\}$ when it substitutes $t_1, \dots, t_k$ respectively for $x_1, \dots, x_k$. If $t_1, \dots, t_k$ are all variables, σ is a *renaming* substitution.

The Herbrand universe of a formula F consists of all the terms that can be built up from constants and function symbols in F.

DEFINITION 3.2. Given a formula F, let H_0 be the set of constants occurring in F. If no constants appear in F, then H_0 consists of a single element, say a. The ith *level constant set* H_i of F is the set of terms that have depth at most i and that are built up from constants in H_0 and function symbols in F. $\bigcup_{i=1}^{\infty} H_i$ is the *Herbrand universe* of F.

The Herbrand base of F is the set of all atoms obtained by using terms in the Herbrand universe.

DEFINITION 3.3. For a given formula F let

$$B_i = \{P(t_1, t_2, \dots, t_n) \mid t_1, \dots, t_n \in H_i \text{ and } P \text{ appears in } F\}.$$

Then $\bigcup_{i=1}^{\infty} B_i$ is the *Herbrand base* of F.

An *Herbrand interpretation* for F assigns a truth value to each atom in the Herbrand base of F. F is true in an Herbrand interpretation I if every ground instance of F using terms in the Herbrand universe is a true propositional formula in I. F is *satisfiable* if it is true in some Herbrand interpretation.

THEOREM 3.4. *A formula is unsatisfiable if and only if it is false in all of its Herbrand interpretations. Furthermore, it is unsatisfiable if and only if some finite conjunction of ground instances is an unsatisfiable propositional formula.*

It will also be convenient to say that F is *M-satisfiable* if there is some Herbrand interpretation I such that every ground instance of F is true in I or contains a term with nesting depth greater than M.

EXAMPLE 3.5. Suppose that F is

$$P(s(a)) \wedge \forall x_1(P(x_1) \rightarrow Q(s(x_1))) \wedge \forall x_2(P(x_2) \rightarrow \neg Q(s(x_2))). \tag{2}$$

F is clearly unsatisfiable, but it is 1-satisfiable because there is a Herbrand interpretation in which every ground instance is true or contains a term of depth greater than one. In other words, consider the Herbrand interpretation I that sets $P(a)$ to false and all other atoms in the Herbrand base to true. Then the last two conjuncts of F contain a term of depth greater than one unless a is substituted for the variable, in which case I makes the ground instance true.

LEMMA 3.6. *A formula that is M-satisfiable for all M is satisfiable. Furthermore, an M-satisfiable formula is $(M-1)$-satisfiable.*

Proof. If a formula F is unsatisfiable, then by Theorem 3.4 there is a finite conjunction of ground instances that is unsatisfiable. Let M be the maximum nesting depth of atoms in these instances. Then F is not M-satisfiable because it is impossible to make true every ground instance with depth less than or equal to M.

Now suppose F is M-satisfiable. In some Herbrand interpretation of F, every ground instance is true or has nesting depth greater than M. Then *a fortiori*, in this interpretation every ground instance is true or has nesting depth that is greater than $M-1$, which implies $(M-1)$-satisfiability.

4. Partial Instantiation and Blockage

We now show when and how a truth valuation that satisfies a partially instantiated formula can be extended to one that satisfies all complete instantiations of the formula.

We begin with some definitions. We define a *unifier* of predicates $P(t)$ and $P(t')$ to be a pair of substitutions σ, τ such that $P(t)\sigma = P(t')\tau$. It is a most general unifier (mgu) if for any unifier (σ', τ'), $P(t)\sigma'$ is an instance of $P(t)\sigma$. In this case $P(t)\sigma$ is a most general common instance of $P(t)$, $P(t')$. (See [23, 25] for a detailed description of an mgu.)

DEFINITION 4.1. Let F, F' be quantifier-free formulas. If F' is an instantiation of F, then F is a *generalization* of F'. Given a set G of formulas containing F, we say that F is a *most specific generalization* of F' with respect to G if F generalizes F' and is an instantiation of every formula in G that generalizes F'.

EXAMPLE 4.2. Given the set $G = \{P(u, v), P(a, x), P(y, b), P(b, b)\}$, the atoms $P(u, v), P(a, x)$ and $P(y, b)$ are all generalizations of $P(a, b)$, but only $P(a, x)$ and $P(y, b)$ are most specific generalizations of $P(a, b)$.

We define two formulas E, F to be *variants* when they can be unified by renaming substitutions. In this case we also say that E is a variant of F and vice versa. For example, the atom $P(x, f(y))$ is a variant of $P(u, f(v))$ but not of $P(x, g(y))$ or of $P(a, f(y))$. This differs slightly from the standard definition, whereby E and F are variants if $E = F\sigma$ and $F = E\theta$ for some pair of substitutions σ, θ [25]. Thus $P(x, y)$ and $P(x, x)$ are variants by our definition but not by the standard one. A quantifier-free first-order formula $F = C_1 \wedge \cdots \wedge C_m$ can be viewed as a propositional formula in which variants of an atom are treated as the same atom. A truth valuation is a function v that assigns true or false to every atom. It satisfies F if it makes at least one literal in each clause true.

Our PI algorithm is more efficient if we use the idea of a "satisfier" introduced in [13], which is a distinguished atom in a clause that makes the clause true.

DEFINITION 4.3. Let S be a mapping that associates each clause C of a quantifier-free formula F with an atom $S(C)$ of C. Let $L(C)$ be the literal of C that contains $S(C)$. Then S is a *satisfier mapping* for F if, for some truth valuation v, v makes $L(C)$ true for every clause C in F. We refer to $S(C)$ as the *satisfier* of C. $S(C)$ is a *true* satisfier if $L(C)$ is $S(C)$ and *false* if $L(C)$ is $\neg S(C)$.

Given a quantifier-free formula F and a satisfier mapping for it, we can attempt to satisfy every complete instantiation of F's clauses as follows. Let each atom in a complete instantiation inherit the truth value of the most specific satisfier in F that generalizes the atom. If no satisfier generalizes the atom, it is given an arbitrary truth value.

EXAMPLE 4.4. Let $F = \forall x C_1 \wedge \forall y C_2$, where

$$\begin{aligned} C_1 &= P(a, x) \vee \underset{\mathrm{T}}{Q(a)} \vee \neg R(x) \\ C_2 &= \neg Q(y) \vee \neg \underset{\mathrm{F}}{P(y, b)}. \end{aligned}$$

Let $Q(a)$ be the satisfier for C_1 and $P(y, b)$ for C_2, as indicated. Now consider the complete instantiations of C_1, C_2.

$$\begin{aligned} &P(a, a) \vee \underset{\mathrm{T}}{Q(a)} \vee \neg R(a) \\ &\underset{\mathrm{F}}{P(a, b)} \vee \underset{\mathrm{T}}{Q(a)} \vee \neg R(b) \\ &\neg \underset{\mathrm{T}}{Q(a)} \vee \neg \underset{\mathrm{F}}{P(a, b)} \\ &\neg Q(b) \vee \neg \underset{\mathrm{F}}{P(b, b)}. \end{aligned}$$

Truth values are inherited as shown, where the unmarked atoms may receive either truth value. Note that all the clauses are satisfied.

This scheme does not always work, however.

EXAMPLE 4.5. Consider the following satisfier mapping.

$$\begin{aligned} C_1 &= \underset{\text{T}}{P(a, x)} \vee Q(a) \vee \neg R(x) \\ C_2 &= \neg Q(y) \vee \neg \underset{\text{F}}{P(y, b)}. \end{aligned}$$

Now the predicate $P(a, b)$ in the complete instantiations inherits both true and false, and the extension is not well defined.

The satisfier mapping in Example 4.4 is "unblocked," whereas the one in Example 4.5 is blocked. Sometimes, however, a conflict of truth values is resolved by the presence of more completely instantiated predicates, in which case there is no true blockage.

EXAMPLE 4.6. Let F be $\forall x C_1 \wedge \forall y C_2 \wedge C_3 \wedge C_4$, where C_1 and C_2 are as in Examples 4.4 and 4.5 and where

$$\begin{aligned} C_3 &= P(a, b) \vee Q(a) \vee \neg R(b), \\ C_4 &= \neg Q(a) \vee \neg P(a, b). \end{aligned}$$

Note that C_3 is obtained from C_1 by applying the substitution $x = b$, and C_4 from C_2 by applying $y = a$. These are the two substitutions that unify the conflicting satisfiers of C_1 and C_2. Now if $P(a, b)$ is a satisfier in C_3 or C_4, the ambiguity of the truth value of $P(a, b)$ is resolved. If it is a satisfier in neither C_3 nor C_4, then C_3 and C_4 are true regardless of the truth value of $P(a, b)$. In either case the conflict between the satisfiers of C_1 and C_2 is innocuous, and there is no blockage. These ideas are made precise below.

DEFINITION 4.7. Given a satisfier mapping S for a quantifier-free formula F, a pair of satisfiers $P(t)$, $P(t')$ is *blocked* if

(1) $P(t)$ is a true satisfier.
(2) $P(t')$ is a false satisfier.
(3) $P(t)$ and $P(t')$ have a most general unifier (σ, τ) such that $P(t)\sigma = P(t')\tau$.
(4) There are clauses C, C' in F of which $P(t)$ and $P(t')$ are respectively satisfiers and for which either (a) $C\sigma$ generalizes no clause in F or (b) $C'\tau$ generalizes no clause in F.

A satisfier is blocked if it is a member of a blocked pair, and S is blocked if some satisfier is blocked.

In Example 4.5, the satisfiers shown are blocked because $S(C_1)\sigma = S(C_2)\tau$ when σ is the substitution $\{x/b\}$ and τ is $\{y/a\}$ and because $C_1\sigma = P(a, b) \vee \neg Q(a) \vee \neg R(b)$ and $C_2\tau = \neg Q(a) \vee \neg P(a, b)$ do not both occur in F (in fact, neither does). Note that a pair of blocked satisfiers cannot be variants of each other, since they receive different truth values in some valuation.

To ensure termination of the partial instantiation algorithm, we exclude only blockages between terms whose unification results in nesting depth at most M. We say that a satisfier mapping is "M-blocked" when at least one blockage is of this sort.

DEFINITION 4.8. Given a satisfier mapping S for a quantifier-free formula F, a pair of satisfiers $P(t)$, $P(t')$ is *M-blocked* if they are blocked and their most general unifier (σ, τ) is such that $P(t)\sigma$ contains no terms of nesting depth strictly greater than M. A satisfier is M-blocked if it is the member of an M-blocked pair, and S is M-blocked if some satisfier is M-blocked.

Thus an unblocked satisfier mapping is never M-blocked, whereas a blocked mapping may or may not be M-blocked. Moreover, if a satisfier mapping is not M-blocked, either each satisfier is not blocked or it unifies with another satisfier in such a way that the unification would create an atom containing terms of nesting depth strictly greater than M.

EXAMPLE 4.9. Consider the partial instantiation that is, the conjunction of the following formulas, and the satisfier mapping shown.

$$\begin{array}{c} P(x, s(x)) \\ \mathrm{T} \\ \neg P(s(a), y) \\ \mathrm{F} \end{array}$$

The two atoms are blocked but are not 1-blocked, because the most general common instance $P(s(a), s(s(a)))$ has a term of depth 2. Thus the satisfier mapping is blocked but not 1-blocked.

THEOREM 4.10. *Given $F = \forall x_1 C_1 \wedge \cdots \wedge \forall x_m C_m$, let S be a satisfier mapping for the quantifier-free formula $F' = C_1 \wedge \cdots \wedge C_m$. Then* (a) *if S is not M-blocked, F is M-satisfiable, and* (b) *if S is unblocked, F is satisfiable.*

Proof. To establish (a), it suffices to show that if S is not M-blocked, one can define a Herbrand interpretation I such that every ground instance of F is true in I or contains a term of depth strictly greater than M. Claim (b) follows from (a): if S is unblocked, then it is not M-blocked for any M. This implies that F is M-satisfiable for any M, which is tantamount to satisfiability, by Lemma 3.6.

Let I be defined as follows. For any atom $P(d)$ in the Herbrand base of F,

- $P(d)$ is true if $P(d) \in B_M$ and F' contains a true satisfier $P(t)$ that is a most specific generalization of $P(d)$ with respect to the satisfiers of F', but no false satisfier that is a most specific generalization of $P(d)$.
- $P(d)$ is false if $P(d) \in B_M$ and F' contains a false satisfier $P(t)$ that is a most specific generalization of $P(d)$ with respect to the satisfiers of F', but no true satisfier that is a most specific generalization of $P(d)$.
- $P(d)$ is arbitrarily true or false otherwise.

Note that no ground predicate can assume both a true and a false value.

We now show that any ground instance of any clause of F is true in I or contains a term of depth greater than M. Let

$$C = \neg P_1(t_1) \vee \cdots \vee \neg P_q(t_q) \vee P_{q+1}(t_{q+1}) \vee \cdots \vee P_m(t_m)$$

be any clause in F' and

$$C_0 = \neg P_1(d_1) \vee \cdots \vee \neg P_q(d_q) \vee P_{q+1}(d_{q+1}) \vee \cdots \vee P_m(d_m)$$

be any ground instance of C. (Not all literals of C_0 need be distinct.)

Of the clauses in F' that are most specific generalizations of C_0 with respect to the clauses in F', at least one of them C' is an instantiation of C (perhaps C itself). Let

$$C' = \neg P_1(u_1) \vee \cdots \vee \neg P_q(u_q) \vee P_{q+1}(u_{q+1}) \vee \cdots \vee P_m(u_m)$$

be one such clause. C' has a satisfier $P_i(u_i)$. We claim that either $P_i(d_i)$ inherits $P_i(u_i)$'s truth value and therefore makes C_0 true, or else $P_i(d_i) \notin B_M$, which means C_0 has a term of depth greater than M. To see this, consider two cases.

1. $P_i(u_i)$ is not blocked. We will show that (a) no most specific satisfier that generalizes $P_i(d_i)$ has a truth value opposite that of $P_i(u_i)$, and (b) some such satisfier has the same truth value of $P_i(u_i)$. It follows that $P_i(d_i)$ inherits $P_i(u_i)$'s truth value and thereby makes C_0 true.

   (a) Suppose there is a most specific satisfier $P_i(v_i)$ that generalizes $P_i(d_i)$ and that has the opposite truth value. Since $P_i(d_i)$ instantiates both $P_i(u_i)$ and $P_i(v_i)$, the latter two have an mgu $P_i(u_i')$ that generalizes $P_i(d_i)$. We write $P_i(u_i') = P_i(u_i)\tau_1$ and observe that for some substitution τ_2, $C'\tau_1\tau_2 = C_0$. We first note that $P(u_i')$ is more specific than $P(u_i)$. This is because $P_i(u_i)$ and $P_i(v_i)$ have different truth values and are therefore not variants of each other. So if $P(u_i)$ and $P(u_i')$ were variants, $P_i(v_i)$ would not be a most specific satisfier that generalizes $P_i(d_i)$. Next, we note that because the pair $P_i(u_i)$ and $P_i(v_i)$ are by assumption not blocked, $C'\tau_1$ must generalize some clause in F'. $C'\tau_1$ is more specific than C', since as just shown $P(u_i') = P(u_i)\tau_1$ is more specific than $P(u_i)$. Also $C'\tau_1$ generalizes C_0 because $C_0 = C'\tau_1\tau_2$. But this contradicts the assumption that C' is a most specific generalization of C_0 that is an instantiation of C. So any most specific satisfier that generalizes $P_i(d_i)$ must have the same truth value as $P_i(u_i)$.

(b) We now show that there in fact exists a most specific satisfier that generalizes $P_i(d_i)$, which by (a) must have the same truth value as $P_i(u_i)$. But this is clear, because some instantiation of $P_i(u_i)$, perhaps $P_i(u_i)$ itself, is such a satisfier.

2. $P_i(u_i)$ is blocked but not M-blocked. There are two cases:

(a) For no satisfier $P_i(u_i')$ that creates a blockage with $P_i(u_i)$ is $P_i(d_i)$ an instance of $P_i(u_i')$. In this case, $P_i(d_i)$ inherits the truth value of $P_i(u_i)$, as desired.

(b) For some satisfier $P_i(u_i')$ that creates a blockage with $P_i(u_i)$, $P_i(d_i)$ is an instance of $P_i(u_i')$. Then $P_i(u_i)$ and $P_i(u_i')$ have an mgu (σ, τ) such that $P_i(d_i)$ is an instance of $P_i(u_i)\sigma$. But because $P_i(u_i)$ and $P_i(u_i')$ are not M-blocked, $P_i(u_i)\sigma \notin B_M$, which implies $P_i(d_i) \notin B_M$.

5. The Primal Approach

Now we present the *primal partial instantiation* method. For $M = 0, 1, \ldots,$ the algorithm tries to find a satisfier mapping for the given formula F that is not M-blocked. If it finds a satisfier mapping that is M-blocked, it partially instantiates two clauses that cause the blockage and conjoins them with F, so that a stronger formula is checked for satisfiability in the next iteration. The procedure terminates when (a) no satisfier mapping exists, in which case F is unsatisfiable, or (b) an unblocked satisfier mapping is found, in which case F is satisfiable.

It will be shown later that if F is unsatisfiable, the algorithm terminates with a proof of unsatisfiability. Because first-order logic is semidecidable, there is no assurance of termination if F is satisfiable.

ALGORITHM PPI (Primal Partial Instantiation):

Let $F = \forall x_1 C_1 \wedge \cdots \wedge \forall x_m C_m$ be a first-order formula.

1. Initialization. Set $F_0 =: C_1 \wedge \cdots \wedge C_m$, $k := 0$, and $M := 0$.

2. Ground satisfiability. Try to find a satisfier mapping S for F_k that treats variants of the same atom as the same atom.

3. Termination check.

- If S does not exist, then **stop**: F is unsatisfiable.
- Otherwise, if S is unblocked, then **stop**: F is satisfiable.
- Otherwise, if S is not M-blocked, then F is M-satisfiable. Let $M := M + 1$, and repeat Step 3.

4. Refinement. (S is M-blocked.) Let C_h and C_i be two clauses in F_k whose satisfiers are M-blocked, and let (σ, τ) be a most general unifier of $S(C_h)$ and

$S(C_i)$. Set $F_{k+1} = F_k \wedge C_h\sigma \wedge C_i\tau$ after standardizing apart, set $k := k + 1$, and go to Step 2.

This algorithm not only generalizes Jeroslow's so as to accommodate functions, but it simplifies it in three ways.

– Since F_0 is a conjunction of clauses that are standardized apart, the PPI algorithm obtains F_{k+1} by conjoining to F_k two partially instantiated clauses rather than two partial instantiations of the entire formula F_0, as in Jeroslow's method.
– PPI eliminates Jeroslow's "direct covering" mechanism, which requires that two atoms not be considered blocked unless their unifying substitutions are "directly covered" by the clauses containing them. This requires that one keep track of which instantiations are covered and directly covered by each clause generated.
– PPI checks only satisfiers with opposite truth values for possible blockage. This significantly reduces the amount of computation, as Jeroslow's method checks all pairs of atoms that are given opposite truth values by a complete satisfying truth valuation.

EXAMPLE 5.1 (Satisfiability with termination). Consider the problem $C_1 \wedge \forall x C_2 \wedge \forall y C_3 \wedge C_4$, where

$$\begin{aligned} C_1 &= \underset{\mathrm{T}}{P(s(a))} \\ C_2 &= \neg \underset{\mathrm{F}}{P(x)} \vee Q(s(x)) \\ C_3 &= \neg \underset{\mathrm{F}}{Q(s(y))} \vee R(s(y)) \\ C_4 &= \neg \underset{\mathrm{F}}{R(s(a))}. \end{aligned} \tag{3}$$

Thus $F_0 = C_1 \wedge C_2 \wedge C_3 \wedge C_4$. A satisfier mapping for F_0 is shown. For $M = 0$ we note that $S(C_1)$ and $S(C_2)$ – that is, $P(s(a))$ and $P(x)$ – are blocked because they have an mgu $P(s(a))\sigma = P(x)\tau = P(s(a))$, where σ is the empty substitution and $\tau = \{x/s(a)\}$. They are not 0-blocked, however, because their common instance contains a term of depth 1, which means that F is 0-satisfiable. Now, setting $M = 1$, we note that this pair is 1-blocked, and Step 4 creates $F_1 = F_0 \wedge C_5$, where $C_5 = C_2\tau$, or

$$C_5 = \neg P(s(a)) \vee \underset{\mathrm{T}}{Q(s(s(a)))}.$$

It is not necessary to add $C_1\sigma$ to the formula because σ has no effect on C_1. The previous satisfier mapping can be extended as shown. $Q(s(y))$ and $Q(s(s(a)))$ are blocked but not 1-blocked, so that F is 1-satisfiable. Setting $M = 2$, Step 4 creates $F_2 = F_1 \wedge C_6$, where

$$C_6 = \neg Q(s(s(a))) \vee \underset{\text{T}}{R(s(s(a)))}.$$

The satisfier mapping shown is unblocked, and PPI terminates with satisfiability.

EXAMPLE 5.2 (Advantage of satisfiers). We next illustrate the advantage of using satisfier mappings rather than complete valuations in the algorithm. Consider the formula $\forall x_1 C_1 \wedge \forall x_2 C_2 \wedge \forall x_3 C_3 \wedge \forall x_4 C_4$, where

$$\begin{aligned}
C_1 &= P(a, x_1) \vee \underset{\text{F}}{\neg Q(x_1, b)}\\
C_2 &= \underset{\text{F}}{\neg P(x_2, b)}\\
C_3 &= \neg P(a, x_3) \vee \underset{\text{F}}{\neg Q(a, x_3)}\\
C_4 &= \underset{\text{T}}{P(x_4, c)}.
\end{aligned}$$

An unblocked satisfier mapping is shown, which proves satisfiability. The example is interesting because, here, the use of satisfiers not only simplifies the job of checking for blockage but also avoids an additional iteration. This is because any truth valuation consistent with the above satisfiers is blocked and therefore would require another iteration if complete valuations were used in the algorithm. To see this, note that if $P(a, x_1)$ is true, it and $P(x_2, b)$ are blocked, whereas if $P(a, x_3)$ is false, $P(a, x_3)$ and $P(x_4, c)$ are blocked.

EXAMPLE 5.3 (Unsatisfiability). Let $F = C_1 \wedge \forall x C_2 \wedge \forall y C_3$, where

$$\begin{aligned}
C_1 &= \underset{\text{T}}{P(s(a))}\\
C_2 &= \underset{\text{F}}{\neg P(x)} \vee Q(s(x))\\
C_3 &= \neg P(y) \vee \underset{\text{F}}{\neg Q(s(y))}.
\end{aligned} \tag{4}$$

The satisfier mapping is not 0-blocked, but the 1-blockage between $P(s(a))$ and $P(x)$ creates $F_1 = F_0 \wedge C_4$, with

$$C_4 = \neg P(s(a)) \vee \underset{\text{T}}{Q(s(s(a)))}.$$

The satisfier mapping can be extended as shown. The 2-blockage between $Q(s(y))$ and $Q(s(s(a)))$ creates $F_2 = F_1 \wedge C_5$, with

$$C_5 = \neg P(s(a)) \vee \neg Q(s(s(a))).$$

Because F_2 is unsatisfiable as a propositional formula, the algorithm terminates.

EXAMPLE 5.4 (Satisfiability without termination). Bernays and Schönfinkel [8] mentioned that the following formula is satisfiable only in an infinite domain:

$$\forall x \forall y \forall z \exists w[\neg P(x,x) \wedge ((P(x,y) \wedge P(y,z)) \rightarrow P(x,z)) \wedge P(x,w)]. \tag{5}$$

We can view $P()$ as representing a precedence relation. If we consider any finite domain whose elements are ordered by this relation, the last element precedes no element of the domain. Hence, F_1 has no finite model. The only Herbrand model for F_1 makes infinitely many members of the Herbrand base true: $P(a, s(a))$, $P(s(a), s(s(a)))$, …, because of the unlimited nesting of function symbols.

By adding a Skolem function s we obtain from (5) a formula $F = \forall x_1 C_1 \wedge \forall(x_2, y_2, z_2)C_2 \wedge \forall x_3 C_3$ in prenex clausal form, where

$$\begin{aligned} C_1 &= \underset{F}{\neg P(x_1, x_1)} \\ C_2 &= \underset{\mathrm{F}}{\neg P(x_2, y_2)} \vee \neg P(y_2, z_2) \vee P(x_2, z_2) \\ C_3 &= \underset{\mathrm{T}}{P(x_3, s(x_3))}. \end{aligned} \tag{6}$$

A satisfier mapping is also shown.

The satisfiers in C_2 and C_3 are 1-blocked. Thus, when $M = 1$, we set $F_1 = F_0 \wedge C_4$, where

$$C_4 = \neg P(x_4, s(x_4)) \vee \underset{\mathrm{F}}{\neg P(s(x_4), z_4)} \vee P(x_4, z_4). \tag{7}$$

Now let $M = 2$. The satisfier mapping can be extended as shown. The 2-blockage between $S(C_3)$ and $S(C_4)$ generates $F_2 = F_1 \wedge C_5 \wedge C_6$, with

$$\begin{aligned} C_5 &= \underset{\mathrm{T}}{P(s(x_5), s(s(x_5)))} \\ C_6 &= \neg P(x_6, s(x_6)) \vee \neg P(s(x_6), s(s(x_6))) \vee \underset{\mathrm{T}}{P(x_6, s(s(x_6)))}, \end{aligned}$$

which has the satisfier mapping indicated. A 2-blockage between $S(C_2)$ and $S(C_5)$ generates $F_3 = F_2 \wedge C_7$, with

$$C_7 = \neg P(s(x_7), s(s(x_7))) \vee \underset{\mathrm{F}}{\neg P(s(s(x_7)), z_7)} \vee P(s(x_7), z_7).$$

Finally, a 2-blockage between $S(C_2)$ and $S(C_6)$ generates the clause

$$C_8 = \neg P(x_8, s(s(x_8))) \vee \underset{\mathrm{F}}{\neg P(s(s(x_8)), z_8)} \vee P(x_8, z_8).$$

The valuation shown is no longer 2-blocked, as all the possible remaining blockages – namely, those between the pairs $\{S(C_3), S(C_7)\}$, $\{S(C_3), S(C_8)\}$, $\{S(C_4), S(C_6)\}$, $\{S(C_5), S(C_7)\}$, $\{S(C_5), S(C_8)\}$, $\{S(C_6), S(C_7)\}$, $\{S(C_6), S(C_8)\}$ – would generate terms of depth greater than 2. So F is 2-satisfiable. The process continues indefinitely, showing that F is M-satisfiable for any given M.

In the above examples, the satisfier mapping obtained for each F_{k+1} assigns the same satisfiers to F_k as obtained in the previous iteration. This is not always possible. In general, the satisfiers of previous clauses may change. Nonetheless, as noted earlier, efficient incremental satisfiability algorithms can use information obtained while solving the satisfiability problem for F_k to help solve the one for F_{k+1}.

THEOREM 5.5. *If a fixed upper bound M^* is placed on M, the algorithm PPI terminates after a finite number of steps.*

Proof of this theorem requires the following lemma.

LEMMA 5.6. *Let $F_0 = C_1 \wedge \cdots \wedge C_m$, and let $F_0, F_1, \ldots, F_k$ be formulas in which each $F_i = F_{i-1} \wedge C\sigma \wedge C'\tau$ (for $i > 0$), where C, C' are clauses of F_{i-1} and (σ, τ) is an mgu of some pair $P(t), P(t')$ of atoms such that*

(1) *$P(t)$ and $P(t')$ are not variants of each other;*
(2) *$P(t)$ occurs positively in C;*
(3) *$P(t')$ occurs negatively in C';*
(4) *$P(t)\sigma \in B_{M^*}$;*
(5) *either $C\sigma$ or $C'\tau$ generalizes no clause in F_{i-1}.*

Let D_p be the maximum nesting depth of terms in any given clause C_p of F_k, and let

$$\Delta_p = \sum_j (M^* - \min\{\delta_{pj}, M^*\}),$$

where δ_{pj} is the maximum nesting depth of terms in the jth atom of C_p, and the sum is taken over all atoms of C_p. Then for any clause C_p in F_k,

$$D_p \leq \max_{i \in \{1,\ldots,m\}} \{M'_i + \Delta_i\}, \tag{8}$$

where M'_i is the maximum nesting depth of terms in C_i, $i = 1, \ldots, m$.

Proof. It suffices to show that the following holds for any clause C_p in F_k:

$$D_p \leq \max_{i \in \{1,\ldots,m\}} \{M'_i + \Delta_i\} - \Delta_p. \tag{9}$$

This suffices because it and $\Delta_p \geq 0$ imply (8).

The proof of (9) is by induction on k. The claim clearly holds for $k = 0$, since the nesting depth of terms in C_p $(p = 1, \dots, m)$ is bounded above by

$$M'_p = (M'_p + \Delta_p) - \Delta_p.$$

Now suppose that the claim is true for $k - 1$. A clause C_q in $F_k \setminus F_{k-1}$ is an instantiation $C_r\sigma$ of some clause C_r in F_{k-1} containing an atom $P_{rj}(t_{rj})$ that has a most general common instance $P_{rj}(t_{rj})\sigma$ with another atom in some other clause of F_{k-1}, where $P_{rj}(t_{rj})\sigma \in B_{M^*}$. Let δ be the amount by which the nesting depth of t_{rj} increases when σ is applied to $P_{rj}(t_{rj})$. Then when σ is applied to C_r, it increases the maximum depth of terms in C_r by at most δ, whence

$$D_q \leq D_r + \delta. \tag{10}$$

But the application of σ to C_r reduces Δ_r by at least δ, whence

$$\delta \leq \Delta_r - \Delta_q.$$

Substituting this into (10), we obtain that

$$D_q \leq D_r + \Delta_r - \Delta_q.$$

From this it follows that (9) is true for $p = q$ because it assumed true for $p = r$.

Proof of Theorem 5.5. Each F_{k+1} in the algorithm consists of F_k conjoined with one or two additional clauses, neither of which is a variant of any clauses in F_k. We will show that if M is bounded by M^*, the nesting depth of F_k (for any k) is bounded above by a number that is independent of k. Because no clause of F_k is a variant of another, it follows that the number of clauses in F_k is similarly bounded, and the algorithm is finite.

If D_p is the nesting depth of any clause C_p in F_k, the desired bound is given by (8) in Lemma 5.6. To apply this lemma, one need only to show that each F_k satisfies conditions (1)–(5). Let C_p be a clause in $F_k \setminus F_{k-1}$, and let $P(t)$, $P(t')$ be the satisfiers with mgu (σ, τ) whose blockage gives rise to the addition of C_p. Condition (1) holds because $P(t)$ and $P(t')$ receive different truth values in a variant-independent valuation. Conditions (2) and (3) hold by definition of blockage, where C and C' are distinct because a single clause cannot contain two satisfiers. Conditions (4) and (5) likewise hold by definition of blockage.

To prove correctness of the algorithm, we need two lemmas. The first is straightforward to prove.

LEMMA 5.7. *A formula F is satisfiable* (*or M-satisfiable*) *if and only if* $F \wedge \bigwedge_{k=1}^{K} F\sigma_k$ *is satisfiable* (*or M-satisfiable, respectively*) *for any set of substitutions* $\sigma_1, \dots, \sigma_K$.

LEMMA 5.8. *Let $F = \forall x_1 C_1 \wedge \cdots \wedge \forall x_m C_m$. If $F' = C_1 \wedge \cdots \wedge C_m$ is unsatisfiable when regarded as a propositional formula in which variants of the same atom are identified, then F is unsatisfiable.*

Proof. We will construct a ground instance $F'\sigma$ of F' such that a pair of atoms in F' are variants of each other if and only if their instances in $F'\sigma$ are identical. Then $F'\sigma$ is unsatisfiable, and by Theorem 3.4 F is unsatisfiable.

To construct $F'\sigma$, consider the following equivalence relation among variables. Whenever two atoms $P(t)$, $P(t')$ are variants, so that $P(t)\sigma = P(t')\tau$ for some pair σ, τ of renaming substitutions, regard any pair of variables x_j, x_k for which $x_j\sigma = x_k\tau$ as equivalent. Associate each equivalence class of variables with a distinct constant that does not already occur in F'. Now we can let σ replace each variable with the constant associated with its equivalence class.

THEOREM 5.9 (Correctness). *The algorithm PPI indicates* (a) *unsatisfiability only if F is unsatisfiable,* (b) *satisfiability only if F is satisfiable, and* (c) *M-satisfiability only if F is M-satisfiable.*

Proof. (a) If PPI indicates unsatisfiability, then in the last step it generates a formula F_k that is unsatisfiable when regarded as a propositional formula in which variants of atoms are identified. Then by Lemmas 5.7 and 5.8, F is unsatisfiable.

(b) If PPI indicates satisfiability, then it obtains for a formula F_k an unblocked satisfier mapping. Then F is satisfiable, by Theorem 4.10 and Lemma 5.7.

(c) If PPI indicates M-satisfiability, then it obtains for a formula F_k a satisfier mapping that is not M-blocked. Then F is M-satisfiable, by Theorem 4.10 and Lemma 5.7.

THEOREM 5.10 (Completeness). *If F is unsatisfiable, then PPI terminates with an indication of unsatisfiability.*

Proof. Because F is unsatisfiable, by Lemma 3.6 there is an M^* such that F it is not M-satisfiable for any $M \geq M^*$. By Theorem 5.9, PPI will not indicate M-satisfiability. So the execution of the algorithm is unaffected by placing an upper bound M^* on M, which means by Theorem 5.5 that the algorithm terminates. By Theorem 5.9 it terminates with an indication of unsatisfiability.

6. The Dual Approach

The primal approach to PI uses a reactive strategy that deals with blockages as they arise. It starts with a weakened form of the satisfiability problem (a partial instantiation) and gradually strengthens it with further instantiations that resolve blockages. Unsatisfiability at any point indicates unsatisfiability of the original formula, but satisfiability indicates the same for the original formula only if the satisfying valuation is unblocked.

The dual approach uses a proactive strategy that tries to anticipate blockages. It starts with a strengthened form of the problem that contains clauses that rule out possible blockages. It gradually weakens the problem as further instantiations are added and fewer possible blockages need to be anticipated. Satisfiability at any point indicates satisfiability of the original formula, but unsatisfiability indicates the same for the original formula only if the problem is unsatisfiable after removing the clauses that rule out blockage.

DEFINITION 6.1. Let $F = C_1 \wedge \cdots \wedge C_m$ be a quantifier-free first-order formula. An *auxiliary clause* for F is any clause of the form $P(t) \rightarrow P(t')$ (i.e., $\neg P(t) \vee P(t')$) for which

(1) $P(t)$ and $P(t')$ are not variants of each other,
(2) $P(t)$ occurs positively in some clause C of F,
(3) $P(t')$ occurs negatively in some other clause C' of F,
(4) $P(t)$ and $P(t')$ have an mgu (σ, τ) such that $P(t)\sigma = P(t')\tau$ belongs to B_M, and
(5) either $C\sigma$ or $C'\tau$ generalizes no clause in F.

The *depth* of the auxiliary clause $P(t) \rightarrow P(t')$ is the maximum nesting depth of $P(t)\sigma$, $P(t')\tau$.

EXAMPLE 6.2. Consider again the formula $F = C_1 \wedge \forall x C_2 \wedge \forall y C_3 \wedge C_4$, where C_1, C_2, C_3, C_4 are given by (3). There are two auxiliary clauses, each of depth 1:

$$\begin{aligned} &P(s(a)) \rightarrow P(x) \\ &R(s(y)) \rightarrow R(s(a)). \end{aligned} \tag{11}$$

ALGORITHM DPI (Dual Partial Instantiation):

Let $F = \forall x_1 C_1 \wedge \cdots \wedge \forall x_m C_m$ be a first-order formula.

1. Initialization. Set $F_0 = C_1 \wedge \cdots \wedge C_m$, $k := 0$, and $M := 0$.

2. Unsatisfiability check. If F_k is satisfied by no variant-independent valuation, stop; F is unsatisfiable.

3. Blockage Avoidance. Generate auxiliary clauses $B_1, \ldots, B_p$ for F_k until conditions (a) and (b) below are satisfied or until all auxiliary clauses have been generated, whichever occurs first.

(a) $F_k \wedge B_1 \wedge \cdots \wedge B_p$ is satisfied by no variant-independent valuation;
(b) some auxiliary clause B_i has depth less than or equal to M.

4. Satisfiability Check. If $B_1, \ldots, B_p$ do not satisfy condition (a), then stop; F is satisfiable. If they do not satisfy (b), let M be one less than the minimum depth of $B_1, \ldots, B_p$; F is M-satisfiable.

5. Refinement. Let $P(t) \to P(t')$ be any auxiliary clause generated in Step 3 with depth at most $M+1$, where $P(t)$ occurs posited in clause C of F_k and $P(t')$ occurs negated in clause C' of F_k. Let (σ, τ) be an mgu of $P(t), P(t')$. Set $F_{k+1} = F_k \wedge C\sigma \wedge C'\tau$, $k := k+1$, and go to Step 2.

In practice one would use an incremental algorithm to check each $F_k \wedge B_1 \wedge \cdots \wedge B_i$ for satisfiability. As in the primal case, one can terminate the algorithm if M attains some large value M^* and conclude that F is at least M^*-satisfiable.

It may be best in practice to stop generating auxiliary clauses in Step 3 when unsatisfiability is detected, rather than to generate them all in order to verify that F is M-satisfiable for some M (namely, an M equal to one less than the minimum depth of the auxiliary clauses). One might therefore replace Steps 3 and 4 with the following. If the modified algorithm terminates when M reaches M^*, F may be M-satisfiable only for some M smaller than M^*.

3. Blockage Avoidance. Generate auxiliary clauses $B_1, \ldots, B_p$ for F_k until $F_k \wedge B_1 \wedge \cdots \wedge B_p$ is satisfied by no variant-independent valuation or until all auxiliary clauses have been generated, whichever occurs first.

4. Satisfiability Check. If $F_k \wedge B_1, \ldots, B_p$ is satisfiable by a variant-independent valuation, then stop; F is satisfiable. Otherwise let M be one less than the minimum depth of the clauses $B_1, \ldots, B_p$.

We will use this modified algorithm in all subsequent examples.

EXAMPLE 6.3 (Satisfiability with termination). Consider again $F = C_1 \wedge \forall x C_2 \wedge \forall y C_3 \wedge C_4$, where $C_1, \ldots, C_4$ are given by (3). The auxiliary clauses B_1, B_2 appear in (11). Both clauses are needed to obtain an unsatisfiable proposition $F_0 \wedge B_1 \wedge B_2$. M is set to 0, and F_1 is $C_1 \wedge \cdots \wedge C_4 \wedge C_5$, where C_5 is obtained either from the substitution $\{x/s(a)\}$ in C_2 or the substitution $\{y/a\}$ in C_4. We will use the former to illustrate the nesting of functions:

$$C_5 = \neg P(s(a)) \vee Q(s(s(a))).$$

Returning to Step 2, we see there are two auxiliary clauses for F_1. Unsatisfiability occurs only when both have been generated:

$$\begin{aligned} B_1 &= R(s(y)) \to R(s(a)), \\ B_2 &= Q(s(s(a))) \to Q(s(y)). \end{aligned}$$

M is set to 1. F_2 is $F_1 \wedge C_6$, where C_6 is obtained by using the substitution that unifies the atoms of B_2:

$$C_6 = \neg Q(s(s(a)) \vee R(s(s(a))).$$

Only one auxiliary clause exists for F_2:

$$B_1 = R(s(y)) \to R(s(a)).$$

Because $F_2 \wedge B_1$ is satisfiable, the algorithm terminates with the conclusion that F is satisfiable.

EXAMPLE 6.4 (Unsatisfiability). Let $F = C_1 \wedge \forall x C_2 \wedge \forall y C_3$, where C_1, C_2, C_3 are given by (4). Generation of one auxiliary clause,

$$B_1 = P(s(a)) \to P(x),$$

results in an unsatisfiable formula $F_0 \wedge B_1$. M is set to 0, and F_1 is set to $F_0 \wedge C_4$, where

$$C_4 = \neg P(s(a)) \vee Q(s(s(a))).$$

Again one auxiliary clause for F_1 results in unsatisfiability:

$$B_1 = P(s(a)) \to P(y).$$

Now $F_2 = F_1 \wedge C_5$, with

$$C_5 = \neg P(s(a)) \vee \neg Q(s(s(a))).$$

Because F_2 is unsatisfiable (Step 2), so is F.

In the special case of datalog formulas, there are useful heuristics, based on shortest paths in hypergraphs [12], for selecting the clause $P(t) \to P(t')$ in Step 3. Similar heuristics may be available in larger fragments.

THEOREM 6.5. *If a fixed upper bound M^* is placed on M, the algorithm DPI terminates after a finite number of steps.*

Proof. As in the proof of Theorem 5.5, it suffices to show that the nesting depth of F_k is bounded above by a number that is independent of k. Again the desired bound is given by (8) in Lemma 5.6. The lemma applies because each F_k in DPI clearly satisfies conditions (1)–(5) stated in the lemma.

THEOREM 6.6 (Correctness). *The algorithm DPI indicates* (a) *unsatisfiability only if F is unsatisfiable,* (b) *satisfiability only if F is satisfiable, and* (c) *M-satisfiability only if F is M-satisfiable.*

Proof. (a) If PPI indicates unsatisfiability, then in the last step it generates an unsatisfiable propositional formula F_k. By Lemmas 5.7 and 5.8, F is unsatisfiable.

(b) If DPI indicates satisfiability, then in the last step it finds the formula $F_k \wedge B_1 \wedge \cdots \wedge B_p$ to have a variant-independent satisfying truth assignment, where $B_1, \ldots, B_p$ are a complete list of auxiliary clauses for F_k. Such an assignment provides an unblocked satisfier mapping for F_k, because any satisfier mapping that creates a blocking pair $P(t), P(t')$ violates the auxiliary clause $P(t) \rightarrow P(t')$. So F is satisfiable by Theorem 4.10 and Lemma 5.7.

(c) If DPI indicates M-satisfiability, then it obtains an unsatisfiable formula $F_k \wedge B_1 \wedge \cdots \wedge B_p$, where $B_1, \ldots, B_p$ is a complete list of auxiliary clauses for F_k, and M is one less than the minimum depth of these clauses. Because F_k is satisfiable (Step 2), it has a variant-independent satisfier mapping. Such a mapping cannot be M-blocked, because if M-blockage were possible, there would be an auxiliary clause B_i of depth M or less. So F is satisfiable by Theorem 4.10 and Lemma 5.7.

THEOREM 6.7 (Completeness). *If F is unsatisfiable, then DPI terminates with an indication of unsatisfiability.*

Proof. Because F is unsatisfiable, by Lemma 3.6 there is an M^* such that F is not M-satisfiable for any $M \geq M^*$. By Theorem 6.6, DPI will not indicate M-satisfiability. So the execution of the algorithm is unaffected by placing an upper bound M^* on M, which means by Theorem 6.5 that the algorithm terminates. By Theorem 6.6 it terminates with an indication of unsatisfiability.

7. Implementation Issues

Since propositional satisfiability problems must be solved many times, with one or two additional clauses each time, we used an incremental satisfiability algorithm [17]. This substantially reduced computation time.

Testing for blockage, however, remained a bottleneck. It involves checking whether the satisfiers of any pair of clauses are unifiable and, if so, whether the partially instantiated clauses so obtained generalize any clause in the current formula F. If F contains clauses $C_1, \ldots, C_n$, the unification test requires $O(n^2)$ attempts to unify (one for each pair of clauses in the worst case). Checking for the existence of a generalized clause requires $O(n)$ tests.

We observed that quite often, the satisfier atoms for most clauses are the same as in the previous iteration (i.e., the last time the propositional satisfiability problem was solved). This led to the idea of testing blockages incrementally by reusing the results of the blockage tests (mgus and generalizations) that were obtained in the previous iteration. This accelerated the procedure considerably by reducing the number of unification attempts, and generalization tests in many cases.

To implement incremental blockage testing, we associated with each clause C_i a list of nodes $N_{i+1}, \ldots, N_n$ corresponding to subsequent clauses $C_{i+1}, \ldots, C_n$ in F. Each N_j stores the results of the most recent check for blockage between C_i and C_j. The results include pointers to the satisfiers of C_i and C_j, the mgu (if any), and, if an mgu was found, indicate whether the generalization test succeeded. Initially all the nodes are empty. When we check for blockage involving a pair C_i, C_j for a second time, we look up N_j in the list for C_i and check whether the satisfiers are still the same. If so, we simply use the stored mgu and generalization test results. If the satisfiers have changed, we redo the computations.

Whenever a partially instantiated clause C_{n+1} is added to the formula, because of blockage involving C_i and C_j, we append an empty result node N_{n+1} to the list for each clause $C_1, \dots, C_n$ and increment n by one. Also, the generalization test is not entirely saved because it is necessary to check whether C_{n+1} generalizes clauses that have been added since the pair C_i, C_j was last checked.

8. Computational Results

We applied the primal PI algorithm to four instances of standard planning problems.

The monkey-banana problem. The problem is stated below as a first-order formula. The first four clauses are the rules, and the last clause is the negation of the query. The PPI method generates the answer by determining the formula to be unsatisfiable.

$$\begin{aligned}
&(\neg P(x, y, z, s) \vee P(z, y, z, w(x, z, s)) \;\wedge \\
&(\neg P(x, y, x, s) \vee P(y, y, y, c(x, y, s)) \;\wedge \\
&(\neg P(B, B, B, s) \vee G(cl(s)) \;\wedge \\
&P(A, B, C, S) \;\wedge \\
&\neg G(s)
\end{aligned}$$

$P(x, y, z, s)$ means that the monkey is at position x, the ladder is at position y, and the banana is at position z, in state s. $G(s)$ means that the monkey can grasp the banana in state s. The letters w, c, cl are abbreviated names for the functions *walk*, *carry*, and *climb*, respectively.

Shortest plan problem. One can move through positions A, B, ..., G in single steps or jumps of two. How does one go from A to F in the fewest number of steps? The problem can be written as follows.

$P(A, s) \wedge$	(initially at A)
$(\neg P(A, s) \vee P(B, \text{step}(s))) \wedge$	(can step from A to B)
$(\neg P(B, s) \vee P(C, \text{step}(s))) \wedge$	(can step from B to C)
$(\neg P(C, s) \vee P(D, \text{step}(s))) \wedge$	(can step from C to D)
$(\neg P(D, s) \vee P(E, \text{step}(s))) \wedge$	(can step from D to E)
$(\neg P(E, s) \vee P(F, \text{step}(s))) \wedge$	(can step from E to F)
$(\neg P(F, s) \vee P(G, \text{step}(s))) \wedge$	(can step from F to G)
$(\neg P(A, s) \vee P(C, \text{jump}(s))) \wedge$	(can jump from A to C)
$(\neg P(B, s) \vee P(D, \text{jump}(s))) \wedge$	(can jump from B to D)
$(\neg P(C, s) \vee P(E, \text{jump}(s))) \wedge$	(can jump from C to E)

$(\neg P(D, s) \vee P(F, \mathrm{jump}(s))) \wedge$	(can jump from D to F)
$(\neg P(E, s) \vee P(G, \mathrm{jump}(s))) \wedge$	(can jump from E to G)
$(\neg P(F, s) \vee \mathrm{ans}(s))$	(cannot move from A to F)

Blocks world. Two instances were solved.

The results in Table I, also reported in [6], demonstrate the substantial advantage of both the incremental satisfiability algorithm and incremental testing for blockage. Detailed results for one problem are given below.

EXAMPLE 8.1 (Monkey-banana problem). PPI resolved 20 blockages, making the final number of clauses 25. There were 15 literals in the final propositional satisfiability problem. The method therefore solved a 25-clause, 15-literal propositional satisfiability problem incrementally (in 20 steps). It performed 427 mgu computations (both successful and unsuccessful) and 4027 generalization tests.

The following are the additional clauses added upon resolution of various blockages encountered. After adding 20 clauses, the formula is determined to be unsatisfiable.

$$\neg P(A, B, C, S) \vee P(C, B, C, w(A, C, S))$$
$$\neg P(C, B, C, w(A, C, S)) \vee P(C, B, C, w(C, C, w(A, C, S)))$$
$$\neg P(C, B, C, w(A, C, S)) \vee P(B, B, B, c(C, B, w(A, C, S)))$$
$$\neg P(C, B, C, w(C, C, w(A, C, S))) \vee P(C, B, C, w(C, C, w(C, C, w(A, C, S))))$$
$$\neg P(B, B, B, c(C, B, w(A, C, S))) \vee P(B, B, B, w(B, B, c(C, B, w(A, C, S))))$$
$$\neg P(C, B, C, w(C, C, w(A, C, S))) \vee P(B, B, B, c(C, B, w(C, C, w(A, C, S))))$$
$$\neg P(B, B, B, c(C, B, w(A, C, S))) \vee P(B, B, B, c(B, B, c(C, B, w(A, C, S))))$$
$$\neg P(B, B, B, c(C, B, w(A, C, S))) + R(cl(c(C, B, w(A, C, S))))$$
$$\neg P(C, B, C, w(C, C, w(C, C, w(A, C, S)))) \vee P(C, B, C, w(C, C, w(C, C, w(C, C, w(A, C, S)))))$$
$$\neg P(B, B, B, w(B, B, c(C, B, w(A, C, S)))) \vee P(B, B, B, w(B, B, w(B, B, c(C, B, w(A, C, S)))))$$
$$\neg P(B, B, B, c(C, B, w(C, C, w(A, C, S)))) \vee P(B, B, B, w(B, B, c(C, B, w(C, C, w(A, C, S)))))$$

Table I. Computational results.

Problem Instance	Number of Clauses		Computation Time (seconds)		
	Initial	Final	Incremental SAT & blockage	Incremental SAT only	Neither
Monkey-banana	5	25	0.24	0.30	0.32
Shortest plan	13	38	0.27	0.43	0.70
Blocks world (smaller)	19	118	2.53	5.48	27.95
Blocks world (larger)	33	279	16.31	47.05	1904.02

$$\neg P(B, B, B, c(B, B, c(C, B, w(A, C, S)))) \vee P(B, B, B, w(B, B, c(B, B, c(C, B, w(A, C, S)))))$$
$$\neg P(C, B, C, w(C, C, w(C, C, w(A, C, S)))) \vee P(B, B, B, c(C, B, w(C, C, w(C, C, w(A, C, S)))))$$
$$\neg P(B, B, B, w(B, B, c(C, B, w(A, C, S)))) \vee P(B, B, B, c(B, B, w(B, B, c(C, B, w(A, C, S)))))$$
$$\neg P(B, B, B, c(C, B, w(C, C, w(A, C, S)))) \vee P(B, B, B, c(B, B, c(C, B, w(C, C, w(A, C, S)))))$$
$$\neg P(B, B, B, c(B, B, c(C, B, w(A, C, S)))) \vee P(B, B, B, c(B, B, c(B, B, c(C, B, w(A, C, S)))))$$
$$\neg P(B, B, B, w(B, B, c(C, B, w(A, C, S)))) \vee R(cl(w(B, B, c(C, B, w(A, C, S)))))$$
$$\neg P(B, B, B, c(C, B, w(C, C, w(A, C, S)))) \vee R(cl(c(C, B, w(C, C, w(A, C, S)))))$$
$$\neg P(B, B, B, c(B, B, c(C, B, w(A, C, S)))) \vee R(cl(c(B, B, c(C, B, w(A, C, S)))))$$
$$\neg R(cl(c(C, B, w(A, C, S))))$$

The monkey gets the banana after he walks from A to C, carries the ladder from C to B, and then climbs the ladder.

9. Concluding Remarks

Both the primal and dual algorithms leave considerable freedom for variation, and the efficiency of the algorithms will depend on how well this freedom is used.

In the primal algorithm, a large number of blockages will typically occur in any given iteration. The key decision is which blockage(s) to remove in Step 3 before re-solving the propositional satisfiability problem. The aim of a selection heuristic should be to find a satisfying solution as soon as possible.

In the dual algorithm, the key decisions are (a) in what order to generate the clauses that avoid blockage (assuming that generation stops as soon as the clause set is unsatisfiable), and (b) which of the generated clauses is selected in Step 3. The aim of a heuristic should be to discover a refutation as soon as possible.

Since the primal algorithm is oriented toward finding a solution and the dual algorithm toward finding a refutation, and since one does not know at the outset which will succeed, it is reasonable to combine the two approaches to obtain a primal-dual algorithm. This device is often used successfully in optimization. In the present context a primal-dual algorithm might begin with a dual phase by generating only a predetermined subset of the clauses that avoid blockage. If unsatisfiability results within this subset, instantiation would proceed as in the dual algorithm. Otherwise the algorithm would move to a primal phase by checking the solution for blockage. If blockage is found, instantiation would proceed as in the primal algorithm.

The rationale for the primal-dual approach is that an intelligent heuristic for generating clauses in the dual phase could rule out the most likely kinds of blockage with a relative small number of clauses. This reduces the number of primal iterations, but since the primal strategy operates as well, it is still possible to find a solution by luck before very many instantiations are generated.

References

1. Ausiello, G. and Italiano, P.: On line algorithms for polynomially solvable satisfiability problems, *J. Logic Programming* **10** (1991), 69–90.

2. Baumgartner, P.: FDPLL: A first-order Davis–Putnam–Logeman–Loveland procedure, in D. McAllester (ed.), *17th International Conference on Automated Deduction (CADE-17)*, Lecture Notes in Comput. Sci. 1831, Springer, 2000, pp. 200–219.
3. Bennaceur, H., Gouachi, I. and Plateau, G.: An incremental branch-and-bound method for the satisfiability problem, *INFORMS J. Comput.* **10** (1998), 301–308.
4. Carraresi, P., Gallo, G. and Rago, G.: A hypergraph model for constraint logic programming and application to bus drivers' scheduling, *Ann. of Math. and AI* **8** (1993), 247–270.
5. Chandru, V. and Hooker, J. N.: *Optimization Methods for Logical Inference*, Wiley, New York, 1999.
6. Chandru, V., Hooker, J. N., Rago, G. and Shrivastava, A.: A partial instantation based first order theorem prover, in *Proceedings, International Workshop on First Order Theorem Proving (FTP98)*, Vienna, Austria, 1988. http://www.logic.at/ftp98.
7. Chu, H. and Plaisted, D. A.: Semantically guided first-order theorem proving using hyper-linking, in A. Bundy (ed.), *Lecture Notes in Artificial Intelligence* 814, Springer-Verlag, Berlin, 1994, pp. 192–206.
8. Church, A.: *Introduction to Mathematical Logic, Part I*, Princeton University Press, 1944.
9. Cook, S. A.: The complexity of theorem proving procedures, in *Proceedings, 3rd ACM Symp. on the Theory of Computing*, 1971, pp. 151–158.
10. Davis, M.: Eliminating the irrelevant from mechanical proofs, in *Proceedings, 15th Symposium in Applied Mathematics*, Amer. Math. Soc., 1963, pp. 15–30.
11. Davis, M. and Putnam, H.: A computing procedure for quantification theory, *J. ACM* **7** (1960), 201–215.
12. Gallo, G. and Rago, G.: A hypergraph approach to logical inference for datalog inference, Tech. Rep. 28/90, Dip. Informatica, Università di Pisa, 1990.
13. Gallo, G. and Rago, G.: The satisfiability problem for the Schönfinkel–Bernays fragment: Partial instantiation and hypergraph algorithms, Tech. Rep. 4/94, Dip. Informatica, Università di Pisa, 1994.
14. Geoffrion, A. M.: Generalized Benders decomposition, *J. Optim. Theory Appl.* **10** (1972), 237–260.
15. Hooker, J. N.: Resolution vs. cutting plane solution of inference problems: Some computational experience, *Oper. Res. Lett.* **7** (1988), 1–7.
16. Hooker, J. N.: New methods for computing inferences in first order logic, *Ann. Oper. Res.* (1993), 479–492.
17. Hooker, J. N.: Solving the incremental satisfiability problem, *J. Logic Programming* **15** (1993), 177–186.
18. Hooker, J. N. and Yan, H.: Logic circuit verification by Benders decomposition, in V. Saraswat and P. Van Hentenryck (eds.), *Principles and Practice of Constraint Programming: The Newport Papers*, MIT Press, Cambridge, MA, 1995, pp. 267–288.
19. Jaumard, B., Stan, M. and Desrosiers, J.: Tabu search and a quadratic relaxation for the satisfiability problem, Ecole Polytechnique de Montréal, Succursale A, Case Postale 6079, Montreal, Quebec H3C 3A7, Canada.
20. Jeroslow, R. G.: Computation-oriented reductions of predicate to propositional logic, *Decision Support Systems* **4** (1988), 183–197.
21. Jeroslow, R. G.: *Logic-based Decision Support: Mixed Integer Model Formulation*, Ann. of Discrete Math. 40, North-Holland, Amsterdam, 1989.
22. Kagan, V., Nerode, A. and Subrahmanian, V. S.: Computing definite logic programs by partial instantiation, *Ann. Pure Appl. Logic* **67** (1994), 161–182.
23. Knight, K.: Unification: A multidisciplinary survey, *ACM Computing Surveys* **21** (1989), 93–124.
24. Lee, S.-J. and Plaisted, D. A.: Eliminating duplication with the hyperlinking strategy, *J. Automated Reasoning* **9** (1992), 25–42.

25. Lloyd, J. W.: *Foundations of Logic Programming,* 2nd edn, Springer-Verlag, Berlin, 1987.
26. Mendelson, E.: *Introduction to Mathematical Logic*, 3rd edn, Wadsworth & Brook/Cole Advanced Books & Software, Monterey, CA, 1987.
27. Plaisted, D. A. and Zhu, Y.: *Proceedings, 14th National Conference on Artificial Intelligence (AAAI-97)*, 1997.
28. Robinson, J. A.: A machine-oriented logic based on the resolution principle, *J. ACM* **12** (1965), 23–41.
29. Trick, M. and Johnson, D. S. (eds.): *Second DIMACS Challenge: Cliques, Coloring and Satisfiability*, Series in Discrete Mathematics and Theoretical Computer Science, Amer. Math. Soc., 1995.