Symbolic Model Construction for Saturated Constrained Horn Clauses

Martin Bromberger[1], Lorenz Leutgeb[1,2](✉), and Christoph Weidenbach[1]

[1] Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
{mbromber,lorenz,weidenb}@mpi-inf.mpg.de

[2] Graduate School of Computer Science, Saarland Informatics Campus, Saarbrücken, Germany

Abstract. Clause sets saturated by hierarchic ordered resolution do not offer a model representation that can be effectively queried, in general. They only offer the guarantee of the existence of a model. We present an effective symbolic model construction for saturated constrained Horn clauses. Constraints are in linear arithmetic, the first-order part is restricted to a function-free language. The model is constructed in finite time, and non-ground clauses can be effectively evaluated with respect to the model. Furthermore, we prove that our model construction produces the least model.

Keywords: Bernays-Schönfinkel Fragment · Linear Arithmetic · Horn Clauses · Superposition · Model Construction

1 Introduction

Constrained Horn Clauses (CHCs) combine logical formulas with constraints over various domains, e.g. linear real arithmetic, linear integer arithmetic, equalities of uninterpreted functions [15]. This formalism has gained widespread attention in recent years due to its applications in a variety of fields, including program analysis and verification: safety, liveness, and termination [17,38], complexity and resource analysis [33], intermediate representation [22], and software testing [35]. Technical controls, so called *Supervisors*, like an electronic engine control unit, or a lane change assistant in a car [8,9] can be modelled, run, and proven safe. Moreover, there exist many different approaches for reasoning in CHCs and associated first-order logic fragments extended with theories [2,5,7,10,15,23–25,28,29,34,37]. Thus, CHCs are a powerful tool for reasoning about complex systems that involve logical constraints, and they have been used to solve a wide range of problems.

A failed proof attempt of some conjecture or undesired run points to a bug. In this case investigation of the cause of the unexpected result or behavior is crucial. Building a model of the situation that can then be effectively queried is an important means towards a repair. However, some algorithms for CHCs, e.g. hierarchic superposition, which boils down to hierarchic ordered resolution in the context of CHCs, do not return a model that can be effectively queried if a proof attempt fails, in general. If so, queries are still restricted to ground clauses [4].

The contribution of our paper can be seen as an extension for these saturation based algorithms that produces models and not just saturated clause sets. In fact, we show how to build symbolic models out of any saturated CHC clause set over linear arithmetic. This fragment is equivalent to Horn clause sets of linear arithmetic combined with the Bernays-Schönfinkel fragment. Recall that although satisfiability in this fragment is undecidable [16,26], in general, for a finitely saturated set we can construct such a representation in finite time.

Our models fulfill all important properties postulated in the literature for automated model building in first-order logic [13,20]. First, they can be *effectively constructed*, i.e., each model is represented by one linear arithmetic formula of finite size for each of its predicates and it can be constructed in finite time. Second, they are *unique*, i.e., the model representation specifies exactly one interpretation; in our case the least model. Third, they can be *effectively queried*, i.e., we provide decision procedures that evaluate whether an atom, clause, or formula is entailed/satisfied by the model. Fourth, it is possible to *test* the *equivalence* of two models. The approach we present does not exploit features of linear arithmetic beyond equality, the existence of a well-founded order for the theories' universe, and decidability of the theory. The results may therefore be adapted to other constraint domains. Model representation that can be effectively constructed and queried like ours are also called *effective model representations*. Moreover, our method is the first effective model construction approach for ordered resolution (or its extension to superposition) that is based on saturation, goes beyond ground clauses, and includes theory constraints. In the future, we plan to use this approach as the basis for a more general model construction approach that also works on more expressive fragments of first-order logic modulo theories.

Our model construction is inspired by the model construction operator used in the proof for refutational completeness of hierarchic superposition [3,6,30]. The main difference is that the model construction operator from the refutational completeness proof is restricted to ground clauses and executed on the potentially infinite ground instances of the saturated clause set (in addition to an infinite axiomatization of the background theory as ground clauses). As a result, the model construction operator from the refutational completeness proof cannot effectively construct a model because iterating over a potentially infinite set means it may diverge. Moreover, in contrast to our model construction, the original model operator cannot effectively evaluate non-ground atoms, clauses, or formulas. It is, however, sufficient, to show the existence of a model if the clause set is saturated and does not contain the empty clause [3,6,30]. In our version of the model construction operator, we managed to lift the restriction to ground clause sets by restricting the input logic to the Horn Bernays-Schönfinkel fragment instead of full first-order logic. This enables us to define a strict propagation/production order for our non-ground clauses instead of just for ground clauses. As a result, we can construct the model one clause at a time.

The paper is organized as follows. In Sect. 2 we clarify notation and preliminaries. The main contribution is presented in Sect. 3. At the end of this section, we also explain how our models satisfy the postulates (see [13, Section 5.1, p. 234]) by Fermüller and Leitsch for automated model building. We conclude in Sect. 4. Proofs were elided in favor of explanations and examples. An extended version, which includes proofs, can be found at [12].

2 Preliminaries and Notation

We briefly recall the basic logical formalisms and notations we build upon [9]. Our starting point is a standard first-order language with *variables* (denoted x, y, z), *predicates* (denoted P, Q) of some fixed *arity*, and *terms* (denoted t, s). An *atom* (denoted A) is an expression $P(t_1, \ldots, t_n)$ for a predicate P of arity $n = \text{arity}(P)$. When the terms $t_1, \ldots, t_n$ in $P(t_1, \ldots, t_n)$ are not relevant in some context, we also write $P(*)$. A *positive literal* is an atom A and a *negative literal* is a negated atom $\neg A$. We define $\text{comp}(A) = \neg A$, $\text{comp}(\neg A) = A$, $|A| = A$ and $|\neg A| = A$. Literals are usually denoted L, K. We sometimes write literals as $[\neg]P(*)$, meaning that the sign of the literal is arbitrary, often followed by a case distinction. Formulas are defined in the usual way using quantifiers $\forall$, $\exists$ and the boolean connectives (in order of decreasing binding strength) $\neg$, $\vee$, $\wedge$, $\rightarrow$, and $\leftrightarrow$. The logic we consider does not feature a first-order equality predicate.

A *clause* (denoted C, D) is a universally closed disjunction of literals $A_1 \vee \cdots \vee A_n \vee \neg B_1 \vee \cdots \vee \neg B_m$. We may equivalently write $B_1 \wedge \cdots \wedge B_m \rightarrow A_1 \vee \cdots \vee A_n$. A clause is *Horn* if it contains at most one positive literal, i.e. $n \leq 1$. In Sect. 3, all clauses considered are Horn clauses. If Y is a term, formula, or a set thereof, $\text{vars}(Y)$ denotes the set of all variables in Y, and Y is *ground* if $\text{vars}(Y) = \emptyset$. Analogously, $\Pi(Y)$ is the set of predicate symbols occurring in Y.

The *Bernays-Schönfinkel Clause Fragment* (BS) in first-order logic consists of first-order clauses where all terms are either variables or constants. The *Horn Bernays-Schönfinkel Clause Fragment* (HBS) is further restricted to Horn clauses.

A *substitution* σ is a function from variables to terms with a finite domain and codomain. We denote substitutions by σ, τ. The application of substitutions is often written postfix, as in $x\sigma$, and is homomorphically extended to terms, atoms, literals, clauses, and quantifier-free formulas. A substitution is *ground* if its codomain is ground. Let Y denote some term, literal, clause, or clause set. A substitution 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 $\text{gnd}(Y)$ the set of all ground instances of Y. The *most general unifier* $\text{mgu}(Z_1, Z_2)$ 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.

2.1 Horn Bernays-Schönfinkel with Linear Arithmetic

The class HBS(LRA) is the extension of the Horn Bernays-Schönfinkel fragment with linear real arithmetic (LRA). Analogously, the classes HBS(LQA) and

HBS(LIA) are the extensions of the Horn Bernays-Schönfinkel fragment with linear rational arithmetic (LQA) and linear integer arithmetic (LIA), respectively. The only difference between the three classes are the sort LA their variables and terms range over and the universe $\mathcal{U}$ over which their interpretations range. As the names already imply LA = LRA and $\mathcal{U} = \mathbb{R}$ for HBS(LRA), LA = LQA and $\mathcal{U} = \mathbb{Q}$ for HBS(LQA), and LA = LIA and $\mathcal{U} = \mathbb{Z}$ for HBS(LIA). The results presented in this paper hold for all three classes and by HBS(LA) we denote that we are talking about an arbitrary one of them.

Linear arithmetic terms are constructed from a set $\mathcal{X}$ of *variables*, the set of constants $c \in \mathbb{Q}$ (if in HBS(LRA) or HBS(LQA)) or $c \in \mathbb{Z}$ (if in HBS(LIA)), and binary function symbols $+$ and $-$ (written infix). Additionally, we allow multiplication $\cdot$ if one of the factors is a constant. Multiplication only serves us as syntactic sugar to abbreviate other arithmetic terms, e.g., $x + x + x$ is abbreviated to $3 \cdot x$. Atoms in HBS(LA) are either *first-order atoms* (e.g., $P(13, x)$) or *(linear) arithmetic atoms* (e.g., $x < 42$). Arithmetic atoms are denoted by λ and may use the predicates $\leq, <, \approx, \not\approx, >, \geq$, which are written infix and have the expected fixed interpretation. We use $\approx$ instead of $=$ to avoid confusion between equality in LA and equality on the meta level. While we do not permit quantifiers in the syntax of clauses, the notion of symbolic interpretations that we will develop does require this, denoted as usual. By $\text{atoms}(Y)/\text{quants}(Y)$ we denote the linear arithmetic atoms/quantifiers in a formula or set of formulas Y. *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., $\neg(x \geq 42)$ is equivalent to $x < 42$.

HBS(LA) clauses are defined as for HBS but using HBS(LA) 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 LA atoms called the *constraint* of the clause. A clause of the form $\Lambda \parallel C$ is therefore also called a *constrained clause.* Since the interpretation of linear arithmetic relations is fixed, we set $\Pi(\Lambda \parallel C) \coloneqq \Pi(C)$.

The fragment we consider in Sect. 3 is restricted even further to *abstracted* clauses: For any clause $\Lambda \parallel C$, all terms in C must be variables. Put differently, we disallow any arithmetic function symbols, including numerical constants, in C. Variable abstraction, e.g. rewriting $x \geq 3 \parallel P(x, 1)$ to $x \geq 3, y \approx 1 \parallel P(x, y)$, is always possible. Hence, the restriction to abstracted clauses is not a theoretical limitation, but allows us to formulate our model construction operator in a more concise way. We assume abstracted clauses for theory development, but we prefer non-abstracted clauses in examples for readability, e.g., a unit clause $P(3, 5)$ is considered in the development of the theory as the clause $x \approx 3, y \approx 5 \parallel P(x, y)$.

In contrast to other works, e.g. [11], we do not permit first-order constants, and consequently also no variables that range over the induced Herbrand universe. All variables are arithmetic in the sense that they are interpreted by $\mathcal{U}$. Since we only allow equalities in the arithmetic constraint, it is possible to simulate variables over first-order constants, by e.g. numbering them, i.e. defining a bijection between $\mathbb{N}$ and constant symbols. So this again not a theoretical limitation.

The semantics of $\Lambda \,\|\, C$ is as follows:

$$\Lambda \,\|\, C \quad \text{iff} \quad \Big(\bigwedge_{\lambda\in\Lambda} \lambda\Big) \rightarrow C \quad \text{iff} \quad \Big(\bigvee_{\lambda\in\Lambda} \neg\lambda\Big) \vee C$$

For example, the clause $x > 1 \vee y \not\approx 5 \vee \neg Q(x) \vee R(x, y)$ is also written $x \leq 1, y \approx 5 \,\|\, \neg Q(x) \vee R(x, y)$. The negation $\neg(\Lambda \,\|\, C)$ of a constrained clause $\Lambda \,\|\, C$ where $C = A_1 \vee \cdots \vee A_n \vee \neg B_1 \vee \cdots \vee \neg B_m$ is thus equivalent to $(\bigwedge_{\lambda\in\Lambda} \lambda) \wedge \neg A_1 \wedge \cdots \wedge \neg A_n \wedge B_1 \wedge \cdots \wedge B_m$. Note that since the neutral element of conjunction is $\top$, an empty constraint is thus valid, i.e. equivalent to true. In analogy to the empty clause in settings without constraints, we write $\Box$ to mean any and all clauses $\Lambda \,\|\, \bot$ where Λ is satisfiable, which are all unsatisfiable.

An *assignment* for a constraint Λ is a substitution (denoted β) that maps all variables in $\text{vars}(\Lambda)$ to values in $\mathcal{U}$. An assignment is a *solution* for a constraint Λ if all atoms $\lambda \in (\Lambda\beta)$ evaluate to true. A constraint Λ is *satisfiable* if there exists a solution for Λ. Otherwise it is *unsatisfiable*.

We assume *pure* input clause sets because otherwise satisfiability is undecidable for impure HBS(LA) [21]. This means the only constants of our sort LA are concrete rational numbers. Irrational numbers are not allowed by the standard definition of the theory. Fractions are not allowed if LA = LIA. Satisfiability of pure HBS(LA) clause sets is semi-decidable, e.g., using *hierarchic superposition* [3] or *SCL(T)* [10]. Note that pure HBS(LA) clauses correspond to *constrained Horn clauses (CHCs)* with LA as background theory.

All arithmetic predicates and functions are interpreted in the usual way denoted by the interpretation $\mathcal{A}^{\text{LA}}$. An interpretation of HBS(LA) coincides with $\mathcal{A}^{\text{LA}}$ on arithmetic predicates and functions, and freely interprets non-arithmetic predicates. For pure clause sets this is well-defined [3]. Logical satisfaction and entailment is defined as usual, and uses similar notation as for HBS.

Example 1. The clause $y \geq 5, x' \approx x + 1 \,\|\, S_0(x, y) \rightarrow S_1(x', 0)$ is part of a timed automaton with two clocks x and y modeled in HBS(LA). It represents a transition from state S_0 to state S_1 that can be traversed only if clock y is at least 5 and that resets y to 0 and increases x by 1.

2.2 Ordering Literals and Clauses

In order to define redundancy for constrained clauses, we need an *order*: Let $\prec_\Pi$ be a total, well-founded, strict ordering on predicate symbols and let $\prec_\mathcal{U}$ be a total, well-founded, strict ordering on the universe $\mathcal{U}$. (Note that $\prec$ cannot be the standard ordering $<$ because it is not well-founded for $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$. In the case of $\mathbb{R}$, the existence of such an order is even dependent on whether we assume the axiom of choice [18].) We extend these orders step by step. First, to atoms, i.e., $P(\vec{a}) \prec Q(\vec{b})$ if $P \prec_\Pi Q$ or $P = Q$, $\vec{a}, \vec{b} \in \mathcal{U}^{|\vec{a}|}$, and $\vec{a} \prec_{\text{lex}} \vec{b}$, where $\prec_{\text{lex}}$ is the lexicographic extension of $\prec_\mathcal{U}$. Next, we extend the order to literals with a strict precedence on the predicate and the polarity, i.e.,

$$P(\vec{t}) \prec \neg P(\vec{s}) \prec Q(\vec{u}) \qquad \text{if } P \prec Q$$

independent of the arguments of the literals. Then, take the multiset extension to order clauses. To handle constrained clauses extend the relation such that constraint literals (in our case arithmetic literals) are always smaller than first-order literals. We conflate the notation of all extensions into the symbol $\prec$ and define $\preceq$ as the reflexive closure of $\prec$. Note that $\prec$ is only total for ground atoms/literals/clauses, which is sufficient for a hierarchic superposition order [6].

Definition 2 ($\prec$-maximal Literal). *A literal L is called* $\prec$-maximal *in a clause C if there exists a grounding substitution σ for C, such that there is no different $L' \in C$ for which $L\sigma \prec L'\sigma$. The literal L is called* strictly $\prec$-maximal *if there is no different $L' \in C$ for which $L\sigma \preceq L'\sigma$.*

Proposition 3. *If $\prec$ is a predicate-based ordering, C is a Horn clause, C has a positive literal L, and L is $\prec$-maximal in C, then L is strictly $\prec$-maximal in C.*

Definition 4 ($\prec$-maximal Predicate in Clause). *A predicate symbol P is called* (strictly) $\prec$-maximal *in a clause C if there is a literal $[\neg]P(*) \in C$ that is (strictly) $\prec$-maximal in C.*

Definition 5. *Let N be a set of clauses, $\prec$ a clause ordering, C a clause, and P a predicate symbol. Then $N^{\prec C} := \{C' \in N \mid C' \prec C\}$ and $N^{\preceq P} := \{C \in N \mid Q$ is $\prec$-maximal in C and $Q \preceq P\}$.*

2.3 Hierarchic Superposition, Redundancy and Saturation

For pure HBS(LA) most rules of the (hierarchic) superposition calculus become obsolete or can be simplified. In fact, in the HBS(LA) case (hierarchic) superposition boils down to (hierarchic) ordered resolution. For a full definition of (hierarchic) superposition calculus in the context of linear arithmetic, consider SUP(LA) [1]. Here, we will only define its simplified version in the form of the hierarchic resolution rule.

Definition 6 (Hierarchic $\prec$-Resolution). *Let $\prec$ be an order on literals and $\Lambda_1 \parallel L_1 \vee C_1$, $\Lambda_2 \parallel L_2 \vee C_2$ be constrained clauses. The inference rule of hierarchic $\prec$-resolution is:*

$$\frac{\Lambda_1 \parallel L_1 \vee C_1 \qquad \Lambda_2 \parallel L_2 \vee C_2 \qquad \sigma = \mathrm{mgu}(L_1, \mathrm{comp}(L_2))}{(\Lambda_1, \Lambda_2 \parallel C_1 \vee C_2)\sigma}$$

where L_1 is $\prec$-maximal in C_1 and L_2 is $\prec$-maximal in C_2.

Note that in the resolution rule we do not enforce explicitly that the positive literal is strictly maximal. This is possible because in the Horn case any positive literal is strictly maximal if it is maximal in the clause.

For saturation, we need a termination condition that defines when the calculus under consideration cannot make any further progress. In the case of superposition, this notion is that any new inferences are *redundant*.

Definition 7 (Clause Redundancy). *A ground clause $\Lambda \,\|\, C \in N$ is* redundant *with respect to a set N of ground clauses and order $\prec$ if $N^{\prec \Lambda \,\|\, C} \models \Lambda \,\|\, C$. A potentially non-ground clause $\Lambda \,\|\, C \in N$ is* redundant *with respect to a potentially non-ground clause set N and order $\prec$ if for all $\Lambda' \,\|\, C' \in \mathrm{gnd}(\Lambda \,\|\, C)$ the clause $\Lambda' \,\|\, C'$ is redundant with respect to $\mathrm{gnd}(N)$.*

If a clause $\Lambda \,\|\, C \in N$ is redundant with respect to a clause set N, then it can be removed from N without changing its semantics. If $\Lambda \,\|\, C$ is newly inferred, then we also call it redundant if $\Lambda \,\|\, C$ is already part of N. The same cannot be said for clauses in N or all clauses in N would be redundant. Determining clause redundancy is an undecidable problem [10,40]. However, there are special cases of redundant clauses that can be easily checked, e.g., tautologies and subsumed clauses. Redundancy also means that $\mathcal{I} \models N^{\prec \Lambda \,\|\, C}$ implies $\mathcal{I} \models \Lambda \,\|\, C$ if $\Lambda \,\|\, C$ is redundant w.r.t. N. We will exploit this fact in the model construction.

Definition 8 (Saturation). *A set of clauses N is* saturated up to redundancy *with respect to some set of inference rules, if application of any rules to clauses in N yields a clause that is redundant with respect to N or is contained in N.*

2.4 Interpretations

In our context, models are interpretations that satisfy (sets of) clauses. The standard notion of an interpretation is fairly opaque and interprets a predicate P as the potentially infinite set of ground arguments that satisfy P.

Definition 9 (Interpretation). *Let P be a predicate symbol with* $\mathrm{arity}(P) = n$*. Then, $P^{\mathcal{I}}$ denotes the subset of $\mathcal{U}^n$ for which the* interpretation *$\mathcal{I}$ maps the predicate symbol P to* true*.*

Since our model construction approach manipulates interpretations directly, we need a notion of interpretations that always has a finite representation and for which it is possible to decide (in finite time) whether a clause is satisfied by the interpretation. Therefore, we rely on the notion of symbolic interpretations:

Definition 10 (Symbolic Interpretation). *Let $x_1, x_2, \ldots$ be an infinite sequence of distinct variables, i.e. $x_i \neq x_j$ for all $1 \leq i < j$. (We assume the same sequence for all symbolic interpretations in order to prevent conflicts when we later combine multiple symbolic interpretations into one.) A* symbolic interpretation *$\mathcal{S}$ is a function that maps every predicate symbol P with* $\mathrm{arity}(P) = n$ *to a formula denoted $P^{\mathcal{S}}(\vec{x})$ of finite size, constructed using the usual boolean connectives over* LA *atoms, where the only free variables appear in $\vec{x} = (x_1, \ldots, x_n)$. The interpretation $\mathcal{I}_{\mathcal{S}}$ corresponding to $\mathcal{S}$ is defined by $P^{\mathcal{I}_{\mathcal{S}}} = \{(\vec{x})\beta \mid \beta \models P^{\mathcal{S}}(\vec{x})\}$ and maps the predicate symbol P to* true *for the subset of $\mathcal{U}^n$ which corresponds to the solutions of $P^{\mathcal{S}}(\vec{x})$.*

Example 11. Let N be a clause set consisting of the clauses $0 \leq x \leq 2, 0 \leq y \leq 2 \| P(x, y)$ and $x_Q \geq x_P + 1, y_Q \geq y_P + 1 \| \neg P(x_P, y_P) \vee Q(x_Q, y_Q)$. An example of a symbolic interpretation $\mathcal{S}$ that satisfies N, would be the function that maps P to $P^{\mathcal{S}}(x_1, x_2) = 0 \leq x_1 \leq 2 \wedge 0 \leq x_2 \leq 2$ and $Q^{\mathcal{S}}(x_1, x_2) = 1 \leq x_1 \wedge 1 \leq x_2$. It corresponds to the interpretation $\mathcal{I}_{\mathcal{S}}$ where $P^{\mathcal{I}_{\mathcal{S}}} = \{(a_1, a_2) \in \mathcal{U} \mid 0 \leq a_1 \leq 2 \wedge 0 \leq a_2 \leq 2\}$ and $Q^{\mathcal{I}_{\mathcal{S}}} = \{(a_1, a_2) \in \mathcal{U} \mid 1 \leq a_1 \wedge 1 \leq a_2\}$.

The notion of symbolic interpretations is closely related to *$\mathcal{A}$-definable models* [7, Definition 7] and *constrained atomic representations* [13, Definition 5.1, pp. 236–237]. Each symbolic interpretation $\mathcal{S}(\vec{x})$ is equivalent to a constrained atomic representation that consists of one constraint atom $[[P(\vec{x}) : P^{\mathcal{S}}(\vec{x})]]$ (written in the notation from [13]) for every predicate P. Note that in this context the constraint is not just a quantifier-free conjunction of linear arithmetic atoms, but a linear arithmetic formula potentially containing quantifiers (although those can be eliminated with quantifier elimination techniques).

Due to the fact that each symbolic interpretation consists of a finite set of formulas of finite size, symbolic interpretations can be considered as finite representations. In contrast, the standard representation of an interpretation as a potentially infinite set of ground atoms is not a finite representation. However, this also means that there are some interpretations for which no corresponding symbolic interpretation exists, for instance the set of prime numbers is a satisfying interpretation for $y \approx 2 \,\|\, P(y)$, but not expressible as a symbolic interpretation (in LA). As we will later see, at least any saturated set of HBS(LA) clauses either is unsatisfiable or has a symbolic interpretation that satisfies it (Theorem 29).

The *top interpretation*, denoted $\mathcal{I}_\top$, is defined as $P^{\mathcal{I}_\top} := \mathcal{U}^n$ for all predicate symbols P with arity$(P) = n$ and corresponds to the *top symbolic interpretation*, denoted $\mathcal{S}_\top$, defined as $P^{\mathcal{S}_\top} := \top$ for all predicate symbols P. The *bottom interpretation* (or *empty interpretation*), denoted $\mathcal{I}_\bot$, and the *bottom symbolic interpretation* (or *empty symbolic interpretation*), denoted $\mathcal{S}_\bot$, are defined analogously. The interpretation of P under $\mathcal{I} \cup \mathcal{J}$ is defined as $P^{\mathcal{I} \cup \mathcal{J}} := P^{\mathcal{I}} \cup P^{\mathcal{J}}$ for every predicate P. In the symbolic case, $\mathcal{S} \cup \mathcal{R}$ is defined as $P^{\mathcal{S} \cup \mathcal{R}}(\vec{x}) := P^{\mathcal{S}}(\vec{x}) \vee P^{\mathcal{R}}(\vec{x})$ for every predicate P. We write $\mathcal{I} \subseteq \mathcal{J}$ or $\mathcal{I}$ is *included in* $\mathcal{J}$ (resp. $\mathcal{I} \subset \mathcal{J}$ or $\mathcal{I}$ is *strictly* included in $\mathcal{J}$) if $P^{\mathcal{I}} \subseteq P^{\mathcal{J}}$ (resp. $P^{\mathcal{I}} \subset P^{\mathcal{J}}$) for all predicate symbols P.

Definition 12 (Entailment of Literal). *Let $\mathcal{I}$ be an interpretation. Given a ground literal $P(a_1, \ldots, a_n)$, where $a_i \in \mathcal{U}$, we write $\mathcal{I} \models P(a_1, \ldots, a_n)$ if $(a_1, \ldots, a_n) \in P^{\mathcal{I}}$. Conversely, we write $\mathcal{I} \not\models P(a_1, \ldots, a_n)$ if $(a_1, \ldots, a_n) \notin P^{\mathcal{I}}$. For a non-ground literal L, we write $\mathcal{I} \models L$ if for all grounding substitutions σ for L, we have $\mathcal{I} \models L\sigma$. Conversely, we write $\mathcal{I} \not\models L$, if there exists a grounding substitution σ for L, such that $\mathcal{I} \not\models L\sigma$.*

We overload $\models$ for symbolic interpretations, i.e. we write $\mathcal{S} \models L$ and mean $\mathcal{I}_{\mathcal{S}} \models L$. The following function encodes a clause as an LA formula for evaluation under a given symbolic interpretation.

Definition 13 (Clause Evaluation Function).*Let $\Lambda \,\|\, C$ be a constrained clause where $C = L_1 \vee \cdots \vee L_m$, $L_i = [\neg]P_i(y_{i,1}, \ldots, y_{i,n_i})$ and let $\mathcal{S}$ be a symbolic interpretation. Then the clause evaluation function $(\Lambda \,\|\, C)^{\mathcal{S}}$ is defined as* *follows based on the definitions for σ_i and ϕ_i (for $1 \leq i \leq m$):*

$$\sigma_i := \{x_j \mapsto y_{i,j} \mid 1 \leq j \leq n_i\} \qquad \phi_i := \begin{cases} P_i^{\mathcal{S}} & L_i \textit{ is positive} \\ \neg P_i^{\mathcal{S}} & L_i \textit{ is negative (otherwise)} \end{cases}$$

$$\big(\Lambda \,\|\, C\big)^{\mathcal{S}} := \Big(\bigwedge_{\lambda \in \Lambda} \lambda\Big) \rightarrow \Big(\bigvee_{i=1}^{m} \phi_i \sigma_i\Big)$$

Note that the free variables of $(\Lambda \,\|\, C)^{\mathcal{S}}$ are exactly the free variables of $(\Lambda \,\|\, C)$. Moreover, the substitutions σ_i are necessary in the above definition in order to map the variables in the symbolic interpretation for the predicates $P_i^{\mathcal{S}}$ to the variables that appear as arguments in the literals $P_i(y_{1,1}, \ldots, y_{1,n_i})$.

Proposition 14. *Given a constrained clause $\Lambda \,\|\, C$ with grounding β, we have*

$$\models \big(\Lambda \,\|\, C\big)^{\mathcal{S}} \beta \qquad \textit{if and only if} \qquad \mathcal{S} \models \big(\Lambda \,\|\, C\big)\beta$$

As a corollary of the previous proposition, the entailment $\mathcal{S} \models \Lambda \,\|\, C$ holds if and only if the universal closure of the formula $(\Lambda \,\|\, C)^{\mathcal{S}}$ is valid. This means that for a symbolic interpretation $\mathcal{S}$ it is always computable whether a clause is entailed by $\mathcal{S}$ because there are decision procedures for quantified LRA, LQA, and LIA formulas of finite size.

We require two functions that manipulate LA-formulas directly to express our model construction (cf. Definition 17), i.e. to map solutions for a clause defined by a formula $\text{vars}(\phi)$ to one atom inside the clause. This requires from us to project away all variables in ϕ that appear in the clause but not in the atom.

Definition 15 (Projection). *Let V be a set of variables and ϕ be an* LA-*formula. The projection function π is defined as follows:*

$$\pi(V, \phi) := \exists x_1 \ldots \exists x_n.\, \phi \qquad \textit{where } \{x_1, \ldots, x_n\} = \text{vars}(\phi) \setminus V$$

$\pi(V, \phi)$ is a standard projection function that binds a subset V of the variables in the formula ϕ with existential quantifiers. Note that we also know that $\pi(V, \phi)$ is equivalent to a quantifier-free LA formula just over the variables $x_1, \ldots, x_n$ because there exist quantifier elimination algorithms for LRA, LQA, and LIA [14,32].

A further function $\curlyvee$ is needed when we encounter literals of the form $P(x, x, \ldots)$, i.e., where one variable is shared among two arguments. In this case, we use $\curlyvee$ to express in our symbolic interpretation that the equivalent argument positions must also be equivalent in our interpretation.

Definition 16 (Sharing). *Let $(y_1, \ldots, y_n)$ and $(x_1, \ldots, x_n)$ be tuples of variables with the same length. The sharing function $\curlyvee$, which encodes variable sharing across different argument positions, is defined as follows:*

$$\curlyvee\big((y_1, \ldots, y_n), (x_1, \ldots, x_n)\big) := \bigwedge_{1 \leq i < j \leq n,\ y_i = y_j} x_i \approx x_j$$

2.5 Consequence and Least Model

The notion of a *least model* is common in logic programming. Horn logic programs admit a least model, which is the intersection of all models of the program (see [31, § 6, p. 36]). In our context, the least model of a set of clauses N is the intersection of all models of N. An alternative characterization of the least model of N is through the least fixed point of the one-step consequence operator, which we define as T_N for the context of LA constraints analogously to [27, Section 4]. The one-step consequence operator T_N takes a set of clauses N and an interpretation $\mathcal{I}$ as input and returns an interpretation:

$$P^{T_N(\mathcal{I})} := \left\{ (\vec{y})\beta \;\middle|\; \begin{array}{l} \Lambda \,\|\, \neg P_1(\vec{y_1}) \vee \cdots \vee \neg P_n(\vec{y_n}) \vee P(\vec{y}) \in N, \\ \vDash \Lambda\beta, \text{and } \mathcal{I} \vDash P_i(\vec{y_i})\beta \text{ for } 1 \leq i \leq n \end{array} \right\}$$

The least fixed point of this operator exists by Tarski's Fixed Point Theorem [39]: Interpretations form a complete lattice under inclusion (supremum given by union, infimum given by intersection), and T_N is monotone.

3 Model Construction

In this section we address construction of models for HBS(LA). Throughout this section, we consider a set of constrained Horn clauses N and an order $\prec$ to be given. Our aim is to define an interpretation $\mathcal{I}_N$, such that

$$\mathcal{I}_N \vDash N \qquad \text{if } N \text{ is saturated and } \Box \notin N$$

Towards that goal, we define the operator $\delta(\mathcal{S}, \Lambda \,\|\, C' \vee P(\vec{y}))$. It takes a symbolic interpretation $\mathcal{S}$, and a Horn clause with maximal literal $P(\vec{y})$. It results in a symbolic interpretation that accounts for $\Lambda \,\|\, C' \vee P(\vec{y})$.

Definition 17 (Production Operator). *Let $\Lambda \,\|\, C$ be a constrained Horn clause, where $C = C' \vee P(\vec{y})$, $P(\vec{y}) \succ C'$, and $C' = \neg P_1(y_{1,1}, \ldots, y_{1,n_1}) \vee \cdots \vee \neg P_m(y_{m,1}, \ldots, y_{m,n_m})$. Let $\mathcal{S}$ be a symbolic interpretation, where the free variables of $P^{\mathcal{S}}$ are $\vec{x}$ and the free variables of $P_i^{\mathcal{S}}$ are $\vec{x_i}$ (for $1 \leq i \leq m$). Note that $n = |\vec{y}| = |\vec{x}| = \text{arity}(P)$.*

The production operator $\delta(\mathcal{S}, \Lambda \,\|\, C)$ *results in a new symbolic interpretation*

$$\begin{aligned} P^{\delta(\mathcal{S},\Lambda \,\|\, C)}(\vec{x}) &:= \Big(\pi\big(\{y_1, \ldots, y_n\}, \bigwedge_{\lambda \in \Lambda} \lambda \wedge \bigwedge_{i=1}^{m} (P_i^{\mathcal{S}})\sigma_i\big)\Big)\sigma \wedge \Upsilon\big(\vec{y}, \vec{x}\big) \\ Q^{\delta(\mathcal{S},\Lambda \,\|\, C)}(\vec{z}) &:= \bot \qquad \text{for all } Q \neq P \text{ where } |\vec{z}| = \text{arity}(Q) \end{aligned}$$

where, to map variables from literal arguments to the variables appearing in the symbolic interpretation $\mathcal{S}$ and back, we have the substitutions

$$\begin{aligned} \sigma &:= \{y' \mapsto x_j \mid y' \in \{y_1, \ldots, y_n\} \text{ and } j \text{ is the smallest index s.t. } y_j = y'\} \\ \sigma_i &:= \{x_{i,j} \mapsto y_{i,j} \mid 1 \leq j \leq n_i\} \qquad \text{for } 1 \leq i \leq m \end{aligned}$$

The goal of the operator $\delta(\mathcal{S}, \Lambda \parallel C)$ is to define an extension of the symbolic interpretation $\mathcal{S}$ such that $\mathcal{S} \cup \delta(\mathcal{S}, \Lambda \parallel C)$ satisfies $\Lambda \parallel C$. Note that δ only extends the interpretation over the strictly maximal predicate P. Moreover, due to our predicate order, it only needs to consider the interpretation $\mathcal{S}$ for predicates Q with $Q \prec P$. δ also satisfies the following two symmetrical properties: On the one hand, every grounding τ of $\Lambda \parallel C' \vee P(\vec{y})$ that is not yet satisfied by $\mathcal{S}$ must correspond to solution β of $P^{\delta(\mathcal{S}, \Lambda \parallel C' \vee P(\vec{y}))}$ that satisfies $P(\vec{y})\tau$. On the other hand, every solution β of $P^{\delta(\mathcal{S}, \Lambda \parallel C' \vee P(\vec{y}))}$ must correspond to a grounding of $\Lambda \parallel C' \vee P(\vec{y})$ that is not yet satisfied by $\mathcal{S}$. The first property is needed so $\mathcal{S} \cup \delta(\mathcal{S}, \Lambda \parallel C' \vee P(\vec{y}))$ satisfies $\Lambda \parallel C' \vee P(\vec{y})$. The second property is needed so we do not accidentally extend our interpretation by any solutions not needed to satisfy $\Lambda \parallel C' \vee P(\vec{y})$.

Note that in the above statements β and τ are generally not the same because the variables $\vec{x}$ used to define $P^{\mathcal{S}}$ are not necessarily the same as the variables appearing in the clause $\Lambda \parallel C$ and literal $P(\vec{y})$. There are three reasons for this that are handled by three different methods in our model construction:

1. The variables in $\mathcal{S}$ and $\Lambda \parallel C$ simply do not match, e.g. in $P^{\mathcal{S}} := x_1 \approx 0$ and $\Lambda \parallel C :=\ y_1 > 0 \parallel P(y_1)$. This is handled by the substitution σ in δ that maps all variables in $P(\vec{y})$ to their appropriate variables in $P^{\mathcal{S}}$, e.g. in the previous example $\sigma = \{y_1 \mapsto x_1\}$ and $P^{\delta(\mathcal{S}, \Lambda \parallel C)} = (y_1 > 0)\sigma = x_1 > 0$.
2. Not all variables in $\Lambda \parallel C$ also appear in $P(\vec{y})$, e.g. in $P^{\mathcal{S}} := x_1 \approx 0$ and $\Lambda \parallel C :=\ x_1 \approx y_1 + 1 \wedge y_1 \approx 0 \parallel P(x_1)$. This is handled in δ by the projection operator π (Definition 15) that binds all variables that appear in $\Lambda \parallel C$ but not in $P(\vec{y})$, e.g. in the previous example $P^{\delta(\mathcal{S}, \Lambda \parallel C)} :=\ \pi(\{y_1\}, x_1 \approx y_1 + 1 \wedge y_1 \approx 0)$, where $\pi(\{y_1\}, x_1 \approx y_1 + 1 \wedge y_1 \approx 0) = \exists y_1.\ x_1 \approx y_1 + 1 \wedge y_1 \approx 0$, which is equivalent to $x_1 \approx 1$.
3. Some variables might occur in multiple argument positions, e.g. in $\Lambda \parallel C := \top \parallel P(y_1, y_1)$. This case is covered in δ by the sharing function $\curlyvee$ (c.f. Definition 16) that expresses which variables in $P^{\delta(\mathcal{S}, \Lambda \parallel C)}$ must map to the same value. Continuing the example, $\curlyvee((y_1, y_1), (x_1, x_2)) = x_1 \approx x_2$ and $P^{\delta(\mathcal{S}, \Lambda \parallel C)}(x_1, x_2) = \curlyvee((y_1, y_1), (x_1, x_2))$.

The parts of $P^{\delta(\mathcal{S}, \Lambda \parallel C)}$ that we have not yet discussed are based on the fact that any constrained Horn clause $\Lambda \parallel C' \vee P(\vec{y})$ can also be written as an implication of the form $\phi \rightarrow P(\vec{y})$, where $\phi := \Lambda \wedge P_1(y_{1,1}, \ldots, y_{1,n_1}) \wedge \cdots \wedge P_m(y_{m,1}, \ldots, y_{m,n_m})$ and $\mathcal{S} \not\models \Lambda \parallel C'\tau$ if and only if $\mathcal{S} \models \phi\tau$. This means the groundings τ of $\Lambda \parallel C'$ not satisfied by $\mathcal{S}$ are also the groundings of ϕ satisfied by $\mathcal{S}$. It is straightforward to express these groundings with a conjunctive formula based on Λ and the $P_i^{\mathcal{S}}$. The only challenge is the reverse problem from before, i.e. mapping the variables of $P_i^{\mathcal{S}}$ to the variables in the literals $P_i(y_{1,1}, \ldots, y_{1,n_i})$. This mapping is done in δ by the substitution σ_i.

Now, based on the production operator δ for one clause, we can use an inductive definition over the order $\prec$ to define an interpretation $\mathcal{S}_N$ for all clauses in N. We distinguish the following auxiliary symbolic interpretations: $\mathcal{S}_{\prec P}$ which captures progress up to but excluding the predicate P, Δ_P which captures how P should be interpreted considering $\mathcal{S}_{\prec P}$, and $\mathcal{S}_{\preceq P}$ which captures progress up to and including the predicate P. The symbolic interpretation $\varDelta_P^{\varLambda \,\|\, C}$ is the extension of $\mathcal{S}_{\prec P}$ w.r.t. the single clause $\varLambda \,\|\, C$.

Definition 18 (Model Construction). *Let N be a finite set of constrained Horn clauses. We define symbolic interpretations $\mathcal{S}_{\prec P}$, $\mathcal{S}_{\preceq P}$ and $\varDelta_P$ for all predicates $P \in \varPi(N)$ by mutual induction over $\prec$:*

$$\mathcal{S}_{\preceq P} := \mathcal{S}_{\prec P} \cup \varDelta_P \qquad \mathcal{S}_{\prec P} := \bigcup_{Q \prec P} \varDelta_Q \qquad \varDelta_P := \bigcup_{\varLambda \,\|\, C' \vee P(*) \in N} \varDelta_P^{\varLambda \,\|\, C' \vee P(*)}$$

$$\varDelta_P^{\varLambda \,\|\, C} := \begin{cases} \delta(\mathcal{S}_{\prec P}, \varLambda \,\|\, C) & \text{if } P(\vec{y}) \text{ maximal in } C, \text{ and } \mathcal{S}_{\prec P} \nvDash \varLambda \,\|\, C \\ \mathcal{S}_{\bot} & \text{otherwise} \end{cases}$$

Finally, based on the above inductive definition of $\mathcal{S}_{\prec P}$ for every predicate symbol $P \in \varPi(N)$, we arrive at an overall interpretation for N.

Definition 19 (Candidate Interpretation). *The* candidate interpretation *for N (w.r.t $\prec$), denoted $\mathcal{I}_N$, is the interpretation associated with the symbolic interpretation $\mathcal{S}_N = \bigcup_{P \in \varPi(N)} \varDelta_P$ where P ranges over all predicate symbols occurring in N.*

Note that $\mathcal{S}_N = \mathcal{S}_{\preceq P}$ where P is $\prec$-maximal in $\varPi(N)$. Obviously, we intend that $\mathcal{S}_N \vDash N$ if N is saturated (Theorem 29). Otherwise, i.e. $\mathcal{S}_N \nvDash N$, we can use our construction to find a non-redundant inference (Corollary 30). Consider the following two examples, demonstrating how δ sits at the core of the aforementioned inductive definitions of symbolic interpretations.

*Example 20 (Dependent Interpretation).*Assume $P \prec Q$ and consider the following set of clauses:

$$N := \left\{ \begin{array}{lll} 0 \le y_1 \le 2, 0 \le y_2 \le 2 & \| \; \underline{P(y_1, y_2)} & (C_1), \\ y_3 \ge y_1 + 1, y_4 \ge y_2 + 1 & \| \; P(y_1, y_2) \rightarrow \underline{Q(y_3, y_4)} & (C_2) \end{array} \right\}$$

Maximal literals are underlined. Since the maximal literals of C_1 and C_2 are both positive, ordered resolution cannot be applied. The set is saturated. Since P is the $\prec$-smallest predicate we have $\mathcal{S}_{\prec P} = \mathcal{S}_\bot$. Applying the δ operator yields the following interpretation for P:

$$P^{\mathcal{S}_{\preceq P}} = P^{\delta(\mathcal{S}_{\prec P}, C_1)}(x_1, x_2) = 0 \le x_1 \le 2 \wedge 0 \le x_2 \le 2$$

Then, Q is interpreted relative to P. Consider the clause C_2: For all solutions of its constraint $y_3 \ge y_1 + 1, y_4 \ge y_2 + 1$ our model must also satisfy its logical part $P(y_1, y_2) \rightarrow Q(y_3, y_4)$. The intuition that Q depends on P arises from the implication in the logical part. Whenever the constraint of C_2 and $P(y_1, y_2)$ are satisfied, $Q(y_3, y_4)$ must be satisfied. These are exactly the points defined through $\delta(\mathcal{S}_{\prec Q}, C_2)$, based on $\mathcal{S}_{\prec Q} = \mathcal{S}_{\preceq P} = \delta(\mathcal{S}_{\prec P}, C_1)$:

$$\begin{aligned} Q^{\delta(\mathcal{S}_{\prec Q}, C_2)}(x_1, x_2) &= \exists z_1, z_2.\; x_1 \ge z_1 + 1 \wedge x_2 \ge z_2 + 1 \wedge 0 \le z_1 \le 2 \wedge 0 \le z_2 \le 2 \\ &= x_1 \ge 1 \wedge x_2 \ge 1 \end{aligned}$$

Whenever the conjuncts $0 \leq y_1 \leq 2$ and $0 \leq y_2 \leq 2$ are satisfied, the premise of the implication is true, thus there must be a solution to the interpretation of Q, additionally abiding the constraint of the clause. Since Q is $\prec$-maximal in N, we arrive at $\mathcal{S}_N = \mathcal{S}_{\preceq Q} = \mathcal{S}_{\preceq P} \cup \delta(\mathcal{S}_{\prec Q}, C_2) = \delta(\mathcal{S}_\perp, C_1) \cup \delta(\mathcal{S}_{\preceq P}, C_2)$. See Fig. 1a for a visual representation of $\mathcal{S}_N$.

Example 21 (Unsaturated Clause Set). Assume $P \prec Q$ and consider the following set of clauses:

$$N := \left\{ \begin{array}{llll} y_1 < 0 \,\|\, \underline{P(y_1)} & (C_1), & y_1 < 1 \,\|\, \underline{Q(y_1)} & (C_3), \\ y_1 > 0 \,\|\, \underline{P(y_1)} & (C_2), & y_1 \leq 0 \,\|\, \underline{Q(y_1)} \rightarrow P(y_1) & (C_4) \end{array} \right\}$$

Maximal literals are underlined. Note that a resolution inference is possible, since the maximal literals of C_3 and C_4 have opposite polarity, use the same predicate symbol, and are trivially unifiable. Thus, in this example we consider the effect of applying our model construction to a clause set that is *not* saturated. Since P is $\prec$-minimal, we start with the following steps:

$$\begin{array}{lll} \mathcal{S}_{\prec P} = \mathcal{S}_\perp & P^{\delta(\mathcal{S}_{\prec P}, C_1)}(x_1) = x_1 < 0 & \\ & P^{\delta(\mathcal{S}_{\prec P}, C_2)}(x_1) = x_1 > 0 & P^{\mathcal{S}_{\preceq P}}(x_1) = x_1 < 0 \lor x_1 > 0 \end{array}$$

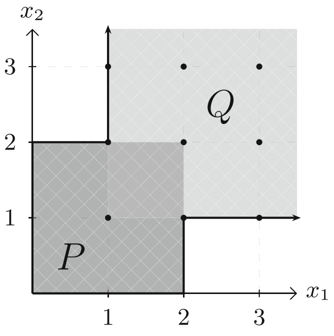
Next, we obtain the following results for Q:

$$\begin{array}{lll} \mathcal{S}_{\prec Q} = \mathcal{S}_{\preceq P} & Q^{\delta(\mathcal{S}_{\prec Q}, C_3)}(x_1) = x_1 < 1 & \\ & Q^{\delta(\mathcal{S}_{\prec Q}, C_4)}(x_1) = \perp & Q^{\mathcal{S}_{\preceq Q}}(x_1) = x_1 < 1 \lor \perp = x_1 < 1 \end{array}$$

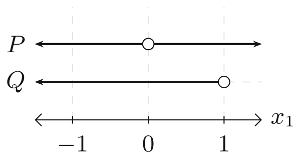
See Fig. 1b for a visual representation of $\mathcal{S}_N = \mathcal{S}_{\preceq Q}$. Note that $\mathcal{S}_N \not\models C_4$, since we have $\mathcal{S}_N \models Q(0)$ but $\mathcal{S}_N \not\models P(0)$. Thus, by using the constructed model, we can pinpoint clauses that contradict that N is saturated. Applying resolution to C_3 and C_4 leads to the clause $y_1 \leq 0 \,\|\, P(y_1)$ labelled C_5. If we then add C_5 to N, we instead get $P^{\mathcal{S}_{\preceq P}}(x_1) = x_1 < 0 \lor x_1 > 0 \lor x_1 \leq 0 = \top$.

In the following, we clarify some properties of the construction. We provide an upper bound for the number of LA atoms and quantifiers in the symbolic model for LRA and LQA. Although we do not state it explicitly, the estimate for LIA works in a similar way, but due to the higher complexity of LIA quantifier elimination, the size of the symbolic model grows triple exponentially [36].

Proposition 22. *If N is a finite set of* LRA/LQA *constrained Horn clauses, and $\mathcal{S}'_N$ the result of applying quantifier elimination to $\mathcal{S}_N$ then, for every predicate symbol $P \in \Pi(N)$, the number of* LA *atoms in $P^{\mathcal{S}'_N}$ is in $O(m^{2 \cdot q^{p-1}} \cdot n^{2 \cdot q^{p-1}} \cdot (l + a^2)^{q^p})$ where n is the max. number of clauses with the same max. predicate, m is the max. number of non-arithmetic literals in a clause, l is the max. number of arithmetic literals in a clause, a is the max. arity of any predicate, $p = |\Pi(N)|$, q is the max. difference of variables in any clause and its positive maximal literal.*



(a) Result of Example 20.



(b) Result of Example 21.

Fig. 1. Visual representation of the models resulting from Examples 20 and 21.

Corollary 23 (Effective Construction). *If N is a finite set of constrained Horn clauses then for every predicate $P \in \Pi(N)$, $P^{\mathcal{S}_N}$ is a linear arithmetic formula of finite size, and can be computed in a finite number of steps.*

We show that all points in $P_N^{\mathcal{I}}$ are necessary and justified in some sense, that $\mathcal{I}_N$ is indeed a model of N, and that $\mathcal{I}_N$ is also the least model of N if N is saturated. The notion of whether a clause is productive captures whether it contributes something to the symbolic interpretation.

Definition 24 (Productive Clause). *Let P be a predicate symbol with* arity$(P) = n$. *We say that $\Lambda \,\|\, C$* produces *$P(a_1, \ldots, a_n)$ if $(a_1, \ldots, a_n) \in P^{\Delta_P^{\Lambda \,\|\, C}}$.*

Next, we want to formally express that every element of the resulting interpretation is justified. Firstly, we express that the operator δ will produce points such that every clause is satisfied whenever necessary, i.e. whenever the maximal literal of the clause is $P(*)$ and the maximal literal not satisfied by $\mathcal{S}_{\prec P}$.

Proposition 25. *Let $\Lambda_C \,\|\, C$ where $C = C' \vee P(\vec{y})$ and $C' \prec P(\vec{y})$. Let τ be a grounding substitution for $\Lambda_C \,\|\, C$. If $\mathcal{S}_{\prec P} \not\models (\Lambda_C \,\|\, C)\tau$, then $\models \Lambda_C\tau$ and $\mathcal{S}_{\preceq P} \models P(\vec{y})\tau$, thus $\mathcal{S}_{\preceq P} \models (\Lambda_C \,\|\, C)\tau$.*

Secondly, we express that for every point in $P_N^{\mathcal{I}}$, it is justified in the sense that there is a clause that produced the point, i.e. this clause would otherwise not be satisfied by the resulting interpretation.

Proposition 26. *If $\mathcal{S}_{\preceq P} \models P(\vec{a})$, then there exists a clause $\Lambda_C \,\|\, C$ where $C = C' \vee P(\vec{y})$ and $C' \prec P(\vec{y})$, and there exists a grounding τ for $\Lambda_C \,\|\, C$, such that $P(\vec{a}) = P(\vec{y})\tau$ and $\mathcal{S}_{\prec P} \not\models (\Lambda_C \,\|\, C)\tau$.*

Also, observe that once the maximal predicate P of a given clause is interpreted by $\mathcal{S}_{\preceq P}$, the interpretation of the clause does not change for $\mathcal{S}_{\preceq Q}$ where $Q \succ P$.

Corollary 27. *Let $P \prec Q \preceq R$, and P be maximal in clause C. If $\mathcal{S}_{\preceq P} \models \Lambda_C \parallel C$ or $\mathcal{S}_{\prec Q} \models \Lambda_C \parallel C$, then $\mathcal{S}_{\prec R} \models \Lambda_C \parallel C$ and $\mathcal{S}_{\preceq R} \models \Lambda_C \parallel C$.*

As a result, we know that the full model satisfies N, i.e., $\mathcal{I}_N \models N$ if every clause is satisfied at the point of the construction, where the interpretation of its maximal predicate P stays fixed.

Proposition 28. *For every clause $\Lambda_C \parallel C \in N$ with maximal predicate P, if $\mathcal{S}_{\preceq P} \models \Lambda_C \parallel C$, then $\mathcal{I}_N \models N$.*

With the above propositions (and some auxiliary properties that can be found in [12]) we show that indeed $\mathcal{I}_N \models N$ if N is saturated and does not contain the empty clause.

Theorem 29. *Let $\prec$ be a clause ordering and N be a set of constrained Horn clauses. If (1.) N is saturated w.r.t. $\prec$-resolution, and (2.) $\square \notin N$, then $\mathcal{I}_N \models N$.*

For clauses with positive maximal literal, the fact that they are satisfied by $\mathcal{I}_N$ follows from Proposition 25. For clauses with maximal literal $\neg P(*)$, we prove this theorem by contradiction: If there is a minimal clause $\Lambda_C \parallel C$ such that $\mathcal{S}_N \not\models \Lambda_C \parallel C$. We can then exploit Proposition 26 to find the smallest clause $\Lambda_D \parallel D$ that produced the respective instance $P(\vec{a})$. Applying hierarchic $\prec$-resolution to $\Lambda_C \parallel C$ and $\Lambda_D \parallel D$ then yields a non-redundant clause. This idea then leads to the following theorem.

Corollary 30. *Let $\prec$ be a clause ordering and N be a set of constrained Horn clauses. If (1.) $\mathcal{I}_N \not\models N$, and (2.)$\square \notin N$, then there exist two clauses $\Lambda_C \parallel C$, $\Lambda_D \parallel D \in N$ such that: (1.) $\Lambda_C \parallel C$ is the smallest clause not satisfied by $\mathcal{I}_N$, i.e. there exists a grounding τ such that $\mathcal{I}_N \not\models (\Lambda_C \parallel C)\tau$, but there does not exist a clause $\Lambda_{C'} \parallel C' \in N$ with grounding τ', such that $\mathcal{I}_N \not\models (\Lambda_{C'} \parallel C')\tau'$ and $(\Lambda_{C'} \parallel C')\tau' \prec (\Lambda_C \parallel C)\tau$, (2.)$\neg P(\vec{a})$ is the maximal literal of $(\Lambda_C \parallel C)\tau$, (3.)$\Lambda_D \parallel D$ is the minimal clause that produces $P(\vec{a})$, (4.)$\prec$-resolution is applicable to $\Lambda_C \parallel C$ and $\Lambda_D \parallel D$, and (5.)the resolvent of $\Lambda_C \parallel C$ and $\Lambda_D \parallel D$ is not redundant w.r.t. N.*

Additionally, we show that $\mathcal{I}_N$ is the least model of N, establishing a connection between our approach and the literature on constrained Horn clauses (see [27, Section 4] and [15, Section 2.4.1]) and logic programming (see [31, § 6, p. 37]).

Theorem 31. *$\mathcal{I}_N$ is the least model of N.*

Fermüller and Leitsch define four postulates (see [19] as cited in [13, Section 5.1, p. 234]) regarding *automated model building*. In the following, we instantiate the postulates for our setting. By $\mathfrak{S}(N)$ we denote the set of all symbolic interpretations of the set of constrained Horn clauses N. We argue how our approach satisfies all postulates, one by one:

Uniqueness. *Each element of $\mathfrak{S}(N)$ specifies a single interpretation of N.*
We have shown (cf. Theorem 31) that $\mathcal{I}_N$, the model represented by $\mathcal{S}_N$, is the least model of N, which is unique.

Atom Test. *There exists a fast procedure to evaluate arbitrary ground atoms over $\Pi(N)$ in the interpretation defined by a $\mathcal{S}$ in $\mathfrak{S}(N)$.*
This is a special case of clause evaluation (cf. Proposition 14): A ground atom $P(\vec{t})$ is true in $\mathcal{S}$ if and only if $\vDash P^{\mathcal{S}}(\vec{x})\{x_i \mapsto t_i \mid 1 \leq i \leq |\vec{x}| = |\vec{t}|\}$. Fulfillment of this property thus hinges on the meaning of "fast". We consider methods for evaluating formulas of LA against points to be fast.

Formula Evaluation. *There exists an algorithm deciding the truth values of arbitrary formulas in interpretations defined by $\mathcal{S} \in \mathfrak{S}(N)$.*
Proposition 14 states that evaluating a constrained clause $\Lambda \parallel C$ is achieved by evaluating the universal closure of $(\Lambda \parallel C)^{\mathcal{S}}$, which is decided by quantifier elimination algorithms for LRA, LQA, and LIA [14,32]. For sets of clauses, evaluate each clause individually and combine the results conjunctively.

Equivalence Test. *There exists an algorithm which decides whether two representations $\mathcal{S}_1$ and $\mathcal{S}_2$ in $\mathfrak{S}(N)$ describe the same interpretation.*
$\mathcal{S}_1$ and $\mathcal{S}_2$ describe the same interpretation if and only if for each predicate $P \in \Pi(N)$ of arity n, we have $\forall x_1 \ldots \forall x_n. P^{\mathcal{S}_1}(\vec{x}) \leftrightarrow P^{\mathcal{S}_2}(\vec{x})$.

4 Conclusion

We have presented the first model construction approach to Horn clauses with linear arithmetic constraints based on hierarchic ordered resolution, (cf. Definition 19). The linear arithmetic constraints may range over the reals, rationals, or integers. The computed model is the canonical least model of the saturated Horn clause set (cf. Theorem 31). Clauses can be effectively evaluated with respect to the model (cf. Proposition 14). This offers a way to explore the properties of a saturated clause set, e.g., if the set represents a failed refutation attempt.

Future Work. It is straightforward to see that any symbolic LQA model is also a symbolic LRA model. (This holds due to convexity of conjunctions of ground LQA atoms.) So even if the axiom of choice is not assumed, there is an alternative way to obtain a model for a HBS(LRA) clause set: Simply treat it as an HBS(LQA) clause set, saturate it and construct its model based on HBS(LQA).

In this work, we restrict ourselves to only one sort LA per set of clauses. An extension to a many-sorted setup, e.g. including first-order variables with sort $\mathcal{F}$ is possible. This can even be simulated, by encoding first-order constants as concrete natural numbers via a bijection to $\mathbb{N}$, since $\mathbb{N} \subset \mathcal{U}$. By not placing any arithmetic constraints on the variables used for the encoding, it can be read off and mapped back from the resulting model.

One obvious challenge is relaxation of the restriction to Horn clauses. With respect to ordered resolution saturation there is typically no difference in the sense that if a Horn fragment can always be finitely saturated, so can the non-Horn fragment be. However, our proposed ordering for the model construction at the granularity of predicate symbols will not suffice in this general case, and the key to overcome this challenge seems to be the appropriate treatment of clauses with maximal literals of the same predicate. Backtracking on the selection of literals might also be sufficient.

The approach we presented does not exploit features of linear arithmetic beyond equality and the existence of a well-founded order for the underlying universe $\mathcal{U}$. The results may therefore be adapted to other constraint domains such as non-linear arithmetic.

Acknowledgements. We thank our reviewers for their constructive comments.

References

1. Althaus, E., Kruglov, E., Weidenbach, C.: Superposition modulo linear arithmetic SUP(LA). In: Ghilardi, S., Sebastiani, R. (eds.) FroCoS 2009. LNCS (LNAI), vol. 5749, pp. 84–99. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04222-5_5
2. Bachmair, L., Ganzinger, H., Waldmann, U.: Superposition with simplification as a decision procedure for the monadic class with equality. In: Gottlob, G., Leitsch, A., Mundici, D. (eds.) KGC 1993. LNCS, vol. 713, pp. 83–96. Springer, Heidelberg (1993). https://doi.org/10.1007/BFb0022557
3. Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierarchic first-order theories. AAECC **5**, 193–212 (1994). https://doi.org/10.1007/BF01190829
4. Basin, D.A., Ganzinger, H.: Automated complexity analysis based on ordered resolution. JACM **48**(1), 70–109 (2001). https://doi.org/10.1145/363647.363681
5. Baumgartner, P., Fuchs, A., Tinelli, C.: (LIA) - model evolution with linear integer arithmetic constraints. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 258–273. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89439-1_19
6. Baumgartner, P., Waldmann, U.: Hierarchic superposition revisited. In: Lutz, C., Sattler, U., Tinelli, C., Turhan, A.-Y., Wolter, F. (eds.) Description Logic, Theory Combination, and All That. LNCS, vol. 11560, pp. 15–56. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22102-7_2
7. Bjørner, N., Gurfinkel, A., McMillan, K., 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. LNCS, vol. 9300, pp. 24–51. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23534-9_2
8. Bromberger, M., et al.: A sorted datalog hammer for supervisor verification conditions modulo simple linear arithmetic. In: TACAS 2022. LNCS, vol. 13243, pp. 480–501. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_27
9. Bromberger, M., Dragoste, I., Faqeh, R., Fetzer, C., Krötzsch, M., Weidenbach, C.: A datalog hammer for supervisor verification conditions modulo simple linear arithmetic. In: Konev, B., Reger, G. (eds.) FroCoS 2021. LNCS (LNAI), vol. 12941, pp. 3–24. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86205-3_1

10. 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.) VMCAI 2021. LNCS, vol. 12597, pp. 511–533. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67067-2_23
11. Bromberger, M., Leutgeb, L., Weidenbach, C.: An efficient subsumption test pipeline for BS(LRA) clauses. In: Blanchette, J., Kovács, L., Pattinson, D. (eds.) IJCAR 2022. LNCS, vol. 13385, pp. 147–168. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-10769-6_10
12. Bromberger, M., Leutgeb, L., Weidenbach, C.: Symbolic model construction for saturated constrained horn clauses. arXiv (2023). https://doi.org/10.48550/arXiv.2305.05064
13. Caferra, R., Leitsch, A., Peltier, N.: Automated Model Building, APLS, vol. 31. Springer, Dordrecht (2004). https://doi.org/10.1007/978-1-4020-2653-9
14. Cooper, D.C.: Theorem proving in arithmetic without multiplication. Mach. Intell. **7**, 91–99 (1972)
15. De Angelis, E., Fioravanti, F., Gallagher, J.P., Hermenegildo, M.V., Pettorossi, A., Proietti, M.: Analysis and transformation of constrained horn clauses for program verification. TPLP **22**(6), 974–1042 (2022). https://doi.org/10.1017/S1471068421000211
16. Downey, P.J.: Undecidability of presburger arithmetic with a single monadic predicate letter. Center for Research in Computer Technology, Harvard University, Technical report (1972)
17. Fedyukovich, G., Zhang, Y., Gupta, A.: Syntax-guided termination analysis. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 124–143. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_7
18. Feferman, S.: Some applications of the notions of forcing and generic sets. Fundamenta Mathematicae. **56**(3), 325–345 (1964). http://eudml.org/doc/213821
19. Fermüller, C.G., Leitsch, A.: Hyperresolution and automated model building. LOGCOM **6**(2), 173–203 (1996). https://doi.org/10.1093/logcom/6.2.173
20. Fermüller, C.G., Leitsch, A.: Decision procedures and model building in equational clause logic. IGPL **6**(1), 17–41 (1998). https://doi.org/10.1093/jigpal/6.1.17
21. Fiori, A., Weidenbach, C.: SCL with theory constraints. arXiv (2020). http://arxiv.org/abs/2003.04627
22. Gange, G., Navas, J.A., Schachte, P., Søndergaard, H., Stuckey, P.J.: Horn clauses as an intermediate representation for program analysis and transformation. TPLP **15**(4–5), 526–542 (2015). https://doi.org/10.1017/S1471068415000204
23. Ganzinger, H., de Nivelle, H.: A superposition decision procedure for the guarded fragment with equality. In: 14th LICS, 1999, pp. 295–303. IEEE Computer Society (1999). https://doi.org/10.1109/LICS.1999.782624
24. Grebenshchikov, S., Lopes, N.P., Popeea, C., Rybalchenko, A.: Synthesizing software verifiers from proof rules. In: PLDI, pp. 405–416. ACM (2012). https://doi.org/10.1145/2254064.2254112
25. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 157–171. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_13
26. Horbach, M., Voigt, M., Weidenbach, C.: The universal fragment of presburger arithmetic with unary uninterpreted predicates is undecidable. arXiv (2017). http://arxiv.org/abs/1703.01212
27. Jaffar, J., Maher, M.J.: Constraint logic programming: a survey. JLP **19**(20), 503–581 (1994). https://doi.org/10.1016/0743-1066(94)90033-7

28. Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 17–34. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_2
29. Korovin, K., Voronkov, A.: Integrating linear arithmetic into superposition calculus. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 223–237. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74915-8_19
30. Kruglov, E.: Superposition modulo theory. Ph.D. thesis, Saarland University (2013). http://scidok.sulb.uni-saarland.de/volltexte/2013/5559/
31. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Cham (1987). https://doi.org/10.1007/978-3-642-83189-8
32. Loos, R., Weispfenning, V.: Applying linear quantifier elimination. Comput. J. **36**(5), 450–462 (1993). https://doi.org/10.1093/comjnl/36.5.450
33. López-García, P., Darmawan, L., Klemen, M., Liqat, U., Bueno, F., Hermenegildo, M.V.: Interval-based resource usage verification by translation into horn clauses and an application to energy consumption. TPLP **18**(2), 167–223 (2018). https://doi.org/10.1017/S1471068418000042
34. McMillan, K.L.: Lazy annotation revisited. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 243–259. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_16
35. Mesnard, F., Payet, É., Vidal, G.: Concolic testing in CLP. TPLP **20**(5), 671–686 (2020). https://doi.org/10.1017/S1471068420000216
36. Oppen, D.C.: A 2ˆ2ˆ2ˆPN upper bound on the complexity of Presburger arithmetic. JCSS **16**(3), 323–332 (1978). https://doi.org/10.1016/0022-0000(78)90021-1
37. Rümmer, P.: A constraint sequent calculus for first-order logic with linear integer arithmetic. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 274–289. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89439-1_20
38. Spoto, F., Mesnard, F., Payet, É.: A termination analyzer for java bytecode based on path-length. TOPLAS **32**(3), 8:1-8:70 (2010). https://doi.org/10.1145/1709093.1709095
39. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pac. J. Math. **5**(2), 285–309 (1955). https://doi.org/10.2140/pjm.1955.5.285
40. Weidenbach, C.: Automated reasoning building blocks. In: Meyer, R., Platzer, A., Wehrheim, H. (eds.) Correct System Design. LNCS, vol. 9360, pp. 172–188. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23506-6_12