SCL(EQ): SCL for First-Order Logic with Equality

Hendrik Leidinger[1,2](✉) and Christoph Weidenbach[1]

[1] Max-Planck Institute for Informatics, Saarbrücken, Germany
{hleiding,weidenbach}@mpi-inf.mpg.de

[2] Graduate School of Computer Science, Saarbrücken, Germany

Abstract. We propose a new calculus SCL(EQ) for first-order logic with equality that only learns non-redundant clauses. Following the idea of CDCL (Conflict Driven Clause Learning) and SCL (Clause Learning from Simple Models) a ground literal model assumption is used to guide inferences that are then guaranteed to be non-redundant. Redundancy is defined with respect to a dynamically changing ordering derived from the ground literal model assumption. We prove SCL(EQ) sound and complete and provide examples where our calculus improves on superposition.

Keywords: First-order logic with equality · Term rewriting · Model-based reasoning

1 Introduction

There has been extensive research on sound and complete calculi for first-order logic with equality. The current prime calculus is superposition [2], where ordering restrictions guide paramodulation inferences and an abstract redundancy notion enables a number of clause simplification and deletion mechanisms, such as rewriting or subsumption. Still this "syntactic" form of superposition infers many redundant clauses. The completeness proof of superposition provides a "semantic" way of generating only non-redundant clauses, however, the underlying ground model assumption cannot be effectively computed in general [31]. It requires an ordered enumeration of infinitely many ground instances of the given clause set, in general. Our calculus overcomes this issue by providing an effective way of generating ground model assumptions that then guarantee non-redundant inferences on the original clauses with variables.

The underlying ordering is based on the order of ground literals in the model assumption, hence changes during a run of the calculus. It incorporates a standard rewrite ordering. For practical redundancy criteria this means that both rewriting and redundancy notions that are based on literal subset relations are permitted to dynamically simplify or eliminate clauses. Newly generated clauses are non-redundant, so redundancy tests are only needed backwards. Furthermore, the ordering is automatically generated by the structure of the clause set.

Instead of a fixed ordering as done in the superposition case, the calculus finds and changes an ordering according to the currently easiest way to make progress, analogous to CDCL (Conflict Driven Clause Learning) [11,21,25,29,34].

Typical for CDCL and SCL (Clause Learning from Simple Models) [1,14,18] approaches to reasoning, the development of a model assumption is done by decisions and propagations. A decision guesses a ground literal to be true whereas a propagation concludes the truth of a ground literal through an otherwise false clause. While propagations in CDCL and propositional logic are restricted to the finite number of propositional variables, in first-order logic there can already be infinite propagation sequences [18]. In order to overcome this issue, model assumptions in SCL(EQ) are at any point in time restricted to a finite number of ground literals, hence to a finite number of ground instances of the clause set at hand. Therefore, without increasing the number of considered ground literals, the calculus either finds a refutation or runs into a *stuck state* where the current model assumption satisfies the finite number of ground instances. In this case one can check whether the model assumption can be generalized to a model assumption of the overall clause set or the information of the stuck state can be used to appropriately increase the number of considered ground literals and continue search for a refutation. SCL(EQ) does not require exhaustive propagation, in general, it just forbids the decision of the complement of a literal that could otherwise be propagated.

For an example of SCL(EQ) inferring clauses, consider the three first-order clauses

$$C_1 := h(x) \approx g(x) \lor c \approx d \qquad C_2 := f(x) \approx g(x) \lor a \approx b$$
$$C_3 := f(x) \not\approx h(x) \lor f(x) \not\approx g(x)$$

with a Knuth-Bendix Ordering (KBO), unique weight 1, and precedence $d \prec c \prec b \prec a \prec g \prec h \prec f$. A Superposition Left [2] inference between C_2 and C_3 results in

$$C_4' := h(x) \not\approx g(x) \lor f(x) \not\approx g(x) \lor a \approx b.$$

For SCL(EQ) we start by building a partial model assumption, called a *trail*, with two decisions

$$\Gamma := [h(a) \approx g(a)^{1:(h(x)\approx g(x) \lor h(x)\not\approx g(x))\cdot\sigma}, f(a) \approx g(a)^{2:(f(x)\approx g(x) \lor f(x)\not\approx g(x))\cdot\sigma}]$$

where $\sigma := \{x \mapsto a\}$. Decisions and propagations are always ground instances of literals from the first-order clauses, and are annotated with a level and a justification clause, in case of a decision a tautology. Now with respect to Γ clause C_3 is false with grounding σ, and rule Conflict is applicable; see Sect. 3.1 for details on the inference rules. In general, clauses and justifications are considered variable disjoint, but for simplicity of the presentation of this example, we repeat variable names here as long as the same ground substitution is shared. The maximal literal in $C_3\sigma$ is $(f(x) \not\approx h(x))\sigma$ and a rewrite refutation using the ground equations from the trail results in the justification clause

$$(g(x) \not\approx g(x) \lor f(x) \not\approx g(x) \lor f(x) \not\approx g(x) \lor h(x) \not\approx g(x))\cdot\sigma$$

where for the refutation justification clauses and all otherwise inferred clauses we use the grounding σ for guidance, but operate on the clauses with variables. The respective ground clause is smaller than $(f(x) \not\approx h(x))\sigma$, false with respect to Γ and becomes our new conflict clause by an application of our inference rule Explore-Refutation. It is simplified by our inference rules Equality-Resolution and Factorize, resulting in the finally learned clause

$$C_4 := h(x) \not\approx g(x) \vee f(x) \not\approx g(x)$$

which is then used to apply rule Backtrack to the trail. Observe that C_4 is strictly stronger than C_4' the clause inferred by superposition and that C_4 cannot be inferred by superposition. Thus SCL(EQ) can infer stronger clauses than superposition for this example.

Related Work: SCL(EQ) is based on ideas of SCL [1,14,18] but for the first time includes a native treatment of first-order equality reasoning. Similar to [14] propagations need not to be exhaustively applied, the trail is built out of decisions and propagations of ground literals annotated by first-order clauses, SCL(EQ) only learns non-redundant clauses, but for the first time conflicts resulting out of a decision have to be considered, due to the nature of the equality relation.

There have been suggested several approaches to lift the idea of an inference guiding model assumption from propositional to full first-order logic [6,12,13,18]. They do not provide a native treatment of equality, e.g., via paramodulation or rewriting.

Baumgartner et al. describe multiple calculi that handle equality by using unit superposition style inference rules and are based on either hyper tableaux [5] or DPLL [15,16]. Hyper tableaux fix a major problem of the well-known free variable tableaux, namely the fact that free variables within the tableau are rigid, i.e., substitutions have to be applied to all occurrences of a free variable within the entire tableau. Hyper tableaux with equality [7] in turn integrates unit superposition style inference rules into the hyper tableau calculus.

Another approach that is related to ours is the model evolution calculus with equality ($\mathcal{ME}_\mathcal{E}$) by Baumgartner et al. [8,9] which lifts the DPLL calculus to first-order logic with equality. Similar to our approach, $\mathcal{ME}_\mathcal{E}$ creates a candidate model until a clause instance contradicts this model or all instances are satisfied by the model. The candidate model results from a so-called context, which consists of a finite set of non-ground rewrite literals. Roughly speaking, a context literal specifies the truth value of all its ground instances unless a more specific literal specifies the complement. Initially the model satisfies the identity relation over the set of all ground terms. Literals within a context may be universal or parametric, where universal literals guarantee all its ground instances to be true. If a clause contradicts the current model, it is repaired by a non-deterministic split which adds a parametric literal to the current model. If the added literal does not share any variables in the contradictory clause it is added as a universal literal.

Another approach by Baumgartner and Waldmann [10] combined the superposition calculus with the Model Evolution calculus with equality. In this calculus the atoms of the clauses are labeled as “split atoms” or “superposition atoms”. The superposition part of the calculus then generates a model for the superposition atoms while the model evolution part generates a model for the split atoms. Conversely, this means that if all atoms are labeled as “split atom”, the calculus behaves similar to the model evolution calculus. If all atoms are labeled as “superposition atom”, it behaves like the superposition calculus.

Both the hyper tableaux calculus with equality and the model evolution calculus with equality allow only unit superposition applications, while SCL(EQ) inferences are guided paramodulation inferences on clauses of arbitrary length. The model evolution calculus with equality was revised and implemented in 2011 [8] and compares its performance with that of hyper tableaux. Model evolution performed significantly better, with more problems solved in all relevant TPTP [30] categories, than the implementation of the hyper tableaux calculus.

Plaisted et al. [27] present the Ordered Semantic Hyper-Linking (OSHL) calculus. OSHL is an instantiation based approach that repeatedly chooses ground instances of a non-ground input clause set such that the current model does not satisfy the current ground clause set. A further step repairs the current model such that it satisfies the ground clause set again. The algorithm terminates if the set of ground clauses contains the empty clause. OSHL supports rewriting and narrowing, but only with unit clauses. In order to handle non-unit clauses it makes use of other mechanisms such as Brand’s Transformation [3].

Inst-Gen [22] is an instantiation based calculus, that creates ground instances of the input first-order formulas which are forwarded to a SAT solver. If a ground instance is unsatisfiable, then the first-order set is as well. If not then the calculus creates more instances. The Inst-Gen-EQ calculus [23] creates instances by extracting instantiations of unit superposition refutations of selected literals of the first-order clause set. The ground abstraction is then extended by the extracted clauses and an SMT solver then checks the satisfiability of the resulting set of equational and non-equational ground literals.

In favor of examples and explanations we omit all proofs. They are available in an extended version published as a research report [24]. The rest of the paper is organized as follows. Section 2 provides basic formalisms underlying SCL(EQ). The rules of the calculus are presented in Sect. 3. Soundness and completeness results are provided in Sect. 4. We end with a discussion of obtained results and future work, Sect. 5. The main contribution of this paper is the SCL(EQ) calculus that only learns non-redundant clauses, permits subset based redundancy elimination and rewriting, and its soundness and completeness.

2 Preliminaries

We assume a standard first-order language with equality and signature $\Sigma = (\Omega, \emptyset)$ where the only predicate symbol is equality $\approx$. N denotes a set of clauses, C, D denote clauses, L, K, H denote equational literals, A, B denote equational atoms, t, s terms from $T(\Omega, \mathcal{X})$ for an infinite set of variables $\mathcal{X}$, f, g, h function symbols from Ω, a, b, c constants from Ω and x, y, z variables from $\mathcal{X}$. The function *comp* denotes the complement of a literal. We write $s \not\approx t$ as a shortcut for

$\neg(s \approx t)$. The literal $s \# t$ may denote both $s \approx t$ and $s \not\approx t$. The semantics of first-order logic and semantic entailment $\models$ is defined as usual.

By σ, τ, δ we denote substitutions, which are total mappings from variables to terms. Let σ be a substitution, then its finite domain is defined as $dom(\sigma) := \{x \mid x\sigma \neq x\}$ and its codomain is defined as $codom(\sigma) = \{t \mid x\sigma = t, x \in dom(\sigma)\}$. We extend their application to literals, clauses and sets of such objects in the usual way. A term, literal, clause or sets of these objects is ground if it does not contain any variable. A substitution σ is *ground* if $codom(\sigma)$ is ground. A substitution σ is *grounding* for a term t, literal L, clause C if $t\sigma$, $L\sigma$, $C\sigma$ is ground, respectively. By $C\cdot\sigma$, $L\cdot\sigma$ we denote a closure consisting of a clause C, literal L and a grounding substitution σ, respectively. The function *gnd* computes the set of all ground instances of a literal, clause, or clause set. The function mgu denotes the most general unifier of terms, atoms, literals, respectively. We assume that mgus do not introduce fresh variables and that they are idempotent.

The set of positions $\text{pos}(L)$ of a literal (term $\text{pos}(t)$) is inductively defined as usual. The notion $L|_p$ denotes the subterm of a literal L ($t|_p$ for term t) at position $p \in \text{pos}(L)$ ($p \in \text{pos}(t)$). The replacement of a subterm of a literal L (term t) at position $p \in \text{pos}(L)$ ($p \in \text{pos}(t)$) by a term s is denoted by $L[s]_p$ ($t[s]_p$). For example, the term $f(a, g(x))$ has the positions $\{\epsilon, 1, 2, 21\}$, $f(a, g(x))|_{21} = x$ and $f(a, g(x))[b]_2$ denotes the term $f(a, b)$.

Let R be a set of rewrite rules $l \to r$, called a *term rewrite system* (TRS). The rewrite relation $\to_R \subseteq T(\Omega, \mathcal{X}) \times T(\Omega, \mathcal{X})$ is defined as usual by $s \to_R t$ if there exists $(l \to r) \in R$, $p \in \text{pos}(s)$, and a matcher σ, such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$. We write $s = t{\downarrow_R}$ if s is the normal form of t in the rewrite relation $\to_R$. We write $s \# t = (s' \# t'){\downarrow_R}$ if s is the normal form of s' and t is the normal form of t'. A rewrite relation is terminating if there is no infinite descending chain $t_0 \to t_1 \to ...$ and confluent if $t \mathrel{{}^*{\leftarrow}} s \to^* t'$ implies $t \leftrightarrow^* t'$. A rewrite relation is convergent if it is terminating and confluent. A rewrite order is a irreflexive and transitive rewrite relation. A TRS R is terminating, confluent, convergent, if the rewrite relation $\to_R$ is terminating, confluent, convergent, respectively. A term t is called irreducible by a TRS R if no rule from R rewrites t. Otherwise it is called reducible. A literal, clause is irreducible if all of its terms are irreducible, and reducible otherwise. A substitution σ is called irreducible if any $t \in codom(\sigma)$ is irreducible, and reducible otherwise.

Let $\prec_T$ denote a well-founded rewrite ordering on terms which is total on ground terms and for all ground terms t there exist only finitely many ground terms $s \prec_T t$. We call $\prec_T$ a *desired* term ordering. We extend $\prec_T$ to equations by assigning the multiset $\{s, t\}$ to positive equations $s \approx t$ and $\{s, s, t, t\}$ to inequations $s \not\approx t$. Furthermore, we identify $\prec_T$ with its multiset extension comparing multisets of literals. For a (multi)set of terms $\{t_1, \ldots, t_n\}$ and a term t, we define $\{t_1, \ldots, t_n\} \prec_T t$ if $\{t_1, \ldots, t_n\} \prec_T \{t\}$. For a (multi)set of Literals $\{L_1, \ldots, L_n\}$ and a term t, we define $\{L_1, \ldots, L_n\} \prec_T t$ if $\{L_1, \ldots, L_n\} \prec_T \{\{t\}\}$. Given a ground term β then $\text{gnd}_{\prec_T\beta}$ computes the set of all ground instances of a literal, clause, or clause set where the groundings are smaller than β according to the ordering $\prec_T$. Given a set (sequence) of ground literals Γ let $\operatorname{conv}(\Gamma)$ be a convergent rewrite system out of the positive equations in Γ using $\prec_T$.

Let $\prec$ be a well-founded, total, strict ordering on ground literals, which is lifted to clauses and clause sets by its respective multiset extension. We overload $\prec$ for literals, clauses, clause sets if the meaning is clear from the context. The ordering is lifted to the non-ground case via instantiation: we define $C \prec D$ if for all grounding substitutions σ it holds $C\sigma \prec D\sigma$. Then we define $\preceq$ as the reflexive closure of $\prec$ and $N^{\preceq C} := \{D \mid D \in N \text{ and } D \preceq C\}$ and use the standard superposition style notion of redundancy [2].

Definition 1 (Clause Redundancy). *A ground clause C is* redundant *with respect to a set N of ground clauses and an ordering $\prec$ if $N^{\preceq C} \models C$. A clause C is* redundant *with respect to a clause set N and an ordering $\prec$ if for all $C' \in gnd(C)$, C' is redundant with respect to $gnd(N)$.*

3 The SCL(EQ) Calculus

We start the introduction of the calculus by defining the ingredients of an SCL(EQ) state.

Definition 2 (Trail). *A* trail *$\Gamma := [L_1^{i_1:C_1\cdot\sigma_1}, ..., L_n^{i_n:C_n\cdot\sigma_n}]$ is a consistent sequence of ground equations and inequations where L_j is annotated by a level i_j with $i_{j-1} \leq i_j$, and a closure $C_j\cdot\sigma_j$. We omit the annotations if they are not needed in a certain context. A ground literal L is true in Γ if $\Gamma \models L$. A ground literal L is false in Γ if $\Gamma \models comp(L)$. A ground literal L is undefined in Γ if $\Gamma \not\models L$ and $\Gamma \not\models comp(L)$. Otherwise it is defined. For each literal L_j in Γ it holds that L_j is undefined in $[L_1, ..., L_{j-1}]$ and irreducible by* $\operatorname{conv}(\{L_1, ..., L_{j-1}\})$.

The above definition of truth and undefinedness is extended to clauses in the obvious way. The notions of true, false, undefined can be parameterized by a ground term β by saying that L is β-undefined in a trail Γ if $\beta \prec_T L$ or L is undefined. The notions of a β-true, β-false term are restrictions of the above notions to literals smaller β, respectively. All SCL(EQ) reasoning is layered with respect to a ground term β.

Definition 3. *Let Γ be a trail and L a ground literal such that L is defined in Γ. By* $\operatorname{core}(\Gamma; L)$ *we denote a minimal subsequence $\Gamma' \subseteq \Gamma$ such that L is defined in Γ'. By* $\operatorname{cores}(\Gamma; L)$ *we denote the set of all cores.*

Note that $core(\Gamma; L)$ is not necessarily unique. There can be multiple cores for a given trail Γ and ground literal L.

Definition 4 (Trail Ordering). *Let $\Gamma := [L_1, ..., L_n]$ be a trail. The (partial) trail ordering $\prec_\Gamma$ is the sequence ordering given by Γ, i.e., $L_i \prec_\Gamma L_j$ if $i < j$ for all $1 \leq i, j \leq n$.*

Definition 5 (Defining Core and Defining Literal). *For a trail Γ and a sequence of literals $\Delta \subseteq \Gamma$ we write $max_{\prec_\Gamma}(\Delta)$ for the largest literal in Δ according to the trail ordering $\prec_\Gamma$. Let Γ be a trail and L a ground literal such that L is defined in Γ. Let $\Delta \in cores(\Gamma; L)$ be a sequence of literals where $max_{\prec_\Gamma}(\Delta) \preceq_\Gamma max_{\prec_\Gamma}(\Lambda)$ for all $\Lambda \in cores(\Gamma; L)$, then* $\max_\Gamma(L) := \max_{\prec_\Gamma}(\Delta)$ *is called the* defining literal *and Δ is called a* defining core *for L in Γ. If $cores(\Gamma; L)$ contains only the empty core, then L has no* defining literal *and no* defining core*.*

Note that there can be multiple defining cores but only one defining literal for any defined literal L. For example, consider a trail $\Gamma := [f(a) \approx f(b)^{1:C_1\cdot\sigma_1}, a \approx b^{2:C_2\cdot\sigma_2}, b \approx c^{3:C_3\cdot\sigma_3}]$ with an ordering $\prec_T$ that orders the terms of the equations from left to right, and a literal $g(f(a)) \approx g(f(c))$. Then the defining cores are $\Delta_1 := [a \approx b, b \approx c]$ and $\Delta_2 := [f(a) \approx f(b), b \approx c]$. The defining literal, however, is in both cases $b \approx c$. Defined literals that have no defining core and therefore no defining literal are literals that are trivially false or true. Consider, for example, $g(f(a)) \approx g(f(a))$. This literal is trivially true in Γ. Thus an empty subset of Γ is sufficient to show that $g(f(a)) \approx g(f(a))$ is defined in Γ.

Definition 6 (Literal Level). *Let Γ be a trail. A ground literal $L \in \Gamma$ is of* level *i if L is annotated with i in Γ. A defined ground literal $L \notin \Gamma$ is of level i if the defining literal of L is of level i. If L has no defining literal, then L is of level 0. A ground clause D is of level i if i is the maximum level of a literal in D.*

The restriction to minimal subsequences for the defining literal and definition of a level eventually guarantee that learned clauses are smaller in the trail ordering. This enables completeness in combination with learning non-redundant clauses as shown later.

Lemma 7. *Let Γ_1 be a trail and K a defined literal that is of level i in Γ_1. Then K is of level i in a trail $\Gamma := \Gamma_1, \Gamma_2$.*

Definition 8. *Let Γ be a trail and $L \in \Gamma$ a literal. L is called a* decision literal *if $\Gamma = \Gamma_0, K^{i:C\cdot\tau}, L^{i+1:C'\cdot\tau'}, \Gamma_1$. Otherwise L is called a* propagated literal*.*

In our above example $g(f(a)) \approx g(f(c))$ is of level 3 since the defining literal $b \approx c$ is annotated with 3. $a \not\approx b$ on the other hand is of level 2.

We define a well-founded total strict ordering which is induced by the trail and with which non-redundancy is proven in Sect. 4. Unlike SCL [14,18] we use this ordering for the inference rules as well. In previous SCL calculi, conflict resolution automatically chooses the greatest literal and resolves with this literal. In SCL(EQ) this is generalized. Coming back to our running example above, suppose we have a conflict clause $f(b) \not\approx f(c) \lor b \not\approx c$. The defining literal for both inequations is $b \approx c$. So we could do paramodulation inferences with both literals. The following ordering makes this non-deterministic choice deterministic.

Definition 9 (Trail Induced Ordering). *Let $\Gamma := [L_1^{i_1:C_1\cdot\sigma_1}, ..., L_n^{i_n:C_n\cdot\sigma_n}]$ be a trail, β a ground term such that $\{L_1, ..., L_n\} \prec_T \beta$ and $M_{i,j}$ all β-defined ground literals not contained in $\Gamma \cup comp(\Gamma)$: for a defining literal $max_\Gamma(M_{i,j}) = L_i$ and for two literals $M_{i,j}$, $M_{i,k}$ we have $j < k$ if $M_{i,j} \prec_T M_{i,k}$. The trail induces a total well-founded strict order $\prec_{\Gamma^*}$ on β-defined ground literals $M_{k,l}, M_{m,n}$, L_i, L_j of level greater than zero, where*

1. $M_{i,j} \prec_{\Gamma^*} M_{k,l}$ *if* $i < k$ *or* $(i = k$ *and* $j < l)$
2. $L_i \prec_{\Gamma^*} L_j$ *if* $L_i \prec_\Gamma L_j$
3. $comp(L_i) \prec_{\Gamma^*} L_j$ *if* $L_i \prec_\Gamma L_j$
4. $L_i \prec_{\Gamma^*} comp(L_j)$ *if* $L_i \prec_\Gamma L_j$ *or* $i = j$
5. $comp(L_i) \prec_{\Gamma^*} comp(L_j)$ *if* $L_i \prec_\Gamma L_j$
6. $L_i \prec_{\Gamma^*} M_{k,l}$, $\mathrm{comp}(L_i) \prec_{\Gamma^*} M_{k,l}$ *if* $i \le k$
7. $M_{k,l} \prec_{\Gamma^*} L_i$, $M_{k,l} \prec_{\Gamma^*} \mathrm{comp}(L_i)$ *if* $k < i$

and for all β-defined literals L of level zero:

8. $\prec_{\Gamma^*} := \prec_T$
9. $L \prec_{\Gamma^*} K$ *if K is of level greater than zero and K is β-defined*

and can eventually be extended to β-undefined ground literals K, H by

10. $K \prec_{\Gamma^*} H$ *if* $K \prec_T H$
11. $L \prec_{\Gamma^*} H$ *if L is β-defined*

The literal ordering $\prec_{\Gamma^*}$ is extended to ground clauses by multiset extension and identified with $\prec_{\Gamma^*}$ as well.

Lemma 10 (Properties of $\prec_{\Gamma^*}$).

1. *$\prec_{\Gamma^*}$ is well-defined.*
2. *$\prec_{\Gamma^*}$ is a total strict order, i.e. $\prec_{\Gamma^*}$ is irreflexive, transitive and total.*
3. *$\prec_{\Gamma^*}$ is a well-founded ordering.*

Example 11. Assume a trail $\Gamma := [a \approx b^{1:C_0\cdot\sigma_0}, c \approx d^{1:C_1\cdot\sigma_1}, f(a') \not\approx f(b')^{1:C_2\cdot\sigma_2}]$, select KBO as the term ordering $\prec_T$ where all symbols have weight one and $a \prec a' \prec b \prec b' \prec c \prec d \prec f$ and a ground term $\beta := f(f(a))$. According to the trail induced ordering we have that $a \approx b \prec_{\Gamma^*} c \approx d \prec_{\Gamma^*} f(a') \not\approx f(b')$ by 9.2. Furthermore we have that

$$a \approx b \prec_{\Gamma^*} a \not\approx b \prec_{\Gamma^*} c \approx d \prec_{\Gamma^*} c \not\approx d \prec_{\Gamma^*} f(a') \not\approx f(b') \prec_{\Gamma^*} f(a') \approx f(b')$$

by 9.3 and 9.4. Now for any literal L that is *β-defined* in Γ and the defining literal is $a \approx b$ it holds that $a \not\approx b \prec_{\Gamma^*} L \prec_{\Gamma^*} c \approx d$ by 9.6 and 9.7. This holds analogously for all literals that are *β-defined* in Γ and the defining literal is $c \approx d$ or $f(a') \not\approx f(b')$. Thus we get:

$$\begin{gathered} L_1 \prec_{\Gamma^*} ... \prec_{\Gamma^*} a \approx b \prec_{\Gamma^*} a \not\approx b \prec_{\Gamma^*} f(a) \approx f(b) \prec_{\Gamma^*} f(a) \not\approx f(b) \prec_{\Gamma^*} \\ c \approx d \prec_{\Gamma^*} c \not\approx d \prec_{\Gamma^*} f(c) \approx f(d) \prec_{\Gamma^*} f(c) \not\approx f(d) \prec_{\Gamma^*} \\ f(a') \not\approx f(b') \prec_{\Gamma^*} f(a') \approx f(b') \prec_{\Gamma^*} a' \approx b' \prec_{\Gamma^*} a' \not\approx b' \prec_{\Gamma^*} K_1 \prec_{\Gamma^*} \ldots \end{gathered}$$

where K_i are the *β-undefined* literals and L_j are the trivially defined literals.

Definition 12 (Rewrite Step). *A* rewrite step *is a five-tuple* $(s\#t\cdot\sigma, s\#t \vee C\cdot\sigma, R, S, p)$ *and inductively defined as follows. The tuple* $(s\#t\cdot\sigma, s\#t \vee C\cdot\sigma, \epsilon, \epsilon, \epsilon)$ *is a rewrite step. Given rewrite steps* R, S *and a position* p *then* $(s\#t\cdot\sigma, s\#t \vee C\cdot\sigma, R, S, p)$ *is a* rewrite step*. The literal* $s\#t$ *is called the* rewrite literal*. In case* R, S *are not* ϵ*, the rewrite literal of* R *is an equation.*

Rewriting is one of the core features of our calculus. The following definition describes a rewrite inference between two clauses. Note that unlike the superposition calculus we allow rewriting below variable level.

Definition 13 (Rewrite Inference). *Let* $I_1 := (l_1 \approx r_1\cdot\sigma_1, l_1 \approx r_1 \vee C_1\cdot\sigma_1, R_1, L_1, p_1)$ *and* $I_2 := (l_2\#r_2\cdot\sigma_2, l_2\#r_2 \vee C_2\cdot\sigma_2, R_2, L_2, p_2)$ *be two variable disjoint rewrite steps where* $r_1\sigma_1 \prec_T l_1\sigma_1$*,* $(l_2\#r_2)\sigma_2|_p = l_1\sigma_1$ *for some position* p*. We distinguish two cases:*

1. *if* $p \in \mathrm{pos}(l_2\#r_2)$ *and* $\mu := \mathrm{mgu}((l_2\#r_2)|_p, l_1)$ *then* $(((l_2\#r_2)[r_1]_p)\mu\cdot\sigma_1\sigma_2, ((l_2\#r_2)[r_1]_p)\mu \vee C_1\mu \vee C_2\mu\cdot\sigma_1\sigma_2, I_1, I_2, p)$ *is the result of a rewrite inference.*
2. *if* $p \notin \mathrm{pos}(l_2\#r_2)$ *then let* $(l_2\#r_2)\delta$ *be the most general instance of* $l_2\#r_2$ *such that* $p \in \mathrm{pos}((l_2\#r_2)\delta)$*,* δ *introduces only fresh variables and* $(l_2\#r_2)\delta\sigma_2\rho = (l_2\#r_2)\sigma_2$ *for some minimal* ρ*. Let* $\mu := \mathrm{mgu}((l_2\#r_2)\delta|_p, l_1)$*. Then* $((l_2\#r_2)\delta[r_1]_p\mu\cdot\sigma_1\sigma_2\rho, (l_2\#r_2)\delta[r_1]_p\mu \vee C_1\mu \vee C_2\delta\mu\cdot\sigma_1\sigma_2\rho, I_1, I_2, p)$ *is the result of a rewrite inference.*

Lemma 14. *Let* $I_1 := (l_1 \approx r_1\cdot\sigma_1, l_1 \approx r_1 \vee C_1\cdot\sigma_1, R_1, L_1, p_1)$ *and* $I_2 := (l_2\#r_2\cdot\sigma_2, l_2\#r_2 \vee C_2\cdot\sigma_2, R_2, L_2, p_2)$ *be two variable disjoint rewrite steps where* $r_1\sigma_1 \prec_T l_1\sigma_1$*,* $(l_2\#r_2)\sigma_2|_p = l_1\sigma_1$ *for some position* p*. Let* $I_3 := (l_3\#r_3\cdot\sigma_3, l_3\#r_3 \vee C_3\cdot\sigma_3, I_1, I_2, p)$ *be the result of a rewrite inference. Then:*

1. $C_3\sigma_3 = (C_1 \vee C_2)\sigma_1\sigma_2$ *and* $l_3\#r_3\sigma_3 = (l_2\#r_2)\sigma_2[r_1\sigma_1]_p$*.*
2. $(l_3\#r_3)\sigma_3 \prec_T (l_2\#r_2)\sigma_2$
3. *If* $N \models (l_1 \approx r_1 \vee C_1) \wedge (l_2\#r_2 \vee C_2)$ *for some set of clauses* N*, then* $N \models l_3\#r_3 \vee C_3$

Now that we have defined rewrite inferences we can use them to define a *reduction chain application* and a *refutation*, which are sequences of rewrite steps. Intuitively speaking, a *reduction chain application* reduces a literal in a clause with literals in $conv(\Gamma)$ until it is irreducible. A *refutation* for a literal L that is *β-false* in Γ for a given β, is a sequence of rewrite steps with literals in Γ, L such that $\bot$ is inferred. Refutations for the literals of the conflict clause will be examined during conflict resolution by the rule Explore-Refutation.

Definition 15 (Reduction Chain). *Let* Γ *be a trail. A* reduction chain $\mathcal{P}$ *from* Γ *is a sequence of rewrite steps* $[I_1, ..., I_m]$ *such that for each* $I_i = (s_i\#t_i\cdot\sigma_i, s_i\#t_i \vee C_i\cdot\sigma_i, I_j, I_k, p_i)$ *either*

1. $s_i\#t_i^{n_i:s_i\#t_i\vee C_i\cdot\sigma}$ *is contained in* Γ *and* $I_j = I_k = p_i = \epsilon$ *or*
2. I_i *is the result of a rewriting inference from rewrite steps* I_j, I_k *out of* $[I_1, ..., I_m]$ *where* $j, k < i$*.*

Let $(l \# r)\delta^{o:l \# r \vee C\cdot\delta}$ *be an annotated ground literal. A* reduction chain application *from* Γ *to* $l \# r$ *is a reduction chain* $[I_1, ..., I_m]$ *from* $\Gamma, (l \# r)\delta^{o:l \# r \vee C\cdot\delta}$ *such that* $l\delta{\downarrow}_{\mathrm{conv}(\Gamma)} = s_m\sigma_m$ *and* $r\delta{\downarrow}_{\mathrm{conv}(\Gamma)} = t_m\sigma_m$*. We assume reduction chain applications to be minimal, i.e., if any rewrite step is removed from the sequence it is no longer a reduction chain application.*

Definition 16 (Refutation). *Let* Γ *be a trail and* $(l \# r)\delta^{o:l \# r \vee C\cdot\delta}$ *an annotated ground literal that is* β*-false in* Γ *for a given* β*. A* refutation $\mathcal{P}$ *from* Γ *and* $l \# r$ *is a reduction chain* $[I_1, ..., I_m]$ *from* $\Gamma, (l \# r)\delta^{o:l \# r \vee C\cdot\delta}$ *such that* $(s_m \# t_m)\sigma_m = s \not\approx s$ *for some* s*. We assume refutations to be minimal, i.e., if any rewrite step* I_k*,* $k < m$ *is removed from the refutation, it is no longer a refutation.*

3.1 The SCL(EQ) Inference Rules

We can now define the rules of our calculus based on the previous definitions. A *state* is a six-tuple $(\Gamma; N; U; \beta; k; D)$ similar to the SCL calculus, where Γ a sequence of annotated ground literals, N and U the sets of initial and learned clauses, β is a ground term such that for all $L \in \Gamma$ it holds $L \prec_T \beta$, k is the decision level, and D a status that is $\top$, $\bot$ or a closure $C\cdot\sigma$. Before we propagate or decide any literal, we make sure that it is irreducible in the current trail. Together with the design of $\prec_{\Gamma^*}$ this eventually enables rewriting as a simplification rule.

Propagate

$(\Gamma; N; U; \beta; k; \top) \Rightarrow_{\mathrm{SCL(EQ)}} \quad (\Gamma, s_m \# t_m\sigma_m^{k:(s_m \# t_m \vee C_m)\cdot\sigma_m}; N; U; \beta; k; \top)$

provided there is a $C \in (N \cup U)$, σ grounding for C, $C = C_0 \vee C_1 \vee L$, $\Gamma \models \neg C_0\sigma$, $C_1\sigma = L\sigma \vee ... \vee L\sigma$, $C_1 = L_1 \vee ... \vee L_n$, $\mu = mgu(L_1, ..., L_n, L)$ $L\sigma$ is *β-undefined* in Γ, $(C_0 \vee L)\mu\sigma \prec_T \beta$, σ is irreducible by *conv*(Γ), $[I_1, \ldots, I_m]$ is a reduction chain application from Γ to $L\sigma^{k:(L \vee C_0)\mu\cdot\sigma}$ where $I_m = (s_m \# t_m\cdot\sigma_m, s_m \# t_m \vee C_m\cdot\sigma_m, I_j, I_k, p_m)$.

Note that the definition of Propagate also includes the case where $L\sigma$ is irreducible by Γ. In this case $L = s_m \# t_m$ and $m = 1$. The rule Decide below, is similar to Propagate, except for the subclause C_0 which must be *β-undefined* or *β-true* in Γ, i.e., Propagate cannot be applied and the decision literal is annotated by a tautology.

Decide

$(\Gamma; N; U; \beta; k; \top) \Rightarrow_{\mathrm{SCL(EQ)}} \quad (\Gamma, s_m \# t_m\sigma_m^{k+1:(s_m \# t_m \vee comp(s_m \# t_m))\cdot\sigma_m}; N; U;$
$\beta; k+1; \top)$

provided there is a $C \in (N \cup U)$, σ grounding for C, $C = C_0 \vee L$, $C_0\sigma$ is *β-undefined* or *β-true* in Γ, $L\sigma$ is *β-undefined* in Γ, $(C_0 \vee L)\sigma \prec_T \beta$, σ is irreducible by *conv*(Γ), $[I_1, \ldots, I_m]$ is a reduction chain application from Γ to $L\sigma^{k+1:L \vee C_0\cdot\sigma}$ where $I_m = (s_m \# t_m\cdot\sigma_m, s_m \# t_m \vee C_m\cdot\sigma_m, I_j, I_k, p_m)$.

Conflict
$(\Gamma; N; U; \beta; k; \top) \Rightarrow_{\mathrm{SCL(EQ)}} \quad (\Gamma; N; U; \beta; k; D)$
provided there is a $D' \in (N \cup U)$, σ grounding for D', $D'\sigma$ is β-*false* in Γ, σ is irreducible by $conv(\Gamma)$, $D = \bot$ if $D'\sigma$ is of level 0 and $D = D' \cdot \sigma$ otherwise.

For the non-equational case, when a conflict clause is found by an SCL calculus [14,18], the complements of its first-order ground literals are contained in the trail. For equational literals this is not the case, in general. The proof showing D to be β-*false* with respect to Γ is a rewrite proof with respect to conv(Γ). This proof needs to be analyzed to eventually perform paramodulation steps on D or to replace D by a $\prec_{\Gamma^*}$ smaller β-*false* clause showing up in the proof.

Skip
$(\Gamma, K^{l:C\cdot\tau}, L^{k:C'\cdot\tau'}; N; U; \beta; k; D \cdot \sigma) \Rightarrow_{\mathrm{SCL(EQ)}} \quad (\Gamma, K^{l:C\cdot\tau}; N; U; \beta; l; D \cdot \sigma)$ if $D\sigma$ is β-*false* in $\Gamma, K^{l:C\cdot\tau}$.

The Explore-Refutation rule is the FOL with Equality counterpart to the resolve rule in CDCL or SCL. While in CDCL or SCL complementary literals of the conflict clause are present on the trail and can directly be used for resolution steps, this needs a generalization for FOL with Equality. Here, in general, we need to look at (rewriting) refutations of the conflict clause and pick an appropriate clause from the refutation as the next conflict clause.

Explore-Refutation
$(\Gamma, L; N; U; \beta; k; (D \lor s \# t) \cdot \sigma)) \Rightarrow_{\mathrm{SCL(EQ)}} \quad (\Gamma, L; N; U; \beta; k; (s_j \# t_j \lor C_j) \cdot \sigma_j)$
if $(s \# t)\sigma$ is strictly $\prec_{\Gamma^*}$ maximal in $(D \lor s \# t)\sigma$, L is the defining literal of $(s \# t)\sigma$, $[I_1, ..., I_m]$ is a refutation from Γ and $(s \# t)\sigma$, $I_j = (s_j \# t_j \cdot \sigma_j, (s_j \# t_j \lor C_j) \cdot \sigma_j, I_l, I_k, p_j)$, $1 \le j \le m$, $(s_j \# t_j \lor C_j)\sigma_j \prec_{\Gamma^*} (D \lor s \# t)\sigma$, $(s_j \# t_j \lor C_j)\sigma_j$ is β-*false* in Γ.

Factorize
$(\Gamma; N; U; \beta; k; (D \lor L \lor L') \cdot \sigma) \Rightarrow_{\mathrm{SCL(EQ)}} \quad (\Gamma; N; U; \beta; k; (D \lor L)\mu \cdot \sigma)$
provided $L\sigma = L'\sigma$, and $\mu = \mathrm{mgu}(L, L')$.

Equality-Resolution
$(\Gamma; N; U; \beta; k; (D \lor s \not\approx s') \cdot \sigma) \Rightarrow_{\mathrm{SCL(EQ)}} \quad (\Gamma; N; U; \beta; k; D\mu \cdot \sigma)$
provided $s\sigma = s'\sigma$, $\mu = mgu(s, s')$.

Backtrack
$(\Gamma, K, \Gamma'; N; U; \beta; k; (D \lor L) \cdot \sigma) \Rightarrow_{\mathrm{SCL(EQ)}} \quad (\Gamma; N; U \cup \{D \lor L\}; \beta; j - i; \top)$
provided $D\sigma$ is of level i' where $i' < k$, K is of level j and Γ, K the minimal trail subsequence such that there is a grounding substitution τ with $(D \lor L)\tau$ β-false in Γ, K but not in Γ; $i = 1$ if K is a decision literal and $i = 0$ otherwise.

Grow
$(\Gamma; N; U; \beta; k; \top) \Rightarrow_{\mathrm{SCL(EQ)}} \quad (\epsilon; N; U; \beta'; 0; \top)$
provided $\beta \prec_T \beta'$.

In addition to soundness and completeness of the SCL(EQ) rules their tractability in practice is an important property for a successful implementation. In particular, finding propagating literals or detecting a false clause under some grounding. It turns out that these operations are NP-complete, similar to first-order subsumption which has been shown to be tractable in practice.

Lemma 17. *Assume that all ground terms t with $t \prec_T \beta$ for any β are polynomial in the size of β. Then testing Propagate (Conflict) is NP-Complete, i.e., the problem of checking for a given clause C whether there exists a grounding substitution σ such that $C\sigma$ propagates (is false) is NP-Complete.*

Example 18 (SCL(EQ) vs. Superposition: Saturation). Consider the following clauses:

$$N := \{C_1 := c \approx d \vee D, C_2 := a \approx b \vee c \not\approx d, C_3 := f(a) \not\approx f(b) \vee g(c) \not\approx g(d)\}$$

where again we assume a KBO with all symbols having weight one, precedence $d \prec c \prec b \prec a \prec g \prec f$ and $\beta := f(f(g(a)))$. Suppose that we first decide $c \approx d$ and then propagate $a \approx b$: $\Gamma = [c \approx d^{1:c \approx d \vee c \not\approx d}, a \approx b^{1:C_2}]$. Now we have a conflict with C_3. Explore-Refutation applied to the conflict clause C_3 results in a paramodulation inference between C_3 and C_2. Another application of Equality-Resolution gives us the new conflict clause $C_4 := c \not\approx d \vee g(c) \not\approx g(d)$. Now we can Skip the last literal on the trail, which gives us $\Gamma = [c \approx d^{1:c \approx d \vee c \not\approx d}]$. Another application of the Explore-Refutation rule to C_4 using the decision justification clause followed by Equality-Resolution and Factorize gives us $C_5 := c \not\approx d$. Thus with SCL(EQ) the following clauses remain:

$$\begin{gathered} C_1' = D \qquad C_5 = c \not\approx d \\ C_3 = f(a) \not\approx f(b) \vee g(c) \not\approx g(d) \end{gathered}$$

where we derived C_1' out of C_1 by subsumption resolution [33] using C_5. Actually, subsumption resolution is compatible with the general redundancy notion of SCL(EQ), see Lemma 25. Now we consider the same example with superposition and the very same ordering (N_i is the clause set of the previous step and N_0 the initial clause set N).

$$\begin{aligned} N_0 &\Rightarrow_{Sup(C_2,C_3)} N_1 \cup \{C_4 := c \not\approx d \vee g(c) \not\approx g(d)\} \\ &\Rightarrow_{Sup(C_1,C_4)} N_2 \cup \{C_5 := c \not\approx d \vee D\} \Rightarrow_{Sup(C_1,C_5)} N_3 \cup \{C_6 := D\} \end{aligned}$$

Thus superposition ends up with the following clauses:

$$\begin{array}{ll} C_2 = a \approx b \vee c \not\approx d & C_3 = f(a) \not\approx f(b) \vee g(c) \not\approx g(d) \\ C_4 = c \not\approx d \vee g(c) \not\approx g(d) & C_6 = D \end{array}$$

The superposition calculus generates more and larger clauses.

Example 19 (SCL(EQ) vs. Superposition: Refutation). Suppose the following set of clauses: $N := \{C_1 := f(x) \not\approx a \vee f(x) \approx b, C_2 := f(f(y)) \approx y, C_3 := a \not\approx b\}$ where again we assume a KBO with all symbols having weight one, precedence

$b \prec a \prec f$ and $\beta := f(f(f(a)))$. A long refutation by the superposition calculus results in the following (N_i is the clause set of the previous step and N_0 the initial clause set N):

$$\begin{aligned} N_0 &\Rightarrow_{Sup(C_1,C_2)} N_1 \cup \{C_4 := y \not\approx a \vee f(f(y)) \approx b\} \\ &\Rightarrow_{Sup(C_1,C_4)} N_2 \cup \{C_5 := a \not\approx b \vee f(f(y)) \approx b \vee y \not\approx a\} \\ &\Rightarrow_{Sup(C_2,C_5)} N_3 \cup \{C_6 := a \not\approx b \vee b \approx y \vee y \not\approx a\} \\ &\Rightarrow_{Sup(C_2,C_4)} N_4 \cup \{C_7 := y \approx b \vee y \not\approx a\} \\ &\Rightarrow_{EqRes(C_7)} N_5 \cup \{C_8 := a \approx b\} \Rightarrow_{Sup(C_3,C_8)} N_6 \cup \{\bot\} \end{aligned}$$

The shortest refutation by the superposition calculus is as follows:

$$\begin{aligned} N_0 &\Rightarrow_{Sup(C_1,C_2)} N_1 \cup \{C_4 := y \not\approx a \vee f(f(y)) \approx b\} \\ &\Rightarrow_{Sup(C_2,C_4)} N_2 \cup \{C_5 := y \approx b \vee y \not\approx a\} \\ &\Rightarrow_{EqRes(C_5)} N_3 \cup \{C_6 := a \approx b\} \Rightarrow_{Sup(C_3,C_6)} N_4 \cup \{\bot\} \end{aligned}$$

In SCL(EQ) on the other hand we would always first propagate $a \not\approx b, f(f(a)) \approx a$ and $f(f(b)) \approx b$. As soon as $a \not\approx b$ and $f(f(a)) \approx a$ are propagated we have a conflict with $C_1\{x \rightarrow f(a)\}$. So suppose in the worst case we propagate:

$$\Gamma := [a \not\approx b^{0:a\not\approx b}, f(f(b)) \approx b^{0:(f(f(y))\approx y)\{y\rightarrow b\}}, f(f(a)) \approx a^{0:(f(f(y))\approx y)\{y\rightarrow a\}}]$$

Now we have a conflict with $C_1\{x \rightarrow f(a)\}$. Since there is no decision literal on the trail, *Conflict* rule immediately returns $\bot$ and we are done.

4 Soundness and Completeness

In this section we show soundness and refutational completeness of SCL(EQ) under the assumption of a regular run. We provide the definition of a regular run and show that for a regular run all learned clauses are non-redundant according to our trail induced ordering. We start with the definition of a sound state.

Definition 20. *A state $(\Gamma; N; U; \beta; k; D)$ is sound if the following conditions hold:*

1. *Γ is a consistent sequence of annotated literals,*
2. *for each decomposition $\Gamma = \Gamma_1, L\sigma^{i:(C\vee L)\cdot\sigma}, \Gamma_2$ where $L\sigma$ is a propagated literal, we have that $C\sigma$ is β-false in Γ_1, $L\sigma$ is β-undefined in Γ_1 and irreducible by $conv(\Gamma_1)$, $N \cup U \models (C \vee L)$ and $(C \vee L)\sigma \prec_T \beta$,*
3. *for each decomposition $\Gamma = \Gamma_1, L\sigma^{i:(L\vee comp(L))\cdot\sigma}, \Gamma_2$ where $L\sigma$ is a decision literal, we have that $L\sigma$ is β-undefined in Γ_1 and irreducible by $conv(\Gamma_1)$, $N \cup U \models (L \vee comp(L))$ and $(L \vee comp(L))\sigma \prec_T \beta$,*
4. *$N \models U$,*
5. *if $D = C \cdot \sigma$, then $C\sigma$ is β-false in Γ, $N \cup U \models C$,*

Lemma 21. *The initial state $(\epsilon; N; \emptyset; \beta; 0; \top)$ is sound.*

Definition 22. *A run is a sequence of applications of SCL(EQ) rules starting from the initial state.*

Theorem 23. *Assume a state $(\Gamma; N; U; \beta; k; D)$ resulting from a run. Then $(\Gamma; N; U; \beta; k; D)$ is sound.*

Next, we give the definition of a regular run. Intuitively speaking, in a regular run we are always allowed to do decisions except if

1. a literal can be propagated before the first decision and
2. the negation of a literal can be propagated.

To ensure non-redundant learning we enforce at least one application of Skip during conflict resolution except for the special case of a conflict after a decision.

Definition 24 (Regular Run). *A run is called* regular *if*

1. *the rules Conflict and Factorize have precedence over all other rules,*
2. *If $k = 0$ in a state $(\Gamma; N; U; \beta; k; D)$, then Propagate has precedence over Decide,*
3. *If an annotated literal $L^{k:C\cdot\sigma}$ could be added by an application of Propagate on Γ in a state $(\Gamma; N; U; \beta; k; D)$ and $C \in N \cup U$, then the annotated literal $comp(L)^{k+1:C'\cdot\sigma'}$ is not added by Decide on Γ,*
4. *during conflict resolution Skip is applied at least once, except if Conflict is applied immediately after an application of Decide.*
5. *if Conflict is applied immediately after an application of Decide, then Backtrack is only applied in a state $(\Gamma, L'; N; U; \beta; k; D\cdot \sigma)$ if $L\sigma = comp(L')$ for some $L \in D$.*

Now we show that any learned clause in a regular run is non-redundant according to our trail induced ordering.

Lemma 25 (Non-Redundant Clause Learning). *Let N be a clause set. The clauses learned during a regular run in SCL(EQ) are not redundant with respect to $\prec_{\Gamma^*}$ and $N \cup U$. For the trail only non-redundant clauses need to be considered.*

The proof of Lemma 25 is based on the fact that conflict resolution eventually produces a clause smaller then the original conflict clause with respect to $\prec_{\Gamma^*}$. All simplifications, e.g., contextual rewriting, as defined in [2,20,33,35–37], are therefore compatible with Lemma 25 and may be applied to the newly learned clause as long as they respect the induced trail ordering. In detail, let Γ be the trail before the application of rule Backtrack. The newly learned clause can be simplified according to the induced trail ordering $\prec_{\Gamma^*}$ as long as the simplified clause is smaller with respect to $\prec_{\Gamma^*}$.

Another important consequence of Lemma 25 is that newly learned clauses need not to be considered for redundancy. Furthermore, the SCL(EQ) calculus always terminates, Lemma 33, because there only finitely many non-redundant clauses with respect to a fixed β.

For dynamic redundancy, we have to consider the fact that the induced trail ordering changes. At this level, only redundancy criteria and simplifications that are compatible with *all* induced trail orderings may be applied. Due to the construction of the induced trail ordering, it is compatible with $\prec_T$ for unit clauses.

Lemma 26 (Unit Rewriting). *Assume a state* $(\Gamma; N; U; \beta; k; D)$ *resulting from a regular run where the current level* $k > 0$ *and a unit clause* $l \approx r \in N$*. Now assume a clause* $C \vee L[l']_p \in N$ *such that* $l' = l\mu$ *for some matcher* μ*. Now assume some arbitrary grounding substitutions* σ' *for* $C \vee L[l']_p$*,* σ *for* $l \approx r$ *such that* $l\sigma = l'\sigma'$ *and* $r\sigma \prec_T l\sigma$*. Then* $(C \vee L[r\mu\sigma\sigma']_p)\sigma' \prec_{\Gamma^*} (C \vee L[l']_p)\sigma'$*.*

In addition, any notion that is based on a literal subset relationship is also compatible with ordering changes. The standard example is subsumption.

Lemma 27. *Let* C, D *be two clauses. If there exists a substitution* σ *such that* $C\sigma \subset D$*, then* D *is redundant with respect to* C *and any* $\prec_{\Gamma^*}$*.*

The notion of redundancy, Definition 1, only supports a strict subset relation for Lemma 27, similar to the superposition calculus. However, the newly generated clauses of SCL(EQ) are the result of paramodulation inferences [28]. In a recent contribution to dynamic, abstract redundancy [32] it is shown that also the non-strict subset relation in Lemma 27, i.e., $C\sigma \subseteq D$, preserves completeness.

If all stuck states, see below Definition 28, with respect to a fixed β are visited before increasing β then this provides a simple dynamic fairness strategy.

When unit reduction or any other form of supported rewriting is applied to clauses smaller than the current β, it can be applied independently from the current trail. If, however, unit reduction is applied to clauses larger than the current β then the calculus must do a restart to its initial state, in particular the trail must be emptied, as for otherwise rewriting may result generating a conflict that did not exist with respect to the current trail before the rewriting. This is analogous to a restart in CDCL once a propositional unit clause is derived and used for simplification. More formally, we add the following new Restart rule to the calculus to reset the trail to its initial state after a unit reduction.

Restart
$(\Gamma; N; U; \beta; k; \top) \Rightarrow_{\mathrm{SCL(EQ)}} (\epsilon; N; U; \beta; 0; \top)$

Next we show refutation completeness of SCL(EQ). To achieve this we first give a definition of a stuck state. Then we show that stuck states only occur if all ground literals $L \prec_T \beta$ are β*-defined* in Γ and not during conflict resolution. Finally we show that conflict resolution will always result in an application of Backtrack. This allows us to show termination (without application of Grow) and refutational completeness.

Definition 28 (Stuck State). *A state* $(\Gamma; N; U; \beta; k; D)$ *is called* stuck *if* $D \neq \bot$ *and none of the rules of the calculus, except for Grow, is applicable.*

Lemma 29 (Form of Stuck States). *If a regular run (without rule Grow) ends in a stuck state* $(\Gamma; N; U; \beta; k; D)$*, then* $D = \top$ *and all ground literals* $L\sigma \prec_T \beta$*, where* $L \vee C \in (N \cup U)$ *are* β*-defined in* Γ*.*

Lemma 30. *Suppose a sound state $(\Gamma; N; U; \beta; k; D)$ resulting from a regular run where $D \notin \{\top, \bot\}$. If Backtrack is not applicable then any set of applications of Explore-Refutation, Skip, Factorize, Equality-Resolution will finally result in a sound state $(\Gamma'; N; U; \beta; k; D')$, where $D' \prec_{\Gamma^*} D$. Then Backtrack will be finally applicable.*

Corollary 31 (Satisfiable Clause Sets). *Let N be a satisfiable clause set. Then any regular run without rule Grow will end in a stuck state, for any β.*

Thus a stuck state can be seen as an indication for a satisfiable clause set. Of course, it remains to be investigated whether the clause set is actually satisfiable. Superposition is one of the strongest approaches to detect satisfiability and constitutes a decision procedure for many decidable first-order fragments [4,19]. Now given a stuck state and some specific ordering such as KBO, LPO, or some polynomial ordering [17], it is decidable whether the ordering can be instantiated from a stuck state such that Γ coincides with the superposition model operator on the ground terms smaller than β. In this case it can be effectively checked whether the clauses derived so far are actually saturated by the superposition calculus with respect to this specific ordering. In this sense, SCL(EQ) has the same power to decide satisfiability of first-order clause sets than superposition.

Definition 32. *A regular run terminates in a state $(\Gamma; N; U; \beta; k; D)$ if $D = \top$ and no rule is applicable, or $D = \bot$.*

Lemma 33. *Let N be a set of clauses and β be a ground term. Then any regular run that never uses Grow terminates.*

Lemma 34. *If a regular run reaches the state $(\Gamma; N; U; \beta; k; \bot)$ then N is unsatisfiable.*

Theorem 35 (Refutational Completeness). *Let N be an unsatisfiable clause set, and $\prec_T$ a desired term ordering. For any ground term β where $gnd_{\prec_T \beta}(N)$ is unsatisfiable, any regular SCL(EQ) run without rule Grow will terminate by deriving $\bot$.*

5 Discussion

We presented SCL(EQ), a new sound and complete calculus for reasoning in first-order logic with equality. We will now discuss some of its aspects and present ideas for future work beyond the scope of this paper.

The trail induced ordering, Definition 9, is the result of letting the calculus follow the logical structure of the clause set on the literal level and at the same time supporting rewriting at the term level. It can already be seen by examples on ground clauses over (in)equations over constants that this combination requires a layered approach as suggested by Definition 9, see [24].

In case the calculus runs into a stuck state, i.e., the current trail is a model for the set of considered ground instances, then the trail information can be effectively used for a guided continuation. For example, in order to use the trail to certify a model, the trail literals can be used to guide the design of a lifted ordering for the clauses with variables such that propagated trail literals are maximal in respective clauses. Then it could be checked by superposition, if the current clause is saturated by such an ordering. If this is not the case, then there must be a superposition inference larger than the current β, thus giving a hint on how to extend β. Another possibility is to try to extend the finite set of ground terms considered in a stuck state to the infinite set of all ground terms by building extended equivalence classes following patterns that ensure decidability of clause testing, similar to the ideas in [14]. If this fails, then again this information can be used to find an appropriate extension term β for rule Grow.

In contrast to superposition, SCL(EQ) does also inferences below variable level. Inferences in SCL(EQ) are guided by a false clause with respect to a partial model assumption represented by the trail. Due to this guidance and the different style of reasoning this does not result in an explosion in the number of possibly inferred clauses but also rather in the derivation of more general clauses, see [24].

Currently, the reasoning with solely positive equations is done on and with respect to the trail. It is well-known that also inferences from this type of reasoning can be used to speed up the overall reasoning process. The SCL(EQ) calculus already provides all information for such a type of reasoning, because it computes the justification clauses for trail reasoning via rewriting inferences. By an assessment of the quality of these clauses, e.g., their reduction potential with respect to trail literals, they could also be added, independently from resolving a conflict.

The trail reasoning is currently defined with respect to rewriting. It could also be performed by congruence closure [26].

Towards an implementation, the aspect of how to find interesting ground decision or propagation literals for the trail can be treated similar to CDCL [11, 21,25,29]. A simple heuristic may be used from the start, like counting the number of instance relationships of some ground literal with respect to the clause set, but later on a bonus system can focus the search towards the structure of the clause sets. Ground literals involved in a conflict or the process of learning a new clause get a bonus or preference. The regular strategy requires the propagation of all ground unit clauses smaller than β. For an implementation a propagation of the (explicit and implicit) unit clauses with variables to the trail will be a better choice. This complicates the implementation of refutation proofs and rewriting (congruence closure), but because every reasoning is layered by a ground term β this can still be efficiently done.

Acknowledgments. This work was partly funded by DFG grant 389792660 as part of TRR 248, see https://perspicuous-computing.science. We thank the anonymous reviewers and Martin Desharnais for their thorough reading, detailed comments, and corrections.

References

1. Alagi, G., Weidenbach, C.: NRCL - a model building approach to the Bernays-Schönfinkel fragment. In: Lutz, C., Ranise, S. (eds.) FroCoS 2015. LNCS (LNAI), vol. 9322, pp. 69–84. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24246-0_5
2. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. **4**(3), 217–247 (1994)
3. Bachmair, L., Ganzinger, H., Voronkov, A.: Elimination of equality via transformation with ordering constraints. In: Kirchner, C., Kirchner, H. (eds.) CADE 1998. LNCS, vol. 1421, pp. 175–190. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054259
4. Bachmair, L., Ganzinger, H., Waldmann, U.: Superposition with simplification as a decision procedure for the monadic class with equality. In: Gottlob, G., Leitsch, A., Mundici, D. (eds.) KGC 1993. LNCS, vol. 713, pp. 83–96. Springer, Heidelberg (1993). https://doi.org/10.1007/BFb0022557
5. Baumgartner, P.: Hyper tableau — the next generation. In: de Swart, H. (ed.) TABLEAUX 1998. LNCS (LNAI), vol. 1397, pp. 60–76. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-69778-0_14
6. Baumgartner, P., Fuchs, A., Tinelli, C.: Lemma learning in the model evolution calculus. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 572–586. Springer, Heidelberg (2006). https://doi.org/10.1007/11916277_39
7. Baumgartner, P., Furbach, U., Pelzer, B.: Hyper tableaux with equality. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 492–507. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73595-3_36
8. Baumgartner, P., Pelzer, B., Tinelli, C.: Model evolution with equality-revised and implemented. J. Symb. Comput. **47**(9), 1011–1045 (2012)
9. Baumgartner, P., Tinelli, C.: The model evolution calculus with equality. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 392–408. Springer, Heidelberg (2005). https://doi.org/10.1007/11532231_29
10. Baumgartner, P., Waldmann, U.: Superposition and model evolution combined. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 17–34. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_2
11. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)
12. Bonacina, M.P., Furbach, U., Sofronie-Stokkermans, V.: On First-Order Model-Based Reasoning. In: Martí-Oliet, N., Ölveczky, P.C., Talcott, C. (eds.) Logic, Rewriting, and Concurrency. LNCS, vol. 9200, pp. 181–204. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23165-5_8
13. Bonacina, M.P., Plaisted, D.A.: SGGS theorem proving: an exposition. In: Schulz, S., Moura, L.D., Konev, B. (eds.) PAAR-2014. 4th Workshop on Practical Aspects of Automated Reasoning. EPiC Series in Computing, vol. 31, pp. 25–38. EasyChair (2015)
14. Bromberger, M., Fiori, A., Weidenbach, C.: Deciding the Bernays-Schoenfinkel fragment over bounded difference constraints by simple clause learning over theories. In: Henglein, F., Shoham, S., Vizel, Y. (eds.) VMCAI 2021. LNCS, vol. 12597, pp. 511–533. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67067-2_23

15. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
16. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM (JACM) **7**(3), 201–215 (1960)
17. Dershowitz, N., Plaisted, D.A.: Rewriting. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, chap. 9, pp. 535–610. Elsevier (2001)
18. Fiori, A., Weidenbach, C.: SCL clause learning from simple models. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 233–249. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_14
19. Ganzinger, H., de Nivelle, H.: A superposition decision procedure for the guarded fragment with equality. In: LICS, pp. 295–304 (1999)
20. Gleiss, B., Kovács, L., Rath, J.: Subsumption demodulation in first-order theorem proving. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12166, pp. 297–315. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51074-9_17
21. Bayardo, R.J., Schrag, R.: Using CSP look-back techniques to solve exceptionally hard SAT instances. In: Freuder, E.C. (ed.) CP 1996. LNCS, vol. 1118, pp. 46–60. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61551-2_65
22. Korovin, K.: Inst-Gen – a modular approach to instantiation-based automated reasoning. In: Voronkov, A., Weidenbach, C. (eds.) Programming Logics. LNCS, vol. 7797, pp. 239–270. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37651-1_10
23. Korovin, K., Sticksel, C.: iProver-Eq: an instantiation-based theorem prover with equality. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS (LNAI), vol. 6173, pp. 196–202. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14203-1_17
24. Leidinger, H., Weidenbach, C.: SCL(EQ): SCL for first-order logic with equality (2022). arXiv: 2205.08297
25. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the Design Automation Conference, pp. 530–535. ACM (2001)
26. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM **27**(2), 356–364 (1980)
27. Plaisted, D.A., Zhu, Y.: Ordered semantic hyper-linking. J. Autom. Reason. **25**(3), 167–217 (2000)
28. Robinson, G., Wos, L.: Paramodulation and theorem-proving in first-order theories with equality. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence 4, pp. 135–150 (1969)
29. Silva, J.P.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: International Conference on Computer Aided Design, ICCAD, pp. 220–227. IEEE Computer Society Press (1996)
30. Sutcliffe, G.: The TPTP problem library and associated infrastructure - from CNF to th0, TPTP v6.4.0. J. Autom. Reasoning **59**(4), 483–502 (2017)
31. Teucke, A.: An approximation and refinement approach to first-order automated reasoning. Doctoral thesis, Saarland University (2018)
32. Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12166, pp. 316–334. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51074-9_18

33. Weidenbach, C.: Combining superposition, sorts and splitting. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. 2, chap. 27, pp. 1965–2012. Elsevier (2001)
34. Weidenbach, C.: Automated reasoning building blocks. In: Meyer, R., Platzer, A., Wehrheim, H. (eds.) Correct System Design. LNCS, vol. 9360, pp. 172–188. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23506-6_12
35. Weidenbach, C., Wischnewski, P.: Contextual rewriting in SPASS. In: PAAR/ESHOL. CEUR Workshop Proceedings, vol. 373, pp. 115–124. Australien, Sydney (2008)
36. Weidenbach, C., Wischnewski, P.: Subterm contextual rewriting. AI Commun. **23**(2–3), 97–109 (2010)
37. Wischnewski, P.: Effcient Reasoning Procedures for Complex First-Order Theories. Ph.D. thesis, Saarland University, November 2012