Decidability of the Monadic Shallow Linear First-Order Fragment with Straight Dismatching Constraints

Andreas Teucke, Christoph Weidenbach

Decidability of the Monadic Shallow Linear First-Order Fragment with Straight Dismatching Constraints*

Andreas Teucke Christoph Weidenbach
Max-Planck Institut für Informatik
Saarland Informatics Campus
66123 Saarbrücken

December 6, 2017

Abstract

The monadic shallow linear Horn fragment is well-known to be decidable and has many application, e.g., in security protocol analysis, tree automata, or abstraction refinement. It was a long standing open problem how to extend the fragment to the non-Horn case, preserving decidability, that would, e.g., enable to express non-determinism in protocols. We prove decidability of the non-Horn monadic shallow linear fragment via ordered resolution further extended with dismatching constraints and discuss some applications of the new decidable fragment.

1 Introduction

Motivated by the automatic analysis of security protocols, the monadic shallow linear Horn (MSLH) fragment was shown to be decidable in [22]. 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 $\mathcal{I}_N$ 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 [6] and shallow linear sort theories [8]. The models are typically infinite. The decidability result of MSLH clauses was rediscovered in the context of tree automata research [7] where in addition DEXPTIME-completeness of the MSLH fragment was shown. The fragment was further extended by disequality constraints [12, 13] still motivated by security protocol analysis [14]. 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

*Appeared in: Proceedings of CADE 26, LNCS 10395, pp. 202-219, Springer

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 disequations of the form $s \not\approx t$, where s and t are not unifiable. Due to the employed superposition calculus, such disequations 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 disequality constraints mentioned above [12, 13]. They do not strictly increase the expressiveness of the MSL theory, but enable up to exponentially more compact saturations. For example, the constrained clause

$$(S(x), T(y) \to S(f(x, y)); y \neq f(x', f(a, y')))$$

over constants a, b describes the same set of ground clauses as the six unconstrained clauses

$$S(x), T(a) \to S(f(x, a)) \qquad S(x), T(b) \to S(f(x, b)) \qquad \dots$$
$$S(x), T(f(b, y')) \to S(f(x, f(b, y')))$$
$$S(x), T(f(f(x'', y''), y')) \to S(f(x, f(f(x'', y''), y')).$$

Furthermore, for a satisfiability equivalent transformation into MSL clauses, the nested terms in the positive literals would have to be factored out by the introduction of further predicates and clauses. E.g., the first clause is replaced by the two MSL clauses $S(x), T(a), R(y) \to S(f(x, y))$ and $R(a)$ where R is a fresh monadic predicate. The constrained clause belongs to the MSL(SDC) fragment. Altogether, the resulting MSL(SDC) fragment is shown to be decidable in Section 3.

The introduction of straight dismatching constraints (SDCs) enables an improved refinement step of our approximation refinement calculus [18]. 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 $S(x), T(x) \to S(f(x, x))$ to $S(x), T(x), S(y), T(y) \to S(f(x, y))$ an instance $\{x \mapsto f(a, x'),\ y \mapsto f(b, y')\}$ is used in the proof, before [18] several clauses were needed to replace $S(x), T(x) \to S(f(x, x))$ in a refinement step in order to rule out this instance. With straight dismatching constraints the clause $S(x), T(x) \to S(f(x, x))$ is replaced by the two clauses $S(f(a, x)), T(f(a, x)) \to S(f(f(a, x), f(a, x)))$ and $(S(x), T(x) \to S(f(x, x)); x \neq f(a, y))$. For the improved approximation refinement approach (FO-AR) presented in this paper, any refinement step results in just two clauses, see Section 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 polynomial time, see Section 2.

In addition to the extension of the known MSLH decidability result and the improved approximation refinement calculus FO-AR, we discuss in Section 5 the potential of the MSL(SDC) fragment in the context of FO-AR, Theorem 4.12, and its prototypical implementation in SPASS-AR (`http://www.mpi-inf.mpg.de/fileadmin/inf/rg1/spass-ar.tgz`). It turns out that for clause sets containing certain structures, FO-AR is superior to ordered resolution/superposition [1] and instance generating methods [10]. The paper ends with a discussion on challenges and future research directions, Section 6. In favor of many illustrating examples, most proofs and further technical details can be found in [20].

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 $\mathcal{V}$ sets of variables. $\overline{L}$ is the complement of L. The signature $\Sigma = (\mathcal{F}, \mathcal{P})$ consists of two disjoint, non-empty, in general infinite sets of function and predicate symbols $\mathcal{F}$ and $\mathcal{P}$, respectively. The set of all *terms* over variables $\mathcal{V}$ is $\mathcal{T}(\mathcal{F}, \mathcal{V})$. If there are no variables, then terms, literals and clauses are called *ground*, respectively. A *substitution* σ is denoted by pairs $\{x \mapsto t\}$ and its update at x by $\sigma[x \mapsto t]$. A substitution σ is a *grounding* substitution for $\mathcal{V}$ if $x\sigma$ is ground for every variable $x \in \mathcal{V}$.

The set of *free* variables of an atom A (term t) denoted by $\mathrm{vars}(A)$ ($\mathrm{vars}(t)$). A *position* is a sequence of positive integers, where ε 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 $f(s_1, \ldots, s_n)$ is called *straight*, if $s_1, \ldots, s_n$ 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 $\Gamma \to \Delta$ where the atoms in the multiset Δ (the *succedent*) denote the positive literals and the atoms in the multiset Γ (the *antecedent*) the negative literals. We write $\Box$ for the empty clause. If Γ is empty we omit $\to$, e.g., we can write $P(x)$ as an alternative of $\to P(x)$. We abbreviate disjoint set union with sequencing, for example, we write $\Gamma, \Gamma' \to \Delta, L$ instead of $\Gamma \cup \Gamma' \to \Delta \cup \{L\}$. A clause $E, E, \Gamma \to \Delta$ is equivalent to $E, \Gamma \to \Delta$ and we call them equal *modulo duplicate literal elimination*. If every term in Δ is shallow, the clause is called *positive shallow*. If all atoms in Δ are linear and variable disjoint, the clause is called *positive linear*. A clause $\Gamma \to \Delta$ is called an *MSL* clause, if it is (i) positive shallow and linear, (ii) all occurring predicates are monadic, (iii) no equations occur in Δ, and (iv) no equations occur in Γ or $\Gamma = \{s \approx t\}$ and Δ is empty where s and t are not unifiable. *MSL* is the first-order clause fragment consisting of MSL clauses. Clauses $\Gamma, s \approx t \to \Delta$ where Γ, Δ 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. The 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.

A *Herbrand interpretation* $\mathcal{I}$ is a - possibly infinite - set of ground atoms. A ground atom

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

A disequation $t \neq s$ is an *atomic straight dismatching constraint* if s and t are variable disjoint terms and s is straight. A straight dismatching constraint π is a conjunction of atomic straight dismatching constraints. Given a substitution σ, $\pi\sigma = \bigwedge_{i \in I} \quad t_i\sigma \neq s_i$. $\mathrm{lvar}(\pi) := \bigcup_{i \in I} \mathrm{vars}(t_i)$ are the left-hand variables of π and the depth of π is the maximal term depth of the s_i. A *solution* of π is a grounding substitution δ such that for all $i \in I$, $t_i\delta$ is not an instance of s_i, i.e., there exists no σ 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 [19]. $\top$ 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.

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

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

A pair of a clause and a constraint $(C; \pi)$ is called a *constrained clause*. Given a substitution σ, $(C; \pi)\sigma = (C\sigma; \pi\sigma)$. $C\delta$ is called a ground clause of $(C; \pi)$ if δ is a solution of π. $\mathcal{G}((C; \pi))$ is the set of ground instances of $(C; \pi)$. If $\mathcal{G}((C; \pi)) \subseteq \mathcal{G}((C'; \pi'))$, then $(C; \pi)$ is an instance of $(C'; \pi')$. If $\mathcal{G}((C; \pi)) = \mathcal{G}((C'; \pi'))$, then $(C; \pi)$ and $(C'; \pi')$ are called variants. A Herbrand interpretation $\mathcal{I}$ satisfies $(C; \pi)$, if $\mathcal{I} \vDash \mathcal{G}((C; \pi))$. A constrained clause $(C; \pi)$ is called *redundant* in N if for every $D \in \mathcal{G}((C; \pi))$, there exist $D_1, \ldots, D_n$ in $\mathcal{G}(N)^{\prec D}$ such that $D_1, \ldots, D_n \vDash D$. A constrained clause $(C'; \pi')$ is called a *condensation* of $(C; \pi)$ if $C' \subset C$ and there exists a substitution σ 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 $\mathcal{G}(N)$ is called an unsatisfiable core of N.

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 [19] 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)**.** Given an MSL(SDC) clause $(C;\pi) = (S_1(t_1), \ldots, S_n(t_n) \to P_1(s_1), \ldots, P_m(s_m);\pi)$. The Superposition Selection function sel is defined by $S_i(t_i) \in \mathrm{sel}(C)$ if (1) t_i is not a variable or (2) $t_1, \ldots, t_n$ are variables and $t_i \notin \mathrm{vars}(s_1, \ldots, s_m)$ or (3) $\{t_1, \ldots, t_n\} \subseteq \mathrm{vars}(s_1, \ldots, s_m)$ and for some $1 \leq j \leq m$, $s_j = t_i$.

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

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

Note that given an MSL(SDC) clause $(C;\pi) = (S_1(t_1), \ldots, S_n(t_n) \to P_1(s_1), \ldots P_m(s_m);\pi)$, 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 δ of π such that for all literals B in C, $B\delta \preceq A\delta$ $[B\delta \prec A\delta]$.

Definition 3.3 (SDC-Resolution)**.**

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

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

Definition 3.4 (SDC-Factoring)**.**

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

1. $\sigma = \mathrm{mgu}(A, B)$
2. $\mathrm{sel}(\Gamma \to \Delta, A, B) = \emptyset$
3. $A\sigma$ is maximal in $(\Gamma \to \Delta, A, B;\pi)\sigma$
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 disequations, i.e., clauses $s \approx t \to$ 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 $s \approx t \to$ affect the models of satisfiable problems. Constrained Resolution and Factoring are sound.

Definition 3.5 (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 $\mathcal{G}((R;\pi)) \subseteq \mathcal{G}(N)$.

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

Definition 3.6 (Partial Minimal Model Construction)**.** Given a constrained clause set N, an ordering $\prec$ and the selection function sel, we construct an interpretation $\mathcal{I}_N$ for N, called a partial model, inductively as follows:

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

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

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

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

Lemma 3.7 (Ordered SDC Resolution Completeness)**.** Let N be a constrained clause set saturated up to redundancy by ordered SDC-resolution with selection. Then N is unsatisfiable, if and only if $\square \in \mathcal{G}(N)$. If $\square \notin \mathcal{G}(N)$ then $\mathcal{I}_N \models N$.

Lemma 3.8. Let N be a set of MSL(SDC) clauses without variants or uncondensed clauses over a finite signature Σ. 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 $\mathrm{vars}(t) \cap \mathrm{vars}(\Delta) = \emptyset$.

Lemma 3.9 (Finite Saturation)**.** Let N be an MSL(SDC) clause set. Then N can be finitely saturated up to redundancy by SDC-resolution with selection function sel.

Theorem 3.10 (MSL(SDC) Decidability)**.** Satisfiability of the MSL(SDC) first-order fragment is decidable.

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 [18].

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), Section 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 anymore 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 Section 3 that a restriction to Horn clauses is not required for decidability anymore. 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 nondeterministically 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 $\mu^T_P(P(\vec{t})) := T(f_p(\vec{t}))$ and $\mu^T_P(Q(\vec{s})) := Q(\vec{s})$ for $P \neq Q$. The function is extended to [constrained] clauses, clause sets and interpretations. Given a signature Σ with non-monadic predicates $P_1, \ldots, P_n$, define $\mu^T_\Sigma(N) := \mu^T_{P_1}(\ldots(\mu^T_{P_n}(N))\ldots)$ and $\mu^T_\Sigma(\mathcal{I}) := \mu^T_{P_1}(\ldots(\mu^T_{P_n}(\mathcal{I}))\ldots)$.

Monadic $\quad N \Rightarrow_{\mathrm{MO}} \mu^T_P(N)$

provided P is a non-monadic predicate in the signature of N.

Shallow
$$N \dot\cup \{(\Gamma \to E[s]_p, \Delta; \pi)\} \Rightarrow_{\mathrm{SH}} N \cup \{(S(x), \Gamma_l \to E[p/x], \Delta_l; \pi);\ (\Gamma_r \to S(s), \Delta_r; \pi)\}$$

provided s is complex, $|p| = 2$, x and S fresh, $\Gamma_l\{x \mapsto s\} \cup \Gamma_r = \Gamma$, $\Delta_l \cup \Delta_r = \Delta$, $\{Q(y) \in \Gamma \mid y \in \mathrm{vars}(E[p/x], \Delta_l)\} \subseteq \Gamma_l$, $\{Q(y) \in \Gamma \mid y \in \mathrm{vars}(s, \Delta_r)\} \subseteq \Gamma_r$.

Linear 1
$$N \dot\cup \{(\Gamma \to \Delta, E'[x]_p, E[x]_q; \pi)\} \Rightarrow_{\mathrm{LI}} N \cup \{(\Gamma\sigma, \Gamma \to \Delta, E'[x]_p, E[q/x']; \pi \wedge \pi\sigma)\}$$

provided x' is fresh and $\sigma = \{x \mapsto x'\}$.

Linear 2
$$N \dot\cup \{(\Gamma \to \Delta, E[x]_{p,q}; \pi)\} \Rightarrow_{\mathrm{LI}} N \cup \{(\Gamma\sigma, \Gamma \to \Delta, E[q/x']; \pi \wedge \pi\sigma)\}$$

provided x' is fresh, $p \neq q$ and $\sigma = \{x \mapsto x'\}$.

Refinement $N \dot\cup \{(C, \pi)\} \Rightarrow_{\text{Ref}} N \cup \{(C; \pi \wedge x \neq t), (C; \pi)\{x \mapsto t\}\}$
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, Γ and Δ are separated into Γ_l, Γ_r, Δ_l, and Δ_r, respectively. The separation can be almost arbitrarily chosen as long as no atom from Γ, Δ is skipped. However, the goal is to minimize the set of shared variables, i.e., the variables of $(\Gamma \to E[s]_p, \Delta; \pi)$ 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 Γ_l and Γ_r ensure that $S(x)$ atoms are not separated from the respective positive occurrence of x in subsequent shallow transformation applications.

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

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 Γ and π is not needed if x does not occur in Γ or π.

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.

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 $\mathcal{I}$, then $\mu_\Sigma^{-1}(\mathcal{I})$ is a model of N.

On the basis of $\Rightarrow_{\text{AP}}$ we can define an ancestor relation $\Rightarrow_{\text{A}}$ that relates clauses, literal occurrences, and variables with respect to approximation. This relation is needed in order to figure out the exact clause, literal, variable for refinement. The definition of $\Rightarrow_{\text{A}}$ itself is rather technical [20].

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 [18], the lifting process is identical with the exception that there is no case for the removed Horn transformation. We only update the definition of conflicting cores to consider constrained clauses.

Definition 4.5 (Conflicting Core)**.** A finite set of unconstrained clauses $N^\perp$ is a conflicting core of N if $N^\perp\sigma$ is an unsatisfiable core of N for all grounding substitutions σ. For a ground clause $D \in N^\perp\sigma$ and $(C;\pi) \in N$ such that $D \in \mathcal{G}((C;\pi))$, the clause $(C;\pi)$ is called the instantiated clause of D. We call $N^\perp$ complete if for every clause $C \in N^\perp$ and literal $L \in C$, there exists a clause $D \in N^\perp$ with $\overline{L} \in D$.

A conflicting core is a generalization of a ground unsatisfiability core that allows global variables to act as parameters. This enables more efficient lifting and refinement compared to a simple ground unsatisfiable core. We show some examples at the end of this section.

We discuss the potential lifting failures and the corresponding refinements only for the linear and shallow case because lifting the satisfiability equivalent monadic and refinement transformations always succeeds. To reiterate from our previous work: 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. We combine these two cases by introducing the notion of a lift-conflict.

Definition 4.6 (Conflict)**.** Let $N \cup \{(C,\pi)\} \Rightarrow_{\text{LI}} N \cup \{(C_a,\pi_a)\}$ and $N^\perp$ be a complete ground conflicting core of $N \cup \{(C_a,\pi_a)\}$. We call a conflict clause $C_c \in N^\perp$ with the instantiated clause (C_a,π_a) a lift-conflict if C_c is not an instance of (C,π) modulo duplicate literal elimination. Then, C_c is an instance of (C_a,π_a), which we call the conflict clause of C_c.

The goal of refinement is to instantiate the original parent clause in such a way that is both satisfiability equivalent and prevents the lift-conflict after approximation. Solving the refined approximation will then either necessarily produce a complete saturation or a new refutation proof, because its conflicting core has to be different. For this purpose, we use the refinement transformation to segment the original parent 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'.

$$\begin{array}{rcl} \to P(x,x) & \Rightarrow_{\text{LI}} & \to P(x,x') \\ P(a,b) \to & \Rightarrow^0_{\text{AP}} & P(a,b) \to \end{array}$$

The ground conflicting core of N' is

$$\begin{array}{c} \to P(a,b) \\ P(a,b) \to \end{array}$$

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. We 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, we use that the conflict clause and its ancestors have strong ties between their term structures and constraints.

Definition 4.7 (Constrained Term Skeleton)**.** The constrained term skeleton of a term t under constraint π, $\mathrm{skt}(t, \pi)$, 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 π are preserved. For (t, π) and a solution δ of π, $t\delta$ is called a ground instance of (t, π).

Lemma 4.8. 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 complete ground conflicting core of N_k. Let δ be a solution of π_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, π_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, π_0), then $L|_q = y$.

Proof. Idea. The proof is by induction on the length of the approximation $N_0 \Rightarrow^*_{\mathrm{AP}} N_k$ and a case distinction of the first transformation $N_0 \Rightarrow_{\mathrm{AP}} N_1$. Most cases are straightforward except for case (i) of the shallow transformation. Because $N_k^\perp$ is complete, any negative extraction literal $S(x)\delta$ matches some positive literal in $N_k^\perp$ which is necessarily an instance of $S(s)$, the extraction term. Therefore, the original term structure is preserved even after subterm extraction. □

Next, we define the notion of descendants and descendant relations to connect lift-conflicts in ground conflicting cores with their corresponding ancestor clauses. The goal, hereby, is that if a ground clause D is not a descendant of a clause in N, then it can never appear in a conflicting core of an approximation of N.

Definition 4.9 (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, we call D a *descendant* of $(C; \pi)$ and 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 generating new distinct descendant relations (Lemma 4.10), a finite number of refinement steps can remove the lift-conflict C_c from the descendants of $(C; \pi)$ (Lemma 4.11). 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 identical but their S-predicate extract different terms in $(C; \pi)$. For example, $P(a) \to S(f(a))$ 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, we have 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 4.10. Let $N_0 = N \cup \{(C;\pi)\} \Rightarrow_{\text{Ref}} N \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\} = N_1$ be a refinement transformation and D a ground clause. If there is a $[(C;\pi \wedge x \neq t), N_1] \Rightarrow^*_{\text{A}} [(C';\pi'), N_2]$- or $[(C;\pi)\{x \mapsto t\}, N_1] \Rightarrow^*_{\text{A}} [(C';\pi'), N_2]$-descendant relation $\Rightarrow^1_D$, then there is an equal $[(C;\pi), N_0] \Rightarrow^*_{\text{A}} [(C';\pi'), N_2]$-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 $[r, L, (C;\pi \wedge x \neq t), N_1] \Rightarrow^*_{\text{A}} [\varepsilon, L', (C';\pi'), N_2]$. Because $[r, L, (C;\pi), N_0] \Rightarrow_{\text{A}} [r, L, (C;\pi \wedge x \neq t), N_1]$, $L' \Rightarrow^0_D (L, r)$. If D is a descendant of $(C;\pi)\{x \mapsto t\}$, the proof is analogous. □ □

Lemma 4.11 (Refinement)**.** Let $N \Rightarrow_{\text{AP}} N'$ and $N^\perp$ be a complete ground conflicting core of N'. If $C_c \in N^\perp$ is a lift-conflict, then there exists a finite refinement $N \Rightarrow^*_{\text{Ref}} N_R$ 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 4.12 (Soundness and Completeness of FO-AR)**.** Let N be an unsatisfiable 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 Lemma 4.3 and Lemma 4.11, 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 4.11 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 [18].

In Lemma 4.11, we segment the conflict clause's immediate parent clause. If the lifting later successfully passes this point, the refinement is lost and will be possibly repeated. Instead, we can refine any ancestor of the conflict clause as long as it contains the ancestor of the variable used in the refinement. By Lemma 4.8-(iii), such an ancestor will contain the ancestor variable at the same positions. If we refine the ancestor in the original clause set, the refinement is permanent because lifting the refinement steps always succeeds. Only variables introduced by shallow transformation cannot be traced to the original clause set. However, these shallow variables are already linear and the partitioning in the shallow transformation can be chosen such that they are not shared variables. Assume a shallow, shared variable y, that is used to extract the term t, in the shallow transformation of $\Gamma \to E[s]_p, \Delta$ into $S(x), \Gamma_l \to E[p/x], \Delta_l$ and $\Gamma_r \to S(s), \Delta_r$. Since $\Delta_l \dot\cup \Delta_r = \Delta$ is a partitioning, y can only appear in either $E[p/x], \Delta_l$ or $S(s), \Delta_r$. If $y \in \text{vars}(E[p/x], \Delta_l)$ we instantiate Γ_r with $\{y \mapsto t\}$ and Γ_l, otherwise. Now, y is no longer a shared variable.

The refinement Lemmas only guarantee a refinement for a given ground conflicting core. In practice, however, conflicting cores contain free variables. We can always generate a ground conflicting core by instantiating the free variables with ground terms. However, if we only exclude a single ground case via refinement, next time the new conflicting core will likely have overlaps with the previous one. Instead, we can often remove all ground instances of a given conflict clause at once.

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 $N = \{P(x,x); P(f(x,a), f(y,b)) \to\}$. 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

$$(P(x,x)\,;x \neq f(v,a))$$
$$(P(f(x,a),f(x,a))\,;\top)$$

$P(f(x,a),f(y,b))$ 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 an second argument are different but the same in $f(x,x)$. A refinement of $P(x,x)$ is

$$(P(x,x)\,;x \neq f(a,v))$$
$$(P(f(a,x),f(a,x)))\,;x \neq a)$$
$$(P(f(a,a),f(a,a)))\,;\top)$$

$P(f(a,b),f(x,x))$ 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 $N = \{P(x,x); (P(x,y) \rightarrow; x \neq a \wedge x \neq b \wedge y \neq$ 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

$$(P(x,x)\,;x \neq a \wedge x \neq b)$$
$$(P(a,a)\,;\top)$$
$$(P(b,b)\,;\top)$$

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

Lastly, we should mention that there are cases where the refinement process does not terminate. For example, consider the clause set $N = \{P(x,x); P(y,g(y)) \rightarrow\}$. N is satisfiable but the linear transformation of N is unsatisfiable with conflict clause $P(y,g(y))$, which is not unifiable with $P(x,x)$. A refinement of $P(x,x)$ based on the ground instance $P(a,g(a))$ is

$$(P(x,x)\,;x \neq g(v))$$
$$(P(g(x),g(x))\,;\top)$$

While $P(y,g(y))$ is not an instance of the refined approximation, it shares ground instances with $P(g(x),g(x'))$. The new conflict clause is $P(g(y),g(g(y)))$ and the refinement will continue to enumerate all $P(g^i(x),g^i(x))$ instances of $P(x,x)$ without ever reaching a satisfiable approximation. Satisfiability of first-order clause sets is undecidable, so termination cannot be expected by any calculus, in general.

5 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 [23], Vampire 4.1 [11, 21], for ordered-resolution/superposition, iProver 2.5 [9] an implementation of Inst-Gen [10], 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, 2.70GHz, 256GB 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 inferences $\to P(a,z_1), P(z_2,z_3), P(z_4,a)$. 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 [15] 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 superpo-

sition, Inst-Gen, or model evolution and not practically solvable with increasing n via testing finite models. For example, we tested

$$P(x,y) \to P(x,z), P(z,y); \quad P(a,a); \quad P(f(a),y) \to;$$
$$P(f(f(x)),y) \to P(x,y); \quad P(x,y) \to P(f(f(x)),y);$$
$$f(x) \approx x \to;, \ldots, f^n(x) \approx x \to;$$

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 disequations 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.

6 Conclusion

The previous section 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.

Our prototypical implementation SPASS-AR cannot compete with systems such as iProver or Vampire on the respective CASC categories of the TPTP [17]. 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, Section 5, 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. However, we are not aware of any proof. If MSL(DSC) has the finite model property, the finite model finding approaches are complete on MSL(SDC). 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 $R(x,x) \to$; $R(x,y), R(y,z) \to R(x,z)$; $R(x,g(x))$ due to an implementation of chaining [2, 16]. 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 $f(x_1, \ldots, x_n) \approx h(y_1, \ldots, y_n)$ [8]. 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] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994. Revised version of Max-Planck-Institut für Informatik technical report, MPI-I-91-208, 1991.

[2] Leo Bachmair and Harald Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *Journal of the ACM*, 45(6):1007–1049, 1998.

[3] Jos C. M. Baeten, Jan A. Bergstra, Jan Willem Klop, and W. P. Weijland. Term-rewriting systems with rule priorities. *Theor. Comput. Sci.*, 67(2&3):283–301, 1989.

[4] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.

[5] Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In Franz Baader, editor, *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2, 2003, Proceedings*, volume 2741 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2003.

[6] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[7] Jean Goubault-Larrecq. Deciding $\mathcal{H}_1$ by resolution. *Information Processing Letters*, 95(3):401 – 408, 2005.

[8] Florent Jacquemard, Christoph Meyer, and Christoph Weidenbach. Unification in extensions of shallow equational theories. In Tobias Nipkow, editor, *Rewriting Techniques and Applications, 9th International Conference, RTA-98*, volume 1379 of *LNCS*, pages 76–90. Springer, 1998.

[9] Konstantin Korovin. iprover - an instantiation-based theorem prover for first-order logic (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008.

[10] Konstantin Korovin. Inst-Gen - A modular approach to instantiation-based automated reasoning. In *Programming Logics - Essays in Memory of Harald Ganzinger*, pages 239–270, 2013.

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

[12] Helmut Seidl and Andreas Reuß. Extending H1-Clauses with Disequalities. *Information Processing Letters*, 111(20):1007–1013, 2011.

[13] Helmut Seidl and Andreas Reuß. *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*, chapter Extending $\mathcal{H}_1$ -Clauses with Path Disequalities, pages 165–179. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[14] Helmut Seidl and Kumar Neeraj Verma. Cryptographic protocol verification using tractable classes of horn clauses. In *Program Analysis and Compilation, Theory and Practice*, pages 97–119. Springer, Juni 2007. Lecture Notes in Computer Science.

[15] John K. Slaney and Timothy Surendonk. Combining finite model generation with theorem proving: Problems and prospects. In Franz Baader and Klaus U. Schulz, editors, *Frontiers of Combining Systems, First International Workshop FroCoS 1996, Munich, Germany, March 26-29, 1996, Proceedings*, volume 3 of *Applied Logic Series*, pages 141–155. Kluwer Academic Publishers, 1996.

[16] Martin Suda, Christoph Weidenbach, and Patrick Wischnewski. On the saturation of yago. In *Automated Reasoning, 5th International Joint Conference, IJCAR 2010*, volume 6173 of *LNAI*, pages 441–456, Edinburgh, United Kingdom, 2010. Springer.

[17] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[18] Andreas Teucke and Christoph Weidenbach. First-order logic theorem proving and model building via approximation and instantiation. In Carsten Lutz and Silvio Ranise, editors, *Frontiers of Combining Systems: 10th International Symposium, FroCoS 2015, Wroclaw, Poland, September 21-24, 2015, Proceedings*, pages 85–100, Cham, 2015. Springer International Publishing.

[19] Andreas Teucke and Christoph Weidenbach. Ordered resolution with straight dismatching constraints. In Pascal Fontaine, Stephan Schulz, and Josef Urban, editors, *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.*, volume 1635 of *CEUR Workshop Proceedings*, pages 95–109. CEUR-WS.org, 2016.

[20] Andreas Teucke and Christoph Weidenbach. Decidability of the monadic shallow linear first-order fragment with straight dismatching constraints. `http://arxiv.org/abs/1703.02837`, 2017.

[21] Andrei Voronkov. AVATAR: the architecture for first-order theorem provers. In Armin Biere and Roderick Bloem, editors, *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*, volume 8559 of *Lecture Notes in Computer Science*, pages 696–710. Springer, 2014.

[22] Christoph Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In Harald Ganzinger, editor, *16th International Conference on Automated Deduction, CADE-16*, volume 1632 of *LNAI*, pages 314–328. Springer, 1999.

[23] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Renate A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.