An Instantiation-Based Theorem Prover for First-Order Programming

Erik P. Zawadzki
Computer Science Department
Carnegie Mellon University
Pittsburgh,PA
epz@cs.cmu.edu

Geoffrey J. Gordon
Machine Department
Carnegie Mellon University
Pittsburgh,PA
ggordon@cs.cmu.edu

André Platzer
Computer Science Department
Carnegie Mellon University
Pittsburgh,PA
aplatzer@cs.cmu.edu

Abstract

First-order programming (FOP) is a new representation language that combines the strengths of mixed-integer linear programming (MILP) and first-order logic (FOL). In this paper we describe a novel feasibility proving system for FOP formulas that combines MILP solving with instance-based methods from theorem proving. This prover allows us to perform lifted inference by repeatedly refining a propositional MILP. We prove that this procedure is sound and refutationally complete: if a formula is infeasible our solver will demonstrate this fact in finite time. We conclude by demonstrating an implementation of our decision procedure on a simple first-order planning problem.

1 INTRODUCTION

Mixed integer linear programming (MILP) is a ubiquitous framework for specifying optimization and decision problems. For example, MILPs are frequently used to solve problems in operations research and artificial intelligence. MILPs are reasonably expressive and can represent any $\mathcal{NP}$-complete problem. They admit natural formulations of many scheduling, resource allocation, VLSI, and planning problems (see, for example, Nemhauser and Wolsey [1988]).

While MILPs are excellent for dealing with problems of a propositional nature, they lack the machinery necessary for handling information about first-order classes and relations. One cannot claim in a MILP that “All cars follow the speed limit” without explicitly enumerating every car and separately claiming that every *particular* car follows the speed limit.

Appearing in Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011, Fort Lauderdale, FL, USA. Volume 15 of JMLR: W&CP 15. Copyright 2011 by the authors.

While macro-languages like AMPL [Fourer et al., 2002] can automate the tedious task of “unrolling” formulas for a given finite number of objects, they still cannot accommodate truly first-order statements. These representations are first-order but the reasoning is not. This is unfortunate for several reasons. First, even if a problem can be specified as a propositional program there may be a computational benefit with first-order reasoning. As an example, to verify the tautology:

$$\text{All humans are mortal} \wedge \text{All students are human} \quad (1)$$
$$\rightarrow \text{All students are mortal}$$

we do not want to enumerate every student (or worse, all 6.9 billion humans known at the time of writing) to look for a counterexample.

Second, there are problems that cannot be expressed by any finite set of propositional statements. For example, in a planning problem, both time and resource limits are potentially unbounded quantities. Planners often side-step this question by insisting on upper bounds on time and resources, but failing to solve a bounded-horizon or bounded-resource truncation proves nothing about the actual unbounded problem. It is always possible that with one more time step, or one additional vehicle, every goal in the plan can be satisfied. Along with demonstrating good plans, being able to prove nonexistence of suitable plans in decision problems is one of the high-level goals of artificial intelligence—this is something that purely propositional models cannot always capture.

These limitations in MILP can be addressed by switching to a more expressive language like first-order logic (FOL). FOL lets us specify a problem in terms of classes and relations, and reason about these classes and relations directly—we can do *lifted* reasoning. Lifted reasoning lets us work with statements about infinite or unknown numbers of objects (*e.g.*, entity resolution problems), and may also have computational benefits. For example, we can prove the validity of (1) in a small number of steps, independent of the number of students or humans. Unlike MILP, however, FOL itself is strictly boolean and has no built-in arithmetic.

First-order programming is a new representation suggested in Gordon et al. [2009] that combines the expressive power of FOL with MILP's ability to represent linear functions of real and integer-valued predicates. These real and integer-valued predicates allow some statements to be more compactly represented in FOP than in FOL.

For example, if one had n atoms and wanted at least k of them to be true (*i.e.*, have value 1), then this can be stated in a FOP formula of length $\mathcal{O}(n)$. An equivalent formula in FOL would require a disjunction of $\binom{n}{k}$ conjunctions of length k. This is exponentially longer in FOL than FOP, yet no properties have a shorter representation in FOL than FOP since there are straightforward translations from FOL into FOP that do not increase formula length [Gordon et al., 2009].

However, a language like FOP has no use if we cannot perform inference in it. Up until now there was no implemented lifted reasoning procedure for FOP. (One was proposed, but not implemented, in the original FOP paper). So, we suggest a new simple way to do lifted inference in the integer fragment of FOP.

Our new approach is radically different than the one previously suggested. The previous approach uses lifted Gomory cuts, a technique that generalizes resolution in FOL. Our approach, on the other hand, never recombines clauses to form either a Gomory cut or resolvent. We therefore expect our procedure to yield smaller proofs than methods based on lifted Gomory cuts as it will not duplicate clause instances (see, for example, Lee and Plaisted [1992]). Our algorithm is also easier to implement than methods based on lifted Gomory cuts. (Our current solver, however, is not automated and requires a human to control search.)

Our inference procedure for FOP draws on ideas from instantiation-based methods in theorem proving [Ganzinger and Korovin, 2003, Korovin, 2009] and MILP. It aggressively propositionalizes a FOP formula, solves the resulting propositional formula, attempts to lift a model of the propositional formula, and instantiates clauses of the FOP formula to refine the propositional formula. We use a MILP solver to find propositional models and check termination conditions.

In this paper we make the following contributions: first, we suggest a new algorithm for performing lifted inference in the integer fragment of FOP; second, we implement this solver and demonstrate its behavior on an example problem; third, we prove the soundness and completeness of our approach. More generally, our work connects theorem proving principles for FOL with modern optimization techniques—we are revisiting some traditional artificial intelligence goals armed with more recent tools and results.

We describe our inference procedure as follows. First we look at related work that inspired our approach. Next, we briefly discuss the syntax and semantics of an important normal form of FOP. Then we state the FOP inference problem that we are interested in—checking whether a formula is feasible or not. After this background material, we present our main result: the algorithm and proof that our algorithm is both sound and refutationally complete. After giving these results we demonstrate an inference problem that can be solved with our solver. We finish by indicating promising directions for future work.

2 RELATED WORK

We are not the first to describe an instantiation-based theorem prover for lifted reasoning. Indeed our application of instantiation-based methods in FOP is inspired by work on instantiation-based provers in FOL and its fragments. Such solvers take advantage of Herbrand's theorem: a conjoined set of first-order formulas is unsatisfiable iff there exists a finite set of ground instances of these formulas that is also unsatisfiable.

The naïve procedure of sweeping through all possible finite ground sets is sound and refutationally complete, but impractical. In particular the naïve procedure ignores the interesting class and relational structure of the original formula. Most automated instantiation-based solvers use features of the instantiated sets of clauses and their propositional models to guide which additional instantiations should be generated. These solvers combine effective instance generation heuristics with redundancy criteria to efficiently reason about how to instantiate the formula.

There are two broad families of lifted instantiation-based solvers. The first class tightly integrates propositional reasoning with instance generation in a single solver. Examples of these tightly-integrated solvers include model evolution calculus [Baumgartner and Tinelli, 2003, 2008], its precursor first-order DPLL [Baumgartner, 2000], and disconnection tableaux [Letz and Stenz, 2001, 2007].

The second class—to which the solver described in this paper is most closely related—treats a propositional SAT solver as a black-box oracle for determining the satisfiability of a propositional formula, and perhaps also for providing a propositional model to semantically guide further instantiation. An advantage of this class of solvers is that the latest and fastest propositional solver can always be plugged into the solver—implementations of this second class of solver get faster every year without even touching them because SAT solvers are improving. Additionally, solvers in this second category tend to be simple and flexible since they delegate all propositional issues to the black-box solver. However, these solvers are—by design—uninterested in applying fine-grained control to propositional model finding. As a result they do not have the same level of information or design freedom as the first class of solvers.

Examples of this second class of solver include Jeroslow's algorithm [Jeroslow, 1988], Hooker's improvement of it [Hooker et al., 2002], and the related Inst-Gen line of work [Ganzinger and Korovin, 2003, Korovin, 2009]. Our algorithm adapts Inst-Gen-style reasoning to FOP. However, unlike algorithms for FOL, the black-box oracle that we use is an ILP solver and not a SAT solver.

FOP is tailored for first-order optimization and planning problems, but it has deep connections to theory reasoning in FOL—a first-order variant of satisfiability modulo theories (*e.g.*, Nieuwenhuis et al. [2006]). In these languages FOL is enriched by (*e.g.*, arithmetical) background decision procedures. Of particular interest is FOL augmented by the theory of linear integer arithmetic and uninterpreted functions—FOL(UFLIA). We hope that our approach will help theory reasoning in FOL, and we expect to draw inspiration from their research (*e.g.*, Ganzinger and Korovin [2006], Korovin and Voronkov [2007], and Baumgartner et al. [2008]).

For example, model evolution (ME) calculus was extended to reason about a fragment of the theory of linear arithmetic, forming the $\mathcal{ME}$(LIA) calculus [Baumgartner et al., 2008]. $\mathcal{ME}$(LIA) is especially relevant to us since FOP also integrates linear arithmetic.

The most important difference between $\mathcal{ME}$(LIA) and FOP is that predicates in the $\mathcal{ME}$(LIA) fragment are binary valued while predicates in FOP can take any value in a bounded continuous or discrete interval. While both logics do linear arithmetic, they occur in entirely different places: integers are *objects* in $\mathcal{ME}$(LIA), and *values* in FOP. Indeed, one could imagine FOP modulo LIA, where linear arithmetic could occur at both the value (predicate) level and the object (function) level.

Researchers have also investigated SAT modulo LIA (e.g., Faure et al. [2008]). The discussion above about $\mathcal{ME}$(LIA) applies to SAT modulo LIA as well: linear arithmetic occurs at the object level and not the value level. However, unlike $\mathcal{ME}$(LIA), SAT modulo LIA is purely propositional and unable to do lifted reasoning.

3 FIRST-ORDER PROGRAMMING

In this paper we will assume that the FOP formula that we wish to reason about is given in a special format known as $\wedge$-normal form ($\wedge$NF). This is not a restriction, since every FOP formula has an equivalent $\wedge$NF representation, but focusing on $\wedge$NF formulas simplifies our analysis and the description of FOP.

We describe FOP briefly in this section; see Gordon et al. [2009] for a complete description.

3.1 Syntax

Just as in FOL, FOP has terms that represent objects and formulas that represent values. Each FOP predicate can take values in some compact interval of the reals or integers. This interval is called the *range* of the predicate and is denoted Range_P. We restrict to integer FOP, so $\text{Range}_P \subset \mathbb{Z}$. A predicate applied to zero or more objects is an *atom*. Like in FOL, there are n-ary functions that map objects to objects. In FOP, *scalars* are literals with a predefined value and, just as in FOL, 0-arity functions are called constants. To avoid technicalities, we assume there is at least one constant symbol.

There are four binary operators and one quantifier in a $\wedge$NF formula. The binary operators are scalar multiplication (denoted $\cdot$), addition ($+$), maximization ($\vee$), and minimization ($\wedge$), and the quantifier is minimization over variables ($\bigwedge_x$).

A generic $\wedge$NF formula looks like:

$$\mathbf{F} = \bigwedge_{\text{Var}} (C_1 \wedge \ldots \wedge C_n)$$
$$C_i = \Sigma_{i1} \vee \ldots \vee \Sigma_{im}$$
$$\Sigma_{ij} = \kappa_{ij1} \cdot P_{ij1} + \ldots + \kappa_{ijk} \cdot P_{ijk}$$

The top level formula is called a $\wedge$-clause, the second-level formulas are $\vee$-clauses, then we have Σ-clauses which are linear combinations of literals. Here $\kappa_{ijk} \in \mathbb{Q}$ is an optional scalar, P_{ijk} is an atomic proposition, and Var is the set of free variables in the $\vee$-clauses $C_1, \ldots C_n$. We call any formula without variables *ground*.

Because $\vee$, $\wedge$, and $+$ commute and associate with themselves, we use notation for the clauses as if they were sets. So $C_i \cap C_j$ will denote all the Σ-clauses that are in both $\vee$-clauses C_i and C_j.

3.2 Semantics

A model is a triple $\mathbf{M} = \langle O, F, V \rangle$, where O is a non-empty list of objects, F is a list of function tables, and V is list of tables of predicate values. Here V assigns a total map $V_P : O^n \rightarrow \text{Range}_P$ to each predicate symbol P with arity n. F defines a similar assignment of total maps to every function symbol.

A model for a formula maps every ground atom to a value in its range, and the values of compound formulas are built from these values. In every model of a ground $\wedge$-clause it has the value of the least-valued $\vee$-clause; each ground $\vee$-clause takes the value of the greatest-valued Σ-clause; and each ground Σ-clause is just a linear combination of the values of its atoms.

A formula that is $\bigwedge$-quantified for some variable x takes the minimum value over all substitutions of an object in the domain O for x. This means that in every model a $\wedge$NF formula takes the value of its least-valuable grounding. We call this grounding the *minimal instance* of a formula for a particular model. This minimal instance might not exist when predicates can take on real values—there might be no minimum, only a convergent sequence. However, this minimal instance always exists in FOP's integer fragment.

We denote the value of a formula $\mathbf{F}$ in a model $\mathbf{M}$ by $value(\mathbf{F}, \mathbf{M})$. We will denote the quantity $\sup_{\mathbf{M}} value(\mathbf{F}, \mathbf{M})$ as $value(\mathbf{F})$.

As an example of a FOP formula in $\wedge$NF, consider the following definition of the equivalence predicate '=' with range $\{0, 1\}$. We can do this by constructing a formula that is non-negative iff the predicate is reflexive, symmetric, and transitive. Indeed, the idea of non-negativity is important to our notion of inference and we will introduce some shorthand notion to express it. By $P(x) \geq c$ we will mean $P(x) - c$ (the latter FOP formula is non-negative iff the former condition is met) and by $P(x) \leq c$ we will mean $c - P(x)$. Therefore, we can insist that '=' is an equivalence relation with following subformula:

$$(i = i) \geq 1 \tag{2}$$
$$(i = j) + (j = i) \leq 0 \vee (i = j) + (j = i) \geq 2 \tag{3}$$
$$\begin{aligned}(i = j) + (j = k) + (i = k) \leq 1& \\ \vee\ (i = j) + (j = k) + (i = k) \geq 3.&\end{aligned} \tag{4}$$

Here, each labeled line is a $\vee$-clause; we join them implicitly by $\wedge$ to form a $\wedge$NF formula.

Since $(i = i) \in \{0, 1\}$, the first $\vee$-clause asserts that $(i = i)$ must be 1 if the formula as a whole is to be non-negative. The second asserts that it is symmetric since either both $(i = j)$ and $(j = i)$ must have value 1, or neither can. The final clause asserts transitivity—either they are all equal or at most one is.

Since a $\vee$-clause is a maximum over Σ-clauses, in every model there is at least one Σ-clause that has the same value as the $\vee$-clause. Covering sets—sets that contain at least one Σ-clauses for every $\vee$-clauses—play an important role in how we think about the value of the formula. As a result we define some special terminology for them.

Definition 1 (Covering sets, active atoms, and tightness)**.** *Let $\mathcal{C}$ be a set of $\vee$-clauses. If $\mathcal{S}$ is a set of Σ-clauses such that for every $C \in \mathcal{C}$ it is the case that $\mathcal{S} \cap C \neq \emptyset$, then $\mathcal{S}$ is a* covering set*. A covering set for a ground formula is* tight *with respect to a model* $\mathbf{M}$ *if the value (in $\mathbf{M}$) of each Σ-clause is equal to the value of the $\vee$-clause that contains it.*

The set of all atomic propositions in a formula or set of formulas $\mathbf{F}$ is denoted by $\mathcal{A}(\mathbf{F})$. For a covering set S we will refer to all atoms in $\mathcal{A}(S)$ as the active atoms*.*

4 INFERENCE

Given a formula $\mathbf{F}$ in $\wedge$NF, there are a number of questions that can be asked about its value. One of the most basic is whether the formula has a model with a non-negative value. We call any such formula *feasible* or *satisfiable*, and this notion generalizes the FOL notion of satisfiability. Using feasibility testing as a primitive, we can define FOP notions of entailment (see Gordon et al. [2009]). We can also check if a formula $\mathbf{F}$ has a particular value $\mathcal{V}$ by checking if $\mathbf{F} - \mathcal{V} \wedge \mathcal{V} - \mathbf{F}$ is feasible.

For any finite ground FOP formula $\underline{\mathbf{F}}$, we can find its value by encoding it as a MILP—*e.g.* the following formulation—and giving it to a MILP solver.

$$\begin{aligned}\max\quad & \mathcal{V} \\ \text{s.t.}\quad & \left(\sum_{k \in \mathbb{I}_{\Sigma_{ij}}} \kappa_{ijk} \cdot p_{ijk}\right) + \mathcal{U}(1 - d_{ij}) \geq \mathcal{V} \\ & \sum_{j \in \mathbb{I}_{C_i}} d_{ij} \geq 1 \\ & p_{ijk} \in \text{Range}_{ijk} \\ & d_{ij} \in \{0, 1\}\end{aligned}$$

The MILP, denoted $\mathsf{MILP}(\underline{\mathbf{F}})$, for finding the value of a ground formula $\underline{\mathbf{F}}$. The MILP variable $\mathcal{V} \in \mathbb{R}$ represents $value(\underline{\mathbf{F}})$. The first type of constraint represents each Σ_{ij}. The $\mathbb{I}_x$ are sets that index the elements x contains so $\mathbb{I}_{\Sigma_{ij}}$ indexes all of its constituent literals $\kappa_{ijk} \cdot p_{ijk}$. The κ_{ijk} are scalars, and so are coefficients of the predicate variables (the p_{ijk}), which can be assigned any value in their range. The constant $\mathcal{U}$ is some sufficiently large number such that the binary fresh MILP variable d_{ij} can be set to 0 and make the bound on $\mathcal{V}$ for any particular Σ_{ij} trivial regardless of the (bounded!) values of the other variables. The d_{ij} indicate a covering set for the maximal model.

While we cannot determine the value of a non-ground FOP formula $\mathbf{F}$ directly by submitting it to an ILP solver, we can show that the value for any instantiation of $\mathbf{F}$—and in particular any ground instance—is an upper-bound on the value of the original formula.

Definition 2 (Instantiation, renaming)**.** *A formula F* instantiates *another formula F', written $F' \succeq F$, if $F = F'\theta$ for some substitution θ. We also say F'* generalizes *F. We write $F \succ F'$ if $F \succeq F'$ but $F' \not\succeq F$ (*strict instantiation*). Non-strict instantiation is also called* renaming.

If $\mathcal{F}$ is a set of formulas then a most specific generalization *(MSG) of F in $\mathcal{F}$, denoted* msg$(F, \mathcal{F})$*, is a set $\mathcal{G}$ of all elements $G \in \mathcal{F}$ such that $G \succeq F$ and there is no more specific element $G' \in \mathcal{F}$ such that $G' \succeq F$ and $G \succ G'$. MSGs are unique up to renaming.*

Proposition 1 (Instance upper-bounding)**.** *For all formulas $\mathbf{F}$, instantiations $\mathbf{F}\theta$ and models $\mathbf{M}$, $value(\mathbf{F}, \mathbf{M}) \leq value(\mathbf{F}\theta, \mathbf{M})$.*

Proof. All free variables of $\mathbf{F}$ are $\bigwedge$-quantified. Instantiation can only restrict which objects the variables can refer to, so by definition of $\bigwedge$, instantiation can only increase the value of $\mathbf{F}$. □

It is also easy to show that adding more clauses to the top-level $\wedge$-clause can only drive down its value.

Proposition 2 (Subproblem upper-bounding)**.** *For all formulas* $\mathbf{F}, C$*,* $value(\mathbf{F}, \mathbf{M}) \geq value(\mathbf{F} \wedge C, \mathbf{M})$.

Proof. Since the model $\mathbf{M}$ is fixed, adding an additional clause to the top-level minimization ($\wedge$) cannot increase the value of the formula. □

While every instance is an upper-bound, we will frequently consider a particular grounding instantiation, $\flat$, where $\flat$ is overloaded to mean both some fresh constant not in $\mathbf{F}$ and the substitution where all variables are replaced with $\flat$. A corollary of Proposition 1 is that the value of the special ground instance $\mathbf{F}\flat$ is an upper bound on the value of $value(\mathbf{F})$.

Corollary 1. *For all formula* $\mathbf{F}$*,* $value(\mathbf{F}) \leq value(\mathbf{F}\flat)$.

The corollary shows us that we can bound the value of the first-order formula by the value of its instances and, in particular, the instance generated by the substitution $\flat$. An arbitrary model that maximizes the value of $\mathbf{F}\flat$ will be frequently used in the following sections, and we will denote this special model $\mathbf{M}_\flat$; we can find it using a MILP encoding.

The reason $\mathbf{F}\flat$ is interesting is that it provides a template for constructing a first-order model of $\mathbf{F}$. We do this by employing a lifting procedure: in a lifted model $\mathbf{M}$ we assign, to each of the (infinitely many) ground atoms, the value that its most specific generalization in $\mathbf{F}$ takes in $\mathbf{M}_\flat$. So if we consider the ground atom P, it takes value $\mathbf{I}(P) = value(Q\flat, \mathbf{M}_\flat)$ where $Q = \text{msg}(P, \mathcal{A}(\mathbf{F}))$.

As an example, suppose $\mathbf{F} = P(x) \geq 1 \wedge P(x) \leq 1$. The maximal ground model $P(\flat) = 1$ suggests that we set $P(x) = 1$, and this is also the maximal model for $\mathbf{F}$. Indeed we will show in Lemma 1 that under certain conditions the lifted model $\mathbf{M}$ has the same value as the $\mathbf{M}_\flat$. This is an attractive observation since we can find $\mathbf{M}_\flat$ efficiently.

In general, the $\flat$-grounding lacks some of the constraints that the minimal instance has. This is because, unlike in $\mathbf{F}\flat$, a formula's minimizing ground instances may force unifiable—but syntactically distinct—terms to take the same value. For example, consider the formula $\mathbf{F} = P(a) \geq 1 \wedge P(x) \leq 0$, where P's range is $\{0, 1\}$. The maximizing model for $\mathbf{F}\flat$ sets $P(a) = 1$ and $P(\flat) = 0$, but there is no way to lift this model to a non-negative model of $\mathbf{F}$ because under the substitution $[x \mapsto a]$ we seem to want both $P(a) = 0$ and $P(a) = 1$.

Whenever $\mathbf{M}_\flat$ assigns unifiable atoms different values, one has to be careful about the value of these atoms in any instantiation that does unify them. Such unifiable pairs of atoms play an important role in both the above example and our actual inference procedure. We call these pairs *discordant.*

Definition 3 (Discordant pairs, witnesses)**.** *Let* $\mathbf{F}$ *be a formula and* S *be a tight covering set w.r.t. a model* $\mathbf{M}_\flat$ *of* $\mathbf{F}\flat$*. A* discordant pair *in* $\mathbf{F}$ *is a pair of propositions* $P, Q \in \mathcal{A}(F)$ *such that* $P\flat, Q\flat \in \mathcal{A}(S)$*,* P *and* Q *unify, but* $value(P\flat, \mathbf{M}_\flat) \neq value(Q\flat, \mathbf{M}_\flat)$*.*

The most general unifier θ *of* P *and* Q *is the* witness *of this pair.*

The basic intuition of our approach is as follows. Whenever we have a discordant pair (P, Q) inspired by $\mathbf{M}_\flat$, then the grounding $\mathbf{F}\flat$ must have missed the fact that P and Q can be forced to assume the same value in some instantiations. We remedy this problem by ensuring that $\mathbf{F}$ mentions their unification $P\theta$ explicitly: then when we try to lift $\mathbf{M}_\flat$, any atom that unifies with both P and Q will take its value from the more specific $P\theta\flat$ instead of from $P\flat$ or $Q\flat$. To take advantage of this intuition, we present the *semantic instance generation* rule, which generates additional clauses to eliminate the connection of a discord.

4.1 Semantic instance generation

We can resolve discordant pairs by generating additional clauses that ensure that any instance that unifies the discordant atoms assigns them consistent values in the $\flat$-model. For example, consider

$$\mathbf{F} = Q(b) = 0 \wedge [Q(x) = 0 \vee Q(x) = 1 \vee Q(x) = 2],$$

where Q's range is $\{0, 1, 2\}$. Suppose that $value(Q(\flat), \mathbf{M}_\flat) = 2$ and $value(Q(b), \mathbf{M}_\flat) = 0$; then if we take $Q(x)$'s value to be the same as $Q(\flat)$'s we can no longer guarantee that we have a maximal value for $\mathbf{F}$ since the special case when $[x \mapsto b]$ may not be properly handled. However, we can be completely confident after generating a new instance $Q(b) = 0 \vee Q(b) = 1 \vee Q(b) = 2$ that forces the propositional solver to consider the special case explicitly. *Semantic instance generation* is an inference rule that accomplishes this.

Definition 4 (Semantic instance generation rule)**.** *The* semantic instance generation rule *(SIG) is*

$$\frac{C_i \vee \Sigma_i \qquad C_j \vee \Sigma_j}{(C_i \vee \Sigma_i)\theta \qquad (C_j \vee \Sigma_j)\theta.}$$

The clauses on the top are the premises of this inference rule, and the clauses on the bottom are the conclusions. Both premises must be $\vee$*-clauses in* $\mathbf{F}$*, where* C_i *and* C_j *are the (possibly empty) sets containing the remaining* Σ*-clauses in their respective* $\vee$*-clauses.* $\Sigma_i\flat$ *and* $\Sigma_j\flat$ *must be members of a covering set* S *that is tight with respect to a maximal model* $\mathbf{M}_\flat =$

$\arg\max_{\mathbf{M}} value(\mathbf{F}\flat, \mathbf{M})$. *Additionally, there must exist propositions in the intersection of each of these Σ-clauses and the covering set, say $P \in \mathcal{A}(\Sigma_i) \cap \mathcal{A}(S)$ and $P' \in \mathcal{A}(\Sigma_j) \cap \mathcal{A}(S)$, that are discordant in $\mathbf{M}_\flat$ with θ as their witness*—i.e. $value(P\flat, \mathbf{M}_\flat) \neq value(P'\flat, \mathbf{M}_\flat)$ *and* $P\theta = P'\theta$.

For any two premises Q and R, we will denote their set of conclusions as $SIG(Q, R)$.
A simple consequence of this definition is that at least one of the conclusions must say something new (the conclusion is not just a renaming of its premise).

Proposition 3. *At least one of the conclusions must strictly instantiate its premise and cannot just be a renaming.*
Proof. If not then the mgu of P and P' is a renaming, and so $P\flat = P'\flat$. Therefore they have the same value in the maximal model and cannot be discordant. □
The conclusions say something new, but they are still a consequence of the respective premises. Adding the conclusions of SIG to the original formula never alters the formulas value, so it is safe to apply. Intuitively this is because SIG is merely explicitly stating a property that was already implied by the original formula.

Proposition 4 (SIG preserves value)**.** *Let $C\theta$ and $D\theta$ be the conclusions of an application of SIG to the FOP formula $\mathbf{F}$. Then $value(\mathbf{F}) = value(\mathbf{F} \wedge C\theta \wedge D\theta)$.*
Proof. Let $\mathbf{F}'$ be $\mathbf{F} \wedge C\theta \wedge D\theta$. The value of $\mathbf{F}'$ cannot be greater than that of $\mathbf{F}$ by Proposition 2.

The value for $\mathbf{F}'$ cannot be less, since $C\theta$ (or $D\theta$) is just an instantiation of some $\vee$-clause C of $\mathbf{F}$: suppose the value of $\mathbf{F}'$ in model $\mathbf{M}$ attains its minimal value in $C\theta$ after applying some grounding substitution σ. Then there is a model for $\mathbf{F}$ with the same value obtained after applying grounding substitution $\theta\sigma$. □
Note while the value of $\mathbf{F}$ is unchanged, the value of $\mathbf{F}\flat$ can drop, but does not have to.

4.2 FOP Feasibility Algorithm

With SIG, our results about the ground instance $\mathbf{F}\flat$, and our ILP for finding the maximal value of any ground formula we can construct an algorithm for determining the feasibility of a FOP formula. It is described in Algorithm 1.

We require that our instance selection policy is fair—it cannot ignore a potential instance in I forever. This restriction is required for the completeness results that we present in the next section.

Definition 5 (Fairness)**.** *A selection rule is* fair *if no application of SIG is possible infinitely often.*

Fairness is not a particularly onerous requirement and there are simple policies that are fair. An example of a fair policy is the chronological selection policy where we select the oldest available option. (The age of an option is the first time-step that it occurs as an option).

Algorithm 1 Feasibility algorithm

```
1: while $true$ do
2:     $\mathbf{M}_\flat$ = solution of $\mathsf{MILP}(\mathbf{F}\flat)$
3:     {Hence $\mathbf{M}_\flat = \arg\max_{\mathbf{M}} value(\mathbf{F}\flat, \mathbf{M})$}
4:     if $value(\mathbf{F}\flat, \mathbf{M}_\flat) < 0$ then
5:         return $value(\mathbf{F}) < 0$ ;
6:     end if
7:     Using $M_\flat$, obtain a covering set $S$
8:         and list of discordant atoms $A$;
9:     if $A = \emptyset$ then
10:        return $value(\mathbf{F}) \geq 0$ ;
11:    end if
12:    $I = \emptyset$ ;
13:    for $(P, Q) \in A$ do
14:        Gather new instances $I = I \cup SIG(P, Q)$;
15:    end for
16:    Select a non-empty subset $I'$ of $I$
17:        using a fair selection rule;
18:    $\mathbf{F} = \mathbf{F} \wedge I'$;
19: end while
```

We will now show that Algorithm 1 is both sound and refutationally complete.

Definition 6 (Soundness and refutational completeness)**.** *A feasibility procedure for FOP is* sound *if it never reports the wrong sign for a formula.*

A procedure is refutationally complete *if it eventually declares that a formula with negative values is negative.*

4.3 Soundness

We will first prove that our algorithm is sound. This theorem relies on two properties: the first is that the value for $\mathbf{F}\flat$ is always an upper-bound on the value of $\mathbf{F}$, and the second is that if $\mathbf{M}_\flat$ is free of discord then it can be used as a template for constructing a lifted model of $\mathbf{F}$—in this case it is lower-bound and so the bounds are tight. We already proved the first property in Corollary 1. We will now prove the second property.

Lemma 1 (Lifting)**.** *If there are no new discordant atoms in some tight covering set S, then $value(\mathbf{F}\flat) = value(\mathbf{F})$.*

The above lemma is proved in our supplemental material. With this result we can finish the proof that our algorithm is sound.

Theorem 1 (Soundness of inference)**.** *Algorithm 1 never reports an incorrect sign for the value for a formula.*

Proof. By Proposition 4, every application of SIG preserves the value of the formula, and this is the only way that we modify the original formula. By Proposition 1 we can safely conclude that the value of a formula is negative if the value for $\mathbf{F}\flat$ is ever negative. Since line 5 is the only time that we declare the value of a formula to be negative, the inference procedure never declares a non-negative formula to be negative.

We only declare a formula to be non-negative when there is a non-negative model for $\mathbf{F}\flat$ and there are no new discordant atoms. By Lemma 1, when there are no novel discordant atoms the value of $\mathbf{F}\flat$ is a lower bound. Since line 10 is the only time that we declare our formula to be non-negative, our algorithm never declares a negative formula to be non-negative. □

4.4 Completeness

In this section we will demonstrate that our solver is refutationally complete. The key property that we use is this: if $value(\mathbf{F}) < 0$ and yet $value(\mathbf{F}\flat) \geq 0$, then there is some discordant pair that has not yet been used to generate an instance. As long as we have a fair way of selecting these discordant pairs, we will show that the procedure only needs a finite number of SIG inferences to find a refutation—our algorithm eventually finds an application of SIG that drives $value(\mathbf{F}\flat)$ below zero.

Lemma 2 (Locality of subproblem discord)**.** *If S is a ground subproblem of $\mathbf{F}$ such that $value(S) < 0$, and if $value(\mathbf{F}\flat) \geq 0$, then the MSG of S in $\mathbf{F}$, $S' = \mathrm{msg}(S, \mathbf{F})$, has a novel discordant pair (P, Q).*

Additionally, the conclusions $(C^P\theta, C^Q\theta)$ of SIG on this pair are members of $S'' = \mathrm{msg}(S, \mathbf{F} \wedge C^P\theta \wedge C^Q\theta)$, the MSG of S in the augmented formula.

A proof of this lemma is in our supplemental material.

This proves that there is always a discordant pair that we can try. We now show that there is a finite sequence of these discordant pairs that eventually drive down the value of $\mathbf{F}\flat$ below zero.

Lemma 3. *If S is a finite and ground subproblem of $\mathbf{F}$ that has negative value, then there exists a finite sequence of MSG $\langle S_0, \ldots, S_n\rangle$ obtained by SIG such that S_i is the MSG of S in $\mathbf{F}$ after i rounds of SIG and $value(S_n\flat) < 0$.*

A proof of this lemma is in our supplemental material.

Putting these two lemmas together proves that our algorithm is complete.

Theorem 2 (Refutational completeness of inference)**.** *If $value(\mathbf{F}) < 0$, the inference procedure will report that after finite fair applications of SIG.*

Proof. Suppose that a formula $\mathbf{F}$ has negative value. Then, by the completeness of the naïve algorithm for FOP [Gordon et al., 2009] there is a finite subset of ground instances, namely S, that exhibits this negative value. Since they are ground instances $S = S\flat$ so $value(S\flat) < 0$.

By Lemma 3 there exists a finite sequence of SIG applications that eventually generates a subproblem S' such that $value(S'\flat) < 0$. Therefore, if the policy for applying SIG is fair our inference procedure will eventually report that $value(\mathbf{F}) < 0$. □

5 EXAMPLES OF REASONING

In this section we present a sample of reasoning in our system given a simple vehicle planning problem[1]. In a vehicle planning problem there are three major components. The first description of the world (*e.g.* obstacles and physical dynamics), the second is a list of N vehicles with different characteristics (*e.g.* acceleration and turn radius), and the final is a description of the goals. The goals could be a number of waypoints with logical, vehicle and temporal constraints over them. For example, waypoint ω_i could be only satisfied by a subset of vehicles (say ones equipped with a winch), and it must be visited before ω_j.

We present a simplified version of this general vehicle planning problem. In our specific instance we have a single vehicle and an uncertain description of the world, due to (say) extremely noisy satellite information. We are able to determine that there are at least eight equivalence classes. Again, because of noisy information we do not know which locations are accessible from other locations, but we do have some concrete information about which nodes are not accessible. We have a single goal: to go from one location to another.

This is not just propositional connectivity problem on eight nodes since the FOP formula given actually describes non-empty equivalence classes and some relationships between them. In the special case of a finite model with only eight objects it is easy to show that this formula is negative using a standard connectivity algorithm. However, we prove something more sophisticated: that there cannot exist any model—even of infinite size—that makes the formula non-negative.

There are eight representative objects denoted by constants $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}$, and $\mathbf{h}$. Each constant is distinct (*e.g.* $-(\mathbf{a} = \mathbf{b})$). We refer to the equivalence class of $\mathbf{h}$—the set of all objects equivalent to $\mathbf{h}$—as $[\mathbf{h}]$. There are additional clauses in the formula that ensures that all relations have consistent value modulo equivalence, so $\texttt{Link}(i, \mathbf{b})$ must have the same value as $\texttt{Link}(\mathbf{a}, \mathbf{b})$ if $(i = \mathbf{a})$. The domain is not exhaustively partitioned into these eight classes and objects are not compelled to be a member of them.

After these preliminary clauses, we give two more interesting predicates. `Link` is a binary predicate between objects in the classes, and `Path` is a 3-ary predicate built on top of `Link` that describes the length shortest path between two classes.

`Link` is underspecified in our problem. The only thing that we know about it is that all links to objects in $[\mathbf{g}]$ and $[\mathbf{h}]$ must have come from $[\mathbf{g}]$ or $[\mathbf{h}]$:

[1]For a more extensive example, see our supplemental material.

$$-\texttt{Link}(i,\mathbf{g}) \vee (i=\mathbf{g}) - 1 \vee (i=\mathbf{h}) - 1 \quad (5)$$
$$-\texttt{Link}(i,\mathbf{h}) \vee (i=\mathbf{g}) - 1 \vee (i=\mathbf{h}) - 1. \quad (6)$$

If there is a shortest path between two classes, they either must be in the same class (and takes no links to get there) or there must be a decomposition of that path that involving one of these links:

$$\begin{aligned} &-\texttt{Path}(i,j,t) && (7)\\ &\quad \vee (t=\mathbf{z}) + (i=j) - 2 \\ &\quad \vee \texttt{Path}\,(i, p_N(j,t), p_T(t)) + \texttt{Link}\,(p_N(j,t), j) - 2 \\ &p_N(j,t) = \mathbf{a} \vee \ldots \vee p_N(j,t) = \mathbf{h}. && (8)\end{aligned}$$

Here, $p_N(i,t)$ and $p_T(t)$ are Skolem functions. So $p_N(i,t)$ is allowed to be any object that is linked to the destination and has a shortest path itself. The temporal Skolem function $p_T(t)$ refers to the time object before—*e.g.* time step $t-1$. $\mathbf{z}$ is the 'zero' time constant that represents needing no links. We establish binary relation '$\geq$' that represents a standard partial ordering over $\mathbf{z}$ and $p_T(t)$.

Finally, we need to eliminate the possibility of a node giving a circular explanation of its position—we bar infinite cycles. We do this by insisting that all shortest paths between two objects must have the same length.

$$\begin{aligned} &(t \geq t') + (t' \geq t) - 2 && (9)\\ &\quad \vee 1 - \texttt{Path}(i,j,t) - \texttt{Path}(i,j,t'). \end{aligned}$$

Now we add the contradictory ground fact: we can connect $\mathbf{a}$ to $\mathbf{h}$ in $\mathbf{T}$ time.

$$\texttt{Path}(\mathbf{a},\mathbf{h},\mathbf{T}) - 1 \quad (10)$$

A sketch of our human-guided proof:

1. Instantiate line 7 to insist that if we are in $\mathbf{h}$, we came from somewhere, namely $p_N(\mathbf{h},\mathbf{T})$
2. From line 6 we force $p_N(\mathbf{h},\mathbf{T})$ to be g or h
3. From line 9 show that h's shortest path cannot be both $\mathbf{T}$ and $p_T(\mathbf{T})$. This forces $p_N(\mathbf{h},\mathbf{T}) = \mathbf{g}$.
4. From line 7 we insist that $\mathbf{g}$ has a predecessor $p_N(\mathbf{g}, p_T(\mathbf{T}))$.
5. From line 5 we force $p_N(\mathbf{g}, p_T(\mathbf{T})) = \mathbf{g}$ or $p_N(\mathbf{g}, p_T(\mathbf{T})) = \mathbf{h}$
6. From line 9 we exclude $p_N(\mathbf{g}, p_T(\mathbf{T})) = \mathbf{g}$ as a possibility since $p_T(\mathbf{T}) \geq p_T(p_T(\mathbf{T}))$.
7. From line 9 we exclude $p_N(\mathbf{g}, p_T(\mathbf{T})) = \mathbf{h}$ as a possibility since $\mathbf{T} \geq p_T(p_T(\mathbf{T}))$. Therefore there are no consistent values for $p_N(\mathbf{g}, p_T(\mathbf{T}))$ and we are done.

There are additional clauses and proof steps omitted for brevity.

This proves that there are no non-negative models of this formula. Notice that our proof applies for any predicates that satisfy the given properties, not just ones that the representation has explicitly declared to be nodes and edges. This is powerful, because the human encoder might not realize that their problem is reducible to showing that a graph is partitioned.

This simple planning example shows that our method of reasoning works in FOP. This is important, because infeasibility (negativity) may manifest in non-obvious ways that may be difficult to detect with a purely propositional planner. For example, if a planning problem had been a more complicated planning problem with multiple vehicles, many constrained way-points and links that were transient (rather than missing), it is difficult or impossible to prove that a solution does not exist using a propositional planner.

This approach can even be useful in problems that have reasonable bounds on its domains—we may only want to consider plans with fewer than V vehicles and T time-steps. However if there are enough of these dimensions, and they each have a large enough reasonable bound, then the problem may still be too large to be solved by blindly propositionalizing. Our method may find a proof of infeasibility that ignores much of the problem and therefore scales better.

6 CONCLUSIONS AND FUTURE WORK

In this paper we developed a new instantiation-based inference method for determining whether a FOP formula is feasible. We proved that this procedure is both sound and refutationally complete. Future directions for work on this reasoning system include improving heuristics for instance selections, investigating redundancy criteria for added clauses, and seeing if we can 'warm-start' propositional ILP solving based on the work done in previous iterations. Other promising directions include supporting object theories (such as equality, time, and fragments of arithmetic).

One major goal for us is to fully automate our inference procedure—our algorithm is currently an open-loop system that requires a human to select SIG applications. These selection decisions are critical because in the worst case, every two $\vee$-clauses could be the premises for a SIG application. Adding all possible applications ($\mathcal{O}(n^2)$ if there are n clauses) would create a formula that has length $\mathcal{O}(2^{2^i})$ after i iterations. Good selection heuristics are therefore essential for tractable inference. We intend to start our search for heuristics by adapting, evaluating and modifying both heuristics and restriction criteria from FOL resolution and instantiation-based theorem provers. Initial experiments also indicate that the policy of randomly selecting a single application induces a heavy-tailed runtime distribution, and this indicates that restarting policies will be a fruitful direction for research.

References

P. Baumgartner. FDPLL–a first-order Davis-Putnam-Logeman-Loveland procedure. *CADE*, 2000.

P. Baumgartner and C. Tinelli. The model evolution calculus. *CADE*, pages 350–364, 2003.

P. Baumgartner and C. Tinelli. The model evolution calculus as a first-order DPLL method. *Artificial Intelligence*, 172(4-5):591–632, 2008. ISSN 0004-3702.

P. Baumgartner, A. Fuchs, and C. Tinelli. ME (LIA)-Model Evolution With Linear Integer Arithmetic Constraints. In *LPAR*, page 258. Springer, 2008.

G. Faure, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. SAT modulo the theory of linear arithmetic: Exact, inexact and commercial solvers. In H. Kleine Büning and X. Zhao, editors, *SAT*, volume 4996 of *LNCS*. Springer, 2008.

R. Fourer, D. Gay, and B.W. Kernighan. *The AMPL book*. Duxbury Press, Pacific Grove, 2002.

H. Ganzinger and K. Korovin. New directions in instantiation-based theorem proving. In *LICS*, 2003.

H. Ganzinger and K. Korovin. Theory instantiation. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 497–511. Springer, 2006.

G.J. Gordon, S.A. Hong, and M. Dudík. First-order mixed integer linear programming. In *Proceedings of the 25 Conference on Uncertainty in Artificial Intelligence*, pages 213–222. AUAI Press, 2009.

J.N. Hooker, G. Rago, V. Chandru, and A. Shrivastava. Partial instantiation methods for inference in first-order logic. *J. Autom. Reas.*, 28(4), 2002.

R.G. Jeroslow. Computation-oriented reductions of predicate to propositional logic. *Decision Support Systems*, 4(2):183–197, 1988. ISSN 0167-9236.

K. Korovin. An invitation to instantiation-based reasoning: From theory to practice. *Volume in memoriam of Harald Ganzinger, LNCS. Springer*, 2009.

K. Korovin and A. Voronkov. Integrating linear arithmetic into superposition calculus. In *Computer Science Logic*, pages 223–237. Springer, 2007.

S.J. Lee and D.A. Plaisted. Eliminating duplication with the hyper-linking strategy. *Journal of Automated Reasoning*, 9(1):25–42, 1992. ISSN 0168-7433.

R. Letz and G. Stenz. Proof and model generation with disconnection tableaux. In *LPAR*, pages 142–156. Springer, 2001.

R. Letz and G. Stenz. The disconnection tableau calculus. *J. Autom. Reason.*, 38(1-3):79–126, 2007.

G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization*. Wiley New York, 1988.

R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL (T). *Journal of the ACM (JACM)*, 53(6):937–977, 2006. ISSN 0004-5411.