Learning SMT($\mathcal{LRA}$) Constraints using SMT Solvers

Samuel Kolb[1], **Stefano Teso**[1], **Andrea Passerini**[2], **Luc De Raedt**[1],
[1] KU Leuven, Belgium
[2] University of Trento, Italy
{samuel.kolb, stefano.teso, luc.deraedt}@kuleuven.be, andrea.passerini@unitn.it

Abstract

We introduce the problem of learning SMT($\mathcal{LRA}$) constraints from data. SMT($\mathcal{LRA}$) extends propositional logic with (in)equalities between numerical variables. Many relevant formal verification problems can be cast as SMT($\mathcal{LRA}$) instances and SMT($\mathcal{LRA}$) has supported recent developments in optimization and counting for hybrid Boolean and numerical domains. We introduce *SMT($\mathcal{LRA}$) learning*, the task of learning SMT($\mathcal{LRA}$) formulas from examples of feasible and infeasible instances, and we contribute *INCAL*, an exact non-greedy algorithm for this setting. Our approach encodes the learning task itself as an SMT($\mathcal{LRA}$) satisfiability problem that can be solved directly by SMT solvers. INCAL is an incremental algorithm that achieves exact learning by looking only at a small subset of the data, leading to significant speed-ups. We empirically evaluate our approach on both synthetic instances and benchmark problems taken from the SMT-LIB benchmarks repository.

1 Introduction

Many modelling problems involve dealing with Boolean and numerical variables. In order to reason in these setting, researchers developed Satisfiability Modulo Linear Real Arithmetic (SMT($\mathcal{LRA}$)), a formalism that combines propositional logic with linear arithmetical expressions (inequalities, sums, products) over continuous variables [Barrett *et al.*, 2009]. Thanks to its flexibility and the availability of very efficient satisfiability solvers, like Z3 [De Moura and Bjørner, 2008] and MathSAT [Cimatti *et al.*, 2013], SMT($\mathcal{LRA}$) has found application in industrial tasks like hardware and software verification [Beyer *et al.*, 2009], as well as engineering of chemical reactions [Fagerberg *et al.*, 2012] and synthetic biology [Yordanov *et al.*, 2013]. SMT($\mathcal{LRA}$) has also contributed to recent developments in optimization and counting for hybrid Boolean and numerical domains, cf. Optimisation Modulo Theories (OMT) [Sebastiani and Tomasi, 2015] and Weighted Model Integration (WMI) [Belle *et al.*, 2015].

Several approaches to learning logical formulas from examples such as concept learning [Valiant, 1984], inductive logic programming [Muggleton and De Raedt, 1994], and constraint learning [Bessiere *et al.*, 2005] are described in the literature. However, these approaches have typically focused on discrete logical variables. The success of SMT($\mathcal{LRA}$) shows that hybrid constraints can be very useful in practice. Yet deriving such models by hand, certainly those involving the numeric constraints, can be a daunting task. Applications like modelling problems in operations research [Pawlak and Krawiec, 2017], programming from examples [Gulwani *et al.*, 2011], designing layouts from incomplete specifications [Harada *et al.*, 1995] and other complex architectures such as cyber-physical systems [Lee, 2008] all suffer from this issue. In all these cases, examples of feasible/infeasible configurations can be readily extracted from existing architectures (viewed as black-boxes) or obtained by interacting with annotators. That is why we propose to automate this task through the use of machine learning techniques. To the best of the authors' knowledge, the learning of SMT($\mathcal{LRA}$) formulas has not been considered yet.

Our first contribution is the introduction of the novel problem of *SMT($\mathcal{LRA}$) learning*, that is, the problem of inducing SMT($\mathcal{LRA}$) formulas from examples of feasible and infeasible assignments.

Our second contribution is a *non-greedy* algorithm for SMT($\mathcal{LRA}$) learning, called INCAL. The proposed approach induces formulas in CNF normal form, that is, conjunctions of clauses (disjunctions) over Boolean literals and linear inequalities. This requires to jointly identify the logical structure of the formula *and* the parameters of the inequalities appearing in it. Being non-greedy, INCAL does not need the candidate inequalities to be enumerated beforehand. Further, post-processing of the learned formula is not necessary: the learned CNF formula can be fed directly to any SMT solver.

In INCAL, the SMT($\mathcal{LRA}$) learning problem is encoded into an equivalent SMT($\mathcal{LRA}$) *satisfaction* problem. Given parameters specifying the maximum complexity (the number of terms and hyperplanes) of the candidate formula, the INCAL encoding is always guaranteed to find a solution if one exists. Despite being NP-complete in general, SMT satisfiability can be solved in practice very efficiently for many real-world problems using off-the-shelf solvers. Rather than fitting the formula on the entire dataset, INCAL employs a much faster incremental learning strategy. An initial candidate is learned from a handful of "active" examples. New examples inconsistent with the current candidate are added to the active

set until a formula consistent with all the data is found. The active set is usually small. This speeds up learning considerably, as shown by our experiments. Finally, since the complexity of the target formula may be difficult to estimate beforehand, especially if the user is not a domain expert, INCAL includes a simple search loop to automatically detect large enough values. The strategy gradually increases the formula complexity, exploiting the fact that too simple formulas can be identified and discarded very quickly.

We implemented INCAL and evaluated it empirically on both synthetic instances of increasing complexity and instances taken from the SMT-LIB benchmark repository [Barrett *et al.*, 2010]. Our results show that our method is able to quickly find accurate theories and that the incremental approach substantially reduces the number of examples used.

2 Background

We will now briefly review concepts from logic and satisfiability used in our work. Propositional logic formulas consist of literals, i.e., Boolean variables and their negations, and logical connectives, e.g., $a \wedge (b \vee \neg c)$. Any logical formula can be rewritten such that it adheres to a normal form, such as the conjunctive (CNF) or disjunctive normal form (DNF). CNF formulas consist of a conjunction of disjunctions of literals, while DNF formulas consist of a disjunction of conjunctions of literals. We use k-clause-CNF to denote the class of CNF formulas that have k clauses, i.e., a formula that is a conjunction of k disjunctions. Analogously, k-term-DNF is used to refer to formulas which consist of a disjunction of k conjunctions.

Example 1. *Consider the two formulas:*

$$\phi_1 = (a \vee \neg b) \wedge (\vee a \vee b \vee \neg c)$$
$$\phi_2 = (\neg a \wedge b) \vee (c) \vee (a \wedge \neg c)$$

then ϕ_1 is a 2-clause-CNF formula, and ϕ_2 is a 3-term-DNF formula.

Satisfiability (SAT) is the problem of deciding whether there exists an assignments of truth values to variables such that a propositional logical formula ϕ is satisfied. Such an assignment is also called a *model* of ϕ. Satisfiability Modulo Theories (SMT) generalizes SAT to deciding satisfiability for formulas with respect to a decidable background theory [Barrett *et al.*, 2009]. In this work we consider the background theory of Linear Real Arithmetic ($\mathcal{LRA}$), which restricts interpretations of numbers and operators (inequalities, sum, product) to their semantics in linear arithmetic. Slighlty abusing terminology, in this work we refer to SMT($\mathcal{LRA}$) formulas simply as SMT formulas.

Example 2. *Consider an autonomous vacuum cleaner characterized by a Boolean variable $extended$ (i.e., is it in extended mode?) and real variable $battery$ (i.e., the remaining battery percentage) and $distance$ (i.e., distance towards base station in m). Consider two constraints: 1) the vacuum cleaner is range limited to $15m$ if not in extended mode; and 2) the battery level discounted by the weighted distance of the vacuum cleaner must remain above 10%. The conjunction of these constraints can be expressed as an SMT formula $\phi^* = (extended \vee distance \leq 15) \wedge (battery - 0.01 \cdot distance \geq 0.1)$.*

Although SMT, just like SAT, is NP-complete, large instances can be solved efficiently in practice by state-of-the-art SMT solvers like Z3 [De Moura and Bjørner, 2008] and MathSAT [Cimatti *et al.*, 2013].

SMT also forms the basis for different types of inference besides satisfiability: Optimization Modulo Theories (OMT) and Weighted Model Integration (WMI). OMT [Sebastiani and Tomasi, 2015] refers to the problem of finding a model for an SMT formula ϕ that is optimal w.r.t. an objective function f. WMI [Belle *et al.*, 2015] generalizes Weighted Model Counting [Chavira and Darwiche, 2008] by integrating over the (possibly infinite) models of an SMT formula.

3 Problem Statement

Given a set of examples labeled each as positive (feasible) or negative (infeasible) according to an unknown SMT formula ϕ^*, our goal is to learn ϕ^* from the examples and their labels. Every example $e = (x_e, y_e)$ consists of a total assignment x_e of values to variables in the problem domain and a label $y_e = [\![x_e \models \phi^*]\!]$, i.e., it is `true` ($\top$) if the assignment satisfies the formula ϕ^* and `false` ($\bot$) otherwise. Given an assignment $x_e = \{v_1 = value_1, ..., v_n = value_n\}$, the value assigned to variable v_j is denoted as $x_e[v_j]$.

Example 3. *Consider the SMT formula ϕ^* of Example 2. A learning algorithm can now learn ϕ^* from examples of feasible and infeasible instances, such as:*
$(\{extended = \top, battery = 0.8, distance = 40.2\}, \top)$, $(\{extended = \top, battery = 0.2, distance = 12\}, \bot)$ *and* $(\{extended = \bot, battery = 0.7, distance = 20.3\}, \bot)$.

In order to define the search space of potential constraints, we fix the form that a learned theory must adhere to using a *bias*. The bias imposes a *hypothesis space* Φ of candidate formulas. As SMT($\mathcal{LRA}$) is typically formalized in terms of CNF formulas, and solvers also employ this format, we focus on the learning of such formulas. Every clause in the formula then constitutes a constraint. More specifically, our bias is *k-clause-CNF*, i.e., CNF formulas that contain at most k clauses.

Definition 1. A k-clause-CNF SMT($\mathcal{LRA}$) formula ϕ is of the form $\bigwedge_k C_k$, where $C_k = L_1 \vee ... \vee L_{n_k}$ and literals L_i are either a Boolean variable b, a linear inequality $\sum_j a_j \cdot x_j \leq b$, or the negation of a Boolean variable or inequality.

Note that the feasible area of a linear program (i.e. $Ax \leq b$) is a special case of k-clause-CNF with $k = \text{rows}(A)$, no Boolean variables and clauses of length 1.

We, additionally, impose an upper limit on the number of inequalities (halfspaces) h that may be used in the formula and require the real variables to be bounded (by arbitrary constants). Subsequently, we will denote the class of k-clause-CNF formulas with at most h inequalities as $\text{CNF}(k, h)$. Formally, our goal is thus:

SMT($\mathcal{LRA}$) Learning Given a set V of Boolean and bounded real variables, a set of examples E over V labeled according to an unknown SMT formula ϕ^*, and a bias-induced hypothesis space Φ, find an SMT formula $\phi \in \Phi$ such that all positive examples are satisfied and none of the negatives is.

Following the framework of concept learning, we assume that labels are noiseless and that there exists a solution to the learning problem. This setup is relevant for practical applications where label noise is unlikely, e.g., monitoring of manufacturing processes [Monostori *et al.*, 1996]. Nevertheless, this setting can be extended to take into account other criteria (such as allowing for exceptions or for the best solution w.r.t. a loss function). As SMT($\mathcal{LRA}$) learning is viewed here as a classification problem, all standard evaluation criteria and techniques apply. For the purpose of evaluating the learning method on synthetic data where the target formula ϕ^* is known, one can define the misclassification error of the learned formula ϕ w.r.t. ϕ^* as the probability that an arbitrary example is incorrectly classified[1].

4 SMT($\mathcal{LRA}$) Learning with INCAL

We start by discussing the case where the maximal number of clauses k and halfspaces h are given and fixed. In this setting, the problem we aim to solve is to find a formula $\phi \in \mathrm{CNF}(k, h)$that is satisfied by all positive and none of the negative examples. Such a formula is guaranteed to exist so long as h and k are large enough and the labels are noiseless. Appropriate values of k and h can be either provided by the user or computed automatically using the strategy discussed in Section 4.3. By encoding this search problem itself as an SMT problem we can use any off-the-shelf SMT(LRA) solver (e.g., MathSAT [Cimatti *et al.*, 2013], Z3 [De Moura and Bjørner, 2008]) to find a satisfying assignment from which we can read off ϕ. The exact encoding of this problem is described in the next section.

We stress that in contrast to most rule learning approaches, this approach is non-greedy, i.e., the formula is learned in one step rather than piece-by-piece. However, contrary to other non-greedy learners, the no-noise assumption enables us to cast learning as satisfiability rather than optimization (as done, for example, in OCT [Bertsimas and Dunn, 2017]). Searching for a satisfying formula is usually faster than searching for an optimal one in practice.

Note that, even though the learned formula ϕ fits all examples, the latter might still have non-zero misclassification loss w.r.t ϕ^*. We evaluate this potential discrepancy in our empirical analysis.

4.1 Encoding

The candidate formula $\phi \in \mathrm{CNF}(k, h)$ has k clauses and h halfspaces over bounded real variables. Clauses have variable length, i.e., each Boolean and halfspace literal (and its negation) can appear in any clause, meaning that the maximum clause length is $2(|B| + h)$. The assignment of literals to clauses is determined by two sets of decision variables. We employ $2(|B| \times k)$ variables l_{cv} and $\hat{l}_{cv}$ to encode whether Boolean variable v or its negation appear in clause c, respectively. Similarly, we use $2(h \times k)$ variables z_{cj} and $\hat{z}_{cj}$ for assigning the halfspace literals. The coefficients $a_{jv} \in \mathbb{R}$ and offsets $b_j \in \mathbb{R}$ of all halfspaces are *not* fixed beforehand: they are determined by the SMT solver along with all the other decision variables. Table 1 summarizes the symbols used in the SMT encoding.

[1]Note that Weighted Model Integration [Belle *et al.*, 2015] could be used to calculate this probability exactly.

Name	Meaning
	Constants (given)
R	The set of real variables
B	The set of Boolean variables
$k \in \mathbb{N}$	The number of clauses
$h \in \mathbb{N}$	The number of inequalities (halfspaces)
$x_e[v]$	The value assigned to variable v in example e
$y_e \in \mathbb{B}$	The Boolean label of example e
	Decision variables (determined by the solver)
$a_{jv} \in \mathbb{R}$	Coefficient of (real) variable v in inequality j
$b_j \in \mathbb{R}$	Offset of inequality j
$z_{cj} \in \mathbb{B}$	Whether clause c includes inequality j
$\hat{z}_{cj} \in \mathbb{B}$	Whether clause c includes negated inequality j
$l_{cv} \in \mathbb{B}$	Whether clause c includes (Boolean) variable v
$\hat{l}_{cv} \in \mathbb{B}$	Whether clause c includes negated (Boolean) variable v
	Auxiliary variables (determined by the solver)
$s_{ej} \in \mathbb{B}$	Whether example e satisfies inequality j
$t_{ec} \in \mathbb{B}$	Whether example e satisfies clause c

Table 1: Symbols used in the SMT($\mathcal{LRA}$) learning encoding

The examples are encoded individually using three constraints [2]. The first constraint defines s_{ej}, i.e., whether example e satisfies inequality j, which is characterized by its coefficients (a_{jv}) and offset (b_j):

$$\bigwedge_{j=1..h} s_{ej} \Leftrightarrow \sum_{v \in R} a_{jv} \cdot x_e[v] \leq b_j \tag{1}$$

The second constraint defines t_{ec}, i.e., whether example e satisfies clause c. Since CNF clauses are disjunctions, an example satisfies c iff it satisfies at least one of the (negated) inequalities or (negated) Boolean variables included in c:

$$\bigwedge_{c=1..k} \left[t_{ec} \Leftrightarrow \left(\bigvee_{j \in 1..h} ((z_{cj} \wedge s_{ej}) \vee (\hat{z}_{cj} \wedge \neg s_{ej})) \vee \bigvee_{v \in B} ((l_{cv} \wedge x_e[v]) \vee (\hat{l}_{cv} \wedge \neg x_e[v])) \right) \right] \tag{2}$$

Finally, since a CNF formula consists of a conjunction of clauses, the third constraint dictates that positive examples must satisfy every clause while negative examples must violate at least one of the clauses:

$$y_e \Leftrightarrow \bigwedge_{c \in 1..k} t_{ec} \tag{3}$$

Given a set of examples E and the example-wise encoding (summarized in Fig. 1) we can encode the full SMT($\mathcal{LRA}$) learning problem as follows:

$$\bigwedge_{e \in E} \mathrm{encoding}(e) \tag{4}$$

This encoding enables a straightforward algorithm for solving the SMT($\mathcal{LRA}$) learning problem that constructs the encoding from a set of examples and passes the encoding to an SMT solver and reads off the solution, i.e., a valid assignment to all decision- and auxiliary variables (see Table 1), if it exists.

$$\text{encoding}(e) =$$

$$\bigwedge_{j=1..h} s_{ej} \Leftrightarrow \sum_{v\in R} a_{jv} \cdot x_e[v] \leq b_j \tag{1}$$

$$\bigwedge_{c=1..k} (t_{ec} \Leftrightarrow (\bigvee_{j\in 1..h} ((z_{cj} \wedge s_{ej}) \vee (\hat{z}_{cj} \wedge \neg s_{ej})) \vee \bigvee_{v\in B} ((l_{cv} \wedge x_e[v]) \vee (\hat{l}_{cv} \wedge \neg x_e[v])))) \tag{2}$$

$$\wedge\, y_e \Leftrightarrow \bigwedge_{c\in 1..k} t_{ec} \tag{3}$$

Figure 1: Encoding the $\mathrm{CNF}(k,h)$ SMT($\mathcal{LRA}$) learning problem for one example

$a_{1,x_1} = 0.2$	$a_{1,x_2} = 0.4$	$b_1 = 1.0$	
$a_{2,x_1} = 0.3$	$a_{2,x_2} = 0.0$	$b_2 = 0.5$	
$z_{1,1} = \top$	$z_{1,2} = \bot$	$z_{2,1} = \top$	$z_{2,2} = \bot$
$\hat{z}_{1,1} = \bot$	$\hat{z}_{1,2} = \bot$	$\hat{z}_{2,1} = \bot$	$\hat{z}_{2,2} = \top$
$l_{1,b_1} = \bot$	$l_{2,b_1} = \top$	$\hat{l}_{1,b_1} = \top$	$\hat{l}_{1,b_1} = \bot$

Table 2: Example assignment A

Example 4. *Consider $k = 2$, $h = 2$, $R = \{x_1, x_2\}$, $B = \{b_1\}$ and an assignment to the decision variables A (Table 2), then A encodes the formula $\phi = (0.2x_1{+}0.4x_2 \leq 1 \vee 0.3x_1 \leq 0.5 \vee \neg b_1) \wedge (0.2x_1 + 0.4x_2 \leq 1 \vee 0.3x_1 > 0.5 \vee b_1)$.*

We note in passing that our encoding can be extended to dealing with bounded integer variables and linear equalities. Integers simply require to use integer arithmetic in place of real arithmetic, at the cost of a potentially harder satisfiability problem. Linear equalities can in principle be learned by altering Eq. 1 to include equality symbols along with inequalities.

4.2 Incremental Learning

Algorithm 1 Incremental SMT Constraint Learner (INCAL)

```
1:  procedure LEARN(E: examples)
2:      i ← 0
3:      E_i ← initial(E)
4:      while |E_i| > 0 do
5:          solver.add(⋀_{e∈E_i} encoding(e))
6:          φ_i ← solver.solve()
7:          if φ_i exists then    ▷ solver does not return unsat
8:              V_i ← {e ∈ E | y_e ≠ [[x_e ⊨ φ_i]]}
9:              E_{i+1} = selection(V_i)
10:             i ← i + 1
11:         else
12:             return unsat
13:     return φ_i
```

The number of variables and constraints in the SMT encoding of the learning problem depend directly on the number of examples. Using fewer examples simplifies the learning task, although the learned constraints might be inaccurate. Adding many examples leads to better constraints, however, it can come at a steep computational cost, as solving SMT problems is NP-complete. Therefore, we have designed and implemented a learning algorithm that tries to prune useless examples by exploiting incremental SMT solving: the *incremental SMT constraint learner* (INCAL).

The idea is that INCAL is given a large set of examples E, however, instead of building one large encoding for all of them, it selects a small subset of examples, learns an SMT formula for this subset and repeatedly extends that subset with some of the examples violated by the last learned formula (Alg. 1). Every time examples are added, their encoded SMT constraints are added (pushed) to the SMT solver such that the solver will always return a solution (formula) that violates none of the added examples. Solvers that support incremental solving will retain information between runs, such that they can solve the next iteration faster. Typically, INCAL will find a solution that is satisfied by all examples E after having added but a small subset of E to the solver, as shown by our empirical evaluation.

Different strategies can be used for the initial selection (initial, line 3) and the selection of violated examples (selection, line 9). A simple but effective strategy is to, given a set of examples to select from, return a fixed-size random subset of those examples. This approach works well in practice, however, heuristic strategies could be used to identify points that are more likely to influence the learned formula.

Theorem 1. *Given examples E, k and h, Alg. 1 will find a formula $\phi \in \mathrm{CNF}(k,h)$ that satisfies all positive examples and none of the negatives, iff such a formula exists.*

Proof. Sketch. At all steps i, either no $\phi \in \mathrm{CNF}(k,h)$ can correctly classify all examples in $E_{1:i} := E_1 \cup \ldots \cup E_i$, in which case the algorithm returns "unsat" (line 12) or ϕ_i classifies them all correctly (line 6). Now, if there are examples in $E \setminus E_{1:i}$ that are inconsistent with ϕ_i (i.e. $|V_i| > 0$), then at least one is added to E_{i+1} and the algorithm continues (line 4). Otherwise, ϕ_i is consistent with both $E_{1:i}$ and $E \setminus E_{1:i}$, whose union is E, and terminates (line 4). □

4.3 Parameter-free SMT($\mathcal{LRA}$) Learning

Until now it was assumed that the values of the parameters k and h were known. This assumption might not always be valid, especially if the user of the learning system does not have any knowledge about the domain. Too small values of k and h lead to unsatisfiable encodings, while too large values may induce formulas that are overly complex and more prone to overfitting. Our goal, therefore, is to automatically choose minimal values for k and h to allow learning formulas ϕ consistent with the examples while avoiding both issues. To this end, we define a complexity function $C(k,h)$ on both k and h. Treating both parameters as equal is achieved by using $C(k,h) = k + h$, alternatively different weights can be used or one of the parameters can be constant. For INCAL we employ a bottom-up search strategy that initializes (k,h) to their minimum values (reasonable default values or $(1,0)$) and explores configurations (k,h) in order of increasing complexity $C(k,h)$ until a consistent model $\phi \in \mathrm{CNF}(k,h)$ is found. Discarding infeasi-

[2]The auxiliary constraints specifying the range of the real variables are left implicit, due to space limitations.

ble configurations proves to be very fast in practice, especially with our incremental approach.

4.4 Learning Rules instead of Constraints

So far we have focused on learning constraints, i.e., learning CNF formulas. CNF formulas allow for compact representations and are widely used in SMT solving [Barrett *et al.*, 2009]. However, the algorithms and ideas described above can also be used to learn rules, i.e., learning DNF formulas. Changing our bias to $\mathrm{DNF}(k,h)$ (i.e., k-term-DNF with h inequalities), only requires adapting constraints 2 and 3 of $\mathrm{encoding}(e)$.

Since for DNF problems every term is a conjunction, an example is covered by a term c only if it satisfies all (negated) inequalities and (negated) Boolean variables included in c and the updated version of constraint 2 becomes:

$$\bigwedge_{c=1..k}\left[t_{ec} \Leftrightarrow \left(\bigwedge_{j\in 1..h}((z_{cj} \Rightarrow s_{ej}) \wedge (\hat{z}_{cj} \Rightarrow \neg s_{ej})) \wedge \bigwedge_{v\in B}((l_{cv} \Rightarrow x_e[v]) \wedge (\hat{l}_{cv} \Rightarrow \neg x_e[v]))\right)\right] \quad (5)$$

Additionally, since DNF problems consist of a top level disjunction, positive examples must only satisfy at least one term, while negative examples must not satisfy any term, which is reflected in the updated version of constraint 3:

$$y_e \Leftrightarrow \bigvee_{c\in 1..k} t_{ec} \quad (6)$$

Using the updated encoding, INCAL can now be used to learn DNF formulas incrementally. In fact, any bias that can be expressed by encoding examples individually, can be plugged into INCAL.

5 Evaluation

We experimentally evaluate the accuracy and performance of our approach to investigate its sensitivity w.r.t. different inputs, parameters and strategies using synthetic and real-world problems. For this evaluation we use, for every problem, random examples for learning and (different) random examples for computing the accuracy. To obtain an example, we sample assignments uniformly from the domains of a problems variables and obtain the label by computing if the assignment satisfies the target theory. Our experimental setup uses MathSAT as SMT solver, and can be found at: `https://github.com/smtlearning/smtlearning.github.io`.

Every synthetic problem $P_{r,b,k,l,h}$ consists of a randomly generated $\mathrm{CNF}(k,h)$ formula where the number of real (r) and Boolean variables (b), clauses (k), literals per terms (l), and distinct inequalities (h) are fixed. The domains of the real variables are set to $[0,1]$. The synthetic problems are generated by incrementally adding random literals and terms. Only literals/terms whose addition changes the label of at least $p\%$ of the assignments (with $p=5$) are added, to guarantee that the decision surface is non-trivial.

Real-world problems are obtained from the SMT-LIB benchmark repository. We selected a subset of problems that is both *interesting* and *feasible* to learn. Interesting problems should contain both conjunctions and disjunctions, and their *support* w.r.t. to a random set of examples E should be at least 20%, where the support is the ratio of the minority label within E. Since the benchmark problems do not specify domains for the variables, we search over various assignments of domains to problems to find the one that brings the support closest to 50%.We identified instances as feasible if they have between 1 and 9 variables and a file size smaller than 1MB (in order to discard very long formulas). 13 problem instances were identified as both interesting and feasible, unfortunately many benchmark instances are either non-interesting or non-feasible.

The research questions we want to answer are:

- **Q1** How accurate is our method?
- **Q2** How fast is our method?
- **Q3** Does incremental learning reduce the learning time?
- **Q4** How do the number of terms, literals, and inequalities in ϕ^* influence the learning algorithm?
- **Q5** How efficient is the parameter-free search?

5.1 Results

For learning a set of 1000 randomly sampled examples is used and another set of 1000 random examples is used for measuring accuracy. The speed is measured by summing up the time taken by the incremental steps for the right parameters k and h. For every setting evaluated using synthetic problems, 100 instances are generated and average results are shown. We use a timeout of $200s$ per individual learning task.

Q1. How accurate is our method? On the SMT-LIB benchmark problems we selected, our method obtained an average accuracy of 0.997 ± 0.003, which shows that it is able to accurately recover SMT formulas from data.

Q2. How fast is our method? On average, parameter free-learning with equal weights for k and h took $1.05 \pm 1.06s$ per benchmark problem.

Q3. Does incremental learning reduce the learning time? Our experiments on synthetic problems ($P_{2,6,2,3,6}$) show that INCAL is able to learn theories much faster (Fig. 2 top left) than its non-incremental counterpart without losing any accuracy (Fig. 2 bottom left). Learning from 1000 examples, our incremental approach is already 5.6 times faster and it can learn from 10000 examples without experiencing time-outs. Meanwhile, the non-incremental version times times out on 23 problems for 2500 examples. The relative number of examples included in the incremental encoding drops significantly as more examples are provided to the learning algorithm (Fig. 3), for 10000 examples INCAL uses on average 120 of them.

Q4. How do the number of terms, literals, and inequalities in ϕ^* influence the learning algorithm? The evaluation on synthetic problems ($P_{2,6,k,3,6}, k = 1..5$) shows that the number of terms k have little impact on the accuracy (it stays close to 1) but significantly influences the running time. For values of k above 4, the majority of learning problems time out ($200s$) and would require more time to solve. Scaling the number of

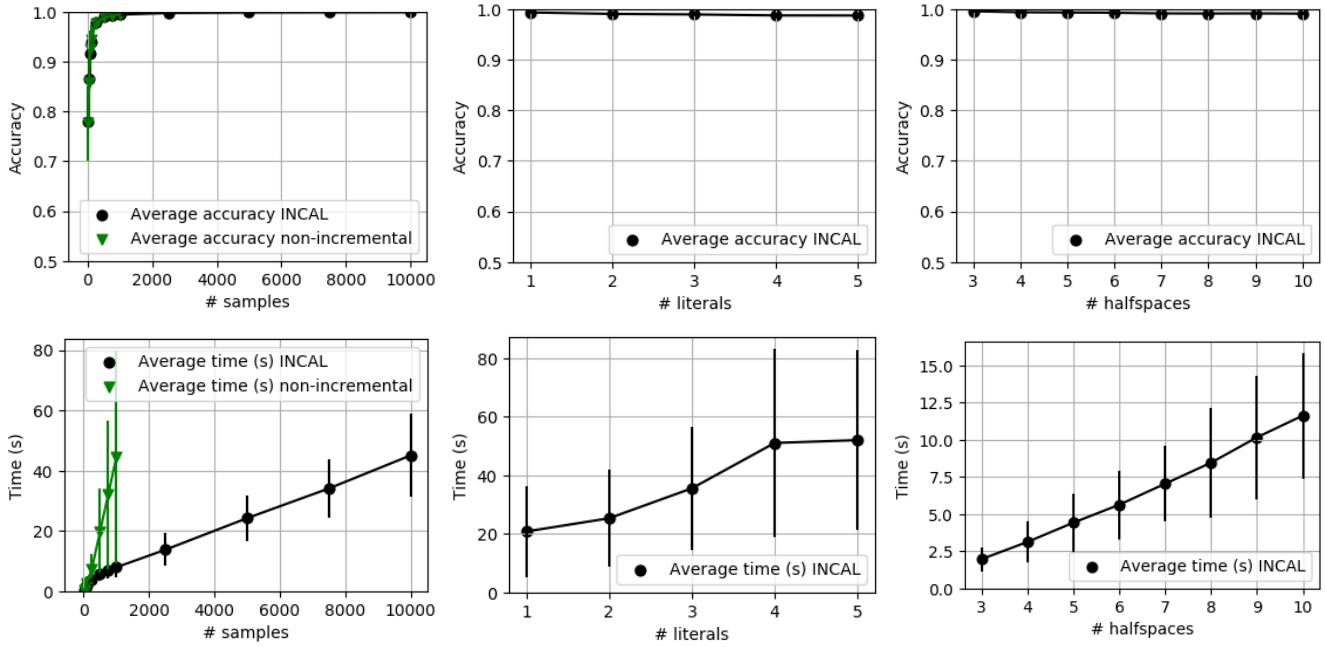

Figure 2: The plots show accuracies (top row) and running times (bottom row) for increasing number of learning samples (left column), literals per term (middle column) and inequalities (right column).

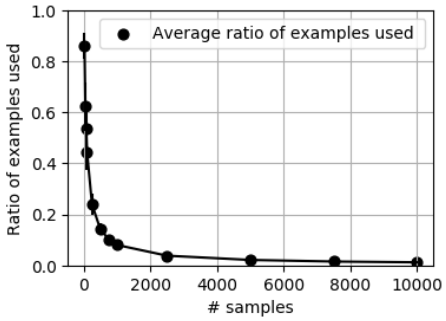

Figure 3: As more examples are provided for learning, the ratio of examples that are actually encoded by INCAL drops.

literals per term (l) on a set of problems ($P_{2,0,3,l,10}, l = 1..5$) is shown not to have a significant impact on the accuracy of the learned theories (Fig. 2 top middle). The size of the SMT encoding of the learning task is constant w.r.t. to l and the influence on the running time is limited (Fig. 2 bottom middle). Evaluation of the sensitivity w.r.t. the number of inequalities (h) on synthetic problems ($P_{2,0,2,3,h}, h = 3..10$) shows that h has a negligible impact on the accuracy (Fig. 2 top right) and demonstrates that there is controlled increase in running time (Fig. 2 bottom right).

Q5. How efficient is the parameter-free search For this evaluation on synthetic problems $P_{2,6,3,3,6}$ we compare the total time spent (total time) learning using the parameter-free approach to the time spend on learning with the final configuration (k, h) (final time). Given the results of the previous section, we chose a complexity function $C(k, h) = 3 \cdot k + h$ that favors increasing h over k. Our results show that the average ratio $\frac{\text{final time}}{\text{total time}}$ is 0.53 ± 0.18, i.e., in the majority of cases the learning time is dominated by learning the final configuration.

6 Related Work

Learning of Boolean concepts is one of the most well studied problems in machine learning [Valiant, 1984]. Here we tackle the more general problem of learning SMT($\mathcal{LRA}$) CNF concepts, which extends the Boolean CNF case to mixtures of Boolean literals and linear inequalities. Weight learning of OMT($\mathcal{LRA}$) formulas (which subsume SMT($\mathcal{LRA}$)) was tackled in [Teso *et al.*, 2015] by reduction to parameter learning of structured-output SVMs. Sparse weight learning has been employed for structure learning of the Boolean part of SMT formulas in [Campigotto *et al.*, 2011], i.e., assuming that the target halfspaces are known and fixed. The approach of [Pawlak and Krawiec, 2017] for learning constraints over the integers leverages the same strategy. Non-greedy approaches to program synthesis employ SMT for inducing a program consistent with the input specification [Gulwani *et al.*, 2017]. Several active learning strategies for SMT programs have been analyzed in [Alur *et al.*, 2013], but no working encoding is given for the $\mathcal{LRA}$ case. To the best of our knowledge INCAL is the first working approach for jointly learning the logical structure and linear constraints of SMT($\mathcal{LRA}$) formulas.

Most concept learning algorithms learn a formula by greedily adding or removing terms or literals, guided by some scoring function (e.g. [Quinlan, 1990]). Heuristics, such as look-ahead, are employed to avoid getting stuck in local optima,

and the learned model is often post-processed (e.g. pruned). The same holds for decision tree learners. It is not obvious how to extend such methods to SMT($\mathcal{LRA}$). Enumerating all possible halfspace literals to be potentially added is impossible, and the score of a single halfspace literal is difficult to define, given the non-linearity of SMT formulas.

INCAL is a non-greedy learning algorithm. Non-greedy approaches have been devised for both (oblique) decision trees [Bennett, 1994; Norouzi *et al.*, 2015; Bertsimas and Dunn, 2017] and structure learning of Bayesian networks [Cussens, 2008; Jaakkola *et al.*, 2010], with positive results. Non-greedy algorithms tend to learn simpler models that generalize well without any post-processing [Bertsimas and Dunn, 2017]. Our approach differs from existing non-greedy decision tree learners in several key aspects. First, we target CNF formulas, which for "close CNF" concepts be exponentially more compact [Mooney, 1995] than their DNF equivalent and therefore than decision trees[3], but also allow learning DNF, should that be more efficient for the application at hand. In addition, contrary to [Bennett, 1994; Bertsimas and Dunn, 2017], our approach avoids global optimization techniques, which can be harder to solve in practice than satisfiability. The core of INCAL is a reduction of the learning problem to a sequence of satisfiability checks over increasingly larger sets of "active" examples. Our main insight is that it is possible to derive formulas consistent with the entire dataset by looking only at a small subset of examples, for improved scalability (as shown in Section 5). This has not been addressed by previous methods.

Similarly to the global decision tree learner of [Bertsimas and Dunn, 2017], we provide a strategy for hyperparameter search. Our bottom-up strategy is able to quickly discard too small values, thus identifying large enough ones quickly. The strategy of [Bertsimas and Dunn, 2017] instead solves an exponential (in the tree depth) number of learning problems and then picks the best ones. An alternative is to search for the hyperparameters directly in the SMT encoding, analogously to [Norouzi *et al.*, 2015]. However, this implies that the efficiency benefits of evaluating the hyperparameters from simple to complex would be lost.

Finally, while related, polyhedral classifiers [Kantchelian *et al.*, 2014] and learners for linear programs [Jabbari *et al.*, 2016] can not be trivially extended to support logical operators.

7 Conclusion

We introduced the task of SMT($\mathcal{LRA}$) learning, where the goal is to induce an SMT($\mathcal{LRA}$) theory from examples of feasible and infeasible assignments. Our main contribution is INCAL, an exact non-greedy algorithm for learning SMT($\mathcal{LRA}$) formulas. INCAL encodes the learning step into a satisfaction problem, which can be efficiently solved by any off-the-shelf SMT solver. INCAL learns the formula by fitting a sequence of (typically small) subsets of the training set. This noticeably enhances computational efficiency and reduces the number of examples needed for effective learning. The proposed approach includes a hyperparameter search procedure to automatically tune the complexity of the formula. INCAL was validated on both synthetic instances and benchmark instances taken from the SMT-LIB benchmark repository.

Future work includes further improving the run-time of INCAL, for instance by incorporating warm starts [Bertsimas and Dunn, 2017] (e.g., exploiting incremental SMT solving also in the parameter learning step), using heuristics for example selection in the iterative step (e.g., by incorporating heuristics used in active learning) and learning from partial assignments [Bessiere *et al.*, 2013]. We also plan to investigate learning from noisy labels by either tweaking the encoding or subsampling the dataset and supporting integer variables and equalities.

[3]For instance, the CNF formula $(x_1 \vee y_1) \wedge \ldots \wedge (x_n \vee y_n)$ becomes exponential in n when converted to DNF.

Acknowledgements

This work has received funding under the European Union's Horizon 2020 research and innovation programme (ERC grant agreement No [694980] SYNTH: Synthesising Inductive Data Models and grant agreement No [732194] the QROWD project) and Samuel Kolb is supported by the Research Foundation-Flanders (FWO).

References

[Alur *et al.*, 2013] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 1–8. IEEE, 2013.

[Barrett *et al.*, 2009] Clark W Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885, 2009.

[Barrett *et al.*, 2010] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. The smt-lib standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*, volume 13, page 14, 2010.

[Belle *et al.*, 2015] Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2770–2776, 2015.

[Bennett, 1994] Kristin P Bennett. Global tree optimization: A non-greedy decision tree algorithm. *Computing Science and Statistics*, pages 156–156, 1994.

[Bertsimas and Dunn, 2017] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, pages 1–44, 2017.

[Bessiere *et al.*, 2005] Christian Bessiere, Remi Coletta, Frédéric Koriche, and Barry O'Sullivan. A sat-based version space algorithm for acquiring constraint satisfaction problems. *Machine Learning: ECML 2005*, pages 23–34, 2005.

[Bessiere *et al.*, 2013] Christian Bessiere, Remi Coletta, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Nina Narodytska, Claude-Guy Quimper, Toby Walsh, et al. Constraint acquisition via partial queries. In *IJCAI*, volume 13, pages 475–481, 2013.

[Beyer *et al.*, 2009] Dirk Beyer, Alessandro Cimatti, Alberto Griggio, M Erkan Keremoglu, and Roberto Sebastiani. Software model checking via large-block encoding. In *Formal Methods in Computer-Aided Design, 2009. FMCAD 2009*, pages 25–32. IEEE, 2009.

[Campigotto *et al.*, 2011] Paolo Campigotto, Andrea Passerini, and Roberto Battiti. Active learning of combinatorial features for interactive optimization. In *International Conference on Learning and Intelligent Optimization*, pages 336–350. Springer, 2011.

[Chavira and Darwiche, 2008] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

[Cimatti *et al.*, 2013] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The mathsat5 smt solver. In *TACAS*, volume 7795, pages 93–107. Springer, 2013.

[Cussens, 2008] James Cussens. Bayesian network learning by compiling to weighted MAX-SAT. *Proc. UAI*, pages 105–112, 2008.

[De Moura and Bjørner, 2008] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[Fagerberg *et al.*, 2012] Rolf Fagerberg, Christoph Flamm, Daniel Merkle, and Philipp Peters. Exploring chemistry using smt. In *Principles and Practice of Constraint Programming*, pages 900–915. Springer, 2012.

[Gulwani *et al.*, 2011] Sumit Gulwani, Vijay Anand Korthikanti, and Ashish Tiwari. Synthesizing geometry constructions. In *ACM SIGPLAN Notices*, volume 46, pages 50–61. ACM, 2011.

[Gulwani *et al.*, 2017] Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, et al. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119, 2017.

[Harada *et al.*, 1995] Mikako Harada, Andrew Witkin, and David Baraff. Interactive physically-based manipulation of discrete/continuous models. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 199–208. ACM, 1995.

[Jaakkola *et al.*, 2010] Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila. Learning Bayesian network structure using LP relaxations. In *Proc. of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 358–365, 2010.

[Jabbari *et al.*, 2016] Shahin Jabbari, Ryan M Rogers, Aaron Roth, and Steven Z Wu. Learning from rational behavior: Predicting solutions to unknown linear programs. In *Advances in Neural Information Processing Systems*, pages 1570–1578, 2016.

[Kantchelian *et al.*, 2014] Alex Kantchelian, Michael C Tschantz, Ling Huang, Peter L Bartlett, Anthony D Joseph, and J Doug Tygar. Large-margin convex polytope machine. In *Advances in Neural Information Processing Systems*, pages 3248–3256, 2014.

[Lee, 2008] Edward A Lee. Cyber physical systems: Design challenges. In *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on*, pages 363–369. IEEE, 2008.

[Monostori *et al.*, 1996] L Monostori, A Márkus, Hendrik Van Brussel, and E Westkämpfer. Machine learning approaches to manufacturing. *CIRP Annals-Manufacturing Technology*, 45(2), 1996.

[Mooney, 1995] Raymond J Mooney. Encouraging experimental results on learning cnf. *Machine Learning*, 19(1):79–92, 1995.

[Muggleton and De Raedt, 1994] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

[Norouzi *et al.*, 2015] Mohammad Norouzi, Maxwell Collins, Matthew A Johnson, David J Fleet, and Pushmeet Kohli. Efficient non-greedy optimization of decision trees. In *Advances in Neural Information Processing Systems*, pages 1729–1737, 2015.

[Pawlak and Krawiec, 2017] Tomasz P Pawlak and Krzysztof Krawiec. Automatic synthesis of constraints from examples using mixed integer linear programming. *European Journal of Operational Research*, 261(3):1141–1157, 2017.

[Quinlan, 1990] J Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 5(3):239–266, 1990.

[Sebastiani and Tomasi, 2015] Roberto Sebastiani and Silvia Tomasi. Optimization modulo theories with linear rational costs. *ACM Transactions on Computational Logic (TOCL)*, 16(2):12, 2015.

[Teso *et al.*, 2015] Stefano Teso, Roberto Sebastiani, and Andrea Passerini. Structured learning modulo theories. *Artificial Intelligence*, 2015.

[Valiant, 1984] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Yordanov *et al.*, 2013] Boyan Yordanov, Christoph M Wintersteiger, Youssef Hamadi, and Hillel Kugler. Smt-based analysis of biological computation. In *NASA formal methods symposium*, pages 78–92. Springer, 2013.