A Datalog Hammer for Supervisor Verification Conditions Modulo Simple Linear Arithmetic

Martin Bromberger[1], Irina Dragoste[2], Rasha Faqeh[2], Christof Fetzer[2], Markus Krötzsch[2], and Christoph Weidenbach[1]

[1] Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken
[2] TU Dresden, Dresden, Germany

Abstract. The Bernays-Schönfinkel first-order logic fragment over simple linear real arithmetic constraints BS(SLR) is known to be decidable. We prove that BS(SLR) clause sets with both universally and existentially quantified verification conditions (conjectures) can be translated into BS(SLR) clause sets over a finite set of first-order constants. For the Horn case, we provide a Datalog hammer preserving validity and satisfiability. A toolchain from the BS(LRA) prover SPASS-SPL to the Datalog reasoner VLog establishes an effective way of deciding verification conditions in the Horn fragment. This is exemplified by the verification of supervisor code for a lane change assistant in a car and an electronic control unit for a supercharged combustion engine.

1 Introduction

The supervision of cyber-physical systems is an answer to the dynamics of already deployed instances. For example, during the life cycle of a modern car, software components are regularly updated and patched. In some cases even through the internet. While the control software at the sensor/actuator physics level is typically too complex for being automatically verified, a *supervisor* of such a software guaranteeing important aspects is in scope of fully automatic verification, though challenging. While supervisor safety conditions formalized as existentially quantified properties can often already be automatically verified, conjectures about invariants formalized as universally quantified properties are a further challenge. In this paper we show that supervisor safety conditions and invariants can be automatically proven by a Datalog hammer.

The underlying logic for both formalizing supervisor behavior and formulating conjectures is the hierarchic combination of the Bernays-Schönfinkel first-order fragment with real linear arithmetic, BS(LRA), also called *Superlog* for Supervisor Effective Reasoning Logics [14]. Satisfiability of BS(LRA) clause sets is undecidable [12,19], in general, however, the restriction to simple linear real arithmetic BS(SLR) yields a decidable fragment [15,18]. Our first contribution is decidability of BS(SLR) with respect to universally quantified conjectures, Section 3, Lemma 11.

Inspired by the test point method for quantifier elimination in arithmetic [23] we show that instantiation with a finite number of first-order constants is sufficient to decide whether a universal/existential conjecture is a consequence of a BS(SLR) clause set.

For our experiments of the test point approach we consider two case studies: verification conditions for a lane change assistant in a car and an electronic control unit (ECU) for a supercharged combustion engine. The supervisors in both cases are formulated by BS(SLR) Horn clauses, the HBS(SLR) fragment. Via our test point technique they are translated together with the verification conditions to Datalog [1] (HBS). The translation is implemented in our Superlog reasoner SPASS-SPL. The resulting Datalog clause set is eventually explored by the Datalog engine VLog [10]. This hammer constitutes a decision procedure for both universal and existential conjectures. The results of our experiments show that we can verify non-trivial existential and universal conjectures in the range of seconds while state-of-the-art solvers cannot solve all problems in reasonable time. This constitutes our second contribution, Section 5.

Related Work: Reasoning about BS(LRA) clause sets is supported by SMT (Satisfiability Modulo Theories) [27,26]. In general, SMT comprises the combination of a number of theories beyond LRA such as arrays, lists, strings, or bit vectors. While SMT is a decision procedure for the BS(LRA) ground case, universally quantified variables can be considered by instantiation [30]. Reasoning by instantiation does result in a refutationally complete procedure for BS(SLR), but not in a decision procedure. The Horn fragment HBS(LRA) out of BS(LRA) is receiving additional attention [16,6], because it is well-suited for software analysis and verification. Research in this direction also goes beyond the theory of LRA and considers minimal model semantics in addition, but is restricted to existential conjectures. Other research focuses on universal conjectures, but over non-arithmetic theories, e.g., invariant checking for array-based systems [11]. Hierarchic superposition [2] and Simple Clause Learning over Theories [8] (SCL(T)) are both refutationally complete for BS(LRA). While SCL(T) can be immediately turned into a decision procedure for even larger fragments than BS(SLR) [8], hierarchic superposition needs to be refined by specific strategies or rules to become a decision procedure already because of the Bernays-Schönfinkel part [17]. Our Datalog hammer translates HBS(SLR) clause sets with both existential and universal conjectures into HBS clause sets which are also subject to first-order theorem proving. There are instance generating approaches such as implemented in iProver [21] which are a decision procedure for this fragment whereas superposition-based [2] first-order provers such as E [33], SPASS [37], Vampire [31], have additional mechanisms implemented to decide HBS. In our experiments, Section 5, we will discuss the differences of all these approaches on a number of benchmark examples in more detail.

The paper is organized as follows: after a section on preliminaries, Section 2, we present the theory of our new Datalog hammer in Section 3. Section 4 introduces our two case studies followed by experiments on respective verification conditions, Section 5. The paper ends with a discussion of the obtained results and directions for future work, Section 6. Binaries of our tools, all benchmark problems, and an extended version of this paper including all proofs can be found under https://github.com/knowsys/eval-datalog-arithmetic.

2 Preliminaries

We briefly recall the basic logical formalisms and notations we build upon. We use a standard first-order language with *constants* (denoted a, b, c), without non-constant function symbols, *variables* (denoted w, x, y, z), and *predicates* (denoted P, Q, R) of some fixed *arity*. *Terms* (denoted t, s) are variables or constants. We write $\bar{x}$ for a vector of variables, $\bar{a}$ for a vector of constants, and so on. An *atom* (denoted A, B) is an expression $P(\bar{t})$ for a predicate P of arity n and a term list $\bar{t}$ of length n. A *positive literal* is an atom A and a *negative literal* is a negated atom $\neg A$. We define $\mathrm{comp}(A) = \neg A$, $\mathrm{comp}(\neg A) = A$, $|A| = A$ and $|\neg A| = A$. Literals are usually denoted L, K, H.

A *clause* is a disjunction of literals, where all variables are assumed to be universally quantified. C, D denote clauses, and N denotes a clause set. We write $\mathrm{atoms}(X)$ for the set of atoms in a clause or clause set X. A clause is *Horn* if it contains at most one positive literal, and a *unit clause* if it has exactly one literal. A clause $A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_m$ can be written as an implication $A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m$, still omitting universal quantifiers. If Y is a term, formula, or a set thereof, $\mathrm{vars}(Y)$ denotes the set of all variables in Y, and Y is *ground* if $\mathrm{vars}(Y) = \emptyset$. A *fact* is a ground unit clause with a positive literal.

Datalog and the Bernays-Schönfinkel Fragment: The *Bernays-Schönfinkel fragment* (BS) comprises all sets of clauses. The more general form of BS in first-order logic allows arbitrary Boolean connectives and leading existential quantifiers, but both can be polynomially removed with common syntactic transformations while preserving satisfiability and all entailments that do not refer to auxiliary constants and predicates introduced in the transformation. BS theories in our sense are also known as *disjunctive Datalog programs* [13], specifically when written as implications. A set of Horn clauses is also called a *Datalog program*. (Datalog is sometimes viewed as a second-order language. We are only interested in query answering, which can equivalently be viewed as first-order entailment or second-order model checking [1].) Again, it is common to write clauses as implications in this case.

Two types of *conjectures*, i.e., formulas we want to prove as consequences of a clause set, are of particular interest: *universal* conjectures $\forall \bar{x} \phi$ and *existential* conjectures $\exists \bar{x} \phi$, where ϕ is any Boolean combination of BS atoms that only uses variables in $\bar{x}$.

A *substitution* σ is a function from variables to terms with a finite domain $\mathrm{dom}(\sigma) = \{x \mid x\sigma \neq x\}$ and codomain $\mathrm{codom}(\sigma) = \{x\sigma \mid x \in \mathrm{dom}(\sigma)\}$. We denote substitutions by σ, δ, ρ. The application of substitutions is often written postfix, as in $x\sigma$, and is homomorphically extended to non-variable terms, atoms, literals, clauses, and other formulas. A substitution σ is *ground* if $\mathrm{codom}(\sigma)$ is ground. Let Y denote some term, literal, clause, or clause set. σ is a *grounding* for Y if $Y\sigma$ is ground, and $Y\sigma$ is a *ground instance* of Y in this case. We denote by $\mathrm{gnd}(Y)$ the set of all ground instances of Y, and by $\mathrm{gnd}_B(Y)$ the set of all ground instances over a given set of constants B. The *most general unifier* $\mathrm{mgu}(Z_1, Z_2)$ of two terms/atoms/literals Z_1 and Z_2 is defined as usual, and we assume that it does not introduce fresh variables and is idempotent.

We assume a standard first-order logic model theory, and write $\mathcal{A} \models \phi$ if an interpretation $\mathcal{A}$ satisfies a first-order formula ϕ. A formula ψ is a logical consequence of ϕ, written $\phi \models \psi$, if $\mathcal{A} \models \psi$ for all $\mathcal{A}$ such that $\mathcal{A} \models \phi$. Sets of clauses are semantically treated as conjunctions of clauses with all variables quantified universally.

BS **with Linear Arithmetic:** The extension of BS with linear arithmetic over real numbers, BS(LRA), is the basis for the formalisms studied in this paper. For simplicity, we assume a one-sorted extension where all terms in BS(LRA) are of arithmetic sort LA, i.e., represent numbers. The language includes free first-order logic constants that are eventually interpreted by real numbers, but we only consider initial clause sets without such constants, called *pure* clause sets. Satisfiability of pure BS(LRA) clause sets is semi-decidable, e.g., using *hierarchic superposition* [2] or *SCL(T)* [8]. Impure BS(LRA) is no longer compact and satisfiability becomes undecidable, but it can be made decidable when restricting to ground clause sets [7].

Example 1. The following BS(LRA) clause from our ECU case study compares the values of speed (Rpm) and pressure (KPa) with entries in an ignition table (IgnTable) to derive the basis of the current ignition value (IgnDeg1):

$$\begin{aligned} &x_1 < 0 \ \lor\ x_1 \geq 13 \ \lor\ x_2 < 880 \ \lor\ x_2 \geq 1100 \ \lor\ \neg\mathrm{KPa}(x_3, x_1) \ \lor \\ &\neg\mathrm{Rpm}(x_4, x_2) \ \lor\ \neg\mathrm{IgnTable}(0, 13, 880, 1100, z) \ \lor\ \mathrm{IgnDeg1}(x_3, x_4, x_1, x_2, z) \end{aligned} \tag{1}$$

Terms of sort LA are constructed from a set $\mathcal{X}$ of *variables*, a set of *first-order arithmetic constants*, the set of integer constants $c \in \mathbb{Z}$, and binary function symbols $+$ and $-$ (written infix). Atoms in BS(LRA) are either *first-order atoms* (e.g., $\mathrm{IgnTable}(0, 13, 880, 1100, z)$) or *(linear) arithmetic atoms* (e.g., $x_2 < 880$). Arithmetic atoms may use the predicates $\leq, <, \neq, =, >, \geq$, which are written infix and have the expected fixed interpretation. Predicates used in first-order atoms are called *free*. *First-order literals* and related notation is defined as before. *Arithmetic literals* coincide with arithmetic atoms, since the arithmetic predicates are closed under negation, e.g., $\mathrm{comp}(x_2 \geq 1100) = x_2 < 1100$.

BS(LRA) clauses and conjectures are defined as for BS but using BS(LRA) atoms. We often write clauses in the form $\Lambda \parallel C$ where C is a clause solely built of free first-order literals and Λ is a multiset of LRA atoms. The semantics of $\parallel$ is implication where Λ denotes a conjunction, e.g., the clause $x > 1 \lor y \neq 5 \lor \neg Q(x) \lor R(x, y)$ is also written $x \leq 1, y = 5 \| \neg Q(x) \lor R(x, y)$. For Y a term, literal, or clause, we write $\mathrm{ints}(Y)$ for the set of all integers that occur in Y.

A clause or clause set is *pure* if it does not contain first-order arithmetic constants, and it is *abstracted* if its first-order literals contain only variables. Every clause C is equivalent to an abstracted clause that is obtained by replacing each non-variable term t that occurs in a first-order atom by a fresh variable x while adding an arithmetic atom $x \neq t$ to C. We restrict to this simplified form in formal arguments, but we prefer non-abstracted clauses in examples for readability.

The semantics of BS(LRA) is based on the standard model $\mathcal{A}^{\mathrm{LRA}}$ of linear arithmetic, which has the domain $\mathrm{LA}^{\mathcal{A}^{\mathrm{LRA}}} = \mathbb{R}$ and which interprets all arithmetic predicates and functions in the usual way. An interpretation of BS(LRA) coincides with $\mathcal{A}^{\mathrm{LRA}}$ on arithmetic predicates and functions, and freely interprets free predicates and first-order arithmetic constants. For pure clause sets this is well-defined [2]. Logical satisfaction and entailment is defined as usual, and uses similar notation as for BS.

Simpler Forms of Linear Arithmetic: The main logic studied in this paper is obtained by restricting BS(LRA) to a simpler form of linear arithmetic. We first introduce a simpler logic BS(SLR) as a well-known fragment of BS(LRA) for which satisfiability is decidable [15,18], and then present the generalization BS(LRA) PP of this formalism that we will use.

Definition 2. *The* Bernays-Schönfinkel fragment over simple linear arithmetic, BS(SLR)*, is a subset of* BS(LRA) *where all arithmetic atoms are of form $x \triangleleft c$ such that $c \in \mathbb{Z}$, $x \in \mathcal{X}$, and $\triangleleft \in \{\leq, <, \neq, =, >, \geq\}$.*

Example 3. The ECU use case leads to BS(LRA) clauses such as

$$x_1 < y_1 \ \vee \ x_1 \geq y_2 \ \vee \ x_2 < y_3 \ \vee \ x_2 \geq y_4 \ \vee \ \neg\mathrm{KPa}(x_3, x_1) \ \vee \\ \neg\mathrm{Rpm}(x_4, x_2) \ \vee \ \neg\mathrm{IgnTable}(y_1, y_2, y_3, y_4, z) \ \vee \ \mathrm{IgnDeg1}(x_3, x_4, x_1, x_2, z). \tag{2}$$

This clause is not in BS(SLR), e.g., since $x_1 > x_5$ is not allowed in BS(SLR). However, clause (1) of Example 1 is a BS(SLR) clause that is an instance of (2), obtained by the substitution $\{y_1 \mapsto 0, y_2 \mapsto 13, y_3 \mapsto 880, y_4 \mapsto 1100\}$. This grounding will eventually be obtained by resolution on the IgnTable predicate, because it occurs only positively in ground unit facts.

Example 3 shows that BS(SLR) clauses can sometimes be obtained by instantiation. Relevant instantiations can be found by *resolution*, in our case by *hierarchic resolution*, which supports arithmetic constraints: given clauses $\Lambda_1 \parallel L \vee C_1$ and $\Lambda_2 \parallel K \vee C_2$ with $\sigma = \mathrm{mgu}(L, \mathrm{comp}(K))$, their *hierarchic resolvent* is $(\Lambda_1, \Lambda_2 \parallel C_1 \vee C_2)\sigma$. A *refutation* is the sequence of resolution steps that produces a clause $\Lambda \parallel \bot$ with $\mathcal{A}^{\mathrm{LRA}} \models \Lambda\delta$ for some grounding δ. *Hierarchic resolution* is sound and refutationally complete for pure BS(LRA), since every set N of pure BS(LRA) clauses N is *sufficiently complete* [2], and hence *hierarchic superposition* is sound and refutationally complete for N [2,5]. Resolution can be used to eliminate predicates that do not occur recursively:

Definition 4 (Positively Grounded Predicate). *Let N be a set of* BS(LRA) *clauses. A free first-order predicate P is a* positively grounded predicate *in N if all positive occurrences of P in N are in ground unit clauses (also called facts).*

For a positively grounded predicate P in a clause set N, let $\mathrm{elim}(P, N)$ be the clause set obtained from N by resolving away all negative occurrences of P in N and finally eliminating all clauses where P occurs. Then N is satisfiable iff $\mathrm{elim}(P, N)$ is satisfiable. We can extend elim to sets of positively grounded predicates in the obvious way. If n is the number of P unit clauses in N, m the maximal number of negative P literals in a clause in N, and k the number of clauses in N with a negative P literal, then $|\,\mathrm{elim}(P, N)| \leq |N| + k \cdot n^m$, i.e., $\mathrm{elim}(P, N)$ is exponential in the worst case.

We further assume that elim simplifies LRA atoms until they contain at most one integer number and that LRA atoms that can be evaluated are reduced to true and false and the respective clause simplified. For example, given the pure and abstracted BS(LRA) clause set $N = \{\mathrm{IgnTable}(0, 13, 880, 1100, 2200),\ x_1 \leq x_2 \ \vee\ z_2 \geq z_1 \parallel \neg\mathrm{IgnTable}(x_1, x_2, y_1, y_2, z_1) \ \vee\ \mathrm{R}(z_2)\}$, the predicate IgnTable is positively grounded. Then $\mathrm{elim}(\mathrm{IgnTable}, N) = \{z_2 \geq 2200 \parallel \mathrm{R}(z_2)\}$ where the unifier $\sigma = \{x_1 \mapsto 0, x_2 \mapsto 13, y_1 \mapsto 880, y_2 \mapsto 110, z_1 \mapsto 2200\}$ is used to eliminate the literal $\neg\mathrm{IgnTable}(x_1, x_2, y_1, y_2, z_1)$ and $(x_1 \leq x_2)\sigma$ becomes true and can be removed.

Definition 5 (Positively Grounded BS(SLR): BS(SLR) P). *A clause set N is out of the fragment* positively grounded BS(SLR), BS(SLR) P *if* $\mathrm{elim}(S, N)$ *is out of the* BS(SLR) *fragment, where S is the set of all positively grounded predicates in N.*

Pure BS(SLR) P clause sets are called BS(SLR) PP and are the starting point for our Datalog hammer.

Lemma 6. *Let $\mathcal{A}$ be an interpretation satisfying the clause set* $\mathrm{elim}(S, N)$*. Then we can construct a satisfying interpretation $\mathcal{A}'$ for N such that $P^{\mathcal{A}'} = \{\bar{a} \in \mathbb{R}^n \mid P(\bar{a}) \in N\}$ if $P \in S$ and otherwise $P^{\mathcal{A}'} = P^{\mathcal{A}}$.*

Proof. By contradiction. Assume that $\mathcal{A}' \not\models N$. Then there must exist a clause $(\Lambda \parallel C) \in N$ and a grounding $\tau : X \to \mathbb{R}$ such that $\mathcal{A}' \not\models (\Lambda \parallel C)\tau$. We can split the clause $C = D \vee D'$ into two clauses D and D' such that D contains all literals $\neg P(\bar{t})$ from C with $P \in S$. The clause D' does not contain any positive literals $P(\bar{t})$ with $P \in S$ or else $\Lambda \parallel C$ would simplify to a fact $P(\bar{a}) \in N$ that is satisfied by $\mathcal{A}'$. Since $P^{\mathcal{A}'} = \{\bar{a} \in \mathbb{R}^n \mid P(\bar{a}) \in N\}$ for $P \in S$, we can also assume that any literal $\neg P(\bar{a})$ in $D\sigma$ must correspond to a fact $P(\bar{a}) \in N$ or D would be satisfied by $P^{\mathcal{A}'}$. This set of facts can be defined as $S' = \{P(\bar{a}) \mid \neg P(\bar{a}) \in D\sigma\}$. As a result, there exists a clause $(\Lambda' \parallel C') \in \mathrm{elim}(S, N)$ such that $(\Lambda' \parallel C')$ is the result of resolving $(\Lambda \parallel D')$ with S'; which also means that $(\Lambda' \parallel C')\tau$ is equivalent to $(\Lambda \parallel D')\tau$. Moreover, $\mathcal{A}' \models (\Lambda' \parallel C') \cdot \tau$ because $\mathcal{A}'$ behaves the same as $\mathcal{A}$ on all clauses without any literal over a $P \in S$. Hence, $\mathcal{A}' \models (\Lambda \parallel C)\tau$ which is a contradiction to our initial assumption, so $\mathcal{A}' \models N$. □

Lemma 7. *Every interpretation $\mathcal{A}$ that satisfies the clause set N also satisfies* $\mathrm{elim}(S, N)$.

Proof. By soundness of hierarchic resolution. □

3 The Theory of the Hammer

In this section, we show how to solve pure HBS(SLR) clause sets with both universally and existentially quantified conjectures. To this end, we first show how to translate any pure BS(SLR) clause set N with a universally/existentially quantified conjecture into an equisatisfiable ground and no longer pure BS(SLR) clause set over a finite set of first-order constants called *test points*. This means we reduce a quantified problem over an infinite domain into a ground problem over a finite domain. The size of the ground problem grows worst-case exponentially in the number of variables and the number of numeric constants in N and the conjecture. For the Horn case, HBS(SLR), we define a Datalog hammer, i.e., a transformation into an equisatisfiable Datalog program that is based on the same set of test points but does not require an overall grounding. It keeps the original clauses almost one-to-one instead of greedily computing all ground instances of those clauses over the test points. The Datalog hammer adds instead a finite set of Datalog facts that correspond to all theory atoms over the given set of test points. With the help of these facts and the original rules, the Datalog reasoner can then derive the same conclusions as it could have done with the ground HBS(SLR) clause set, however, all groundings that do not lead to new ground facts are neglected. Therefore, the Datalog approach is much faster in practice because the Datalog reasoner wastes no time (and space) on trivially satisfied ground rules that would have been part of the greedily computed ground HBS(SLR) clause set. Moreover, Datalog reasoners are well suited to the resulting structure of the problem, i.e., many facts but a small set of rules.

Hammering BS(SLR) Clause Sets with a Universal Conjecture: Let $\forall \bar{y}.\phi$ be a universal conjecture, where ϕ is a quantifier free pure BS(SLR) formula and $\text{vars}(\phi) = \text{vars}(\bar{y})$, let N be a BS(SLR) PP clause set, let S be the set of positively grounded predicates in N such that none of the predicates from S appear in ϕ. $N \models \forall \bar{y}.\phi$ if $\forall \bar{y}.\phi$ is satisfied by every interpretation $\mathcal{A}$ that also satisfies N, i.e., $\forall \mathcal{A}.(\mathcal{A} \models N \rightarrow \forall \bar{y}.\phi)$. This problem can be solved by negating $\forall \mathcal{A}.(\mathcal{A} \models N \rightarrow \forall \bar{y}.\phi)$. This changes our task to finding a counter example, i.e., one interpretation $\mathcal{A}$ that satisfies N but does not satisfy $\forall \bar{y}.\phi$, or formally: $\exists \mathcal{A}.(\mathcal{A} \models N \wedge \exists \bar{y}.\neg\phi)$. For the negated formulation, we can restrict our solution space from the infinite reals to a finite set of test points and still preserve satisfiability. This *finite abstraction* is computed as follows:

First, we partition $\mathbb{R}$ into intervals. The endpoints of the intervals are defined by the following set constructed from the variable bounds in $\text{elim}(S, N)$ and ϕ:

$$\begin{aligned}\mathcal{C} = \{&(c, 0, 1) \mid x \triangleleft c \in \text{atoms}(\text{elim}(S, N)) \cup \text{atoms}(\phi) \text{ where } \triangleleft \in \{\leq, =, \neq, >\}\} \cup \\ \{&(c, 1, -1) \mid x \triangleleft c \in \text{atoms}(\text{elim}(S, N)) \cup \text{atoms}(\phi) \text{ where } \triangleleft \in \{\leq, =, \neq, >\}\} \cup \\ \{&(c, 0, -1) \mid x \triangleleft c \in \text{atoms}(\text{elim}(S, N)) \cup \text{atoms}(\phi) \text{ where } \triangleleft \in \{\geq, =, \neq, <\}\} \cup \\ \{&(c, -1, 1) \mid x \triangleleft c \in \text{atoms}(\text{elim}(S, N)) \cup \text{atoms}(\phi) \text{ where } \triangleleft \in \{\geq, =, \neq, <\}\} \cup \\ \{&(-\infty, 1, -1), (\infty, -1, 1)\}\end{aligned}$$

It is not necessary to compute $\text{elim}(S, N)$ to compute $\mathcal{C}$. It is enough to iterate over all theory atoms in N and compute all of their instantiations in $\text{elim}(S, N)$ based on the facts in N for predicates in S. This can be done in $O(\mu(n_v) \cdot n_A \cdot n_S^{n_v})$, where n_v is the maximum number of variables in any theory atom in N, n_A is the number of theory atoms in N, n_S is the number of facts in N for predicates in S, and $\mu(x)$ is the time needed to simplify a theory constraint over x variables to a variable bound.

The intervals themselves can be constructed by sorting the above set in an ascending lexicographical order. The result is a sequence $[\ldots, (l, \delta_l, -1), (u, \delta_u, 1) \ldots]$, where we always have one triple of the form $(l, \delta_l, -1)$, representing a lower bound, followed by one triple of the form $(u, \delta_u, 1)$, representing an upper bound. We can turn such a pair of triples back into an interval with the function $\text{intv}((l, \delta_l, -1), (u, \delta_u, 1))$. $\text{intv}((l, \delta_l, -1), (u, \delta_u, 1))$ returns (l, u) if $(\delta_l, \delta_u) = (1, -1)$, $[l, u)$ if $(\delta_l, \delta_u) = (0, -1)$, $(l, u]$ if $(\delta_l, \delta_u) = (1, 0)$, and $[l, u]$ if $(\delta_l, \delta_u) = (0, 0)$. If we map intv to each subsequent pair of triples, then we receive our partition of intervals $\mathcal{I} = \text{ipart}(N, \phi)$. Our intervals partition the reals in such a way that either all values inside one of our intervals $I \in \mathcal{I}$ uniformly satisfy a variable bound from $\text{atoms}(\text{elim}(S, N)) \cup \text{atoms}(\phi)$ or none do.

Corollary 8. *Let* $\triangleleft \in \{<, \leq, =, \neq, \geq, >\}$. *For each interval* $I \in \mathcal{I}$, *every two points* $a, b \in I$, *and every variable bound* $x \triangleleft c \in \text{atoms}(\text{elim}(S, N)) \cup \text{atoms}(\phi)$, $a \triangleleft c$ *if and only if* $b \triangleleft c$.

We will now construct a finite abstraction ψ of $N \wedge \exists \bar{y}.\neg\phi$ over B, i.e., a grounding of N and ϕ over a finite set of test points B that is equisatisfiable. The variables $\text{vars}(\phi)$ in the universal conjecture may stand for up to $|\text{vars}(\phi)|$ different points in any given interval that is not just a point. Therefore, a counter example for ϕ may require up to $|\text{vars}(\phi)|$ different values in one interval. This means that the finite abstraction needs to pick $|\text{vars}(\phi)|$ test points/constants for every interval that is not just one point. We will ensure that a test point a belongs to a certain interval I by adding a set of variable bounds to our formula. We define these bounds with the two functions ilbd and iubd that turn intervals into lower and upper bounds, respectively: $\mathrm{ilbd}((-\infty, u), x) = \emptyset$, $\mathrm{ilbd}((-\infty, u], x) = \emptyset$, $\mathrm{ilbd}((l, u), x) = \{l < x\}$, $\mathrm{ilbd}((l, u], x) = \{l < x\}$, $\mathrm{ilbd}([l, u), x) = \{l \le x\}$, $\mathrm{ilbd}([l, u], x) = \{l \le x\}$ for $l \neq -\infty$; $\mathrm{iubd}((l, \infty), x) = \emptyset$, $\mathrm{iubd}([l, \infty), x) = \emptyset$, $\mathrm{iubd}((l, u), x) = \{x < u\}$, $\mathrm{iubd}((l, u], x) = \{x \le u\}$, $\mathrm{iubd}([l, u), x) = \{x < u\}$, $\mathrm{iubd}([l, u], x) = \{x \le u\}$ for $u \neq \infty$.

For the remaining subsection, we fix the following notations: $m = \max(1, |\mathrm{vars}(\phi)|)$ and $\mathcal{I} = \mathrm{ipart}(N, \phi)$; $\mathcal{I}_= = \{I \in \mathcal{I} \mid I = [l, l]\}$ is the set of all intervals from $\mathcal{I}$ that are just points; $\mathcal{I}_\infty = \mathcal{I} \setminus \mathcal{I}_=$ is the set of all intervals that are not just points and therefore contain infinitely many values; $B = \{a_{I,1} | I \in \mathcal{I}_=\} \cup \{a_{I,j} | I \in \mathcal{I}_\infty \text{ and } j = 1, \ldots, m\}$ is the set of test points for our abstraction; $\mathrm{idef}(B) = \bigcup_{a_{I,i} \in B} \mathrm{ilbd}(I, a_{I,i}) \cup \bigcup_{a_{I,i} \in B} \mathrm{iubd}(I, a_{I,i})$ is a set of bounds that defines to which interval each constant belongs; and $\psi = \mathrm{gnd}_B(N) \cup \mathrm{idef}(B) \wedge (\bigvee_{\rho:\mathrm{vars}(\phi) \to B} \neg\phi\rho)$ is the finite abstraction of $N \wedge \exists \bar{y}.\neg\phi$.

Lemma 9. *Let $\mathcal{A}'$ be an interpretation satisfying the finite abstraction ψ of $N \wedge \exists \bar{y}.\neg\phi$. Moreover, let $\rho : \mathrm{vars}(\phi) \to B$ be a substitution such that $\mathcal{A}'$ satisfies $\neg\phi\rho$. Then the interpretation $\mathcal{A}$ satisfies $N \wedge \exists \bar{y}.\neg\phi$ if it is constructed as follows:*
$P^{\mathcal{A}} = \{\bar{a} \in \mathbb{R}^n \mid P(\bar{a}) \in N\}$ if $P \in S$ and $P^{\mathcal{A}} = \{\bar{a} \in \mathbb{R}^n \mid \bar{a}\sigma \in P^{\mathcal{A}'}\}$ if $P \notin S$ and $\sigma = \{a \mapsto a^{\mathcal{A}'}_{I,1} \mid I \in \mathcal{I}$ and $a \in I \setminus \{a^{\mathcal{A}'}_{I,2}, \ldots, a^{\mathcal{A}'}_{I,n}\}\}$.

Proof. We split this proof into two parts: 1) we show that $\mathcal{A}$ satisfies N; 2) after that we show that there exists a $\rho' : \mathrm{vars}(\phi) \to \mathbb{R}$ such that $\mathcal{A}$ also satisfies $\neg\phi\rho'$.

1) We show that $\mathcal{A}$ satisfies $\mathrm{elim}(S, N)$ because then Lemma 6 implies that $\mathcal{A}$ also satisfies N. We start by constructing another substitution

$$\sigma' = \left\{a \mapsto a_{I,1} \mid a \in I \in \mathcal{I} \setminus \left\{a^{\mathcal{A}'}_{I,2}, \ldots, a^{\mathcal{A}'}_{I,n}\right\}\right\} \cup \left\{a \mapsto a_{I,j} \mid a = a^{\mathcal{A}'}_{I,j}\right\}.$$

We know that $\mathcal{A}' \models (\Lambda \parallel C)\tau\sigma'$ because $\mathcal{A}' \models N'$ and there exists a $(\Lambda' \parallel C') \in \mathrm{gnd}_B(N)$ and a set of ground facts $S' \in \mathrm{gnd}_B(N)$ that resolve to $(\Lambda \parallel C)\tau\sigma'$ by definition of $\mathrm{elim}(S, N)$. Due to the definition of $\mathcal{A}$, we also know that $\bar{a} \in P^{\mathcal{A}}$ if and only if $\bar{a}\sigma' \in P^{\mathcal{A}'}$. Similarly, Corollary 8 and the set of bounds $\mathrm{idef}(B)$ in ψ imply that $a \in \mathbb{R}$ satisfies a variable bound from $\mathrm{elim}(S, N)$ if and only if $a\sigma$ satisfies it. Therefore, $\mathcal{A} \models \Lambda \parallel C\tau$ if and only if $\mathcal{A}' \models (\Lambda \parallel C)\tau\sigma'$. Hence, $\mathcal{A}$ satisfies $\mathrm{elim}(S, N)$ and by Lemma 6 also N.

2) We start by constructing a substitution from the variables of ϕ to $\mathbb{R}$: $\rho' = \{y_i \mapsto a^{\mathcal{A}'}_{I,j} \mid y_i\rho = a_{I,j}\}$. $\mathcal{A}$ satisfies $\neg\phi\rho'$ because $\mathcal{A}'$ satisfies $\neg\phi\rho$.

Now the two subproofs combined prove that $\mathcal{A}$ is satisfying $N \wedge \exists \bar{y}.\neg\phi$. □

Lemma 10. *Let $\mathcal{A}$ be an interpretation satisfying the formula $N \wedge \exists \bar{y}.\neg\phi$. Then we can construct an interpretation $\mathcal{A}'$ that satisfies its finite abstraction ψ.*

Proof. If $\mathcal{A}$ satisfies $N \wedge \exists \bar{y}.\neg\phi$, then (i) $\mathcal{A}$ satisfies N and (ii) there exists a $\rho' : \mathrm{vars}(\phi) \to \mathbb{R}$ such that $\mathcal{A}$ also satisfies $\neg\phi\rho'$. The image B' of ρ' can be defined as follows: $B' = \{b \in \mathbb{R} \mid b = y_i\rho'$ and $y_i \in \mathrm{vars}(\phi)\}$. Next we select one value $a_I \in I$ for each interval I. Based on the values a_I and the set B', we now extend the interpretation $\mathcal{A}$ to our constants B:
$a^{\mathcal{A}}_{I,j} := a_I$ if $j > |B' \setminus I|$; $a^{\mathcal{A}}_{I,j} := b_j$ if $j \le |B' \setminus I|$ and $B' \setminus I = \{b_1, \ldots, b_m\}$.
Moreover, we construct a substitution $\rho = \{y_i \mapsto a_{I,j} \mid y_i\rho' = a^{\mathcal{A}}_{I,j}\}$ that maps the

variables in ϕ to a subset of the constants B. By definition of $\mathcal{A}$, $\mathcal{A}$ satisfies $\text{gnd}_B(N)$ because $\mathcal{A}$ satisfies N. Due to the way we extended $\mathcal{A}$ over the constants B, $\mathcal{A}$ also satisfies each bound in $\text{idef}(B)$. By definition of ρ and ρ', $\mathcal{A}$ also satisfies $\neg\phi\rho$ and therefore $\left(\bigvee_{\rho:\text{vars}(\phi)\to B} \neg\phi\rho\right)$. Hence, the extended $\mathcal{A}$ satisfies ψ. □

Lemma 11. *$N \wedge \exists\bar{y}.\neg\phi$ has a satisfying interpretation if and only if its finite abstraction ψ has a satisfying interpretation.*

Proof. The first part of the equivalence follows from Lemma 9. The second part follows from Lemma 10. □

The finite abstraction for the case with a universal conjecture can also be used to construct a finite abstraction for the case without a conjecture and the case with an existential conjecture. Let N be a BS(SLR) PP clause set and let S be the set of all positively grounded predicates in N. N is satisfiable if and only if $N \not\models \bot$. Hence, we get a finite abstraction for N if we build one for $N \models \bot$, which can be treated as a universal conjecture because all variables in $\bot$ are universally quantified. The existential case works similarly: $N \models \exists\bar{y}.\phi$ if and only if $N \cup N' \models \bot$, where N' is the universal BS(SLR) clause set we get from applying a CNF transformation [28] to $\forall\bar{y}.\neg\phi$.

A Datalog Hammer for HBS(SLR) PP: The set $\text{gnd}_B(N)$ grows exponentially with regard to the maximum number of variables n_C in any clause $(\Lambda \parallel C) \in N$, i.e., $O(|\text{gnd}_B(N)|) = O(|N| \cdot |B|^{n_C})$. Since B is large for realistic examples (e.g., in our examples the size of B ranges from 15 to 1609 constants), the finite abstraction is often too large to be solvable in reasonable time. As an alternative approach, we propose a Datalog hammer for the Horn fragment of BS(SLR) PP clause sets, called HBS(SLR)PP. This hammer exploits the ideas behind the finite abstraction and will allow us to make the same ground deductions, but the resulting formula is much more concise.

The Datalog hammer takes as input (i) a HBS(SLR)PP clause set N (where S is the set of all positively grounded predicates in N) and (ii) optionally a universal conjecture $\forall\bar{y}.P(\bar{y})$ where $P \notin S$. Restricting the conjecture to a single positive literal may seem like a drastic restriction, but we will later show that we can transform any universal conjecture into this form if it contains only positive atoms. Given this input, the Datalog hammer first computes the same interval partition $\mathcal{I}$ and test point/constant set B needed for the finite abstraction. Then it computes an assignment β for the constants in B that corresponds to the interval partition, i.e., $a_{I,i}\beta \in I$ and $a_{I,i}\beta \neq a_{I,j}\beta$ if $i \neq j$. Next, it computes three clause sets that will make up the Datalog formula. The first set $\text{tren}_N(N)$ is computed out of N by replacing each theory atom A in N with a literal $P_A(\bar{x})$, where $\text{vars}(A) = \text{vars}(\bar{x})$ and P_A is a fresh predicate. This is necessary to eliminate all non-constant function symbols (e.g., $+, -$) in positively grounded theory atoms because Datalog does not support non-constant function symbols. (It is possible to reduce the number of fresh predicates needed, e.g., by reusing the same predicate for two theory atoms that are equivalent up to variable renaming.) The second set is empty if we have no universal conjecture or it contains the ground and negated version ϕ of our universal conjecture $\forall\bar{y}.P(\bar{y})$. Since we restricted the conjecture to a single positive literal, ϕ has the form $C_\phi \to \bot$, where C_ϕ contains all literals $P(\bar{y})\rho$ for all groundings

ρ : vars$(\bar{y}) \to B$. We cannot skip this grounding but the worst-case size of C_ϕ is $O(\text{gnd}_B(N)) = O(|B|^{n_\phi})$, where $n_\phi = |\bar{y}|$, which is in our applications typically much smaller than the maximum number of variables n_C contained in any clause in N.The last set is denoted by tfacts(N, B) and contains a fact tren$_N(A)$ for every ground theory atom A contained in the constraint Λ of a clause $(\Lambda \parallel C) \in \text{gnd}_B(N)$ such that $A\beta$ simplifies to true. (Alternatively, it is also possible to use a set of axioms and a smaller set of facts and let the Datalog reasoner compute all relevant theory facts for itself.) The set tfacts(N, B) can be computed without computing gnd$_B(N)$ if we simply iterate over all theory atoms A in all constraints Λ of all clauses $(\Lambda \parallel C) \in N$ and compute all groundings τ : vars$(A) \to B$ such that $A\tau\beta$ simplifies to true. This can be done in time $O(n_L \cdot |B|^{n_v})$ and the resulting set tfacts(N, B) has worst-case size $O(n_A \cdot |B|^{n_v})$, where n_L is the number of literals in N, n_v is the maximum number of variables $|\text{vars}(A)|$ in any theory atom A in N, and n_A is the number of different theory atoms in N. Please note that already satifiability testing for BS clause is NEXPTIME-complete in general, and DEXPTIME-complete for the Horn case [22,29]. So when abstracting to a polynomially decidable clause set (ground HBS) an exponential factor is unavoidable.

Lemma 12. *$N \wedge \exists\bar{y}.\neg P(\bar{y})$ is equisatisfiable to its hammered version N_D =* tren$_N(N) \cup$ tfacts$(N, B) \cup \{\phi\}$. *N is equisatisfiable to its hammered version* tren$_N(N) \cup$tfacts(N, B).

Proof. Let Π' be the set of new predicate symbols introduced by tren$_N$. We will prove that $\psi = \text{gnd}_B(N) \cup \text{idef}(B) \cup \{\phi\}$, the finite abstraction of $N \wedge \exists\bar{y}.\neg P(\bar{y})$, is equisatisfiable to $N_D = \text{tren}_N(N) \cup \text{tfacts}(N, B) \cup \{\phi\}$. Then we get from Lemma 11 that $N \wedge \exists\bar{y}.\neg P(\bar{y})$ is equisatisfiable to its hammered version. The case for N without conjecture works exactly the same.
$\Rightarrow$: Let $\mathcal{A}$ be an interpretation satisfying ψ. Then we can extend $\mathcal{A}$ over Π' so it also satisfies N_D. The extension sets exactly those arguments for $P_A \in \Pi'$ to true that appear in tfacts(N, B), i.e., if $P_A \in \Pi'$, then $P_A^{\mathcal{A}} = \{\bar{a}^{\mathcal{A}} \mid P_A(\bar{a}) \in \text{tfacts}(N, B)\}$. As a result, $\mathcal{A}$ automatically satisfies tfacts(N, B) and $\mathcal{A}$ also trivially satisfies ϕ because it also appears in ψ. Moreover, we can proof that $\mathcal{A}$ satisfies any clause $D \vee C \in \text{gnd}_B(\text{tren}_N(N))$ by case distinction over the corresponding clause $\Lambda \parallel C \in \text{gnd}_B(N)$ with $\text{tren}_N(\Lambda) = \neg D$: since $\mathcal{A}$ satisfies $\Lambda \parallel C$ (i) either $\mathcal{A}$ satisfies C or (ii) $\mathcal{A}$ does not satisfy Λ and therefore one of the atoms $P(\bar{a})$ in tren$_N(\Lambda)$ does not appear in tfacts(N, B) by definition of tfacts and thus D that contains $\neg P(\bar{a})$ is satisfied by $\mathcal{A}$. Hence, $\mathcal{A}$ satisfies $tren_N(N)$.
$\Leftarrow$: Let $\mathcal{A}$ be an interpretation satisfying N_D. Then there exists an $\mathcal{A}'$ that satisfies ψ. $\mathcal{A}'$ interprets each constant $a_{I,i}$ in B as $a_{I,i}\tau$ and each predicate $P \in \Pi$ as $P^{\mathcal{A}'} = \{\bar{a}\tau \mid \bar{a} \in P^{\mathcal{A}}\}$. By definition of τ, $\mathcal{A}'$ satisfies idef(B). As in the previous case, $\mathcal{A}'$ satisfies ϕ because $\mathcal{A}$ satisfies ϕ. Moreover, we can proof that $\mathcal{A}'$ satisfies any clause $\Lambda \parallel C \in \text{gnd}_B(N)$ by case distinction over the corresponding clause $D \vee C \in \text{gnd}_B(\text{tren}_N(N))$ with $\text{tren}_N(\Lambda) = \neg D$: since $\mathcal{A}$ satisfies $D \vee C$ (i) either $\mathcal{A}$ satisfies C and therefore $\mathcal{A}'$ satisfies C or (ii) $\mathcal{A}$ satisfies D and therefore at least one of the atoms $P(\bar{a}) = \text{tren}_N(A)$ with $A \in \Lambda$ does not appear in tfacts(N, B), which can only be that case if $A\tau$ is not satisfiable and thus Λ is not satisfied by $\mathcal{A}'$. Hence, $\mathcal{A}'$ also satisfies gnd$_B(N)$. □

Note that tren$_N(N) \cup$ tfacts$(N, B) \cup \{\phi\}$ is actually a HBS clause set over a finite set of constants B and not yet a Datalog input file. It is well known that such a formula can be transformed easily into a Datalog problem by adding a ground Goal atom and adding it as a positive literal to any clause without a positive literal. Querying for the Goal atom returns true if the HBS clause set was unsatisfiable and false otherwise.

Positive Conjectures: One of the seemingly biggest restrictions of our Datalog hammer is that it only accepts universal conjectures over a single positive literal $\forall \bar{y}. P(\bar{y})$. We made this restriction because it is the easiest way to guarantee that our negated and finitely abstracted goal takes the form of a Horn clause. However, there is a way to express any positive universal conjecture — i.e., any universal conjecture where all atoms have positive polarity — as a universal conjecture over a single positive literal. (Note that any negative theory literal can be turned into a positive theory literal by changing the predicate symbol, e.g., $\neg(x \leq 5) \equiv (x > 5)$.) Similarly as in a typical first-order CNF transformation [28], we can simply rename all subformulas, i.e., recursively replace all subformulas with some some fresh predicate symbols and add suitable Horn clause definitions for these new predicates to our clause set N.

Let $\forall \bar{y}.\phi'$ be a universal conjecture where all atoms have positive polarity. Then we define the functions $\mathrm{rflat}(\phi')$ and $\mathrm{pflat}(\phi')$ recursively as follows: $\mathrm{pflat}(\phi')$ returns an atom $P_{\phi'}(\bar{x})$ over a fresh predicate $P_{\phi'}$ for any formula ϕ' with $\mathrm{vars}(\phi') = \mathrm{vars}(\bar{x})$ that is not just a free first-order atom and otherwise the atom itself. $\mathrm{rflat}(\phi')$ on the other hand introduces a set of new rules that define the fresh predicates $P_{\phi'}$: $\mathrm{rflat}(\phi') := \{(\mathrm{pflat}(\phi'_1), \ldots, \mathrm{pflat}(\phi'_m) \to \mathrm{pflat}(\phi'))\} \cup \mathrm{rflat}(\phi'_1) \cup \ldots \cup \mathrm{rflat}(\phi'_m)$ if $\phi' = \phi'_1 \wedge \ldots \wedge \phi'_m$, $\mathrm{rflat}(\phi') := \{(\mathrm{pflat}(\phi'_1) \to \mathrm{pflat}(\phi')), \ldots, (\mathrm{pflat}(\phi'_m) \to \mathrm{pflat}(\phi'))\} \cup \mathrm{rflat}(\phi'_1) \cup \ldots \cup \mathrm{rflat}(\phi'_m)$ if $\phi' = \phi'_1 \vee \ldots \vee \phi'_m$, $\mathrm{rflat}(\phi') := \{(\Lambda \parallel\!\to \mathrm{pflat}(\phi'))\}$ if $\phi' = \Lambda$ is a theory constraint, and $\mathrm{rflat}(\phi') := \emptyset$ if ϕ' is a free first-order atom.

Lemma 13. *Let $\mathcal{A}$ be an interpretation that satisfies N. Let $\tau : \mathrm{vars}\,\phi^* \to \mathbb{R}$ be a grounding for ϕ'. Then $\mathcal{A} \models \phi'\tau$ if and only if $(\mathcal{A} \wedge \mathrm{rflat}(\phi')) \models \mathrm{pflat}(\phi')\tau$.*

Proof. $\Rightarrow$: Assume $\mathcal{A}$ is an interpretation that satisfies $\mathcal{A} \models \phi'\tau$. Then we show by induction that $\mathcal{A} \wedge \mathrm{rflat}(\phi') \models \mathrm{pflat}(\phi^*)\tau$ for all subformulas ϕ^* of ϕ', where $\mathcal{A} \models \phi^*\tau$. Case 1: if ϕ^* is a free first-order atom, then $(\mathcal{A} \wedge \mathrm{rflat}(\phi')) \models \mathrm{pflat}(\phi^*)\tau$ because $\mathrm{pflat}(\phi^*) = \phi^*\tau$. Case 2: if $\phi^* = \Lambda$ is a theory constraint, then $(\Lambda \parallel\!\to \mathrm{pflat}(\phi^*)) \in \mathrm{rflat}(\phi')$ and $\mathcal{A} \models \phi^*\tau$ entails $(\mathcal{A} \wedge \mathrm{rflat}(\phi')) \models \mathrm{pflat}(\phi^*)\tau$. Case 3: if $\phi^* = \phi^*_1 \vee \ldots \vee \phi^*_m$, then there must exist a ϕ^*_j with $\mathcal{A} \models \phi^*_j\tau$ and by induction hypothesis $(\mathcal{A} \wedge \mathrm{rflat}(\phi')) \models \mathrm{pflat}(\phi^*_j)\tau$. Together with $(\mathrm{pflat}(\phi^*_j) \to \mathrm{pflat}(\phi^*)) \in \mathrm{rflat}(\phi')$ this means $(\mathcal{A} \wedge \mathrm{rflat}(\phi')) \models \mathrm{pflat}(\phi^*)\tau$. Case 4: if $\phi^* = \phi^*_1 \wedge \ldots \wedge \phi^*_m$, then $\mathcal{A} \models \phi^*_j\tau$ for all ϕ^*_j and by induction hypothesis $(\mathcal{A} \wedge \mathrm{rflat}(\phi')) \models \mathrm{pflat}(\phi^*_j)\tau$ for all ϕ^*_j. Together with $(\mathrm{pflat}(\phi^*_1), \ldots, \mathrm{pflat}(\phi^*_m) \to \mathrm{pflat}(\phi^*)) \in \mathrm{rflat}(\phi')$ this means $(\mathcal{A} \wedge \mathrm{rflat}(\phi')) \models \mathrm{pflat}(\phi^*)\tau$.
$\Leftarrow$: Assume $\mathcal{A}$ is an interpretation that satisfies $(\mathcal{A} \wedge \mathrm{rflat}(\phi')) \models \mathrm{rflat}(\phi')\tau$. Then we show by induction that $\mathcal{A} \models \phi^*\tau$ for all subformulas ϕ^* of ϕ', where $(\mathcal{A} \wedge \mathrm{rflat}(\phi')) \models \mathrm{pflat}(\phi^*)\tau$. Case 1: if ϕ^* is a free first-order atom, then $\mathrm{pflat}(\phi^*) = \phi^*$ only appears as a negative literal in $\mathrm{rflat}(\phi')$. Hence, $\mathcal{A} \models \phi^*\tau$. Case 2: if $\phi^* = \Lambda$ is a theory constraint, then $(\Lambda \parallel\!\to \mathrm{pflat}(\phi^*)) \in \mathrm{rflat}(\phi')$ is the only clause in $N \cup \mathrm{rflat}(\phi')$, where $\mathrm{pflat}(\phi^*)$ appears positively. Hence, $\mathcal{A} \models \phi^*\tau$. Case 3: if $\phi^* = \phi^*_1 \vee \ldots \vee \phi^*_m$, then there only exist m clauses in $\mathrm{rflat}(\phi')$ where $\mathrm{pflat}(\phi')$ is positive. This means $(\mathcal{A} \wedge \mathrm{rflat}(\phi')) \models$

$\text{pflat}(\phi^*)\tau$ can only be true if there exist a rule $(\text{pflat}(\phi_j^*) \rightarrow \text{pflat}(\phi^*)) \in \text{rflat}(\phi')$ with $(\mathcal{A} \wedge \text{rflat}(\phi')) \models \text{pflat}(\phi_j^*)\tau$ and by induction hypothesis $\mathcal{A} \models \phi_j^*\tau$. However, this also means $\mathcal{A} \models \phi^*\tau$. Case 4: if $\phi^* = \phi_1^* \wedge \ldots \wedge \phi_m^*$, then $(\text{pflat}(\phi_1^*), \ldots, \text{pflat}(\phi_m^*) \rightarrow \text{pflat}(\phi^*)) \in \text{rflat}(\phi')$ is the only clause in $\text{rflat}(\phi')$ where $\text{pflat}(\phi')$ is positive. This means $(\mathcal{A} \wedge \text{rflat}(\phi')) \models \text{pflat}(\phi^*)\tau$ can only be true if $(\mathcal{A} \wedge \text{rflat}(\phi')) \models \text{pflat}(\phi_j^*)\tau$ for all ϕ_j^* and by induction hypothesis $\mathcal{A} \models \phi_j^*\tau$. However, this also means $\mathcal{A} \models \phi^*\tau$. □

Corollary 14. $N \models (\forall \bar{y}.\phi')$ *is equivalent to* $(N \cup \text{rflat}(\phi')) \models (\forall \bar{y}.\, \text{pflat}(\phi'))$.

Using the same technique, we can also express any positive existential conjecture — i.e., any existential conjecture where all atoms have positive polarity — as additional clauses in our set of input clauses N.

Corollary 15. $N \models (\exists \bar{y}.\phi')$ *is equivalent to* $N \cup \text{rflat}(\phi') \cup (\text{pflat}(\phi') \rightarrow \bot)$ *is unsatisfiable.*

4 Two Supervisor Case Studies

We consider two supervisor case studies: a lane change assistant and the ECU of a suprecharged combustion engine.

1. Lane Assistant Modern dynamic dependable systems (e.g., autonomous driving) continuously update software components to fix bugs and to introduce new features. However, the safety requirement of such systems requires software to be safety certified before it can be used, which is typically a lengthy process that hinders the dynamic update of software. Our approach is to enable the *continuous certification* of variants of safety critical software components; by running multiple variants (*certified* and *updated not-certified*) in parallel and by collecting enough evidence of agreement in between, we can prove the safety of the updated variant. Specifically, variants produce explications to justify their behavior. A supervising instance compares the behavior of variants and analyses their explications. The output of the updated variant is considered by the supervising instance if it is in agreement with the output of the certified variant. The absence of discrepancy between the two variants for long-enough period of running both variants in parallel, allows to dynamically certify it as a safe software variant. This use case focuses on the lane changing maneuver *i.e.*, the safe *lane* selection and the *speed*.

Architecture We run multiple *variants* of software components (*e.g.*, two: updated and certified) in parallel with a *supervisor* (see Fig. 1). The variants are connected to different *sensors* that capture the state of the freeway such as video or LIDAR signal sensors. The variants process the sensors' data and suggest the safe lanes to change to in addition to the evidence that justify the given selection. The supervisor is responsible for the selection of which variant output to forward to other system components *i.e.*, the execution units (*actuators*) that perform the maneuver. Variants categorize the set of available actions for each

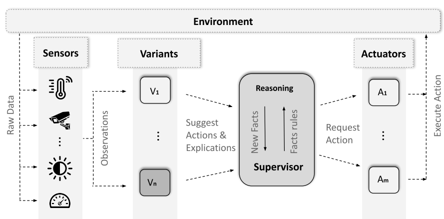

Fig. 1. The supervisor architecture.

time frame into *safe/unsafe* actions and provide *explications*. The supervisor collects the variants output and processes them to reason about (a) if enough evidence is provided by the variants to consider actions safe (b) find the actions that are considered safe by all variants.

Explications and rules Variants formulate their explications as *facts* using first-order predicates. The supervisor uses a set of logical *rules* formulated in BS(SLR) PP to reason about the suggestions and the explications (see List. 1.1).

```
1 ## Exclude actions per variant if safety disproved or declared unsafe.
2 SuggestionDisproven(xv, xa), VariantName(xv) -> ExcludedAction(xv, xa).
3 VariantName(xv), LaneNotSafe(xv, xl, xa)     -> ExcludedAction(xv, xa).
4 ## Exclude actions for all variants if declared unsafe by the certified
5 CertifiedVariant(xv1), UpdatedVariant(xv2), LaneNotSafe(xv1, xl, xa)
6   -> ExcludedAction(xv2,xa).
7
8 ## A safe action is disproven
9 SafeBehindDisproven(xv, xenl, xecl, xecs, xes, xa), LaneSafe(xv, xl, xa),
10   SuggestedAction(xv, xa)  -> SuggestionDisproven(xv, xa).
11 SafeFrontDisproven(xv, xenl, xecl, xecs, xes, xa),  LaneSafe(xv, xl, xa),
12   SuggestedAction(xv, xa)  -> SuggestionDisproven(xv, xa).
13
14 ## Unsafe left lane: speed decelerated and unsafe distance front
15 >(xh1, xfd), !=(xecl, xenl),  =(xh1,-(xes,1)) ||
16   LaneSafe(xv, xenl, adecelerateleft), EgoCar(xv, xecl, xecs, xes),
17   DistanceFront(xv, xenl, xofp, xfd, adecelerateleft),
18   SpeedFront(xv, xenl, xofp, xofs, adecelerateleft)
19   -> SafeFrontDisproven(xv, xenl, xecl, xecs, xes, adecelerateleft).
```

List. 1.1. The rules snippets for the lane changing use case in SPASS-SPL.

Variants explications: The `SuggestedAction` predicate encodes the actions suggested by the variants. `LaneSafe` and `LaneNotSafe` specify the lanes that are safe/unsafe to be used with the different actions. `DistanceFront` and `DistanceBehind` provide the explications related to the obstacle position, while their speeds are `SpeedFront` and `SpeedBehind`. `EgoCar` predicate reports the speed and the position of the ego vehicle.

Supervisor reasoning: To select a safe action, the supervisor must exclude all unsafe actions. The supervisor considers actions to be excluded per variant (`ExcludedAction`) if (a) `SuggestionDisproven`; the variant fails to prove that the suggested action is safe (line 2), or (b) the action is declared unsafe (line 3). The supervisor declares an action to be excluded cross all variants if the certified variant declares it unsafe (lines 5-6). To consider an action as `SuggestionDisproven`, the supervisor must check for each `LaneSafe` the existence of unsafe distances between the ego vehicle in the given lane and the other vehicles approaching either from behind (`SafeBehindDisproven`) or in front (`SafeFrontDisproven`). The rule `SafeFrontDisproven` (lines 15-19) checks in the left lane, if using the ego vehicle decelerated speed (`=(xh1,-(xes,1))`) the distance between the vehicles is not enough (`>(xh1, xfd)`). The supervisor checks `ExcludeAction` for all variants. If all actions are excluded, the supervisor uses an emergency action as no safe action exists. Otherwise, selects a safe action from the not-excluded actions suggested by the updated variant, if not found, by the certified.

2. Engine ECU The GM LSJ Ecotec engine[3] is a supercharged combustion engine that was almost exclusively deployed in the US, still some of those run also in Europe. The main sensor inputs of the LSJ ECU consist of an inlet air pressure and temperature sensor (in KPa and in degree Celsius), a speed sensor (in Rpm), a throttle pedal sensor, a throttle sensor, a coolant temperature sensor, oxygen sensors, a knock sensor, and its main actuators controlling the engine are ignition and injection timing, and throttle position. For the experiments conducted in this paper we have taken the routines of the LSJ ECU that compute ignition and injection timings out of inlet air pressure, inlet air temperature, and engine speed. For this part of the ECU this is a two stage process where firstly, basic ignition and injection timings are computed out of engine speed and inlet air pressure and secondly, those are adjusted with respect to inlet air temperature. The properties we prove are safety properties, e.g., certain injection timings are never generated and also invariants, e.g., the ECU computes actuator values for all possible input sensor data and they are unique. Clause 2, page 5, is an actual clause from the ECU case study computing the base ignition timing.

5 Implementation and Experiments

We have implemented the Datalog hammer into our BS(LRA) system SPASS-SPL and combined it with the Datalog reasoner Rulewerk. The resulting toolchain is the first implementation of a decision procedure for HBS(SLR) with positive conjectures.

SPASS-SPL is a new system for BS(LRA) based on some core libraries of the first-order theorem prover SPASS [37] and including the CDCL(LA) solver SPASS-SATT [9] for mixed linear arithmetic. Eventually, SPASS-SPL will include a family of reasoning techniques for BS(LRA) including SCL(T) [8], hierarchic superposition [2,5] and hammers to various logics. Currently, it comprises the Datalog hammer described in this paper and hierarchic UR-resolution [24] (Unit Resulting resolution) which is complete for pure HBS(LRA). The Datalog hammer can produce the clause format used in the Datalog system *Rulewerk* (described below), but also the SPASS first-order logic clause format that can then be translated into the first-order TPTP library [34] clause format. Moreover, it can be used as a translator from our own input language into the SMT-LIB 2.6 language [4] and the CHC competition format [32].

Note that our implementation of the Datalog hammer is of prototypical nature. It cannot handle positively grounded theory atoms beyond simple bounds, unless they are variable comparisons (i.e., $x \triangleleft y$ with $\triangleleft \in \{\leq, <, \neq, =, >, \geq\}$). Moreover, positive universal conjectures have to be flattened until they have the form $\Lambda \parallel P(\bar{x})$. On the other hand, we already added some improvements, e.g., we break/eliminate symmetries in the hammered conjecture and we exploit the theory atoms Λ in a universal conjecture $\Lambda \parallel P(\bar{x})$ so the hammered conjecture contains only groundings for $P(\bar{x})$ that satisfy Λ.

Rulewerk (formerly *VLog4j*) is a rule reasoning toolkit that consists of a Java API and an interactive shell [10]. Its current main reasoning back-end is the rule engine *VLog*

[3] https://en.wikipedia.org/wiki/GM_Ecotec_engine

[35], which supports Datalog and its extensions with stratified negation and existential quantifiers, respectively. VLog is an in-memory reasoner that is optimized for efficient use of resources, and has been shown to deliver highly competitive performance in benchmarks [36].

We have not specifically optimized VLog or Rulewerk for this work, but we have tried to select Datalog encodings that exploit the capabilities of these tools. The most notable impact was observed for the encoding of universal conjectures. A direct encoding of (grounded) universal claims in Datalog leads to rules with many (hundreds of thousands in our experiments) ground atoms as their precondition. Datalog reasoners (not just VLog) are not optimized for such large rules, but for large numbers of facts. An alternative encoding in plain Datalog would therefore specify the expected atoms as facts and use some mechanism to iterate over all of them to check for goal. To accomplish this iteration, the facts that require checking can be endowed with an additional identifier (given as a parameter), and an auxiliary binary successor relation can be used to specify the iteration order over the facts. This approach requires only few rules, but the number of rule applications is proportional to the number of expected facts.

In Rulewerk/VLog, we can encode this in a simpler way using negation. Universal conjectures require us to evaluate ground queries of the form $\mathit{entailed}(\bar{c}_1) \wedge \ldots \wedge \mathit{entailed}(\bar{c}_\ell)$. If we add facts $\mathit{expected}(\bar{c}_i)$ for the constant vectors $\bar{c}_1, \ldots, \bar{c}_\ell$, we can equivalently use a smaller (first-order) query $\forall \bar{x}.(\mathit{expected}(\bar{x}) \rightarrow \mathit{entailed}(\bar{x}))$, which in turn can be written as $\neg\big(\exists \bar{x}.(\mathit{expected}(\bar{x}) \wedge \neg\mathit{entailed}(\bar{x}))\big)$. This can be expressed in Datalog with negation, where *Goal* encodes that the query matches:

$$\mathit{expected}(\bar{x}) \wedge \neg\mathit{entailed}(\bar{x}) \rightarrow \mathit{missing} \tag{3}$$

$$\neg\mathit{missing} \rightarrow \mathit{Goal} \tag{4}$$

This use of negation is *stratified*, i.e., not entwined with recursion [1]. Note that stratified negation is a form of non-monotonic negation, so we can no longer read such rules as first-order formulae over which we compute entailments. Nevertheless, implementation is simple and stratified negation is a widely supported feature in Datalog engines, including Rulewerk. The encoding is particularly efficient since each of the rules (3) and (4) is evaluated only once.

Benchmark Experiments To test the efficiency of our toolchain, we ran benchmark experiments on the two real world HBS(SLR) PP supervisor verification conditions. The two supervisor use cases are described in Section 4. The names of the problems are formatted so the lane change assistant examples start with lc and the ECU examples start with ecu. The lc problems with existential conjectures test whether an action suggested by an updated variant is contradicted by a certified variant. The lc problems with universal conjectures test whether an emergency action has to be taken because we have to exclude all actions for all variants. The ecu problems with existential conjectures test safety properties, e.g., whether a computed actuator value is never outside of the allowed safety bounds. The ecu problems with universal conjectures test whether the ecu computes an actuator value for all possible input sensor data. Our benchmarks are prototypical for the complexity of HBS(SLR) reasoning in that they cover all abstract relationships between conjectures and HBS(SLR) clause sets. With respect to our two case studies we have many more examples showing respective characteristics. We would

Problem	Q	Status	X	Y	B	Size	t-time	h-time	p-time	r-time	vampire	spacer	z3	cvc4
lc_e1	∃	true	9	3	19	12/30	231	12	71	148	39	37	29	48
lc_e2	∃	false	9	3	17	13/27	199	10	70	119	35	70	timeout	timeout
lc_e3	∃	false	9	3	15	12/22	244	10	84	150	35	19	timeout	timeout
lc_e4	∃	true	9	3	21	12/35	223	10	75	138	40	28	28	56
lc_u1	∀	false	9	2	29	12/25	207	10	75	122	40	N/A	timeout	timeout
lc_u2	∀	false	9	2	26	12/25	218	10	77	131	44	N/A	timeout	timeout
lc_u3	∀	true	9	2	23	12/22	213	10	81	122	41	N/A	42	58
lc_u4	∀	false	9	2	32	12/33	232	12	84	136	46	N/A	timeout	timeout
ecu_e1	∃	false	10	6	311	27/649	1109	80	281	748	518	123	timeout	timeout
ecu_e2	∃	true	10	6	311	27/649	1092	83	270	739	546	120	2388	373
ecu_u1	∀	true	11	1	310	27/651	1084	82	275	727	94644	N/A	145226	339
ecu_u2	∀	false	11	1	310	27/651	1097	83	280	734	80744	N/A	timeout	timeout
ecu_u3	∀	true	9	2	433	27/1291	968	122	435	411	12014	N/A	209704	113
ecu_u4	∀	true	9	2	1609	26/20459	12381	2860	3204	6317	526544	N/A	167696	132
ecu_u5	∀	true	10	3	629	28/17789	22590	726	2086	19778	timeout	N/A	timeout	timeout
ecu_u6	∀	false	10	3	618	27/15667	11555	672	1743	9140	timeout	N/A	timeout	timeout

Fig. 2. Benchmark results and statistics

have liked to run benchmarks from other sources too, but we could not find any suitable HBS(SLR) problems in the SMT-LIB or CHC-COMP benchmarks.

We also tested the same benchmarks on two state-of-the-art satisfiability modulo theories (SMT) solvers, one state-of-the-art constrained horn clause (CHC) solver, and one first-order theorem prover: *cvc4-1.8* [3], with the best setting we found, which are the options `--multi-trigger-cache --full-saturate-quant`; *z3-4.8.10* [25] in its default settings; *spacer* [20], which is a CHC solver within z3 that is called if we select "Horn" as SMT input logic, and *vampire-4.5.1* [31] with settings `--memory_limit 8000 -p off`, i.e., with memory extended to 8GB and without proof output.

For the experiments, we used a Debian Linux server with 32 Intel Xeon Gold 6144 (3.5 GHz) processors and 754 GB RAM. Our toolchain employs no parallel computing, except for the java garbage collection. The tested SMT, CHC solvers, and first-order provers employ no parallel computing at all. Each tool got a time limit of 40 minutes for each problem.

The table in Fig. 2 lists for each benchmark problem: the name of the problem (Problem); the type of conjecture (Q), i.e., whether the conjecture is existential $\exists$ or universal $\forall$; the status of the conjecture (Status), i.e., true if the conjecture is a consequence and false otherwise; the maximum number of variables in any clause (X); the number of variables in the conjecture (Y); the number of test points/constants introduced by the Hammer (B); the size of the formula in kilobyte before and after the hammering (Size); the total time (in ms) needed by our toolchain to solve the problem (t-time); the time (in ms) spent on hammering the input formula (h-time); the time (in ms) spent on parsing the hammered formula by Rulewerk (p-time); the time (in ms) Rulewerk actually spent on reasoning (r-time). The remaining four columns list the time in ms needed by the other tools to solve the benchmark problems. An entry "N/A" means that the benchmark example cannot be expressed in the tools input format, e.g., it is not possible to encode a universal conjecture (or, to be more precise, its negation) in the CHC format. An entry "timeout" means that the tool could not solve the problem in the given time limit of 40 minutes. VLog is connected to SPASS-SPL via a file interface. Therefore, we show parsing time separately.

The experiments show that only our toolchain can solve all the problems in reasonable time. It is also the only solver that can decide in reasonable time whether a universal conjecture is not a consequence. This is not surprising because to our knowledge our toolchain is the only theorem prover that implements a decision procedure for HBS(SLR). On the other types of problems, our toolchain might not be the fastest, but it solves all of the problems in the range of seconds and with comparable times to the best tool for the problem. For problems with existential conjectures, the CHC solver spacer is the best, but as a trade-off it is unable to handle universal conjectures at all. The instantiation techniques employed by cvc4 are good for proving some universal conjectures, but both SMT solvers seem to be unable to disprove conjectures. Vampire performed best on the hammered problems among all first-order theorem provers we tested, including iProver [21], E [33], and SPASS [37]. We tested all provers in default theorem proving mode, but adjusted the memory limit of Vampire, because it ran out of memory on ecu_u4 with the default setting. The experiments with the first-order provers showed that our hammer also works reasonably well for them, e.g., they can all solve all lane change problems in less than a second, but they are simply not specialized for the HBS fragment.

6 Conclusion

We have presented several new techniques that allow us to translate BS(SLR) PP clause sets with both universally and existentially quantified conjectures into logics for which efficient decision procedures exist. The first set of translations returns a finite abstraction for our clause set and conjecture, i.e., an equisatisfiable ground BS(LRA) clause set over a finite set of test points/constants that can be solved in theory by any SMT solver for linear arithmetic. The abstraction grows exponentially in the maximum number of variables in any input clause. Realistic supervisor examples have clauses with 10 or more variables and the basis of the growth exponent is also typically large, e.g., in our examples it ranges from 15 to 1500, so this leads immediately to very large clause sets. An exponential growth in grounding is also unavoidable, because the abstraction reduces a NEXPTIME-hard problem to an NP-complete problem (ground BS, i.e., SAT). As an alternative, we also present a Datalog hammer, i.e., a translation to an equisatisfiable HBS clause set without any theory constraints. The hammer is restricted to the Horn case, i.e., HBS(SLR) PP clauses, and the conjectures to positive universal/existential conjectures. Its advantage is that the formula grows only exponentially in the number of variables in the universal conjecture, which is typically much smaller than the maximum number of variables in any input clause. For instance, in our examples it ranges only from zero to three.

We have implemented the Datalog hammer into our BS(LRA) system SPASS-SPL and combined it with the Datalog reasoner Rulewerk. The resulting toolchain is an effective way of deciding verification conditions for supervisors if the supervisors can be modeled as HBS(SLR) clause sets and the conditions as positive BS(SLR) conjectures. To confirm this, we have presented two use cases for real-world supervisors: (i) the verification of supervisor code for the electrical control unit of a super-charged combustion engine and (ii) the continuous certification of lane assistants. Our experiments show that for these use cases our toolchain is overall superior to existing solvers. Over existential conjectures, it is comparable with existing solvers (e.g., CHC solvers). Moreover, our toolchain is the only solver we are aware of that can proof and disproof universal conjectures for our use cases.

For future work, we want to further develop our toolchain in several directions. First, we want SPASS-SPL to produce explications that prove that its translations are correct. Second, we plan to exploit specialized Datalog expressions and techniques (e.g., aggregation and stratified negation) to increase the efficiency of our toolchain and to lift some restrictions from our input formulas. Third, we want to optimize the selection of test points. For instance, we could partition all predicate argument positions into independent sets, i.e., two argument positions are dependent if they are assigned the same variable in the same rule. For each of these partitions, we should be able to create an independent and much smaller set of test points because we only have to consider theory constraints connected to the argument positions in the respective partition. In many cases, this would lead to much smaller sets of test points and therefore also to much smaller hammered and finitely abstracted formulas.

Acknowledgments: This work was funded by DFG grant 389792660 as part of TRR 248. We thank Pascal Fontaine, Alberto Griggio, Andrew Reynolds, and Stephan Schulz for discussing various aspects of this paper.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley (1994)
2. Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierarchic first-order theories. Applicable Algebra in Engineering, Communication and Computing, AAECC **5**(3/4), 193–212 (1994)
3. Barrett, C., Conway, C., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: CAV, LNCS, vol. 6806 (2011)
4. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Tech. rep., Department of Computer Science, The University of Iowa (2017), available at `www.SMT-LIB.org`
5. Baumgartner, P., Waldmann, U.: Hierarchic superposition revisited. In: Lutz, C., Sattler, U., Tinelli, C., Turhan, A., Wolter, F. (eds.) Description Logic, Theory Combination, and All That - Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 11560, pp. 15–56. Springer (2019)
6. Bjørner, N., Gurfinkel, A., McMillan, K.L., Rybalchenko, A.: Horn clause solvers for program verification. In: Beklemishev, L.D., Blass, A., Dershowitz, N., Finkbeiner, B., Schulte, W. (eds.) Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday. Lecture Notes in Computer Science, vol. 9300, pp. 24–51. Springer (2015)
7. Bromberger, M., Fiori, A., Weidenbach, C.: Scl with theory constraints (2020)
8. Bromberger, M., Fiori, A., Weidenbach, C.: Deciding the bernays-schoenfinkel fragment over bounded difference constraints by simple clause learning over theories. In: Henglein, F., Shoham, S., Vizel, Y. (eds.) Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12597, pp. 511–533. Springer (2021)
9. Bromberger, M., Fleury, M., Schwarz, S., Weidenbach, C.: SPASS-SATT - A CDCL(LA) solver. In: Fontaine, P. (ed.) Automated Deduction - CADE 27 - 27th International Conference

on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11716, pp. 111–122. Springer (2019)
10. Carral, D., Dragoste, I., González, L., Jacobs, C., Krötzsch, M., Urbani, J.: VLog: A rule engine for knowledge graphs. In: Ghidini et al., C. (ed.) Proc. 18th Int. Semantic Web Conf. (ISWC'19, Part II). LNCS, vol. 11779, pp. 19–35. Springer (2019)
11. Cimatti, A., Griggio, A., Redondi, G.: Universal invariant checking of parametric systems with quantifier-free SMT reasoning. In: Proc. CADE-28 (2021), to appear
12. Downey, P.J.: Undecidability of presburger arithmetic with a single monadic predicate letter. Tech. rep., Center for Research in Computer Technology, Harvard University (1972)
13. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive datalog. ACM Trans. Database Syst. **22**(3), 364–418 (1997)
14. Faqeh, R., Fetzer, C., Hermanns, H., Hoffmann, J., Klauck, M., Köhl, M.A., Steinmetz, M., Weidenbach, C.: Towards dynamic dependable systems through evidence-based continuous certification. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12477, pp. 416–439. Springer (2020)
15. Ge, Y., de Moura, L.M.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5643, pp. 306–320. Springer (2009)
16. Grebenshchikov, S., Lopes, N.P., Popeea, C., Rybalchenko, A.: Synthesizing software verifiers from proof rules. In: Vitek, J., Lin, H., Tip, F. (eds.) ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12, Beijing, China - June 11 - 16, 2012. pp. 405–416. ACM (2012)
17. Hillenbrand, T., Weidenbach, C.: Superposition for bounded domains. In: Bonacina, M.P., Stickel, M. (eds.) McCune Festschrift. LNCS, vol. 7788, pp. 68–100. Springer (2013)
18. Horbach, M., Voigt, M., Weidenbach, C.: On the combination of the bernays-schönfinkel-ramsey fragment with simple linear integer arithmetic. In: de Moura, L. (ed.) Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10395, pp. 77–94. Springer (2017)
19. Horbach, M., Voigt, M., Weidenbach, C.: The universal fragment of presburger arithmetic with unary uninterpreted predicates is undecidable. CoRR **abs/1703.01212** (2017)
20. Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. In: CAV. Lecture Notes in Computer Science, vol. 8559, pp. 17–34. Springer (2014)
21. Korovin, K.: iprover - an instantiation-based theorem prover for first-order logic (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5195, pp. 292–298. Springer (2008)
22. Lewis, H.R.: Complexity results for classes of quantificational formulas. Journal of Compututer and System Sciences **21**(3), 317–353 (1980)
23. Loos, R., Weispfenning, V.: Applying linear quantifier elimination. The Computer Journal **36**(5), 450–462 (1993)
24. McCharen, J., Overbeek, R., Wos, L.: Complexity and related enhancements for automated theorem-proving programs. Computers and Mathematics with Applications **2**, 1–16 (1976)
25. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 4963 (2008)
26. de Moura, L.M., Bjørner, N.: Satisfiability modulo theories: introduction and applications. Communications of the ACM **54**(9), 69–77 (2011)

27. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). Journal of the ACM **53**, 937–977 (November 2006)
28. Nonnengart, A., Weidenbach, C.: Computing small clause normal forms. In: Handbook of Automated Reasoning, pp. 335–367. Elsevier and MIT Press (2001)
29. Plaisted, D.A.: Complete problems in the first-order predicate calculus. Journal of Computer and System Sciences **29**, 8–35 (1984)
30. Reynolds, A., Barbosa, H., Fontaine, P.: Revisiting enumerative instantiation. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10806, pp. 112–131. Springer (2018)
31. Riazanov, A., Voronkov, A.: The design and implementation of vampire. AI Communications **15**(2-3), 91–110 (2002)
32. Rümmer, P.: Competition report: CHC-COMP-20. In: Fribourg, L., Heizmann, M. (eds.) Proceedings 8th International Workshop on Verification and Program Transformation and 7th Workshop on Horn Clauses for Verification and Synthesis, VPT/HCVS@ETAPS 2020, Dublin, Ireland, 25-26th April 2020. EPTCS, vol. 320, pp. 197–219 (2020)
33. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) Proc. of the 27th CADE, Natal, Brasil. pp. 495–507. No. 11716 in LNAI, Springer (2019)
34. Sutcliffe, G.: The TPTP problem library and associated infrastructure - from CNF to th0, TPTP v6.4.0. J. Autom. Reason. **59**(4), 483–502 (2017)
35. Urbani, J., Jacobs, C., Krötzsch, M.: Column-oriented Datalog materialization for large knowledge graphs. In: Schuurmans, D., Wellman, M.P. (eds.) Proc. 30th AAAI Conf. on Artificial Intelligence (AAAI'16). pp. 258–264. AAAI Press (2016)
36. Urbani, J., Krötzsch, M., Jacobs, C.J.H., Dragoste, I., Carral, D.: Efficient model construction for Horn logic with VLog: System description. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) Proc. 9th Int. Joint Conf. on Automated Reasoning (IJCAR'18). LNCS, vol. 10900, pp. 680–688. Springer (2018)
37. Weidenbach, C., Dimova, D., Fietzke, A., Suda, M., Wischnewski, P.: Spass version 3.5. In: Schmidt, R.A. (ed.) 22nd International Conference on Automated Deduction (CADE-22). Lecture Notes in Artificial Intelligence, vol. 5663, pp. 140–145. Springer, Montreal, Canada (August 2009)