Ordered Resolution with Straight Dismatching Constraints

Andreas Teucke, Christoph Weidenbach

Ordered Resolution with Straight Dismatching Constraints

Andreas Teucke
Max-Planck Institute for Informatics,
Campus E1.4 66123 Saarbrücken, Germany
Graduate School of Computer Science,
Saarbrücken, Germany

Christoph Weidenbach
Max-Planck Institute for Informatics,
Campus E1.4 66123 Saarbrücken, Germany

Abstract

We present a sound and complete ordered resolution calculus for first-order clauses with straight dismatching constraints. The extended clause language is motivated by our first-order theorem proving approach through approximation and refinement. Using a clause language with straight dismatching constraints, single refinement steps do not result in a worst-case quadratic blowup in the number of clauses anymore. The refinement steps can now be represented by replacing one input clause with two equivalent clauses. We show soundness and completeness of ordered resolution with straight dismatching constraints. All needed operations on straight dismatching constraints take linear or linear logarithmic time in the size of the constraint.

1 Introduction

Recently, we introduced a first-order calculus based on approximation and refinement [TW15]. There, a first-order clause set is approximated to a decidable fragment using an unsatisfiability preserving transformation. In case of a false positive, i.e., a resolution refutation of the approximation that does not imply unsatisfiability of the original clause set, a refinement via instantiation of the original clause set removes the false positive. The refinement involves instantiating an original clause C such that no ground instances of C are lost while two specific ground instances $C\sigma_1$ and $C\sigma_2$ get separated in the process. This requires $(|\Sigma| - 1) \cdot d + 1$ instantiations where Σ is the signature and d is the minimal term depth at which $C\sigma_1$ and $C\sigma_2$ differ from each other.

As an example, consider the first-order Horn clauses

$$\begin{aligned} Q(x) &\rightarrow P(g(x, f(x))) \\ &\rightarrow Q(f(g(a, a))) \\ &\rightarrow Q(f(g(b, b))) \\ P(g(f(g(a, a)), f(f(g(b, b))))) &\rightarrow \end{aligned}$$

under signature $\{a/0, b/0, f/1, g/2\}$ that are approximated into

In: P. Fontaine, S. Schulz, J. Urban (eds.): Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning (PAAR 2016), Coimbra, Portugal, 02-07-2016, published at http://ceur-ws.org

$$\begin{array}{rcl} S(y), Q(x) & \to & P(g(x,y)) \\ Q(z) & \to & S(f(z)) \\ & \to & Q(f(g(a,a))) \\ & \to & Q(f(g(b,b))) \\ P(g(f(g(a,a)), f(f(g(b,b)))) & \to & \end{array}$$

via linearization of $g(x, f(x))$ to $g(x, f(z))$ and then deep variable term extraction of $f(z)$ through the introduction of a fresh predicate S [TW15]. The approximated clause set has a refutation and the corresponding conflicting core, a minimal unsatisfiable set of instances from the above clauses generating this refutation, is

$$\begin{array}{rcl} S(f(f(g(b,b)))), Q(f(g(a,a))) & \to & P(g(f(g(a,a)), f(f(g(b,b))))) \\ Q(f(g(b,b))) & \to & S(f(f(g(b,b)))) \\ & \to & Q(f(g(a,a))) \\ & \to & Q(f(g(b,b))) \\ P(g(f(g(a,a)), f(f(g(b,b)))) & \to & \end{array}$$

Lifting the conflicting core to the original clause set fails, because the resolvent of the first two conflict clauses, eliminating the introduced S predicate

$$Q(f(g(b,b))), Q(f(g(a,a))) \quad \to \quad P(g(f(g(a,a)), f(f(g(b,b)))))$$

is not an instance of the original clause $Q(x) \to P(g(x, f(x)))$, modulo duplicate literal elimination. A refinement step replaces $Q(x) \to P(g(x, f(x)))$ by the instance $Q(f(g(a, y))) \to P(g(f(g(a, y)), f(f(g(a, y)))))$, and instances representing $Q(x) \to P(g(x, f(x)))$, where x is not instantiated with $f(g(a, y))$. The former clause contains the ground instance $Q(f(g(a, a))) \to P(g(f(g(a, a)), f(f(g(a, a)))))$ and the latter clauses include the ground instance $Q(f(g(b, b))) \to P(g(f(g(b, b)), f(f(g(b, b)))))$:

$$\begin{array}{rcl} Q(a) & \to & P(g(a, f(a))) \\ Q(b) & \to & P(g(b, f(b))) \\ Q(g(x,y)) & \to & P(g(g(x,y), f(g(x,y)))) \\ Q(f(a)) & \to & P(g(f(a), f(f(a)))) \\ Q(f(b)) & \to & P(g(f(b), f(f(b)))) \\ Q(f(f(x))) & \to & P(g(f(f(x)), f(f(f(x))))) \\ Q(f(g(b,y))) & \to & P(g(f(g(b,y)), f(f(g(b,y))))) \\ Q(f(g(f(x),y))) & \to & P(g(f(g(f(x),y)), f(f(g(f(x),y))))) \\ Q(f(g(g(x,y),z))) & \to & P(g(f(g(g(x,y),z)), f(f(g(g(x,y),z))))) \end{array}$$

Then, the approximation of the above nine clauses, via linearization and deep variable term extraction, excludes the previously found refutation. Actually, the refined approximated clause set yields a finite saturation and therefore shows satisfiability of the original clause set.

In this paper we introduce the new notion of straight dismatching constraints $x \neq t$ attached to clauses, and a corresponding ordered resolution calculus. A term t is straight if it is linear and all arguments of a nested straight term t are either variables or at most one straight term (Definition 1), i.e., the term contains at most one path along non-variable symbols. Straight dismatching constraints enable a refinement where a clause is replaced by exactly two new clauses. For the above example, they are

$$\begin{array}{rcl} Q(f(g(a,y))) & \to & P(g(f(g(a,y)), f(f(g(a,y))))) \\ Q(x) & \to & P(g(x, f(x))) \; ; \; x \neq f(g(a,v)) \end{array}$$

where the second clause represents all ground instances where x is not instantiated with an instance of $f(g(a, v))$, a straight term. This results in the refined approximation

$$\begin{array}{lrcll} (1) & S_1(y), S_2(x), Q(x) & \to & P(g(x,y))^* & \\ (2) & S_2(x), Q(x) & \to & S_1(f(x))^* & \\ (3) & S_3(x) & \to & S_2(f(x))^* & \\ (4) & & \to & S_3(g(a,y))^* & \\ (5) & S_4(y), Q(x) & \to & P(g(x,y))^* & ; \; x \neq f(g(a,v)) \\ (6) & Q(x) & \to & S_4(f(x))^* & ; \; x \neq f(g(a,v)) \\ (7) & & \to & Q(f(g(a,a)))^* & \\ (8) & & \to & Q(f(g(b,b)))^* & \\ (9) & P(g(f(g(a,a)), f(f(g(b,b)))))^+ & \to & & \end{array}$$

where under the lpo ordering $P \prec Q \prec S_4 \prec S_3 \prec S_2 \prec S_1 \prec a \prec b \prec f \prec g$ and selection of the first complex negative literal, the only inferences are

$$
\begin{array}{lrcl}
[10 : Res : 1,\ 9] & S_1(f(f(g(b,b))))^+, S_2(f(g(a,a))), Q(f(g(a,a))) & \rightarrow & \\
[11 : Res : 2, 10] & S_2(f(g(b,b)))^+, Q(f(g(b,b))), S_2(f(g(a,a))), Q(f(g(a,a))) & \rightarrow & \\
[12 : Res : 3, 11] & S_3(g(b,b))^+, Q(f(g(b,b))), S_2(f(g(a,a))), Q(f(g(a,a))) & \rightarrow &
\end{array}
$$

A resolution between the fifth and ninth clause has the tautologous result

$$S(f(f(g(b,b)))), Q(f(g(a,a))) \quad \rightarrow \quad ;\ f(g(a,a)) \neq f(g(a,v)).$$

Clauses with straight dismatching constraints are closed under resolution and factoring. Repeated refinement steps and resolution inferences can add further atomic constraints to a clause, which are interpreted as conjunctions, e.g., $(C; x \neq t \wedge y \neq s)$ represents instances of C where neither x is an instance of t nor y is an instance of s. Straight dismatching constraints can be efficiently encoded in amortized constant space once the input constraint terms are stored. Any straight term generated by the calculus is a subterm of an input straight term. Relevant operations (substitution, intersection, solvability and subset tests) take linear time in the size of the constraints, once they are ordered. Ordering can be done as usual in linear logarithmic time. The ordered resolution calculus with straight dismatching constraints is sound and complete, enables an abstract redundancy criterion and follows the saturation principle: if all non-redundant inferences from a clause set are performed and the empty clause is not generated then the resulting saturated clause set has a model computed by a model operator.

Ordered resolution on constraint clauses has already been investigated. Clauses with dismatching constraints are an instance of the constraint clauses introduced in [NR01]. The authors prove soundness and completeness for ordered superposition extended with general syntactic equality and ordering inequality constraints powerful enough to encode unification and maximality restrictions. Algorithms for solving these constraints are NP-hard. In our more specific setting, the operations on the constraints are linear or linear logarithmic. Furthermore, our completeness proof is an extension of the original superposition completeness proof without constraints [BG94], enabling the standard superposition redundancy criterion.

Our constraints are also a restricted version of the dismatching constraints from [AW15]. These constraints are more powerful than straight constraints as they also allow dismatching between variables in a clause. However, solving these constraints is again NP-hard and completeness for the NRCL calculus is only shown for a finite domain first-order fragment.

The same dismatching constraints are used in DInst-Gen [Kor13] to exclude redundant instances of clauses. Solving constraints can even be done in linear time because of an extended signature with an additional fresh constant $\perp$. All satisfiable atomic constraints share the solution where all variables are substituted with $\perp$. Solving constraints therefore reduces to the matching problem on the atomic constraints. Our constraint clause language, however, does not rely on an extended signature.

In [CZ92], clauses are constrained by equational systems [CL89]. An equational system is a quantifier free formula over (dis)equations that are syntactically interpreted over a Herbrand domain. The authors show soundness and completeness of an ordered resolution calculus for the extended clauses. They devise a separate calculus for model building. Testing solvability of equational systems is NP-hard. Straight dismatching constraints are a special case of equational systems. They enable more efficient algorithms (Section 4) that are not instances of the general algorithms for systems. Furthermore, our correctness result for ordered resolution (Section 3) includes a notion of redundancy and the concept of saturation, i.e., model building is the result of a saturated clause set.

Our approach to solving constraints is a special case of the solved form construction described in [CL89]. Normalizing straight constraints only requires the Clash (C_2), Decomposition (D_2) and Universality (U_1) rules. For straight terms the Decomposition rule simplifies to $f(t_1, \ldots, t_n) \neq f(y_1, \ldots, y_{i-1}, s_i, y_{i+1}, \ldots, y_n) \Rightarrow t_i \neq s_i$. The Explosion ($E$) rule is refined to instantiations with shallow instances (Definition 12).

An alternative to using constraints is the explicit representation described in [LM87]. Their algorithm allows computing a finite set of clauses covering the same ground clauses as a clause with constraint. For example, for $(Q(x) \rightarrow P(g(x, f(x))); x \neq f(g(a, v)))$ the explicit representation is exactly the nine clauses listed in the example. This always succeeds because by definition our constraint represents unrestricted counter-examples. While using the explicit representations is equivalent to using constraints, we require quadratically fewer clauses.

The paper is organized as follows. After the preliminaries, we formally define straight dismatching constraints in Section 2. Then, Section 3 contains the soundness and completeness proofs for our ordered resolution calculus with constraints. We conclude with discussing the complexity of operations on straight dismatching constraints.

2 Preliminaries

We consider a standard first-order language without equality. The 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. Capital letters S, P, Q, R denote predicates, A, B atoms, E, L literals, C, D clauses.

A first-order language is constructed over a signature $\Sigma = (\mathbb{F}, \mathbb{P})$, where $\mathbb{F}$ and $\mathbb{P}$ are non-empty, disjoint, in general infinite sets of function and predicate symbols, respectively. Every function or predicate symbol has a fixed arity and nullary functions are called constants. Further, we assume an infinite set X of variable symbols disjoint from Σ. Then, the set of all *terms* $\mathbb{T}(\mathbb{F}, X)$ is defined recursively by: (i) every constant symbol $c \in \mathbb{F}$ is a term, (ii) every variable $x \in X$ is a term and (iii) whenever $t_1, \ldots, t_n$ are terms and $f \in \mathbb{F}$ is a function symbol with arity n, then $f(t_1, \ldots, t_n)$ is a term. If $t_1, \ldots, t_n$ are terms and $P \in \mathbb{P}$ is a predicate symbol with arity n, then $P(t_1, \ldots, t_n)$ is an *atom* and the t_i are its *arguments*. A term is called *linear* if there are no duplicate variable occurrences. For brevity, we also write $\vec{t}$ instead of $t_1, \ldots, t_n$ for the argument of functions and predicates. A *literal* is either an atom or an atom preceded by $\neg$ and it is then called positive or negative, respectively.

The set of *free* variables of an atom (term) denoted by $\mathrm{vars}(\_)$ is defined as follows: $\mathrm{vars}(P(t_1, \ldots, t_n)) = \bigcup_i \mathrm{vars}(t_i)$, $\mathrm{vars}(f(t_1, \ldots, t_n)) = \bigcup_i \mathrm{vars}(t_i)$ and $\mathrm{vars}(x) = \{x\}$. The function naturally extends to literals, clauses and sets thereof.

A *multiset* over a set A is a function M from A to the natural numbers. For an element $e \in A$, $M(e)$ is the number of occurrences of e in M. We say e is an element of M if $M(e) > 0$. The union, intersection, and difference of multisets are defined by $M_1(x) \cup M_2(x) = M_1(x) + M_2(x)$, $M_1(x) \cap M_2(x) = \min(M_1(x), M_2(x))$, and $M_1(x) \setminus M_2(x) = \max(0, M_1(x) - M_2(x))$. We use set-like notation for multisets.

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 write $P(x)$ as an alternative of $\to P(x)$ whereas if Δ is empty $\to$ is always shown. We abbreviate disjoint set union with sequencing, for example, we write $\Gamma, \Gamma' \to \Delta, A$ instead of $\Gamma \cup \Gamma' \to \Delta \cup \{A\}$. Terms, literals and clauses are called *ground* if they contain no variables.

A *substitution* σ is a mapping from variables $\mathbb{V}$ into terms denoted by pairs $\{x \mapsto t\}$ where $x\sigma \neq x$ holds only for finitely many variables. The update $\sigma[x \mapsto t]$ denotes the substitution where x maps to t and y maps to $y\sigma$ for all $y \neq x$. The composition $\sigma\tau$ denotes the substitution $(x\sigma)\tau$ for all variables x. A substitution σ is a *grounding* substitution for $\mathbb{V}$ if $x\sigma$ is ground for every variable $x \in \mathbb{V}$.

Given two terms (atoms) s, t, a substitution σ is called a *unifier* for s and t if $s\sigma = t\sigma$. It is called a *most general unifier (mgu)* if for any other unifier τ of s and t there exists a substitution δ with $\sigma\delta = \tau$. s is called an instance of t if there exists a substitution σ such that $t\sigma = s$.

An *atom ordering* $\prec$ is a well-founded, total ordering on ground atoms. For the atom ordering $\prec$, $A \preceq B$ is defined by $A \prec B$ or $A = B$. An atom A is *maximal* with respect to a multiset of atoms Δ, if for all $B \in \Delta$ we have $A \not\prec B$. Any atom ordering $\prec$ is extended to literals by $A \prec \neg A \prec B$ if $A \prec B$. The multiset extension $\prec_{mul}$ of an ordering $\prec$ is defined by $M_1 \prec_{mul} M_2$ iff $M_1 \neq M_2$ and for all $m \in M_1$ with $M_1(m) > M_2(m)$ there is an $m' \in M_2$ with $m \prec m'$. 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 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 literal E (A or $\neg A$) is *(strictly) maximal* in a clause $\Gamma \to \Delta, A$ ($\Gamma, A \to \Delta$) if there is no literal in $\Gamma \to \Delta$ that is greater (or equal) than E with respect to $\prec$.

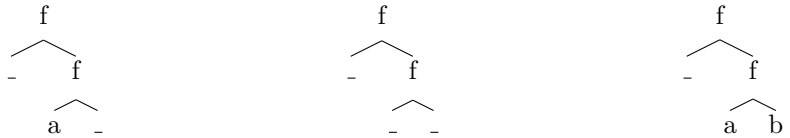
A *Herbrand interpretation* I is a - possibly infinite - set of ground atoms. A ground atom A is called *true* in I if $A \in \mathrm{I}$ and *false*, otherwise. I is said to *satisfy* a ground clause $C = \Gamma \to \Delta$, denoted by $\mathrm{I} \models C$, if $\Delta \cap \mathrm{I} \neq \emptyset$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

2.1 Straight Dismatching Constraints

In the following, we extend clauses with dismatching constraints $x \neq t$ which restrict the allowed ground instances. An important restriction is that the term t is straight, i.e., it only has a single branch.

Definition 1 (Straight Terms)**.** A term $f(s_1, \ldots, s_n)$ is called straight, if $f(s_1, \ldots, s_n)$ is linear and all arguments are variables except for at most one straight argument s_i.

For example, the terms $f(x, f(a, y))$ and $f(x, f(y, z))$ are straight, while $f(x, f(a, b))$ is not. We can identify straight terms visually in their tree form, if there is exactly one branch with function symbols on each node.



Note that for two different ground terms t_1 and t_2, there always exists a position in both where they have different function symbols. The path from the root to this position in t_1 defines a straight term t that has only t_1 as an instance but not t_2. Thus, a straight terms are sufficient to isolate two ground instances.

Definition 2 (Straight Dismatching Constraint)**.** A straight dismatching constraint π is of the form

$$\bigwedge_{i \in \mathbb{I}} \; s_i \neq t_i$$

where $\mathbb{I}$ is a finite set of indices, the s_i are arbitrary terms and the t_i are straight terms.

Given a substitution σ, $\pi\sigma = \bigwedge_{i \in \mathbb{I}} \; s_i\sigma \neq t_i$. $\mathrm{lvars}(\pi) = \bigcup_{i \in \mathbb{I}} \mathrm{vars}(s_i)$ are the left hand variables in constraint π. We further extend the set of constraints with the constants $\top$ and $\bot$ representing the empty and the unsolvable constraint, respectively. An atomic constraint $s \neq t$ occurring in π is called a subconstraint of π. The length $|\pi|$ is defined as the number of subconstraints, i.e., $|\mathbb{I}|$. The size, $\mathrm{size}(\pi)$, and depth, $\mathrm{d}(\pi)$, of a constraint $\pi = \bigwedge_{i \in \mathbb{I}} \; s_i \neq t_i$ are the sum and maximum of the term depths of the t_i, respectively. For brevity, we will call straight dismatching constraints just constraints in the following.

Definition 3 (Solutions)**.** Let π be a constraint, Σ a signature and $\mathbb{V}$ a set of variables with $\mathrm{lvars}(\pi) \subseteq \mathbb{V}$. A solution of π over $\mathbb{V}$ is a grounding substitution δ over $\mathbb{V}$ such that for all $i \in \mathbb{I}$, $s_i\delta$ is not an instance of t_i. $\mathbb{D}^{\mathbb{V}}(\pi) := \{\delta \mid \delta \text{ is a solution of } \pi \text{ over } \mathbb{V}\}$. A constraint is solvable if it has a solution and unsolvable, otherwise. Two constraints are equivalent if they have the same solutions over all $\mathbb{V}$. In particular, all grounding substitutions are solutions of $\top$, and $\bot$ has no solution.

For example, consider the constraint $\pi = x \neq b \wedge x \neq f(f(u)) \wedge x \neq g(v, w)$ with the signature $\mathbb{F} = \{a/0, b/0, f/1, g/2\}$. π is solvable and $\mathbb{D}^{\mathbb{V}}(\pi) = \{\{x \mapsto a\}, \{x \mapsto f(a)\}, \{x \mapsto f(b)\}, \{x \mapsto f(g(s, t))\} \mid s, t \in \mathbb{T}(\mathbb{F}, \emptyset)\}$ is the set of solutions of π over $\{x\}$.

Note that atomic constraints are not symmetric. For example $x \neq a$ can have a solution while $a \neq x$ is unsolvable. Furthermore, the right-hand variables in a constraint can be arbitrarily renamed while the left-hand variables are fixed. For example, the constraints $x \neq f(v)$ and $x \neq f(w)$ are equivalent but $x \neq f(v)$ and $y \neq f(v)$ are not. For an atom $P(g(x, y))$ and a constraint $x \neq f(y)$, the two y variables are not connected since for $\sigma = \{y \mapsto t\}$, $P(g(x, y))\sigma = P(g(x, t))$ but $(x \neq f(y))\sigma = (x \neq f(y))$. To avoid confusion, we generally rename right-hand variables to be unique in a given context.

If δ is a solution of $\pi \wedge \pi'$, then δ is a solution of π and if δ is a solution of $\pi\sigma$, then $\sigma\delta$ is a solution of π. These two properties follow directly from the definition. We will ignore $\mathbb{V}$ if it is clear in a given context. For example, if π restricts the clause C, $\mathbb{V}$ is always $\mathrm{vars}(C) \cup \mathrm{lvars}(\pi)$.

Definition 4 (Constraint Normal Form)**.** A constraint $\pi = \bigwedge_{i \in \mathbb{I}} \; s_i \neq t_i$ is called normal, if all s_i are variables and for all $s_i = s_j$ with $i \neq j$, t_i is not an instance of t_j.

Definition 5 (Constraint Normalization)**.** We define constraint normalization $\pi\!\downarrow$ as the normal form of the following transition system over constraints.

1	$\pi \wedge s \neq y$	$\Rightarrow$	$\bot$	
2	$\pi \wedge f(s_1, \ldots, s_n) \neq f(y_1, \ldots, y_n)$	$\Rightarrow$	$\bot$	
3	$\pi \wedge f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_n)$	$\Rightarrow$	$\pi \wedge s_i \neq t_i$	if t_i is complex
4	$\pi \wedge f(s_1, \ldots, s_n) \neq g(t_1, \ldots, t_m)$	$\Rightarrow$	π	if $f \neq g$
5	$\pi \wedge x \neq t \wedge x \neq t\sigma$	$\Rightarrow$	$\pi \wedge x \neq t$	

Lemma 6. Constraint normalization terminates.

Proof. The first and second rule trivially terminate the normalization. In the third rule, the size of the constraint decreases. The last two rules reduce the length of the constraint. □

Lemma 7. $\pi\downarrow$ is a normal constraint and equivalent to π.

Proof. In the first and second rule, $f(s_1, \ldots, s_n)$ is an instance of y and $f(y_1, \ldots, y_n)$, respectively. Hence, there are no solutions, which is equivalent to the unsatisfiable constraint $\bot$. In the third rule, a grounding substitution δ is a solution of $f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_n)$, if and only if at least one $s_j\delta$ is not an instance of t_j, i.e. $f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_n)$ is equivalent to $s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n$. However, since $f(t_1, \ldots, t_n)$ is straight and the second rule does not apply, there is exactly one complex t_i. Thus, each $s_j\delta$ is an instance of the respective variable t_j for $j \neq i$ and therefore $s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n$ simplifies to just $s_i \neq t_i$. In the forth rule, $f(s_1, \ldots, s_n)$ can never be an instance of $g(t_1, \ldots, t_m)$ for any grounding substitution. Hence, all solutions of π are solutions of $\pi \wedge f(s_1, \ldots, s_n) \neq g(t_1, \ldots, t_m)$. In the last rule, let δ be a solution of $\pi \wedge x \neq t$. Then, $x\delta$ is not an instance of t and hence, not an instance of $t\sigma$ either. Therefore, δ is a solution of $\pi \wedge x \neq t \wedge x \neq t\sigma$. All rules together cover every case where the left-hand side of a subconstraint is not a variable or there are two constraints $x \neq t$ and $x \neq s$ where s is an instance of t. Hence, $\pi\downarrow$ is a normal constraint. □

Definition 8 (Constrained Clause)**.** A pair of a clause and a constraint $(C; \pi)$ is called a constrained clause. A constrained clause is normal, if π is normal and $\mathrm{lvars}(\pi) \subseteq \mathrm{vars}(C)$. For a given signature Σ, the set $\mathbb{G}((C;\pi)) = \{C\delta \mid \delta \in \mathbb{D}^{\mathbb{V}}(\pi), \mathbb{V} = \mathrm{vars}(C) \cup \mathrm{lvars}(\pi)\}$ is called the set of ground instances of $(C;\pi)$. The notion extends to sets of constrained clauses. Two constrained clauses are equivalent if they have the same ground instances. A Herbrand interpretation I satisfies $(C;\pi)$, if $\mathrm{I} \vDash \mathbb{G}((C;\pi))$.

Note that in the context of a constrained clause $(C;\pi)$, a solution δ of π is implicitly over $\mathbb{V} = \mathrm{vars}(C) \cup \mathrm{lvars}(\pi)$ such that $C\delta \in \mathbb{G}((C;\pi))$. We can now also formally define the refinement described in the introduction.

Lemma 9 (Refinement)**.** Let $N \cup \{(C;\pi)\}$ be a constrained clause set, $x \in \mathrm{vars}(C)$ and t a straight term with $\mathrm{vars}(t) \cap \mathrm{vars}(C) = \emptyset$. Then $N \cup \{(C;\pi)\}$ and $N \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\}$ are equisatisfiable.

Proof. Let $C\delta \in \mathbb{G}((C;\pi))$. If $x\delta$ is not an instance of t, then δ 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 δ'. Then, δ' 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 I is a model of $N \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\}$, then I is also a model of $N \cup \{(C;\pi)\}$.

Let $D \in \mathbb{G}((C;\pi \wedge x \neq t)) \cup \mathbb{G}((C;\pi)\{x \mapsto t\})$. If $D = C\delta \in \mathbb{G}((C;\pi \wedge x \neq t))$, then δ is also a solution of π and therefore $D \in \mathbb{G}((C;\pi))$. If $D = C\{x \mapsto t\}\delta \in \mathbb{G}((C;\pi)\{x \mapsto t\})$, then $\{x \mapsto t\}\delta$ is a solution of π and therefore $D \in \mathbb{G}((C;\pi))$. Hence, $\mathbb{G}((C;\pi \wedge x \neq t)) \cup \mathbb{G}((C;\pi)\{x \mapsto t\}) \subseteq \mathbb{G}((C;\pi))$. Therefore, if I is a model of $N \cup \{(C;\pi)\}$, then I is also a model of $N \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\}$. □

Consider again the introductory example. Lifting the conflicting core of the approximation failed because the so-called lift-conflict $D = Q(f(f(b))), Q(f(f(a))) \to P(g(f(f(a)), f(f(b))))$ is not an instance of the original clause $Q(x) \to P(g(x, f(x)))$. Specifically, because x cannot be instantiated with an instance of both $f(f(a))$ and $f(f(b))$. Therefore, we refine the original clause with the instantiation $\{x \mapsto f(f(a))\}$ and the opposing constraint $x \neq f(f(a))$, resulting in $C_1 = Q(f(f(a))) \to P(g(f(f(a)), f(f(f(a)))))$ and $C_2 = (Q(x) \to P(g(x, f(x)))\ ;\ x \neq f(f(a)))$. D differs from C_1 because both occurrences of x are already instantiated with $f(f(a))$, excluding any approximations, where the second x is instantiated with $f(f(b))$. Since $x \neq f(f(a))$ has no solutions where x is instantiated with $f(f(a))$ and our approximation preserves this fact, D can not be inferred from the approximation of C_2. Therefore, the lift-conflict D can not happen again.

Lemma 10 (Clause Simplification)**.** A constrained clause $(C;\pi \wedge \bigwedge_{i\in\mathbb{I}} x \neq t_i)$ is equivalent to $(C;\pi)$, if π is normal, $x \notin \mathrm{vars}(C) \cup \mathrm{lvars}(\pi)$ and $\bigwedge_{i\in\mathbb{I}} x \neq t_i$ is solvable.

Proof. Let $C\delta \in \mathbb{G}(C;\pi)$, where δ is a solution of π over $\mathbb{V}$. Because $\bigwedge_{i\in\mathbb{I}} x \neq t_i$ is solvable, there is a solution $\{x \mapsto t\}$ over $\{x\}$. Then, $\delta[x \mapsto t]$ is a solution of $\bigwedge_{i\in\mathbb{I}} x \neq t_i \wedge \pi$ over $\mathbb{V} \cup \{x\}$. Hence, $C\delta[x \mapsto t] \in \mathbb{G}(C; \bigwedge_{i\in\mathbb{I}} x \neq t_i \wedge \pi)$. The reverse direction is trivial. □

Lemmas 7 and 10 show that for any constrained clause there exists an equivalent normal clause. In the following we assume that constrained clauses are normal.

Lastly in this section, we show that solvability of a constraint is decidable. Note that not all unsolvable constraints trivially normalize to $\bot$. A normal constraint can actually be both solvable and unsolvable depending on the signature. For example, $x \neq a$ is a normal constraint that is solvable for signature $\{a/0, b/0\}$ but unsolvable for signature $\{a/0\}$. A discussion of the implementation is deferred to Section 4.

Proposition 11. A constraint π has no solution if and only if $\pi{\downarrow}{=} \bot$ or there exists a subset $\pi_x = \bigwedge_{i\in\mathbb{I}} x \neq s_i$ of $\pi{\downarrow}$ that has no solution.

Definition 12 (Shallow Instances)**.** Let $\{y_1, y_2, \ldots\}$ be an infinite set of pairwise distinct variables. We define the set of shallow instantiations of x under signature Σ as the set of substitutions

$$\Theta_x^\Sigma = \{\{x \mapsto f(y_1, \ldots, y_n)\} \mid f \in \mathbb{F}\}.$$

Note that if the set of function symbols $\mathbb{F}$ is infinite, Θ_x^Σ is also infinite, but then any constraint is trivially solvable if its normal form is different from $\bot$. In the context of oredered resolution, the function symbols appearing in the input clause set constitute a finite signature.

Lemma 13. A constraint $\pi_x = \bigwedge_{i\in\mathbb{I}} x \neq s_i$ has no solution under signature Σ if and only if $\pi_x\sigma$ has no solution for every $\sigma \in \Theta_x^\Sigma$.

Proof. "$\Rightarrow$". Assume δ is a solution of $\pi_x\sigma$ for some shallow instantiation σ. Then, $\sigma\delta$ is a solution of π_x, which contradicts the assumption.

"$\Leftarrow$". Assume δ is a solution of π_x. Then, $x\delta$ is ground and hence, $x\delta = f(y_1, \ldots, y_n)\sigma$ for some function $f \in \mathbb{F}$ and substitution σ. Thus, $\pi_x\{x \mapsto f(y_1, \ldots, y_n)\} = \bigwedge_{i\in\mathbb{I}} f(y_1, \ldots, y_n) \neq s_i$ has σ as a solution, which contradicts the assumption. □

Lemma 14. Solvability of a constraint π is decidable.

Proof. If $\mathbb{F}$ is not finite, a constraint π has no solution if and only if $\pi{\downarrow}{=} \bot$. Let $\pi{\downarrow}{=} \pi_{x_1} \wedge \ldots \wedge \pi_{x_n}$ with $x_i \neq x_j$ for $i \neq j$. Because $|\pi_{x_i}|$ is finite, there exists a function symbol $f \in \mathbb{F}$ such that $x_i \neq f(\vec{t}) \notin \pi_{x_i}$ for all straight terms $f(\vec{t})$. Thus, for an arbitrary ground term $f(\vec{s})$, $\sigma_i = \{x_i \mapsto f(\vec{s})\}$ is a solution of π_{x_i}. Then, $\sigma_1 \cup \ldots \cup \sigma_n$ is a solution of π.

If $\mathbb{F}$ is finite, we use Proposition 11 and Lemma 13 recursively to check solvability of π. Using Proposition 11 we pick the constraints of one variable. Then the constraints are instantiated according to Lemma 13. The instantiated constraints are normalized and then the overall procedure is repeated until either $\bot$ or $\top$ is derived. The procedure terminates because each instantiation strictly decreases the term depths of the respective constraint and the signature is finite. □

3 Ordered Resolution with Straight Dismatching Constraints

In this section we define an ordered resolution calculus with selection and redundancy on constrained clauses. In [NR01] the authors show that superposition without equality, i.e., ordered resolution, is compatible with arbitrary constraint systems. So their completeness results provide completeness for a straightforward combination of straight dismatching constraints with ordered resolution. However, our calculus is by far more refined. The constraints are already considered in order to determine maximal literals, they are an inherent part of our notion of redundancy and normalization and inheritance is built into the inference rules. Therefore, we provide here a separate proof of completeness that follows the classical superposition completeness recipe without constraints [BG94].

Definition 15 (Orderings with Constraints)**.** Let $\prec$ be an ordering on ground atoms. A literal A is called *(strictly) maximal* in a 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 16 (Selection)**.** A *selection function* assigns to a clause $\Gamma \to \Delta$ a possibly empty subset of Γ. For a clause C and selection function sel, the literals in $\mathrm{sel}(C)$ are called selected.

Definition 17 (Redundancy)**.** For a given ordering on ground clauses $\prec$, a constrained clause $(C; \pi)$ is redundant w.r.t. N if for every $D \in \mathbb{G}((C; \pi))$, there exist $D_1, \ldots, D_n \in \mathbb{G}(N)^{\prec D}$ with $D_1, \ldots, D_n \vDash D$.

Note that a constrained clause $(C;\pi)$ with a tautology C or an unsolvable constraint π is redundant. As usual, our notion of redundancy does not cover cases such as $A(x)$ subsuming $A(f(x))$. It can be extended to consider such matches by extending the ordering [BG94] or the notion of fairness [NR01].

Definition 18 (Constrained Subsumption)**.** A clause $(C;\pi)$ subsumes $(D;\pi')$ if there is a substitution σ such that $C\sigma \subsetneq D$ and $\mathbb{D}^{\mathbb{V}}(\pi') \subseteq \mathbb{D}^{\mathbb{V}}(\pi\sigma)$.

Lemma 19. If $(C;\pi)$ subsumes $(D;\pi')$, $(D;\pi')$ is redundant in $N \cup \{(C;\pi)\}$.

Proof. Let $D\delta \in \mathbb{G}((D;\pi'))$. Then, δ is a solution of $\pi\sigma$ and hence, $C\sigma\delta \in \mathbb{G}((C;\pi))$. Since $C\sigma\delta \subset D\delta$, $C\sigma\delta \in \mathbb{G}((C;\pi))^{\prec D\delta}$ and $C\sigma\delta \vDash D\delta$. Therefore, $(D;\pi')$ is redundant in $N \cup \{(C;\pi)\}$. □

Definition 20 (Constrained Condensation)**.** A constrained clause $(C;\pi)$ can be condensed if there exists a substitution σ such that $(C\sigma;\pi\sigma)$ with duplicate literals removed subsumes $(C;\pi)$. $(C\sigma;\pi\sigma)$ is then called the condensation of $(C;\pi)$.

Definition 21 (Constrained Subsumption Resolution)**.** Let $(C \vee A;\ \pi)$ and $(D \vee \neg A';\ \pi')$ be two constrained clauses. Let σ be the matcher $A\sigma = A'$ and let $(C;\ \pi)\sigma$ subsume $(D;\ \pi')$, then $(D;\ \pi')$ is called a *subsumption resolvent* of the clauses $(C \vee A;\ \pi)$ and $(D \vee \neg A';\ \pi')$.

In the presence of $(D;\ \pi')$ the clause $(D \vee \neg A';\ \pi')$ is redundant. Therefore, in practice, subsumption resolution directly replaces $(D \vee \neg A';\ \pi')$ with $(D;\ \pi')$.

Definition 22 (Constrained 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 $(\Gamma_1 \to \Delta_1, A; \pi_1 \wedge \pi_2)\sigma$ and $\mathrm{sel}(\Gamma_1 \to \Delta_1, A) = \emptyset$;
4. $B \in \mathrm{sel}(\Gamma_2, B \to \Delta_2)$ or $\mathrm{sel}(\Gamma_2, B \to \Delta_2) = \emptyset$ and $\neg B\sigma$ maximal in $(\Gamma_2, B \to \Delta_2; \pi_1 \wedge \pi_2)\sigma$.

Definition 23 (Constrained 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. $\pi\sigma\downarrow$ is solvable;
3. $\mathrm{sel}(\Gamma \to \Delta, A, B) = \emptyset$;
4. $A\sigma$ is maximal in $(\Gamma \to \Delta, A, B; \pi)\sigma$

Lemma 24 (Soundness)**.** Constrained Resolution and Factoring are sound.

Proof. Let $(\Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\sigma\delta$ be a ground instance of $((\Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\sigma; (\pi_1 \wedge \pi_2)\sigma)$. Then, δ is a solution of $(\pi_1 \wedge \pi_2)\sigma$ and $\sigma\delta$ is a solution of π_1 and π_2. Hence, $(\Gamma_1 \to \Delta_1, A)\sigma\delta$ and $(\Gamma_2, B \to \Delta_2)\sigma\delta$ are ground instances of $(\Gamma_1 \to \Delta_1, A; \pi_1)$ and $(\Gamma_2, B \to \Delta_2; \pi_2)$, respectively. Because $A\sigma\delta = B\sigma\delta$, if $(\Gamma_1 \to \Delta_1, A)\sigma\delta$ and $(\Gamma_2, B \to \Delta_2)\sigma\delta$ are satisfied, then $(\Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\sigma\delta$ is also satisfied. Therefore, Constrained Resolution is sound.

Let $(\Gamma \to \Delta, A)\sigma\delta$ be a ground instance of $((\Gamma \to \Delta, A)\sigma; \pi\sigma)$. Then, δ is a solution of $\pi\sigma$ and $\sigma\delta$ is a solution of π. Hence, $(\Gamma \to \Delta, A, B)\sigma\delta$ is a ground instance of $(\Gamma \to \Delta, A, B; \pi)$. Because $A\sigma\delta = B\sigma\delta$, if $(\Gamma \to \Delta, A, B)\sigma\delta$ is satisfied, then $(\Gamma \to \Delta, A)\sigma\delta$ is satisfied. Thus, Constrained Factoring is sound. □

Definition 25 (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 $\mathbb{G}((R;\pi)) \subseteq \mathbb{G}(N)$.

Definition 26 (Partial Minimal Model Construction)**.** Given a constrained clause set N, an ordering $\prec$ and a selection function sel, we construct an interpretation I_N for N, called a partial model, inductively on the ground instances of clauses from N as follows:

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

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

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

A clause C with $\delta_C \neq \emptyset$ is called productive. The smallest ground instance $C \in \mathbb{G}(N)$, where $\mathrm{I}_N \not\models C$, is called the minimal false clause. If no minimal false clause exists, then I_N is a model of N.

Recall that a ground instance of a constraint clause is a standard clause, Definition 8.

Lemma 27 (Completeness)**.** Let N be a constrained clause set saturated up to redundancy by ordered resolution with selection. Then N is unsatisfiable, if and only if $(\Box; \top) \in N$.

Proof. Assume N is unsatisfiable, but $(\Box; \top) \notin N$. For the partial model I_N, there exists a minimal false clause $C\sigma \in \mathbb{G}((C; \pi))$ for some $(C; \pi) \in N$.

$C\sigma$ is not productive, because otherwise $\mathrm{I}_N \models C\sigma$. Hence, either $\mathrm{sel}(C) \neq \emptyset$ or no positive literal in $C\sigma$ is strictly maximal. Assume $C = \Gamma_2, B \to \Delta_2$ with $B \in \mathrm{sel}(C)$ or $\neg B\sigma$ maximal. Then, $B\sigma \in \mathrm{I}_{C\sigma}$ and there exists a ground instance $(\Gamma_1 \to \Delta_1, A)\tau = D\tau \prec C\sigma$ of some clause $(D; \pi') \in N$, which produces $A\tau = B\sigma$. Therefore, there exists a $\rho = \mathrm{mgu}(A, B)$ and ground substitution δ such that $C\sigma = C\rho\delta$, $D\tau = D\rho\delta$. Since $\rho\delta = \sigma$ is a solution of π and π', δ is a solution of $(\pi \wedge \pi')\rho$. Under these conditions, constrained resolution can be applied to $(\Gamma_1 \to \Delta_1, A; \pi')$ and $(\Gamma_2, B \to \Delta_2; \pi)$. Their resolvent $(R; \pi_R) = ((\Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\rho; (\pi \wedge \pi')\rho)$ is either redundant in N or $\mathbb{G}((R; \pi_R)) \subseteq \mathbb{G}(N)$. Its ground instance $R\delta$ is false in I_N and $R\delta \prec C\sigma$. If $(R; \pi_R)$ is redundant in N, there exist $C_1, \ldots, C_n$ in $\mathbb{G}(N)^{\prec R\delta}$ with $C_1, \ldots, C_n \models R\delta$. Because $C_i \prec R\delta \prec C\sigma$, $\mathrm{I}_N \models C_i$ and hence $\mathrm{I}_N \models R\delta$, which contradicts $\mathrm{I}_N \not\models R\delta$. Otherwise, if $\mathbb{G}((R; \pi_R)) \subseteq \mathbb{G}(N)$, then $R\delta \in \mathbb{G}(N)$, which contradicts $C\sigma$ being minimal false.

Now, assume $\mathrm{sel}(C) = \emptyset$ and $C = \Gamma \to \Delta, B$ with $B\sigma$ maximal. Then, $C = \Gamma \to \Delta', A, B$ with $A\sigma = B\sigma$. Therefore, there exists a $\rho = \mathrm{mgu}(A, B)$ and ground substitution δ such that $C\sigma = C\rho\delta$ and $\rho\delta$ is a solution of π. Hence, δ is a solution of $\pi\rho$. Under these conditions, constrained factoring can be applied to $(\Gamma \to \Delta', A, B; \pi)$. The result $(R; \pi_R) = ((\Gamma \to \Delta', A)\rho; \pi\rho)$ is either redundant in N or $\mathbb{G}((R; \pi_R)) \subseteq \mathbb{G}(N)$. Its ground instance $R\delta$ is false in I_N and $R\delta \prec C\sigma$. If $(R; \pi_R)$ is redundant in N, there exist $C_1, \ldots, C_n$ in $\mathbb{G}(N)^{\prec R\delta}$ with $C_1, \ldots, C_n \models R\delta$. Because $C_i \prec R\delta \prec C\sigma$, $\mathrm{I}_N \models C_i$ and hence $\mathrm{I}_N \models R\delta$, which contradicts $\mathrm{I}_N \not\models R\delta$. Otherwise, if $\mathbb{G}((R; \pi_R)) \subseteq \mathbb{G}(N)$, then $R\delta \in \mathbb{G}(N)$, which contradicts $C\sigma$ being minimal false.

Therefore, if $(\Box; \top) \notin N$, no minimal false clause exists and N is satisfiable. □

As an example for an application of the calculus we consider the following puzzle [Pel86]: "Someone who lives in Dreadsbury Mansion killed Aunt Agatha. Agatha (a), the butler (b), and Charles (c) live in Dreadsbury Mansion, and are the only people who live therein."

$$[1 : Inp] \qquad (\to K(a,a),\ K(b,a),\ K(c,a)\ ;\ \top)$$

"A killer always hates his victim, and is never richer than his victim."

$$[2 : Inp] \qquad (K(x,y) \to H(x,y)\ ;\ \top)$$

$$[3 : Inp] \qquad (K(x,y),\ R(x,y) \to\ ;\ \top)$$

"Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler." The compact representation of this exception results in a non-trivial straight dismatching constraint.

$$[4 : Inp] \qquad (H(a,x),\ H(c,x) \to\ ;\ \top)$$

$$[5 : Inp] \qquad (\to H(a,x)\ ;\ x \neq b)$$

"The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Agatha hates."

$$[6 : Inp] \qquad (\rightarrow R(x,a),\ H(b,x)\ ;\ \top)$$
$$[7 : Inp] \qquad (H(a,x) \rightarrow H(b,x)\ ;\ \top)$$

"No one hates everyone. Agatha is not the butler. Therefore Agatha killed herself."

$$[8 : Inp] \qquad (H(x,a),\ H(x,b),\ H(x,c) \rightarrow\ ;\ \top)$$
$$[9 : Inp] \qquad (K(a,a) \rightarrow\ ;\ \top)$$
$$[10 : SubRes : 1,9] \qquad (\rightarrow K(b,a),\ K(c,a)\ ;\ \top)$$
$$[11 : Res : 5,4] \qquad (H(c,x) \rightarrow\ ;\ x \neq b)$$
$$[12 : Res : 5,7] \qquad (\rightarrow H(b,x)\ ;\ x \neq b)$$
$$[13 : Res : 2,11] \qquad (K(c,x) \rightarrow\ ;\ x \neq b)$$
$$[14 : SubRes : 10,13] \qquad (\rightarrow K(b,a)\ ;\ \top)$$
$$[15 : Res : 14,3] \qquad (R(b,a) \rightarrow\ ;\ \top)$$
$$[16 : Res : 6,15] \qquad (\rightarrow H(b,b)\ ;\ \top)$$
$$[17 : Res : 16,8] \qquad (H(b,a),\ H(b,c) \rightarrow\ ;\ \top)$$
$$[18 : SubRes : 17,12] \qquad (H(b,c) \rightarrow\ ;\ \top)$$
$$[19 : SubRes : 18,12] \qquad (\square\ ;\ \top)$$

4 Operations on Constraints

In this section we consider implementation aspects of constraints including the representation of constraints, the solvability test and operations on constraints.

Straight Term Encoding

Looking again at constrained resolution, factoring, and normalization, note that the constraint-terms of the inference are all subterms of the constraints of the parent clauses. This means that throughout a saturation, every straight constraint term is a subterm of the constraints in the input set. Therefore, in an implementation we can store the input constraints separately and represent atomic constraints as just a variable and a pointer to the respective subterm.

Furthermore, by adding for each n-ary function symbol f the unary functions $f_1, \ldots, f_n$ and a constant symbol f_0, we can encode and store any straight term as an array of function symbols terminated by a constant symbol. For example, $f(x, f(a, y))$ and $f(x, f(y, z))$ become f_2, f_1, a_0 and f_2, f_0, respectively.

Sorting

All of the operations on constraints can be applied on arbitrary constraints, but most of them become significantly faster if the constraints are sorted. Given total orderings on X and $\mathbb{F}$, we sort normal atomic constraints ascending the following order:

$$\begin{aligned}
&x \neq [l] &&< y \neq [r] && \text{if} \quad x <_X y\\
&x \neq f_i, [l] &&< x \neq g_j, [r] && \text{if} \quad f <_{\mathbb{F}} g\\
&x \neq f_i, [l] &&< x \neq f_j, [r] && \text{if} \quad i <_{\mathbb{N}} j\\
&x \neq f_i, [l] &&< x \neq f_i, [r] && \text{if} \quad 0 <_{\mathbb{N}} i \text{ and } x \neq [l] < x \neq [r]
\end{aligned}$$

The first operation that becomes faster after sorting is the fifth normalization rule (Definition 5). In a sorted constraint, $x \neq t$ will be immediately followed by any $x \neq t\sigma_1, \ldots, x \neq t\sigma_n$, because any straight instance of a straight term $f_i, \ldots, g_0$ has the form $f_i, \ldots, g_j, \ldots, h_0$.

While initially sorting a constraint takes linear-logarithmic time, sorting new constraints created from already sorted constraints is generally faster. Adding a fresh constraint and intersection require only a linear insert and merge operation, respectively.

Substitution and normalization, $\pi\{x \mapsto t\}\downarrow$, preserves the sorting if t is linear and variable disjoint from the left-hand variables of π. Otherwise, we can first bucket sort the constraint depending on the left-hand variables without changing the ordering of the subconstraints with the same variables. Then, we merge the individual partially sorted intervals of subconstraints. For example, $\{x \neq f(t_1, z) \wedge \ldots \wedge x \neq f(t_n, z) \wedge x \neq f(z, s_1) \wedge \ldots \wedge x \neq f(z, s_m)\}\{x \mapsto f(y, y)\}$ normalizes to $\{y \neq t_1 \wedge \ldots \wedge y \neq t_n \wedge y \neq s_1 \wedge \ldots \wedge y \neq s_m\}$, where $y \neq t_1$ to $y \neq t_n$ and $y \neq s_1$ to $y \neq s_m$ are still sorted.

Solvability Check

Since clauses with an unsolvable constraint are tautologies, checking constraint solvability is an important tool to avoid unnecessary inferences. Compared to the at least NP-complete solvability tests for general constraints [Com91], solvability of straight dismatching constraints can be tested in linear time.

Considering again Lemma 14, note that for any shallow instantiation $\sigma \in \Theta_x^\Sigma$ an atomic constraint $(x \neq t)\sigma$ normalizes to $\top$ except for exactly the one σ where both $x\sigma$ and t have the same top function symbol. In that case, normalization reduces the size of t by removing the top symbol of t. This means that in total every non-variable position in the straight right-hand terms of the constraint is considered exactly once for each $\sigma \in \Theta_x^\Sigma$. Solvability can therefore be decided in $O(|\mathbb{F}|\text{size}(\pi))$.

If the subconstraints are sorted, the solvability check can be computed in-place, because each recursive call on some π_x and $\pi\sigma\downarrow$ applies to non-overlapping and consecutive sections in the sorted constraint. Recall that right hand sides of constraints do not share any variables. Now, looking at the overall recursive derivation tree generated along the proof of Lemma 14, the follwing invariant holds: Each subterm of the right hand sides of the inital constraint occurs at most once as a top level term on a right hand side constraint in the overall derivation tree. Solvability of a sorted constraint π can therefore be decided independently of the size of $\mathbb{F}$ in $O(\text{size}(\pi))$.

Furthermore, we can use intermediate results of the solvability check to simplify the constraint. If $\pi_x\{x \mapsto t\}$ has no solutions for some straight term t but $\pi_x\{x \mapsto t\}\downarrow \neq \bot$, we can add $x \neq t$ to π (Corollary 29) which replaces the sub-constraints in $\pi_x\{x \mapsto t\}\downarrow$ by Definition 5.5.

Example

Consider the constrained clause set N consisting of

$$\begin{array}{l} (P(x, x) \rightarrow \quad ;\ x \neq f(a)) \\ (P(f(y), z) \quad ;\ y \neq f(f(v)) \wedge z \neq f(f(a))). \end{array}$$

Without the constraints, N is unsatisfiable because we could derive $\square$ using the unifier

$$\sigma = \text{mgu}(P(x, x), P(f(y), z)) = \{x \mapsto f(y), z \mapsto f(y)\}.$$

Instead, we use σ to analyze the constraints.

$$\begin{array}{rl} & y\sigma \neq f(f(v)) \wedge z\sigma \neq f(f(a)) \wedge x\sigma \neq f(a) \\ = & y \neq f(f(v)) \wedge f(y) \neq f(f(a)) \wedge f(y) \neq f(a) \\ \Rightarrow & y \neq f(f(v)) \wedge y \neq f(a) \wedge y \neq a \\ \text{sorted} & y \neq a \wedge y \neq f(a) \wedge y \neq f(f(v)) \end{array}$$

For the solvability check, we use the signature $\{a/0, f/1\}$ given by the input.

$$\begin{array}{ll} & (y \neq a)\{y \mapsto a\} \text{ or } (y \neq f(a) \wedge y \neq f(f(v)))\{y \mapsto f(y)\}. \\ \Rightarrow & a \neq a \text{ or } y \neq a \wedge y \neq f(v). \\ \Rightarrow & \bot \text{ or } [(y \neq a)\{y \mapsto a\} \text{ or } (y \neq f(v))\{y \mapsto f(y)\}]. \\ \Rightarrow & \bot \text{ or } [a \neq a \text{ or } f(y) \neq f(v)]. \\ \Rightarrow & \bot \text{ or } [\bot \text{ or } \bot]. \\ \Rightarrow & \bot. \end{array}$$

Since the constraint is unsolvable, no resolution is performed and N is saturated. If the constraint were solvable, for example, without the $y \neq a$ constraint, we could replace $y \neq f(a) \wedge y \neq f(f(v))$ with $y \neq f(v)$ since $(y \neq f(a) \wedge y \neq f(f(v)))\{y \mapsto f(y)\}$ is unsolvable.

Many-sorted Signature

We can further improve the chance to identify unsolvable constraints by using many-sorted signatures. Then, the shallow instantiations θ_x^Σ only range over the function symbols in the sort of variable x. In array-theories, for example, data, indices, and arrays are usually modeled to be distinct from each other. A constraint on an index-variable is therefore already unsolvable if just all ground terms of the index-sort are excluded. Without sort information, such a constraint is still solvable.

An algorithm to extract a many-sorted signature with a maximal number of sorts was introduced in [Cla03]. At first, all function and predicate symbols start with different sorts. Then, compute equivalence classes of sorts that should be equal for the problem to be well-sorted: Wherever a function symbol is the argument of another function or predicate, their result and argument sorts are the same and for each variable in each clause, the sorts of its occurrences are also the same. Using union-find, the whole algorithm has linear complexity.

For first-order logic without equality, the resulting sorts are always *monotone* [CLS11], which means that the unsorted and many-sorted versions of the same problem are equisatisfiable. As a consequence, the resolution calculus can continue to use the unsorted signature while the constraint operations use the corresponding many-sorted signature.

Subset Check

Another important advantage of straight constraints is the ability to efficiently check whether the solutions of one constraint are a subset of the solutions of another constraint. The subset check allows common redundancy eliminations such as subsumption, condensation, and subsumption resolution to account for constraints. For example, $(P(x); x \neq a)$ subsumes $(P(x), Q(y); x \neq a \wedge y \neq b)$, while $(Q(x); x \neq a)$ does not.

If $\pi\sigma{\downarrow}{=} \top$ the subset condition on the solutions is trivially fulfilled, but even for general constraints, we can decide the subset relation.

Lemma 28. Let π and π' be solvable normal constraints. Then $\mathbb{D}^{\mathbb{V}}(\pi) \subseteq \mathbb{D}^{\mathbb{V}}(\pi')$ if and only if $\pi\{x \mapsto t\}$ has no solutions for every subconstraint $x \neq t$ in π'.

Proof. "$\Rightarrow$". Assume there exists a subconstraint $x \neq t$ in π', such that δ is a solution of $\pi\{x \mapsto t\}$. Then, $\{x \mapsto t\}\delta$ is a solution of π. Because $x\{x \mapsto t\}\delta = t\delta$ is an instance of t, $\{x \mapsto t\}\delta$ is not a solution of π'. This contradicts the assumption that $\mathbb{D}^{\mathbb{V}}(\pi) \subseteq \mathbb{D}^{\mathbb{V}}(\pi')$.

"$\Leftarrow$". Assume $\mathbb{D}^{\mathbb{V}}(\pi) \not\subseteq \mathbb{D}^{\mathbb{V}}(\pi')$. Let δ be a solution of π, but not of π'. There exists a subconstraint $x \neq t$ of π' such that $x\delta$ is an instance of t. Hence, there is a substitution σ such that $\delta = \{x \mapsto t\}\sigma$. Then, σ is a solution of $\pi\{x \mapsto t\}$. This contradicts the assumption that $\pi\{x \mapsto t\}$ has no solutions. □

Corollary 29. A constraint π is equivalent to $\pi \wedge x \neq t$, if $\pi\{x \mapsto t\}$ has no solution.

Proof. $\mathbb{D}^{\mathbb{V}}(\pi \wedge x \neq t) \subseteq \mathbb{D}^{\mathbb{V}}(\pi)$ holds trivially and $\mathbb{D}^{\mathbb{V}}(\pi) \subseteq \mathbb{D}^{\mathbb{V}}(\pi \wedge x \neq t)$ follows from Lemma 28. □

Note that if π was fully simplified according to Corollary 29, then $\pi\{x \mapsto t\}$ has no solutions if and only if $\pi\{x \mapsto t\}{\downarrow}{=} \bot$. This means $\pi\{x \mapsto t\}$ is unsolvable if there exists a constraint $x \neq s \in \pi$ such that t is an instance of s. If both π and π' are also sorted, we can therefore implement the subset check similar to a merge in merge-sort in $O(\text{size}(\pi) + \text{size}(\pi'))$.

Ordering

As the name ordered resolution suggests the ordering on atoms holds another key role in the efficiency of the calculus. Just as their unconstrained counterparts, the constrained resolution and factoring rules (Definitions 22 and 23) are only applied if the unified literals are (strictly) maximal. This means that we can avoid inferences if we can show that these literals are not maximal.

In general, we can continue to use any traditional ordering by ignoring the constraints. If an atom is not (strictly) maximal in the clausal part, it is not (strictly) maximal for any ground instance including the ones solving the constraint. On the other hand, the constraint could be excluding all ground instances where the atom is (strictly) maximal. For example, the lexicographic path ordering (LPO) can be extended with straight dismatching constraints. Similar extensions are possible for the RPO and KBO.

Definition 30 (LPO with Constraints)**.** Let s, t be terms, π a constraint and $\mathbb{F}$ finite. We define the constrained

lexicographic path ordering with the following transition system on pairs $(s \prec t; \pi)$:

$$
\begin{aligned}
(f(\vec{s_n}) \prec g(\vec{t_m})\,;\pi) &\Rightarrow \bigvee_{1\le i\le m} (f(\vec{s_n}) \preceq t_i;\pi) && \text{where } g \prec f\\
(f(\vec{s_n}) \prec g(\vec{t_m})\,;\pi) &\Rightarrow \bigwedge_{1\le i\le n} (s_i \prec g(\vec{t_m});\pi) && \text{where } f \prec g\\
(f(t_1,\dots,t_{i-1},s_i,\dots,s_n) \prec f(\vec{t_n})\,;\pi) &\Rightarrow \bigwedge_{i<j\le n} (s_j \prec f(\vec{t_n});\pi) && \wedge\,(s_i \preceq t_i;\pi)\\
(x \prec g(\vec{t_m})\,;\pi) &\Rightarrow \top && \text{where } x \in \mathrm{vars}(g(\vec{t_m}))\\
(s \prec x\quad\;;\pi) &\Rightarrow \bigwedge_{\sigma\in\theta_x^\Sigma} (s\sigma \prec x\sigma;\pi\sigma\!\downarrow) && \text{where } x \in \mathrm{lvars}(\pi)\\
(x \prec t\quad\;;\pi) &\Rightarrow \bigwedge_{\sigma\in\theta_x^\Sigma} (x\sigma \prec t\sigma;\pi\sigma\!\downarrow) && \text{where } x \in \mathrm{lvars}(\pi)\\
(s \prec t\quad\;;\bot) &\Rightarrow \top
\end{aligned}
$$

Lemma 31. Constrained lexicographic path ordering terminates.

Proof. The first to third rule each decrease the depth of s or t, while the constraint remains the same. The fourth and last rules are trivially terminating. For the fifth and sixth rule, $\mathrm{size}(\pi\sigma\!\downarrow) < \mathrm{size}(\pi)$. Therefore, the constrained lexicographic path ordering terminates. □

Note that the first four rules directly follow the definition of $\prec_{\mathrm{lpo}}$ and therefore $(s \prec t; \top)$ is the same as computing $s \prec_{\mathrm{lpo}} t$. The fifth and sixth rule reuse the principle idea of the solvability check to instantiate π until it normalizes to $\top$ or $\bot$. In the latter case, the last rule applies to remove instantiations that are excluded by the constraint. For example, consider $(a \prec x\ ;\ x \neq a)$ with precedence $a \prec b \prec f \prec g$. The only ground case where $a \not\prec_{\mathrm{lpo}} x$ is $(a \prec a\ ;\ a \neq a)$, but the constraint $a \neq a$ is unsolvable. Therefore, $a \prec_{\mathrm{lpo}} x$ under constraint $x \neq a$.

Furthermore, note that $(a \prec b\ ;\ b \neq a) \Rightarrow^* \top$ directly implies that also $(a \prec f(x)\ ;\ f(x) \neq a) \Rightarrow^* \top$ and $(a \prec g(x,y)\ ;\ g(x,y) \neq a) \Rightarrow^* \top$. In general, we need to check only the "minimal" solution with respect to of $\pi_x \prec_{\mathrm{lpo}}$ in the fifth rule. Analogously, the "maximal" solution of π_x suffices for the sixth rule, if it exists. A solution δ of π_x is minimal (maximal) if for every solution δ' of π_x, $x\delta \preceq_{\mathrm{lpo}} x\delta'$ ($x\delta \succeq_{\mathrm{lpo}} x\delta'$). For example, the constraint $x \neq a \wedge x \neq f(b) \wedge x \neq f(f(u))$ has under signature $a \prec b \prec f$ the minimal solution $\{x \mapsto b\}$ and maximal solution $\{x \mapsto f(a)\}$, but no maximal solution exists under signature $a \prec b \prec f \prec g$. If there is no maximal solution, it means that arbitrarily large terms are solutions for π_x. Then, we can immediately conclude $(x \prec t; \pi) \Rightarrow \bot$ if $x \notin \mathrm{vars}(t)$. Therefore, we can refine these rules to

$$
\begin{aligned}
(s \prec x; \pi) &\Rightarrow (s\delta \prec x\delta; \pi\delta\!\downarrow) && \text{where } x \in \mathrm{lvars}(\pi) \text{ and } \delta \text{ is the minimal solution of } \pi_x\\
(x \prec t\,;\pi) &\Rightarrow (x\delta \prec t\delta; \pi\delta\!\downarrow) && \text{where } x \in \mathrm{lvars}(\pi) \text{ and } \delta \text{ is the maximal solution of } \pi_x
\end{aligned}
$$

Just like the solvability check, minimal and maximal solutions can be computed in linear time. Unless an earlier case allows an early termination, the previous rules would eventually generate all cases for solutions of x including the minimal and maximal. Generating the cases alone corresponds to the work required to find the minimal or maximal solution.

Lemma 32. Iff $(s \prec t; \pi) \Rightarrow^* \top$, then $s\delta \prec_{\mathrm{lpo}} t\delta$ for every solution δ of π.

Proof. "⇒": By induction on the derivation $(s \prec t; \pi) \Rightarrow^* \top$.

The first four rules follow directly from the definition of $\prec_{\mathrm{lpo}}$. For example, consider the first rule. Let $(f(\vec{s_n}) \prec g(\vec{t_m}); \pi) \Rightarrow \bigvee_{1\le i\le m}(f(\vec{s_n}) \preceq t_i; \pi) \Rightarrow^* \top$. Then, $(f(\vec{s_n}) \preceq t_i; \pi) \Rightarrow^* \top$ for at least one $1 \le i \le m$. By the inductive hypothesis, $f(\vec{s_n})\delta \prec_{\mathrm{lpo}} t_i\delta$ for every solution δ of π. Therefore by the definition of $\prec_{\mathrm{lpo}}$, $f(\vec{s_n})\delta \prec_{\mathrm{lpo}} g(\vec{t_m})\delta$ for every solution δ of π.

For the fifth rule, let $(s \prec x; \pi) \Rightarrow \bigwedge_{\sigma\in\theta_x^\Sigma}(s\sigma \prec x\sigma; \pi\sigma) \Rightarrow^* \top$. Assume there is a solution δ of π such that $s\delta \not\prec_{\mathrm{lpo}} x\delta$. Then, $x\delta = f(y_1,\dots,y_n)\delta'$ for some substitution δ' and there is a $\sigma = \{x \mapsto f(y_1,\dots,y_n)\} \in \Theta_x^\Sigma$.

Since δ' is a solution of $\pi\sigma$, $s\sigma\delta' \not\prec_{\text{lpo}} x\sigma\delta'$ contradicts the inductive hypothesis on $(s\sigma \prec x\sigma; \pi\sigma) \Rightarrow^* \top$. The sixth rule is analogous. The last rule is trivial as $\bot$ has no solutions.

"$\Leftarrow$": Let $(s \prec t; \pi) \not\Rightarrow^* \top$, i.e., there is normal form $(s \prec t; \pi) \Rightarrow^* F$, where F is either $\bot$ or a formula consisting of conjunctions and disjunctions of pairs $(s \prec t; \pi)$ where no rule can be applied. We prove by induction on the derivation $(s \prec t; \pi) \Rightarrow^* F$, that there is a solution δ of π such that $s\delta \not\prec_{\text{lpo}} t\delta$.

Again, the first four rules follow directly from the definition of $\prec_{\text{lpo}}$. For example, consider the second rule. Let $(f(\vec{s_n}) \prec g(\vec{t_m}); \pi) \Rightarrow \bigwedge_{1 \le i \le n}(s_i \preceq g(\vec{t_m}); \pi) \not\Rightarrow^* \top$. Then, $(f(\vec{s_n}) \preceq t_i; \pi) \not\Rightarrow^* \top$ for at least one $1 \le i \le n$. By the inductive hypothesis, $s_i\delta \not\prec_{\text{lpo}} g(\vec{t_m})\delta$ for a solution δ of π. Therefore, by the definition of $\prec_{\text{lpo}}$, $f(\vec{s_n})\delta \not\prec_{\text{lpo}} g(\vec{t_m})\delta$ for the solution δ of π.

For the fifth rule, let $(s \prec x; \pi) \Rightarrow \bigwedge_{\sigma \in \theta_x^\Sigma}(s\sigma \prec x\sigma; \pi\sigma) \not\Rightarrow^* \top$. Then, $(s\sigma \prec x\sigma; \pi\sigma) \not\Rightarrow^* \top$ for at least one $\sigma \in \Theta_x^\Sigma$. By the inductive hypothesis, $s\sigma\delta \not\prec_{\text{lpo}} x\sigma\delta$ for a solution δ of $\pi\sigma$. Therefore, $s\sigma\delta \not\prec_{\text{lpo}} x\sigma\delta$ for the solution $\sigma\delta$ of π. The sixth rule is analogous and the last rule contradicts $(s \prec t; \pi) \not\Rightarrow^* \top$. □

Constrained *LPO* extends to atoms and literals in the usual way.

Corollary 33. Let $(C \vee E; \pi)$ be a constrained clause. If $(E \prec L; \pi) \Rightarrow^* \top$ ($(E \preceq L; \pi) \Rightarrow^* \top$) for some literal L in C, then E is not (strictly) maximal in $(C \vee E; \pi)$ under $\prec_{\text{lpo}}$.

5 Conclusion

We have presented a sound and complete ordered resolution calculus for first-order clauses with straight dismatching constraints. This constraint clause language enables a refinement method for our abstraction-refinement calculus [TW15] by replacing an input clause by exactly two constrained clauses instead of a quadratically $((|\Sigma| - 1) \cdot \mathrm{d}(t)$, see Section 1) sized explicit representation using standard clauses [LM87]. At the same time, a resolution prover only needs to store the input constraint terms once and can then store atomic constraints in constant space. Aside from sorting constraints, all operations on constraints perform in linear time. This is in contrast to general dismatching constraints where solvability is NP-hard [CL89]. SAT can be encoded as solvability of dismatching constraints with only ground terms on the right-hand side. For example, the SAT clause $C_i = x_1 \vee \neg x_2 \vee x_3$ is encoded by the atomic constraint $\pi_i = f_i(x_1, x_2, x_3) \neq f_i(\text{false}, \text{true}, \text{false})$. Then a general dismatching constraint $\pi_1 \wedge \cdots \wedge \pi_n$ as the result of the encoding of a clause set $\{C_1, \ldots, C_n\}$ with signature $\Sigma = \{\text{true}, \text{false}\}$ is solvable iff the clause set $\{C_1, \ldots, C_n\}$ is satisfiable. Currently, we are working on the extension of classical decidable clause fragments to clauses with straight dismatching constraints.

Acknowledgments

The authors would like to acknowledge discussions with and helpful comments by Gábor Alagi. We further thank our reviewers for their helpful and constructive comments.

References

[AW15] Gábor Alagi and Christoph Weidenbach. NRCL - a model building approach to the Bernays-Schönfinkel fragment. In Carsten Lutz and Silvio Ranise, editors, *Frocos (Wroclaw)*, volume 9322 of *Lecture Notes in Artificial Intelligence*, pages 69–84. Springer, 2015.

[BG94] 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.

[CL89] Hubert Comon and Pierre Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7(3/4):371–425, 1989.

[Cla03] Koen Claessen. New techniques that improve MACE-style finite model finding. In *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.

[CLS11] Koen Claessen, Ann Lilliesträom, and Nicholas Smallbone. Sort it out with monotonicity - translating between many-sorted and unsorted first-order logic. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 207–221. Springer, 2011.

[Com91] Hubert Comon. Disunification: A survey. In Jean-Louis Lassez and Gordon D. Plotkin, editors, *Computational Logic - Essays in Honor of Alan Robinson*, pages 322–359. The MIT Press, 1991.

[CZ92] Ricardo Caferra and Nicolas Zabel. A method for simultanous search for refutations and models by equational constraint solving. *Journal of Symbolic Computation*, 13(6):613–642, 1992.

[Kor13] 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.

[LM87] J.-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–317, September 1987.

[NR01] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 371–443. Elsevier and MIT Press, 2001.

[Pel86] Francis Jeffry Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2(2):191–216, 1986.

[TW15] Andreas Teucke and Christoph Weidenbach. First-order logic theorem proving and model building via approximation and instantiation. In Carsten Lutz and Silvio Ranise, editors, *Frontiers of Combining Systems: 10th International Symposium, FroCoS 2015, Wroclaw, Poland, September 21-24, 2015, Proceedings*, pages 85–100, Cham, 2015. Springer International Publishing.