SCL with Theory Constraints

Martin Bromberger, Alberto Fiori, Christoph Weidenbach

SCL with Theory Constraints

Martin Bromberger
Max Planck Institute for Informatics
Saarland Informatics Campus Saarbrücken

Alberto Fiori
Max Planck Institute for Informatics
Saarland Informatics Campus Saarbrücken and
Graduate School of Computer Science Saarbrücken, Germany

Christoph Weidenbach
Max Planck Institute for Informatics
Saarland Informatics Campus, Saarbrücken Germany

October 12, 2020

Abstract

We lift the SCL calculus for first-order logic without equality to the SCL(T) calculus for first-order logic without equality modulo a background theory. In a nutshell, the SCL(T) calculus describes a new way to guide hierarchic resolution inferences by a partial model assumption instead of an *a priori* fixed order as done for instance in hierarchic superposition. The model representation consists of ground background theory literals and ground foreground first-order literals. One major advantage of the model guided approach is that clauses generated by SCL(T) enjoy a non-redundancy property that makes expensive testing for tautologies and forward subsumption completely obsolete. SCL(T) is a semi-decision procedure for pure clause sets that are clause sets without first-order function symbols ranging into the background theory sorts. Moreover, SCL(T) can be turned into a decision procedure if the considered combination of a first-order logic modulo a background theory enjoys an abstract finite model property.

1 Introduction

The combination of first-order logic reasoning with theories is still a challenge. More specifically, there are many open problems left if first-order clauses containing universally quantified variables are combined with theories such as (linear) arithmetic. While the ground case of such a combination is known to be solvable by SMT [NOT06], hierarchic superposition [BGW94, KW12], the sequent calculus [Rüm08], or model evolution [BFT08], the existence of universally quantified variables poses a number of new challenges.

For instance, the combination of linear rational arithmetic (LRA) with the Bernays Schoenfinkel fragment of first-order logic (BS(LRA)) is already undecidable if we allow a single, free constant ranging into the arithmetic sort. [Dow72, HVW17]. This can be shown by encoding the halting problem of a 2-counter machine [Min67]. In Section 2 we show another example, Example 4, where we force a predicate to contain exactly the natural numbers. The natural numbers cannot be defined in the first-order theory of LRA without extra first-order predicates, nor in pure first-order logic. For a number of universally quantified fragments there exist complete methods and some fragments are even decidable [GdM09, KW12, Voi17]. In addition, completeness for undecidable fragments can be obtained. For example, if the first-order part of BS(LRA) consists only of variables and predicates, then hierarchic superposition is refutationally complete [BGW94]. Already the addition of a single, free constant ranging into the arithmetic sort may destroy refutational completeness, if such a constant cannot be mapped to a background theory domain constant, see Example 4.

In this paper we introduce a new calculus SCL(T) for the combination of a background theory with a foreground first-order logic without equality. As usual in a hierarchic setting, we assume the background theory to be term-generated and compact. In this paper we only consider *pure* clause sets where the only symbols occurring in the clause set from the foreground logic are predicates and variables. As a running example, we present the combination of linear rational arithmetic (LRA) with the Bernays Schoenfinkel fragment of first-order logic (BS(LRA)) where for simplicity we only consider constants from LRA. In Section 2, we discuss that this does not restrict expressiveness.

For example, for a clause set out of BS(LRA) where we write clauses in a constraint form $\Lambda \| C$ where Λ is a conjunction of LRA atoms of the form $x = k \cdot y$ for $k \in \mathbb{N}$ and LRA sort variables x, y, or atoms $x > 0$, and the first-order part C only contains a single monadic predicate P applied to some variable, unsatisfiability is already undecidable [Dow72, HVW17]. This can be shown by encoding the halting problem of a 2-counter machine [Min67].

For a further example, the two clauses

$$x = 0 \| \operatorname{Nat}(x) \quad y = x + 1 \| \neg \operatorname{Nat}(x) \vee \operatorname{Nat}(y)$$

belong to our fragment as well. For any BS(LRA) algebra satisfying the two clauses, the predicate Nat contains the natural numbers. In Section 2 we will even show that it can be forced to contain exactly the natural numbers by adding further clauses from our fragment, Example 4.

In contrast to superposition based approaches to hierarchic reasoning [BGW94, KW12, BW19] where inferences are restricted by an *a priori* ordering, our idea is to select inferences via a partial model assumption, similar to CDCL [SS96, JS97] and respective calculi using an explicit model assumption to direct reasoning in first-order logic [BFT06, PdMB10, Kor13, TW15, AW15, BP16]. In particular, we want to lift our previously obtained results for model-based reasoning in first-order logic [FW19] to first-order logic modulo a background theory. There, models are ground and either extended by guessing an undefined ground literal or by propagating a new ground literal with respect to a clause and the current ground model assumption. The two clauses $x = 0 \| \operatorname{Nat}(x)$, $y = x + 1 \| \neg \operatorname{Nat}(x) \vee \operatorname{Nat}(y)$ enable already infinitely many propagations

$$\operatorname{Nat}(0), \operatorname{Nat}(1), \operatorname{Nat}(2), \ldots.$$

Therefore, exhaustive propagation cannot be permitted. On the other hand, exhaustive propagation guarantees that learned clauses are not redundant with

respect to an adopted superposition redundancy notion [Wei15, FW19]. Restricting a calculus to non-exhaustive propagation while having non-redundancy properties for learned clauses is difficult to obtain. One reason why model-based reasoning for LIA is currently inferior compared to classical branch and bound approaches used in SMT [NOT06, BSW15] is a missing non-redundancy guarantee for inferred inequations. This is partly a result of the fact that exhaustive propagation does not terminate in LIA as well and is therefore not permitted. Our solution in this paper is to restrict model assumptions to finite ground models, similar to [FW19]. The models are build with respect to a fixed, finite set B of foreground constants of the arithmetic sort. Propagations are not exhaustive but restricted to a finite number, typically not exhausting B. If B is not grown, the proposed calculus SCL(T) always terminates. Either by finding a contradiction or finding a model with respect to B. Of course, this (finite) model needs not to be extendable to a model of the clause set. Satisfiability is undecidable for pure clause sets. Still for some further restricted fragments of pure clause sets the approach can be turned into a decision procedure, see Section 4. In general, the set B has to be extended during the search for a refutation. Nevertheless, we can prove that learned clauses in SCL(T) are always non-redundant with respect to an adopted superposition redundancy notion, Lemma 22. The notion includes subsumption of constraint clauses and semantic tautologies. That means, SCL(T) will never learn a true clause, nor a clause that is subsumed by an existing clause. Restricting the model representation language to ground literals of the background and foreground logic enables detecting false clauses or propagating clauses efficiently by SMT. Still, the SCL(T) calculus is sound, Lemma 15, and refutationally complete, Theorem 25, i.e., a regular strategy is guaranteed to find a refutation for pure clause sets if it exists.

Related Work: In contrast to variants of hierarchic superposition [BGW94, KW12, BW19] SCL(T) selects clauses via a partial model assumption and not via an ordering. This has the advantage that SCL(T) does not generate redundant clauses. It lacks the implicit model building capabilities of hierarchic superposition. Instead it generates finite models candidates that need to be extended to overall models of the clause set, see Example 14. One way to deal with universally quantified variables in an SMT setting is via instantiation [GdM09, RBF18]. This has shown to be practically useful in many applications. It typically comes without completeness guarantees and it does not learn any new constrained clauses. An alternative is to combine SMT techniques with superposition [dMB08] where the ground literals from an SMT model assumption are resolved by superposition with first-order clauses. SCL(T) does not resolve with respect to its ground model assumption but on the original clauses with variables. Background theories can also be build into first-order superposition in a kind of lazy way. This direction has been followed by SPASS+T [PW06] and Vampire [KV13]. The idea is to axiomatize part of the background theory in first-order logic and to direct ground literals of the background theory to SMT solver. Also this approach has shown to be practically useful but comes without any completeness guarantees and generated clauses may be redundant. Model evolution [BFT06] has also been extended with linear integer arithmetic [BFT08] where universally quantified integer variables are finitely bound from the beginning. A combination of first-order logic with linear integer arithmetic has also

been built into a sequent calculus [Rüm08] that operates in the style of a free-variable tableau calculus with incremental closure. No new clauses are learned.

Organization of the Paper: After a section fixing notation, notions and some preliminary work, Section 2, the following Section 3 introduces the SCL(T) calculus and proves its properties. The final Section 4 discusses extensions to model building, further improvements and summarizes the obtained results.

2 Preliminaries

Many-Sorted First-Order Logic without Equality: A *many-sorted signature* $\Sigma = (\mathcal{S}, \Omega, \Pi)$ is a triple consisting of a finite, non-empty set $\mathcal{S}$ of *sort symbols*, a non-empty set Ω of *operator symbols* (also called *function symbols*) over $\mathcal{S}$ and a finite set Π of *predicate symbols* over $\mathcal{S}$. For every sort from $\mathcal{S}$ there is at least one constant symbol in Ω of this sort. First-order terms, atoms, literals, clauses, formulas and substitutions are defined in the usual many-sorted way where an additional infinite set $\mathcal{X}$ of variables is assumed, such that for each sort from $\mathcal{S}$ there are infinitely many variables of this sort in $\mathcal{X}$. For each sort $S \in \mathcal{S}$, $T_S(\Sigma, \mathcal{X})$ denotes the set of all terms of sort S and $T_S(\Sigma)$ the set of all ground terms of sort S.

For notation, a, b, c are constants from Ω, w, x, y, z variables from $\mathcal{X}$, and if we want to emphasize the sort of a variable, we write x_S for a variable of sort S; t, s denote terms, P, Q, R predicates from Π, A, B atoms, L, K, H denote literals, C, D denote clauses, and N denotes a clause set. For substitutions we write σ, δ, ρ. Substitutions are well-sorted: if $x_s\sigma = t$ then $t \in T_S(\Sigma, \mathcal{X})$, they have a finite domain $\mathrm{dom}(\sigma) = \{x \mid x\sigma \neq x\}$ and their codomain is denoted by $\mathrm{codom}(\sigma) = \{x\sigma \mid x \in \mathrm{dom}(\sigma)\}$. The application of substitutions is homomorphically extended to non-variable terms, atoms, literals, clauses, and formulas. The complement of a literal is denoted by the function comp. For a literal L, $|L|$ denotes its respective atom. The function atoms computes the set of atoms from a clause or clause set. The function vars maps terms, literals, clauses to their respective set of contained variables. The function con maps terms, literals, clauses to their respective set of constants. A term, atom, clause, or a set of these objects is *ground* if it does not contain any variable, i.e., the function vars returns the empty set. A substitution σ is *ground* if $\mathrm{codom}(\sigma)$ is ground. A substitution σ is *grounding* for a term t, literal L, clause C if $t\sigma$, $L\sigma$, $C\sigma$ is ground, respectively. The function gnd computes the set of all ground instances of a literal, clause, or clause set. Given a set of constants B, the function gnd_B computes the set of all ground instances of a literal, clause, or clause set where the grounding is restricted to use constants from B. The function mgu denotes the *most general unifier* of two terms, atoms, literals. As usual, we assume that any mgu of two terms or literals does not introduce any fresh variables and is idempotent.

The semantics of many-sorted first-order logic is given by the notion of an algebra: let $\Sigma = (\mathcal{S}, \Omega, \Pi)$ be a many-sorted signature. A Σ-*algebra* $\mathcal{A}$, also called Σ-*interpretation*, is a mapping that assigns (i) a non-empty carrier set $S^{\mathcal{A}}$ to every sort $S \in \mathcal{S}$, so that $(S_1)^{\mathcal{A}} \cap (S_2)^{\mathcal{A}} = \emptyset$ for any distinct sorts $S_1, S_2 \in \mathcal{S}$, (ii) a total function $f^{\mathcal{A}} : (S_1)^{\mathcal{A}} \times \ldots \times (S_n)^{\mathcal{A}} \to (S)^{\mathcal{A}}$ to every operator $f \in \Omega$, $\mathrm{arity}(f) = n$ where $f : S_1 \times \ldots \times S_n \to S$, (iii) a relation

$P^{\mathcal{A}} \subseteq ((S_1)^{\mathcal{A}} \times \ldots \times (S_m)^{\mathcal{A}})$ to every predicate symbol $P \in \Pi$ with arity$(P) = m$. The semantic entailment relation $\models$ is defined in the usual way. We call a Σ-algebra $\mathcal{A}$ *term-generated* if $\mathcal{A}$ fulfills the following condition: whenever $\mathcal{A}$ entails all groundings $C\sigma$ of a clause C (i.e., $\mathcal{A} \models C\sigma$ for all grounding substitutions σ of a clause C), then $\mathcal{A}$ must also entail C itself (i.e., $\mathcal{A} \models C$).

Hierarchic Reasoning: Starting point of a hierarchic reasoning [BGW94, BW19] is a background theory $\mathcal{T}^{\mathcal{B}}$ over a many-sorted signature $\Sigma^{\mathcal{B}} = (\mathcal{S}^{\mathcal{B}}, \Omega^{\mathcal{B}}, \Pi^{\mathcal{B}})$ and a non-empty set of term-generated $\Sigma^{\mathcal{B}}$-algebras $\mathcal{C}^{\mathcal{B}}$: $\mathcal{T}^{\mathcal{B}} = (\Sigma^{\mathcal{B}}, \mathcal{C}^{\mathcal{B}})$. A constant $c \in \Omega^{\mathcal{B}}$ is called a *domain constant* if $c^{\mathcal{A}} \neq d^{\mathcal{A}}$ for all $\mathcal{A} \in \mathcal{C}^{\mathcal{B}}$ and for all $d \in \Omega^{\mathcal{B}}$ with $d \neq c$. The background theory is then extended via a foreground signature $\Sigma^{\mathcal{F}} = (\mathcal{S}^{\mathcal{F}}, \Omega^{\mathcal{F}}, \Pi^{\mathcal{F}})$ where $\mathcal{S}^{\mathcal{B}} \subseteq \mathcal{S}^{\mathcal{F}}$, $\Omega^{\mathcal{B}} \cap \Omega^{\mathcal{F}} = \emptyset$, and $\Pi^{\mathcal{B}} \cap \Pi^{\mathcal{F}} = \emptyset$. Hierarchic reasoning is based on a background theory $\mathcal{T}^{\mathcal{B}}$ and a respective foreground signature $\Sigma^{\mathcal{F}}$: $\mathcal{H} = (\mathcal{T}^{\mathcal{B}}, \Sigma^{\mathcal{F}})$. It has its associated signature $\Sigma^{\mathcal{H}} = (\mathcal{S}^{\mathcal{F}}, \Omega^{\mathcal{B}} \cup \Omega^{\mathcal{F}}, \Pi^{\mathcal{B}} \cup \Pi^{\mathcal{F}})$ generating *hierarchic* $\Sigma^{\mathcal{H}}$-algebras. A $\Sigma^{\mathcal{H}}$-algebra $\mathcal{A}$ is called *hierarchic* with respect to its background theory $\mathcal{T}^{\mathcal{B}}$, if $\mathcal{A}^{\mathcal{H}}|_{\Sigma^{\mathcal{B}}} \in \mathcal{C}^{\mathcal{B}}$. As usual, $\mathcal{A}^{\mathcal{H}}|_{\Sigma^{\mathcal{B}}}$ is obtained from a $\mathcal{A}^{\mathcal{H}}$-algebra by removing all carrier sets $S^{\mathcal{A}}$ for all $S \in (\mathcal{S}^{\mathcal{F}} \setminus \mathcal{S}^{\mathcal{B}})$, all functions from $\Omega^{\mathcal{F}}$ and all predicates from $\Pi^{\mathcal{F}}$. We write $\models_{\mathcal{H}}$ for the entailment relation with respect to hierarchic algebras and formulas from $\Sigma^{\mathcal{H}}$ and $\models_{\mathcal{B}}$ for the entailment relation with respect to the $\mathcal{C}^{\mathcal{B}}$ algebras and formulas from $\Sigma^{\mathcal{B}}$.

Terms, atoms, literals build over $\Sigma^{\mathcal{B}}$ are called *pure background terms*, *pure background atoms*, and *pure background literals*, respectively. All terms, atoms, with a top-symbol from $\Omega^{\mathcal{B}}$ or $\Pi^{\mathcal{B}}$, respectively, are called *background terms*, *background atoms*, respectively. A background atom or its negation is a *background literal*. All terms, atoms, with a top-symbol from $\Omega^{\mathcal{F}}$ or $\Pi^{\mathcal{F}}$, respectively, are called *foreground terms*, *foreground atoms*, respectively. A foreground atom or its negation is a *foreground literal*. Given a set (sequence) of $\mathcal{H}$ literals, the function bgd returns the set (sequence) of background literals and the function fgd the respective set (sequence) of foreground literals. A substitution σ is called *simple* if $x_S\sigma \in T_S(\Sigma^{\mathcal{B}}, \mathcal{X})$ for all $x_S \in \mathrm{dom}(\sigma)$ and $S \in \mathcal{S}^{\mathcal{B}}$.

As usual, clauses are disjunctions of literals with implicitly universally quantified variables. We often write a $\Sigma^{\mathcal{H}}$ clause as a *constrained clause*, denoted $\Lambda \parallel C$ where Λ is a conjunction of background literals and C is a disjunction of foreground literals semantically denoting the clause $\neg\Lambda \vee C$. A *constrained closure* is denoted as $\Lambda \parallel C \cdot \sigma$ where σ is grounding for Λ and C. A constrained closure $\Lambda \parallel C \cdot \sigma$ denotes the ground constrained clause $\Lambda\sigma \parallel C\sigma$.

In addition, we assume a well-founded, total, strict ordering $\prec$ on ground literals, called an $\mathcal{H}$-order, such that background literals are smaller than foreground literals. This ordering is then lifted to constrained clauses and sets thereof by its respective multiset extension. We overload $\prec$ for literals, constrained clauses, and sets of constrained clause if the meaning is clear from the context. We define $\preceq$ as the reflexive closure of $\prec$ and $N^{\preceq \Lambda \parallel C} := \{D \mid D \in N \text{ and } D \preceq \Lambda \parallel C\}$. An instance of an LPO with according precedence can serve as $\prec$.

Definition 1 (Clause Redundancy)**.** A ground constrained clause $\Lambda \parallel C$ is *redundant* with respect to a set N of ground constrained clauses and an order $\prec$ if $N^{\preceq \Lambda \parallel C} \models_{\mathcal{H}} \Lambda \parallel C$. A clause $\Lambda \parallel C$ is *redundant* with respect to a clause set N, an $\mathcal{H}$-order $\prec$, and a set of constants B if for all $\Lambda' \parallel C' \in \mathrm{gnd}_B(\Lambda \parallel C)$

the clause $\Lambda' \parallel C'$ is redundant with respect to $\cup_{D \in N} \mathrm{gnd}_B(D)$.

Example 2 (BS(LRA))**.** The running example in this paper is the Bernays-Schoenfinkel clause fragment over linear arithmetic: BS(LRA). The background theory is linear rational arithmetic over the many-sorted signature $\Sigma^{\mathrm{LRA}} = (\mathcal{S}^{\mathrm{LRA}}, \Omega^{\mathrm{LRA}}, \Pi^{\mathrm{LRA}})$ with $\mathcal{S}^{\mathrm{LRA}} = \{\mathrm{LRA}\}$, $\Omega^{\mathrm{LRA}} = \{0, 1, +, -\} \cup \mathbb{Q}$, $\Pi^{\mathrm{LRA}} = \{\leq, <, \neq, =, >, \geq\}$) where LRA is the linear arithmetic sort, the function symbols consist of $0, 1, +, -$ plus the rational numbers and predicate symbols $\leq, <, =, \neq, >, \geq$. The linear arithmetic theory $\mathcal{T}^{\mathrm{LRA}} = (\Sigma^{\mathrm{LRA}}, \{\mathcal{A}^{\mathrm{LRA}}\})$ consists of the linear arithmetic signature together with the standard model $\mathcal{A}^{\mathrm{LRA}}$ of linear arithmetic. This theory is then extended by the free (foreground) first-order signature $\Sigma^{\mathrm{BS}} = (\{\mathrm{LRA}\}, \Omega^{\mathrm{BS}}, \Pi^{\mathrm{BS}})$ where Ω^{BS} is a set of constants of sort LRA different from Ω^{LRA} constants, and Π^{BS} is a set of first-order predicates over the sort LRA. We are interested in hierarchic algebras $\mathcal{A}^{\mathrm{BS(LRA)}}$ over the signature $\Sigma^{\mathrm{BS(LRA)}} = (\{\mathrm{LRA}\}, \Omega^{\mathrm{BS}} \cup \Omega^{\mathrm{LRA}}, \Pi^{\mathrm{BS}} \cup \Pi^{\mathrm{LRA}})$ that are $\Sigma^{\mathrm{BS(LRA)}}$ algebras such that $\mathcal{A}^{\mathrm{BS(LRA)}}|_{\Sigma^{\mathrm{LRA}}} = \mathcal{A}^{\mathrm{LRA}}$.

Note that our definition of the BS(LRA) fragment restricted to the linear arithmetic sort does not restrict expressiveness compared to a definition adding further free sorts to Σ^{BS}. Free sorts containing only constants can be simulated by the linear arithmetic sort in a many-sorted setting. For example, assume a free sort S with constants $k_1, \ldots, k_n$, then this sort can be represented by the LRA sort with domain constants $1, \ldots, n$ where each occurrence of a k_i in a clause set is replaced by i and each occurrence of a variable u_S with a fresh variable x of sort LRA plus the additional constraint $P(x)$. The fresh predicate P is true exactly for the numbers $1, \ldots, n$. This is encoded by the first-order $\Sigma^{\mathrm{BS(LRA)}}$ formula $\forall x[P(x) \leftrightarrow (x = 1 \vee \ldots \vee x = n)]$. For example, a clause $y > 1 \parallel R(y, k_5, u_P)$ is encoded by the clause $y > 1 \parallel \neg P(x) \vee R(y, 5, x)$ preserving satisfiability. Please note that this encoding does not introduce any free constants of the LRA sort. The restriction to background sorts simplifies the presentation of the $SCL(T)$ calculus in Section 3 significantly.

We call a clause set N *abstracted* if the arguments of any predicate from $\Pi^{\mathcal{F}}$ are only variables. Abstraction can always be obtained by adding background constraints, e.g., the BS(LRA) clause $x > 1 \parallel R(x, 5)$ can be abstracted to $x > 1, y = 5 \parallel R(x, y)$, preserving satisfiability. Recall that even in the foreground signature we only consider background sorts and that the only operators in the foreground signature are constants.

A set N of $\mathcal{H}$ clauses is called *pure* if it does not contain symbols from $\Omega^{\mathcal{F}}$ ranging into a sort of $\mathcal{S}^{\mathcal{B}}$. In this case N is *sufficiently complete* according to [BGW94], hence hierarchic superposition is complete for N [BGW94, BW19]. As a consequence, a pure clause set N is unsatisfiable iff $\mathrm{gnd}_B(N)$ is unsatisfiable for a sufficiently large set B of constants. We will make use of this result in the completeness proof for our calculus, Theorem 25. A set N of $\mathcal{H}$ clauses is called *almost pure*, if the only symbols it contains from $\Omega^{\mathcal{F}}$ ranging into a sort of $\mathcal{S}^{\mathcal{B}}$ are constants. In general, there cannot be a complete calculus for almost pure clause sets, see Example 4. Satisfiability of pure clause sets is undecidable. We already mentioned in the introduction that this can be shown through a reduction to the halting problem for two-counter machines [Min67, HVW17]. Clause redundancy for pure clause sets cannot be decided as well.

Lemma 3 (Non-Redundancy for Pure Clause Sets is Undecidable)**.** For a pure

clause set N it is undecidable whether some clause C is non-redundant with respect to N.

Proof. The construction is similar to the construction taken in [Wei15]. Let $N = \{C_1, \ldots, C_n\}$ be an arbitrary pure clause set. Then N is satisfiable iff $N' = \{P \vee C_1, Q \vee C_2, C_3, \ldots, C_n, \neg P, \neg Q\}$ is satisfiable for two fresh propositional variables (predicates of arity zero) P, Q. Then N' is satisfiable iff $P \vee Q$ is non-redundant with respect to $N' \setminus \{\neg P, \neg Q\}$ where we assume that P, Q are maximal in the atom ordering $\prec$. □

Example 4 (Non-Compactness of Almost Pure BS(LRA) Clause Sets)**.** Note that for BS(LRA) clauses sets that are not pure, i.e., they contain Σ^{BS} constants, there cannot be a complete calculus, in general, because compactness [BGW94] is lost. For example, consider the five abstracted clauses

$$\mathrm{Nat}(0),\ y = x + 1 \parallel \neg\,\mathrm{Nat}(x) \vee \mathrm{Nat}(y),$$
$$x < 0 \parallel \neg\,\mathrm{Nat}(x),\ 0 < x < 1 \parallel \neg\,\mathrm{Nat}(x),\ x > 0, y = x + 1 \parallel \neg\,\mathrm{Nat}(y) \vee \mathrm{Nat}(x)$$

defining the natural numbers with respect to $\mathcal{T}^{\mathrm{LRA}}$ and a foreground predicate Nat. The four clauses

$$x = a \parallel \mathrm{Nat}(x),\ P(0),\ y = x + 1 \parallel \neg P(x) \vee P(y),\ x = a \parallel \neg P(x)$$

express that a is a natural number but not contained in P that itself contains all natural numbers. The clause set is unsatisfiable, but there is no proof by hierarchic superposition (resolution), because any finite set of ground clauses out of this clause set is satisfiable. If the semantics of BS(LRA) clause sets is not restricted to hierarchic algebras, there is a model for the clause set by adding some junk element to $\mathbb{Q}$ and assigning it to a.

3 SCL(T)

Assumptions: For this section we consider only pure, abstracted clause sets N. We assume that the background theory $\mathcal{T}^{\mathcal{B}}$ is term-generated, compact, contains an equality $=$, and that all constants of the background signature are domain constants. We further assume that the set $\Omega^{\mathcal{F}}$ contains infinitely many constants for each background sort.

Example 5 (Pure Clauses)**.** With respect to BS(LRA) the unit clause $x \geq 5, 3x + 4y = z \parallel Q(x, y, z)$ is abstracted and pure while the clause $x \geq 5, 3x + 4y = a, z = a \parallel Q(x, y, z)$ is abstracted but not pure because of the foreground constant a of the LRA sort, and the clause $x \geq 5, 3x + 4y = 7 \parallel Q(x, y, 7)$ is not abstracted.

Note that for pure, abstracted clause sets, any unifier between two foreground literals is simple and its codomain consists of variables only.

In order for the SCL(T) calculus to be effective, decidability in $\mathcal{T}^{\mathcal{B}}$ is needed as well. For the calculus we implicitly use the following equivalence: A $\Sigma^{\mathcal{B}}$ sentence $\exists x_1, \ldots, x_n \phi$ where ϕ is quantifier free is true, i.e., $\models_{\mathcal{B}} \exists x_1, \ldots, x_n \phi$ iff the ground formula $\phi\{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\}$ where the a_i are $\Omega^{\mathcal{F}}$ constants of the respective background sorts is $\mathcal{H}$ satisfiable. Together with decidability in $\mathcal{T}^{\mathcal{B}}$ this guarantees decidability of the satisfiability of ground constraints from constrained clauses.

If not stated otherwise, satisfiability means satisfiability with respect to $\mathcal{H}$. The function $\text{adiff}(B)$ for some finite sequence of background sort constants denotes a constraint that implies different interpretations for the constants in B. In case the background theory enables a strict ordering $<$ as LRA does, then the ordering can be used for this purpose. For example, $\text{adiff}([a, b, c])$ is then the constraint $a < b < c$. In case the background theory does not enable a strict ordering, then inequations can express disjointness of the constants. For example, $\text{adiff}([a, b, c])$ is then constraint $a \neq b \wedge a \neq c \wedge b \neq c$. An ordering constraint has the advantage over an inequality constraint that it also breaks symmetries. Assuming all constants to be different will eventually enable a satisfiability test for foreground literals based on purely syntactic complementarity.

The inference rules of SCL(T) are represented by an abstract rewrite system. They operate on a problem state, a six-tuple $\Gamma = (M; N; U; B; k; D)$ where M is a sequence of annotated ground literals, the *trail*; N and U are the sets of *initial* and *learned* constrained clauses; B is a finite sequence of constants of background sorts for instantiation; k counts the number of decisions in M; and D is a constrained closure that is either $\top$, $\Lambda \parallel \bot \cdot \sigma$, or $\Lambda \parallel C \cdot \sigma$. Foreground literals in M are either annotated with a number, a level; i.e. , they have the form L^k meaning that L is the k-th guessed decision literal, or they are annotated with a constrained closure that propagated the literal to become true, i.e. , they have the form $(L\sigma)^{(\Lambda \parallel C \vee L) \cdot \sigma}$. An annotated literal is called a decision literal if it is of the form L^k and a propagation literal or a propagated literal if it of in the form $L \cdot \sigma^{(\Lambda \parallel C \vee L) \cdot \sigma}$. A ground foreground literal L is of *level* i with respect to a problem state $(M; N; U; B; k; D)$ if L or $\text{comp}(L)$ occurs in M and the first decision literal left from L ($\text{comp}(L)$) in M, including L, is annotated with i. If there is no such decision literal then its level is zero. A ground constrained clause $\Lambda \parallel C$ is of *level* i with respect to a problem state $(M; N; U; B; k; D)$ if i is the maximal level of a foreground literal in C; the level of an empty clause $\Lambda \parallel \bot \cdot \sigma$ is 0. A ground literal L is *undefined* in M if neither L nor $\text{comp}(L)$ occur in M. The initial state for a first-order, pure, abstracted $\mathcal{H}$ clause set N is $(\epsilon; N; \emptyset; B; 0; \top)$, where B is a finite sequence of foreground constants of background sorts. These constants cannot occur in N, because N is pure. The final state $(\epsilon; N; U; B; 0; \Lambda \parallel \bot)$ denotes unsatisfiability of N. Given a trail M and its foreground literals $\text{fgd}(M) = \{L_1, \ldots, L_n\}$ an $\mathcal{H}$ ordering $\prec$ *induced* by M is any $\mathcal{H}$ ordering where $L_i \prec L_j$ if L_i occurs left from L_j in M, or, L_i is defined in M and L_j is not.

The transition rules for SCL(T) are

Propagate $(M; N; U; B; k; \top) \Rightarrow_{\text{SCL(T)}} (M, L\sigma^{(\Lambda \parallel C_0 \vee L)\delta \cdot \sigma}, \Lambda'\sigma; N; U; B; k; \top)$
provided $\Lambda \parallel C \in (N \cup U)$, σ is grounding for $\Lambda \parallel C$, $\text{adiff}(B) \wedge \text{bgd}(M) \wedge \Lambda\sigma$ is satisfiable, $C = C_0 \vee C_1 \vee L$, $C_1\sigma = L\sigma \vee \ldots \vee L\sigma$, $C_0\sigma$ does not contain $L\sigma$, δ is the mgu of the literals in C_1 and L, $\Lambda'\sigma$ are the background literals from $\Lambda\sigma$ that are not yet on the trail, $\text{fgd}(M) \models \neg(C_0\sigma)$, $\text{codom}(\sigma) \subseteq B$, and $L\sigma$ is undefined in M

The rule Propagate applies exhaustive factoring to the propagated literal with respect to the grounding substitution σ and annotates the factored clause to the propagation. By writing $M, L\sigma^{(\Lambda \parallel C_0 \vee L)\delta \cdot \sigma}, \Lambda'\sigma$ we denote that all background literals from $\Lambda'\sigma$ are added to the trail.

Decide $(M; N; U; B; k; \top) \Rightarrow_{\mathrm{SCL(T)}} (M, L\sigma^{k+1}, \Lambda\sigma; N; U; B; k+1; \top)$
provided $L\sigma$ is undefined in M, $|L\sigma| \in \mathrm{atoms}(\mathrm{gnd}_B(N \cup U))$, $|K\sigma| \in \mathrm{atoms}(\mathrm{gnd}_B(N \cup U))$ for all $K\sigma \in \Lambda\sigma$, σ is grounding for Λ, all background literals in $\Lambda\sigma$ are undefined in M, $\mathrm{adiff}(B) \wedge \mathrm{bgd}(M) \wedge \Lambda\sigma$ is satisfiable, and $\mathrm{codom}(\sigma) \subseteq B$

Making sure that no duplicates of background literals occur on the trail by rules Propagate and Decide together with a fixed finite sequence B of constants and the restriction of Propagate and Decide to undefined literals guarantees that the number of potential trails of a run is finite. Requiring the constants from B to be different by the $\mathrm{adiff}(B)$ constraint enables a purely syntactic consistency check for foreground literals.

Conflict $(M; N; U; B; k; \top) \Rightarrow_{\mathrm{SCL(T)}} (M; N; U; B; k; \Lambda \parallel D \cdot \sigma)$
provided $\Lambda \parallel D \in (N \cup U)$, σ is grounding for $\Lambda \parallel D$, $\mathrm{adiff}(B) \wedge \mathrm{bgd}(M) \wedge \Lambda\sigma$ is satisfiable, $\mathrm{fgd}(M) \models \neg(D\sigma)$, and $\mathrm{codom}(\sigma) \subseteq B$

Resolve $(M, L\rho^{\Lambda \parallel C \vee L \cdot \rho}; N; U; B; k; (\Lambda' \parallel D \vee L') \cdot \sigma) \Rightarrow_{\mathrm{SCL(T)}} (M, L\rho^{\Lambda \parallel C \vee L \cdot \rho}; N; U; B; k; (\Lambda \wedge \Lambda' \parallel D \vee C)\eta \cdot \sigma\rho)$
provided $L\rho = \mathrm{comp}(L'\sigma)$, and $\eta = \mathrm{mgu}(L, \mathrm{comp}(L'))$

Note that Resolve does not remove the literal $L\rho$ from the trail. This is needed if the clause $D\sigma$ contains further literals complementary of $L\rho$ that have not been factorized.

Factorize $(M; N; U; B; k; (\Lambda \parallel D \vee L \vee L') \cdot \sigma) \Rightarrow_{\mathrm{SCL(T)}} (M; N; U; B; k; (\Lambda \parallel D \vee L)\eta \cdot \sigma)$
provided $L\sigma = L'\sigma$, and $\eta = \mathrm{mgu}(L, L')$

Note that Factorize is not limited with respect to the trail. It may apply to any two literals that become identical by application of the grounding substitution σ.

Skip $(M, L; N; U; B; k; \Lambda' \parallel D \cdot \sigma) \Rightarrow_{\mathrm{SCL(T)}} (M; N; U; B; l; \Lambda' \parallel D \cdot \sigma)$
provided L is a foreground literal and $\mathrm{comp}(L)$ does not occur in $D\sigma$, or L is a background literal; if L is a foreground decision literal then $l = k - 1$, otherwise $l = k$

Note that Skip can also skip decision literals. This is needed because we won't eventually require exhaustive propagation. While exhaustive propagation in CDCL is limited to the number of propositional variables, in the context of our logic, for example BS(LRA), it is exponential in the arity of foreground predicate symbols and can lead to an unfair exploration of the space of possible inferences, harming completeness, see Example 8.

Backtrack $(M, K^{i+1}, M'; N; U; B; k; (\Lambda \parallel D \vee L) \cdot \sigma) \Rightarrow_{\mathrm{SCL(T)}} (M, L\sigma^{(\Lambda \parallel D \vee L) \cdot \sigma}, \Lambda'\sigma; N; U \cup \{\Lambda \parallel D \vee L\}; B; i; \top)$
provided $L\sigma$ is of level k, and $D\sigma$ is of level i, $\Lambda'\sigma$ are the background literals from $\Lambda\sigma$ that are not yet on the trail

The definition of Backtrack requires that if $L\sigma$ is the only literal of level k in $(D \vee L)\sigma$ then additional occurrences of $L\sigma$ in D have to be factorized first before Backtrack can be applied.

Grow $(M; N; U; B; k; \top) \Rightarrow_{\mathrm{SCL(T)}} (\epsilon; N; U; B \cup B'; 0; \top)$
provided B' is a non-empty sequence of foreground constants of background sorts distinct from the constants in B

In case the adiff constraint is implemented by a strict ordering predicate on the basis of the sequence B, it can be useful to inject the new constants B' into $B \cup B'$ such that the ordering of the constants from B is not changed. This can help caching background theory results for testing trail satisfiability.

Definition 6. The rules Propagate, Decide, Grow, and Conflict are called *conflict search* rules and the rules Resolve, Skip, Factorize, and Backtrack are called *conflict resolution* rules.

Recall that the goal of our calculus is to replace the ordering restrictions of the hierarchic superposition calculus with a guiding model assumption. All our inferences are hierarchic superposition inferences where the ordering restrictions are neglected.

Example 7 (Inconsistent Trail)**.** Consider a clause set $N = \{R(x, y), x \leq y \parallel \neg R(x, y) \vee P(x), x \geq y \parallel \neg R(x, y) \vee \neg P(y)\}$; if we were to remove the adiff(B) constraint from the side conditions of rule Propagate we would be able to obtain inconsistent trails. Starting with just $B = \{a, b\}$ as constants it is possible to propagate three times and obtain the trail $M = [R(a, b), P(a), a \leq b, \neg P(b), a \geq b]$, M is clearly inconsistent as $M \models P(a)$, $M \models \neg P(b)$ yet $a = b$.

Example 8 (Exhaustive Propagation)**.** Consider a BS(LRA) clause set $N = \{x = 0 \parallel \mathrm{Nat}(x),\ y = x + 1 \parallel \neg \mathrm{Nat}(x) \vee \mathrm{Nat}(y)\} \cup N'$ where N' is unsatisfiable and nothing can be propagated from N'. Let us further assume that N' is satisfiable with respect to any instantiation of variables with natural numbers. If propagation is not restricted, then the first two clauses will consume all constants in B. For example, if $B = [a, b, c]$ then the trail $[\mathrm{Nat}(a), a = 0, \mathrm{Nat}(b), b = a + 1, \mathrm{Nat}(c), c = b + 1]$ will be derived. Now all constants are fixed to natural numbers. So there cannot be a refutation of N' anymore. An application of Grow will not solve the issue, because again the first two rules will fix all constants to natural numbers via exhaustive propagation.

Definition 9 (Well-formed States)**.** A state $(M; N; U; B; k; D)$ is *well-formed* if the following conditions hold:

1. all constants appearing in $(M; N; U; B; k; D)$ are from B or occur in N.
2. $M \wedge \mathrm{adiff}(B)$ is satisfiable
3. $N \models_{\mathcal{H}} U$,
4. Propagating clauses remain propagating and conflict clauses remain false:
   (a) if $D = \Lambda \parallel C \cdot \sigma$ then $C\sigma$ is false in $\mathrm{fgd}(M)$ and $\mathrm{bgd}(M) \wedge \mathrm{adiff}(B) \wedge \Lambda\sigma$ is satisfiable,

(b) if $M = M_1, L\sigma^{(\Lambda \| C \vee L)\cdot\sigma}, M_2$ then $C\sigma$ is false in $\mathrm{fgd}(M_1)$, $L\sigma$ is undefined in M_1, and $\mathrm{bgd}(M_1) \wedge \mathrm{adiff}(B) \wedge \Lambda\sigma$ is satisfiable.

5. All clauses in $N \cup U$ are pure. In particular, they don't contain any constants from B.

Lemma 10 (Rules preserve Well-Formed States)**.** The rules of SCL(T) preserve well-formed states.

Proof. We prove each of the five properties by induction on the length of a derivation starting from the initial state $(\epsilon; N; \emptyset; B; k; \top)$. The induction step for the first two claims is

$$(M; N; U; B; k; D) \Rightarrow_{\mathrm{SCL(T)}} (M'; N'; U'; B'; k'; D').$$

1. In the initial state $(\epsilon; N; \emptyset; B; k; \top)$ constants can only appear in N and B, so it satisfies the claim. For the inductive step we do a case analysis on the rule application and prove $\mathrm{con}((M'; N'; U'; B'; k'; D')) \subseteq \mathrm{con}(N') \cup B'$ by case analysis on the rules of SCL(T). If we have applied Propagate or Decide then $N' = N$, $U' = U$, $B' = B$, $D' = D = \top$ and $M' = M, L\sigma$. So we only need to prove that $\mathrm{con}(M') \subseteq \mathrm{con}(N) \cup B$. Both rules require $|L\sigma| \in \mathrm{atoms}(\mathrm{gnd}_B(N \cup U))$ satisfying the claim.

In case of the rules Grow, Skip, or Backtrack, then $\mathrm{con}(N) \cup B \subseteq \mathrm{con}(N') \cup B'$ and $\mathrm{con}(M') \cup \mathrm{con}(U') \cup \mathrm{con}(D') \subseteq \mathrm{con}(M) \cup \mathrm{con}(U) \cup \mathrm{con}(D)$.

If the rule Conflict was used then only the last component of the state changed $D' = \Lambda \parallel C \cdot \sigma$ with $\mathrm{codom}(\sigma) \subseteq B$ and $\mathrm{con}(\Lambda \parallel C) \subseteq \mathrm{con}(N) \cup B$ by induction hypothesis.

If one of the rules Resolve or Factorize were used then as for Conflict the only component of the state that changed was the conflict clause and the constants in D' are a subset of the constants in M and D.

2. In the initial state $(\epsilon; N; \emptyset; B; k; \top)$ the condition $\mathrm{adiff}(B)$ is satisfied as we assume constants in B to be distinct. For the inductive step we do a case analysis on the rule application. If the rule used was one of Conflict, Backtrack, Skip, Resolve, or Factorize, then $M = M', M''$ with M'' possibly empty and $B = B'$, so that $M \wedge \mathrm{adiff}(B)$ satisfiable implies $M' \wedge \mathrm{adiff}(B')$ to be satisfiable. If the rule Grow was used then we have $M' = \epsilon$ and that all constants in $B' = B \oplus B''$ are distinct, so that satisfiability of $\mathrm{adiff}(B')$ is immediate. If the rule used was Propagate or Decide then we have $M' = M, L\sigma, \Lambda\sigma$ and $B' = B$, from the preconditions on the rules we also know that $L\sigma$ is undefined in M and that $\mathrm{bgd}(M') \wedge \mathrm{adiff}(B')$ is satisfiable.

3. By induction on the number of learned clauses. We prove that for each application of Backtrack

$$\begin{aligned}&(M, K^{i+1}, M'; N; U; B; k; D \vee L \cdot \sigma)\\ &\quad \Rightarrow^{\text{Backtrack}}_{\text{SCL(T)}} (M, L\sigma^{(\Lambda \| D \vee L)\cdot\sigma}, \Lambda'\sigma; N; U \cup \{\Lambda \parallel D \vee L\}; B; i; \top)\end{aligned}$$

we have $N \cup U \models_{\mathcal{H}} D \vee L$. Following conflict resolution backward we can find a sequence of constrained closures $C_1 \cdot \sigma_1, \ldots, C_n \cdot \sigma_n$, where $C_n \cdot \sigma_n = D \vee L \cdot \sigma$ such that $C_1 \in (N \cup U)$ is the most recent conflict clause, and C_{j+1} is either the result of a factorization on C_j or the result of a resolution inference between C_j

and a clause in $(N \cup U)$. By induction on the length of conflict resolution and soundness of resolution and factoring we get $N \cup U \models_{\mathcal{H}} D \vee L$.

4. For the initial state the properties 4a and 4b obviously hold. For the induction step and an application of the rules Decide, Skip, and Grow there is nothing to show.

Consider a state $(M; N; U; B; k; \Delta \parallel D \cdot \delta)$ obtained by an application of Conflict. By the side conditions of Conflict $\mathrm{adiff}(B) \wedge \mathrm{bgd}(M) \wedge \Delta\delta$ is satisfiable and $\mathrm{fgd}(M) \models \neg(D\delta)$ is shown for 4a. There is nothing to show for 4b.

Consider an application of rule Resolve

$$\begin{aligned}&(M, L\rho^{\Lambda \parallel (C \vee L) \cdot \rho}; N; U; B; k; \Lambda' \parallel (D \vee L') \cdot \sigma)\\ &\quad \Rightarrow^{\text{Resolve}}_{\text{SCL(T)}} (M, L\rho^{\Lambda \parallel (C \vee L) \cdot \rho}; N; U; B; k; \Lambda \wedge \Lambda' \parallel (D \vee C)\eta \cdot \rho\sigma)\end{aligned}$$

by induction hypothesis $\mathrm{bgd}(M) \wedge \mathrm{adiff}(B) \wedge \Lambda\eta\rho\sigma$ is satisfiable and $C\eta\rho\sigma$ is false in $\mathrm{fgd}(M)$, because $\Lambda\eta\rho\sigma = \Lambda\rho$ and $C\eta\rho\sigma = C\rho$ because η is the mgu and we always assume clauses to be variable disjoint. Using the same argument $\mathrm{bgd}(M, L\sigma) \wedge \mathrm{adiff}(B) \wedge \Lambda'\eta\delta\sigma$ is satisfiable and $(D \vee L')\eta\delta\sigma)$ is false in $\mathrm{fgd}(M, L\sigma)$. Therefore $(D \vee C)\eta \cdot \rho\sigma$ is false in $\mathrm{fgd}(M, L\sigma)$ and $\mathrm{bgd}(M, L\sigma) \wedge \mathrm{adiff}(B) \wedge (\Lambda' \wedge \Lambda)\eta\delta\sigma$ is satisfiable, proving 4a. There is nothing to show for 4b.

For an application of the rule Factorize there is nothing to show for 4b and 4a obviously holds because the set of different ground literals in $(D \vee L \vee L')\sigma$ and $(D \vee L)\sigma$ is identical.

For an application of the rule Propagate there is nothing to show for 4a. For 4b consider the step

$$\begin{aligned}&(M; N; U; B; k; \top)\\ &\quad \Rightarrow^{\text{Propagate}}_{\text{SCL(T)}} (M, L\sigma^{(\Lambda \parallel C_0 \vee L) \delta \cdot \sigma}, \Lambda'\sigma; N; U; B; k; \top)\end{aligned}$$

where the side conditions of the rule imply the claim modulo the removal of duplicate literals $L\sigma$.

Finally, when applying Backtrack

$$\begin{aligned}&(M, K^{i+1}, M'; N; U; B; k; (\Lambda \parallel D \vee L) \cdot \sigma)\\ &\quad \Rightarrow^{\text{Backtrack}}_{\text{SCL(T)}} (M, L\sigma^{(\Lambda \parallel D \vee L) \cdot \sigma}, \Lambda'\sigma; N; U \cup \{\Lambda \parallel D \vee L\}; B; i; \top)\end{aligned}$$

there is nothing to show for 4a. For 4b we know by induction hypothesis that $(D \vee L)\sigma$ is false in $\mathrm{fgd}(M, K^{i+1}, M')$. The literal $L\sigma$ is of level k and $D\sigma$ of level i, $k > i$, hence $D\sigma$ is false in $\mathrm{fgd}(M)$ and $L\sigma$ undefined in $\mathrm{fgd}(M)$. Furthermore, $\mathrm{bgd}(M, K^{i+1}, M') \wedge \mathrm{adiff}(B) \wedge \Lambda\sigma$ is satisfiable by induction hypothesis, so $\mathrm{bgd}(M) \wedge \mathrm{adiff}(B) \wedge \Lambda\sigma$ is satisfiable as well.

4. For the initial state all clauses are pure by assumption. Conflict picks a clause from $N \cup U$ that is pure by induction hypothesis. Resolve and Factorize only apply unifiers between pure literals to the resulting clause, hence also only produce pure clauses from pure clauses. Finally, Backtrack adds the pure learned clause to $N \cup U$. □

Definition 11 (Stuck State)**.** A state $(M; N; U; B; k; D)$ is called *stuck* if $D \neq \Lambda \parallel \bot \cdot \sigma$ and none of the rules Propagate, Decide, Conflict, Resolve, Factorize, Skip, or Backtrack is applicable.

Proposition 12 (Form of Stuck States)**.** If a run (without rule Grow) ends in a stuck state $(M; N; U; B; k; D)$ where Conflict was applied eagerly, then $D = \top$ and all ground foreground literals that can be build from the foreground literals in N by instantiation with constants from B are defined in M.

Proof. First we prove that stuck states never appear during conflict resolution. Consider a well-formed state $(M; N; U; B; k; \Delta \parallel D{\cdot}\delta)$, we prove by case analysis that either Skip, Resolve, Factorize or Backtrack can be applied. If $M = M', L\sigma$ and $L\sigma$ is either a background literal or a foreground literal such that $\mathrm{comp}(L\sigma)$ is not contained in $D\delta$ then Skip can be applied. If $M = M', L\sigma^{\Lambda \parallel C\cdot\sigma}$ with $D\delta = D' \vee \mathrm{comp}(L\sigma)$ then Resolve can be applied. If $M = M', L\sigma^k, M''$ and D' contains multiple occurrences of $\mathrm{comp}(L\sigma)$ then Factorize can be applied. In summary, we can reach a state with a unique literal $L\delta$ of level k in $D\delta$. Then Backtrack is applicable. Finally, if in some state $(M; N; U; B; k; \top)$ where Conflict is not applicable, some atom $|L| \in \mathrm{atoms}(\mathrm{gnd}_B(N))$ is undefined, we can always apply Decide. □

Lemma 13 (Stuck States Produce Ground Models)**.** If a state $(M; N; U; B; k; \top)$ is stuck then $M \wedge \mathrm{adiff}(B) \models \mathrm{gnd}_B(N \cup U)$.

Proof. By contradiction. Note that $M \wedge \mathrm{adiff}(B)$ is satisfiable, Lemma 10.2. Consider any clause $(\Lambda \parallel C)\sigma \in \mathrm{gnd}_B(N \cup U)$. It can only be not true in $M \wedge \mathrm{adiff}(B)$ if $\mathrm{fgd}(M) \models \neg(C\sigma)$ and $\mathrm{bgd}(M) \wedge \mathrm{adiff}(B) \wedge \Lambda\sigma$ is satisfiable. But then Conflict would be applicable, a contradiction. □

Example 14 (SCL(T) Model Extraction)**.** In some cases it is possible to extract an overall model from the ground trail of a stuck state of an SCL(T) derivation. Consider $B = [a, b, c]$ and a satisfiable BS(LRA) constrained clause set $N = \{x \geq 1 \parallel P(x), x < 0 \parallel P(x), 0 \leq x \wedge x < 1 \parallel \neg P(x), 2x \geq 1 \parallel P(x) \vee Q(x)\}$. Starting from state $(\epsilon; N; \emptyset; B; 0; \top)$ and applying Propagate fairly a regular run can derive the following trail

$$\begin{aligned} M = \quad & P(a)^{x\geq 1 \parallel P(x)\cdot\{x\mapsto a\}}, a \geq 1, P(b)^{x<0 \parallel P(x)\cdot\{x\mapsto b\}}, b < 0, \\ & \neg P(c)^{0\leq x\wedge x<1 \parallel \neg P(x)\cdot\{x\mapsto c\}}, 0 \leq c, c < 1, Q(c)^{2x\geq 1 \parallel P\vee Q(x)\cdot\{x\mapsto c\}}, 2c \geq 1 \end{aligned}$$

The state $(M; N; \emptyset; B; 0; \top)$ is stuck and $M \models_{\mathcal{H}} \mathrm{gnd}_B(N)$. Moreover from M we can generate an interpretation $\mathcal{A}^{\mathrm{BS(LRA)}}$ of N by generalizing the foreground constants used for instantiation and interpreting the predicates P and Q as formulas over $\Sigma^{\mathcal{B}}$, $P^{\mathcal{A}} = \{q \in \mathbb{Q} \mid q < 0 \vee q \geq 1\}$ and $Q^{\mathcal{A}} = \{q \in \mathbb{Q} \mid 2q \geq 1 \wedge q < 1\}$.

Lemma 15 (Soundness)**.** If a derivation reaches the state $(M; N; U; B; k; \Lambda \parallel \bot \cdot \sigma)$, then N is unsatisfiable.

Proof. All learned clauses are consequences of $N \cup U$, Lemma 10.3. Furthermore $\mathrm{bgd}(M) \wedge \mathrm{adiff}(B) \wedge \Lambda\sigma$ is satisfiable, Lemma 10.4a. □

Definition 16 (Reasonable Run)**.** A sequence of SCL(T) rule applications is called a *reasonable run* if the rule Decide is only applied if there exists no application of the rule Propagate that would generate a conflict.

Definition 17 (Regular Run)**.** A sequence of SCL(T) rule applications is called a *regular run* if it is a reasonable run the rule Conflict has precedence over all other rules, and Resolve resolves away at least the rightmost foreground literal from the trail.

Example 18 (SCL(T) Refutation)**.** Given a set of foreground constants $B = [a, b, c]$ and a BS(LRA) constrained clause set $N = \{C_1 \colon x = 0 \parallel P(x), C_2 \colon y = x + 1 \parallel \neg P(x) \vee P(y), C_3 \colon z = 2 \parallel \neg P(z)\}$ the following is a regular derivation

$$
\begin{aligned}
& (\epsilon; N; \emptyset; B; 0; \top) \\
\Rightarrow^{\text{Propagate}}_{\text{SCL(T)}} \quad & (P(a)^{C_1 \cdot \{x \mapsto a\}}, a = 0; N; \emptyset; B; 0; \top) \\
\Rightarrow^{\text{Propagate}}_{\text{SCL(T)}} \quad & (\ldots, P(b)^{C_2 \cdot \{x \mapsto a, y \mapsto b\}}, b = a + 1; N; \emptyset; B; 0; \top) \\
\Rightarrow^{\text{Propagate}}_{\text{SCL(T)}} \quad & (\ldots, P(c)^{C_2 \cdot \{x \mapsto b, y \mapsto c\}}, c = b + 1; N; \emptyset; B; 0; \top) \\
\Rightarrow^{\text{Conflict}}_{\text{SCL(T)}} \quad & (\ldots, P(c)^{C_2 \cdot \{x \mapsto b, y \mapsto c\}}, c = b + 1; N; \emptyset; B; 0; z = 2 \parallel \neg P(z) \cdot \{z \mapsto c\}) \\
\Rightarrow^{\text{Resolve}}_{\text{SCL(T)}} \quad & (\ldots, P(c)^{C_2 \cdot \{x \mapsto b, y \mapsto c\}}, c = b + 1; N; \emptyset; B; 0; \\
& z = x + 1 \wedge z = 2 \parallel \neg P(x) \cdot \{z \mapsto c, x \mapsto b\}) \\
\Rightarrow^{\text{Skip}}_{\text{SCL(T)}} \quad & (\ldots, P(b)^{C_2 \cdot \{x \mapsto a, y \mapsto b\}}, b = a + 1; N; \emptyset; B; 0; \\
& z = x + 1 \wedge z = 2 \parallel \neg P(x) \cdot \{z \mapsto c, x \mapsto b\}) \\
\Rightarrow^{\text{Resolve}}_{\text{SCL(T)}} \quad & (\ldots, P(b)^{C_2 \cdot \{x \mapsto a, y \mapsto b\}}, b = a + 1; N; \emptyset; B; 0; \\
& z = x + 1 \wedge z = 2 \wedge x = x_1 + 1 \parallel \neg P(x_1) \cdot \{z \mapsto c, x \mapsto b, x_1 \mapsto a\}) \\
\Rightarrow^{\text{Skip}}_{\text{SCL(T)}} \quad & (P(a)^{C_1 \cdot \{x \mapsto a\}}, a = 0; N; \emptyset; B; 0; \\
& z = x + 1 \wedge z = 2 \wedge x = x_1 + 1 \parallel \neg P(x_1) \cdot \{z \mapsto c, x \mapsto b, x_1 \mapsto a\}) \\
\Rightarrow^{\text{Resolve}}_{\text{SCL(T)}} \quad & (P(a)^{C_1 \cdot \{x \mapsto a\}}, a = 0; N; \emptyset; B; 0; \\
& z = x + 1 \wedge z = 2 \wedge x = x_1 + 1 \wedge x_1 = 0 \parallel \bot \cdot \{z \mapsto c, x \mapsto b, x_1 \mapsto a\})
\end{aligned}
$$

N is proven unsatisfiable as we reach a state in the form $(M; N; U; B; k; \Lambda \parallel \bot \cdot \sigma)$.

Example 19 (SCL(T) Clause learning)**.** Given an initial constant set $B = [a]$ of fresh foreground constants and a BS(LRA) constrained clause set $N = \{C_1 \colon x \geq y \parallel \neg P(x, y) \vee Q(z), C_2 \colon z = u + v \parallel \neg P(u, v) \vee \neg Q(z), \}$ the following is an example of a regular run

$$
\begin{aligned}
& (\epsilon; N; \emptyset; B; 0; \top) \\
\Rightarrow^{\mathit{Decide}}_{\text{SCL(T)}} \quad & (P(a, b)^1; N; \emptyset; B; 1; \top) \\
\Rightarrow^{\mathit{Propagate}}_{\text{SCL(T)}} \quad & (P(a, a)^1, Q(a)^{C_1 \cdot \{x \mapsto a, y \mapsto a, z \mapsto a\}}, a \geq a; N; \emptyset; B; 1; \top) \\
\Rightarrow^{\mathit{Conflict}}_{\text{SCL(T)}} \quad & (P(a, a)^1, Q(a)^{C_1 \cdot \{u \mapsto a, v \mapsto a, z \mapsto a\}}, a \geq a; N; \emptyset; B; 1; \\
& C_2 \cdot \{x \mapsto a, y \mapsto a, z \mapsto a\}) \\
\Rightarrow^{\mathit{Resolve}}_{\text{SCL(T)}} \quad & (P(a, a)^1, Q(a)^{C_1 \cdot \{x \mapsto a, y \mapsto a, z \mapsto a\}}, a \geq a; N; \emptyset; B; 1; x \geq y \wedge z = u + v \parallel \\
& \neg P(x, y) \vee \neg P(u, v) \cdot \{x \mapsto a, y \mapsto a, z \mapsto a, u \mapsto a, v \mapsto a\}) \\
\Rightarrow^{\mathit{Skip}*}_{\text{SCL(T)}} \quad & (P(a, a)^1; N; \emptyset; B; 1; x \geq y \wedge z = u + v \parallel \\
& \neg P(x, y) \vee \neg P(u, v) \cdot \{x \mapsto a, y \mapsto a, z \mapsto a, u \mapsto a, v \mapsto a\}) \\
\Rightarrow^{\mathit{Factorize}}_{\text{SCL(T)}} \quad & (P(a, a)^1; N; \emptyset; B; 1; x \geq y \wedge z = x + y \parallel \neg P(x, y) \cdot \{x \mapsto a, y \mapsto a, z \mapsto a\}) \\
\Rightarrow^{\mathit{Backtrack}}_{\text{SCL(T)}} \quad & (\neg P(a, a)^{(x \geq y \wedge z = x + y \parallel \neg P(x, y)) \cdot \{x \mapsto a, y \mapsto a\}}, a \geq a, a = a + a; N; \\
& \{x \geq y \wedge z = x + y \parallel \neg P(x, y)\}; B; 1; \top)
\end{aligned}
$$

In this example the learned clauses $x \geq y \wedge z = x + y \parallel \neg P(x, y)$; note how there are two distinct variables in the learned clause even if we had to use a single constant for instantiations in conflict search.

Proposition 20. Let N be a set of constrained clauses. Then any application of Decide in an SCL(T) regular run from starting state $(\epsilon; N; \emptyset; B; 0; \top)$ does not create a conflict.

Proof. Assume the contrary: then Propagate would have been applicable before Decide, contradicting with the definition of a regular and hence reasonable run. □

Corollary 21. Let N be a set of constrained clauses. Then any conflict in an SCL(T) regular run from starting state $(\epsilon; N; \emptyset; B; 0; \top)$ admits a regular conflict resolution.

Proof. We need to prove that it is possible to apply Resolve during conflict resolution. By Proposition 20 the rightmost foreground literal on the trail is a propagation literal and by regularity we know that this literal appears in the conflict clause. So a conflict resolution can start by skipping over the background literals and then resolving once with the rightmost foreground literal. □

Lemma 22 (Non-Redundant Clause Learning)**.** Let N be a set of constrained clauses. Then clauses learned in an SCL(T) regular run from starting state $(\epsilon; N; \emptyset; B; 0; \top)$ are not redundant.

Proof. Consider the following fragment of a derivation learning a clause:

$$\begin{aligned}
&\Rightarrow^{\text{Conflict}}_{\text{SCL(T)}} && (M''; N; U; B; k; \Lambda_0 \parallel C_0 \cdot \sigma_0)\\
&\Rightarrow^{\{\text{Skip, Factorize, Resolve}\}^*}_{\text{SCL(T)}} && (M, K^{i+1}, M'; N; U; B; k; \Lambda_n \parallel C_n \cdot \sigma_n)\\
&\Rightarrow^{\text{Backtrack}}_{\text{SCL(T)}} && (M, L\sigma^{(\Lambda_n \parallel D \vee L)\cdot\sigma}, \Lambda'_n\sigma; N; U \cup \{\Lambda_n \parallel D \vee L\}; B; i; \top).
\end{aligned}$$

where $C_n = D \vee L$ and $\sigma = \sigma_n$. Let $\prec$ be any $\mathcal{H}$ order induced by M. We prove that $\Lambda_n\sigma \parallel C_n\sigma$ is not redundant with respect to $\prec$, B, and $(N \cup U)$. By soundness of hierarchic resolution $(N \cup U) \models \Lambda_n \parallel C_n$ and $\Lambda_n\sigma$ is satisfiable with $M \wedge \text{adiff}(B)$, and $C_n\sigma$ is false under both M and M, K^{i+1}, M', Lemma 10. For a proof by contradiction, assume there is a $N' \subseteq \text{gnd}_B(N \cup U)^{\preceq \Lambda_n\sigma \parallel C_n\sigma}$ such that $N' \models_{\mathcal{H}} \Lambda_n\sigma \parallel C_n\sigma$. As $\Lambda_n\sigma \parallel C_n\sigma$ is false under M, there is a ground constrained clause $\Lambda' \parallel C' \in N'$ with $\Lambda' \parallel C' \preceq \Lambda_n\sigma \parallel C_n\sigma$, and all literals from C' are defined in M and false by the definition of $\prec$. Furthermore, we can assume that $\text{adiff}(B) \wedge \text{bgd}(M) \wedge \Lambda'$ is satisfiable or $C_n\sigma$ would be a tautology, because $\text{adiff}(B) \wedge \text{bgd}(M) \wedge \Lambda_n\sigma$ is satisfiable.

The clause $\Lambda_0\sigma_0 \parallel C_0\sigma_0$ has at least one literal of level k and due to a regular run, Definition 17, the rightmost trail literal is resolved away in $\Lambda_n\sigma \parallel C_n\sigma$, Corollary 21. Therefore, the rightmost foreground literal does not appear in $\Lambda' \parallel C'$, so by regularity $\Lambda' \parallel C'$ would have created a conflict at a previous state. □

Of course, in a regular run the ordering of foreground literals on the trail will change, i.e., the ordering underlying Lemma 22 will change as well. Thus the non-redundancy property of Lemma 22 reflects the situation at the time of creation of the learned clause. A non-redundancy property holding for an overall run must be invariant against changes on the ordering. However, the ordering underlying Lemma 22 also entails a fixed subset ordering that is invariant against changes on the overall ordering. This means that our dynamic ordering entails non-redundancy criteria based on subset relations including forward redundancy. From an implementation perspective, this means that learned clauses

need not to be tested for forward redundancy. Current resolution, or superposition based provers spent a reasonable portion of their time in testing forward redundancy of newly generated clauses. In addition, also tests for backward reduction can be restricted knowing that learned clauses are not redundant.

Lemma 23 (Termination of SCL(T))**.** Let N be a set of constrained clauses and B be a finite set of background constants. Then any regular run with start state $(\epsilon; N; \emptyset; B; 0; \top)$ that uses Grow only finitely often terminates.

Proof. Since Grow can only be used a finite number of times we consider as a start state the state after the final application of Grow and prove termination of runs that never use Grow. We do so by giving an explicit termination measure on the SCL(T) states. Given a state $(M; N; U; B; k; D)$ we define a termination measure μ as $\mu(M; N; U; B; k; D) = (u, s, m, r, d) \in \mathbb{N}^5$ with a lexicographical combination of $>$ where

- $l = |\operatorname{atoms}(\operatorname{gnd}_B(N \cup U))|$, $u = 3^l - |\operatorname{gnd}_B(U)|$, and $m = |M|$,
- in the case $D = \top$:
  - $s = 1 + l - m$, $d = 0$, and $r = 0$,
- otherwise if $D = \Delta \parallel D' \cdot \delta$:
  - $s = 0$,
  - if $M = M', L$ with L foreground literal then r is the number of copies of L in $D'\delta$
  - if the rightmost literal of M is a background literal or if M is empty then $r = 0$
  - d is the number of literals in D'

The number of ground atoms $l = |\operatorname{atoms}(\operatorname{gnd}_B(N \cup U))|$ is an upper bound to the length of the trail because the trail is consistent and no literal can appear more than once on the trail. Similarly, every learned clause has at least one non-redundant ground instance so $|\operatorname{gnd}_B(U)|$ increases whenever SCL(T) learns a new clause and 3^l is an upper bound to the ground instances of all learned clauses in a regular run. This means that Backtrack strictly decreases u, Decide, Propagate, and Conflict strictly decrease s without modifying u, Skip strictly decreases m without modifying u or s, Resolve strictly decreases r without modifying u, s, or m, and finally Factorize strictly decreases d possibly decreases r and does not modify u, s, or m. □

Theorem 24 (Hierarchic Herbrand Theorem)**.** Let N be a finite set of clauses. N is unsatisfiable iff there exists a finite set $N' = \{\Lambda_1 \parallel C_1, \ldots, \Lambda_n \parallel C_n\}$ of variable renamed copies of clauses from N and a finite set B of fresh constants and a substitution σ, grounding for N' where $\operatorname{codom}(\sigma) = B$ such that $\bigwedge_i \Lambda_i\sigma$ is $\mathcal{T}^{\mathcal{B}}$ satisfiable and $\bigwedge_i C_i\sigma$ is first-order unsatisfiable over $\Sigma^{\mathcal{F}}$.

Proof. Recall that N is a pure, abstracted clause set and that $\mathcal{T}^{\mathcal{B}}$ is term-generated, compact background theory that contains an equality $=$, and that all constants of the background signature are domain constants. Then by completeness of hierarchic superposition [BGW94], N is unsatisfiable iff there exists

a refutation by hierarchic superposition. Let $N' = \{\Lambda_1 \parallel C_1, \ldots, \Lambda_n \parallel C_n\}$ be a finite set renamed copies of clauses from N such that there is a refutation by hierarchic superposition such that each clause in N' and each derived clause is used exactly once. This set exists because the refutation is finite and any hierarchic superposition refutation can be transformed into a refutation where every clause is used exactly once. Now let δ be the overall unifier of this refutation. This unifier exists, because all clauses in N' have disjoint variables and all clauses in the refutation are used exactly once. Now we consider a finite set of constants B and a substitution σ, $\mathrm{codom}(\sigma) = B$, σ grounding for N', and for all $x, y \in \mathrm{dom}(\delta)$ we have $x\sigma = y\sigma$ iff $x\delta = y\delta$. Now there is also a refutation for $N'\sigma$ by hierarchic superposition where the clauses are inferred exactly in the way they were inferred for N'. It remains to be shown that $\bigwedge_i \Lambda_i\sigma$ is $\mathcal{T}^{\mathcal{B}}$ satisfiable and $\bigwedge_i C_i\sigma$ is $\mathcal{A}^{\mathcal{H}}$ unsatisfiable. The hierarchic superposition refutation terminates with the clause $\bigwedge_i \Lambda_i\sigma \parallel \bot$ where $\bigwedge_i \Lambda_i\sigma$ is satisfiable. Furthermore, the refutation derives $\bot$ from $\{C_1\sigma, \ldots, C_n\sigma\}$ via superposition, proving the theorem. □

Finally, we show that an unsatisfiable clause set can be refuted by SCL(T) with any regular run if we start with a sufficiently large sequence of constants B and apply Decide in a fair way. In addition, we need a Restart rule to recover from a stuck state.

Restart $\quad (M; N; U; B; k; \top) \Rightarrow_{\mathrm{SCL(T)}} (\epsilon; N; U; B; 0; \top)$

Of course, an unrestricted use of rule Restart immediately leads to non-termination.

Theorem 25 (Refutational Completeness of SCL(T))**.** Let N be an unsatisfiable clause set. Then any regular SCL(T) run will derive the empty clause provided (i) Rule Grow and Decide are operated in a fair way, such that all possible trail prefixes of all considered sets B during the run are eventually explored, and (ii) Restart is only applied to stuck states.

Proof. If N is unsatisfiable then by Theorem 24 there exists a a finite set $N' = \{\Lambda_1 \parallel C_1, \ldots, \Lambda_n \parallel C_n\}$ of variable renamed copies of clauses from N and a finite set B of fresh constants and a substitution σ, grounding for N' where $\mathrm{codom}(\sigma) = B$ such that $\bigwedge_i \Lambda_i\sigma$ is $\mathcal{T}^{\mathcal{B}}$ satisfiable and $\bigwedge_i C_i\sigma$ is first-order unsatisfiable over $\Sigma^{\mathcal{F}}$. If the SCL(T) rules are applied in a fair way, then they will in particular produce trails solely consisting of literals from $N'\sigma$. For these trails all theory literals are satisfiable, because $\bigwedge_i \Lambda_i\sigma$ is $\mathcal{T}^{\mathcal{B}}$ satisfiable. Furthermore, the states corresponding to these trails cannot end in a stuck state, because this contradicts the unsatisfiability of $\bigwedge_i C_i\sigma$. Instead, they all end in a conflict with some clause in $N'\sigma$. In addition, there are only finitely many such trails, because the number of literals in $N'\sigma$ is finite. Now let $\mu((M; N; U; B; k; \top))$ be the multiset of the levels of all states with trails from $N'\sigma$ until a conflict occurs. Each time a state with a trail from $N'\sigma$ results in a conflict, SCL(T) learns a non-redundant clause that propagates at a strictly smaller level, Lemma 22. Thus $\mu((M; N; U; B; k; \top))$ strictly decreases after each Backtrack step after a conflict on a trail with atoms from $N'\sigma$. The clause learnt at level zero is the empty clause. □

Condition (i) of the above theorem is quite abstract. It can, e.g., be made effective by applying rule Grow only after all possible trail prefixes with respect to the current set B have been explored and to make sure that Decide does not produce the same stuck state twice.

ToDo:{Don't know how useful it would be but we can detect saturation of SCL(T) inferences:

Definition 26. Let N and U be sets of clauses. Then $N \cup U$ is a *saturated* clause set if SCL(T) cannot learn new clauses from $N \cup U$, i.e., there exists no sequence of rule applications following a state $(\epsilon; N; U; B; 0; \top)$ that contains the rule Conflict.

Theorem 27. Let N and U be sets of clauses and B be a sequence of constants such that $|B| \geq 2 \cdot \max_{C \in N \cup U} |\operatorname{vars}(C)|$. Then $N \cup U$ is a saturated clause set iff there exists a regular SCL(T) run starting in state $(\epsilon; N; U; B; 0; \top)$ that does not use rule Grow, explores all possible trail prefixes for B, and does not encounter a single conflict.

Proof. $\Rightarrow$: Assume $N \cup U$ is saturated, then by definition all regular SCL(T) runs starting in $(\epsilon; N; U; B; 0; \top)$ encounter no conflicts. This also includes all runs that do not use rule Grow and explore all possible trail prefixes for B.

$\Leftarrow$: Assume $N \cup U$ is not saturated. Then there exists a regular SCL(T) run $(\epsilon; N; U; B; 0; \top) \Rightarrow^*_{\text{SCL(T)}} \ldots \Rightarrow^{\text{Conflict}}_{\text{SCL(T)}} (M; N; U; B'; k; (\Lambda' \parallel D) \cdot \sigma)$. Due to Corollary 21, $\operatorname{fgd}(M) = M'$, $L\rho^{(\Lambda \parallel C \vee L) \cdot \rho}$, $D \cdot \sigma' = (D' \vee L') \cdot \sigma$, $L \cdot \rho = \operatorname{comp}(L' \cdot \sigma)$, $(\Lambda' \parallel D) \in N \cup U$, and there exists $(\Lambda \parallel C \vee C_1 \vee L) \in N \cup U$ such that $(\Lambda \parallel C \vee C_1 \vee L) \cdot \rho = (\Lambda \parallel C \vee L) \cdot \rho$. Next we create a run that can be executed purely over the constants in B but still leads to a conflict. To this end, we assume w.l.o.g. that $\operatorname{codom}(\rho) \cup \operatorname{codom}(\sigma) \subseteq B$. This is possible because $|\operatorname{codom}(\rho) \cup \operatorname{codom}(\sigma)| \leq |\operatorname{vars}(\Lambda \parallel C \vee C_1 \vee L)| + |\operatorname{vars}(\Lambda' \parallel D)| \leq |B|$. Moreover, we assume that $L_1, \ldots, L_n$ are the complements of the ground literals occurring in $C \cdot \rho$ and $D' \cdot \sigma$. For our new run over B, we start from state $(\epsilon; N; U; B; 0; \top)$ and add the literals $L_1^1, \ldots, L_n^n$ via decisions on the trail. Note that we cannot encounter a conflict by just deciding the literals $L_1, \ldots, L_n$ because (i) a regular run must apply conflict greedily, (ii) the trail M' also contained the literals $L_1, \ldots, L_n$, and (iii) in the last run conflict was only applicable after $L \cdot \rho$ was propagated on top of M'. However, we will encounter one of two cases that will lead to a conflict over B: Case 1: The conditions of a regular run force us to stop our sequence of decisions at a prefix trail $M^* = L_1^1, \ldots, L_i^i$ with $i < n$ because there are two clauses $C^*, D^* \in N \cup U$ such that we can propagate $L^* \cdot \rho^*$ from $C^* \cdot \rho^*$ and receive a conflict in $D^* \cdot \sigma^*$. Hence, we have a regular run over B without rule Grow that leads to a conflict. Case 2: We are able to decide all literals $L_1, \ldots, L_n$. From the trail $M^* = L_1^1, \ldots, L_n^n$ we can propagate $L \cdot \rho$ via the clause $(\Lambda \parallel C \vee C_1 \vee L) \in N \cup U$ and the substitution ρ. This turns $(\Lambda' \parallel D) \cdot \sigma$ into a conflict. Hence, we have a regular run over B without rule Grow that leads to a conflict. In both cases, all regular SCL(T) runs starting in state $(\epsilon; N; U; B; 0; \top)$ that do not use rule Grow and explore all possible trail prefixes for B must encounter this trail prefix and therefore also this conflict. □

}

4 SCL(T) Extensions and Discussion

Example 14 demonstrates that stuck SCL(T) states can sometimes be turned into models of the considered clause set. There exist decidable fragments where this can be done systematically. Consider a pure BS(LRA) clause set N where arithmetic atoms have the form $x\#q$, $x\#y$ or $x - y\#q$ where q is a domain constant and $\# \in \{\leq, <, \neq, =, >, \geq\}$. Furthermore, if a clause contains a difference constraint $x - y\#q$, then x and y are in addition bounded from below and above in the respective clause constraint. The resulting fragment is called BS(BD) [Voi17] and satisfiability of clause sets in this fragment is decidable. This can be effectively done by searching for a refutation with respect to an *a priori* fixed set of constants B. The constants in B are all different, ordered, and enjoy additional bounds with respect to the domain constants occurring in N [Voi17]. The clause set is then unsatisfiable iff it is unsatisfiable after instantiation with B. Otherwise a model can be generated, i.e., BS(BD) enjoys an abstract finite model property. The existence of a refutation can be checked by SCL(T) where the additional bounds on the constants in B can be introduced with the first decision. Actually, [Voi17] was part of our motivation for the model building in SCL(T).

There are further extensions to pure clause sets that still enable a refutationally complete calculus. In particular, first-order function symbols that do not range into a background theory sort and equality. The properties of the SCL(T) calculus rely on finite trails with respect to a fixed, finite set B of constants. By adding non-constant first-order function symbols trails will typically be infinite without further restrictions. Finite trails can, e.g., still be obtained by limiting nestings of function symbols in terms. Thus it seems to us that an extension to first-order function symbols that do not range into a background theory sort should be possible while keeping the properties of SCL(T). From an abstract point of view, also the addition of equality on the first-order side should be possible, because there exist complete procedures such as hierarchic superposition [BGW94, BW19]. Then also foreground function symbols may range into a background theory sort, but the respective terms have to satisfy further conditions in order to preserve completeness. However, even in the pure first-order case there has not been a convincing solution so far of how to combine equational reasoning with explicit model building. One challenge is how to learn a clause from a conflict out of a partial model assumption that enjoys superposition style ordering restrictions on terms occurring in equations. If this can be sufficiently solved, the respective calculus should also be extendable to a hierarchic set up.

An efficient implementation of SCL(T) requires efficient algorithmic solutions to a number of concepts out of the theory. For fast model building an efficient implementation of Propagate is needed. This was our motivation for adding the all-different constraints on the constants, because they enable syntactic testing for complementary or defined literals. In addition, satisfiability of constraints needs to be tested. The trail behaves like a stack and it is ground. This fits perfectly the strengths of SMT-style satisfiability testing. Dealing with the non-domain constants out of the set B needs some care. They behave completely symmetric with respect to the instantiation of clauses in $(N \cup U)$. An easy way to break symmetry here is the addition of linear ordering constraints on these constants. If more is known about the specific fragment some clause set

N belongs to, additional constraints with respect to the constraints or domain constants out of $(N \cup U)$ may be added as well. This is for example the case for the BS(BD) fragment, see above. Exploring all trail prefixes, as required by Theorem 25, requires book-keeping on visited stuck states and an efficient implementation of the rule Restart. The former can be done by actually learning new clauses that represent stuck states. Such clauses are not logical consequences out of N, so they have to be treated special. In case of an application of Grow all these clauses and all the consequences thereof have to be updated. An easy solution would be to forget the clauses generated by stuck states. This can be efficiently implemented. Concerning the rule Restart, from the SAT world it is known that restarts do not have to be total [RvdTH11], i.e., if a certain prefix of a trail will be reproduced after a restart, it can be left on the trail. It seems possible to extend this concept towards SCL(T).

There seems to be the possibility to learn more from a stuck state than just a clause preventing it for the current run until the next application of Grow. For example, if it turns out that the finite model out of a stuck state cannot be extended to an overall model, it might trigger some non-redundant inference, similar to the InstGen calculus [GK03], or result in further constraints for the constants out of B, or actually show that Grow needs to be applied in order to find a model. Finding answers to these problems is subject to future research.

It was enough to restrict SCL(T) to regular runs, Definition 17, to prove the termination of our calculus. An actual implementation might benefit from further restrictions to the explored sequences of rule applications. For SAT solving, one of those restrictions is a focus on greedy propagation. The same might not be true for SCL(T). Still, we assume that a restriction that enforces some progress through propagation might be necessary, e.g., the rule Decide is only applied if the rule Propagate has been applied at least once to all applicable clauses since the last application of Decide or the the initial state. We also assume that some kind of fairness guarantee must be kept to prevent a small subset of clauses from causing the majority of propagated literals. But these are for sure not the only options.

In summary, we have presented the new calculus SCL(T) for pure clause sets of first-order logic modulo a background theory. The calculus is sound and complete. It does not generate redundant clauses. There is a fair strategy that turns it into a semi-decision procedure. It constitutes a decision procedure for certain decidable fragments of pure clause sets, such as BS(BD).

References

[AW15] Gábor Alagi and Christoph Weidenbach. NRCL – A model building approach to the Bernays-Schönfinkel fragment. In Carsten Lutz and Silvio Ranise, editors, *Frontiers of Combining Systems – 10th International Symposium, FroCoS 2015, Wroclaw, Poland, September 21-24, 2015. Proceedings*, volume 9322 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 2015.

[BFT06] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Lemma learning in the model evolution calculus. In *LPAR*, volume 4246 of *Lecture Notes in Computer Science*, pages 572–586. Springer, 2006.

[BFT08] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. (LIA) - model evolution with linear integer arithmetic constraints. In *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings*, volume 5330 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2008.

[BGW94] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Refutational theorem proving for hierarchic first-order theories. *Applicable Algebra in Engineering, Communication and Computing, AAECC*, 5(3/4):193–212, 1994.

[BP16] Maria Paola Bonacina and David A. Plaisted. Semantically-guided goal-sensitive reasoning: Model representation. *Journal of Automated Reasoning*, 56(2):113–141, 2016.

[BSW15] Martin Bromberger, Thomas Sturm, and Christoph Weidenbach. Linear integer arithmetic revisited. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 623–637. Springer, 2015.

[BW19] Peter Baumgartner and Uwe Waldmann. Hierarchic superposition revisited. In Carsten Lutz, Uli Sattler, Cesare Tinelli, Anni-Yasmin Turhan, and Frank Wolter, editors, *Description Logic, Theory Combination, and All That - Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday*, volume 11560 of *Lecture Notes in Computer Science*, pages 15–56. Springer, 2019.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Engineering DPLL(T) + saturation. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008*, volume 5195 of *LNCS*, pages 475–490. Springer, 2008.

[Dow72] Peter J. Downey. Undecidability of presburger arithmetic with a single monadic predicate letter. Technical report, Center for Research in Computer Technology, Harvard University, 1972.

[FW19] Alberto Fiori and Christoph Weidenbach. SCL clause learning from simple models. In *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, pages 233–249, 2019.

[GdM09] Yeting Ge and Leonardo Mendonça de Moura. 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*, volume 5643 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2009.

[GK03] Harald Ganzinger and Konstantin Korovin. New directions in instatiation–based theorem proving. In Samson Abramsky, editor, *18th Annual IEEE Symposium on Logic in Computer Science, LICS'03*, LICS'03, pages 55–64. IEEE Computer Society, 2003.

[HVW17] Matthias Horbach, Marco Voigt, and Christoph Weidenbach. The universal fragment of presburger arithmetic with unary uninterpreted predicates is undecidable. *CoRR*, abs/1703.01212, 2017.

[JS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In Benjamin Kuipers and Bonnie L. Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA.*, pages 203–208, 1997.

[Kor13] Konstantin Korovin. Inst-gen - A modular approach to instantiation-based automated reasoning. In Andrei Voronkov and Christoph Weidenbach, editors, *Programming Logics - Essays in Memory of Harald Ganzinger*, volume 7797 of *Lecture Notes in Computer Science*, pages 239–270. Springer, 2013.

[KV13] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.

[KW12] Evgeny Kruglov and Christoph Weidenbach. Superposition decides the first-order logic fragment over ground theories. *Mathematics in Computer Science*, 6(4):427–456, 2012.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines.* Automatic Computation. Prentice-Hall, 1967.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 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.

[PdMB10] Ruzica Piskac, Leonardo Mendonça de Moura, and Nikolaj Bjørner. Deciding effectively propositional logic using DPLL and substitution sets. *Journal of Automated Reasoning*, 44(4):401–424, 2010.

[PW06] Virgile Prevosto and Uwe Waldmann. SPASS+T. In Geoff Sutcliffe, Renate Schmidt, and Stephan Schulz, editors, *ESCoR: FLoC'06 Workshop on Empirically Successful Computerized Reasoning*, volume 192 of *CEUR Workshop Proceedings*, pages 18–33, Seattle, WA, USA, 2006.

[RBF18] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. Revisiting enumerative instantiation. In Dirk Beyer and Marieke Huisman, editors, *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*, volume 10806 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 2018.

[Rüm08] Philipp Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *LPAR*, volume 5330 of *LNCS*, pages 274–289. Springer, 2008.

[RvdTH11] Antonio Ramos, Peter van der Tak, and Marijn Heule. Between restarts and backjumps. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 216–229. Springer, 2011.

[SS96] João P. Marques Silva and Karem A. Sakallah. Grasp - a new search algorithm for satisfiability. In *International Conference on Computer Aided Design, ICCAD*, pages 220–227. IEEE Computer Society Press, 1996.

[TW15] Andreas Teucke and Christoph Weidenbach. First-order logic theorem proving and model building via approximation and instantiation. In Carsten Lutz and Silvio Ranise, editors, *Frontiers of Combining Systems, 10th International Symposium, FroCoS 2015, Wroslav, Poland, 2015. Proceedings*, volume 9322 of *LNCS*, pages 85–100. Springer, 2015.

[Voi17] Marco Voigt. The Bernays-Schönfinkel-Ramsey fragment with bounded difference constraints over the reals is decidable. In Clare Dixon and Marcelo Finger, editors, *Frontiers of Combining Systems - 11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27-29, 2017, Proceedings*, volume 10483 of *Lecture Notes in Computer Science*, pages 244–261. Springer, 2017.

[Wei15] Christoph Weidenbach. Automated reasoning building blocks. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015. Proceedings*, volume 9360 of *Lecture Notes in Computer Science*, pages 172–188. Springer, 2015.