Scaling Relational Inference Using Proofs and Refutations

Ravi Mangal*, Xin Zhang*, Aditya Kamath*, Aditya V. Nori†, Mayur Naik*
*Georgia Institute of Technology, USA †Microsoft Research, UK

Abstract

Many inference problems are naturally formulated using hard and soft constraints over relational domains: the desired solution must satisfy the hard constraints, while optimizing the objectives expressed by the soft constraints. Existing techniques for solving such constraints rely on efficiently grounding a sufficient subset of constraints that is tractable to solve. We present an eager-lazy grounding algorithm that eagerly exploits proofs and lazily refutes counterexamples. We show that our algorithm achieves significant speedup over existing approaches without sacrificing soundness for real-world applications from information retrieval and program analysis.

1 Introduction

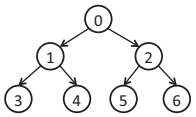
Many inference tasks in diverse application domains such as machine learning, information retrieval, mathematical optimization, and many others require optimizing certain objectives in addition to satisfying soundness conditions. A natural specification for such tasks appears in the form of relational constraints with optional weights in logics such as Markov Logic Networks (Richardson and Domingos 2006) and Probabilistic Soft Logic (Kimmig et al. 2012). The *relational inference problem* in such logics seeks a solution that satisfies all unweighted ("hard") constraints while maximizing the sum of the weights of satisfied weighted ("soft") constraints. Hard constraints thus enable to express soundness conditions while soft constraints allow to express the objectives to optimize.

Figure 1 shows an example that specifies a graph reachability problem using such constraints. Input predicate $e(n_1, n_2)$ is true if the graph has an edge from node n_1 to n_2. Derived predicate $p(n_1, n_2)$ is true if the graph has a path from n_1 to n_2. Hard constraints (1) and (2) enforce reflexivity and transitivity conditions, while soft constraint (3) ensures that the number of paths derived is minimal. An example input and solution are shown in Figure 2.

Existing techniques reduce the relational inference problem to the Weighted Partial Maximum Satisfiability (WPMS) problem (Papadimitriou 1994). This reduction happens via *grounding*, which is the process of instantiating the predicates or relations over all constants in the corresponding input domains. The WPMS problem is computationally hard. To reduce the burden on the WPMS solver, the key challenge is to efficiently ground a sufficient subset of constraints that is tractable to solve. For the example in Figure 1, eagerly grounding constraint (2) entails instantiating n_1, n_2, n_3 over all nodes in the graph, producing N^3 grounded constraints, where N is the number of nodes.

$$
\begin{array}{lll}
 & \forall n_1, & p(n_1, n_1) \quad (1)\\
\bigwedge & \forall n_1, n_2, n_3, & p(n_1, n_2) \wedge e(n_2, n_3) \Rightarrow p(n_1, n_3) \quad (2)\\
\bigwedge & 1.5 : \forall n_1, n_2, & \neg p(n_1, n_2) \quad (3)
\end{array}
$$

Figure 1: Graph reachability using relational constraints.



Input facts:		Output facts:	
$e(0,1)$	$e(0,2)$	$p(0,0)$	$p(0,1)$
$e(1,3)$	$e(1,4)$	$p(0,2)$	$p(0,3)$
$e(2,5)$	$e(2,6)$	$p(0,4)$	$p(0,5)$
		...	

Figure 2: Example graph reachability input and solution.

Several techniques have been proposed to lazily ground constraints (Niu et al. 2011; Kok et al. 2007; Chaganty et al. 2013; Noessner, Niepert, and Stuckenschmidt 2013; Riedel 2008; 2009). For the scale of problems we consider, however, these techniques are either too lazy and converge very slowly, or too eager and produce instances that are beyond the reach of sound WPMS solvers. For instance, a recent technique (Chaganty et al. 2013) grounds hard constraints too lazily and soft constraints too eagerly. Specifically, for the graph reachability problem in Figure 1, this technique takes L iterations to lazily ground hard constraint (2) (where L is the length of the longest path in the input graph), and generates N^2 constraints upfront by eagerly grounding soft constraint (3).

In this paper, we propose an iterative eager-lazy algorithm that strikes a balance between eager grounding and lazy grounding. Our key underlying idea comprises two complementary optimizations: eagerly exploiting proofs and lazily refuting counterexamples.

To eagerly exploit proofs, our algorithm uses an efficient procedure to upfront ground constraints that will necessarily be grounded during the iterative process. As a concrete in-

stance of this procedure, we use a Datalog solver, which efficiently computes the least solution of a set of recursive Horn constraints. [1] In practice, most constraints in many inference tasks are Horn, allowing to effectively leverage a Datalog solver. For instance, both hard constraints (1) and (2) in our graph reachability example are Horn. Our algorithm therefore applies a Datalog solver to efficiently ground them upfront. On the example graph in Figure 2, this produces 7 ground instances of (1) and only 10 instances of (2).

To lazily refute counterexamples, our algorithm uses an efficient procedure to check for violations of constraints by the WPMS solution to the set of ground constraints in each iteration, terminating the iterative process in the absence of violations. We use a Datalog solver as a concrete instance of this procedure as well. Existing lazy techniques, such as Cutting Plane Inference (CPI) (Riedel 2008; 2009) and SoftCegar (Chaganty et al. 2013), can be viewed as special cases of our algorithm in that they only lazily refute counterexamples. For this purpose, CPI uses a relational database query engine and SoftCegar uses a satisfiability modulo theories or SMT theorem prover (de Moura and Bjørner 2008), whereas we use a Datalog solver; engineering differences aside, the key motivation underlying all three approaches is the same: to use a separate procedure that is efficient at refuting counterexamples. Our main technical insight is to apply this idea analogously to eagerly exploit proofs. Our resulting strategy of guiding the grounding process based on both proofs and counterexamples gains the benefits of both eager and lazy grounding without suffering from the disadvantages of either.

We apply our algorithm to enable sound and efficient inference for large problem instances from two different application domains: *information retrieval*, which concerns discovering relevant information from large, unstructured data collections, and *program analysis*, which concerns discovering relevant information from large, structured programs—this information includes specifications, proofs of correctness properties, witnesses of harmful bugs and vulnerabilities, and resource bounds.

The graph reachability example illustrates important aspects of inference tasks in both these domains. In the information retrieval domain, an example task is the entity resolution problem for removing duplicate entries in a citation database (Singla and Domingos 2005). In this problem, a hard constraint similar to transitivity constraint (2) in Figure 1 encodes an axiom about the equivalence of citations:

$$sameBib(v_1, v_2) \wedge sameBib(v_2, v_3) \Rightarrow sameBib(v_1, v_3)$$

Likewise, in the program analysis domain, many emerging tasks require optimizing certain objectives in addition to satisfying soundness conditions. Consider, for instance, the taint analysis problem which determines whether a given program may leak sensitive data to untrusted users (Myers 1999; Livshits and Lam 2005). In this problem, hard constraints encode soundness conditions of the analysis, such as: if variable v_2 contains sensitive data, and the program contains a statement of the form $v_1 = v_2$, then variable v_1 also contains sensitive data.

$$tainted(v_2) \wedge assign(v_1, v_2) \Rightarrow tainted(v_1)$$

This constraint is akin to transitivity constraint (2) in Figure 1. Many program analysis problems besides taint analysis can be expressed solely using such Horn constraints (Whaley et al. 2005; Bravenboer and Smaragdakis 2009; Smaragdakis and Bravenboer 2010). Soft constraints, on the other hand, express optimizing diverse objectives common to different analysis problems, such as minimizing assumptions about unknown information (Kremenek et al. 2006; Kremenek, Ng, and Engler 2007; Livshits et al. 2009; Beckman and Nori 2011; Zhu, Dillig, and Dillig 2013), minimizing interaction with analysis users (Dillig, Dillig, and Aiken 2012; Bastani, Anand, and Aiken 2015), maximizing preferences of analysis users (Mangal et al. 2015), and striking various analysis tradeoffs (Zhang et al. 2014; Larraz et al. 2013; 2014).

We evaluate our algorithm on three benchmarks with three different input datasets generated from real-world information retrieval and program analysis applications. Our empirical evaluation shows that our approach achieves significant improvement over three state-of-art approaches, CPI (Riedel 2008; 2009), RockIt (Noessner, Niepert, and Stuckenschmidt 2013), and Tuffy (Niu et al. 2011), in running time as well as the quality of the solution.

2 Preliminaries

Figure 3 defines the abstract syntax of the relational constraints that we consider in this paper. A system of *relational constraints* C consists of a set of hard constraints H, and a set of soft constraints S.

A *hard constraint* $h \in H$ is an inference rule $A \Rightarrow B$, where A is a conjunction of facts and B is a disjunction of facts. A *fact* t consists of a relation name together with a tuple of arguments, which include variables and constants; a fact is a *ground fact* g when all arguments are constants.

A *soft constraint* $s \in S$ is a hard constraint along with a positive real weight w. A weight has a natural probabilistic interpretation, where the confidence associated with a soft constraint increases with the weight. For more details on the precise semantics of weights, the reader is referred to (Domingos and Lowd 2009).

For convenience, we augment the constraint system with an *input* P which defines a set of ground facts (extensional database or EDB). The solution to the constraints, *output* Q, defines a set of ground facts that are true (intensional database or IDB).

Example. The graph reachability problem applied to the graph in Figure 2 can be formulated as a system of constraints (H, S) where H is a set that contains the hard constraints (1) and (2) in Figure 1, and S is a set that contains the soft constraint (3) with weight 1.5. Further, the input P consists of all ground facts in relation $e(n_1, n_2)$, denoting all edges in the graph, while output Q contains all ground facts in relation $p(n_1, n_2)$, denoting all paths in the graph. □

[1] A Horn constraint is a disjunction of literals with at most one positive (i.e., unnegated) literal. It is typically written in the form of an inference rule $(u_1 \wedge ... \wedge u_n) \Rightarrow u$. Datalog is a decidable logic based on Horn constraints (Ceri, Gottlob, and Tanca 1989).

$$
\begin{array}{rrcl}
\text{(relation)} & r & \in & \mathbf{R} \\
\text{(constant)} & c & \in & \mathbf{C} \\
\text{(variable)} & v & \in & \mathbf{V} \\
\text{(valuation)} & \sigma & \in & \mathbf{V} \to \mathbf{C} \\
\\
\text{(argument)} & a & \in & \mathbf{A} = \mathbf{V} \cup \mathbf{C} \\
\text{(fact)} & t & \in & \mathbf{T} = \mathbf{R} \times \mathbf{A}^* \\
\text{(ground fact)} & g & \in & \mathbf{G} = \mathbf{R} \times \mathbf{C}^* \\
\text{(weight)} & w & \in & \mathbb{R}+ = (0, \infty] \\
\\
\text{(hard constraints)} & H & ::= & \{h_1, ..., h_n\},\ h ::= \bigwedge_{i=1}^{n} t_i \Rightarrow \bigvee_{i=1}^{m} t'_i \\
\text{(soft constraints)} & S & ::= & \{s_1, ..., s_n\},\ \ s ::= (h, w) \\
\\
\text{(constraints)} & C & ::= & (H, S) \\
\text{(input, output)} & P, Q & \subseteq & \mathbf{G}
\end{array}
$$

Figure 3: Abstract syntax of relational constraints.

$$
\begin{array}{rcl}
[\![(H, S)]\!] & = & ([\![H]\!], [\![S]\!]) \\
[\![\{h_1, ..., h_n\}]\!] & = & \bigwedge_{i=1}^{n} [\![h_i]\!] \\
[\![\{s_1, ..., s_n\}]\!] & = & \bigwedge_{i=1}^{n} [\![s_i]\!] \\
[\![h]\!] & = & \bigwedge_{\sigma} [\![h]\!]_\sigma \\
[\![(h, w)]\!] & = & \bigwedge_{\sigma} ([\![h]\!]_\sigma, w) \\
[\![\bigwedge_{i=1}^{n} t_i \Rightarrow \bigvee_{i=1}^{m} t'_i]\!]_\sigma & = & (\bigvee_{i=1}^{n} \neg [\![t_i]\!]_\sigma \vee \bigvee_{i=1}^{m} [\![t'_i]\!]_\sigma) \\
[\![r(a_1, ..., a_n)]\!]_\sigma & = & r([\![a_1]\!]_\sigma, ..., [\![a_n]\!]_\sigma) \\
[\![v]\!]_\sigma & = & \sigma(v) \\
[\![c]\!]_\sigma & = & c
\end{array}
$$

$$
\begin{array}{rrcl}
\text{(ground clause)} & \rho & ::= & \bigvee_{i=1}^{n} \neg g_i \vee \bigvee_{i=1}^{m} g'_i \\
\text{(hard clauses)} & \phi & ::= & \bigwedge_{i=1}^{n} \rho_i \\
\text{(soft clauses)} & \psi & ::= & \bigwedge_{i=1}^{n} (\rho_i, w_i)
\end{array}
$$

Figure 4: From constraints to ground clauses.

Grounding. Constraints are *grounded* by instantiating the relations over all constants in their respective input domain. The grounding procedure is shown in Figure 4. The procedure grounds each constraint into a set of corresponding clauses. In particular, the conversion $[\![h]\!] = \bigwedge_\sigma [\![h]\!]_\sigma$ grounds hard constraint h by enumerating all possible groundings σ of variables to constants, yielding a different clause for each unique valuation to the variables in h. Enumerating all possible valuations, called *full grounding*, does not scale to real-world problems.

Example. Figure 5 shows a subset of the ground clauses constructed for the constraints from Figure 1 applied to the graph in Figure 2. Along with the input ground facts from relation $e(n_1, n_2)$, hard constraints (1) and (2) in the graph reachability problem are grounded to construct the shown set of hard clauses, while soft constraint (3) is grounded to produce the shown set of soft clauses. □

Solving. A set of ground clauses is solved to produce a solution that satisfies all hard clauses and maximizes the sum of the weights of satisfied soft clauses. This problem is called Weighted Partial Maximum Satisfiability or WPMS (Papadimitriou 1994). In the solving process, ground clauses are first converted into a Boolean WPMS formula by replacing each unique ground fact with a separate Boolean variable. Every `WPMS` solver must satisfy the specification

Grounded constraints:			**WPMS formula:**	
Hard clauses:	$e(0,1)$	$\wedge$	$b0$	$\wedge$
	$e(0,2)$	$\wedge$	$b1$	$\wedge$
	$p(0,0)$	$\wedge$	$b2$	$\wedge$
	$p(1,1)$	$\wedge$	$b3$	$\wedge$
	$(\neg p(0,0) \vee \neg e(0,1) \vee p(0,1))$	$\wedge$	$(\neg b2 \vee \neg b0 \vee b4)$	$\wedge$
	$(\neg p(0,1) \vee \neg e(1,3) \vee p(0,3))$	$\wedge$	$(\neg b4 \vee \neg b5 \vee b6)$	$\wedge$
	...		...	
Soft clauses:	$1.5 : (\neg p(0,1))$	$\wedge$	$1.5 : (\neg b4)$	$\wedge$
	$1.5 : (\neg p(1,1))$	$\wedge$	$1.5 : (\neg b3)$	$\wedge$
	...		...	

Figure 5: Example grounded constraints and corresponding WPMS formula.

$$
\texttt{WPMS}(\phi, \textstyle\bigwedge_{i=1}^{n} (\rho_i, w_i)) =
$$
$$
\begin{cases}
\text{UNSAT} & \text{if } \nexists Q:\ Q \models \phi \\
Q \text{ such that } \left[\begin{array}{l} Q \models \phi \text{ and} \\ \Sigma_{i=1}^{n} \{ w_i \mid Q \models \rho_i \} \text{ is maximized} \end{array} \right] & \text{otherwise}
\end{cases}
$$

$$
\begin{array}{rcl}
Q & \models & \bigwedge_{i=1}^{n} \rho_i \quad \text{iff } \forall i : Q \models \rho_i \\
Q & \models & \bigvee_{i=1}^{n} \neg g_i \vee \bigvee_{i=1}^{m} g'_i \quad \text{iff } \exists i : g_i \notin Q \text{ or } \exists i : g'_i \in Q
\end{array}
$$
$$
\texttt{Weight}(Q, \textstyle\bigwedge_{i=1}^{n} (\rho_i, w_i)) \ = \ \Sigma_{i=1}^{n} \{\ w_i \mid Q \models \rho_i\ \}
$$

Figure 6: Specification for solving WPMS formulae.

shown in Figure 6. A `WPMS` solver takes as input a set of grounded hard and soft constraints (ϕ, ψ), and returns: (1) `UNSAT`, if no assignment of truth values to the Boolean variables satisfies the set of hard clauses ϕ, or (2) a solution Q, denoting the assignment "$\lambda g.(g \in Q)\,?\,\mathsf{true} : \mathsf{false}$" that sets variables corresponding to ground facts contained in Q to true, and the rest to false. Solution Q not only satisfies all hard clauses in ϕ (it is *sound*), but it also maximizes the sum of the weights of satisfied soft clauses in ψ (it is *optimal*). Q is not necessarily unique; two solutions Q_1 and Q_2 are *equivalent* if $\texttt{Weight}(Q_1, \psi) = \texttt{Weight}(Q_2, \psi)$.

Example. Figure 5 shows a snippet of the WPMS formula constructed for the system of constraints from Figure 1 applied to the graph in Figure 2. The formula is constructed from the hard and soft ground clauses, shown in the same figure, by replacing each unique ground fact with a separate Boolean variable. It is then fed to a WPMS solver to generate the final solution Q. □

Note that reducing relational constraints to WPMS via full grounding (the eager approach) is not tractable as the size of such generated WPMS instances is beyond the scope of the state-of-the-art solvers. Existing techniques (Kok et al. 2007; Niu et al. 2011; Chaganty et al. 2013; Riedel 2008; 2009; Noessner, Niepert, and Stuckenschmidt 2013) reduce the burden on the WPMS solver by considering subsets of the full grounding (the lazy approach) in an iterative manner. These approaches, however, are either overly conservative in growing the considered subsets resulting in too many iterations, or are excessively aggressive leading to intractable instances. Our algorithm, described in the next section, explores a sweet spot between both these approaches.

3 The IPR Algorithm

We propose an efficient iterative algorithm IPR (Inference via Proof and Refutation) for solving relational constraint systems described in the previous section. The algorithm has the following key features:

1. **Eager proof exploitation.** IPR eagerly explores the logical structure of the relational constraints to generate an initial grounding, which has the effect of speeding up the convergence of the algorithm. When the relational constraints are in the form of Horn constraints, we show that such an initial grounding is optimal (Theorem 3.2).
2. **Lazy counterexample refutation.** After solving the constraints in the initial grounding, IPR applies a refutation-based technique to refine the solution: it lazily grounds the constraints that are violated by the current solution, and solves the accumulated grounded constraints in an iterative manner.
3. **Termination with soundness and optimality.** IPR performs a termination check that guarantees the soundness and optimality of the solution if an exact WPMS solver is used. Moreover, this check is more precise than the termination checks in existing refutation-based algorithms (Riedel 2008; 2009; Chaganty et al. 2013), therefore leading to faster convergence.

Algorithm 1 IPR: the eager-lazy algorithm.

1: **INPUT** (H, S): Relational constraints.
2: **OUTPUT** Q: Solution (assumes $[\![H]\!]$ is satisfiable).
3: $\phi :=$ any $\bigwedge_{i=1}^{n} \rho_i$ such that $\forall i : \exists h \in H : \exists \sigma : \rho_i = [\![h]\!]_\sigma$
4: $\psi :=$ any $\bigwedge_{i=1}^{n} \rho_i$ such that $\forall i : \exists s \in S : \exists \sigma : \rho_i = [\![s]\!]_\sigma$
5: $Q := \emptyset$; $w := 0$
6: **while true do**
7: $\quad \phi' := \bigwedge_{h \in H} \bigwedge \texttt{Violations}(h, Q)$
8: $\quad \psi' := \bigwedge_{(h,w) \in S} \bigwedge \{ (\rho, w) \mid \rho \in \texttt{Violations}(h, Q) \}$
9: $\quad (\phi, \psi) := (\phi \wedge \phi', \psi \wedge \psi')$
10: $\quad Q' :=$ $\texttt{WPMS}(\phi, \psi)$
11: $\quad w' :=$ $\texttt{Weight}(Q', \psi)$
12: $\quad$ **if** $(w' = w \wedge \phi' = \mathsf{true})$ **then return** Q
13: $\quad Q := Q'$; $w := w'$
14: **end while**

IPR (Algorithm 1) takes relational constraints (H, S) as input and produces a Boolean assignment Q as output.[1] We next explain each component of IPR separately.

Eager proof exploitation. To start with, IPR computes an initial set of hard clauses ϕ and soft clauses ψ by exploiting the logical structure of the constraints (line 3–4 of Algorithm 1). The sets ϕ and ψ can be arbitrary subsets of the hard clauses and soft clauses in the full grounding. When ϕ and ψ are both empty, the behavior of IPR defaults to lazy approaches like CPI (Riedel 2008; 2009) (Definition 3.1).

[1] We assume that any input P is encoded as part of the hard constraints H. For clarity, we also assume that the hard constraints H are satisfiable, allowing us to elide showing UNSAT as a possible alternative to output Q.

As a concrete instance of eager proof exploitation, when a subset of the relational constraints have a recursive Horn form, IPR applies a Datalog solver to efficiently compute the least solution of this set of recursive Horn constraints, and find a relevant subset of clauses to be grounded upfront. In particular, when all the hard constraints are Horn, IPR prescribes a recipe for generating an optimal initial set of grounded constraints.

Theorem 3.2 shows that for hard relational constraints in Horn form, lazy approaches like CPI ground at least as many hard clauses as the number of true ground facts in the least solution of such Horn constraints. Also, and more importantly, we show there exists a strategy that can upfront discover the set of all these necessary hard clauses and guarantee that no more clauses, besides those in the initial set, will be grounded. In practice, eager proof exploitation is employed for both hard and soft Horn clauses. Theorem 3.2 guarantees that if upfront grounding is applied to hard Horn clauses, then only relevant clauses are grounded. While this guarantee does not apply to upfront grounding of soft Horn clauses, we observe empirically that most such grounded clauses are relevant. Moreover, as Section 4 shows, using this strategy for initial grounding allows the iterative process to terminate in far fewer iterations while ensuring that each iteration does approximately as much work as before (without initial grounding).

Definition 3.1 (Lazy Algorithm) *Algorithm* Lazy *is an instance of* IPR *with $\phi = \psi = \emptyset$ as the initial grounding.*

Theorem 3.2 (Optimal initial grounding for Horn clauses) *If a relational constraint system contains a set of hard constraints H, each of which is a Horn clause $\bigwedge_{i=1}^{n} t_i \Rightarrow t_0$, whose least solution is desired:*

$$G = \mathsf{lfp}\ \lambda G'.\ G' \cup \{\ [\![t_0]\!]_\sigma \mid (\textstyle\bigwedge_{i=1}^{n} t_i \Rightarrow t_0) \in H \text{ and } \forall i \in [1..n] : [\![t_i]\!]_\sigma \in G'\ \},$$

then for such a system, (a) $\texttt{Lazy}(H, \emptyset)$ grounds at least $|G|$ clauses, and (b) $\texttt{IPR}(H, \emptyset)$ with the initial grounding ϕ does not ground any more clauses where:

$$\phi = \bigwedge\{\ \textstyle\bigvee_{i=1}^{n} \neg [\![t_i]\!]_\sigma \vee [\![t_0]\!]_\sigma \mid (\bigwedge_{i=1}^{n} t_i \Rightarrow t_0) \in H \text{ and } \forall i \in [0..n] : [\![t_i]\!]_\sigma \in G\ \}.$$

Proof. See Appendix A. □

Lazy counterexample refutation. After generating the initial grounding, IPR iteratively grounds more clauses and refines the solution by refutation (lines 6–14 of Algorithm 1). In each iteration, the algorithm keeps track of the previous solution Q, and the weight w of the solution Q by calling the Weight procedure specified in Figure 6. Initially, the solution is empty with weight zero (line 5).

In line 7, IPR computes all the violations of the hard constraints for the previous solution Q. The related procedure Violations is formally defined below:

$$\texttt{Violations}(h, Q) = \{\ [\![h]\!]_\sigma \mid Q \not\models [\![h]\!]_\sigma\ \}.$$

Similarly, in line 8, the set of soft clauses ψ' violated by the previous solution Q is computed. In line 9, both sets of

Initial			Iteration 1		
$\neg p(0,0) \vee \neg e(0,1) \vee p(0,1)$	$\neg p(0,0) \vee \neg e(0,2) \vee p(0,2)$	$p(0,0)$	$1.5 : \neg p(0,0)$	$1.5 : \neg p(1,1)$	$1.5 : \neg p(2,2)$
$\neg p(1,1) \vee \neg e(1,3) \vee p(1,3)$	$\neg p(1,1) \vee \neg e(1,4) \vee p(1,4)$	$p(1,1)$	$1.5 : \neg p(3,3)$	$1.5 : \neg p(4,4)$	$1.5 : \neg p(5,5)$
$\neg p(2,2) \vee \neg e(2,5) \vee p(2,5)$	$\neg p(2,2) \vee \neg e(2,6) \vee p(2,6)$	$p(2,2)$	$1.5 : \neg p(6,6)$	$1.5 : \neg p(0,1)$	$1.5 : \neg p(0,2)$
$\neg p(0,1) \vee \neg e(1,3) \vee p(0,3)$	$\neg p(0,1) \vee \neg e(1,4) \vee p(0,4)$	$p(3,3)$	$1.5 : \neg p(1,3)$	$1.5 : \neg p(1,4)$	$1.5 : \neg p(2,5)$
$\neg p(0,2) \vee \neg e(2,5) \vee p(0,5)$	$\neg p(0,2) \vee \neg e(2,6) \vee p(0,6)$	$p(4,4)$	$1.5 : \neg p(2,6)$	$1.5 : \neg p(0,3)$	$1.5 : \neg p(0,4)$
$p(5,5)$	$p(6,6)$		$1.5 : \neg p(0,5)$	$1.5 : \neg p(0,6)$	

Table 1: Clauses in the initial grounding and additional clauses grounded in each iteration of IPR for graph reachability example.

violations ϕ' and ψ' are added to the corresponding sets of grounded hard clauses ϕ and grounded soft clauses ψ respectively. The intuition for adding violated hard clauses ϕ' to the set ϕ is straightforward—the set of hard clauses ϕ is not sufficient to prevent the WPMS procedure from producing a solution Q that violates the set of hard constraints H. The intuition for soft clauses is similar—since the goal of WPMS is to maximize the sum of the weights of satisfied soft constraints in S, and all weights in our relational constraint system are positive, any violation of a soft clause possibly leads to a sub-optimal solution which could have been avoided if the violated clause was present in the set of soft clauses ψ.

In line 10, this updated set ϕ of hard clauses and set ψ of soft clauses are fed to the WPMS procedure to produce a new solution Q' and its corresponding weight w'. At this point, in line 12, the algorithm checks if the terminating condition is satisfied by the solution Q'.

Termination check. IPR terminates when no hard clauses are violated by current solution ($\phi' = true$) and the current objective cannot be improved by adding more soft clauses ($w' = w$). This termination check improves upon that in previous works (Riedel 2008; 2009; Chaganty et al. 2013), and speeds up the convergence of the algorithm in practice. Theorem 3.3 shows that our IPR algorithm always terminates with a sound and optimum solution if the underlying WPMS solver is exact.

Theorem 3.3 (Soundness and Optimality of IPR) *For any relational constraint system (H, S) where H is satisfiable, IPR(H, S) produces a sound and optimal solution.*

Proof. See Appendix B. □

Example. The IPR algorithm takes two iterations and grounds 17 hard clauses and 17 soft clauses to solve the graph reachability example in Figures 1 and 2. Table 1 shows the clauses in the initial grounding computed using a Datalog solver and additional clauses grounded in each iteration of IPR. IPR grounds no additional clauses in Iteration 2. Therefore, the related column is omitted in Table 1.

On the other hand, an eager approach with full grounding needs to ground 392 hard clauses and 49 soft clauses, which is 12× of the number of clauses grounded in IPR. Moreover, the eager approach generates clauses such as $\neg p(0,1) \vee \neg e(1,5) \vee p(1,5)$ and $\neg p(1,4) \vee \neg e(4,2) \vee p(1,2)$, that are trivially satisfied given the input edge relation.

A lazy approach with an empty initial grounding grounds the same number of hard clauses and soft clauses as IPR. However, it takes 5 iterations to terminate, which is 2.5× of the number of iterations needed by IPR.

	# relations	# rules	# EDB tuples			# clauses in full grounding		
			E1	E2	E3	E1	E2	E3
PA	94	89	1,274	3.9×10^6	1.1×10^7	2×10^{10}	1.6×10^{28}	1.2×10^{30}
AR	14	24	607	3,595	7,010	1.7×10^8	1.1×10^9	2.4×10^{10}
RC	5	17	1,656	3,190	7,766	7.9×10^{11}	1.2×10^{13}	3.8×10^{14}

Table 2: Statistics of application constraints and datasets.

The results on the graph reachability example show that, IPR combines the benefits of the eager approach and the lazy approach while avoiding their drawbacks. □

4 Empirical Evaluation

In this section, we evaluate IPR and compare it with three state-of-the-art approaches, CPI (Riedel 2008; 2009), ROCKIT (Noessner, Niepert, and Stuckenschmidt 2013), and TUFFY (Niu et al. 2011), on three different benchmarks with three different-sized inputs per benchmark.

We implemented IPR in roughly 10,000 lines of Java. To compute the initial grounded constraints, we use bddbddb (Whaley and Lam 2004), a Datalog solver. The same solver is used to identify grounded constraints that are violated by a solution. For our evaluation, we use two different instances of IPR, referred as IPR1 and IPR2, that vary in the underlying solver used for solving the constraints. For IPR1, we use LBX (Mencia, Previti, and Marques-Silva 2015) as the underlying WPMS solver which guarantees soundness of the solution (i.e., does not violate any hard clauses). Though LBX does not guarantee the optimality of the solution, in practice, we find the cost of the solution computed by LBX is close to that of the solution computed by an exact solver. For IPR2, we use GUROBI as the underlying solver. GUROBI is an integer linear program (ILP) solver which guarantees soundness of the solution. Additionally, it guarantees that the cost of the generated solution is within a limited bound from that of the optimal solution. We incorporate it in our approach by replacing the call to WPMS (line 10 of Algorithm 1) with a call to an ILP encoder followed by a call to GUROBI. The ILP encoder translates the WPMS problem to an equivalent ILP formulation.

All experiments were done using Oracle HotSpot JVM 1.6.0 on a Linux server with 64GB RAM and 3.0GHz processors. The three benchmarks are as follows:

Program Analysis (PA): This application performs *pointer analysis*, a foundational program analysis that approximates which program variables point to which memory locations. The hard constraints here express the soundness conditions of the analysis while the soft constraints encode user preferences about the analysis results as described in (Mangal et

	EDB	# iterations				total time (m = min., s = sec.)					# ground clauses (K = thousand, M = million)					solution cost (K = thousand, M = million)				
		CPI	**IPR1**	ROCKIT	**IPR2**	CPI	**IPR1**	ROCKIT	**IPR2**	TUFFY	CPI	**IPR1**	ROCKIT	**IPR2**	TUFFY	CPI	**IPR1**	ROCKIT	**IPR2**	TUFFY
PA	E1	23	6	19	3	29s	28s	53s	13s	6m56s	0.6K	0.6K	0.6K	0.6K	0.6K	0	0	0	0	743
	E2	171	8	185	3	235m	23m	286m	150m	–	3M	3.2M	2.9M	3.2M	–	46	46	35	35	–
	E3	–	11	–	–	–	114m	–	–	–	–	13M	–	–	–	–	345	–	–	–
AR	E1	6	5	4	3	8s	8s	4s	8s	2m25s	9.8K	9.8K	27K	83K	589K	6.4K	6.4K	5.8K	6.1K	7K
	E2	6	6	7	7	34m	36m	37s	42s	–	2.3M	2.3M	0.4M	0.4M	–	0.4M	0.4M	0.39M	0.39M	–
	E3	6	6	7	7	141m	124m	1m45s	2m1s	–	8M	8M	1.4M	1.4M	–	0.68M	0.68M	0.67M	0.67M	–
RC	E1	17	6	5	4	3m5s	1m4s	6s	6s	11s	0.5M	0.3M	55K	68K	28K	5.7K	5.7K	5.7K	5.7K	160K
	E2	8	8	5	3	6m19s	3m41s	9s	10s	2m28s	2.4M	1.3M	0.11M	0.17M	64K	10.7K	10.7K	10.6K	10.6K	42.7K
	E3	17	13	20	20	150m	46m20s	2m35s	2m54s	180m	14M	4.5M	0.5M	1.2M	0.35M	25.1K	25.1K	24.9K	24.9K	0.25M

Table 3: Results of evaluating CPI, IPR1, ROCKIT, IPR2, and TUFFY, on three benchmark applications. CPI and IPR1 use LBX as the underlying solver, while ROCKIT and IPR2 use GUROBI. In all experiments, we used a memory limit of 64GB and a time limit of 24 hours. Timed out experiments (denoted '–') exceeded either of these limits.

al. 2015). The datasets for this application are derived from three real-world Java programs ranging in size from 1.4K to 190K lines of code.

Advisor Recommendation (AR): This is an advisor recommendation system to aid new graduate students in finding good PhD advisors. The datasets for this application were generated from the AI Genealogy Project (http://aigp.eecs.umich.edu) and from DBLP (http://dblp.uni-trier.de).

Relational Classification (RC): In this application, papers from the Cora dataset (McCallum et al. 2000) are classified into different categories based on the authors and their main area of research.

Table 2 shows statistics of our three benchmarks (PA, AR, RC) and the corresponding EDBs used in our evaluation.

We compare IPR1 and IPR2 with three state-of-the-art techniques, CPI, ROCKIT, and TUFFY.

TUFFY employs a non-iterative approach which is composed of two steps: first, it generates an initial set of grounded constraints that is expected to be a superset of all the required grounded constraints; next, TUFFY uses a highly efficient but approximate WPMS solver to solve these grounded constraints. In our evaluation, we use the TUFFY executable available from its website http://i.stanford.edu/hazy/hazy/tuffy/.

CPI is a fully lazy iterative approach, which refutes counterexamples in a way similar to IPR. However, CPI does not employ proof exploitation and applies a more conservative termination check. In order to ensure a fair comparison between IPR and CPI, we implement CPI in our framework by incorporating the above two differences and use LBX as the underlying solver.

ROCKIT is also a fully lazy iterative approach similar to CPI. Additionally, like CPI, ROCKIT does not employ proof exploitation and its termination check is as conservative as CPI. The main innovation of ROCKIT is a clever ILP encoding for solving the underlying constraints. This reduces the time per iteration for solving, but does not necessarily reduce the number of iterations. In our evaluation, we use the ROCKIT executable available from its website https://code.google.com/p/rockit/.

The primary innovation of ROCKIT is complementary to our approach. In fact, to ensure a fair comparison between IPR and ROCKIT, in IPR2 we use the same ILP encoding as used by ROCKIT. This combined approach yields the benefits of both ROCKIT and our approach.

Table 3 summarizes the results of running CPI, IPR1, ROCKIT, IPR2, and TUFFY on our benchmarks. IPR1 significantly outperforms CPI in terms of running time. Similarly, IPR2 outperforms ROCKIT in running time and the number of iterations needed while TUFFY has the worst performance. IPR1 terminates under all nine experiment settings, while IPR2, CPI and ROCKIT terminate under eight settings and TUFFY only terminates under five settings. In terms of the quality of the solution, IPR1 and CPI produce solutions with similar costs under all settings. IPR2 and ROCKIT produce solutions with slightly lower costs as they employ an ILP solver that guarantees a solution whose cost is within a fixed bounded from that of the optimal solution. TUFFY produces solutions with significantly higher costs. We next study the results for each benchmark more closely.

Program Analysis (PA): For PA, we first compare IPR1 with CPI. IPR1 significantly outperforms CPI on larger datasets, with CPI not terminating on E3 even after 24 hours while IPR1 terminates in under two hours. This is because most of the relational constraints in PA are Horn, allowing IPR1 to effectively perform eager proof exploitation, and ground relevant clauses upfront. This is also reflected in the reduced number of iterations for IPR1 compared to CPI. IPR2, that also uses eager proof exploitation, similarly outperforms ROCKIT on E1 and E2. However, both IPR2 and ROCKIT fail to terminate on E3. This indicates that the underlying type of solver plays an important role in the performance of these approaches. For PA, the WPMS solver employed by IPR1 is better suited to the problem compared to the ILP solver employed by IPR2. TUFFY performs the worst out of all the approaches, only terminating on the smallest dataset E1. Even for E1, the cost of the final solution generated by TUFFY is significantly higher compared to the other approaches. More acutely, TUFFY violates *ten hard clauses* on E1 which is absolutely unacceptable for program analysis benchmarks.

Advisor Recommendation (AR): On AR, IPR1, IPR2, CPI and ROCKIT terminate on all three datasets and produce similar results while TUFFY only terminates on the smallest dataset. IPR1 and CPI have comparable performance on AR as a fully lazy approach suffices to solve the relational constraint system efficiently. Similarly, IPR2 and ROCKIT have similar performance. However, both IPR2 and ROCKIT

significantly outperform IPR1 and CPI in terms of the running time although they need similar number of iterations. This indicates that for AR, the smart ILP encoding leads to fewer constraints and also that the ILP solver is better suited than the WPMS solver, leading to a lower running time per iteration. Note that, all theses approaches except TUFFY terminate within seven iterations on all three datasets. On the other hand, TUFFY times out on the two larger datasets without passing its grounding phase, reflecting the need for lazily grounding the constraints.

Relational Classification (RC): All the approaches terminate on RC for all the three datasets. However, IPR1 outperforms CPI and TUFFY significantly in terms of runtime on the largest dataset. On the other hand, IPR2 and ROCKIT outperform IPR1. This is again due to the faster solving time per iteration enabled by the smart ILP encoding and the use of ILP solver. Unlike other benchmarks, the running time of TUFFY is comparable to the other approaches. However, the costs of its solutions are on average 14× more than the costs of the solutions produced by the other approaches.

5 Related Work

A large body of work exists to solve weighted relational constraints in a lazy manner. Lazy inference techniques (Singla and Domingos 2006; Poon, Domingos, and Sumner 2008) rely on the observation that most ground facts in the final solution to a relational inference problem have default values (a value appearing much more often than the others). These techniques start by assuming a default value for all ground facts, and gradually ground clauses as the values of facts are determined to be changed for a better objective. Such techniques apply a loose termination check, and therefore do not guarantee soundness nor optimality of the final solution. Iterative ground-and-solve approaches such as CPI (Riedel 2008) and RockIt (Noessner, Niepert, and Stuckenschmidt 2013) solve constraints lazily by iteratively grounding only those constraints that are violated by the current solution. Compared to lazy inference techniques, they apply a conservative termination check which guarantees the soundness and optimality of the solution. Compared to our approach, all of the above techniques either need more iterations to converge or have to terminate prematurely with potentially unsound and suboptimal solutions.

Tuffy (Niu et al. 2011) applies a non-iterative technique which is divided into a grounding phase and a solving phase. In the grounding phase, Tuffy grounds a set of clauses that it deems relevant to the solution of the inference problem. In practice, this set is often imprecise, which either hinders the scalability of the overall process (when the set is too large), or degrades the quality of the solution (when the set is too small). Soft-CEGAR (Chaganty et al. 2013) grounds the soft constraints eagerly and solves the hard constraints in a lazy manner. It uses a SMT solver to efficiently find the hard constraints violated by a solution. Lifted inference techniques (Braz, Amir, and Roth 2005; Milch et al. 2008; Poole 2003) use approaches from first-order logic, like variable elimination, to simplify the system of relational constraints. Such techniques can be used in conjunction with various iterative techniques including our approach for solving such constraints.

6 Conclusion

We presented a new technique for solving weighted relational constraints. Existing approaches either ground the constraints too eagerly, which produces intractably large propositional instances to WPMS solvers, or they ground the constraints too lazily, which prohibitively slows down the convergence of the overall process. Our approach strikes a balance between these two extremes by applying two complementary optimizations: eagerly exploiting proofs and lazily refuting counterexamples. Our empirical evaluation showed that our technique achieves significant improvement over two existing techniques in both performance and quality of the solution.

Acknowledgements. We thank the referees for useful feedback. This work was supported by DARPA under agreement #FA8750-15-2-0009, NSF awards #1253867 and #1526270, and a Facebook Fellowship. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright thereon.

A Proof of Theorem 3.2

Theorem A.1 (Optimal Initial Grounding for Horn Rules) *If a relational constraint system contains a set of hard constraints H, each of which is a Horn clause $\bigwedge_{i=1}^{n} t_i \Rightarrow t_0$, whose least solution is desired:*

$$G = \mathsf{lfp}\ \lambda G'.\ G' \cup \{\ [\![t_0]\!]_\sigma \mid (\textstyle\bigwedge_{i=1}^{n} t_i \Rightarrow t_0) \in H \text{ and } \forall i \in [1..n]\colon [\![t_i]\!]_\sigma \in G'\ \},$$

then for such a system, (a) $\texttt{Lazy}(H, \emptyset)$ grounds at least $|G|$ clauses, and (b) $\texttt{IPR}(H, \emptyset)$ with the initial grounding ϕ does not ground any more clauses where:

$$\phi = \bigwedge\{\ \textstyle\bigvee_{i=1}^{n} \neg[\![t_i]\!]_\sigma \vee [\![t_0]\!]_\sigma \mid (\bigwedge_{i=1}^{n} t_i \Rightarrow t_0) \in H \text{ and } \forall i \in [0..n]\colon [\![t_i]\!]_\sigma \in G\ \}.$$

Proof. To prove (a), we will show that for each $g \in G$, $\texttt{Lazy}(H, \emptyset)$ must ground some clause with g on the r.h.s. Let the sequence of sets of clauses grounded in the iterations of this procedure be $C_1, ..., C_n$. Then, we have:
Proposition (1): each clause $\bigwedge_{i=1}^{m} g_i \Rightarrow g'$ in any Q_j was added because the previous solution set all g_i to true and g' to false. This follows from the assumption that all rules in H are Horn rules. Let $x \in [1..n]$ be the earliest iteration in whose solution g was set to true. Then, we claim that g must be on the r.h.s. of some clause ρ in Q_x. Suppose for the sake of contradiction that no clause in Q_x has g on the r.h.s. Then, it must be the case that there is some clause ρ' in Q_x where g is on the l.h.s. (the WPMS procedure will not set variables to true that do not even appear in any clause in Q_x). Suppose clause ρ' was added in some iteration $y < x$. Applying proposition (1) above to clause ρ' and $j = x$, it must be that g was true in the solution to iteration y, contradicting the assumption above that x was the earliest iteration in whose solution g was set to true.

To prove (b), suppose $\texttt{IPR}(H, \emptyset)$ grounds an additional clause, that is, there exists a $(\bigwedge_{i=1}^{n} t_i \Rightarrow t_0) \in H$ and a σ

such that $G \not\models \bigvee_{i=1}^{n} \neg[\![t_i]\!]_\sigma \vee [\![t_0]\!]_\sigma$. The only way by which this can hold is if $\forall i \in [1..n] : [\![t_i]\!]_\sigma \in G$ and $[\![t_0]\!]_\sigma \notin G$, but this contradicts the definition of G. □

B Proof of Theorem 3.3

Theorem B.1 (Soundness and Optimality of IPR) *For any relational constraint system (H, S) where H is satisfiable, $\texttt{IPR}(H, S)$ produces a sound and optimal solution.*

Proof. We extend the function `Weight` (Figure 6) to hard clauses, yielding $-\infty$ if any such clause is violated:

$$\texttt{W} = \lambda(Q, \phi \cup \psi).\ \textit{if}\ (\exists \rho \in \phi : Q \not\models \rho)\ \textit{then}\ -\infty \ \textit{else}\ \texttt{Weight}(Q, \psi).$$

It suffices to show that the solution produced by $\texttt{IPR}(H, S)$ has the same weight as the solution produced by the eager approach. The eager approach (denoted by `Eager`) generates the solution by posing clauses generated from full grounding to a WPMS solver.

First, observe that $\texttt{IPR}(H, S)$ terminates: in each iteration of the loop in line 6 of Algorithm 1, it must be the case that at least one new hard clause is added to ϕ or at least one new soft clause is added to ψ, because otherwise the condition on line 12 will hold and the loop will be exited.

Now, suppose that in last iteration of the loop in line 6 for computing $\texttt{IPR}(H, S)$, we have:
(1) $gc_1 = hgc_1 \cup sgc_1$ is the set of hard and soft clauses accumulated in (ϕ, ψ) so far (line 9);
(2) $Q_1 = \emptyset$ and $w_1 = 0$ (line 5), or $Q_1 = \texttt{WPMS}(hgc_1, sgc_1)$ and its weight is w_1 (lines 10 and 11);
(3) $gc_2 = hgc_2 \cup sgc_2$ is the set of all hard and soft clauses that are violated by Q_1 (lines 7 and 8);
(4) $Q_2 = \texttt{WPMS}(hgc_1 \cup hgc_2, sgc_1 \cup sgc_2)$ and its weight is w_2 (lines 10 and 11);
and the condition on line 12 holds as this is the last iteration:
(5) $w_1 = w_2$ and $hgc_2 = \emptyset$.
Then, the result of $\texttt{IPR}(H, S)$ is Q_1. On the other hand, the result of $\texttt{Eager}(H, S)$ is:
(6) $Q_f = \texttt{WPMS}(hgc_f, sgc_f)$ where:
(7) $gc_f = hgc_f \cup sgc_f$ is the set of fully grounded hard and soft clauses (Figure 4).
Thus, it suffices to show that Q_1 and Q_f are equivalent.
Define $gc_m = gc_2 \setminus gc_1$.
(8) For any Q, we have:

$$\begin{aligned}
\texttt{W}(Q, gc_1 \cup gc_2) &= \texttt{W}(Q, gc_1) + \texttt{W}(Q, gc_m) \\
&= \texttt{W}(Q, gc_1) + \texttt{W}(Q, hgc_2 \setminus hgc_1) + \\
&\quad\ \texttt{W}(Q, sgc_2 \setminus sgc_1) \\
&= \texttt{W}(Q, gc_1) + \texttt{W}(Q, sgc_2 \setminus sgc_1) \quad [a] \\
&\geq \texttt{W}(Q, gc_1) \quad [b]
\end{aligned}$$

where [a] follows from (5), and [b] from $\texttt{W}(Q, sgc_2) >= 0$ (i.e., soft clauses do not have negative weights). Instantiating (8) with Q_1, we have: (9): $\texttt{W}(Q_1, gc_1 \cup gc_2) \geq \texttt{W}(Q_1, gc_1)$. Combining (2), (4), and (5), we have: (10): $\texttt{W}(Q_1, gc_1) = \texttt{W}(Q_2, gc_1 \cup gc_2)$. Combining (9) and (10), we have: (11)$\texttt{W}(Q_1, gc_1 \cup gc_2) \geq \texttt{W}(Q_2, gc_1 \cup gc_2)$. This means Q_1 is a better solution than Q_2 on $gc_1 \cup gc_2$. But from (4), we have that Q_2 is an optimum solution to $gc_1 \cup gc_2$, so we have: (12): Q_1 is also an optimum solution to $gc_1 \cup gc_2$.

It remains to show that Q_1 is also an optimum solution to the set of fully grounded hard and soft clauses gc_f, from which it will follow that Q_1 and Q_f are equivalent. Define $gc_r = gc_f \setminus (gc_1 \cup gc_2)$. For any Q, we have:

$$\begin{aligned}
\texttt{W}(Q, gc_f) &= \texttt{W}(Q, gc_1 \cup gc_2 \cup gc_r) \\
&= \texttt{W}(Q, gc_1 \cup gc_2) + \texttt{W}(Q, gc_r) \\
&\leq \texttt{W}(Q_1, gc_1 \cup gc_2) + \texttt{W}(Q, gc_r) \quad [c] \\
&\leq \texttt{W}(Q_1, gc_1 \cup gc_2) + \texttt{W}(Q_1, gc_r) \quad [d] \\
&= \texttt{W}(Q_1, gc_1 \cup gc_2 \cup gc_r) \\
&= \texttt{W}(Q_1, gc_f)
\end{aligned}$$

i.e. $\forall Q, \texttt{W}(Q, gc_f) \leq \texttt{W}(Q_1, gc_f)$, proving that Q_1 is an optimum solution to gc_f. Inequality [c] follows from (11), that is, Q_1 is an optimum solution to $gc_1 \cup gc_2$. Inequality [d] holds because from (3), all clauses that Q_1 possibly violates are in gc_2, whence Q_1 satisfies all clauses in gc_r, whence $\texttt{W}(Q, gc_r) \leq \texttt{W}(Q_1, gc_r)$. □

References

Bastani, O.; Anand, S.; and Aiken, A. 2015. Interactively verifying absence of explicit information flows in Android apps. In *OOPSLA*.

Beckman, N., and Nori, A. 2011. Probabilistic, modular and scalable inference of typestate specifications. In *PLDI*.

Bravenboer, M., and Smaragdakis, Y. 2009. Strictly declarative specification of sophisticated points-to analyses. In *OOPSLA*.

Braz, R. D. S.; Amir, E.; and Roth, D. 2005. Lifted first-order probabilistic inference. In *IJCAI*.

Ceri, S.; Gottlob, G.; and Tanca, L. 1989. What you always wanted to know about Datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on* 1(1):146–166.

Chaganty, A.; Lal, A.; Nori, A.; and Rajamani, S. 2013. Combining relational learning with SMT solvers using CEGAR. In *CAV*.

de Moura, L., and Bjørner, N. 2008. Z3: an efficient SMT solver. In *TACAS*.

Dillig, I.; Dillig, T.; and Aiken, A. 2012. Automated error diagnosis using abductive inference. In *PLDI*.

Domingos, P., and Lowd, D. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Kimmig, A.; Bach, S. H.; Broecheler, M.; Huang, B.; and Getoor, L. 2012. A short introduction to probabilistic soft logic. In *NIPS Workshop on Probabilistic Programming: Foundations and Applications*.

Kok, S.; Sumner, M.; Richardson, M.; Singla, P.; Poon, H.; Lowd, D.; and Domingos, P. 2007. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA.

Kremenek, T.; Twohey, P.; Back, G.; Ng, A.; and Engler, D. 2006. From uncertainty to belief: Inferring the specification within. In *OSDI*.

Kremenek, T.; Ng, A.; and Engler, D. 2007. A factor graph model for software bug finding. In *IJCAI*.

Larraz, D.; Oliveras, A.; Rodríguez-Carbonell, E.; and Rubio, A. 2013. Proving termination of imperative programs using Max-SMT. In *FMCAD*.

Larraz, D.; Nimkar, K.; Oliveras, A.; Rodríguez-Carbonell, E.; and Rubio, A. 2014. Proving non-termination using Max-SMT. In *CAV*.

Livshits, V. B., and Lam, M. S. 2005. Finding security vulnerabilities in java applications with static analysis. In *Usenix Security*.

Livshits, B.; Nori, A.; Rajamani, S.; and Banerjee, A. 2009. Merlin: specification inference for explicit information flow problems. In *PLDI*.

Mangal, R.; Zhang, X.; Nori, A. V.; and Naik, M. 2015. A user-guided approach to program analysis. In *FSE*.

McCallum, A. K.; Nigam, K.; Rennie, J.; and Seymore, K. 2000. Automating the construction of Internet portals with machine learning. In *Information Retrieval Journal*.

Mencia, C.; Previti, A.; and Marques-Silva, J. 2015. Literal-based MCS extraction. In *IJCAI*.

Milch, B.; Zettlemoyer, L. S.; Kersting, K.; Haimes, M.; and Kaelbling, L. P. 2008. Lifted probabilistic inference with counting formulas. In *AAAI*.

Myers, A. C. 1999. JFlow: practical mostly-static information flow control. In *POPL*.

Niu, F.; Ré, C.; Doan, A.; and Shavlik, J. W. 2011. Tuffy: Scaling up statistical inference in Markov Logic Networks using an RDBMS. In *VLDB*.

Noessner, J.; Niepert, M.; and Stuckenschmidt, H. 2013. RockIt: Exploiting parallelism and symmetry for MAP inference in statistical relational models. In *AAAI*.

Papadimitriou, C. H. 1994. *Computational complexity*. Addison-Wesley.

Poole, D. 2003. First-order probabilistic inference. In *IJCAI*.

Poon, H.; Domingos, P.; and Sumner, M. 2008. A general method for reducing the complexity of relational inference and its application to MCMC. In *AAAI*.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62(1-2).

Riedel, S. 2008. Improving the accuracy and efficiency of MAP inference for Markov Logic. In *UAI*.

Riedel, S. 2009. Cutting plane map inference for markov logic. In *Intl. Workshop on Statistical Relational Learning*.

Singla, P., and Domingos, P. 2005. Discriminative training of Markov Logic Networks. In *AAAI*.

Singla, P., and Domingos, P. 2006. Memory-efficient inference in relational domains. In *AAAI*.

Smaragdakis, Y., and Bravenboer, M. 2010. Using Datalog for fast and easy program analysis. In *Datalog 2.0 Workshop*.

Whaley, J., and Lam, M. 2004. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams. In *PLDI*.

Whaley, J.; Avots, D.; Carbin, M.; and Lam, M. 2005. Using Datalog with binary decision diagrams for program analysis. In *APLAS*.

Zhang, X.; Mangal, R.; Grigore, R.; Naik, M.; and Yang, H. 2014. On abstraction refinement for program analyses in Datalog. In *PLDI*.

Zhu, H.; Dillig, T.; and Dillig, I. 2013. Automated inference of library specifications for source-sink property verification. In *APLAS*.