Journal of Automated Reasoning (2020) 64:611–640
https://doi.org/10.1007/s10817-020-09546-z

# SPASS-AR: A First-Order Theorem Prover Based on Approximation-Refinement into the Monadic Shallow Linear Fragment

**Andreas Teucke**[1] · **Christoph Weidenbach**[1]

Received: 30 January 2020 / Accepted: 4 February 2020 / Published online: 25 February 2020

**Abstract**
We introduce FO-AR, an approximation-refinement approach for first-order theorem proving based on counterexample-guided abstraction refinement. A given first-order clause set $N$ is transformed into an over-approximation $N'$ in a decidable first-order fragment. That means if $N'$ is satisfiable so is $N$. However, if $N'$ is unsatisfiable, then the approximation provides a lifting terminology for the found refutation which is step-wise transformed into a proof of unsatisfiability for $N$. If this fails, the cause is analyzed to refine the original clause set such that the found refutation is ruled out for the future and the procedure repeats. The target fragment of the transformation is the monadic shallow linear fragment with straight dismatching constraints, which we prove to be decidable via ordered resolution with selection. We further discuss practical aspects of SPASS-AR, a first-order theorem prover implementing FO-AR. We focus in particularly on effective algorithms for lifting and refinement.

**Keywords** First-order theorem proving · Model generation · Decidability · Approximation refinement

## 1 Introduction

The Inst-Gen calculus by Ganzinger and Korovin [11] and its implementation in iProver has shown to be very successful. The calculus is based on a under-approximation—instantiation refinement loop. A given first-order clause set is under-approximated by finite grounding and afterwards a SAT-solver is used to test unsatisfiability. If the ground clause set is unsatisfiable then a refutation for the original clause set is found. If it is satisfiable, the model generated by the SAT-solver is typically not a model for the original clause set. If it is not, it is used to instantiate the original clause such that the found model is ruled out for the future.

In this paper, we define an approximation-based first-order theorem proving approach based on counterexample-guided abstraction refinement that is dual to the Inst-Gen calculus. A given first-order clause set $N$ is step by step transformed into an over-approximation $N_k$ in

✉ Christoph Weidenbach
weidenbach@mpi-inf.mpg.de

1 Max-Planck Institut für Informatik, Saarland Informatics Campus E1 4, 66123 Saarbrücken, Germany

a decidable fragment of first-order logic. That means if $N_k$ is satisfiable so is $N$. However, if $N_k$ is unsatisfiable, then it is not known whether $N$ in unsatisfiable, in general. In that case, the approximation provides a lifting terminology for the found refutation. Each step of the transformation is considered separately to attempt to transform the proof of unsatisfiability for $N_k$ to a proof of unsatisfiability for $N$. If this fails, the cause is analyzed to refine the original clause set such that the found refutation is ruled out for the future and the procedure repeats.

The target fragment of the transformation is the monadic shallow linear fragment with straight dismatching constraints MSL(SDC). It is derived from the monadic shallow linear Horn (MSLH) fragment, which was shown to be decidable in [24]. In addition to the restriction to monadic Horn clauses, the main restriction of the fragment is positive literals of the form $S(f(x_1,\ldots,x_n))$ or $S(x)$ where all $x_i$ are different, i.e., all terms are shallow and linear. The fragment can be finitely saturated by superposition (ordered resolution) where negative literals with non-variable arguments are always selected. As a result, productive clauses with respect to the superposition model operator $\mathbb{I}_N$ have the form $S_1(x_1),\ldots,S_n(x_n) \rightarrow S(f(x_1,\ldots,x_n))$. Therefore, the models of saturated MSLH clause sets can both be represented by tree automata [7] and shallow linear sort theories [9]. The models are typically infinite. The decidability result of MSLH clauses was rediscovered in the context of tree automata research [8] where in addition DEXPTIME-completeness of the MSLH fragment was shown. The fragment was further extended by inequality constraints [13,14] still motivated by security protocol analysis [15]. Although from a complexity point of view, the difference between Horn clause fragments and the respective non-Horn clause fragments is typically reflected by membership in the deterministic vs. the non-deterministic respective complexity fragment, for monadic shallow linear clauses so far there was no decidability result for the non-Horn case.

The results of this paper close this gap. We show the monadic shallow linear non-Horn (MSL) clause fragment to be decidable by superposition (ordered resolution). From a security protocol application point of view, non-Horn clauses enable a natural representation of non-determinism. Our second extension to the fragment are unit clauses with inequations of the form $s \not\approx t$, where $s$ and $t$ are not unifiable. Due to the employed superposition calculus, such inequations do not influence saturation of an MSL clause set, but have an effect on potential models. They can rule out identification of syntactically different ground terms as it is, e.g., desired in the security protocol context for syntactically different messages or nonces. Our third extension to the fragment are straight dismatching constraints. These constraints are incomparable to the inequality constraints mentioned above [13,14]. They do not strictly increase the expressiveness of the MSL theory, but enable up to exponentially more compact saturations. Altogether, the resulting MSL(SDC) fragment is shown to be decidable in Sect. 3.

The introduction of straight dismatching constraints (SDCs) enables an improved refinement step of our approximation refinement calculus [20]. Before, several clauses were needed to rule out a specific instance of a clause in an unsatisfiable core. For example, if due to a linearity approximation from clause $S(x), T(x) \rightarrow S(f(x,x))$ to $S(x), T(x), S(y), T(y) \rightarrow S(f(x,y))$ an instance $\{x \mapsto f(a,x'), y \mapsto f(b,y')\}$ is used in the proof, before [20] several clauses were needed to replace $S(x), T(x) \rightarrow S(f(x,x))$ in a refinement step in order to rule out this instance. With straight dismatching constraints the clause $S(x), T(x) \rightarrow S(f(x,x))$ is replaced by the two clauses $S(f(a,x)), T(f(a,x)) \rightarrow S(f(f(a,x), f(a,x)))$ and $(S(x), T(x) \rightarrow S(f(x,x)); x \neq f(a,y))$. For the improved approximation refinement approach (FO-AR) presented in this paper, any refinement step results in just two clauses, see Sect. 4. The additional expressiveness of constraint clauses comes almost for free, because necessary computations, like, e.g., checking emptiness of SDCs, can all be done in linear-logarithmic time, see Sect. 2.

In addition to the extension of the known MSLH decidability result and the improved approximation refinement calculus FO-AR, we discuss in Sects. 5 and 6 the implementation and potential of the MSL(SDC) fragment in the context of FO-AR, Lemma 4.15, and its prototypical implementation in SPASS-AR (http://www.mpi-inf.mpg.de/fileadmin/inf/rg1/spass-ar.tgz). A particular focus lies in the efficient implementation of lifting and the selection of the refinement.

The theoretical description of lifting based on conflicting cores, i.e., an abstraction of unsatisfiability cores, is intractable in practice because conflicting cores can be exponentially larger than the resolution proofs they are generated from. Therefore, we present a lifting algorithm that avoids the exponential blow-up by directly lifting the resolution refutation represented as a directed acyclic graph (DAG).

For the refinement it is important to note that there can be multiple causes for lifting to fail and certain types of conflicts can be refined more effectively than others. So we introduce a modification of the standard unification algorithm that extracts the specific causes from a given lift-failure and a heuristic to choose which conflict is used for the refinement.

It turns out that for clause sets containing certain structures, FO-AR is superior to ordered resolution/superposition [1] and instance generating methods [11]. The paper ends with a discussion on challenges and future research directions, Sect. 7.

## 2 First-Order Clauses with Straight Dismatching Constraints: MSL(SDC)

We consider a standard first-order language where letters $v, w, x, y, z$ denote variables, $f, g, h$ functions, $a, b, c$ constants, $s, t$ terms, $p, q, r$ positions and Greek letters $\sigma, \tau, \rho, \delta$ are used for substitutions. $S, P, Q, R$ denote predicates, $\approx$ denotes equality, $A, B$ atoms, $E, L$ literals, $C, D$ clauses, $N$ clause sets and $\mathscr{V}$ sets of variables. $\overline{L}$ is the complement of $L$. The signature $\Sigma = (\mathscr{F}, \mathscr{P})$ consists of two disjoint, non-empty, in general infinite sets of function and predicate symbols $\mathscr{F}$ and $\mathscr{P}$, respectively. The set of all *terms* over variables $\mathscr{V}$ is $\mathscr{T}(\mathscr{F}, \mathscr{V})$. If there are no variables, then terms, literals and clauses are called *ground*, respectively. The *depth* of a term $t$ is recursively defined by $\operatorname{depth}(x) = 0$, $\operatorname{depth}(c) = 0$, and $\operatorname{depth}(f(t_1, \dots, t_n)) = 1 + \max_{1 \leq i \leq n}(\operatorname{depth}(t_i))$.

A *substitution* $\sigma$ is denoted by pairs $\{x \mapsto t\}$ and its update at $x$ by $\sigma[x \mapsto t]$. A substitution $\sigma$ is a *grounding* substitution for $\mathscr{V}$ if $x\sigma$ is ground for every variable $x \in \mathscr{V}$. A substitution $\sigma$ is called a *unifier* of the terms $s$ and $t$ if $s\sigma = t\sigma$. $\sigma$ is called a *most general unifier* (mgu) if for every unifier $\delta$ there exists a substitution $\sigma'$ such that $\sigma\sigma' = \delta$.

The set of *free* variables of an atom $A$ (term $t$) is denoted by $\operatorname{vars}(A)$ ($\operatorname{vars}(t)$). A *position* is a sequence of positive integers, where $\varepsilon$ denotes the empty position. As usual $t|_p = s$ denotes the subterm $s$ of $t$ at position $p$, which we also write as $t[s]_p$, and $t[p/s']$ then denotes the replacement of $s$ with $s'$ in $t$ at position $p$. These notions are extended to literals and multiple positions.

A predicate with exactly one argument is called *monadic*. A term is *complex* if it is not a variable and *shallow* if it has at most depth one. It is called *linear* if there are no duplicate variable occurrences. A literal, where every argument term is shallow, is also called *shallow*. A variable and a constant are called *straight*. A term $f(s_1, \dots, s_n)$ is called *straight*, if $s_1, \dots, s_n$ are different variables except for at most one straight term $s_i$.

A *clause* is a multiset of literals which we write as an implication $\Gamma \rightarrow \Delta$ where the atoms in the multiset $\Delta$ (the *succedent*) denote the positive literals and the atoms in the multiset $\Gamma$ (the *antecedent*) the negative literals. We write $\square$ for the empty clause. If $\Gamma$ is empty we omit $\rightarrow$, e.g., we can write $P(x)$ as an alternative of $\rightarrow P(x)$. We abbreviate disjoint set union with sequencing, for example, we write $\Gamma, \Gamma' \rightarrow \Delta, L$ instead of $\Gamma \cup \Gamma' \rightarrow \Delta \cup \{L\}$. A clause $E, E, \Gamma \rightarrow \Delta$ is equivalent to $E, \Gamma \rightarrow \Delta$ and we call them equal *modulo duplicate literal elimination*. If every term in $\Delta$ is shallow, the clause is called *positive shallow*. If all atoms in $\Delta$ are linear and variable disjoint, the clause is called *positive linear*. A clause $\Gamma \rightarrow \Delta$ is called an *MSL* clause, if it is (i) positive shallow and positive linear, (ii) all occurring predicates are monadic, (iii) no equations occur in $\Delta$, and (iv) no equations occur in $\Gamma$ or $\Gamma = \{s \approx t\}$ and $\Delta$ is empty where $s$ and $t$ are not unifiable. *MSL* is the first-order clause fragment consisting of MSL clauses. Clauses $\Gamma, s \approx t \rightarrow \Delta$ where $\Gamma$, $\Delta$ are non-empty and $s, t$ are not unifiable could be added to the MSL fragment without changing any of our results. Considering the superposition calculus, it will select $s \approx t$. Since the two terms are not unifiable, no inference will take place on such a clause and the clause will not contribute to the model operator. In this sense such clauses do not increase the expressiveness of the fragment.

An *atom ordering* $\prec$ is an irreflexive, well-founded, total ordering on ground atoms. It is lifted to literals by representing $A$ and $\neg A$ as multisets $\{A\}$ and $\{A, A\}$, respectively. The multiset extension of the literal ordering induces an ordering on ground clauses. This clause ordering is compatible with the atom ordering: if the maximal atom in $C$ is greater than the maximal atom in $D$ then $D \prec C$. We use $\prec$ simultaneously to denote an atom ordering and its multiset, literal, and clause extensions. For a ground clause set $N$ and clause $C$, the set $N^{\prec C} = \{D \in N \mid D \prec C\}$ denotes the clauses of $N$ smaller than $C$.

An *Herbrand interpretation* $\mathbb{I}$ is a - possibly infinite - set of ground atoms. A ground atom $A$ is called *true* in $\mathbb{I}$ if $A \in \mathbb{I}$ and *false*, otherwise. $\mathbb{I}$ is said to *satisfy* a ground clause $C = \Gamma \rightarrow \Delta$, denoted by $\mathbb{I} \models C$, if $\Delta \cap \mathbb{I} \neq \emptyset$ or $\Gamma \not\subseteq \mathbb{I}$. A non-ground clause $C$ is satisfied by $\mathbb{I}$ if $\mathbb{I} \models C\sigma$ for every grounding substitution $\sigma$. An Herbrand interpretation $\mathbb{I}$ is called a *model* of $N$, $\mathbb{I} \models N$, if $\mathbb{I} \models C$ for every $C \in N$. A model $\mathbb{I}$ of $N$ is considered *minimal* with respect to set inclusion, if there is no model $\mathbb{I}'$ with $\mathbb{I}' \subset \mathbb{I}$ and $\mathbb{I}' \models N$. A set of clauses $N$ is *satisfiable*, if there exists a model that satisfies $N$. Otherwise, the set is *unsatisfiable*.

A inequation $t \neq s$ is an *atomic straight dismatching constraint* if $s$ and $t$ are variable disjoint terms and $s$ is straight. A straight dismatching constraint $\pi = \bigwedge_{i \in I} \; t_i \neq s_i$ is a conjunction of atomic straight dismatching constraints. Given a substitution $\sigma$, $\pi\sigma = \bigwedge_{i \in I} \; t_i\sigma \neq s_i$. $\text{lvar}(\pi) := \bigcup_{i \in I} \text{vars}(t_i)$ are the left-hand variables of $\pi$ and the depth of $\pi$ is the maximal term depth of the $s_i$. A *solution* of $\pi$ is a grounding substitution $\delta$ such that for all $i \in I$, $t_i\delta$ is not an instance of $s_i$, i.e., there exists no $\sigma$ such that $t_i\delta = s_i\sigma$. A dismatching constraint is solvable if it has a solution and unsolvable, otherwise. Whether a straight dismatching constraint is solvable, is decidable in linear-logarithmic time [21]. $\top$ and $\bot$ represent the true and false dismatching constraint, respectively.

We define constraint normalization $\pi{\downarrow}$ as the normal form of the following rewriting rules over straight dismatching constraints.

$$
\begin{aligned}
\pi \wedge f(t_1,\dots,t_n) \neq y &\Rightarrow \bot \\
\pi \wedge f(t_1,\dots,t_n) \neq f(y_1,\dots,y_n) &\Rightarrow \bot \\
\pi \wedge f(t_1,\dots,t_n) \neq f(s_1,\dots,s_n) &\Rightarrow \pi \wedge t_i \neq s_i, \quad \text{if } s_i \text{ is complex} \\
\pi \wedge f(t_1,\dots,t_n) \neq g(s_1,\dots,s_m) &\Rightarrow \pi \\
\pi \wedge x \neq s \wedge x \neq s\sigma &\Rightarrow \pi \wedge x \neq s.
\end{aligned}
$$

Note that $f(t_1,\dots,t_n) \neq f(s_1,\dots,s_n)$ normalizes to $t_i \neq s_i$ for some $i$, where $s_i$ is the one straight complex argument of $f(s_1,\dots,s_n)$. Furthermore, the depth of $\pi\!\downarrow$ is less or equal to the depth of $\pi$ and both have the same solutions.

A pair of a clause and a constraint $(C;\pi)$ is called a *constrained clause*. Given a substitution $\sigma$, $(C;\pi)\sigma = (C\sigma;\pi\sigma)$. A solution $\delta$ of $\pi$ is a solution of $(C;\pi)$ if it is grounding for $C$ and $C\delta$ is then called a ground instance of $(C;\pi)$. $\mathbb{G}((C;\pi))$ is the set of ground instances of $(C;\pi)$. If $\mathbb{G}((C;\pi)) \subseteq \mathbb{G}((C';\pi'))$, then $(C;\pi)$ is an instance of $(C';\pi')$. If $\mathbb{G}((C;\pi)) = \mathbb{G}((C';\pi'))$, then $(C;\pi)$ and $(C';\pi')$ are called variants. An Herbrand interpretation $\mathbb{I}$ satisfies $(C;\pi)$, if $\mathbb{I} \models \mathbb{G}((C;\pi))$. A constrained clause $(C;\pi)$ is called *redundant* in $N$ if for every $D \in \mathbb{G}((C;\pi))$, there exist $D_1,\dots,D_n$ in $\mathbb{G}(N)^{\prec D}$ such that $D_1,\dots,D_n \models D$. A constrained clause $(C';\pi')$ is called a *condensation* of $(C;\pi)$ if $C' \subset C$ and there exists a substitution $\sigma$ such that, $\pi\sigma = \pi'$, $\pi' \subseteq \pi$, and for all $L \in C$ there is an $L' \in C'$ with $L\sigma = L'$. A finite unsatisfiable subset of $\mathbb{G}(N)$ is called an unsatisfiable core of $N$. We don't require unsatisfiable cores to be minimal.

An MSL clause with straight dismatching constraints is called an *MSL(SDC)* clause with MSL(SDC) being the respective first-order fragment. Note that any clause set $N$ can be transformed into an equivalent constrained clause set by changing each $C \in N$ to $(C;\top)$.

## 3 Decidability of the MSL(SDC) Fragment

In the following we will show that the satisfiability of the MSL(SDC) fragment is decidable. For this purpose we will define ordered resolution with selection on constrained clauses [21] and show that with an appropriate ordering and selection function, saturation of an MSL(SDC) clause set terminates.

For the rest of this section we assume an atom ordering $\prec$ such that a literal $\neg Q(s)$ is not greater than a literal $P(t[s]_p)$, where $p \neq \varepsilon$. For example, a KBO where all symbols have weight one has this property.

**Definition 3.1** (*sel*) Let $(C;\pi) = (S_1(t_1),\dots,S_n(t_n) \rightarrow P_1(s_1),\dots,P_m(s_m);\pi)$ be an MSL(SDC) clause. The Selection function sel is defined by $S_i(t_i) \in \operatorname{sel}(C)$ if (1) $t_i$ is not a variable or (2) $t_1,\dots,t_n$ are variables and $t_i \notin \operatorname{vars}(s_1,\dots,s_m)$ or (3) $\{t_1,\dots,t_n\} \subseteq \operatorname{vars}(s_1,\dots,s_m)$ and for some $1 \leq j \leq m$, $s_j = t_i$.

The selection function sel (Definition 3.1) ensures that a clause $\Gamma \rightarrow \Delta$ can only be resolved on a positive literal if $\Gamma$ contains only variables, which also appear in $\Delta$ at a non-top position. For example:

$$
\begin{aligned}
\operatorname{sel}(P(f(x)), P(x), Q(z) \rightarrow Q(x), R(f(y))) &= \{P(f(x))\} \\
\operatorname{sel}(P(x), Q(z) \rightarrow Q(x), R(f(y))) &= \{Q(z)\} \\
\operatorname{sel}(P(x), Q(y) \rightarrow Q(x), R(f(y))) &= \{P(x)\} \\
\operatorname{sel}(P(x), Q(y) \rightarrow Q(f(x)), R(f(y))) &= \emptyset.
\end{aligned}
$$

Note that given an MSL(SDC) clause $(C;\pi) = (S_1(t_1),\dots,S_n(t_n) \rightarrow P_1(s_1),\dots P_m(s_m);\pi)$, if some $S_i(t_i)$ is maximal in $C$, then at least one literal is selected.

**Definition 3.2** A literal $A$ is called *[strictly] maximal* in a constrained clause $(C \vee A; \pi)$ if and only if there exists a solution $\delta$ of $\pi$ such that for all literals $B$ in $C$, $B\delta \preceq A\delta$ $[B\delta \prec A\delta]$.

**Definition 3.3** (*Ordered SDC-resolution with selection*)

$$\frac{(\Gamma_1 \rightarrow \Delta_1, A\ ;\ \pi_1) \qquad (\Gamma_2, B \rightarrow \Delta_2\ ;\ \pi_2)}{((\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma\ ;\ (\pi_1 \wedge \pi_2)\sigma\downarrow)}\ \text{, if}$$

1. $\sigma = \operatorname{mgu}(A, B)$.
2. $(\pi_1 \wedge \pi_2)\sigma\downarrow$ is solvable.
3. $A\sigma$ is strictly maximal in $(\Gamma_1 \rightarrow \Delta_1, A; \pi_1)\sigma$ and $\operatorname{sel}(\Gamma_1 \rightarrow \Delta_1, A) = \emptyset$.
4. $B \in \operatorname{sel}(\Gamma_2, B \rightarrow \Delta_2)$ or $\operatorname{sel}(\Gamma_2, B \rightarrow \Delta_2) = \emptyset$ and $\neg B\sigma$ maximal in $(\Gamma_2, B \rightarrow \Delta_2; \pi_2)\sigma$.

**Definition 3.4** (*Ordered SDC-factoring with selection*)

$$\frac{(\Gamma \rightarrow \Delta, A, B\ ;\ \pi)}{((\Gamma \rightarrow \Delta, A)\sigma; \pi\sigma\downarrow)}\ \text{, if}$$

1. $\sigma = \operatorname{mgu}(A, B)$.
2. $\operatorname{sel}(\Gamma \rightarrow \Delta, A, B) = \emptyset$.
3. $A\sigma$ is maximal in $(\Gamma \rightarrow \Delta, A, B; \pi)\sigma$.
4. $\pi\sigma\downarrow$ is solvable.

Note that while the above rules do not operate on equations, we can actually allow unit clauses that consist of non-unifiable inequations, i.e., clauses $s \approx t \rightarrow$ where $s$ and $t$ are not unifiable. There are no potential superposition inferences on such clauses as long as there are no positive equations. So, resolution and factoring suffice for completeness. Nevertheless, clauses such as $s \approx t \rightarrow$ affect the models of satisfiable problems.

**Lemma 3.5** (Soundness) *Ordered SDC-Resolution and ordered SDC-Factoring with selection are sound.*

***Proof*** Let the clause $(\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma\delta$ be a ground instance of the resolvent clause $((\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma; (\pi_1 \wedge \pi_2)\sigma)$. Then, $\delta$ is a solution of $(\pi_1 \wedge \pi_2)\sigma$ and $\sigma\delta$ is a solution of $\pi_1$ and $\pi_2$. Hence, $(\Gamma_1 \rightarrow \Delta_1, A)\sigma\delta$ and $(\Gamma_2, B \rightarrow \Delta_2)\sigma\delta$ are ground instances of $(\Gamma_1 \rightarrow \Delta_1, A; \pi_1)$ and $(\Gamma_2, B \rightarrow \Delta_2; \pi_2)$, respectively. Because $A\sigma\delta = B\sigma\delta$, if $(\Gamma_1 \rightarrow \Delta_1, A)\sigma\delta$ and $(\Gamma_2, B \rightarrow \Delta_2)\sigma\delta$ are satisfied, then $(\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma\delta$ is also satisfied. Therefore, ordered SDC-Resolution with selection is sound.

The proof is analogous for ordered SDC-Factoring with selection. □

**Definition 3.6** (*Saturation*) A constrained clause set $N$ is called saturated up to redundancy, if for every inference between clauses in $N$ the result $(R; \pi)$ is either redundant in $N$ or $\mathbb{G}((R; \pi)) \subseteq \mathbb{G}(N)$.

Note that our redundancy notion includes condensation and the condition $\mathbb{G}((R; \pi)) \subseteq \mathbb{G}(N)$ allows ignoring variants of clauses.

**Definition 3.7** (*Partial minimal model construction*) Given a constrained clause set $N$, an ordering $\prec$ and the selection function sel, we construct an Herbrand interpretation $\mathbb{I}_N$ for $N$, called a partial model, inductively as follows:

$$\mathbb{I}_C := \bigcup_{D \prec C}^{D \in \mathbb{G}(N)} \delta_D, \text{ where } C \in \mathbb{G}(N)$$

$$\delta_D := \begin{cases} \{A\} & \text{if } D = \Gamma \to \Delta, A \\ & A \text{ strictly maximal, } \mathrm{sel}(D) = \emptyset \text{ and } \mathbb{I}_D \not\models D \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathbb{I}_N := \bigcup_{C \in \mathbb{G}(N)} \delta_C$$

Clauses $D$ with $\delta_D \neq \emptyset$ are called productive.

**Lemma 3.8** (Ordered SDC-resolution completeness) *Let $N$ be a constrained clause set saturated up to redundancy by ordered SDC-Resolution and SDC-Factoring with selection. Then $N$ is unsatisfiable, if and only if $\square \in \mathbb{G}(N)$. If $\square \notin \mathbb{G}(N)$ then $\mathbb{I}_N \models N$.*

***Proof*** Since $\mathbb{I}_N$ is defined on the ground clauses of $N$, the proof is analogous to the standard proof of resolution completeness. □

**Lemma 3.9** *Let $N$ be a set of MSL(SDC) clauses without variants or uncondensed clauses over a finite signature $\Sigma$. $N$ is finite if there exists an integer $d$ such that for every $(C;\pi) \in N$, depth($\pi$)$\leq d$ and*

(1) $C = S_1(x_1), \ldots, S_n(x_n), S'_1(t), \ldots, S'_m(t) \to \Delta$ *or*
(2) $C = S_1(x_1), \ldots, S_n(x_n), S'_1(t), \ldots, S'_m(t) \to S(t), \Delta$ *with $t$ shallow and linear, and* $\mathrm{vars}(t) \cap \mathrm{vars}(\Delta) = \emptyset$.

***Proof*** Let $(C;\pi) \in N$. $(C;\pi)$ can be separated into variable disjoint components $(\Gamma_1, \ldots, \Gamma_n \to \Delta_1, \ldots, \Delta_n; \pi_1 \wedge \ldots \wedge \pi_n)$, where $|\Delta_i| \leq 1$ and $\mathrm{lvar}(\pi_i) \subseteq \mathrm{vars}(\Gamma_i \to \Delta_i)$. For each positive literal $P(s) \in \Delta$ there is a fragment

$$(A)\ \ (S_1(x_1), \ldots, S_k(x_k) \to P(s); \pi')$$

with $\{x_1, \ldots, x_k\} \subseteq \mathrm{vars}(s)$. If $m > 0$, there is another fragment

$$(B)\ \ (S_1(x_1), \ldots, S_k(x_k), S'_1(t), \ldots, S'_m(t) \to; \pi') \text{ or}$$
$$(C)\ \ (S_1(x_1), \ldots, S_k(x_k), S'_1(t), \ldots, S'_m(t) \to S(t); \pi')$$

with $\{x_1, \ldots, x_k\} \subseteq \mathrm{vars}(t)$, respectively. Lastly, for each variable $x \in \mathrm{vars}(C)$ with $x \notin \mathrm{vars}(t) \cup \mathrm{vars}(\Delta)$ there is a fragment

$$(D)\ \ (S_1(x), \ldots, S_k(x) \to; \pi').$$

Since there are only finitely many terms $s$ with depth($s$)$\leq d$ modulo renaming, there are only finitely many atomic constraints $x \neq s$ for a given variable $x$ different up to renaming $s$. Thus, a normal constraint can only contain finitely many combinations of sub-constraints $\bigwedge_{i \in \mathscr{I}} x \neq s_i$ without some $s_i$ being an instance of another $s_j$. Therefore, for a fixed set of variables $x_1, \ldots, x_k$, there are only finitely many constraints $\pi = \bigwedge_{i \in \mathscr{I}} z_i \neq s_i$ with $\mathrm{lvar}(\pi) \subseteq \{x_1, \ldots, x_k\}$ up to variants.

Since the number of predicates, function symbols, and their ranks is finite, the number of possible shallow and linear atoms $S(t)$ different up to variants is finite. For a given shallow and linear $t$, there exist only finitely many clauses of the form $(S_1(t), \ldots, S_n(t) \to S(t); \pi)$ or $(S_1(t), \ldots, S_n(t) \to; \pi)$ with $\mathrm{lvar}(\pi) \subseteq \mathrm{vars}(t)$ modulo condensation and variants. For a fixed set of variables $x_1, \ldots, x_k$, there exist only finitely many clauses of the form $(S_1(y_1), \ldots, S_k(y_l) \rightarrow; \pi)$ modulo condensation and variants where $\text{lvar}(\pi) \subseteq \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}$. Therefore, there are only finitely many distinct clauses of each form (A)-(D) without variants or condensations.

If in the clause $(C; \pi) = (\Gamma_1, \ldots, \Gamma_n \rightarrow \Delta_1, \ldots, \Delta_n; \pi_1 \wedge \ldots \wedge \pi_n)$ for some $i \neq j$, $(\Gamma_i \rightarrow \Delta_i; \pi_i)$ is a variant of $(\Gamma_j \rightarrow \Delta_j; \pi_j)$, then $(C; \pi)$ has a condensation and is therefore not part of $N$. Hence, there can be only finitely many different $(C; \pi)$ without variants or condensations and thus $N$ is finite. □

**Lemma 3.10** (Finite saturation) *Let $N$ be an MSL(SDC) clause set. Then $N$ can be finitely saturated up to redundancy by ordered SDC-Resolution with selection function* sel.

***Proof*** The general idea is that given the way sel is defined the clauses involved in ordered SDC-Resolution and SDC-Factoring can only fall into certain patterns. Any result of such inferences then is either strictly smaller than one of its parents by some terminating measure or falls into a set of clauses that is bounded by Lemma 3.9. Thus, there can be only finitely many inferences before $N$ is saturated.

Let $d$ be an upper bound on the depth of constraints found in $N$ and $\Sigma$ be the finite signature consisting of the function and predicate symbols occurring in $N$. Let $(\Gamma_1 \rightarrow \Delta_1, S(t); \pi_1)$ and $(\Gamma_2, S(t') \rightarrow \Delta_2; \pi_2)$ be clauses in $N$ where ordered SDC-Resolution applies with the most general unifier $\sigma$ of $S(t)$ and $S(t')$ and resolvent $R = ((\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma; (\pi_1 \wedge \pi_2)\sigma\downarrow)$.

Because no literal is selected by sel, $\Gamma_1 \rightarrow \Delta_1, S(t)$ can match only one of two patterns:

$$(A)\;\; S_1(x_1), \ldots, S_n(x_n) \rightarrow S(f(y_1, \ldots, y_k)), \Delta$$

where $t = f(y_1, \ldots, y_k)$ and $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_k\} \cup \text{vars}(\Delta)$ and

$$(B)\;\; S_1(x_1), \ldots, S_n(x_n) \rightarrow S(y), \Delta$$

where $t = y$ and $x_1, \ldots, x_n$ are variables in $\text{vars}(\Delta)$, i.e., $y$ occurs only once.
The literal $S(t')$ is selected by sel in $\Gamma_2, S(t') \rightarrow \Delta_2$, and therefore $\Gamma_2, S(t') \rightarrow \Delta_2$ can match only one of the following three patterns:

(1) $S(f(t_1, \ldots, t_k)), \Gamma' \rightarrow \Delta'$.
(2) $S(y'), \Gamma' \rightarrow \Delta'$ where $\Gamma'$ has no function terms and $y \notin \text{vars}(\Delta')$.
(3) $S(y'), \Gamma' \rightarrow S'(y'), \Delta'$ where $\Gamma'$ has no function terms.

This means that the clausal part $(\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma$ of $R$ has one of six forms:

$$(A1)\;\; S_1(x_1)\sigma, \ldots, S_n(x_n)\sigma, \Gamma' \rightarrow \Delta, \Delta' \text{ with } \sigma = \{y_1 \mapsto t_1, \ldots\}.$$

$\Delta\sigma = \Delta$ because $S(f(y_1, \ldots, y_k))$ and $\Delta$ do not share variables.

$$(B1)\;\; S_1(x_1), \ldots, S_n(x_n), \Gamma' \rightarrow \Delta, \Delta'.$$

The substitution $\{y \mapsto f(t_1, \ldots, t_k)\}$ is irrelevant since $S(y)$ is the only literal with variable $y$.

$$(A2)\;\; S_1(x_1), \ldots, S_n(x_n), \Gamma'\tau \rightarrow \Delta, \Delta' \text{ with } \tau = \{y' \mapsto f(y_1, \ldots, y_k)\}.$$

$\Delta'\tau = \Delta'$ because $y' \notin \text{vars}(\Delta')$.

$(B2)$ $S_1(x_1), \ldots, S_n(x_n), \Gamma' \rightarrow \Delta, \Delta'$.
$(A3)$ $S_1(x_1), \ldots, S_n(x_n), \Gamma'\tau \rightarrow S'(f(y_1, \ldots, y_k)), \Delta, \Delta'$ with $\tau = \{y \mapsto f(y_1, \ldots, y_k)\}$.

$\Delta'\tau = \Delta'$ because $y' \notin \text{vars}(\Delta')$.

$$(B3)\;\; S_1(x_1), \ldots, S_n(x_n), \Gamma' \rightarrow S'(y'), \Delta, \Delta'.$$

In the constraint $(\pi_1 \wedge \pi_2)\sigma\!\downarrow$ the maximal depth of the sub-constraints is less or equal to the maximal depth of $\pi_1$ or $\pi_2$. Hence, $d$ is also an upper bound on the constraint of the resolvent. In each case, the resolvent is again an MSL(SDC) clause.

In the first and second case, the multiset of term depths of the negative literals in $R$ is strictly smaller than for the right parent. In both, the $\Gamma$ is the same between the right parent and the resolvent. Only the $f(t_1, \ldots, t_k)$ term is replaced by $x_1\sigma, \ldots, x_n\sigma$ and $x_1, \ldots, x_n$ respectively. In the first case, the depth of the $x_i\sigma$ is either zero if $x_i \notin \{y_1, \ldots, y_k\}$ or at least one less than $f(t_1, \ldots, t_k)$ since $x_i\sigma = t_i$. In the second case, the $x_i$ have depth zero which is strictly smaller than the depth of $f(t_1, \ldots, t_k)$. Since the multiset ordering on natural numbers is terminating, the first and second case can only be applied finitely many times by ordered SDC-Resolution.

In the third to sixth cases $R$ is either $(S_1(x_1), \ldots, S_l(x_l), S'_1(t), \ldots, S'_m(t) \rightarrow \Delta; \pi)$ or $(S_1(x_1), \ldots, S_l(x_l), S'_1(t), \ldots, S'_m(t) \rightarrow S(t)), \Delta; \pi)$ with $t = f(y_1, \ldots, y_k)$. By Lemma 3.8, there are only finitely many such clauses after condensation and removal of variants. Therefore, these four cases can apply only finitely many times during saturation.

The proof is analogous for ordered SDC-Factoring. □

**Theorem 3.11** *Satisfiability of the MSL(SDC) first-order fragment is decidable.*

***Proof*** Follows from Lemmas 3.10 and 3.8. □

## 4 Approximation and Refinement

In the following, we show how decidability of the MSL(SDC) fragment can be used to improve the approximation refinement calculus presented in [20].

Our approach is based on a counter-example guided abstraction refinement (CEGAR) idea. The procedure loops trough four steps: approximation, testing (un)satisfiability, lifting, and refinement. The approximation step transforms any first-order logic clause set into the decidable MSL(SDC) fragment while preserving unsatisfiability. The second step employs the decidability result for MSL(SDC), Sect. 3, to test satisfiability of the approximated clause set. If the approximation is satisfiable, the original problem is satisfiable as well and we are done. Otherwise, the third step, lifting, tests whether the proof of unsatisfiability found for the approximated clause set can be lifted to a proof of the original clause set. If so, the original clause set is unsatisfiable and we are again done. If not, we extract a cause for the lifting failure that always amounts to two different instantiations of the same variable in a clause from the original clause set. This is resolved by the fourth step, the refinement. The crucial clause in the original problem is replaced and instantiated in a satisfiability preserving way such that the different instantiations do not reoccur any more in subsequent iterations of the loop.

As mentioned before, our motivation to use dismatching constraints is that for an unconstrained clause the refinement adds quadratically many new clauses to the clause set. In contrast, with constrained clauses the same can be accomplished with adding just a single new clause. This extension is rather simple as constraints are treated the same as the antecedent literals in the clause. Furthermore we present refinement as a separate transformation rule.

The second change compared to the previous version is the removal of the Horn approximation rule, where we have now shown in Sect. 3 that a restriction to Horn clauses is not required for decidability any more. Instead, the linear and shallow approximations are extended to apply to non-Horn clauses instead.

The approximation consists of individual transformation rules $N \Rightarrow N'$ that are non-deterministically applied. They transform a clause that is not in the MSL(SDC) fragment in finite steps into MSL(SDC) clauses. Each specific property of MSL(SDC) clauses, i.e, monadic predicates, shallow and linear positive literals, is generated by a corresponding rule: the Monadic transformation encodes non-Monadic predicates as functions, the shallow transformation extracts non-shallow subterms by introducing fresh predicates and the linear transformation renames non-linear variable occurrences.

Starting from a constrained clause set $N$ the transformation is parameterized by a single monadic projection predicate $T$, fresh to $N$ and for each non-monadic predicate $P$ a separate projection function $f_P$ fresh to $N$. The clauses in $N$ are called the original clauses while the clauses in $N'$ are the approximated clauses. We assume all clauses in $N$ to be variable disjoint.

**Definition 4.1** Given a predicate $P$, projection predicate $T$, and projection function $f_P$, define the injective function $\mu_P^T(P(\bar{t})) := T(f_p(\bar{t}))$ and $\mu_P^T(Q(\bar{s})) := Q(\bar{s})$ for $P \neq Q$. The function is extended to [constrained] clauses, clause sets and Herbrand interpretations. Given a signature $\Sigma$ with non-monadic predicates $P_1, \ldots, P_n$, define $\mu_\Sigma^T(N) := \mu_{P_1}^T(\ldots(\mu_{P_n}^T(N))\ldots)$ and $\mu_\Sigma^T(\mathbb{I}) := \mu_{P_1}^T(\ldots(\mu_{P_n}^T(\mathbb{I}))\ldots)$.

**Monadic** $\quad N \Rightarrow_{\text{MO}} \mu_P^T(N)$
provided $P$ is a non-monadic predicate in the signature of $N$.

**Shallow** $\quad N \mathbin{\dot\cup} \{(\Gamma \rightarrow E[s]_p, \Delta; \pi)\}$
$\qquad\Rightarrow_{\text{SH}} N \cup \{(S(x), \Gamma_l \rightarrow E[p/x], \Delta_l; \pi); (\Gamma_r \rightarrow S(s), \Delta_r; \pi)\}$
provided $s$ is complex, $|p| = 2$, $x$ and $S$ fresh, $\Gamma_l\{x \mapsto s\} \cup \Gamma_r = \Gamma$, $\Delta_l \cup \Delta_r = \Delta$, $\{Q(y) \in \Gamma \mid y \in \text{vars}(E[p/x], \Delta_l)\} \subseteq \Gamma_l$, $\{Q(y) \in \Gamma \mid y \in \text{vars}(s, \Delta_r)\} \subseteq \Gamma_r$.

**Linear1** $\quad N \mathbin{\dot\cup} \{(\Gamma \rightarrow \Delta, E'[x]_p, E[x]_q; \pi)\}$
$\qquad\Rightarrow_{\text{LI}} N \cup \{(\Gamma\sigma, \Gamma \rightarrow \Delta, E'[x]_p, E[q/x']; \pi \wedge \pi\sigma)\}$
provided $x'$ is fresh and $\sigma = \{x \mapsto x'\}$.

**Linear2** $\quad N \mathbin{\dot\cup} \{(\Gamma \rightarrow \Delta, E[x]_{p,q}; \pi)\}$
$\qquad\Rightarrow_{\text{LI}} N \cup \{(\Gamma\sigma, \Gamma \rightarrow \Delta, E[q/x']; \pi \wedge \pi\sigma)\}$
provided $x'$ is fresh, $p \neq q$ and $\sigma = \{x \mapsto x'\}$.

**Refinement** $\quad N \mathbin{\dot\cup} \{(C, \pi)\} \Rightarrow_{\text{Ref}} N \cup \{(C; \pi \wedge x \neq t), (C; \pi)\{x \mapsto t\}\}$
provided $x \in \text{vars}(C)$, $t$ straight and $\text{vars}(t) \cap \text{vars}((C, \pi)) = \emptyset$.

Note that variables are not renamed unless explicitly stated in the rule. This means that original clauses and their approximated counterparts share variable names. We use this to trace the origin of variables in the approximation.

The refinement transformation $\Rightarrow_{\text{Ref}}$ is not needed to eventually generate MSL(SDC) clauses, but can be used to achieve a more fine-grained approximation of $N$, see below.

In the shallow transformation, $\Gamma$ and $\Delta$ are separated into $\Gamma_l$, $\Gamma_r$, $\Delta_l$, and $\Delta_r$, respectively. The separation can be almost arbitrarily chosen as long as no atom from $\Gamma$, $\Delta$ is skipped. However, the goal is to minimize the set of shared variables, i.e., the variables of $(\Gamma \rightarrow E[s]_p, \Delta; \pi)$ that are inherited by both approximation clauses, $\text{vars}(\Gamma_r, s, \Delta_r) \cap \text{vars}(\Gamma_l, E[p/x], \Delta_l)$. If there are no shared variables, the shallow transformation is satisfiability equivalent. The conditions on $\Gamma_l$ and $\Gamma_r$ ensure that $S(x)$ atoms are not separated from the respective positive occurrence of $x$ in subsequent shallow transformation applications.

Consider the clause $Q(f(x), y) \rightarrow P(g(f(x), y))$. The shallow transformation $S(x'), Q(f(x), y) \rightarrow P(g(x', y)); S(f(x))$ is not satisfiability equivalent – nor with any alternative partitioning of $\Gamma$. However, by replacing the occurrence of the extraction term $f(x)$ in $Q(f(x), y)$ with the fresh variable $x'$, the approximation $S(x'), Q(x', y) \rightarrow P(g(x', y)); S(f(x))$ is satisfiability equivalent. Therefore, we allow the extraction of $s$ from the terms in $\Gamma_l$ and require $\Gamma_l\{x \mapsto s\} \cup \Gamma_r = \Gamma$.

While in theory, a literal from $\Delta$ can be added to both $\Delta_l$ and $\Delta_r$, in the implementation, $\Delta_l$ and $\Delta_r$ are a partition of $\Delta$. Otherwise, we run the risk of producing an exponential number of approximation clauses because the duplicated literal might trigger further Shallow transformations for both approximation clauses instead of just one of them.

We consider Linear 1 and Linear 2 as two cases of the same linear transformation rule. Their only difference is whether the two occurrences of $x$ are in the same literal or not. The duplication of literals and constraints in $\Gamma$ and $\pi$ is not needed if $x$ does not occur in $\Gamma$ or $\pi$. The Linear transformation also carries the risk of exponentially increasing the size of clauses. For example, the clause $P(x, y) \rightarrow Q(x, \dots, x, y, \dots, y)$ where $x$ and $y$ appear $n$ and $m$ times respectively has a linear approximation with $n \cdot m$ negative literals. In these extreme cases, we pre-process such clauses via a renaming. For the example, that means replacing the clause with $T(f(x_1, \dots, x_n, y)) \rightarrow Q(x_1, \dots, x_n, y, \dots, y)$ and $P(x, y) \rightarrow T(f(x, \dots, x, y))$ where $f$ is a fresh function symbol.

Further, consider a linear transformation $N \cup \{(C; \pi)\} \Rightarrow_{\text{LI}} N \cup \{(C_a; \pi_a)\}$, where a fresh variable $x'$ replaces an occurrence of a non-linear variable $x$ in $(C; \pi)$. Then, $(C_a; \pi_a)\{x' \mapsto x\}$ is equal to $(C; \pi)$ modulo duplicate literal elimination. A similar property can be observed of a resolvent of $(C_l; \pi)$ and $(C_r; \pi)$ resulting from a shallow transformation $N \cup \{(C; \pi)\} \Rightarrow_{\text{SH}} N \cup \{(C_l; \pi), (C_r; \pi)\}$. Note that by construction, $(C_l; \pi)$ and $(C_r; \pi)$ are not necessarily variable disjoint. To simulate standard resolution, we need to rename at least the shared variables in one of them.

**Definition 4.2** ($\Rightarrow_{\text{AP}}$) We define $\Rightarrow_{\text{AP}}$ as the priority rewrite system [3] consisting of $\Rightarrow_{\text{Ref}}$, $\Rightarrow_{\text{MO}}$, $\Rightarrow_{\text{SH}}$ and $\Rightarrow_{\text{LI}}$ with priority $\Rightarrow_{\text{Ref}} > \Rightarrow_{\text{MO}} > \Rightarrow_{\text{SH}} > \Rightarrow_{\text{LI}}$, where $\Rightarrow_{\text{Ref}}$ is only applied finitely many times.

**Lemma 4.3** ($\Rightarrow_{\text{AP}}$ is a terminating over-approximation) *(i) $\Rightarrow^*_{\text{AP}}$ terminates, (ii) if $N \Rightarrow_{\text{AP}} N'$ and $N'$ is satisfiable, then $N$ is also satisfiable.*

***Proof*** (i) The transformations can be considered sequentially, because of the imposed rule priority. There are, by definition, only finitely many refinements at the beginning of an approximation $\Rightarrow^*_{\text{AP}}$. The Monadic transformation strictly reduces the number of non-monadic atoms. The Shallow transformation strictly reduces the multiset of term depths of the newly introduced clauses compared to the removed approximated clause. The Linear transformation strictly reduces the number of duplicate variable occurrences in positive literals. Hence $\Rightarrow_{\text{AP}}$ terminates.

(ii) Let $N \cup \{(C; \pi)\} \Rightarrow_{\text{LI}} N \cup \{(C_a; \pi_a)\}$ where an occurrence of a variable $x$ in $(C; \pi)$ is replaced by a fresh $x'$. As $(C_a; \pi_a)\{x' \mapsto x\}$ is equal to $(C; \pi)$ modulo duplicate literal elimination, $\mathbb{I} \models (C; \pi)$ if $\mathbb{I} \models (C_a; \pi_a)$. Therefore, the Linear transformation is an over-approximation.

Let $N \cup \{(C; \pi)\} \Rightarrow_{\text{SH}} N \cup \{(C_l; \pi_l), (C_r; \pi_r)\}$ and $(C_a; \pi_a)$ be the shallow $\rho$-resolvent. As $(C_a; \pi_a)\rho^{-1}$ equals $(C; \pi)$ modulo duplicate literal elimination, $\mathbb{I} \models (C; \pi)$ if $\mathbb{I} \models (C_l; \pi_l), (C_r; \pi_r)$. Therefore, the Shallow transformation is an over-approximation.

Let $N \Rightarrow_{\text{MO}} \mu_P(N) = N'$. Then, $N = \mu_P^{-1}(N')$. Let $\mathbb{I}$ be a model of $N'$ and $(C; \pi) \in N$. Since $\mu_P((C; \pi)) \in N'$ , $\mathbb{I} \models \mu_P((C; \pi))$ and thus, $\mu_P^{-1}(\mathbb{I}) \models (C; \pi)$. Hence, $\mu_P^{-1}(\mathbb{I})$ is a model of $N$. Therefore, the Monadic transformation is an (non-strict) over-approximation. Actually, it is even a satisfiability equivalent transformation.

Let $N \cup \{(C;\pi)\} \Rightarrow_{\text{Ref}} N \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\}$. Let $C\delta \in \mathbb{G}((C;\pi))$. If $x\delta$ is not an instance of $t$, then $\delta$ is a solution of $\pi \wedge x \neq t$ and $C\delta \in \mathbb{G}((C;\pi \wedge x \neq t))$. Otherwise, $\delta = \{x \mapsto t\}\delta'$ for some substitution $\delta'$. Then, $\delta$ is a solution of $\pi\{x \mapsto t\}$ and thus, $C\delta = C\{x \mapsto t\}\delta' \in \mathbb{G}((C\{x \mapsto t\};\pi\{x \mapsto t\}))$. Hence, $\mathbb{G}((C;\pi)) \subseteq \mathbb{G}((C;\pi \wedge x \neq t)) \cup \mathbb{G}((C;\pi)\{x \mapsto t\})$. Therefore, if $\mathbb{I}$ is a model of $N \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\}$, then $\mathbb{I}$ is also a model of $N \cup \{(C;\pi)\}$. □

Note that $\Rightarrow_{\text{Ref}}$ and $\Rightarrow_{\text{MO}}$ are also satisfiability preserving transformations.

**Corollary 4.4** *If $N \Rightarrow^*_{\text{AP}} N'$ and $N'$ is satisfied by a model $\mathbb{I}$, then $\mu^{-1}_{\Sigma}(\mathbb{I})$ is a model of $N$.*

On the basis of $\Rightarrow_{\text{AP}}$ we define an *ancestor relation* $\Rightarrow_{\text{A}}$ that relates clauses, literal occurrences, and variables with respect to approximation. For example, for a Shallow transformation of the clause $C = P(f(f(x)))$ in a clause set $N$ into $C_l = S(x') \rightarrow P(f(x'))$ and $C_r = S(f(x))$ in the approximation $N'$, $\Rightarrow_{\text{A}}$ relates $[C, N] \Rightarrow^*_{\text{A}} [C_l, N']$ and $[C, N] \Rightarrow^*_{\text{A}} [C_r, N']$. This is further extended to variables, positions and literals; e.g., $[x, C, N] \Rightarrow^*_{\text{A}} [x, C_r, N']$, $[x, C, N] \Rightarrow^*_{\text{A}} [x', C_l, N']$, $[1, P(f(f(x))), C, N] \Rightarrow^*_{\text{A}} [1, P(f(x')), C_l, N']$, $[1.1, P(f(f(x))), C, N] \Rightarrow^*_{\text{A}} [1, S(f(x)), C_r, N']$, and $[1.1, P(f(f(x))), C, N] \Rightarrow^*_{\text{A}} [\varepsilon, S(f(x)), C_r, N']$. In essence, the ancestor relation tracks the origin of clauses, variables, and symbols in the approximation. The full definition of $\Rightarrow_{\text{A}}$ itself is found in [22], the definition of a parent clause below shows the relation in the case of clauses.

**Definition 4.5** (*Parent clause*) For an approximation step $N \Rightarrow_{\text{AP}} N'$ and two clauses $(C;\pi) \in N$ and $(C';\pi') \in N'$, define $[(C;\pi), N] \Rightarrow_{\text{A}} [(C';\pi'), N']$ expressing that $(C;\pi)$ in $N$ is the *parent clause* of $(C';\pi')$ in $N'$:
If $N \Rightarrow_{\text{MO}} \mu^T_P(N)$, then $[(C;\pi), N] \Rightarrow_{\text{A}} [\mu^T_P((C;\pi)), \mu^T_P(N)]$ for all $(C;\pi) \in N$.
If $N = N'' \cup \{(C;\pi)\} \Rightarrow_{\text{SH}} N'' \cup \{(C_l;\pi_l), (C_r;\pi_r)\} = N'$, then $[(D,\pi'), N] \Rightarrow_{\text{A}} [(D,\pi'), N']$ for all $(D,\pi') \in N''$ and $[(C,\pi), N] \Rightarrow_{\text{A}} [(C_l;\pi_l), N']$, $[(C,\pi), N] \Rightarrow_{\text{A}} [(C_r;\pi_r), N']$ and $[(C,\pi), N] \Rightarrow_{\text{A}} [(C_a;\pi_a), N']$ for any shallow resolvent $(C_a;\pi_a)$.
If $N = N'' \cup \{(C;\pi)\} \Rightarrow_{\text{LI}} N'' \cup \{(C_a;\pi_a)\} = N'$, then $[(D,\pi'), N] \Rightarrow_{\text{A}} [(D,\pi'), N']$ for all $(D,\pi') \in N''$ and $[(C,\pi), N] \Rightarrow_{\text{A}} [(C_a,\pi_a), N']$.
If $N = N'' \cup \{(C;\pi)\} \Rightarrow_{\text{Ref}} N'' \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\} = N'$, then $[(D,\pi'), N] \Rightarrow_{\text{A}} [(D,\pi'), N']$ for all $(D,\pi') \in N''$, $[(C,\pi), N] \Rightarrow_{\text{A}} [(C;\pi \wedge x \neq t), N']$ and $[(C,\pi), N] \Rightarrow_{\text{A}} [(C;\pi)\{x \mapsto t\}, N']$.

*Lifting* The over-approximation of a clause set $N$ can introduce resolution refutations that have no corresponding equivalent in $N$ which we consider a lifting failure. Compared to our previous calculus [20], the lifting process is identical with the exception that there is no case for the removed Horn transformation. Since the conflicting cores are being superseded in the implementation by an algorithm on DAGs, we can simplify their definition.

**Definition 4.6** (*Conflicting core*) A finite set of ground clauses $N^{\perp}$ are a conflicting core of $N$ if $N^{\perp}$ is unsatisfiable and for every $C \in N^{\perp}$ there is a clause $(C',\pi') \in N$ such that $C \in \mathbb{G}((C';\pi'))$ modulo duplicate literal elimination. We call $C$ a conflict-instance of $(C';\pi')$ in $N^{\perp}$. A conflicting core $N^{\perp}$ of $N$ is minimal if for any subset $M \subsetneq N^{\perp}$, $M$ is not a conflicting core of $N$.

We discuss the lifting and the corresponding refinements only for the linear and shallow case because lifting the satisfiability equivalent monadic and refinement transformations always succeeds.

**Lemma 4.7** (Linear lifting) *Let $N \cup \{(C;\pi)\} \Rightarrow_{\text{LI}} N \cup \{(C_a;\pi_a)\}$ and $N^\perp$ be a conflicting core of $N \cup \{(C_a;\pi_a)\}$. Let $C_1, \ldots, C_m$ be all conflict-instances of $(C_a;\pi_a)$ in $N^\perp$. If each $C_i$ is an instance of $(C;\pi)$ modulo duplicate literal elimination, then $N^\perp$ is a conflicting core of $N \cup \{(C;\pi)\}$.*

***Proof*** Follows trivially from the definition of conflicting cores, i.e., $N^\perp$ is already a conflicting core and for any clause $C^\perp \in N^\perp$ that is not among $C_1, \ldots, C_m$, there is a clause $(D, \pi') \in N$ such that $C^\perp$ is a ground instance of $(D, \pi')$ modulo duplicate literal elimination. □

**Lemma 4.8** (Shallow lifting) *Let $N \cup \{(C;\pi)\} \Rightarrow_{\text{SH}} N \cup \{(C_l;\pi_l), (C_r;\pi_r)\}$ with the fresh predicate $S$ and $N^\perp$ be a conflicting core of $N \cup \{(C_l;\pi_l), (C_r;\pi_r)\}$. Let $N_S$ be the set of all resolvents of clauses in $N^\perp$ on $S$-atoms. If each clause $C_a \in N_S$ is an instance of $(C;\pi)$ modulo duplicate literal elimination, then $\{D \in N^\perp \mid \text{no } S\text{-atom in} D\} \cup N_S$ is a conflicting core of $N \cup \{(C;\pi)\}$.*

***Proof*** Let $N'^\perp = \{D \in N^\perp \mid \text{ no S-atom in } D\} \cup N_S$ and $\mathbb{I}$ an Herbrand interpretation. Then, there exists a $C^\perp \in N^\perp$ such that $\mathbb{I} \not\models C^\perp$. If $C^\perp$ does not contain an $S$-atom, then $C^\perp \in N'^\perp$. Thus, $\mathbb{I} \not\models N'^\perp$. Otherwise, at least one clause with an $S$-atom is false under $\mathbb{I}$ in $N^\perp$. By construction, any such clause is an conflict-instance of either $C_l$ or $C_r$. Let $C_l\tau_1, \ldots, C_l\tau_m$ and $C_r\rho_1, \ldots, C_r\rho_n$ be all clauses in $N^\perp$ that are false under $\mathbb{I}$. Let

$$\mathbb{I}' := \mathbb{I} \setminus \{S(x)\tau_1, \ldots, S(x)\tau_m\} \cup \{S(s)\rho_1, \ldots, S(s)\rho_n\},$$

i.e., change the truth value for $S$-atoms such that the clauses unsatisfied under $\mathbb{I}$ are satisfied under $\mathbb{I}'$. Because $\mathbb{I}$ and $\mathbb{I}'$ only differ on $S$-atoms, there exists a clause $D \in N^\perp$ that is false under $\mathbb{I}'$ and contains an $S$-atom. Let $D = C_l\sigma$. Since $\mathbb{I} \models D$, $S(x)\sigma$ was added to $\mathbb{I}'$ by some clause $C_r\rho_j$, where $S(s)\rho_j = S(x)\sigma$. Let $R$ be the resolvent of $C_r\rho_j$ and $C_1\sigma$ on $S(s)\rho_j$ and $S(x)\sigma$. Then, $\mathbb{I} \not\models R$ because $\mathbb{I} \not\models C_r\rho_j$ and $\mathbb{I} \cup \{S(s)\rho_j\} \not\models C_l'$. Thus, $\mathbb{I} \not\models N'^\perp$. For $D = C_r\sigma$, the proof is analogous. Therefore, $N'^\perp$ is a conflicting core of $N \cup \{(C;\pi)\}$. □

In the case of a refinement $N \cup \{(C;\pi)\} \Rightarrow_{\text{Ref}} N \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\}$, the clauses $(C;\pi \wedge x \neq t)$ and $(C;\pi)\{x \mapsto t\}$ are both instances of $(C;\pi)$. Therefore, any conflict-instance of either clause in a conflicting core of $N \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\}$ is also an instance of $(C;\pi)$. Hence, $N^\perp$ is also a conflicting core of $N \cup \{(C;\pi)\}$.

**Lemma 4.9** (Refinement lifting) *Let $N \Rightarrow_{\text{Ref}} N'$. If $(N^\perp;\pi)$ is a conflicting core of $N'$, then $(N^\perp;\pi)$ is a conflicting core of $N$.*

*Approximation-refinement* Thanks to the addition of straight dismatching constraints, the refinement is now quite different from the one in [20]. As a result of the changed refinement, we can adopt a new approach to the approximation-refinement. Whereas previously the refinement was defined separately for the linear and shallow cases, now both are generalized and treated as the same type of lift-conflict. Intuitively, a lift-conflict occurs whenever non-linear variable occurrences in the original clause set are approximated with linear variable occurrences that are then instantiated by non-unifiable terms.

Consider the two cases where a lift-conflict can occur. In the linear case, there exists a clause in the conflicting core that is not an instance of the original clauses. In the shallow case, there exists a pair of clauses whose resolvent is not an instance of the original clauses. Since lifting monadic and refinement transformations always succeeds, there are no lift-conflicts in those cases.

Whenever lifting fails, there is a clause of some form, the so-called lift-conflict, that is not an instance of an approximated clause. In the following this clause is defined as a general lift-conflict which is then used to determine the refinement independently of the rule that caused it.

**Definition 4.10** (*The lift-conflict*) Let $N \cup \{(C,\pi)\} \Rightarrow_{\mathrm{LI}} N \cup \{(C_a,\pi_a)\}$ and $N^{\perp}$ be a minimal conflicting core of $N \cup \{(C_a,\pi_a)\}$. A conflict-instance $C_c \in N^{\perp}$ of $(C_a,\pi_a)$ is called a lift-conflict if $C_c$ is not an instance of $(C,\pi)$ modulo duplicate literal elimination.

Let $N \cup \{(C,\pi)\} \Rightarrow_{\mathrm{SH}} N \cup \{(C_l,\pi_l),(C_r,\pi_r)\}$, $(C_a;\pi_a)$ be the shallow resolvent and $N^{\perp}$ be a minimal conflicting core of $N \cup \{(C_l,\pi_l),(C_r,\pi_r)\}$. The resolvent $C_c$ of $C_l\delta_l \in N^{\perp}$ and $C_r\delta_r \in N^{\perp}$ is called a lift-conflict if $C_c$ is not an instance of $(C,\pi)$ modulo duplicate literal elimination. We also consider $C_c$ as a conflict-instance of $(C_a;\pi_a)$.

The goal of refinement is then to change the parent clause in such a way that is both satisfiability equivalent and prevents the lift-conflict from again appearing during the lifting of the refined approximations. Solving the refined approximation will then necessarily produce a new proof because its conflicting core has to be different. For this purpose, the refinement transformation segments the original clause $(C;\pi)$ into two parts $(C;\pi \wedge x \neq t)$ and $(C;\pi)\{x \mapsto t\}$.

For example, consider $N$ and its Linear transformation $N'$.

$$\begin{array}{rcl} & \rightarrow & P(x,x) \\ P(a,b) & \rightarrow & \end{array} \quad \Rightarrow_{\mathrm{LI}} \quad \begin{array}{rcl} & \rightarrow & P(x,x') \\ P(a,b) & \rightarrow & \end{array}$$

The ground conflicting core of $N'$ is

$$\begin{array}{rcl} & \rightarrow & P(a,b) \\ P(a,b) & \rightarrow & \end{array}$$

Because $P(a,b)$ is not an instance of $P(x,x)$, lifting fails. $P(a,b)$ is the lift-conflict. Specifically $\{x \mapsto a\}$ and $\{x \mapsto b\}$ are conflicting substitutions for the parent variable $x$. Pick $\{x \mapsto a\}$ to segment $P(x,x)$ into $(P(x,x); x \neq a)$ and $P(x,x)\{x \mapsto a\}$. Now, any descendant of $(P(x,x); x \neq a)$ cannot have $a$ at the position of the first $x$, and any descendant of $P(x,x)\{x \mapsto a\}$ must have an $a$ at the position of the second $x$. Thus, $P(a,b)$ is excluded in both cases and no longer appears as a lift-conflict.

To show that the lift-conflict will not reappear in the general case, I use that the conflict clause and its ancestors have strong ties between their term structures and constraints.

**Definition 4.11** (*Constrained term skeleton*) The *constrained term skeleton* of a term $t$ under constraint $\pi$, $\mathrm{skt}(t,\pi)$, is defined as the normal form of the following transformation:

$$(t[x]_{p,q};\pi) \Rightarrow_{\mathrm{skt}} (t[q/x'];\pi \wedge \pi\{x \mapsto x'\}), \text{ where } p \neq q \text{ and } x' \text{ is fresh.}$$

The constrained term skeleton of a term $t$ is essentially a linear version of $t$ where the restrictions on each variable position imposed by $\pi$ are preserved. For $(t,\pi)$, a solution $\delta$ of $\pi$ is over $\mathrm{lvar}(\pi) \cup \mathrm{vars}(t)$ and $t\delta$ is called a ground instance of $(t,\pi)$.

**Lemma 4.12** *Let $N_0 \Rightarrow^{*}_{\mathrm{AP}} N_k$, $(C_k;\pi_k)$ in $N$ with the ancestor clause $(C_0;\pi_0) \in N_0$ and $N_k^{\perp}$ be a minimal conflicting core of $N_k$. Let $\delta$ be a solution of $\pi_k$ such that $C_k\delta$ is in $N_k^{\perp}$. If $(L',q')$ is a literal position in $(C_k;\pi_k)$ with the ancestor $(L,q)$ in $(C_0,\pi_0)$, then (i) $L'\delta|_{q'}$ is an instance of $\mathrm{skt}(L|_q,\pi_0)$, (ii) $q = q'$ if $L$ and $L'$ have the same predicate, and (iii) if $L'|_{q'} = x$ and there exists an ancestor variable $y$ of $x$ in $(C_0,\pi_0)$, then $L|_q = y$.*

***Proof*** By induction on the length of the approximation $N_0 \Rightarrow^*_{\mathrm{AP}} N_k$.

The base case $N_k = N_0$, is trivial.

Let $N_0 = N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{SH}} N \cup \{(C_l;\pi_l),(C_r;\pi_r)\} = N_k$, $(C_k;\pi_k)$ be the shallow $\rho$-resolvent and $C_k\delta$ be the resolvent of two instances of $(C_l;\pi_l)$ and $(C_r;\pi_r)$ in $N_k^\perp$. Then, $(C_k;\pi_k)\rho^{-1}$ is equal to $(C;\pi)$ modulo duplicate literal elimination. Thus, by definition $(L,q) = (L',q')\rho^{-1}$. Therefore, (i) $L'\delta|_{q'}$ is an instance of $\mathrm{skt}(L|_q,\pi_0)$, (ii) $q = q'$ if $L$ and $L'$ have the same predicate, and (iii) if $L'|_{q'} = x$ and there exists an ancestor variable $y$ of $x$ in $(C_0,\pi_0)$, then $L|_q = y$.

Now, let $N_0 \Rightarrow_{\mathrm{AP}} N_1 \Rightarrow^*_{\mathrm{AP}} N_k$. Since $(L',p)$ has an ancestor literal position in $(C_0,\pi_0)$, the ancestor clause of $(C_k;\pi_k)$ in $N_1$, $(C_1,\pi_1)$, contains the the ancestor literal position $(L_1,q_1)$, which has $(L,q)$ as its parent literal position. By the induction hypothesis on $N_1 \Rightarrow^*_{\mathrm{AP}} N_k$, (i) $L'\delta|_{q'}$ is an instance of $\mathrm{skt}(L_1|_{q_1},\pi_1)$, (ii) $q_1 = q'$ if $L_1$ and $L'$ have the same predicate, and (iii) if $L'|_{q'} = x$ and there is an ancestor variable $y_1$ of $x$ in $(C_1,\pi_1)$, then $L_1|_{q_1} = y_1$.

The proof follows by a case distinction on $N_0 \Rightarrow_{\mathrm{AP}} N_1$, however, it is trivial except for the $\Rightarrow_{\mathrm{SH}}$ case. Let $N_0 = N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{SH}} N \cup \{(C_l;\pi_l),(C_r;\pi_r)\} = N_1$ where a term $s$ is extracted from a positive literal $Q(s'[s]_p)$ via introduction of the fresh predicate $S$ and variable $x$. The core insight in this case is that since $N_k^\perp$ is minimal and the predicate $S$ only occurs in $(C_l;\pi_l)$ and $(C_r;\pi_r)$, there exists for every clause $(C_l;\pi_l)\delta$ in $N_k^\perp$ with the literals $S(x)\delta$ and $Q(s'[p/x])\delta$ a clause $(C_r;\pi_r)\sigma$ in $N_k^\perp$ with the literal $S(s)\sigma$ such that $S(s)\sigma = S(x)\delta$. Otherwise, $(C_l;\pi_l)\delta$ could never be fully resolved in a resolution proof and would therefore be superfluous. As a consequence, $x\delta$ is always an instance of $s$. □

Next, we define the notion of descendants and descendant relations to connect lift-conflicts in ground conflicting cores with their corresponding ancestor clauses. For an approximation $N \Rightarrow^*_{\mathrm{AP}} N'$ and a conflicting core $N'^\perp$ of $N'$, any conflict clause in $N'^\perp$ is a descendant of a clause in $N$. Inversely, if a clause $C$ is not a descendant of a clause in $N$, then $C$ is not a conflict clause for any approximation of $N$.

**Definition 4.13** (*Descendants*) Let $N \Rightarrow^*_{\mathrm{AP}} N'$, $[(C;\pi),N] \Rightarrow^*_{\mathrm{A}} [(C';\pi'),N']$ and $D$ be a ground instance of $(C';\pi')$. Then, $D$ is called a *descendant* of $(C;\pi)$. Define the $[(C;\pi),N] \Rightarrow^*_{\mathrm{A}} [(C';\pi'),N']$-descendant relation $\Rightarrow_D$ that maps literals in $D$ to literal positions in $(C;\pi)$ using the following rule:

$$L'\delta \Rightarrow_D (L,r) \text{ if } L'\delta \in D \text{ and } [r,L,(C;\pi),N] \Rightarrow^*_{\mathrm{A}} [\varepsilon,L',(C';\pi'),N']$$

For the descendant relations it is of importance to note that while there are potentially infinite ways that a lift-conflict $C_c$ can be a descendant of an original clause $(C;\pi)$, there are only finitely many distinct descendant relations over $C_c$ and $(C;\pi)$. This means, if a refinement transformation can prevent one distinct descendant relation without allowing new distinct descendant relations (Lemma 4.14), a finite number of refinement steps can remove the lift-conflict $C_c$ from the descendants of $(C;\pi)$ (Lemma 4.15). Thereby, preventing any conflicting cores containing $C_c$ from being found again.

A clause $(C;\pi)$ can have two descendants that are the same except for the names of the $S$-predicates introduced by Shallow transformations. Because the used approximation $N \Rightarrow^*_{\mathrm{AP}} N'$ is arbitrary and therefore also the choice of fresh $S$-predicates, if $D$ is a descendant of $(C;\pi)$, then any clause $D'$ equal to $D$ up to a renaming of $S$-predicates is also a descendant of $(C;\pi)$. On the other hand, the actual important information about an $S$-predicate is which term it extracts. Two descendants of $(C;\pi)$ might be the exactly the same but their $S$-predicate extract different terms in $(C;\pi)$. For example, $P(a) \rightarrow S(f(a))$ is a descendant of $P(x), P(y) \to Q(f(x), g(f(x)))$ but might extract either occurrence of $f(x)$. These cases are distinguished by their respective descendant relations. In the example, they are either $S(f(a)) \Rightarrow_D (Q(f(x), g(f(x))), 1)$ or $S(f(a)) \Rightarrow_D (Q(f(x), g(f(x))), 2.1)$.

**Lemma 4.14** *Let* $N_0 = N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{Ref}} N \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\} = N_1$ *be a refinement and* $D$ *be a ground clause. If there exists a* $[(C;\pi \wedge x \neq t), N_1] \Rightarrow^*_{\mathrm{A}} [(C';\pi'), N_2]$*- or* $[(C;\pi)\{x \mapsto t\}, N_1] \Rightarrow^*_{\mathrm{A}} [(C';\pi'), N_2]$*-descendant relation* $\Rightarrow^1_D$*, then there is an equal* $[(C;\pi), N_0] \Rightarrow^*_{\mathrm{A}} [(C';\pi'), N_2]$*-descendant relation* $\Rightarrow^0_D$.

***Proof*** Let $L_D$ be a literal of $D$ and $L' \Rightarrow^1_D (L, r)$. If $D$ is a descendant of $(C;\pi \wedge x \neq t)$, then $[r, L, (C;\pi \wedge x \neq t), N_1] \Rightarrow^*_{\mathrm{A}} [\varepsilon, L', (C';\pi'), N_2]$. Because $[r, L, (C;\pi), N_0] \Rightarrow_{\mathrm{A}} [r, L, (C;\pi \wedge x \neq t), N_1]$, $L' \Rightarrow^0_D (L, r)$. If $D$ is a descendant of $(C;\pi)\{x \mapsto t\}$, the proof is analogous. □

**Lemma 4.15** (Refinement) *Let* $N \Rightarrow_{\mathrm{AP}} N'$ *and* $N^\perp$ *be a minimal conflicting core of* $N'$*. If* $C_c \in N^\perp$ *is a lift-conflict, then there exists a finite refinement* $N \Rightarrow^*_{\mathrm{Ref}} N_R$ *such that for any approximation* $N_R \Rightarrow^*_{\mathrm{AP}} N'_R$ *and ground conflicting core* $N^\perp_R$ *of* $N'_R$*,* $C_c$ *is not a lift-conflict in* $N^\perp_R$ *modulo duplicate literal elimination.*

***Proof*** Let $(C_a, \pi_a)$ be the conflict clause of $C_c$ and $(C;\pi) \in N$ be the parent clause of $(C_a, \pi_a)$. $C_c$ is a descendant of $(C;\pi)$ with the corresponding $[(C;\pi), N] \Rightarrow_{\mathrm{A}} [(C_a;\pi_a), N']$-descendant relation $\Rightarrow^0_{C_c}$. Apply induction on the set of distinct $[(C;\pi), N] \Rightarrow^*_{\mathrm{A}} [(C';\pi'), N'']$-descendant relations $\Rightarrow_{C_c}$ for arbitrary approximations $N \Rightarrow^*_{\mathrm{AP}} N''$.

Since only the Shallow and Linear transformations can produce lift-conflicts, the clause $(C;\pi)$ is replaced by either a linearized clause $(C';\pi')$ or two shallow clauses $(C_l;\pi)$ and $(C_r;\pi)$. Then, the conflict clause $(C_a;\pi_a)$ of $C_c$ is either the linearized $(C';\pi')$ or the resolvent of $(C_l;\pi)$ and $(C_r;\pi)$. In either case, $C_c = C_a\delta$ for some solution $\delta$ of $\pi_a$. Furthermore, there exists a substitution $\tau = \{x'_1 \mapsto x_1, \ldots, x'_n \mapsto x_n\}$ such that $(C;\pi)$ and $(C_a;\pi_a)\tau$ are equal modulo duplicate literal elimination. That is, $\tau = \{x' \mapsto x\}$ for a Linear transformation and $\tau = \rho^{-1}$ for Shallow transformation.

Assume $C_c = C_a\tau\sigma$ for some grounding substitution $\sigma$, where $\tau\sigma$ is a solution of $\pi_a$. Thus, $\sigma$ is a solution of $\pi_a\tau$, which is equivalent to $\pi$. Then, $C_c$ is equal to $C\sigma$ modulo duplicate literal elimination an instance of $(C;\pi)$, which contradicts with $C_c$ being a lift-conflict. Hence, $C_c = C_a\delta$ is not an instance of $C_a\tau$ and thus, $x_i\delta \neq x'_i\delta$ for some $x_i$ in the domain of $\tau$.

Because $x_i\delta$ and $x'_i\delta$ are ground, there is a position $p$ where $x_i\delta|_p$ and $x'_i\delta|_p$ have different function symbols. Construct the straight term $t$ using the path from the root to $p$ on $x_i\delta$ with variables that are fresh in $(C, \pi)$. Then, use $x_i$ and $t$ to segment $(C;\pi)$ into $(C;\pi \wedge x_i \neq t)$ and $(C;\pi)\{x_i \mapsto t\}$ for the refinement $N \Rightarrow_{\mathrm{Ref}} N_R$. Note, that $x_i\delta$ is a ground instance of $t$, while $x'_i\delta$ is not.

Let $(L'_1, r'_1)$ and $(L'_2, r'_2)$ in $(C_a, \pi_a)$ be literal positions of the variables $x_i$ and $x'_i$ in $C_a$, and $(L_1, r_1)$ and $(L_2, r_2)$ in $(C, \pi)$ be the parent literal positions of $(L'_1, r'_1)$ and $(L'_2, r'_2)$, respectively. Because $(C_a, \pi_a)\tau$ is equal to $(C;\pi)$ modulo duplicate literal elimination, $L_1|_{r_1} = L_2|_{r_2} = x_i$. Let $N \Rightarrow_{\mathrm{Ref}} N_1$ be the refinement where $(C;\pi)$ is segmented into $(C;\pi \wedge x_i \neq t)$ and $(C;\pi)\{x_i \mapsto t\}$.

By Lemma 4.14, for every $[(C;\pi \wedge x_i \neq t), N_1] \Rightarrow^*_{\mathrm{A}} [(C'_a;\pi'_a), N_2]$- or $[(C;\pi)\{x_i \mapsto t\}, N_1] \Rightarrow^*_{\mathrm{A}} [(C'_a;\pi'_a), N_2]$-descendant relation there exists a corresponding equal $[(C;\pi), N] \Rightarrow_{\mathrm{A}} [(C'_a;\pi'_a), N_2]$-descendant relation. Assume there is a

$[(C;\pi \wedge x_i \neq t), N_1] \Rightarrow^*_{\text{A}} [(C'_a;\pi'_a), N_2]$-descendant relation $\Rightarrow^1_{C_c}$ that is not distinct from $\Rightarrow^0_{C_c}$. Because $L'_1\delta \Rightarrow^0_{C_c} (L_1, r)$ for some literal position $(L_1, r)$ in $(C;\pi)$, which is the parent literal position of $(L_1, r)$ in $(C;\pi \wedge x_i \neq t)$, $L'_1\delta \Rightarrow^1_{C_c} (L_1, r)$. However, this contradicts Lemma 4.12 because $x_i\delta$ is not an instance of $\text{skt}(L_1|_{r_1}, \pi \wedge x_i \neq t) = \text{skt}(x_i, \pi \wedge x_i \neq t)$. The case that there is a $[(C;\pi)\{x_i \mapsto t\}, N_1] \Rightarrow^*_{\text{A}} [(C'_a;\pi'_a), N_2]$-descendant relation that is not distinct from $\Rightarrow^0_{C_c}$ is analogous using the argument that $x'_i\delta$ is not an instance of $\text{skt}(L_2\{x_i \mapsto t\}|_{r_2}, \pi) = \text{skt}(t, \pi)$. Hence, there are strictly less distinct descendant relations over $C_c$ and $(C;\pi \wedge x \neq t)$ or $(C;\pi)\{x \mapsto t\}$ than there are distinct descendant relations over $C_c$ and $(C, \pi)$.

If there are no descendant relations, then $C_c$ can no longer appear as a lift conflict. Otherwise, by the inductive hypothesis, there exists a finite refinement $N \Rightarrow_{\text{Ref}} N_1 \Rightarrow^*_{\text{Ref}} N_R$ such that for any approximation $N_R \Rightarrow_{\text{AP}} N'_R$ and ground conflicting core $N^{\perp}_R$ of $N'_R$, $C_c$ is not a lift-conflict in $N^{\perp}_R$ modulo duplicate literal elimination. □

**Theorem 4.16** (Soundness and completeness of FO-AR) *Let $N$ be a clause set and $N'$ its MSL(SDC) approximation: (i) if $N$ is unsatisfiable then there exists a conflicting core of $N'$ that can be lifted to a refutation in $N$, (ii) if $N'$ is satisfiable, then $N$ is satisfiable too.*

***Proof*** (Idea) By Lemmas 4.3 and 4.15, where the latter can be used to show that a core of $N'$ that cannot be lifted also excludes the respective instance for unsatisfiability of $N$. □

Actually, Lemma 4.15 can be used to define a fair strategy on refutations in $N'$ in order to receive also a dynamically complete FO-AR calculus, following the ideas presented in [20].

## 5 Implementation

We have implemented the first-order approximation-refinement calculus (FO-AR) as an extension of SPASS v.3.8 called SPASS-AR. As shown, the MSL(SDC) fragment can be decided using ordered resolution with the specific selection function sel. Therefore, by just using sel as the default selection function, SPASS already provides the decision procedure for MSL.

The addition of the straight dismatching constraints to the signature of clauses follows in a straight-forward way from their description in [21]. In relation to the main saturation loop of SPASS, the constraints are treated as a black-box that given constraints checks solvability or subsumption. Internally, the constraint of a new derived clause is constructed by concatenating the constraints of the parent clauses and applying the mgu then normalizing and checking for solvability. If the constraint is unsolvable, it is deleted as the clause is redundant. The solvability check also sorts and shortens the constraint if possible, e.g., with signature $\{a, f\}$, the constraints $x \neq f(a)$ and $x \neq f(f(v))$ are combined into $x \neq f(v)$.

The remaining steps of FO-AR, approximation, lifting, and refinement, are build around the core routine of SPASS. A given clause set is first approximated. To allow extracting the ancestor relation later, each transformation step is documented in the same way as inferences by adding the information of rule, parent clauses and affected literals to the approximation clause. The resulting approximation is given as input to the modified solver.

Once finished, the result is analyzed. If satisfiable, the result is reported and the routine finished. Otherwise, the empty clause returned by SPASS is given as input to the lifting. For the sake of the lifting, each inferred clause had its parents documented and even if it was redundant was not deleted. This information is then used to extract a DAG of the refutation.

In a first version of SPASS-AR, the conflicting core was then recursively constructed and step-wise lifted. However, the exponential size of the conflicting core compared to the refutation DAG made this approach impractical in many examples. We, therefore, replaced it by a lifting that, while more complex, works directly on the DAG (see Sect. 5.1).

If the lifting succeeds, the conflicting core or the refutation DAG lifted to the original clause set can be reported as a proof of unsatisfiability. Otherwise, all relevant information of the failed lifting step is returned as the lift-conflict and given to the refinement function.

First, the refinement uses a modified unification algorithm on the conflict clause and its parent to identify the positions of the two subterms in the conflict clause that caused the unification to fail (see Sect. 5.2). One of those positions is then traced upwards to the point were in the parent clause a variable is found. As the conflict clause is by construction an instance of the parent clause's term skeleton, this tracing necessarily leads to the same variable for both positions. Next, we climb the ancestry tree of the conflict clause to its corresponding original clause while keeping track of the incorrectly instantiated variable. The original clause $(C;\pi)$, its variable $x$ and one of the instantiations $t$ are used to replace $(C;\pi)$ with the refinements $(C;\pi)\{x \mapsto t\}$ and $(C;\pi \wedge x \neq t)$. Lastly after resetting the solvers data-structures, the refined clause set is again fed to the approximation.

## 5.1 DAG Lifting

As mentioned before, theory and implementation differ in that we actually lift the resolution proof directly instead of creating and lifting a conflicting core. The simple reason is that a conflicting core can be exponentially larger than its corresponding resolution proof which is stored compactly as a directed acyclic graph (DAG). Note for the following that in a resolution refutation tree, the leaves are at the top while the root is at the bottom.

The general idea is to recursively traverse the proof DAG down from the approximation clauses in the leaves to the root and replace each node with a corresponding clause derived from the original problem. If this fails at some node, we can either immediately identify the conflict or check which parent node contains the conflict. In the latter case, we move back up the DAG towards the leaves looking for the conflict.

The lifting is done at each node in two steps: first the linear lifting, then the shallow lifting on top. For clarity however, we will explain each lifting separately.

*Linear DAG lifting* Assume the original clause set $N$ is already monadic and shallow such that its approximation $N'$ is created using only Linear transformations. Furthermore, for simplicity, we will ignore constraints.

Each leaf in the resolution proof contains an approximation clause $C$ in $N'$ which has an ancestor clause $C^A$ in $N$. We revert each Linear transformation that was applied to $C^A$ to create $C$ by substituting any fresh variables with their original variable. Specifically, if $x$ is replaced with $x'$ at some position, then one lifting step of $C$ is $C\{x' \mapsto x\}$. We repeat this for each Linear transformation step. The end result $C\{x'_1 \mapsto x_1; \ldots; x'_n \mapsto x_n\}$ is called the *linear lifting clause* $C^L$ of $C$. $C^L$ is then both an instance of $C$ and can be derived from clauses in $N$, specifically, $C^L$ is satisfiability equivalent to $C^A$.

For the internal nodes of the proof DAG, we will here assume that there are only resolution inferences in the proof. The method is analogous for factorization inferences and clause reductions, e.g. condensation.

Let $R$ be the clause at the current node with $C$ and $D$ as the clauses at its parent nodes. First, we recursively compute their respective linear lifting clauses $C^L$ and $D^L$. By construction, $R$ is the resolvent of $C$ and $D$. Next, we try to recreate this exact resolution inference using

$C^L$ and $D^L$ instead. If this succeeds, their resolvent $R^L$ is the linear lifting clause of $R$. As before, $R^L$ is an instance of $R$ and can be derived from $N$. If $R$ is the empty clause, then $R^L$ is also empty and we have therefore shown that $N$ is unsatisfiable.

Otherwise, if $C^L$ and $D^L$ have no resolvent, lifting failed and we climb back up the DAG to find the leaf containing the linear conflict. The recursive climbing algorithm receives as input a refutation node with clause $R$ and the so-called *conflicting substitution* $\rho$. In particular, $R\rho$ is called the *conflicting clause* and has the property that it can be derived from $N'$ but, using the given refutation, none of its ground instances can be derived from $N$. We begin climbing at the node where lifting failed with the identity as the conflicting substitution.

Let $R$ be the clause at the current node with left and right parent clauses $C$ and $D$ and their respective linear lifting clauses $C^L$ and $D^L$. Again, $R$ is the result of a resolution inference between $C$ and $D$ using their mgu $\sigma$. We check whether $C^L$ has common ground instances with $C\sigma\rho$. If this is not the case, then $C\sigma\rho$ is a conflicting clause and the lift-conflict is on the left parent branch. The algorithm continues on the left parent node with $\sigma\rho$ as the conflicting substitution. Otherwise, do the same with $D^L$ and $D\sigma\rho$.

If both have common ground instances, let $\delta$ be the mgu of $C^L$ and $C\sigma\rho$. The algorithm then continues with the right parent node and the conflicting substitution $\sigma\rho\delta$. Note that $D\sigma\rho\delta$ is a conflicting clause because otherwise $R\rho\delta$ contains ground instances that are derivable from $N$ which contradicts $R\rho$ being a conflicting clause itself.

Once a leaf with clause $C$ is reached, we have a conflicting clause $C\rho$ which has no common instances with the linear lifting clause $C^L$. We again revert the Linear transformations on $C$ step by step while unifying the instantiations $x\rho$ and $x'\rho$ until we reach the transformation step where this fails. The variable $x$ and the two instantiations $x\rho$ and $x'\rho$ constitute the lift-conflict and are used to refine the original ancestor clause $C^A$.

To extend this method to constraints, unification-checks are also failed if the mgu $\sigma$ exists but $\pi\sigma$ is unsolvable in the given context.

Consider as an example the clause set consisting of $C_1 = \to P(x,x)$; $C_2 = P(a,y) \to Q(y,b)$; and $C_3 = Q(b,x) \to$ where the first clause has the linear approximation $C_1' = \to P(x,x')$. A possible resolution refutation tree of the approximation is

$$\cfrac{\cfrac{C_1' :\to P(x,x') \quad C_2 : P(a,y) \to Q(y,b)}{C_4 :\to Q(x',b)} \qquad C_3 : Q(b,x) \to}{C_5 : \Box}$$

The linear lifting clauses $C'^{L_1}$, $C_2^L$ and $C_3^L$ of $C_1'$, $C_2$ and $C_3$ are $C_1$, $C_2$ and $C_3$ respectively. The resolvent $C_4 = \to Q(x',b)$ has the linear lifting clause $C_4^L = \to Q(a,b)$. However, there is no linear lifting clause of the empty clause $C_5$ because $C_4^L$ and $C_3^L$ have no resolvent.

$$\cfrac{\cfrac{C'^{L_1} :\to P(x,x) \quad C_2^L : P(a,y) \to Q(y,b)}{C_4^L :\to Q(a,b)} \qquad C_3^L : Q(b,x) \to}{C_5^L : -}$$

Thus, $\Box$ is the initial conflicting clause. Since the mgu of $C_4$ and $C_3$ is $\rho = \{x \mapsto b, x' \mapsto b\}$, $C_4\rho = \to Q(b,b)$ and $C_4^L = \to Q(a,b)$ have no common instances and thus $C_4\rho$ is the next conflicting clause. After applying the mgu $\sigma = \{x \mapsto a, y \mapsto x'\}$ of $C_1'$ and $C_2$, $C_1'\sigma\rho = \to P(a,b)$ has no common instances with $C'^{L_1} = \to P(x,x)$ and is therefore the last conflicting clause. Arriving at a leaf, we see that $x\sigma\rho = a$ and $x'\sigma\rho = b$ are the conflicting instantiations.

$$\dfrac{\dfrac{C_1'\sigma\rho :\rightarrow P(a,b) \quad C_2\sigma\rho : P(a,b) \rightarrow Q(b,b)}{C_4\rho :\rightarrow Q(b,b)} \qquad C_3\rho : Q(b,b) \rightarrow}{C_5\,\mathrm{id} : \square}$$

*Shallow DAG lifting* The shallow lifting follows the same idea as the linear lifting. This time, assume the original clause set $N$ is already monadic and linear such that its approximation $N'$ is created using only Shallow transformations.

Unlike the Linear transformation, lifting a Shallow transformation cannot be done locally on each leaf. Reverting a Shallow transformation would require combining an approximation clause $(C; \pi)$ with its left or right counterpart and thereby drastically changing the shape of the resolution proof DAG. Instead, we replace the Shallow transformation steps with so-called generalized Shallow transformation by adding the shared variables as new arguments to the shallow predicate $S$. Specifically, when $\Gamma \rightarrow E[s]_p, \Delta$ is approximated by $S(x), \Gamma_l \rightarrow E[p/x], \Delta_l$ and $\Gamma_r \rightarrow S(s), \Delta_r$, we store the shared variables $\{y_1, \ldots, y_n\}$ defined by the intersection $\mathrm{vars}(\Gamma_l, \Delta_l, E[p/x]) \cap \mathrm{vars}(\Gamma_r, \Delta_r, S[s])$ such that they can be retrieved using the fresh predicate $S$ as a key. During shallow lifting, we replace any literal $S(t)$ in a leaf node's clause $C$ with $S(t, y_1, \ldots, y_n)$ to create the shallow lifting clause $C^L$. Because this general Shallow transformation is actually satisfiability equivalent, the original clause set $N$ is proven unsatisfiable if the lifted clauses create a valid resolution refutation, i.e., whenever two $S$-literals $S(t)$ and $S(s)$ are unified in the refutation, the lifted $S(t, t_1, \ldots, t_n)$ and $S(s, s_1, \ldots, s_n)$ are unifiable as well.

For the internal nodes of the proof DAG, we again only describe the case for resolution inferences where the lifting is analogous to the linear case. The difference lies in how the climbing algorithm finds the lift-conflict.

In most cases, when shallow lifting fails at an internal node with clause $R$, two shallow literals $\neg S(t, t_1, \ldots, t_n)$ and $S(s, s_1, \ldots, s_n)$ in $C^L$ and $D^L$ are not unifiable. Since each shallow predicate $S$ is unique to its respective Shallow transformation, this reveals the responsible ancestor clause that needs to be refined. Then, we sequentially unify each $t_i$ and $s_i$ until we find the first pair that does not allow unification. Those $t_j$ and $s_j$ and their shared variable constitute the lift-conflict.

However, in the general case, this may not be the first node at which shallow lifting fails. The correct conflicting node can be found by climbing up the refutation tree until there is no conflict in either parent node. Checking which parent node contains a conflict works with the same idea as in the linear case with the exception that when comparing instances of an atom $S(t)$ with those of a lifted atom $S(s, s_1, \ldots, s_n)$ we ignore the added terms.

Consider as an example the clause set consisting of $C_1 = \rightarrow P(x, f(x))$ and $C_2 = P(x, x) \rightarrow$ where the fist clause has the shallow approximations $C_l = S(x') \rightarrow P(x, x')$ and $C_r = \rightarrow S(f(x))$. A possible resolution refutation tree is

$$\dfrac{C_r :\rightarrow S(f(x)) \qquad \dfrac{C_l : S(x') \rightarrow P(x,x') \quad C_2 : P(x,x) \rightarrow}{C_3 : S(x') \rightarrow}}{C_4 : \square}$$

For the shallow lifting, we add the shared variable $x$ to each $S$-atom in the leaves and then repeat the refutation steps.

$$\dfrac{C_r^L :\rightarrow S(f(x), x) \qquad \dfrac{C_l^L : S(x', x) \rightarrow P(x,x') \quad C_2^L : P(x,x) \rightarrow}{C_3^L : S(x', x') \rightarrow}}{C_4^L : -}$$

Again, the empty clause $C_4$ has no shallow lifting clause because $C_3^L = S(x', x') \rightarrow$ and $C_r^L = \rightarrow S(f(x), x)$ can not be resolved. Next, we check whether a lift-conflict also exists at one of the parents nodes. However this is not the case because for the mgu $\rho = \{x' \mapsto f(x)\}$ in the resolution of $C_r$ and $C_3$, $C_r\rho = \rightarrow S(f(x))$ and $C_3\rho = S(f(x)) \rightarrow$ each have instances in common with $C_r^L = \rightarrow S(f(x), x)$ and $C_3^L = S(x', x') \rightarrow$ when ignoring the added arguments.

Then, after unifying the first arguments of the shallow atoms $S(f(x), x)$ and $S(x', x')$, we get $S(f(x), x)$ and $S(f(x), f(x)) \rightarrow$ which reveals that the shared variable $x$ has the conflicting instantiation terms $x$ and $f(x)$.

## 5.2 The Lift-Conflict Selection

The main work in the refinement is the extraction of the straight term used to generate the refined clauses. A lift-conflict, when found during lifting, entails a corresponding original clause $(C; \pi)$, a variable $x$ in $C$ and two terms $t$ and $t'$ that are not unifiable under a constraint $\pi'$. However, the refinement requires an appropriate straight term $s$ to create the refinements $(C; \pi)\{x \mapsto s\}$ and $(C; \pi \wedge x \neq s)$.

The most reliable method is to first find the minimal solution $\delta$ of $\pi'$ under the term ordering $\prec$ also used by the MSL(SDC) solver, i.e. where $x\delta \preceq x\delta'$ for every variable $x$ and solution $\delta'$ of $\pi'$. The resulting ground terms $t\delta$ and $t'\delta$ are by construction not equal and therefore, there is a position $p$ where the top-symbol of $t|_p$ and $t'|_p$ differ while they are the same for every position above $p$. The function symbols on the path from the root of $t\delta$ up to and including $p$ describe the required straight term $s$ that *differentiates* $t\delta$ and $t'\delta$, i.e., $t\delta$ is an instance of $s$ while $s$ and $t'\delta$ have no common ground instances. In the worst case, however, this refinement only prevents one unliftable ground resolution proof from reoccurring in the following loops.

However, there are cases where a better straight term can be chosen. The simplest case is when unifying the conflict clause with the original clause fails because their instantiations differ at some equivalent positions. For example, consider $N = \{P(x, x);\ P(f(x, a), f(y, b)) \rightarrow\}$. $N$ is satisfiable but the linear transformation is unsatisfiable with conflict clause $P(f(x, a), f(y, b))$ which is not unifiable with $P(x, x)$, because the two terms $f(x, a)$ and $f(y, b)$ have different constants at the second argument. A refinement of $P(x, x)$ is

$$\begin{array}{c}(P(x, x)\,;\, x \neq f(v, a))\\ (P(f(x, a), f(x, a))\,;\, \top)\end{array}$$

$P(f(x, a), f(y, b))$ shares no ground instances with the approximations of the refined clauses.

Next, assume that again unification fails due to structural difference, but this time the differences lie at different positions. For example, consider $N = \{P(x, x);\ P(f(a, b), f(x, x)) \rightarrow\}$. $N$ is satisfiable but the linear transformation of $N$ is unsatisfiable with conflict clause $P(f(a, b), f(x, x))$ which is not unifiable with $P(x, x)$ because in $f(a, b)$ the first and second argument are different but the same in $f(x, x)$. A refinement of $P(x, x)$ is

$$\begin{array}{c}(P(x, x)\,;\, x \neq f(a, v))\\ (P(f(a, x), f(a, x)))\,;\, x \neq a)\\ (P(f(a, a), f(a, a)))\,;\, \top)\end{array}$$

$P(f(a,b), f(x,x))$ shares no ground instances with the approximations of the refined clauses. It is also possible that the conflict clause and original clause are unifiable by themselves, but the resulting constraint has no solutions.

For example, consider $N = \{P(x,x); (P(x,y) \rightarrow; x \neq a \wedge x \neq b \wedge y \neq c \wedge y \neq d)\}$ with signature $\Sigma = \{a,b,c,d\}$. $N$ is satisfiable but the linear transformation of $N$ is unsatisfiable with conflict clause $(\rightarrow P(x,y); x \neq a \wedge x \neq b \wedge y \neq c \wedge y \neq d)$. While $P(x,x)$ and $P(x,y)$ are unifiable, the resulting constraint $x \neq a \wedge x \neq b \wedge x \neq c \wedge x \neq d$ has no solutions. A refinement of $P(x,x)$ is

$$\begin{array}{l}(P(x,x)\,;\, x \neq a \wedge x \neq b)\\ (P(a,a)\,;\,\top)\\ (P(b,b)\,;\,\top)\end{array}$$

$(P(x,y); x \neq a \wedge x \neq b \wedge y \neq c \wedge y \neq d)$ shares no ground instances with the approximations of the refined clauses.

Furthermore, there can be several straight terms $s$ that distinguish the same $t$ and $t'$. For example, for $g(f(a),a)$ and $g(f(b),b)$, both $g(f(a),v)$ and $g(v,a)$ are feasible for refinement but we prefer the shorter second term. Further, for $g(f(f(a)),x)$ and $g(f(f(b)),f(x))$, we use $g(f(f(a)),v)$ rather than $g(v,f(w))$ because the latter is derived from the occurrence conflict when trying to unify $x$ with $f(x)$ in the second argument. For this purpose, we use a modified unification algorithm that searches for an 'optimal' straight refinement term.

The rules are applied on triples $(\mathrm{G},\mathrm{S},\pi)$ where S is the result set containing straight terms and G is the set of unifications $(t \doteq t', s, p)$ which track previous decompositions with the straight term $s$ and position $p$. Given two terms $t$ and $t'$ that are not unifiable under the constraint $\pi$, the algorithm starts with the triple $(\{(t \doteq t', x, \varepsilon)\}, \{\}, \pi)$.

**Delete** $(\mathrm{G} \,\dot\cup\, \{(t \doteq t, s, p)\}, \mathrm{S}, \pi) \;\Rightarrow\; (\mathrm{G}, \mathrm{S}, \pi)$

**Decompose** $(\mathrm{G} \,\dot\cup\, \{(f(t_1,\ldots,t_n) \doteq f(t'_1,\ldots,t'_n), s, p)\}, \mathrm{S}, \pi)$
$\Rightarrow\; (\mathrm{G} \,\dot\cup\, \{(t_i \doteq t'_i, (s[p/f(v_1,\ldots,v_n)]), p.i) \mid 1 \leq i \leq n\}, \mathrm{S}, \pi)$

where $v_1,\ldots,v_n$ are fresh variables in $s$.

**Conflict** $(\mathrm{G} \,\dot\cup\, \{(f(\bar{t}) \doteq g(\bar{s}), s, p)\}, \mathrm{S}, \pi) \;\Rightarrow\; (\mathrm{G}, \mathrm{S} \cup \{s[p/f(\overline{v})], s[p/g(\overline{w})]\}, \pi)$

where $\overline{v}$ and $\overline{w}$ are fresh variables in $s$.

**Swap** $(\mathrm{G} \,\dot\cup\, \{(f(\bar{t}) \doteq x, s, p)\}, \mathrm{S}, \pi) \;\Rightarrow\; (\mathrm{G} \,\dot\cup\, \{(x \doteq f(\bar{t}), s, p)\}, \mathrm{S}, \pi)$

**Constraint** $(\mathrm{G} \,\dot\cup\, \{(x \doteq t, s, p)\}, \mathrm{S}, \pi)$
$\Rightarrow\; (\mathrm{G}, \mathrm{S} \cup \{s[p/s'\rho] \mid x \neq s' \in \pi \text{ and } t \neq s'{\downarrow} = \bot\}, \pi)$

where $s$ and $s'\rho$ are variable disjoint and $\pi\{x \mapsto t\}{\downarrow} = \bot$.

**Eliminate** $(\mathrm{G} \,\dot\cup\, \{(x \doteq t, s, p)\}, \mathrm{S}, \pi) \;\Rightarrow\; (\mathrm{G}\{x \mapsto t\}, \mathrm{S}, \pi\{x \mapsto t\})$

where $x \notin \mathrm{vars}(t)$, $\pi\{x \mapsto t\}$ is solvable, and $(t \doteq t', s, p)\sigma = (t\sigma \doteq t'\sigma, s, p)$.

The generalized unification terminates. This can be shown by a measure $(n, M)$ where $n$ is the number of different variables in $G$ and $M$ is the multiset of term depths of the left-hand sides of the equations $t \doteq t'$ in $G$. The well-defined ordering is a lexicographic combination of $>$ on the naturals and the extension of $>$ on the naturals to multisets. The rules Delete, Decompose, Conflict, Swap, and Constraint rules each strictly lower $M$ but leave the number $n$ of different variables unchanged. The Eliminate rule strictly lowers $n$.

Each rule preserves the invariant that if $\sigma$ is the total substitution applied on $G$ by the Eliminate rule and $t$ and $t'$ are the starting terms, then for any $(u \doteq u', s, p) \in G$, $t\sigma$ and $t'\sigma$ are instances of either $s[p/u]$ or $s[p/u']$, respectively, and any term $s \in S$ is a straight term that differentiates $t\sigma$ and $t'\sigma$.

While standard unification tracks the substitutions and fails upon reaching a conflict, this version instead stores for each found conflict the corresponding straight term. These straight terms are generated by extending a term with hole with each application of the decompose rule and completing them upon reaching a conflict. The added constraint rule catches a new case of conflict that can occur under the presence of constraints. For example, $f(x)$ and $f(f(a))$ are not unifiable under the constraint $x \neq f(v)$ and we generate $f(f(x))$ as the distinguishing straight term using the constraint rule. Additionally, each rule is given a priority defined by the order they are listed here, i.e., they are only applied if all previous rules are exhausted.

Once exhausted, one straight term is chosen from S depending on criteria such as depth, size, and the number of eliminate steps used. However, note that the rules are not complete. If the algorithm stops without result, we fall back on generating the straight term from a minimal ground core instead. This is because there are no cases for when $x$ occurs in $t$ or when $\pi\{x \mapsto t\}$ is unsolvable. The latter case could be refined but has proven too rare in experiments to be worth implementing. The former case is impossible to refine completely and therefore, we try to avoid it whenever possible (see also the next section).

## 5.3 Preprocessing Reflexive Predicates

The approximation-refinement cannot show satisfiability of the simple clause set with the two clauses $\rightarrow P(x, x)$ and $P(y, g(y)) \rightarrow$. The problem is that $\rightarrow P(x, x)$ cannot be refined in such a way that all instances of the lift-conflict $\rightarrow P(y, g(y))$ are excluded. The refinement loop instead ends up enumerating all $(\rightarrow P(g^i(x), g^i(x)); x \neq g(v))$ but $\rightarrow P(g^{i+1}(y), g^{i+2}(y))$ will always remain as a conflict clause.

We have not found a proper solution to this problem that works in all cases, but as a partial solution to this problem, if the input clause set contains reflexivity axioms such as $\rightarrow P(x, x)$, we tag each occurrence of $P$ as reflexive or irreflexive, i.e, whether an atom $P(s, t)$ is unifiable with $P(x, x)$ or not, by creating two new predicates $P_{\text{ref}}$ and $P_{\text{irr}}$. We essentially separate the predicate $P$ into reflexive and irreflexive parts. For the example, this looks like $\rightarrow P_{\text{ref}}(x, x)$ and $P_{\text{irr}}(y, g(y)) \rightarrow$.

Since the two literals were not resolvable anyway, this replacement is satisfiability equivalent. Now, the approximation is also satisfiable.

In general, for each predicate $P$ with a reflexivity axiom, We replace all occurrences of atoms $P(s, t)$ with $P_{\text{ref}}(s, t)$ and/or $P_{\text{irr}}(s, t)$. If $s$ and $t$ are not unifiable, we replace $P(s, t)$ with $P_{\text{irr}}(s, t)$. Otherwise, there is a most general unifier $\sigma$ of $s$ and $t$. In that case, we replace the clause $C \vee [\neg]P(s, t)$ with $C_{\text{irr}} = C \vee [\neg]P_{\text{irr}}(s, t)$ and $C_{\text{ref}} = C\sigma \vee [\neg]P_{\text{ref}}(s\sigma, t\sigma)$. If now $C_{\text{ref}}$ contains an atom $P_{\text{irr}}(s, s)$, we remove $C_{\text{ref}}$ again. We repeat this as long as $C_{\text{irr}}$ and $C_{\text{ref}}$ still contain the predicate $P$.

To give the idea why this is satisfiability equivalent, let $N'$ be the transformation of a clause set $N$ which has $P$ as its only predicate and contains the reflexivity axiom $\{\rightarrow P(x, x)\}$. If $\mathbb{I}$ is a Herbrand model of $N$ then $\{P_{\text{ref}}(s, t), P_{\text{irr}}(s, t) \mid P(s, t) \in \mathbb{I}\}$ is a model of $N'$ and if $\mathbb{I}$ is a Herbrand model of $N'$ then we can construct as the model of $N$ the set $\{P(s, s) \mid P_{\text{ref}}(s, s) \in \mathbb{I}\} \cup \{P(s, t) \mid s \neq t \text{ and } P_{\text{irr}}(s, t) \in \mathbb{I}\}$ .

For example, consider the satisfiable clause set $N$

$$\{\to P(x,x);\ P(f(x),f(y)) \to P(x,y);\ P(f(x),c) \to\}$$

which has only infinite models [6]. As with the first example, its approximation is unsatisfiable no matter the refinement. However, after tagging the reflexive predicate $P$ and deleting subsumed clauses, the resulting tagged set $N'$

$$\{\to P_{\mathrm{ref}}(x,x);\ P_{\mathrm{irr}}(f(x),f(y)) \to P_{\mathrm{irr}}(x,y);\ P_{\mathrm{irr}}(f(x),c) \to\}$$

and its approximation are both immediately saturated. With this the approximation-refinement can show the satisfiability of $N'$ and thereby also the satisfiability of the satisfiability equivalent $N$. However, note that while $N$ has only infinite models, $N'$ does have finite models. Hence, this does not show that the MSL fragment does not have the finite model property.

Lastly this method is not restricted to just binary predicates, but can be applied to any functions and predicates with arity larger than one [19].

## 6 Experiments

In the following we discuss several first-order clause classes for which FO-AR implemented in SPASS-AR immediately decides satisfiability but superposition and instantiation-based methods fail. We argue both according to the respective calculi and state-of-the-art implementations, in particular SPASS 3.9 [25], Vampire 4.1 [12,23], for ordered-resolution/superposition, iProver 2.5 [10] an implementation of Inst-Gen [11], and Darwin v1.4.5 [4] an implementation of the model evolution calculus [5]. All experiments were run on a 64-Bit Linux computer (Xeon(R) E5-2680, 2.70 GHz, 256 GB main memory). For Vampire and Darwin we chose the CASC-sat and CASC settings, respectively. For iProver we set the schedule to "sat" and SPASS, SPASS-AR were used in default mode. Please note that Vampire and iProver are portfolio solvers including implementations of several different calculi including superposition (ordered resolution), instance generation, and finite model finding. SPASS, SPASS-AR, and Darwin only implement superposition, FO-AR, and model evolution, respectively.

For the first example

$$P(x,y) \to P(x,z), P(z,y); \quad P(a,a)$$

and second example,

$$Q(x,x); \quad Q(v,w), P(x,y) \to P(x,v), P(w,y); \quad P(a,a)$$

the superposition calculus produces independently of the selection strategy and ordering an infinite number of clauses of form

$$\to P(a,z_1),\ P(z_1,z_2),\ \ldots,\ P(z_n,a).$$

Using linear approximation, however, FO-AR replaces $P(x,y) \to P(x,z), P(z,y)$ and $\to Q(x,x)$ with $P(x,y) \to P(x,z), P(z',y)$ and $\to Q(x,x')$, respectively. Consequently, ordered resolution derives $\to P(a,z_1), P(z_2,a)$ which subsumes any further inference clauses $\to P(a,z_1), P(z_2,z_3), P(z_4,a)$. Hence, saturation of the approximation terminates immediately. Both examples belong to the Bernays–Schönfinkel fragment, so model evolution (Darwin) and Inst-Gen (iProver) can decide them as well. Note that the concrete behavior of superposition is not limited to the above examples but potentially occurs whenever there are variable chains in clauses.

On the third problem

$$P(x, y) \to P(g(x), z); \quad P(a, a)$$

superposition derives all clauses of the form $\to P(g(\ldots g(a)\ldots), z)$. With a shallow approximation of $P(x, y) \to P(g(x), z)$ into $S(v) \to P(v, z)$ and $P(x, y) \to S(g(x))$, FO-AR (SPASS-AR) terminates after deriving $\to S(g(a))$ and $S(x) \to S(g(x))$. Again, model evolution (Darwin) and Inst-Gen (iProver) can also solve this example.

The next example

$$P(a); \quad P(f(a)) \to; \quad P(f(f(x))) \to P(x); \quad P(x) \to P(f(f(x)))$$

is already saturated under superposition. For FO-AR, the clause $P(x) \to P(f(f(x)))$ is replaced by $S(x) \to P(f(x))$ and $P(x) \to S(f(x))$. Then ordered resolution terminates after inferring $S(a) \to$ and $S(f(x)) \to P(x)$.

The Inst-Gen and model evolution calculi, however, fail. In either, a satisfying model is represented by a finite set of literals, i.e, a model of the propositional approximation for Inst-Gen and the trail of literals in case of model evolution. Therefore, there necessarily exists a literal $P(f^n(x))$ or $\neg P(f^n(x))$ with a maximal $n$ in these models. This contradicts the actual model where either $P(f^n(a))$ or $P(f^n(f(a)))$ is true. However, iProver can solve this problem using its built-in ordered resolution solver whereas Darwin does not terminate on this problem.

Lastly consider an example of the form

$$f(x) \approx x \to; \ \ f(f(x)) \approx x \to; \ \ \ldots; f^n(x) \approx x \to$$

which is trivially satisfiable, e.g., saturated by superposition, but any model has at least $n+1$ domain elements. Therefore, adding these clauses to any satisfiable clause set containing $f$ forces calculi that explicitly consider finite models to consider at least $n+1$ elements. The performance of final model finders [16] typically degrades in the number of different domain elements to be considered.

Combining each of these examples into one problem is then solvable by neither superposition, Inst-Gen, or model evolution and not practically solvable with increasing $n$ via testing finite models. For example, we tested

$$\begin{gathered} P(x, y) \to P(x, z), P(z, y); \quad P(a, a); \quad P(f(a), y) \to; \\ P(f(f(x)), y) \to P(x, y); \quad P(x, y) \to P(f(f(x)), y); \\ f(x) \approx x \to; , \ldots, f^n(x) \approx x \to; \end{gathered}$$

for $n = 20$ against SPASS, Vampire, iProver, and Darwin for more than one hour each without success. Only SPASS-AR solved it in less than one second.

For iProver we added an artificial positive equation $b \approx c$. For otherwise, iProver throws away all inequations while preprocessing. This is a satisfiability preserving operation, however, the afterwards found (finite) models are not models of the above clause set due to the collapsing of ground terms. Another such example is

**Table 1** The non-equality problems of the TPTP v.7.0.0 broken up by the ten largest domains

| Domain | All | Solved | % | Sat | Compl. | % | Unsat | Proof | % |
|---|---|---|---|---|---|---|---|---|---|
| SYN | 1128 | 806 | 71 | 275 | 166 | 60 | 875 | 640 | 73 |
| LCL | 564 | 115 | 20 | 139 | 48 | 35 | 430 | 67 | 16 |
| CSR | 379 | 142 | 37 | 4 | 1 | 25 | 372 | 141 | 38 |
| FLD | 281 | 58 | 21 | 3 | | | 183 | 58 | 32 |
| GEO | 264 | 172 | 65 | 16 | 1 | 6 | 249 | 171 | 69 |
| SWV | 260 | 84 | 32 | 16 | 16 | 100 | 248 | 68 | 27 |
| NLP | 206 | 129 | 63 | 210 | 115 | 55 | 14 | 14 | 100 |
| KRS | 188 | 94 | 50 | 85 | 30 | 35 | 103 | 64 | 62 |
| GRP | 129 | 69 | 53 | 62 | 11 | 18 | 77 | 58 | 75 |
| PUZ | 95 | 68 | 72 | 23 | 16 | 70 | 70 | 52 | 74 |

The columns show total number versus number solved and the respective percentage, additionally separate columns for satisfiable and unsatisfiable problems

$$\begin{aligned}
&\{\rightarrow P(a,a,a);\\
&\quad P(x,y',z'), P(x',y,z) \rightarrow P(f^2(x),y,z);\\
&\quad P(x',y,z'), P(x,y',z) \rightarrow P(x,f^3(y),z);\ P(x',y',z), P(x,y,z') \rightarrow P(x,y,f^5(z));\\
&\quad P(f^2(x),y,z) \rightarrow P(x,y,z);\ P(x,f^3(y),z) \rightarrow P(x,y,z);\ P(x,y,f^5(z)) \rightarrow P(x,y,z);\\
&\quad P(f(a),y,z) \rightarrow;\ P(x,f(a),z) \rightarrow;\ P(x,f^2(a),z) \rightarrow;\\
&\quad P(x,y,f(a)) \rightarrow;\ P(x,y,f^2(a)) \rightarrow;\ P(x,y,f^3(a)) \rightarrow;\ P(x,y,f^4(a)) \rightarrow\}
\end{aligned}$$

SPASS-AR saturates its approximation in under one second, while SPASS, Vampire, iProver, and Darwin each cannot solve it within a test-run of over one hour. Note that while again finite models exist, they require at least 30 domain elements which is too many for the finite model finders build into iProver and Vampire.

Additionally, we have tested SPASS-AR on the non-equality problems in the TPTP version 7.0.0 [18]. The problems were run for one hour each using a cluster with 64-Bit Linux servers (Intel Xeon E5620 @ 2.40 GHz, 6x 8GiB DDR3 1067 MHz ECC memory). Overall, SPASS-AR solved 2277 of the 4130 problems (55%), 1803 of the 3075 unsatisfiable problems (59%), and 484 of the 927 satisfiable problems (52%). A detailed breakdown of the result for individual problem classes is presented in Table 1. On average SPASS-AR refines a solved problem 8.5 times, 2.9 times for satisfiable and 10.1 times for unsatisfiable problems. 1403 problems (62%) are solved without requiring a refinement, 409 (84%) are satisfiable and 994 (55%) unsatisfiable. For solved problems that are refined there are on average 21.3 refinements, 17.3 refinements for satisfiable and 21.6 refinements for unsatisfiable problems, and each with a median of 10.

For comparison, under the same conditions with default schedules, SPASS v3.9 solved 2852 problems, Vampire v4.1 solved 3152, and iProver v2.7 solved 3565. SPASS-AR solved 133 problems that SPASS did not solve, 116 problems that Vampire did not solve, and 16 problems that iProver did not solve. When restricted to using only the Inst-Gen calculus, iProver v2.7 solved 3433 problems with 19 unsolved problems solved by SPASS-AR.

341 problems are already in the MSL fragment of which SPASS-AR solves 320 (94%). Most of the unsolved MSL problems are actually large ground problems and therefore better

**Table 2** The MSL approximation distance table

| Distance | Total | Solved | % | SumT | SumT % | SumS | SumS % |
|---|---|---|---|---|---|---|---|
| 0 | 341 | 320 | 94 | 341 | 9 | 320 | 14 |
| 1 | 149 | 146 | 98 | 490 | 12 | 466 | 21 |
| 2 | 147 | 123 | 84 | 637 | 16 | 589 | 26 |
| 3 | 81 | 67 | 83 | 718 | 18 | 656 | 29 |
| 4 | 137 | 98 | 72 | 855 | 22 | 754 | 33 |
| 5-10 | 692 | 372 | 54 | 1547 | 39 | 1126 | 50 |
| 11–20 | 459 | 241 | 53 | 2006 | 51 | 1367 | 60 |
| 21–30 | 217 | 88 | 40 | 2223 | 56 | 1455 | 64 |
| 31–40 | 290 | 84 | 29 | 2508 | 64 | 1539 | 68 |
| 41–50 | 187 | 46 | 25 | 2689 | 68 | 1585 | 70 |
| 51–100 | 517 | 210 | 41 | 3196 | 81 | 1795 | 79 |
| 101–200 | 355 | 76 | 21 | 3517 | 89 | 1871 | 82 |
| 201–300 | 150 | 117 | 78 | 3649 | 93 | 1988 | 87 |
| 301–400 | 239 | 207 | 87 | 3860 | 98 | 2195 | 97 |
| 401+ | 80 | 78 | 98 | 3940 | 100 | 2273 | 100 |

The first block of three columns shows the total number of problems with the given distance, as well as the respective number of solved problems and their share compared to the total. The second block lists the total number of problems with distance less or equal than the given distance, as well as the percentage compared to the whole set of problems. The last block shows the analogous result for solved problems

**Table 3** Overall solved non-equality problems from TPTP v.7.0.0

| Problems | SPASS-AR | SPASS 3.9 | Vampire 4.1 | iProver 2.7 |
|---|---|---|---|---|
| 4130 | 2277 | 2852 | 3152 | 3565 |

suited for a propositional satisfiability solver. Additionally, we have measured the MSL approximation distance of each problem, i.e., the number of non-satisfiability equivalent transformation steps required to approximate the problem into MSL. Table 2 shows that most problems are very close to the MSL fragment with about a third at distance below ten and half below twenty. Furthermore, for problems very close to MSL (distance 0–3) SPASS-AR has a high success-rate of 91% where 20% of the solved problems are satisfiable. Surprisingly, problems with a large distance (200+) are also solved with an 86% success-rate of which 99% are unsatisfiable. Note that 190 problems are missing from the total in Table 2 because SPASS-AR was unable to compute their MSL distance due to problem size.

As mentioned, certain lift-conflicts cannot be fully refined away. For example, when the conflicting instantiations, $t$ and $t'$, cannot be unified because $t$ occurs as a subterm in $t'$. Of the 2277 solved problems, only 198 (9%) contain such an occurrence-conflict, of which all are unsatisfiable problems. On the other hand, 755 of the 1824 unsolved problems (41%) reach at least one occurrence-conflict before time-out; 129 out of 442 are satisfiable (29%) and 532 out of 1265 are unsatisfiable (42%).

In summary, the abstraction refinement calculus implemented in SPASS-AR solves fewer problems from the TPTP version 7.0.0 than state-of-the-art implementations of superposition, e.g., SPASS. Portfolio solvers are even more successful, see Vampire, iProver, Table 3. On the other hand already the rather prototypical implementation of FO-AR in SPASS-AR can solve problems from the TPTP the other provers cannot solve. There are problem classes where FO-AR is in principle superior to superposition, model evolution, and instance generation. Of course, there are problem classes for which the other calculi are superior to FO-AR. There is room for improvement by a more sophisticated and dedicated implementation of FO-AR in SPASS-AR. The decision procedure for the MSL fragment is a simple instance of SPASS not dedicated to the fragment. The FO-AR calculus has a great potential of reusing inferences. However, the current prototypical implementation SPASS-AR does not explore this potential. Table 2 shows that this may result in a significant gain in performance.

## 7 Conclusion

In conclusion, we have shown the decidability of the MSL fragment. The MSL fragment is instrumented by the approximation-refinement calculus FO-AR and implemented in SPASS-AR. SPASS-AR is an extension of SPASS-3.9. Aside from the three practical aspects explained in Sect. 5, there are further possible improvements not mentioned and yet to be implemented.

For one we can reduce repeated work by not resetting the MSL solver for each loop but instead keep derived clauses in the proof state that are unaffected by the refinement. Another field of future work is to further specialize the MSL solver by taking advantage of the structure of MSL clauses. For example, any non-Horn clause without selected literals has the form $\Gamma_1, \ldots, \Gamma_n \rightarrow E_1, \ldots, E_n$ where each $E_i$ is maximal but the $\Gamma_i$, $E_i$ are pairwise variable disjoint. This means that these clauses can be split into Horn segments $\Gamma_i \rightarrow E_i$ which can drastically reduce the search space.

Section 6 showed FO-AR is superior to superposition, instantiation-based methods on certain classes of clause sets. Of course, there are also classes of clause sets where superposition and instantiation-based methods are superior to FO-AR, e.g., for unsatisfiable clause sets where the structure of the clause set forces FO-AR to enumerate failing ground instances due to the approximation in a bottom-up way. An extension to FO-AR detecting such situations and then moving to a different type of approximation is another direction of potential future research.

Our prototypical implementation SPASS-AR cannot compete with systems such as iProver or Vampire on the respective CASC categories of the TPTP [18]. This is already due to the fact that they are all meanwhile portfolio solvers. For example, iProver contains an implementation of ordered resolution and Vampire an implementation of Inst-Gen. Our results, Sect. 6, however, show that these systems may benefit from FO-AR by adding it to their portfolio.

The DEXPTIME-completeness result for MSLH strongly suggest that both the MSLH and also our MSL(SDC) fragment have the finite model property. Meanwhile this has been shown [19]. Therefore, finite model finding approaches are complete on MSL(SDC), however a saturated MSL(SDC) clause set constitutes an exponentially more compact representation [19]. The models generated by FO-AR and superposition are typically infinite. It remains an open problem, even for fragments enjoying the finite model property, e.g., the first-order monadic fragment, to design a calculus that combines explicit finite model finding with a structural representation of infinite models. For classes that have no finite models this problem seems to become even more difficult. To the best of our knowledge, SPASS is currently the only prover that can show satisfiability of the clauses $R(x, x) \rightarrow$; $R(x, y), R(y, z) \rightarrow R(x, z)$; $R(x, g(x))$ due to an implementation of chaining [2,17]. Apart from the superposition calculus, it is unknown to us how the specific inferences for transitivity can be combined with any of the other discussed calculi, including the abstraction refinement calculus introduced in this paper.

Finally, there are not many results on calculi that operate with respect to models containing positive equations. Even for fragments that are decidable with equality, such as the Bernays–Schoenfinkel–Ramsey fragment or the monadic fragment with equality, there seem currently no convincing suggestions compared to the great amount of techniques for these fragments without equality. Adding positive equations to MSL(SDC) while keeping decidability is, to the best of our current knowledge, only possible for at most linear, shallow equations $f(x_1, \dots, x_n) \approx h(y_1, \dots, y_n)$ [9]. However, approximation into such equations from an equational theory with nested term occurrences typically results in an almost trivial equational theory. So this does not seem to be a very promising research direction.

**Acknowledgements** We thank the reviewers as well as Konstantin Korovin and Giles Reger for a number of important remarks.

## References

1. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Logic Comput. **4**(3), 217–247 (1994). Revised version of Max-Planck-Institut für Informatik technical report, MPI-I-91-208, 1991
2. Bachmair, L., Ganzinger, H.: Ordered chaining calculi for first-order theories of transitive relations. J. ACM **45**(6), 1007–1049 (1998)
3. Baeten, J.C.M., Bergstra, J.A., Klop, J.W., Weijland, W.P.: Term-rewriting systems with rule priorities. Theor. Comput. Sci. **67**(2 & 3), 283–301 (1989)
4. Baumgartner, P., Fuchs, A., Tinelli, C.: Implementing the model evolution calculus. Int. J. Artif. Intell. Tools **15**(1), 21–52 (2006). https://doi.org/10.1142/S0218213006002552
5. Baumgartner, P., Tinelli, C.: The model evolution calculus. In: Baader, F. (ed.) Automated Deduction—CADE-19, 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28–August 2, 2003, Proceedings, Lecture Notes in Computer Science, vol. 2741, pp. 350–364. Springer, Berlin (2003). https://doi.org/10.1007/978-3-540-45085-6_32
6. Caferra, R., Leitsch, A., Peltier, N.: Automated Model Building. Applied Logic, vol. 31. Springer, Berlin (2004)
7. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata (2007). Release October, 12th 2007
8. Goubault-Larrecq, J.: Deciding $\mathscr{H}_1$ by resolution. Inf. Process. Lett. **95**(3), 401–408 (2005). https://doi.org/10.1016/j.ipl.2005.04.007
9. Jacquemard, F., Meyer, C., Weidenbach, C.: Unification in extensions of shallow equational theories. In: Nipkow, T. (ed.) Rewriting Techniques and Applications, 9th International Conference, RTA-98, LNCS, vol. 1379, pp. 76–90. Springer, Berlin (1998)
10. Korovin, K.: iprover—an instantiation-based theorem prover for first-order logic (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12–15, 2008, Proceedings, Lecture Notes in Computer Science, vol. 5195, pp. 292–298. Springer, Berlin (2008). https://doi.org/10.1007/978-3-540-71070-7_24
11. Korovin, K.: Inst-Gen—a modular approach to instantiation-based automated reasoning. In: Programming Logics—Essays in Memory of Harald Ganzinger, pp. 239–270 (2013). https://doi.org/10.1007/978-3-642-37651-1_10
12. Kovács, L., Voronkov, A.: First-order theorem proving and vampire. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification—25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13–19, 2013. Proceedings, Lecture Notes in Computer Science, vol. 8044, pp. 1–35. Springer, Berlin (2013)
13. Seidl, H., Reuß, A.: Extending H1-clauses with disequalities. Inf. Process. Lett. **111**(20), 1007–1013 (2011)
14. Seidl, H., Reuß, A.: Extending $\mathscr{H}$_1-clauses with path disequalities. In: L. Birkedal (ed.) Foundations of Software Science and Computational Structures—15th International Conference, FOSSACS 2012, Held

as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24–April 1, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7213, pp. 165–179. Springer (2012)
15. Seidl, H., Verma, K.N.: Cryptographic protocol verification using tractable classes of horn clauses. In: Reps, T., Sagiv, M., Bauer, J. (eds.) Program Analysis and Compilation, Theory and Practice. Lecture Notes in Computer Science, pp. 97–119. Springer, Berlin (2007)
16. Slaney, J.K., Surendonk, T.: Combining finite model generation with theorem proving: problems and prospects. In: Baader, F., Schulz, K.U. (eds.) Frontiers of Combining Systems, First International Workshop FroCoS 1996, Munich, Germany, March 26–29, 1996, Proceedings, Applied Logic Series, vol. 3, pp. 141–155. Kluwer Academic Publishers, Berlin (1996)
17. Suda, M., Weidenbach, C., Wischnewski, P.: On the saturation of YAGO. In: Automated Reasoning, 5th International Joint Conference, IJCAR 2010, LNAI, vol. 6173, pp. 441–456. Springer, Edinburgh (2010)
18. Sutcliffe, G.: The TPTP problem library and associated infrastructure: the FOF and CNF parts, v3.5.0. J. Autom. Reason. **43**(4), 337–362 (2009)
19. Teucke, A., Voigt, M., Weidenbach, C.: On the expressivity and applicability of model representation formalisms. In: Herzig, A., Popescu, A. (eds.) Frontiers of Combining Systems—12th International Symposium, FroCoS 2019, London, UK, September 4–6, 2019, Proceedings, Lecture Notes in Computer Science, vol. 11715, pp. 22–39. Springer, Berlin (2019)
20. Teucke, A., Weidenbach, C.: First-order logic theorem proving and model building via approximation and instantiation. In: Lutz, C., Ranise, S. (eds.) Frontiers of Combining Systems: 10th International Symposium, FroCoS 2015, Wroclaw, Poland, September 21–24, 2015, Proceedings, pp. 85–100. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24246-0_6
21. Teucke, A., Weidenbach, C.: Ordered resolution with straight dismatching constraints. In: P. Fontaine, S. Schulz, J. Urban (eds.) Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning co-located with International Joint Conference on Automated Reasoning (IJCAR 2016), Coimbra, Portugal, July 2nd, 2016. CEUR Workshop Proceedings, vol. 1635, pp. 95–109. CEUR-WS.org (2016). http://ceur-ws.org/Vol-1635/paper-09.pdf
22. Teucke, A., Weidenbach, C.: Decidability of the monadic shallow linear first-order fragment with straight dismatching constraints. arXiv:1703.02837 (2017)
23. Voronkov, A.: AVATAR: the architecture for first-order theorem provers. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification—26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18—22, 2014. Proceedings, Lecture Notes in Computer Science, vol. 8559, pp. 696–710. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-08867-9_46
24. Weidenbach, C.: Towards an automatic analysis of security protocols in first-order logic. In: Ganzinger, H. (ed.) 16th International Conference on Automated Deduction, CADE-16, LNAI, vol. 1632, pp. 314–328. Springer, Berlin (1999)
25. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: Schmidt, R.A. (ed.) Automated Deduction—CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2–7, 2009. Proceedings, Lecture Notes in Computer Science, vol. 5663, pp. 140–145. Springer, Berlin (2009). https://doi.org/10.1007/978-3-642-02959-2_10