8.2 Syntax and Semantics

Similar to the Nelson-Oppen theory combination, Section 7.1, the starting point are two theories over two disjoint signatures. However, in first-order logic modulo theories there are universally quantified variables.

As usual in the superposition context I consider equality as the only predicate. Nevertheless, for convenience, I write $x \geq 1 \lor P(x) \lor \neg N(x) \lor f(x) \approx x$ instead of $x \geq 1 \lor f_P(x) \approx \text{true} \lor f_N(x) \not\approx \text{true} \lor f(x) \approx x$.

Definition 8.2.1 (Hierarchic Theory and Specification)**.** Let $\mathcal{T}^B = (\Sigma^B, \mathcal{C}^B)$ be a many-sorted theory, called the *background theory* and Σ^B the *background signature.* Let Σ^F be a many sorted signature with $\Omega^B \cap \Omega^F = \emptyset$, $\mathcal{S}^B \subset \mathcal{S}^F$, called the *foreground signature* or *free signature.* Let $\Sigma^H = (\mathcal{S}^B \cup \mathcal{S}^F, \Omega^B \cup \Omega^F)$ be the union signature and N be a set of clauses over Σ^H, and $\mathcal{T}^H = (\Sigma^H, N)$ called a *hierarchic theory.* A pair $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$ is called a *hierarchic specification.*

For the rest I assume a hierarchic specification $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$. I abbreviate $\models_{\mathcal{T}^B} \phi$ ($\models_{\mathcal{T}^H} \phi$) with $\models_B \phi$ ($\models_H \phi$), meaning that ϕ is valid in the respective theory, see Definition 3.17.1. Terms, atoms, literals build over Σ^B are called *pure background terms*, *pure background atoms*, and *pure background literals*, respectively. Non-variable terms, atoms, literals build over Σ^F are called *free terms*, *free atoms*, *free literals.* A variable of sort $S \in (\mathcal{S}^F \setminus \mathcal{S}^B)$ is also called a *free variable* and a *free term.* Any term of some sort $S \in \mathcal{S}^B$ built out of Σ^H is called a *background term.* A substitution σ is called *simple* if $x_S\sigma \in T_S(\Sigma^B, \mathcal{X})$ for all $S \in \mathcal{S}^B$.

Example 8.2.2 (Classes of Terms)**.** Let $\mathcal{T}^B$ be linear rational arithmetic and $\Sigma^F = (\{S, \text{LA}\}, \{g, a\})$ where $a\colon S$ and $g\colon \text{LA} \to \text{LA}$. Then the terms $x_{\text{LA}} + 3$ and $g(x_{\text{LA}})$ are all of sort LA, but $x_{\text{LA}} + 3$ is a pure background term whereas $g(x_{\text{LA}})$ is a free term and an unpure background term. So the substitution $\sigma = \{y_{\text{LA}} \mapsto x_{\text{LA}} + 3\}$ is simple while $\sigma = \{y_{\text{LA}} \mapsto g(x_{\text{LA}})\}$ is not.

Definition 8.2.3 (Hierarchic Algebras)**.** Given a hierarchic specification $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$, $\mathcal{T}^B = (\Sigma^B, \mathcal{C}^B)$, $\mathcal{T}^H = (\Sigma^H, N)$, a Σ^H-algebra $\mathcal{A}$ is called *hierarchic* if $\mathcal{A}|_{\Sigma^B} \in \mathcal{C}^B$. A hierarchic algebra $\mathcal{A}$ is called a *model of a hierarchic specification* $\mathcal{H}$, if $\mathcal{A} \models N$.

Definition 8.2.4 (Abstracted Term, Atom, Literal, Clause)**.** A term t is called *abstracted* with respect to a hierarchic specification $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$, if $t \in T_S(\Sigma^B, \mathcal{X})$ or $t \in T_T(\Sigma^F, \mathcal{X})$ for some $S \in \mathcal{S}^B$, $T \in \mathcal{S}^B \cup \mathcal{S}^F$. An equational atom $t \approx s$ is called *abstracted* if t and s are abstracted and both pure or both unpure, accordingly for literals. A clause is called *abstracted* of all its literals are abstracted.

Given a clause set N of a hierarchic specification $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$ the clauses in N can be transformed into abstracted clauses, preserving satisfiability by the below abstraction rule.

Abstraction $N \uplus \{C \lor E[t]_p[s]_q\} \;\Rightarrow_{\text{ABSTR}}\; N \cup \{C \lor x_s \not\approx s \lor E[x_S]_q\}$

provided t, s are non-variable terms, $q \not< p$, $\operatorname{sort}(s) = S$, and either $\operatorname{top}(t) \in \Sigma^F$ and $\operatorname{top}(s) \in \Sigma^B$ or $\operatorname{top}(t) \in \Sigma^B$ and $\operatorname{top}(s) \in \Sigma^F$

Proposition 8.2.5 (Properties of the Abstraction)**.** Given a finite clause set N out of a hierarchic specification $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$, $\Rightarrow_{\text{ABSTR}}$ terminates on N and preserves satisfiability. For any clause $C \in (N \Downarrow_{\text{ABSTR}})$ and any literal $E \in C$, E does not both contain a function symbol from Σ^B and a function symbol from Σ^F.

Proof. For termination consider the number of all different term occurrences in all atoms E that have a Σ^B and Σ^F top symbol, respectively. An application of the rule strictly decreases the number of such occurrences. For the preservation of satisfiability consider the clause $(C \vee x_s \not\approx s \vee E[x_S]_q)\{x_s \mapsto s\}$. Finally, if some equation E contains a function symbol from Σ^B and a function symbol from Σ^F, then $\Rightarrow_{\text{ABSTR}}$ is applicable. $\square$

From now on I assume fully abstracted clauses C, i.e., for all atoms $s \approx t$ occurring in C, either $s, t \in T(\Sigma^B, \mathcal{X})$ or $s, t \in T(\Sigma^F, \mathcal{X})$. This justifies the notation of clauses $\Lambda \parallel C$ where all pure background literals are in Λ and belong to $\operatorname{FOL}(\Sigma^B, \mathcal{X})$ and all literals in C belong to $\operatorname{FOL}(\Sigma^F, \mathcal{X})$. The literals in Λ form a conjunction and the literals in C a disjunction and the overall clause the implication $\Lambda \rightarrow C$. For a clause $\Lambda \parallel C$ the background theory part Λ is called the *constraint* and C the *free part* of the clause.

Note that a clause $C \vee x \not\approx y$ with $\operatorname{sort}(x) \in \mathcal{S}^B$ can be replaced by $C\{x \mapsto y\}$ and for an atom $x \approx y$ with $\operatorname{sort}(x) \in \mathcal{S}^B$ it is moved to the constraint.

Example 8.2.6 (Abstracted Clause)**.** Continuing Example 8.2.2, the unabstracted clause

$$g(x) \leq 1 + y \vee g(g(1)) \approx 2$$

corresponds to the abstracted clause

$$z \not\approx g(x) \vee z \leq 1 + y \vee u \not\approx 2 \vee v \not\approx 1 \vee g(g(v)) \approx u$$

that is written

$$z > 1 + y \wedge u \approx 2 \wedge v \approx 1 \parallel z \not\approx g(x) \vee g(g(v)) \approx u$$

8.3 The SUP(T) Calculus on Abstracted Clauses

As usual the calculus is presented with respect to a reduction ordering $\prec$, total on ground terms. For the SUP(T) calculus I assume that any pure base term is strictly smaller than any term containing a function symbol from Σ^F. This justifies the below ordering conditions with respect to the constraint notation of clauses and can, e.g., be obtained by an LPO where all symbols from Σ^B are smaller in the precedence than the symbols from Σ^F.

Superposition Right $(N \uplus \{\Lambda \parallel D \vee t \approx t', \Gamma \parallel C \vee s[u] \approx s'\}) \Rightarrow_{\text{SUPT}}$ $(N \cup \{\Lambda \parallel D \vee t \approx t', \Gamma \parallel C \vee s[u] \approx s'\} \cup \{(\Lambda, \Gamma \parallel D \vee C \vee s[t'] \approx s')\sigma\})$

8.10 Complexity

In Section 3.15 I have already shown that unsatisfiability of a first-order clause set is undecidable. Extending first-order clauses with linear arithmetic generalizes the logic, therefore first-order logic modulo linear arithmetic is undecidable as well. However, for FOL(LIA) there is a more compact and a more natural proof in the sense that a natural reduction of the halting problem for a simple programming language becomes available.

The simple programming language was invented by Minsky [**?**] and is based on a so called 2-counter machine. The memory of the machine are two integer counters k_1, k_2, where the integers are not limited in size, resulting in the name. The counters may be initialized at the beginning with arbitrary positive values.

A program consists of a finite number of programming lines, each coming with a unique and consecutive line number and containing exactly one instruction. The available instructions are:

$\mathrm{inc}(k_i)$	increment counter k_i and proceed with the next line,
$\mathrm{td}(k_i, n)$	if $k_i > 0$ then decrement k_i and proceed with the next line, otherwise goto line n and leave the counters unchanged,
$\mathrm{goto}\, n$	goto line n,
halt	halt the computation.

The following 2-counter machine program adds the initial values of the counters and stores the values in k_1.

```
1   td(k_2, 4)
2   inc(k_1)
3   goto 1
4   halt
```

Theorem 8.10.1 (2-Counter Machine Halting Problem)**.** The halting problem for 2-counter machines is undecidable [**?**].

Proof. (Idea) By a reduction to the halting problem for Turing machines. □

A 2-counter machine program can be immediately be translated into a FOL(LIA) clause set N such that the N is unsatisfiable iff the program reaches the halt instruction. There is only one predicate symbol R of arity three needed, where the first argument encodes the current line and the other two arguments the respective counters. Then clauses corresponding to the above program with start values n, m for the k_1, k_2, respectively is:

$$\begin{aligned}
x = n, y = m &\rightarrow R(1, x, y),\\
y > 0, y' = y - 1, R(1, x, y) &\rightarrow R(2, x, y'),\\
y = 0, R(1, x, y) &\rightarrow R(4, x, y'),\\
x' = x + 1, R(2, x, y) &\rightarrow R(3, x', y),\\
R(3, x, y) &\rightarrow R(1, x, y),\\
R(4, x, y) &\rightarrow .
\end{aligned}$$

A consequence is that unsatisfiability of a FOL(LIA) clause set with a single ternary predicate symbol is undecidable.

Proposition 8.10.2 (FOL(LIA) Undecidability with a Single Ternary Predicate)**.** Unsatisfiability of a FOL(LIA) clause set with a single ternary predicate is undecidable.

The ternary predicate is not mandatory. There is a translation of a 2-counter machine program into FOL(LIA) using exactly one monadic predicate. This is surprising because, the monadic fragment, i.e., arbitrary FOL formulas over solely monadic predicates is decidable. The idea of the translation is to encode a state (i, n, m) where the program is at line i with respective counter values n, m by the integer $2^n \cdot 3^m \cdot p_i$ where p_i is the i^{th} prime number following 3. The state $2^n \cdot 3^m \cdot p_i$ corresponds to the clause $x = n, y = m \rightarrow R(i, x, y)$ to be true with respect to the ternary predicate encoding. For example, the initial state correspond, expressed by the above clause $x = n, y = m \rightarrow R(1, x, y)$ is expressed by the integer $2^n \cdot 3^m \cdot 5$. The above 2-counter machine program is then translated into

$$\begin{aligned}
x = k &\rightarrow S(x),\\
5y = x, 3y' = y, x' = 7y', S(x) &\rightarrow S(x')\\
5y = x, 3y' + 1 = y, x' = 13y', S(x) &\rightarrow S(x')\\
5y = x, 3y' + 2 = y, x' = 13y', S(x) &\rightarrow S(x')\\
7y = x, x' = 2y, x'' = 11x', S(x) &\rightarrow S(x'')\\
11y = x, x' = 5y, S(x) &\rightarrow S(x')\\
13y = x, S(x) &\rightarrow
\end{aligned}$$

where $k = 2^n \cdot 3^m \cdot 5$, and the line numbers $1, 2, 3, 4$ of the original 2-counter machine program correspond to the prime numbers $5, 7, 11, 13$, respectively. Note that divisibility in general can only be expressed by a quantifier alternation on the basis of multiplication: $\forall x, y.x \mid y$ iff $\forall x, y \exists z.x \cdot z = y$. But since in the encoding constant prime numbers are used, divisibility can be encoded without a quantifier alternation, by explicitely considering all finitely many remainders.

Proposition 8.10.3 (FOL(LIA) Undecidability with a single Monadic Predicate)**.** Unsatisfiability of a FOL(LIA) clause set with a single monadic predicate is undecidable [**?**].

8.11 Further Decidable FOL(T) Fragments

In this section I study three FOL(T) fragments that are decidable. The fragments are obtained by syntactic restrictions. I assume in this section that the considered clause sets are sufficiently complete, but compactness needs not to hold. Furthermore, I don't consider equations, i.e., the SUP(T) calculus instantiates to the ordered resolution calculus modulo theories: Superposition Right only generates tautologies, Superposition Left becomes ordered resolution, Equality Factoring becomes factoring and Equality Resolution is not applicable.

8.11.1 Totally Ordered Clause Sets

For this fragment the only requirement is that satisfiability of constraints is decidable. For example, a constraint language of non-linear real arithmetic.

Definition 8.11.1 (Closed Literal Set)**.** Let M be a set of first-order (non-equational) literals over Σ^F closed under SUP(T) inferences: for any two clauses $C_1, C_2 \in 2^M$ and SUP(T) inference D out of C_1, C_2, it holds $D \subset M$. Then M is called a *closed literal set.*

Definition 8.11.2 (Totally Ordered Horn Clause Sets)**.** Let M be a closed literal set, and $\prec$ be a well-founded partial ordering on M stable under substitution and instantiation such that for all $\Lambda \parallel C \in N$: (i) $C \subset M$, (ii) C is Horn, (iii) if $C = C' \lor P(t_1, \ldots, t_n)$ then for all $L \in C'$: $L \prec P(t_1, \ldots, t_n)$. Then N is called a *totally ordered Horn clause set.*

Recall that an ordering $\prec$ is stable under substitution if $L \prec K$ implies $L\sigma \prec K\sigma$ for any σ. It is stable under instantiation if $L\sigma \preceq L$ for any σ.

Lemma 8.11.3 (Saturation of Totally Ordered Horn Clause Sets Terminates)**.** Let N be a totally ordered Horn clause set. Then SUP(T) terminates on N.

Proof. For Horn clause sets every SUP(T) inference has the form: from $\Lambda_1 \parallel C \lor P(t_1, \ldots, t_n)$ and $\Lambda_2 \parallel D \lor \neg P(s_1, \ldots, s_n)$ the clause $(\Lambda_1, \Lambda_2 \parallel C \lor D)\sigma$ is inferred where σ is the mgu of $P(t_1, \ldots, t_n)$ and $P(s_1, \ldots, s_n)$. Now $(C \lor D)\sigma$ is a purely negative clause and $(C \lor D)\sigma \prec_{\text{mul}} (D \lor \neg P(s_1, \ldots, s_n))\sigma$, by definition of the total literal ordering $\preceq$. The ordering $\prec_{\text{mul}}$ is well-founded, hence SUP(T) terminates. □

Theorem 8.11.4 (Satisfiability of Totally Ordered Horn Clause Sets is Decidable)**.** Let N be a totally ordered Horn clause set. Then satisfiability of N is decidable.

Example 8.11.5 (Predicate Preference)**.** Let N be a clause set and $P_1, \ldots, P_n$ be the predicates in N. Let $\prec$ be a total order on the P_i. It can be extended to literals by $P_i(t_1, \ldots, t_n) \prec P_j(t_1, \ldots, t_n)$ if $P_i \prec P_j$. The extension is stable under substitution and instantiation. Then satisfiability of any totally ordered Horn clause set with respect to $\prec$ is decidable.

8.12 Bernays-Schönfinkel with Simple Bounds

In this section I only consider clauses $\Lambda \parallel C$ where Λ is a conjunction of simple bounds over LRA and C is a Bernays-Schönfinkel clauses, i.e., the free part only consists of variables and constants. A *simple bound* is an (in)equality $x \# k$ where $k \in \mathbb{Z}$ and $\# \in \{<, \leq, >, \geq, =, \neq\}$. In Section 3.16 I have introduced a number of calculi that can decide the Bernays-Schönfinkel fragment. Here I prove that Bernays-Schönfinkel with simple bounds can also be decided by exactly the superposition variant introduced in Section 3.16.1. I assume that in any inferred clauses by superposition or instantiation the constraint is always simplified, i.e., for any clause $\Lambda \parallel C$ all constraint variables occur in C, for every such variable x there is at most one upper and one lower bound and duplicates are removed.

Lemma 8.12.1 (BS with Simple Bounds Invariants)**.** Let N be a clause set of the Bernay-Schönfinkel fragment with simple bounds. Then

1. Any inference between clauses from N results again in a BS clause with simple bounds. The class of Bernays-Schoenfinkel clauses with simple bounds is closed under SUP(T) inferences.
2. Let $\{k_1, \ldots, k_n\}$ be all numeric values occurring in the constraints in N. Then also for any clause inferred by SUP(T) from N, only the numeric values $\{k_1, \ldots, k_n\}$ occur.
3. For any arithmetic variable x at most n non-redundant simple bounds out of $\{k_1, \ldots, k_n\}$ can be generated.

Condensation-BS $(N \uplus \{\Lambda \parallel L_1 \vee \cdots \vee L_n\}) \Rightarrow_{\text{SUP}} (N \cup \{\text{rdup}((\Lambda \parallel L_1 \vee \ldots \vee L_n)\sigma_{i,j}) \mid \sigma_{i,j} = \text{mgu}(L_i, L_j) \text{ and } \sigma_{i,j} \neq \bot\})$

provided any ground instantiation on the free variables $(L_1 \vee \cdots \vee L_n)\delta$ contains at least two duplicate literals with identical simple bounds

Lemma 8.12.2. Let N be a BS clause set with simple bounds. There are only finitely many BSS clauses derivable from N where Condensation-BS is not applicable.

Theorem 8.12.3. Satisfiability of a BS clause set with simple bounds is decidable.

8.13 Bernays-Schönfinkel with Bounded Variables

In this section I only consider clauses $\Lambda \parallel C$ where for each arithmetic variable $x \in \text{vars}(C)$ there are bounds $i \leq x \leq j$ in Λ, $i, j \in \mathbb{Z}$, $i \leq j$ and all arithmetic variables are integer variables. The constraint λ may contain any further, even non-linear constraints. For example, the constraint language of non-linear integer arithmetic, i.e., polynomials are allowed.

Theorem 8.13.1 (BS with Bounded Constraints is Decidable)**.** Satisfiability of BS clause sets with bounded constraints over the integers is decidable.

Proof. For any arithmetic variable x occurring in a clause $\Lambda \parallel C$ there are exactly $j - i$ integer instances satisfying the bound $i \leq x \leq j$. Thus by eager instantiation of the integer variables any BS clause set with bounded NIA constraints can be transformed into a BS clause set, where the arithmetic values are considered as extra constants of the arithmetic sort. $\square$

8.14 SCLT Clause Learning from Simple Models Modulo Theories

Let $\mathcal{T}^{\mathcal{B}}$ be first-order logic *background theory* over signature $\Sigma^{\mathcal{B}} = (\mathcal{S}^{\mathcal{B}}, \Omega^{\mathcal{B}}, \Pi^{\mathcal{B}})$ and 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 $d \neq c$. Let $\Sigma^{\mathcal{F}} = (\mathcal{S}^{\mathcal{F}}, \Omega^{\mathcal{F}}, \Pi^{\mathcal{F}})$ be a *foreground signature* with respect to $\mathcal{T}^{\mathcal{B}}$ where $\mathcal{S}^{\mathcal{B}} \subseteq \mathcal{S}^{\mathcal{F}}$, $\Omega^{\mathcal{B}} \cap \Omega^{\mathcal{F}} = \emptyset$, and $\Pi^{\mathcal{B}} \cap \Pi^{\mathcal{F}} = \emptyset$.

Definition 8.14.1 (Hierarchic Specification)**.** A *hierarchic specification* is a pair $\mathcal{H} = (\mathcal{T}^{\mathcal{B}}, \Sigma^{\mathcal{F}})$ with associated signature $\Sigma^{\mathcal{H}} = (\mathcal{S}^{\mathcal{F}}, \Omega^{\mathcal{B}} \cup \Omega^{\mathcal{F}}, \Pi^{\mathcal{B}} \cup \Pi^{\mathcal{F}})$. It generates *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.

As a running example, I consider in detail the Bernays-Schoenfinkel clause fragment over linear arithmetic: BS(LRA). The background theory is linear rational arithmetic over the many-sorted signature $\Sigma^{\text{LRA}} = (\mathcal{S}^{\text{LRA}}, \Omega^{\text{LRA}}, \Pi^{\text{LRA}})$ with $\mathcal{S}^{\text{LRA}} = \{\text{LRA}\}$, $\Omega^{\text{LRA}} = \{0, 1, +, -\} \cup \mathbb{Q}$, $\Pi^{\text{LRA}} = \{\leq, <, \neq, =, >, \geq\})$ 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 $\mathcal{T}^{\text{LRA}} = (\Sigma^{\text{LRA}}, \{\mathcal{A}^{\text{LRA}}\})$ consists of the linear arithmetic signature together with the standard model $\mathcal{A}^{\text{LRA}}$ of linear arithmetic. This theory is then extended by the free (foreground) first-order signature $\Sigma^{\text{BS}} = (\{\text{LRA}\}, \Omega^{\text{BS}}, \Pi^{\text{BS}})$ 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 $\mathcal{A}^{\text{BS(LRA)}}$ over the signature $\Sigma^{\text{BS(LRA)}} = (\{\text{LRA}\}, \Omega^{\text{BS}} \cup \Omega^{\text{LRA}}, \Pi^{\text{BS}} \cup \Pi^{\text{LRA}})$ that are $\Sigma^{\text{BS(LRA)}}$ algebras such that $\mathcal{A}^{\text{BS(LRA)}}|_{\Sigma^{\text{LRA}}} = \mathcal{A}^{\text{LRA}}$.

Definition 8.14.2 (Simple Substitutions)**.** A substitution σ is called *simple* if $x_S\sigma \in T_S(\Sigma^{\mathcal{B}}, \mathcal{X})$ for all $x_S \in \text{dom}(\sigma)$ 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 $N^{\preceq\Lambda\parallel C} := \{D \mid D \in N \text{ and } D \preceq \Lambda \parallel C\}$. For example, an instance of an LPO with according precedence can serve as $\prec$.

Definition 8.14.3 (Abstracted/Pure Clause)**.** A clause $\Lambda \parallel C$ is *abstracted* if the arguments of $\mathcal{S}^{\mathcal{B}}$ sort of any predicate from $\Pi^{\mathcal{F}}$ in an atom in C are only variables. $\Lambda \parallel C$ is called *pure* if it does not contain symbols from $\Omega^{\mathcal{F}}$ ranging into a sort of $\mathcal{S}^{\mathcal{B}}$.

These two notions are extended to clause sets in the natural way. Any clause set can be transformed into an abstracted clause set.

Abstraction $N \uplus \{C \vee E[t]_p[s]_q\} \;\Rightarrow_{\text{ABSTR}}\; N \cup \{C \vee x_s \not\approx s \vee E[x_S]_q\}$
provided t, s are non-variable terms, $q \not< p$, $\text{sort}(s) = S$, and either $\text{top}(t) \in \Sigma^F$ and $\text{top}(s) \in \Sigma^B$ or $\text{top}(t) \in \Sigma^B$ and $\text{top}(s) \in \Sigma^F$

In case of BS(LRA) abstraction can only be applied to constants below a predicate.

Definition 8.14.4 (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 $\Lambda' \parallel C' \in \text{grd}((\mathcal{S}^{\mathcal{F}}, B, \Pi^{\mathcal{B}} \cup \Pi^{\mathcal{F}}), \Lambda \parallel C)$ the clause $\Lambda' \parallel C'$ is redundant with respect to $\cup_{D\in N}\, \text{grd}((\mathcal{S}^{\mathcal{F}}, B, \Pi^{\mathcal{B}} \cup \Pi^{\mathcal{F}}), D)$.

Assumption 8.14.5 (Considered Clause Sets)**.** For the rest of this section I consider only pure, abstracted clause sets N. I 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. I further assume that the set $\Omega^{\mathcal{F}}$ contains infinitely many constants for each background sort.

Example 8.14.6 (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 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, 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, 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 comp(L) occurs in M and the first decision literal left from L (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 $(M; N; U; B; k; \top) \Rightarrow_{\text{SCL(T)}} (M, L\sigma^{k+1}, \Lambda\sigma; N; U; B; k+1; \top)$

provided $L\sigma$ is undefined in M, $|L\sigma| \in \text{atoms}(\text{grd}((\mathcal{S}, B, \Pi), N \cup U))$, $|K\sigma| \in \text{atoms}(\text{grd}((\mathcal{S}, B, \Pi), N \cup U))$ for all $K\sigma \in \Lambda\sigma$, σ is grounding for Λ, all background literals in $\Lambda\sigma$ are undefined in M, $\text{adiff}(B) \wedge \text{bgd}(M) \wedge \Lambda\sigma$ is satisfiable, and $\text{codom}(\sigma) \subseteq B$

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 $\text{adiff}(B)$ constraint enables a purely syntactic consistency check for foreground literals.

Conflict $(M; N; U; B; k; \top) \Rightarrow_{\text{SCL(T)}} (M; N; U; B; k; \Lambda \parallel D \cdot \sigma)$

provided $\Lambda \parallel D \in (N \cup U)$, σ is grounding for $\Lambda \parallel D$, $\text{adiff}(B) \wedge \text{bgd}(M) \wedge \Lambda\sigma$ is satisfiable, $\text{fgd}(M) \models \neg(D\sigma)$, and $\text{codom}(\sigma) \subseteq B$

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

provided $L\rho = \text{comp}(L'\sigma)$, and $\eta = \text{mgu}(L, \text{comp}(L'))$

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 $(M; N; U; B; k; (\Lambda \parallel D \lor L \lor L') \cdot \sigma) \Rightarrow_{\mathrm{SCL(T)}} (M; N; U; B; k; (\Lambda \parallel D \lor L)\eta \cdot \sigma)$
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 8.14.9.

Backtrack $(M, K^{i+1}, M'; N; U; B; k; (\Lambda \parallel D \lor L) \cdot \sigma) \Rightarrow_{\mathrm{SCL(T)}} (M, L\sigma^{(\Lambda \parallel D \lor L) \cdot \sigma}, \Lambda'\sigma; N; U \cup \{\Lambda \parallel D \lor L\}; B; i; \top)$
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 $(D \lor L)\sigma$ 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 8.14.7. 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 8.14.8 (Inconsistent Trail)**.** Consider a clause set $N = \{R(x,y), x \leq y \parallel \neg R(x,y) \lor P(x), x \geq y \parallel \neg R(x,y) \lor \neg P(y)\}$; 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 8.14.9 (Exhaustive Propagation)**.** Consider a BS(LRA) clause set $N = \{x = 0 \parallel \mathrm{Nat}(x),\ y = x+1 \parallel \neg\,\mathrm{Nat}(x) \lor \mathrm{Nat}(y)\} \cup N'$ 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 8.14.10 (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. $M \land \mathrm{adiff}(B)$ 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 fgd(M) and bgd$(M) \land \mathrm{adiff}(B) \land \Lambda\sigma$ is satisfiable,
   (b) if $M = M_1, L\sigma^{(\Lambda \parallel C \lor L)\cdot\sigma}, M_2$ then $C\sigma$ is false in fgd(M_1), $L\sigma$ is undefined in M_1, and bgd$(M_1) \land \mathrm{adiff}(B) \land \Lambda\sigma$ is satisfiable.
5. All clauses in $N \cup U$ are pure. In particular, they don't contain any constants from B.

Lemma 8.14.11 (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 $\text{con}((M'; N'; U'; B'; k'; D')) \subseteq \text{con}(N') \cup B'$ 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 $\text{con}(M') \subseteq \text{con}(N) \cup B$. Both rules require $|L\sigma| \in \text{atoms}(\text{grd}((\mathcal{S}, B, \Pi), N \cup U))$ satisfying the claim.

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

If the rule Conflict was used then only the last component of the state changed $D' = \Lambda \parallel C \cdot \sigma$ with $\text{codom}(\sigma) \subseteq B$ and $\text{con}(\Lambda \parallel C) \subseteq \text{con}(N) \cup B$ 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 $\text{adiff}(B)$ 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 $M \wedge \text{adiff}(B)$ satisfiable implies $M' \wedge \text{adiff}(B')$ 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 $\text{adiff}(B')$ 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 $\text{bgd}(M') \wedge \text{adiff}(B')$ is satisfiable.

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

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

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 $\text{adiff}(B) \wedge \text{bgd}(M) \wedge \Delta\delta$ is satisfiable and $\text{fgd}(M) \models \neg(D\delta)$ is shown for 4a. There is nothing to show for 4b.

Consider an application of rule Resolve

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

by induction hypothesis $\operatorname{bgd}(M) \wedge \operatorname{adiff}(B) \wedge \Lambda\eta\rho\sigma$ is satisfiable and $C\eta\rho\sigma$ is false in $\operatorname{fgd}(M)$, 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 $\operatorname{bgd}(M, L\sigma) \wedge \operatorname{adiff}(B) \wedge \Lambda'\eta\delta\sigma$ is satisfiable and $(D \vee L')\eta\delta\sigma)$ is false in $\operatorname{fgd}(M, L\sigma)$. Therefore $(D \vee C)\eta \cdot \rho\sigma$ is false in $\operatorname{fgd}(M, L\sigma)$ and $\operatorname{bgd}(M, L\sigma) \wedge \operatorname{adiff}(B) \wedge (\Lambda' \wedge \Lambda)\eta\delta\sigma$ 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

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

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

Finally, when applying Backtrack

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

there is nothing to show for 4a. For 4b we know by induction hypothesis that $(D \vee L)\sigma$ is false in $\operatorname{fgd}(M, K^{i+1}, M')$. The literal $L\sigma$ is of level k and $D\sigma$ of level i, $k > i$, hence $D\sigma$ is false in $\operatorname{fgd}(M)$ and $L\sigma$ undefined in $\operatorname{fgd}(M)$. Furthermore, $\operatorname{bgd}(M, K^{i+1}, M') \wedge \operatorname{adiff}(B) \wedge \Lambda\sigma$ is satisfiable by induction hypothesis, so $\operatorname{bgd}(M) \wedge \operatorname{adiff}(B) \wedge \Lambda\sigma$ 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 8.14.12 (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 8.14.13 (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 $\text{comp}(L\sigma)$ is not contained in $D\delta$ then Skip can be applied. If $M = M', L\sigma^{\Lambda\parallel C\cdot\sigma}$ with $D\delta = D' \vee \text{comp}(L\sigma)$ then Resolve can be applied. If $M = M', L\sigma^k, M''$ and D' contains multiple occurrences of $\text{comp}(L\sigma)$ 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 $|L| \in \text{atoms}(\text{grd}((\mathcal{S}, B, \Pi), N))$ is undefined, we can always apply Decide. □

Lemma 8.14.14 (Stuck States Produce Ground Models)**.** If a state $(M; N; U; B; k; \top)$ is stuck then $M \wedge \text{adiff}(B) \models \text{grd}((\mathcal{S}, B, \Pi), N \cup U)$.

Proof. By contradiction. Note that $M \wedge \text{adiff}(B)$ is satisfiable, Lemma 8.14.11.2. Consider any clause $(\Lambda \parallel C)\sigma \in \text{grd}((\mathcal{S}, B, \Pi), N \cup U)$. It can only be not true in $M \wedge \text{adiff}(B)$ if $\text{fgd}(M) \models \neg(C\sigma)$ and $\text{bgd}(M) \wedge \text{adiff}(B) \wedge \Lambda\sigma$ is satisfiable. But then Conflict would be applicable, a contradiction. □

Example 8.14.15 (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 $M \models_{\mathcal{H}} \text{grd}((\mathcal{S}, B, \Pi), N)$. Moreover from M we can generate an interpretation $\mathcal{A}^{\text{BS(LRA)}}$ 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 8.14.16 (Soundness)**.** If a derivation reaches the state $(M; N; U; B; k; \Lambda \parallel \perp \cdot \sigma)$, then N is unsatisfiable.

Proof. All learned clauses are consequences of $N \cup U$, Lemma 8.14.11.3. Furthermore $\text{bgd}(M) \wedge \text{adiff}(B) \wedge \Lambda\sigma$ is satisfiable, Lemma 8.14.11.4a. □

Definition 8.14.17 (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 8.14.18 (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 8.14.19 (SCL(T) Refutation)**.** Given a set of foreground constants $B = [a, b, c]$ and a BS(LRA) constrained clause set $N = \{C_1\colon x = 0 \parallel P(x), C_2\colon y = x + 1 \parallel \neg P(x) \lor P(y), C_3\colon z = 2 \parallel \neg P(z)\}$ the following is a regular derivation

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

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

Example 8.14.20 (SCL(T) Clause learning)**.** Given an initial constant set $B = [a]$ of fresh foreground constants and a BS(LRA) constrained clause set $N = \{C_1\colon x \geq y \parallel \neg P(x, y) \lor Q(z), C_2\colon z = u + v \parallel \neg P(u, v) \lor \neg Q(z), \}$ the following is an example of a regular run

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

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 8.14.21. 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 8.14.22. 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 8.14.21 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 8.14.23 (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:

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

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 8.14.11. For a proof by contradiction, assume there is a $N' \subseteq \text{grd}((\mathcal{S}, B, \Pi), N \cup U)^{\preceq \Lambda_n\sigma \parallel C_n\sigma}$ 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 8.14.18, the rightmost trail literal is resolved away in $\Lambda_n\sigma \parallel C_n\sigma$, Corollary 8.14.22. 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 8.14.23 will change as well. Thus the non-redundancy property of Lemma 8.14.23 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 8.14.23 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 8.14.24 (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

- $l = |\,\text{atoms}(\text{grd}((\mathcal{S}, B, \Pi), N \cup U))|$, $u = 3^l - |\,\text{grd}((\mathcal{S}, B, \Pi), U)|$, 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 $l = |\,\text{atoms}(\text{grd}((\mathcal{S}, B, \Pi), N \cup U))|$ 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 $|\,\text{grd}((\mathcal{S}, B, \Pi), U)|$ 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 8.14.25 (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 [**?**], 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, $\operatorname{codom}(\sigma) = B$, σ grounding for N', and for all $x, y \in \operatorname{dom}(\delta)$ 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 $\quad (M; N; U; B; k; \top) \;\Rightarrow_{\text{SCL(T)}}\; (\epsilon; N; U; B; 0; \top)$

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

Theorem 8.14.26 (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 8.14.25 there exists a a finite set $N' = \{\Lambda_1 \parallel C_1, \dots, \Lambda_n \parallel C_n\}$ of variable renamed copies of clauses from N and a finite set B of fresh constants and a substitution σ, grounding for N' where $\text{codom}(\sigma) = B$ 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 8.14.23. 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. $\square$

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.

Historic and Bibliographic Remarks