Model Evolution with Equality – Revised and Implemented

Peter Baumgartner[1]

NICTA and The Australian National University, Canberra, Australia

Björn Pelzer

Institute for Computer Science, Universität Koblenz-Landau, Germany

Cesare Tinelli

Department of Computer Science, The University of Iowa, USA

Abstract

In many theorem proving applications, a proper treatment of equational theories or equality is mandatory. In this paper we show how to integrate a modern treatment of equality in the Model Evolution calculus ($\mathcal{ME}$), a first-order version of the propositional DPLL procedure. The new calculus, $\mathcal{ME}_\mathrm{E}$, is a proper extension of the $\mathcal{ME}$ calculus without equality. Like $\mathcal{ME}$ it maintains an explicit *candidate model*, which is searched for by DPLL-style splitting. For equational reasoning $\mathcal{ME}_\mathrm{E}$ uses an adapted version of the superposition inference rule, where equations used for superposition are drawn (only) from the candidate model. The calculus also features a generic, semantically justified simplification rule which covers many simplification techniques known from superposition-style theorem proving. Our main theoretical result is the correctness of the $\mathcal{ME}_\mathrm{E}$ calculus in the presence of very general redundancy elimination criteria. We also describe our implementation of the calculus, the *E-Darwin* system, and we report on practical experiments with it on the TPTP problems library.

Keywords: Automated Theorem Proving, Instance-Based Methods

1. Introduction

The Model Evolution ($\mathcal{ME}$) Calculus [7, 9] is a refutational calculus for clause logic developed as a first-order extension of the propositional DPLL procedure [14]. Compared to its predecessor, the FDPLL calculus [4], $\mathcal{ME}$ lifts to first-order logic without equality not just the core of the DPLL procedure, the splitting rule, but also its simplification rules, which are crucial for its practical feasibility.

Email addresses: `Peter.Baumgartner@nicta.com.au` (Peter Baumgartner), `bpelzer@uni-koblenz.de` (Björn Pelzer), `tinelli@cs.uiowa.edu` (Cesare Tinelli)

[1]NICTA is funded by the Australian Government's *Backing Australia's Ability* initiative.

Our implementation of $\mathcal{ME}$, the Darwin theorem prover [6], performs well in some domains but, unsurprisingly, not on problems with equality which it can treat only with the explicit addition of equality axioms. This is a serious shortcoming in practice because the majority of automated reasoning applications work in logics with equality. In [8] we addressed the problem by proposing an extension of $\mathcal{ME}$ with dedicated inference rules for equality reasoning. These rules were centered around a version of the *ordered paramodulation* inference rule adapted to $\mathcal{ME}$. This paper presents an extensively revised and improved version of that calculus, called $\mathcal{ME}_{\mathrm{E}}$, that relies more heavily on notions and techniques originally developed for the Superposition calculus [2]. As a result, $\mathcal{ME}_{\mathrm{E}}$ features more powerful redundancy criteria, and, by means of *selection functions*, removes some non-determinism from the calculus in [8]. We prove the soundness and completeness of $\mathcal{ME}_{\mathrm{E}}$ in full detail and discuss an initial implementation in the *E-Darwin* theorem prover, largely based on Darwin. The completeness proof is obtained as an extension of the $\mathcal{ME}$ completeness proof by adapting techniques from the Bachmair-Ganzinger framework developed for proving the completeness of superposition [2, 26, e.g.]. The underlying model construction technique allows us to justify a rather general simplification rule on semantic grounds. The simplification rule is based on a general redundancy criterion that covers many simplification techniques known from superposition-style theorem proving.

These adaptations are non-trivial because of the rather different layout of the two calculi. While superposition maintains clause sets as its main data structure, $\mathcal{ME}_{\mathrm{E}}$ works with a set of literals, which we call a *context*, and a set of (constrained) clauses. $\mathcal{ME}_{\mathrm{E}}$ has, correspondingly, two kinds of inference rules, one for modifying contexts, and one for deriving new (constrained) clauses, with the latter consisting mostly of unit-superposition style inference rule. Since there is no counterpart to contexts in superposition calculi, the inference rules on contexts are specific to our calculus, as are the redundancy criteria we present.

Related Work. Like $\mathcal{ME}$, the $\mathcal{ME}_{\mathrm{E}}$ calculus is related to *instance based methods (IMs)*, a family of calculi and proof procedures for first-order (clausal) logic that share the principle of carrying out proof search by maintaining a set of instances of input clauses and analyzing it for satisfiability until completion [5, 21]. Most IMs are based on resolving *pairs* of complementary literals (connections) from *two* clauses in order to determine these instances. In contrast, $\mathcal{ME}$'s main derivation rule (splitting) is based on evaluating *all* literals of a *single* clause against a current candidate model in order to determine an instance. See [9] for a more detailed discussion of $\mathcal{ME}$ in relation to IMs, which also applies to $\mathcal{ME}_{\mathrm{E}}$ when equality is not an issue.

There are only a few IMs that include inference rules for equality reasoning. Ordered Semantic Hyperlinking (OSHL) by Plaisted and Zhu [29] uses rewriting and narrowing (paramodulation) with unit equations, but requires some other mechanism such as Brand's transformation to handle equations that appear in nonunit clauses.

To our knowledge there are only two IMs that have been extended with dedicated equality inference rules for full equational clausal logic. One of them is described by Ganzinger and Korovin [18] as an extension of the earlier IM by the same authors [17]. It is conceptually rather different from $\mathcal{ME}_{\mathrm{E}}$: the main inference rule for equational reasoning requires, as a subtask, the refutation of a set of unit clauses (which is obtained by picking literals from the current clause set). More closely related to $\mathcal{ME}_\mathrm{E}$ is an IM based on disconnection tableaux by Letz and Stenz, a successor of Billon's disconnection method [12].[2] Letz and Stenz discuss various ways of integrating equality reasoning in disconnection tableaux [22], including a variant based on ordered paramodulation. These paramodulation inferences combine (equational) branch literals and clauses into new clauses, where the equation used for paramodulation is added in negated form (as a condition) to the paramodulant. Our main equality inference rules work similarly, and in this sense the calculus of [22] and $\mathcal{ME}_\mathrm{E}$ are conceptually related. But $\mathcal{ME}_\mathrm{E}$ features more powerful and general concepts of redundancy detection and elimination.

Finally, $\mathcal{ME}$ has been combined with the superposition calculus in a single framework, including equality inference rules by Baumgartner and Waldmann [10]. On the one hand, by this very combination, their calculus is more general than $\mathcal{ME}_\mathrm{E}$. On the other hand, $\mathcal{ME}_\mathrm{E}$ has some features that are not present in that calculus e.g., universal variables, which enable the optional derivation rules Assert and Compact and lead to a more powerful redundancy criterion. Another difference lies in the way constrained clauses are ordered, which enables additional redundancy criteria that would be non-trivial to integrate in the calculus in [10].

Paper organization. We start with an informal explanation of the main ideas behind the $\mathcal{ME}_\mathrm{E}$ calculus in Section 2. After some technical preliminaries in Section 3, we provide a more formal treatment of the main data structures of the calculus in Section 4, where we describe *contexts* and their associated interpretations, and Section 5, where we describe *constrained clauses* and inference rules for performing equality reasoning on them. We then present the $\mathcal{ME}_\mathrm{E}$ calculus in Section 6, and discuss its correctness in Section 7. In Section 8 we discuss an initial implementation of the calculus together with comparative experimental results. We conclude in Section 9, suggesting a few directions for further research. Detailed proofs of all the results in the paper can be found in the appendix.

2. Main Ideas

Similarly to the $\mathcal{ME}$ calculus, $\mathcal{ME}_\mathrm{E}$ is informally best described with an eye to the propositional DPLL procedure it extends. DPLL can be viewed as a procedure that searches the space of possible interpretations for a given clause set until it finds one that satisfies the clause set, if it exists. This can be done by keeping a current candidate model and *repairing* it as needed until it satisfies every input clause. The repairs are done incrementally by changing the truth value of one (propositional) clause literal at a time, and involve a non-deterministic guess, a "split", on whether the value of a selected literal should be changed or kept as it is. The number of guesses is reduced thanks to a constraint propagation process, usually referred to as "unit propagation", that is able to deduce deterministically the value of some input literals.

Both $\mathcal{ME}$ and $\mathcal{ME}_\mathrm{E}$ lift this idea to first-order logic by (*i*) maintaining a *first-order* candidate model, (*ii*) deriving new clauses based on that model, (*iii*) identifying *in-*

[2] Even in that early paper a paramodulation-like inference rule was considered, albeit a rather weak one.

stances of derived clauses that are falsified by the model, and (*iv*) repairing the model incrementally until it satisfies all of those instances. The main difference between the two calculi is that $\mathcal{ME}$ works with Herbrand models while $\mathcal{ME}_{\mathrm{E}}$ works with *equational* models, or *E-interpretations*, that is, Herbrand interpretations in which the equality symbol is the only predicate symbol and always denotes a congruence relation. The current E-interpretation is constructed from the current *context*, a finite set of oriented equational literals processed by the calculus. As we will show, each context Λ determines a (convergent) ground rewrite system R_Λ. The E-interpretation associated with the context is the congruence closure $R_\Lambda^\star$ of R_Λ.

Context literals can be of two types: *universal* and *parametric*. As we will see later, the difference between the two lies in how they constrain the possible modifications of a context and, as a consequence, the possible repairs to its induced E-interpretation. As far as determining the induced E-interpretation, however, the two types of context literals are interchangeable.

For convenience, let us say that a context Λ satisfies/falsifies a literal or a clause if its associated E-interpretation $R_\Lambda^\star$ does so. As mentioned above, during the course of a derivation, the current context Λ is modified if it falsifies a current, derived clause. Such a *repair* involves in essence two steps: (*i*) identifying an instance of a current clause that is falsified by the current context, and (*ii*) adding a new literal to the context so that the new context satisfies that instance. The first step is achieved by deriving new clauses by applying (unit) superposition-style inference rules to literals in the context and clauses in the current clause set until an empty clause is derived.

Each derived clause carries with it a *constraint* recording all the equalities and disequalities of Λ involved in the inferences used to derive it. In particular, a derived empty clause D has the form $\Box \cdot \Gamma$ where $\Box$ is the empty disjunction and Γ, a set of equational literals, is the constraint. A (ground) constraint is satisfied by a context Λ if all of its equalities are contained in R_Λ, the ground rewrite system induced by Λ, and none of its disequalities is. The clause D is then falsified if for some substitution σ, $\Gamma\sigma$ is satisfied in this sense. In that case, the calculus addresses the problem by adding to Λ, if possible, (a variant of) the complement $\overline{K}$ of some literal K in Γ.[3] This typically ensures that the new context satisfies some (possibly, all) ground instances of $\overline{K}$ and, as a consequence, some ground instances of D. If some instances of D remain falsified, a new empty clause reflecting that will be derivable later.

Note that the rules of $\mathcal{ME}_{\mathrm{E}}$ do not actually work with R_Λ but with the literals in Λ and their instances obtained by unification with clause literals. The rewrite system R_Λ, a conceptual construction we use to explain the calculus and prove its completeness, is only approximated at the calculus level. In practice, this may sometimes lead to unnecessary inferences. That, however, is not a problem from a correctness perspective as long as all inferences theoretically necessary for completeness are carried out.

The choice to repair the context with the complement of a constraint literal K of $\Box \cdot \Gamma$ is *don't-know* non-deterministic since it just produces a local repair that may fail to lead to a model for the input clause set. Hence, it must be paired in the calculus

[3]Adding $\overline{K}$ is not possible if it is *contradictory* with Λ (see later) because then $\Lambda \cup \{\overline{K}\}$ denotes no E-interpretation.

with a corresponding complementary action. When K shares variables with some other literal in Γ, this action consists in adding (a parametric variant of) K instead of $\overline{K}$, *a la* DPLL. Otherwise, it consists in replacing $\Box \cdot \Gamma$ with $\Box \cdot (\Gamma \setminus K)$. While neither of these alternative complementary actions repairs the context, each one constitutes progress in the derivation since it effectively forces the calculus to look at other constraint literals in $\Box \cdot \Gamma$ for the repair. When no literals in Γ can be used, the context is unrepairable and backtracking to a previous choice point is necessary. With a fair derivation strategy, the calculus guarantees that whenever no more clauses are derivable (modulo redundancy) and the context has been repaired, perhaps in the limit, to satisfy all current constrained empty clauses, its associated E-interpretation is a model of the input clause set.

For an example of how a derivation might proceed consider the following input clause set Φ_0:

$$Q(x, a) \approx \mathbf{t} \vee P(f(x)) \approx \mathbf{t} \tag{1}$$
$$g(x) \approx x \tag{2}$$
$$f(g(x)) \approx x \vee h(x) \approx x \tag{3}$$

where x is a variable and $\mathbf{t}$ a constant.

The initial context, Λ_0, consists of a pseudo-literal of the form $\neg v$. Its associated E-interpretation, as we will become clear later, is the identity relation over the set $T(\Sigma)$ of all ground terms, where Σ is the signature of Φ_0. Note that such an E-interpretation falsifies each clause in Φ_0. In fact, it falsifies every ground instance of each clause. This situation is reflected in the calculus by the fact that a unit resolution-like rule applies to the pseudo-literal $\neg v$ and a clause in Φ_0. Repeated applications of such a rule may lead to the derivation of the constrained clauses below.[4]

$$Q(x, a) \approx \mathbf{t} \quad \cdot \quad P(f(x)) \not\approx \mathbf{t} \tag{4}$$
$$\Box \quad \cdot \quad Q(x, a) \not\approx \mathbf{t}, P(f(x)) \not\approx \mathbf{t} \tag{5}$$
$$\Box \quad \cdot \quad g(x) \not\approx x \tag{6}$$
$$f(g(x)) \approx x \quad \cdot \quad h(x) \not\approx x \tag{7}$$

The constraint of each derived clause can be understood as a set of *preconditions* for the clause. Roughly speaking, it represents a set of ground facts that held in the current interpretation at the time the clause was derived. If later one of these ground facts does not hold anymore because of changes to the context, the corresponding ground instances of the constrained clause are, in essence, removed from consideration. For instance, the constraint of clause (4) states that when the clause was derived (from clause (1)) all ground instances of $P(f(x)) \not\approx \mathbf{t}$ were satisfied in the current context. The derivation of empty constrained clauses, like (5) and (6) above, indicates that the current context needs to be repaired. For clause (6) the only possible repair is to add the universal literal $g(x) \approx x$ to Λ_0. In this case, the alternative, namely replacing $\Box \cdot g(x) \not\approx x$ with $\Box \cdot \emptyset$, not only fails to repair the context, it also makes it unrepairable—because of the addition of the unconstrained empty clause $\Box \cdot \emptyset$ to the clause set.

[4]Later in the paper we will use $\rightarrow$ instead of $\approx$ in constraint and context literals to stress that they are oriented (dis)equalities. But we can overlook this technicality for now.

With the new context $\Lambda_1 = \{\neg v,\ g(x) \approx x\}$, the induced rewrite system R_{Λ_1} is a certain subset of $E = \{g(t) \approx t \mid t \in T(\Sigma)\}$ having the same congruence closure as E.[5] The associated interpretation (the congruence closure of R_{Λ_1}) now satisfies every ground instance of clause (6). A superposition rule applied to the new context literal $g(x) \approx x$ and to clause (3) derives the clause

$$f(x) \approx x \vee h(x) \approx x \quad\cdot\quad g(x) \approx x \tag{8}$$

The fact that $g(x) \approx x$ in Λ_1 is universal guarantees that any later extensions of Λ_1 will satisfy every ground instance of the constraint literal $g(x) \approx x$. This allows the simplification of (8) to the unconstrained clause

$$f(x) \approx x \vee h(x) \approx x \tag{9}$$

From that clause, the calculus can derive

$$\Box \quad\cdot\quad f(x) \not\approx x,\ h(x) \not\approx x \tag{10}$$

because the current context Λ_1 satisfies every ground instance of $f(x) \not\approx x$ and of $h(x) \not\approx x$. Since clause (10) is empty, a repair is necessary. That can be done with the addition of the parametric literal $f(u) \approx u$, resulting in the new context $\Lambda_2 = \{\neg v,\ g(x) \approx x,\ f(u) \approx u\}$. From this context and clause (4) it is possible to derive, in order, the clauses

$$P(x) \approx \mathbf{t} \quad\cdot\quad Q(x,a) \not\approx \mathbf{t},\ f(x) \not\approx x \tag{11}$$

$$\Box \quad\cdot\quad Q(x,a) \not\approx \mathbf{t},\ f(x) \not\approx x,\ P(x) \not\approx \mathbf{t} \tag{12}$$

Clause (12) requires another repair for the context, for instance into $\Lambda_3 = \{\neg v,\ g(x) \approx x,\ f(u) \approx u, P(v) \approx \mathbf{t}\}$ with $P(v) \approx \mathbf{t}$ parametric. At this point, no new clauses can be derived. Furthermore, no more repairs are needed, or in fact possible, since the latest one took care of both clause (12) and clause (5)—the latter of which was also pending. At this point the calculus stops. The E-interpretation associated to Λ_3 is the congruence closure of

$$\{g(t) \approx t \mid t \in T(\Sigma)\} \cup \{f(t) \approx t \mid t \in T(\Sigma)\} \cup \{P(t) \approx \mathbf{t} \mid t \in T(\Sigma)\}\,.$$

This interpretation is indeed a model of the initial clause set $\{Q(x,a) \approx \mathbf{t} \vee P(f(x)) \approx \mathbf{t},\ g(x) \approx x,\ f(g(x)) \approx x \vee h(x) \approx x\}$.

If the clause set also contained the clause $C = P(x) \not\approx \mathbf{t}$, say, the context Λ_3 would not only still fail to satisfy the clause set, for falsifying C; it would also be unrepairable with respect to C. In that case, the addition of $P(v) \not\approx \mathbf{t}$ to Λ_2 instead of $P(v) \approx \mathbf{t}$ would have to be considered, leading then to a repair with $Q(v,a) \approx \mathbf{t}$ from clause (12).

Because of the splitting rules the $\mathcal{ME}_\mathrm{E}$ calculus formally generates derivation *trees*. What we described in the example above is essentially one branch of a derivation tree

[5] R_{Λ_1} is a proper subset of E. For instance, if it contains the rewrite rule $g(a) \approx a$ it will not contain the superfluous rule $g(g(a)) \approx g(a)$.

for the given clause set that ends with a model. As discussed in that example, some derivation tree branches may end with a context that still falsifies some input clause but is unrepairable. The generation of a *closed tree*, a tree all of whose branches end with an unrepairable context, is a proof that the initial clause set is unsatisfiable in first-order logic with equality. This makes the $\mathcal{ME}_\mathrm{E}$ calculus refutationally sound for that logic. The calculus is also refutationally complete, that is, guaranteed to generate a closed derivation tree starting from any unsatisfiable clause set when used with a *fair* derivation strategy. As expected, $\mathcal{ME}_\mathrm{E}$ is not terminating for all input clause sets. Specifically, for a satisfiable clause set it can generate a derivation tree all of whose non-closed branches need infinitely-many repairs to produce a model. We will show however that, similarly to $\mathcal{ME}$, it is terminating for input sets that are clause forms of sets of $\exists^*\forall^*$-formulas with equality, a well known decidable fragment of first-order logic [11, 30].

3. Formal Preliminaries

Most of the notions and notation we use in this paper are the standard ones in the field (see, e.g., [32]). We report here only notable differences and additions.

3.1. Terms and Substitutions

We will use two disjoint, infinite sets of variables: a set X of *universal* variables, which we will refer to just as variables, and another set V, which we will always refer to as *parameters*. We will use u and v to denote elements of V and x and y to denote elements of X. We fix a signature Σ throughout the paper, which is left implicit when we speak of terms, formulas, and interpretations. We assume that Σ contains at least one constant symbol. If t is a term we denote by $\mathcal{V}ar(t)$ the set of t's variables and by $\mathcal{P}ar(t)$ the set of t's parameters. A term t is *parameter-free* iff $\mathcal{P}ar(t) = \emptyset$, it is *variable-free* iff $\mathcal{V}ar(t) = \emptyset$, and it is *ground* iff $\mathcal{V}ar(t) = \mathcal{P}ar(t) = \emptyset$. A substitution ρ is a *renaming on* $W \subseteq (V \cup X)$ iff its restriction to W is a bijection of W onto itself; ρ is simply a *renaming* if it is a renaming on $V \cup X$. A substitution σ is *p-preserving* (short for parameter preserving) if it is a renaming on V. If s and t are two terms, we write $s \gtrsim t$, iff there is a substitution σ such that $s\sigma = t$.[6] We say that s is *a variant of* t, and write $s \sim t$, iff $s \gtrsim t$ and $t \gtrsim s$ or, equivalently, iff there is a renaming ρ such that $s\rho = t$. We write $s \gnsim t$ if $s \gtrsim t$ but $s \nsim t$. We write $s \geq t$ and say that t is *a p-instance of* s iff there is a p-preserving substitution σ such that $s\sigma = t$. We say that s is *a p-variant of* t, and write $s \simeq t$, iff $s \geq t$ and $t \geq s$; equivalently, iff there is a p-preserving renaming ρ such that $s\rho = t$. The expression $s[t]_p$ denotes, as usual, the term obtained from term s by replacing the subterm of s at position p by term t.

3.2. Clauses

Essentially without loss of generality, we assume that the signature Σ contains only one predicate symbol, $\approx$ (equality) although it may contain a finite set of function

[6]Note that many authors would write $s \lesssim t$ in this case.

symbols of given arity; 0-ary function symbols are called *constants*. Because equality is the only predicate symbol, an *atom* is always a (positive) equation $s \approx t$, where s and t are terms, which is identified with the multiset $\{s, t\}$. Consequently, the equations $s \approx t$ and $t \approx s$ are the same. A *literal* is an atom, a *positive literal*, or the negation of an atom, a *negative literal*. Negative literals are generally written $s \not\approx t$ instead of $\neg(s \approx t)$.

We call a literal *universal* iff it is parameter-free. In particular, ground literals are universal. *Non-universal*, or *parametric*, literals then contain at least one parameter.

In the examples below we often write non-equational literals like $P(t_1, \ldots, t_n)$ as an abbreviation for the equational literal $P(t_1, \ldots, t_n) \approx \mathbf{t}$, where $\mathbf{t}$ is a distinguished constant. We denote literals by the letters K and L. We write $\overline{L}$ to denote the complement of a literal L, i.e. $\overline{A} = \neg A$ and $\overline{\neg A} = A$, for any atom A. We denote clauses by the letters C and D, and the empty clause by $\Box$.

A *clause* is a parameter-free multiset of literals $\{L_1, \ldots, L_n\}$, generally written as a disjunction $L_1 \vee \cdots \vee L_n$. We write $L \vee C$ to denote the clause $\{L\} \cup C$. The empty clause is written as $\Box$.

All the notions on substitutions above are extended from terms to atoms, literals and clauses in the obvious way.

3.3. Orderings and Rewrite Systems

A *reduction ordering* is a well-founded partial ordering $\succ$ on terms that is closed under context, i.e., $s \succ s'$ implies $t[s]_p \succ t[s']_p$ for all terms t and positions p, and liftable, i.e., $s \succ t$ implies $s\sigma \succ t\sigma$ for every substitution σ.

We assume as given a reduction ordering $\succ$ that is total on ground terms and that the constant $\mathbf{t}$ is the smallest ground term in this ordering. As usual, we will use $\succeq$ to denote the reflexive closure of $\succ$.

A *(rewrite) rule* is an expression of the form $l \rightarrow r$ where l and r are terms. A *rewrite system* is a set of rewrite rules. We say that a rewrite system R is *ordered by* $\succ$ iff $l \succ r$, for every rule $l \rightarrow r \in R$. *We will consider only ground rewrite systems ordered by* $\succ$. A term t is *reducible by* $l \rightarrow r$ iff $t = t[l]_p$ for some position p, and t is *reducible wrt.* R if it is reducible by some rule in R. *Irreducible* means "not reducible". A rewrite system R is *left-reduced* (*fully reduced*) iff for every rule $l \rightarrow r \in R$, l is (l and r are) irreducible wrt $R \setminus \{l \rightarrow r\}$. In other words, in a fully reduced rewrite system no rule is reducible by another rule, neither its left hand side *nor its right hand side*. A rewrite system is *convergent* iff it is confluent and terminating.

3.4. Interpretations

A *(Herbrand) interpretation I* is a set of ground Σ-atoms—those meant to be true in the interpretation. Satisfiability of first-order formulas in a Herbrand interpretation is defined as usual. In particular, if F is a literal, a clause or a clause (set), we say that *I satisfies F*, or is *a model of F*, and write $I \models F$, if I satisfies every ground instance of (every element of) F. An *E-interpretation* is an interpretation that is also a congruence relation on the ground terms. We say that F is *E-(un)satisfiable* if it is satisfied by (no) E-interpretations. We say that *F E-entails F'*, written $F \models_{\mathrm{E}} F'$, iff every E-interpretation that satisfies F also satisfies F'. Since this is the only notion of entailment considered in the paper, we will often write just $F \models F'$. If I is an interpretation, we denote by $I^\star$ the congruence closure of I, i.e., the smallest congruence relation on all ground terms that includes I, which is an E-interpretation. The above notions are applied to ground rewrite systems instead of interpretations by considering their rewrite rules as equations. For instance, we write $R^\star \models F$ and mean $\{l \approx r \mid l \to r \in R\}^\star \models F$. It is well-known that every left-reduced (and hence also every fully reduced) ground ordered rewrite system R is convergent and that any ground equation $s \approx t$ is E-satisfied by R (i.e., $R^\star \models s \approx t$) if and only if s and t have the same (unique) normal form wrt. R [1].

4. Contexts

The Model Evolution calculus as presented in [7] works with sequents of the form $\Lambda \vdash \Phi$, where Λ is a finite set of literals possibly with variables or with parameters, called a context, and Φ is a finite set of clauses possibly with variables. As in [7], we will consider for simplicity only contexts whose literals do not contain both parameters and variables.[7] In [8] a context Λ consisted of equational literals, and we treated Λ as if it contained the symmetric versions of each of its equations, negated or not. We depart from that by working with *rewrite literals* now, which is simpler and more practical. More formally, when l and r are terms, a *rewrite literal* is an expression of the form $l \to r$ or its negation $\neg(l \to r)$, the latter generally written as $l \not\to r$. By treating $\to$ as a predicate symbol, all operations defined on equational literals apply to rewrite literals as well. For instance, $\overline{l \to r} = l \not\to r$ and $\overline{l \not\to r} = l \to r$. If clear from the context, we use the term "literal" to refer to equational literals as introduced earlier or to rewrite literals.

Definition 4.1 *A* context *is a set of the form* $\{\neg v\} \cup S$ *where* $v \in V$ *and* S *is a finite set of rewrite literals, each of which is parameter-free or variable-free.*

For brevity, we will omit writing the *pseudo-literal* $\neg v$ in examples of contexts. For a context Λ, we will denote with $\Lambda_\geq$ the set of all p-instances of the literals in Λ.

A literal L is *contradictory with* a context Λ iff $\overline{L\sigma} \in \Lambda_\geq$ for some p-preserving substitution σ. A context Λ is *contradictory* iff it contains a literal that is contradictory with Λ. We will work only with non-contradictory contexts. Note that membership in $\Lambda_\geq$ is (efficiently) decidable for being reducible to a syntactic unification problem over Λ. The same is true for contradiction with Λ.

Example 4.2 If $\Lambda = \{f(u) \to a, f(x) \not\to x\}$ then $f(v) \to a, f(a) \not\to a \in \Lambda_\geq$ but $f(x) \to a \notin \Lambda_\geq$, Furthermore, $f(v) \not\to a$ and $f(a) \to a$ are contradictory with Λ[8] while $f(a) \not\to a$ and $a \to f(a)$ are not. □

[7] In [9] we have shown how this limitation can be overcome.

[8] The complement of the former is a p-instance of $f(u) \to a$, the complement of the latter is a p-instance of $f(x) \not\to x$.

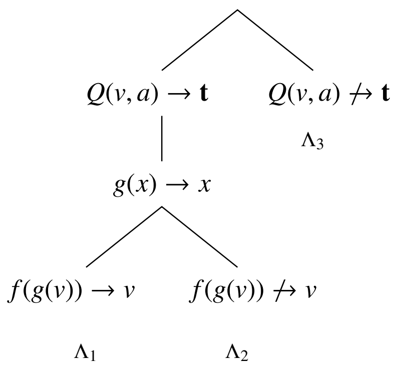

Figure 1: Contexts derivable by the calculus from the clauses (1), (2) and (3) in Section 2. The pseudo-literal $\neg v$, which goes into the root node, is not written.

We will also regularly return to our main example in Section 2 and use it to illustrate key ideas. When we speak of clauses (1), (2) and (3) we mean those given there.

Example 4.3 (Main Example) The $\mathcal{ME}_{\mathrm{E}}$ calculus is based, essentially, on growing contexts and using context literals for paramodulation and resolution type inferences. How the contexts grow exactly will be explained below. Intuitively, unit clauses like (2) can be directly inserted into a context as universal literals, whereas literals that share variables with other clause literals, like $Q(x,a) \approx \mathbf{t}$ in (1) are turned into parametric variants first and are subject to complementary splitting. As a result, the context structure in Figure 1 could evolve in a concrete derivation. □

Thanks to the notions introduced next, contexts can be used as finite denotations of (certain) Herbrand interpretations.

Definition 4.4 (Productivity [7]) *Let L be a rewrite literal and Λ a context. A rewrite literal K* produces *L in Λ iff (i) $K \gtrsim L$*[9] *and (ii) there is no $K' \in \Lambda_{\geq}$ such that $K \gnsim \overline{K'} \gtrsim L$. The context Λ* produces *L iff some $K \in \Lambda$ produces L in Λ. We say that K* strongly produces *L in Λ iff K produces L in Λ but Λ does not produce $\overline{L}$, and that Λ* strongly produces *L iff some $K \in \Lambda$ strongly produces L in Λ.*

For instance, the context Λ in Example 4.2 produces $f(a) \not\twoheadrightarrow b$, as per $\neg v \in \Lambda$, and Λ produces $f(b) \to a$ but it does not produce $f(a) \to a$. Instead Λ produces $f(a) \not\twoheadrightarrow a$ as per $f(a) \not\twoheadrightarrow a \in \Lambda_{\geq}$.

Example 4.5 (Main Example) The ground literals produced by the context Λ_1 in Figure 1 are $Q(a,a) \to \mathbf{t}$, $Q(f(a),a) \to \mathbf{t}$, $Q(g(a),a) \to \mathbf{t}, \ldots$, $g(a) \to a$, $g(f(a)) \to f(a)$,

[9]In [7] condition (i) is replaced by the stronger condition "K is an msg of L in Λ", where K is a *most specific generalization (msg) of L in* Λ iff $K \gtrsim L$ and there is no $K' \in \Lambda$ such that $K \gnsim K' \gtrsim L$. Working with $K \gtrsim L$ alone is somewhat simpler and achieves the same for all purposes.

$g(g(a)) \to g(a), \ldots, f(g(a)) \to a, f(g(f(a))) \to f(a), f(g(g(a))) \to g(a), \ldots$. Similarly for Λ_2, which produces the literals in last segment negated. $\square$

The ground literals produced by a context serve as an *approximation* of a canonical interpretation for the given clause set, as explained next.

4.1. Model Construction

In this section, we show how a context Λ induces a canonical E-interpretation represented by a ground rewrite system R_Λ. It is this "model construction" that, ultimately, justifies defining powerful notions of redundancy for $\mathcal{ME}_\mathrm{E}$ and enables proving its completeness.[10] It is a key component to understand the working of the calculus and this is why we introduce it here already.

The general technique for defining R_Λ is borrowed from the completeness proof of the Superposition calculus [2, 25], but adapted to our needs. One difference is that $\mathcal{ME}_\mathrm{E}$ requires the construction of a fully reduced rewrite system, whereas for Superposition a left-reduced rewrite system is sufficient.[11]

Let Λ be a non-contradictory context. By identifying a rewrite rule $l \to r$ with the multiset $\{l, r\}$ and using the multiset extension of the term ordering $\succ$, again denoted by $\succ$ itself, we define by well-founded induction, and with respect to Λ, sets of ground rewrite rules ϵ_K and R_K, for every ground rewrite rule K. Assume that ϵ_L has already been defined for all such L with $K \succ L$ and let $R_K = \bigcup_{K \succ L} \epsilon_L$. Then, ϵ_K is generally defined as follows.

$$\epsilon_{l \to r} = \begin{cases} \{l \to r\} & \text{if } \Lambda \text{ produces } l \to r,\, l \succ r, \text{ and } l \text{ and } r \text{ are irreducible wrt. } R_{l \to r} \\ \emptyset & \text{otherwise} \end{cases}$$

When $\epsilon_{l \to r} = \{l \to r\}$, we say that Λ *generates* $l \to r$. Finally, the rewrite system associated with Λ is the set

$$R_\Lambda = \bigcup_{l \to r} \epsilon_{l \to r} \ .$$

Example 4.6 Let $\Lambda = \{a \to x, b \to c, a \not\to c\}$. With $a \succ b \succ c$ the associated rewrite system R_Λ is $\{b \to c\}$. To see why, observe that the candidate rule $a \to c$ is not included in R_Λ, as Λ does not produce $a \to c$, and that the other candidate $a \to b$, although produced in Λ, is reducible by the smaller rule $b \to c$. Had we chosen to omit in the definition of $\epsilon_{l \to r}$ the condition "r is irreducible wrt. $R_{l \to r}$" [12] the construction would have given $R_\Lambda = \{a \to b, b \to c\}$. This leads to the undesirable situation that a constrained clause, say, $a \not\approx c \cdot \emptyset$ is falsified by $R_\Lambda^\star$, the E-interpretation induced by R_Λ. But the calculus cannot modify Λ to revert this situation, and to detect the inconsistency (ordered) paramodulation into variables would be needed. $\square$

[10] The proof of soundness relies on different, and simpler, arguments.

[11] Because of this difference, reflected also in the preconditions of the Sup-Pos rule defined later, one could argue that our approach is more similar to ordered paramodulation [26, e.g.] than to Superposition.

[12] This condition is absent in the model construction in the superposition calculus. Its presence explains why paramodulation into smaller sides of positive split literals in clauses is necessary.

Example 4.7 (Main Example) Consider Figure 1 again. For $\Lambda_1 = \{Q(v,a) \to \mathbf{t}, g(x) \to x,\ f(g(v)) \to v\}$ the induced rewrite system, which must be a subset of its produced ground literals (see Example 4.3), is

$$\begin{aligned} R_{\Lambda_1} = \{&Q(a,a) \to \mathbf{t},\ Q(f(a),a) \to \mathbf{t},\ Q(f(f(a)),a) \to \mathbf{t}, \ldots, \\ &g(a) \to a,\ g(f(a)) \to f(a),\ g(f(f(a))) \to f(f(a)), \ldots\}\ . \end{aligned}$$

Notice that, e.g., $g(g(a)) \to g(a)$, although produced by Λ_1 cannot be in R_{Λ_1} because $R_{g(g(a))\to g(a)}$ contains $g(a) \to a$, which reduces $g(g(a)) \to g(a)$. It is straightforward to check that every ground instance of the literal $f(g(v)) \to v$ in Λ_1, which are all produced by Λ_1 are all reducible by smaller rules. In other words, $f(g(v)) \to v$ does not generate a single rule. The same applies to Λ_2, as with $f(g(v)) \not\to v \in \Lambda_2$ no such instance is even produced. □

It is not difficult to see that R_Λ is a fully reduced rewrite system. Since $\succ$ is a well-founded ordering, R_Λ is convergent. It follows from standard results in term rewriting that the satisfaction of ground literals $s \approx t$ (or $s \not\approx t$) in $R_\Lambda^\star$ can be decided by checking if the normal forms of s and t wrt. R_Λ are the same.

Lemma 4.8 *Let l and r be ground terms with $l \succ r$.*

(i) If $l \to r \in R_\Lambda$ then Λ produces $l \to r$.

(ii) If l and r are irreducible wrt. R_Λ then Λ strongly produces $l \not\to r$.

The lemma above establishes an important relationship between ground rewrite literals produced by Λ and the rewrite system R_Λ. It connects membership in R_Λ, a theoretical construction, with productivity, a syntactical notion that can be readily used in inference rules. More technically, say that $l \to r$ is a *rule candidate* if $l \succ r$ and both l and r are irreducible wrt. $R_{l\to r}$. Then either the rule candidate $l \to r$ is in R_Λ and by Lemma 4.8-(*i*) it is produced, or $l \to r$ is not in R_Λ. In the latter case, as l and r are irreducible wrt. $R_{l\to r}$, it follows that l and r are irreducible wrt. R_Λ.[13] But then, Lemma 4.8 gives us that Λ (strongly) produces $l \not\to r$. In other words, productivity *approximates* membership of (possibly negated) rules in R_Λ and it is *precise* for membership of (possibly negated) rule candidates in R_Λ.

5. Constrained Clauses

A *constraint* is a finite multiset of pairs $\Gamma = \{K_1' \rhd K_1, \ldots, K_m' \rhd K_m\}$, where $m \geq 0$ and for $i = 1, \ldots, m$, K_i' is a rewrite literal, and K_i is a parameter-free rewrite literal with the same sign, but is not of the form $x \to t$, where x is a variable and t is a term. We will consider only constraints where each K_i' is taken from a context Λ, and we call K_i' a *context literal (of* Γ*)* and K_i a *constraint literal (of* Γ*)*. Intuitively, a pair $K_i' \rhd K_i$ is satisfied if K_i' produces K_i in Λ, among other properties, as explained below. For

[13]The only rules that could reduce l or r would have to be smaller or equal than $l \to r$. The former case is excluded explicitly, and the latter case is impossible because $l \to r \notin R_\Lambda$.

economy of notation we let Γ stand also for the multiset of its constraint literals (only), and thus leave the context literals implicit. This allows us to write, e.g., $l \to r \in \Gamma$ instead of the more verbose "$K' \rhd l \to r \in \Gamma$, for some K'". Whether Γ is taken this way or not will always be clear from context. The application of a substitution σ to Γ, written as $\Gamma\sigma$, means to apply σ to each constraint literal of Γ (only), i.e., $\Gamma\sigma = \{K'_1 \rhd K_1\sigma, \ldots, K'_m \rhd K_m\sigma\}$. A constraint is *ground* iff each of its constraint literals K_i is ground. A substitution γ is a *grounding substitution for* Γ iff $\Gamma\gamma$ is ground.

Let $C = L_1 \vee \cdots \vee L_n$ be a clause and Γ a constraint. The expression $C \cdot \Gamma$ is a *constrained clause (with constraint* Γ*)*. With the convention explained above, and dropping the set braces, we will often write $C \cdot K_1, \ldots, K_m$ instead of $C \cdot K'_1 \rhd K_1, \ldots, K'_m \rhd K_m$, thus leaving the constraint's context literals implicit. The notation $C \cdot \Gamma, K' \rhd K$ means $C \cdot \Gamma \cup \{K' \rhd K\}$. Sometimes we write $C \cdot \Gamma, K$ instead $C \cdot \Gamma, K' \rhd K$, again leaving K' implicit. The application of a substitution σ to $C \cdot \Gamma$, written as $(C \cdot \Gamma)\sigma$, yields the constrained clause $(C\sigma \cdot \Gamma\sigma)$. A constrained clause $C \cdot \Gamma$ is *ground* iff both C and Γ are ground. When $C \cdot \Gamma$ is a constrained clause and γ is a grounding substitution for $C \cdot \Gamma$, i.e., a substitution with domain $\mathcal{V}ar(C) \cup \mathcal{V}ar(\Gamma)$ and whose range consists of ground terms, we call the pair $(C \cdot \Gamma, \gamma)$ a *ground closure (of* $C \cdot \Gamma$*)*, or just *closure (of* $C \cdot \Gamma$*)*. For a set of constrained clauses Φ, Φ^{gr} denotes the set of all ground closures of all constrained clauses in Φ.

Constrained clauses are compared by associating to every constraint clause $C \cdot \Gamma$ the multiset of multisets of terms

$$\begin{aligned} &\{\{l,l,l,l,r,r,r,r\} \mid l \not\approx r \in C\} \\ \cup\, &\{\{l,l,l,r,r,r\} \mid l \approx r \in C\} \\ \cup\, &\{\{l,l,r,r\} \mid l \not\to r \in \Gamma\} \\ \cup\, &\{\{l,r\} \mid l \to r \in \Gamma\} \end{aligned}$$

and then using the two-fold multiset extension of the term ordering $\succ$ on the associated multisets. Again we use the symbol $\succ$ to denote this (strict) ordering on constrained clauses. It follows from well-known results (see, e.g., [32]) that $\succ$ is a well-founded ordering and is total on ground constrained clauses. For instance, if $a \succ b$ then

$$a \not\approx b \cdot \emptyset \;\succ\; b \not\approx b \cdot a \to b \;\succ\; \square \cdot a \to b \;\; .$$

All inference rules on constrained clauses defined in Section 5.1 are order-decreasing at the ground level. While different orderings exist that also satisfy this property, the particular definition above also enables a certain redundancy concept based on reducible constraints, explained below.

To obtain a total and well-founded ordering on ground closures, we combine the ordering on constrained clauses lexicographically with an arbitrary ordering $\succ'$ on ground closures that is total (up to variable renaming),[14] that is, we define $(C \cdot \Gamma, \gamma) \succ (C' \cdot \Gamma', \gamma')$ iff $(C \cdot \Gamma)\gamma \succ (C' \cdot \Gamma')\gamma'$ or $(C \cdot \Gamma)\gamma = (C' \cdot \Gamma')\gamma'$ and $(C \cdot \Gamma, \gamma) \succ' (C' \cdot \Gamma', \gamma')$.

[14]Since for every ground closure $(C \cdot \Gamma, \gamma)$ there are only finitely many closures $(C' \cdot \Gamma', \gamma')$ such that $(C \cdot \Gamma)\gamma = (C' \cdot \Gamma')\gamma'$, the lexicographic combination is well-founded even if $\succ'$ is not well-founded.

To reason about the soundness of the $\mathcal{ME}_{\mathrm{E}}$ calculus it is enough to treat each constrained clause

$$C \cdot l_1 \to r_1, \ldots, l_k \to r_k, l_{k+1} \not\to r_{k+1}, \ldots, l_n \not\to r_n$$

as the ordinary clause

$$C \vee l_1 \not\approx r_1 \vee \cdots \vee l_k \not\approx r_k \vee l_{k+1} \approx r_{k+1} \vee \cdots \vee l_n \approx r_n \ .$$

From a completeness perspective, however, a different reading of constrained clauses is appropriate. The clause part C of a (ground) constrained clause $C \cdot \Gamma$ is evaluated in an E-interpretation I, whereas the literals in Γ are evaluated wrt. a context Λ in terms of productivity. The definitions below make this precise.

A ground constraint Γ is *ordered (wrt. $\succ$)* iff $l \succ r$ for every $l \to r \in \Gamma$ and every $l \not\to r \in \Gamma$. A non-ground constraint Γ is *ordered (wrt. $\succ$)* iff $\Gamma\gamma$ is ordered, for some grounding substitution γ for Γ.

Definition 5.1 (Satisfaction of Constraints) *Let Λ be a context, Γ a constraint and γ a grounding substitution for Γ. We say that Λ* satisfies the pair (Γ, γ), *and write $\Lambda \models (\Gamma, \gamma)$, iff*

(i) *$\Gamma\gamma$ is ordered, and*

(ii) *for every $K' \rhd K \in \Gamma$, K' produces both K and $K\gamma$ in Λ.*

We say that Γ is *satisfiable in* Λ, or that Λ *satisfies* Γ, written as $\Lambda \models \exists\,\Gamma$, iff $\Lambda \models (\Gamma, \gamma)$ for some γ.

Definition 5.2 (Satisfaction of Constrained Clauses) *Let Λ be a context, I an E-interpretation, and $(C \cdot \Gamma, \gamma)$ a ground closure. We say that the pair (Λ, I)* satisfies $(C \cdot \Gamma, \gamma)$ *and write $(\Lambda, I) \models (C \cdot \Gamma, \gamma)$ iff $\Lambda \not\models (\Gamma, \gamma)$ or $I \models C\gamma$.*

The pair (Λ, I) *satisfies* a possibly non-ground constrained clause (set) F, written as $(\Lambda, I) \models F$ iff (Λ, I) satisfies all ground closures of (all elements in) F. For a set of constrained clauses Φ and $C \cdot \Gamma$ a (possibly non-ground) constrained clause, we say that Φ *entails* $C \cdot \Gamma$, and write $\Phi \models C \cdot \Gamma$ iff for every (Λ, I) we have $(\Lambda, I) \not\models \Phi$ or $(\Lambda, I) \models (C \cdot \Gamma)$.

The definitions above also apply to pairs (Λ, R), where R is a rewrite system, by implicitly taking $(\Lambda, R^\star)$. In the main applications of Definition 5.2 such a rewrite system R will be determined by the model construction in Section 4.1 above.

Example 5.3 Let

$$\begin{array}{lcllcl}
\Lambda & = & \{f(x) \to x, f(c) \not\to c\} & \gamma_a & = & \{x \mapsto a\}, \\
R & = & \{f(a) \to a, f(b) \to b\}, & \gamma_b & = & \{x \mapsto b\}, \\
C \cdot \Gamma & = & f(f(a)) \approx x \cdot f(x) \to x \rhd f(x) \to x & \gamma_c & = & \{x \mapsto c\}\ .
\end{array}$$

Suppose that $a \succ b \succ c$ and observe that $\Gamma\gamma_a$, $\Gamma\gamma_b$ and $\Gamma\gamma_c$ are ordered. Then, $\Lambda \models (\Gamma, \gamma_a)$, as $f(x) \to x$ produces both $f(x) \to x$ and $f(a) \to a$ in Λ, and so we need to check $R^\star \models f(f(a)) \approx a$, which is the case, to conclude $(\Lambda, R) \models (C \cdot \Gamma, \gamma_a)$. As $\Lambda \models (\Gamma, \gamma_b)$ but $R^\star \not\models f(f(a)) \approx b$ we have $(\Lambda, R) \not\models (C \cdot \Gamma, \gamma_b)$. Finally, $f(x) \to x$ does not produce $f(c) \to c$ in Λ, and with $\Lambda \not\models (\Gamma, \gamma_c)$ it follows $(\Lambda, R) \models (C \cdot \Gamma, \gamma_c)$ □

Example 5.4 (Main Example) Clause (1) in Figure 1 can be written as the constrained clause $C \cdot \Gamma = Q(x, a) \approx \mathbf{t} \vee P(f(x)) \approx \mathbf{t} \cdot \emptyset$ by attaching an empty constraint, and analogously for (2) and (3). Let $\gamma = \{x \mapsto g(g(f(a)))\}$. As $\Gamma = \emptyset$, to check that $(\Lambda_1, R_{\Lambda_1})$ satisfies $(C \cdot \Gamma, \gamma)$ we only need to check that $R_{\Lambda_1} \models Q(g(g(f(a))), a) \approx \mathbf{t} \vee P(f(g(g(f(a))))) \approx \mathbf{t}$. Indeed, by using the second part of the rewrite rules in R_{Λ_1}, as shown in Example 4.7, we obtain first $Q(f(a), a) \approx \mathbf{t} \vee P(f(f(a))) \approx \mathbf{t}$ and then $\mathbf{t} \approx \mathbf{t} \vee P(f(f(a))) \approx \mathbf{t}$, which is trivially satisfied in R_{Λ_1}. In fact, this holds for every ground closure of $C \cdot \Gamma$, and so $C \cdot \Gamma$ is already satisfied in R_{Λ_1}. As a consequence, the calculus does not need to work on modifying Λ_1 to become a model of $C \cdot \Gamma$.

By contrast, R_{Λ_3} does not satisfy any ground closure of $C \cdot \Gamma$. Notice that $R_{\Lambda_3} = \emptyset$, and hence for every ground closure $(C \cdot \Gamma, \gamma)$ the clause $C\gamma$ is irreducible wrt. R_{Λ_3}. By Lemma 4.8 then Λ_3 produces the complement of every literal in $C\gamma$, a situation the calculus can detect and make progress on by splitting.

Now let $C \cdot \Gamma = f(g(x)) \approx x \vee h(x) \approx x \cdot \emptyset$ be the constrained clause version of clause (2). By taking, e.g., $\gamma = \{x \mapsto a\}$ one sees that $(\Lambda_1, R_{\Lambda_1})$ does not satisfy $(C \cdot \Gamma, \gamma)$. More explicitly, $R_{\Lambda_1} \not\models f(g(a)) \approx a \vee h(a) \approx a$ because the normal form $f(a) \approx a \vee h(a) \approx a$ of $C\gamma$ wrt. R_{Λ_1} does not contain a trivial equation of the form $t \approx t$. On the other hand, the calculus can reflect this normal form computation at the first-order level by a superposition step from the rewrite literal $g(x) \to x$ taken from Λ_1 into clause (2), which gives the constrained clause

$$C' \cdot \Gamma' = f(x) \approx x \vee h(x) \approx x \cdot g(x) \to x \rhd g(x) \to x \ . \tag{13}$$

Observe that $(C' \cdot \Gamma', \{x \mapsto a\})$ is still not satisfied by $(\Lambda_1, R_{\Lambda_1})$ as $\Lambda \models (\{g(x) \to x \rhd g(x) \to x\}, \{x \mapsto a\})$ but $R_{\Lambda_1} \not\models f(a) \approx a \vee h(a) \approx a$. Progress has been made though, as $(C' \cdot \Gamma', \{x \mapsto a\})$ is smaller wrt. $>$ than $(C \cdot \Gamma, \{x \mapsto a\})$. As will be explained below in Example 6.2, the calculus will make sure, by extending Λ_1 with, say, $f(v) \to v$, that $f(a) \to a$ will go into the induced rewrite system, so that the considered ground closure will be satisfied afterwards. □

In summary, the main idea of the calculus is to (*i*) *identify, at the first-order level, ground closures whose (instantiated) clause part is normalized with respect to the rewrite system induced by the current context, and,* (*ii*) *modify the context as needed so that the new context and its induced rewrite system satisfy one of the previously falsified clauses.*

It follows from the above that the (instantiated) constraints of these ground closures are always satisfied in terms of Definition 5.5. This leads immediately to the definition of relevant closures introduced below, which are the *only* ones the calculus needs to consider satisfying.

Definition 5.5 (Satisfaction of Constraints wrt. Rewrite Systems) *Let* Γ *be a ground constraint. We say that* R_Λ satisfies Γ, *and write* $R_\Lambda \models \Gamma$, *iff*

(i) $l \to r \in R_\Lambda$ *for all* $l \to r \in \Gamma$, *and*
(ii) l *and* r *are irreducible wrt.* R_Λ *for all* $l \nrightarrow r \in \Gamma$.

Note 5.6 The previous definition can be understood to evaluate a set Γ of (possibly negated) rule candidates wrt. a set of rewrite rules: R_Λ satisfies a ground constraint Γ if and only if $l \to r$ is a rule candidate, for every $l \to r \in \Gamma$ or $l \not\to r \in \Gamma$, and in the former case $l \to r$ is included in R_Λ because Λ produces $l \to r$, and in the latter case $l \to r$ is not included in R_Λ *only* because Λ does not produce $l \to r$. As a consequence, if Λ is modified such that it produces $l \to r$, it will no longer satisfy Γ. Example 6.2 below demonstrates the relevance of these properties. $\square$

By combining Definition 5.5 and Lemma 4.8, we immediately conclude that Λ produces every rewrite literal in a ground constraint Γ if $R_\Lambda \models \Gamma$. The relevance of this result is that for relevant closures the syntactic notion of productivity can be used to identify relevant constrained clauses $C \cdot \Gamma$ that are falsified by (Λ, R_Λ), as introduced next.

Definition 5.7 (Relevant Closure wrt. Λ) *Let Λ be a context and $(C \cdot \Gamma, \gamma)$ a ground closure. We say that $(C \cdot \Gamma, \gamma)$ is a* relevant (ground) closure wrt. Λ *iff $R_\Lambda \models \Gamma\gamma$.*

Note that, by definition, all ground closures of a clause with an empty constraint are relevant. Also note that for $C \cdot \Gamma$ to have relevant closures it does not have to belong to a specific clause set.

Example 5.8 Assume constants a, b and c with $a \succ b$. Let $\Lambda = \{P(x) \to \mathbf{t}, a \to b, P(b) \not\to \mathbf{t}\}$, which results in $R_\Lambda = \{P(c) \to \mathbf{t}, a \to b\}$. (Notice that $P(a) \to \mathbf{t} \notin R_\Lambda$ as $P(a) \to \mathbf{t}$ is reducible by the smaller rule $a \to b$, and $P(b) \to \mathbf{t} \notin R_\Lambda$ as Λ does not produce $P(b) \to \mathbf{t}$.) If $C \cdot \Gamma = x \approx c \cdot P(x) \to \mathbf{t} \rhd P(x) \to \mathbf{t}$, the substitution $\gamma = \{x \mapsto a\}$ gives a non-relevant ground closure $(C \cdot \Gamma, \gamma)$, as $P(a) \to \mathbf{t} \notin R_\Lambda$, and likewise for $\gamma = \{x \mapsto b\}$. With $\gamma = \{x \mapsto c\}$ the ground closure $(C \cdot \Gamma, \gamma)$ is relevant, and it follows $(\Lambda, R_\Lambda) \models (C \cdot \Gamma, \gamma)$, as R_Λ trivially satisfies $C\gamma = c \approx c$ (cf. Definition 5.2). $\square$

5.1. Inference Rules on Constrained Clauses

In the following, we introduce a number of superposition-style inference rules on rewrite literals and constrained clauses that will be used for defining the derivation rules of $\mathcal{ME}_\mathrm{E}$. The inference rules assume a *selection function* that maps a constrained clause $C \cdot \Gamma$ with non-empty C to a non-empty subset of the literals in C, the *selected literals (in $C \cdot \Gamma$)*.[15]

$$\text{Ref} \ \frac{s \not\approx t \vee C \cdot \Gamma}{(C \cdot \Gamma)\sigma}$$

where (*i*) σ is a most general unifier (mgu) of s and t, (*ii*) $s \not\approx t$ is selected in $s \not\approx t \vee C \cdot \Gamma$, and (*iii*) $\Gamma\sigma$ is ordered.

The next three rules combine a rewrite literal and a constrained clause. Within the $\mathcal{ME}_\mathrm{E}$ calculus, which operates on sequents consisting of a context and constrained

[15]Selection functions on *negative* literals are a well-known device in the superposition calculus. In $\mathcal{ME}_\mathrm{E}$, we can be more liberal and select *any* literal(s) in C, which results in more stringent selection functions.

clause set, the rewrite literal will come from the current context and the clause from the current constrained clause set.

$$\textsf{Sup-Neg}\ \frac{l \to r \qquad s[l']_p \not\approx t \vee C \cdot \Gamma}{(s[r]_p \not\approx t \vee C \cdot \Gamma, l \to r \rhd l \to r)\sigma}$$

where (*i*) $l \to r$ is a rewrite literal, (*ii*) σ is a mgu of l and l', (*iii*) l' is not a variable, (*iv*) $r\sigma \not\succeq l\sigma$, (*v*) $t\sigma \not\succeq s\sigma$, (*vi*) $s \not\approx t$ is selected in $s \not\approx t \vee C \cdot \Gamma$, and (*vii*) $\Gamma\sigma$ is ordered.

Recall that substitutions are not applied to a constraint's context literals. The conclusion of the above inference rule could equally be written as $(s[r]_p \not\approx t \vee C)\sigma \cdot \Gamma\sigma, l \to r \rhd (l \to r)\sigma$.

$$\textsf{Sup-Pos}\ \frac{l \to r \qquad s[l']_p \approx t \vee C \cdot \Gamma}{(s[r]_p \approx t \vee C \cdot \Gamma, l \to r \rhd l \to r)\sigma}$$

where (*i*) $l \to r$ is a rewrite literal, (*ii*) σ is a mgu of l and l', (*iii*) l' is not a variable, (*iv*) $r\sigma \not\succeq l\sigma$, (*v*) $s \approx t$ is selected in $s \approx t \vee C \cdot \Gamma$, and (*vi*) $\Gamma\sigma$ is ordered.[16]

Example 5.9 (Main Example) Sup-Pos can be applied to $g(x) \to x$ (from Λ_1 in Figure 1) and clause (3) in the following way:

$$\textsf{Sup-Pos}\ \frac{g(x) \to x \qquad f(g(x)) \approx x \vee h(x) \approx x \cdot \emptyset}{f(x) \approx x \vee h(x) \approx x \cdot g(x) \to x} \tag{14}$$

We refer back to Example 5.4 which explained the purpose of such superposition steps. Notice the conclusion of this inference is the clause (13) given there. □

The rules Sup-Pos and Sup-Neg are the only ones that create new positive rewrite literals $(l \to r)\sigma$ in the constraint part. It is possible in both cases that the left premise is of the form $x \to t$ ($x \to t$ might stem from a clause literal $x \approx t$), but because l' is not a variable, $l\sigma$ is not a variable either. Thus, adding to $\Gamma\sigma$ the rewrite literal $(l \to r)\sigma$ preserves the property of being a constraint (recall that constraints cannot contain rewrite literals of the form $x \to t$). As an easy inductive consequence all expressions $C \cdot \Gamma$ derivable by the calculus are indeed constrained clauses.

$$\textsf{Neg-Res}\ \frac{\neg A \qquad s \approx t \vee C \cdot \Gamma}{(C \cdot \Gamma, \neg A \rhd s \not\to t)\sigma}$$

where (*i*) $\neg A$ is the pseudo literal $\neg v$ or a negative rewrite literal $l \not\to r$, (*ii*) σ is a mgu of A and $s \to t$, (*iii*) $t\sigma \not\succeq s\sigma$, (*iv*) $s \approx t$ is selected in $s \approx t \vee C \cdot \Gamma$, and (*v*) $\Gamma\sigma$ is ordered.

[16]Technically, and differently from Sup-Neg, since it can apply to both sides of an equation Sup-Pos is an ordered paramodulation rule, not a superposition one.

Example 5.10 (Main Example) Continuing Example 5.9, by two-fold application of Neg-Res with left premise $\neg v$ one obtains from clause (13) the clause

$$\square \cdot g(x) \to x, f(x) \not\to x, h(x) \not\to x \ . \tag{15}$$

Recall from Example 5.4 the closure $((13), \{x \mapsto a\})$ whose clause part instantiated by $\{x \mapsto a\}$ is already normalized wrt. R_{Λ_1}, yet falsified by R_{Λ_1}. Clause (15) then serves to enable repairing this situation by a Split inference, as will be demonstrated in Example 6.2 below. $\square$

In the Sup-Neg, Sup-Pos and Neg-Res rules *we implicitly assume that the conclusion is parameter-free*. That can always be achieved by renaming parameters in the conclusion to fresh variables or by taking parameter-free variants of the left premise. We also assume that the two premises are variable-disjoint, which can be achieved by taking a fresh variant of the, say, right premise.

An *inference system* ι is a set of inference rules. By an *ι-inference* we mean an instance of an inference rule from ι such that the conditions stated with that rule are satisfied. An inference is *ground* if all its premises and the conclusion are ground.

The *base inference system* ι_{Base} consists of Ref, Sup-Neg, Sup-Pos and Neg-Res. If a ground ι_{Base}-inference results from a given ι_{Base}-inference by applying a substitution γ to all premises and the conclusion, we call the resulting ground inference a *ground instance via γ (of the ι_{Base}-inference)*. This is not always the case, as, e.g., ordering constraints can become unsatisfiable after application of γ.

An important consequence of the ordering restrictions for the inference rules is that the conclusion of a ground ι_{Base} inference is always strictly smaller than the rightmost premise.

6. The $\mathcal{ME}_{\text{E}}$ Calculus

In this section we describe the $\mathcal{ME}_{\text{E}}$ calculus, starting with its basic derivation rules. Then we define a model construction mechanism that will allow us to associate with each context a convergent rewrite system. On top of that, we then define concepts of redundancy and relevance, which together are the main tools to explain the calculus' completeness. Finally, we introduce certain optional derivation rules, which are important for practical efficiency.

6.1. Basic Derivation Rules

We now introduce the basic set of derivation rules that define the $\mathcal{ME}_{\text{E}}$ calculus. The rules operate on *sequents*, pairs of the form $\Lambda \vdash \Phi$ where Λ is a context and Φ is a set of constrained clauses. In the rules, we use the notation $\Lambda, K \vdash \Phi, C$ as an abbreviation of $\Lambda \cup \{K\} \vdash \Phi \cup \{C\}$.

The first rule of the calculus extends the inference rules ι_{Base} from Section 5.1 to sequents.

$$\text{Deduce} \ \frac{\Lambda \vdash \Phi}{\Lambda \vdash \Phi, C \cdot \Gamma}$$

if one of the following cases applies:

- $C \cdot \Gamma$ is the conclusion of a Ref inference with a premise from Φ.
- $C \cdot \Gamma$ is the conclusion of a Sup-Neg, Sup-Pos or Neg-Res inference with a right premise from Φ and a left premise $K \in \Lambda$ that produces $K\sigma$ in Λ, where σ is the mgu used in that inference.

  The (pseudo) literal K is the *selected rewrite literal (of a* Deduce *inference).*

In all cases above, the rightmost premise of the ι_{Base} inference rules applied by Deduce is called the *selected clause*.

Example 6.1 (Main Example) Consider again the context Λ_1 in Figure 1 and the sequent $\Lambda_1 \vdash (1), (2), (3)$. Starting with clause (3), by three Deduce inferences with underlying Sup-Pos and Neg-Res inferences one obtains a sequent containing clause (15). $\square$

The intuition behind the next rule, Split, is to make a constrained empty clause $\square \cdot \Gamma$ true, which is false when Λ satisfies Γ (in the sense of Definition 5.2). In this sense, the current context is "repaired" towards a model for a constrained empty clause. But as Λ is only an approximation of the intended interpretation, the repair mechanism will sometimes overshoot, yet it will cover all relevant cases (Lemma 4.8). The Split rule comes in two variants, depending on whether the chosen literal from Γ satisfies a certain variable-disjointness condition or not.

$$\text{U-Split}\ \frac{\Lambda \vdash \Phi,\ \square \cdot K, \Gamma}{\Lambda,\ \overline{K} \vdash \Phi,\ \square \cdot K, \Gamma \qquad \Lambda \vdash \Phi,\ \square \cdot \Gamma}$$

if (*i*) K is variable-disjoint with Γ (i.e., $\mathcal{V}ar(K) \cap \mathcal{V}ar(\Gamma) = \emptyset$), and (*ii*) $\overline{K}$ is not contradictory with Λ.

The literal $\overline{K}$ is the *split literal (of the* U-Split *inference).*

Because constraints are parameter-free, the split literal is parameter-free, too. Intuitively, U-Split can be understood to make the selected clause redundant in the left conclusion,[17] and in the right conclusion its constraint is shortened by (the complement of) the split literal. The case of $\Gamma = \emptyset$ is permissible; then, the clause set in the right conclusion will contain the constrained empty clause $\square \cdot \emptyset$.

For example, U-Split applied to $\square \cdot \neg v \rhd P(x, a) \not\to \mathbf{t}, \neg v \rhd Q(y) \not\to \mathbf{t}$ and $\Lambda = \emptyset$ gives the context $\{P(x, a) \to \mathbf{t}\}$ in the left conclusion, this way preventing $\neg v$ from producing $P(x, a) \not\to \mathbf{t}$ in $\{P(x, a) \to \mathbf{t}\}$. In the right conclusion the derived clause is $\square \cdot \neg v \rhd Q(y) \not\to \mathbf{t}$ and the context is still $\emptyset$.

$$\text{P-Split}\ \frac{\Lambda \vdash \Phi}{\Lambda,\ \overline{L} \vdash \Phi \qquad \Lambda,\ L \vdash \Phi}$$

if there is a constrained clause $\square \cdot K, \Gamma \in \Phi$ such that (*i*) K is not variable-disjoint with Γ, (*ii*) L is a variable-free variant of K, and (*iii*) neither $\overline{L}$ nor L is contradictory with Λ.

[17] So that it can be deleted in derivations by using the Simp rule, introduced below.

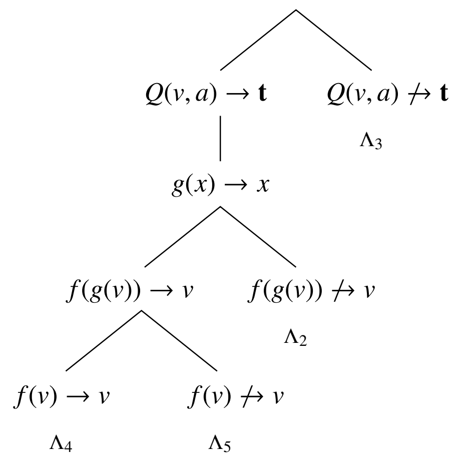

Figure 2: Context structure from Figure 1 after split.

The literal $\overline{L}$ is the *split literal (of the* P-Split *inference).*

For instance, in contrast to the just given example, P-Split applied to $\Box \cdot \neg v \rhd P(x,a) \not\to \mathbf{t}, \neg v \rhd Q(x) \not\to \mathbf{t}$ and $\Lambda = \emptyset$ gives the contexts $\{P(v,a) \to \mathbf{t}\}$ in the left conclusion and $\{P(v,a) \not\to \mathbf{t}\}$ in the right conclusion. Observe that P-Split no longer applies with the same literal, neither to the left-hand nor to the right-hand conclusion.

We will refer to either of the two rules above as a a Split rule and call the clause $\Box \cdot K, \Gamma$ the *selected clause (of the* Split-*inference).*

Example 6.2 (Main Example) Recall clause (15),

$$\Box \cdot g(x) \to x, f(x) \not\to x, h(x) \not\to x \ , \tag{15}$$

which was obtained from clause (3),

$$f(g(x)) \approx x \vee h(x) \approx x \cdot \emptyset \tag{3}$$

by application of three Deduce inferences in the context of Λ_1, cf. Example 6.1. Let $\Lambda_1 \vdash (1), (2), (3), \ldots, (15)$ be the resulting sequent. Now P-Split is applicable with selected clause (15), where the split literal is either $f(v) \to v$ or $g(v) \to v$. Let us chose $f(v) \to v$. The resulting context structure is visualized in Figure 2.

Extending Λ_1 with $f(v) \to v$ to obtain Λ_4 has the effect that the rule $f(a) \to a$ is added to the induced rewrite system, hence all ground terms collapse to a. More explicitly,

$$R_{\Lambda_4} = \{Q(a,a) \to \mathbf{t},\ g(a) \to a,\ f(a) \to a\} \ .$$

As emphasized at the end of Example 5.4, the main idea of the calculus is to identify ground closures that are normalized in their (instantiated) clause part, and to satisfy them. In the example, $(\Lambda_1, R_{\Lambda_1})$ falsifies the ground closure $((3), \{x \mapsto a\})$ but considers its (falsified) normalized version $((15), \{x \mapsto a\}) =: (\Box \cdot \Gamma, \gamma)$ instead. As a consequence of this normalization every positive rewrite literal in $\Gamma\gamma$, i.e., $g(a) \to a$, must be a rule in R_{Λ_1}, and every negative literal in $\Gamma\gamma$, i.e., $f(a) \not\to a$ and $h(a) \not\to a$, must be irreducible wrt. R_{Λ_1}, the latter because otherwise they would stem from a ground closure with non-normalized (instantiated) clause part.

In other words, $(\square \cdot \Gamma, \gamma)$ is a relevant closure wrt. R_{Λ_1}. The Split application above that led to Λ_4 achieves that $f(a) \to a$ is produced in Λ_4, hence by Note 5.6 we get $f(a) \to a \in R_{\Lambda_4}$, and so $((15), \{x \mapsto a\})$ is satisfied by $(\Lambda_4, R_{\Lambda_4})$, and also $((15), \{x \mapsto a\})$ is not relevant wrt. Λ_4. $\square$

Notice that a Split inference can never add an oriented literal to a context that already contains a variant of it or its complement, as this would contradict condition (*ii*) of U-Split and corresponding condition (*iii*) of P-Split.[18]

$$\text{Close}\ \frac{\Lambda \vdash \Phi, \square \cdot \Gamma}{\Lambda \vdash \Phi, \square \cdot \emptyset}$$

if for every $K \in \Gamma$ there is a $L \in \Lambda_{\geq}$ such that $K \gtrsim L$ if K is variable-disjoint with $\Gamma \setminus K$ and $K \sim L$ otherwise.

The constrained clause $\square \cdot \Gamma$ is the *selected clause (of the* Close *inference).*

In terms of standard notions, the condition in the Close rule is equivalent to have that for every literal $K \in \Gamma$, if (a) K does not share variables with other literals in Γ then K can be instantiated to a parametric context literal in Λ or K is unifiable with a (fresh variant of a) universal literal in Λ, or else (b) K is a variant of a parametric context literal in Λ or K is an instance of a universal literal in Λ.

The $\mathcal{ME}_\text{E}$ calculus consists of the *mandatory* inference rules Deduce, U-Split, P-Split and Close. By an $\mathcal{ME}_\text{E}$ *inference* we mean an instance of any of these inference rules. Below we will add optional inference rules to the $\mathcal{ME}_\text{E}$ calculus.

6.2. Redundancy

From a completeness perspective it is sufficient to work with a sufficiently developed *limit* context $\Lambda_\mathbf{B}$ of a derivation, formally introduced below, and check that all relevant closures wrt. $\Lambda_\mathbf{B}$ are satisfied by the pair $(\Lambda_\mathbf{B}, R_{\Lambda_\mathbf{B}})$. It would be useful to define clauses as redundant if they are satisfied by $(\Lambda_\mathbf{B}, R_{\Lambda_\mathbf{B}})$. However, this is difficult to check as $\Lambda_\mathbf{B}$, and hence its induced rewrite system, is generally not known at any (finite) point in the derivation. The calculus' notions of redundancy introduced below can therefore work only approximatively wrt. satisfaction in $(\Lambda_\mathbf{B}, R_{\Lambda_\mathbf{B}})$. In addition, they need to be robust under changes of contexts. For instance, to justify deleting a clause that is redundant at one point in the derivation it must be made sure that this clause *remains* redundant, independently from how the contexts and clause sets evolve.

To introduce our notion of redundancy we need one more prerequisite: we say that a ground constraint Γ is *reducible* by a ground rule $l \to r$ iff one of the following two cases applies:

[18]The Deduce rule could be strengthened to exclude adding variants to the clauses sets in the conclusion. We ignore this (trivial) aspect.

(i) there is a $s \to t \in \Gamma$ with $s \to t \succ l \to r$ such that s or t is reducible by $l \to r$, or

(ii) there is a $s \not\to t \in \Gamma$ such that s or t is reducible by $l \to r$.

It is not difficult to see that a ground closure $(C \cdot \Gamma, \gamma)$ cannot be a relevant closure wrt. Λ if the constraint $\Gamma\gamma$ is reducible by a rule in R_Λ. This fact is exploited to justify a specific case of redundancy, the second part in item (iii) in the following definition.

Definition 6.3 (Redundancy) *Let $\Lambda \vdash \Phi$ be a sequent, and $\mathcal{D}$ and $(C \cdot \Gamma, \gamma)$ ground closures. We say that $(C \cdot \Gamma, \gamma)$ is* redundant wrt. $\Lambda \vdash \Phi$ and $\mathcal{D}$, *if $\Lambda \not\models (\Gamma, \gamma)$ or there exist $n \geq 0$ ground closures $(C_i \cdot \Gamma_i, \gamma_i)$ of constrained clauses $C_i \cdot \Gamma_i \in \Phi$, with $i = 1, \ldots, n$, such that*

(i) *if $L' \rhd L \in \Gamma_i$ then there exists a $K' \rhd K \in \Gamma$ such that $L' \sim K'$, $L \sim K$ and $L\gamma_i = K\gamma$,*

(ii) *$\mathcal{D} \succ (C_i \cdot \Gamma_i, \gamma_i)$, for every i, and*

(iii) *$C_1\gamma_1, \ldots, C_n\gamma_n \models C\gamma$ or $C_1\gamma_1, \ldots, C_n\gamma_n \models l \approx r$ for some ground terms l and r with $l \succ r$ and such that $\Gamma\gamma$ is reducible by $l \to r$.*

We say that $(C \cdot \Gamma, \gamma)$ is *redundant wrt.* $\Lambda \vdash \Phi$, iff $(C \cdot \Gamma, \gamma)$ is redundant wrt. $\Lambda \vdash \Phi$ and $(C \cdot \Gamma, \gamma)$, and we say that $C \cdot \Gamma$ is *redundant wrt.* $\Lambda \vdash \Phi$ iff every ground closure $(C \cdot \Gamma, \gamma)$ of $C \cdot \Gamma$ is redundant wrt. $\Lambda \vdash \Phi$.

Intuitively, condition (i) ensures that if the given closure $(C \cdot \Gamma, \gamma)$ is relevant wrt. a rewrite system R_Λ and Λ satisfies (Γ, γ) (cf. Definition 5.1) then the same holds for each ground closure $(\Gamma_i \cdot C_i, \gamma_i)$ used to establish the redundancy. Condition (ii) makes sure that only ground closures smaller than the given closure $\mathcal{D}$ are used. Condition (iii) then adds sufficient conditions so that satisfaction in terms of Definition 5.2 is preserved or that $(C \cdot \Gamma, \gamma)$ is irrelevant because $\Gamma\gamma$ is not satisfied by R_Λ.

Referring to the notion of derivation trees formally defined in Section 7 below, it can be shown that a constrained clause that is redundant at some node of the derivation tree will remain redundant in all successor nodes. Consequently, a redundant clause can be deleted from a clause set without endangering refutational completeness.

A practically useful case of redundancy is when $\Lambda \not\models \exists\Gamma$ holds, say, because Γ contains an element $K' \rhd K$ such that K' does not produce K in Λ, as then $C \cdot \Gamma$ is redundant wrt. $\Lambda \vdash \Phi$, for every C and Φ. Another useful case is when a constraint literal in a constrained clause can be demodulated by an orientable positive unit clause, which justifies the deletion of the clause. For instance $C \cdot \Gamma, P(f(a)) \to \mathbf{t}$, can be deleted in presence of $f(x) \approx x \cdot \emptyset$. In our running example:

Example 6.4 (Main Example) Let $\Box \cdot \Gamma = (15)$ as discussed in Example 6.2 above. With $\gamma' = \{x \mapsto g(a)\}$ we get $\Gamma\gamma' = \{g(g(a)) \to g(a), f(g(a)) \not\to g(a), h(g(a)) \not\to g(a)\}$ and Condition (iii) applies to make $(\Box \cdot \Gamma, \gamma')$ redundant (take the ground closure $(g(x) \approx x \cdot \emptyset, \{x \mapsto a\})$ to check that $\Gamma\gamma'$ is reducible by $g(a) \to a$). This is justified as $(\Box \cdot \Gamma, \gamma')$ is not relevant wrt. Λ_1, and hence need not be considered. $\Box$

Any constrained clause $C \cdot L, \Gamma$ is redundant wrt. every $\Lambda \vdash \Phi$ such that $C \cdot \Gamma \in \Phi$. If $L \in \Lambda_\geq$ then the clausal form of $C \cdot \Gamma$ is a consequence of the clausal form of $C \cdot L, \Gamma$, and the Simp rule below then can be used to simplify $C \cdot L, \Gamma$ to $C \cdot \Gamma$. Dually, if $\overline{L} \in \Lambda_\geq$ then L

cannot be produced in Λ, it follows $\Lambda \not\models \exists L, \Gamma$ and so $C \cdot L, \Gamma$ is redundant wrt. $\Lambda \vdash \Phi$, for every C and Φ. These considerations show that $\mathcal{ME}_E$ generalizes corresponding simplification rules by unit clauses in the propositional DPLL-procedure.

Also, a constrained clause $C \cdot \Gamma'$ is redundant wrt. any sequent containing a constrained clause $C \cdot \Gamma$ such that $\Gamma \subset \Gamma'$.

Definition 6.5 (Redundant $\mathcal{ME}_E$ Inference) *Let $\Lambda \vdash \Phi$ and $\Lambda' \vdash \Phi'$ be sequents. An $\mathcal{ME}_E$ inference with premise $\Lambda \vdash \Phi$ and selected clause $C \cdot \Gamma \in \Phi$ is* redundant wrt. *$\Lambda' \vdash \Phi'$ iff for every grounding substitution γ, $(C \cdot \Gamma, \gamma)$ is redundant wrt. $\Lambda' \vdash \Phi'$, or the following holds, depending on the inference rule applied:*

Deduce: *applying γ to all premises and the conclusion $C' \cdot \Gamma'$ of the underlying ι_{Base} inference does not result in a ground instance via γ of this ι_{Base} inference, or $(C' \cdot \Gamma', \gamma)$ is redundant wrt. $\Lambda' \vdash \Phi'$ and $(C \cdot \Gamma)\gamma$.*

Split: *$C \cdot \Gamma = \square \cdot \Gamma$ and there is a $K' \rhd K \in \Gamma$ such that K' does not produce K in Λ', or, in case of* P-Split, *the split literal of the inference is contradictory with Λ'.*

Close: *$C \cdot \Gamma = \square \cdot \emptyset \in \Phi'$.*

It is not too difficult to see that actually carrying out an inference makes it redundant wrt. the resulting sequent(s). For Split, in particular, the condition "there is a literal $K \in \Gamma$ such that Λ' does not produce K" achieves that for the left sequent in the conclusion, by means of adding the split literal to Λ; for the right sequent, in the P-Split case, that is achieved by the condition "the split literal of the inference is contradictory with Λ'", and in the U-Split case the selected clause becomes redundant because of the new (shortened) clause. With a view to implementation, this indicates that effective proof procedures for $\mathcal{ME}_E$ indeed exist.

Finally, a sequent $\Lambda \vdash \Phi$ is *saturated* iff every $\mathcal{ME}_E$ inference with premise $\Lambda \vdash \Phi$ is redundant wrt. $\Lambda \vdash \Phi$.

6.3. Static Completeness

The calculus derives, possibly in the limit, a *saturated sequent* $\Lambda \vdash \Phi$, and the rewrite system $R^{\star}_{\Lambda}$ induced from that limit context Λ satisfies all ordinary clauses in Φ, i.e., $R^{\star}_{\Lambda} \models \{C \mid C \cdot \emptyset\}$, unless Φ contains the empty constrained clause with empty constraint, $\square \cdot \emptyset$. Note that the saturated sequent is not necessarily such that $(\Lambda, R_\Lambda) \models \Phi$. Intuitively, the evaluation of constraints according to Definition 5.1 is too strong to obtain $(\Lambda, R_\Lambda) \models \Phi$ in general: given a saturated sequent $\Lambda \vdash \Phi$ and a ground closure $(C \cdot \Gamma, \gamma)$ for some $C \cdot \Gamma \in \Phi$ we might have $\Lambda \models (\Gamma, \gamma)$ but $R_\Lambda \not\models C\gamma$, and the calculus cannot detect this situation (see the statement of Theorem 6.6 for an example). However, $(\Lambda, R_\Lambda) \models (C \cdot \Gamma, \gamma)$ does hold for all *relevant* ground closures $(C \cdot \Gamma, \gamma)$ of all constrained clauses in $C \cdot \Gamma \in \Phi$.

Theorem 6.6 (Static Completeness) *If $\Lambda \vdash \Phi$ is a saturated sequent with a non-contradictory context Λ and $\square \cdot \emptyset \notin \Phi$ then (Λ, R_Λ) satisfies all relevant instances of all clauses in Φ wrt. Λ , i.e., $(\Lambda, R_\Lambda) \models \Phi^{\Lambda}$. Moreover, if Ψ is a clause set and Φ includes Ψ, i.e., $\{D \cdot \emptyset \mid D \in \Psi\} \subseteq \Phi$, then $R^{\star}_{\Lambda} \models \Psi$.*

The stronger statement $(\Lambda, R_\Lambda) \models \Phi$ does in general not follow, as (Λ, R_Λ) possibly falsifies a *non-relevant* closure of a constrained clause in Φ. An example is the sequent (with $a \succ b$)

$$\Lambda \vdash \Phi \;=\; P(u) \to \mathbf{t},\ a \to b,\ P(b) \not\to \mathbf{t} \vdash P(x) \cdot P(x) \to \mathbf{t} \rhd P(x) \to \mathbf{t}\ .$$

We get $R_\Lambda = \{a \to b\}$. By taking $\gamma = \{x \mapsto a\}$ observe that $\Lambda \models (P(x) \to \mathbf{t} \rhd P(x) \to \mathbf{t}, \gamma)$ but $R^\star_\Lambda \not\models P(x)\gamma$, hence $(\Lambda, R_\Lambda) \not\models P(x) \cdot P(x) \to \mathbf{t} \rhd P(x) \to \mathbf{t}$. Deriving $\Box \cdot \neg v \rhd P(x) \not\to \mathbf{t}, P(x) \to \mathbf{t} \rhd P(x) \to \mathbf{t}$ does not help to close Λ. But notice that $P(x)\gamma = P(a)$ is reducible wrt. R_Λ, and so $(P(x) \cdot P(x) \to \mathbf{t} \rhd P(x) \to \mathbf{t}, \gamma)$ is not a *relevant* closure wrt. Λ, and Theorem 6.6 is not violated.

Theorem 6.6 applies to a *statically* given sequent $\Lambda \vdash \Phi$. The connection to the *dynamic* derivation process of the $\mathcal{ME}_\mathrm{E}$ calculus will be given later, and Theorem 6.6 will be essential then in proving the completeness of the $\mathcal{ME}_\mathrm{E}$ calculus.

6.4. Optional Derivation Rules

The calculus can be enhanced with a few optional and rather general derivation rules. Suitable specializations of these rules are useful in producing efficient implementations of $\mathcal{ME}_\mathrm{E}$. Some of these rules refer to the *clausal form* of a constrained clause $C \cdot \Gamma = C \cdot l_1 \to r_1, \ldots, l_k \to r_k, l_{k+1} \not\to r_{k+1}, \ldots, l_n \not\to r_n$, defined as the ordinary clause $C \vee l_1 \not\approx r_1 \vee \cdots \vee l_k \not\approx r_k \vee l_{k+1} \approx r_{k+1} \vee \cdots \vee l_n \approx r_n$ and denoted by $(C \cdot \Gamma)^\mathrm{c}$—note that the constraint's context literals are ignored in the clausal form. We define the clausal form of a set $\Phi = \{C_i \cdot \Gamma_i\}_i$ of constrained claused as the set $\Phi^\mathrm{c} = \{(C_i \cdot \Gamma_i)^\mathrm{c}\}_i$.

For the rest of the paper we fix a constant a from the signature Σ and the substitution $\alpha := \{v \mapsto a \mid v \in V\}$ that maps each parameter to a.[19] For each literal L, we denote by L^a the literal $L\alpha$. Note that L^a is ground if, and only if, L is variable-free. For each context Λ we will consider the set $\Lambda^a = \{(l \approx r)^a \mid l \to r \in \Lambda\} \cup \{(l \not\approx r)^a \mid l \not\to r \in \Lambda\}$ and treat it as a set of *unit clauses*.

The first optional derivation rule, Compact simplifies a context by removing a superfluous literal.

$$\textsf{Compact}\ \frac{\Lambda, L \vdash \Phi}{\Lambda \vdash \Phi}$$

if $L \in \Lambda_\geq$ (i.e., if there is a literal $K \in \Lambda$ such that $K \geq L$).

The Compact rule is the only rule that can remove literals from a context. It is easy to see that any literal produced by L in Λ is also produced by K in Λ, provided Λ is non-contradictory. Notice that K must be a universal literal, otherwise $\Lambda \cup \{L\}$ would contain two p-variants of the same (parametric) literal, which is impossible by construction of $\Lambda \cup \{L\}$. This means that it is possible, although not mandatory, to remove with each Compact inference all occurrences of L in the constraints in the clauses in Φ. See the discussion under *Simplification by Context Literals* further below for the justification.

[19]Strictly speaking, α is not a substitution in the standard sense because its domain is not finite. But this will cause no problems here.

Like DPLL, the $\mathcal{ME}$ calculus includes an optional derivation rule, called Assert, to insert a literal into a context without causing branching. In $\mathcal{ME}$ this rule bears close resemblance to the unit-resulting resolution rule. Here we propose the rather general rule Assert rule, defined below.

$$\text{Assert}\ \frac{\Lambda \vdash \Phi}{\Lambda, L \vdash \Phi}$$

if (*i*) $\Lambda^a \cup \Phi^{\mathrm{c}} \models L^a$, (*ii*) L is non-contradictory with Λ, and (*iii*) $L \notin \Lambda_{\geq}$.

As an example, Assert is applicable to the sequent $\neg v, P(u,b) \to \mathbf{t}, b \to c \vdash \neg P(x,y) \lor f(x) \approx y \cdot \emptyset$ to yield the new context literal $f(u) \to c$, as $\{P(a,b),\ b \approx c,\ \neg P(x,y) \lor f(x) \approx y\} \models f(a) \approx c$.

The third condition of Assert avoids the introduction of superfluous literals in the context. The first condition is needed for soundness. It is not decidable in its full generality and so can only be approximated with an incomplete test. This, however, is not a problem given that Assert is optional.

To make derivation in $\mathcal{ME}_{\mathrm{E}}$ practical the redundancy criteria defined in Subsection 6.2 should be made available not only to avoid redundant inferences, but also to simplify constrained clauses or delete redundant clauses. Instead of attempting to define individual derivation rules covering specific situations we provide a generic simplification rule Simp and discuss some of its instantiations.

$$\text{Simp}\ \frac{\Lambda \vdash \Phi, C \cdot \Gamma}{\Lambda \vdash \Phi, C' \cdot \Gamma'}$$

if (*i*) $C \cdot \Gamma$ is redundant wrt. $\Lambda \vdash \Phi, C' \cdot \Gamma'$, and (*ii*) $\Lambda^a \cup \Phi^{\mathrm{c}} \cup (C \cdot \Gamma)^{\mathrm{c}} \models (C' \cdot \Gamma')^{\mathrm{c}}$.

Condition (*ii*) is needed for soundness.

Trivial Equations. Any constrained clause $C \cdot \Gamma$ of the form $s \approx s \lor D \cdot \Gamma$ can be simplified to $\mathbf{t} \approx \mathbf{t} \cdot \emptyset$. Such a simplification step has the same effect as if $C \cdot \Gamma$ were deleted. Dually, any constrained clause $C \cdot \Gamma$ of the form $s \not\approx s \lor D \cdot \Gamma$ can be simplified to $D \cdot \Gamma$.

Proper Subsumption. The Simp rule also enables deletion of a constrained clause that is properly subsumed by another one. More formally, we say that $C' \cdot \Gamma'$ *properly subsumes* $C \cdot \Gamma$ iff there is a substitution σ such that $C'\sigma \subset C$ and $\Gamma'\sigma \subseteq \Gamma$, or $C'\sigma \subseteq C$ and $\Gamma'\sigma \subset \Gamma$.

Simplification by Context Literals. Another practically relevant application of Simp corresponds to applications of the unit resolution rule, both into the clause part and into the constraint part of a constrained clause. Regarding the former, suppose a context Λ, $K \vdash \Phi$, $C \lor L \cdot \Gamma$ where K is a universal rewrite literal, taken as an equational literal, and suppose there is a mgu of K and $\overline{L}$ such that $(C \cdot \Gamma)\sigma = C \cdot \Gamma$. Because K is parameter-free it follows that $(\Lambda \cup \{K\})^a \cup \{(C \lor L \cdot \Gamma)^{\mathrm{c}}\} \models ((C \cdot \Gamma)\sigma)^{\mathrm{c}}$. Together with $(C \cdot \Gamma)\sigma = C \cdot \Gamma$, this implies that the clause $C \lor L \cdot \Gamma$ can thus be simplified to $C \cdot \Gamma$. Dually, if there is a mgu of K and L such that $(C \lor L \cdot \Gamma)\sigma = C \lor L \cdot \Gamma$ then $(C \lor L \cdot \Gamma)$ can be deleted.

Even parametric context literals can be used for certain simplifications: in a sequent $\Lambda \vdash \phi$ a clause of the form $C \cdot \Gamma, K \rhd L \in \Phi$ can be deleted from Φ in particular if there is a (possibly parametric) literal $K' \in \Lambda_{\geq}$ such that $K \gtrsim \overline{K'} \gtrsim L$. Observe that this situation *always* comes up in the left context after a P-Split application. Consequently, in the left context the clause can again be deleted. However, in the right context the clause cannot be simplified as for U-Split above, such a simplification step would not be justified. For example, $\square \cdot P(x) \to \mathbf{t}, Q(x) \to \mathbf{t}$ cannot be simplified by means of a context literal $P(u) \to \mathbf{t}$ to $\square \cdot Q(x) \to \mathbf{t}$ (by U-Split then $Q(x) \not\to \mathbf{t}$ could be added to the context, which is clearly not sound).

The applications of Simp above have an explicit counterpart in $\mathcal{ME}$ and (on the propositional logic level) in DPLL, the Resolve and the Subsume rules. In other words, Simp covers—and generalizes—these rules.

Demodulation with Unit Constrained Clauses and Context Literals. A constrained clause comprised of an orientable positive unit clause and an empty constraint, i.e., a constrained clause of the form $l \approx r \cdot \emptyset$ with $l \succ r$, can be used to simplify by demodulation the clause part of a constrained clause. For instance, if $f(x) \approx x \cdot \emptyset$ and $f(y) \approx g(y) \vee f(a) \not\approx a \cdot f(y) \to a$ are among the current constrained clauses, then the latter can be simplified by two-fold demodulation with the former to obtain $y \approx g(y) \vee a \not\approx a \cdot f(y) \to a$.[20]

On the other hand, even when orientable and parameter-free, (positive) rewrite literals from a context cannot be used for demodulation. If that is desired and if the current context contains such a parameter-free literal $l \to r$ one can always *add* the corresponding unit clause $l \approx r \cdot \emptyset$ to the current constrained clause set and use that one for demodulation, as indicated above. From a soundness perspective a unit clause $l \approx r \cdot \emptyset$ and a parameter-free rewrite literal $l \to r$ are *indistinguishable*; both stand for the same ordinary unit clause $l \approx r$ (see Section 7 below).

From now on, we will consider the inference rules Simp, Assert and Compact part of the $\mathcal{ME}_{\mathrm{E}}$ calculus.

6.5. Derivations

Derivations in $\mathcal{ME}_{\mathrm{E}}$ are sequences of trees constructed with the inference rules above. Formally, we consider ordered trees $\mathbf{T} = (\mathbf{N}, \mathbf{E})$ where $\mathbf{N}$ and $\mathbf{E}$ are the sets of nodes and edges of $\mathbf{T}$, respectively, and the nodes N are labeled with sequents. We will often identify a tree's node with its label. Also, we will use κ to denote any ordinal up to and including the first infinite one.

Derivation trees $\mathbf{T}$ *(of a set* $\{C_1, \ldots, C_n\}$ *of clauses)* are defined inductively as follows: an *initial tree*, a single-node tree with a root of the form $\neg v \vdash C_1 \cdot \emptyset, \ldots, C_n \cdot \emptyset$, is a derivation tree; if $\mathbf{T}$ is a derivation tree, N is a leaf node of $\mathbf{T}$ and $\mathbf{T}'$ is a tree obtained from $\mathbf{T}$ by adding one or two child nodes to N so that N is the premise of an inference and the child node(s) is (are) its conclusion(s), then $\mathbf{T}'$ is a derivation

[20] Demodulation with unit clauses with non-empty constraints is also possible, as long as the instantiated constraints of the demodulating clause are among the constraints of the demodulated clause, in the sense of Definition 6.3-(i).

tree. In this case we say that $\mathbf{T}'$ is *derived* from $\mathbf{T}$. A *refutation tree* is a derivation tree all of whose leaves have a clause set containing the clause $\square \cdot \emptyset$. A *derivation (of $\{C_1, \ldots, C_n\}$)* is a possibly infinite sequence of derivation trees that starts with an initial tree and continues with trees each of which is derived from its immediate predecessor. Each derivation $\mathbf{D} = (\mathbf{N}_i, \mathbf{E}_i)_{i<\kappa}$ for some κ determines a *limit tree* $(\bigcup_{i<\kappa} \mathbf{N}_i, \bigcup_{i<\kappa} \mathbf{E}_i)$. Note that a limit tree is indeed a tree but is not a derivation tree unless it is finite—in which case it coincides with the last tree in the derivation.

Now let $\mathbf{T}$ be the limit tree of some derivation, let $\mathbf{B} = (N_i)_{i<\kappa}$ be a branch in $\mathbf{T}$ with κ nodes, and let $\Lambda_i \vdash \Phi_i$ be the sequent labeling node N_i, for all $i < \kappa$. Define $\Lambda_{\mathbf{B}} = \bigcup_{i<\kappa} \bigcap_{i \le j < \kappa} \Lambda_j$ and $\Phi_{\mathbf{B}} = \bigcup_{i<\kappa} \bigcap_{i \le j < \kappa} \Phi_j$, the sets of *persistent context literals* and *persistent clauses*, respectively. These two sets can be combined to obtain the *limit sequent* $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$ (of $\mathbf{T}$). The *limit rewrite system* is the rewrite system $R_{\Lambda_{\mathbf{B}}}$, written as $R_{\mathbf{B}}$ for convenience.

7. Correctness of the $\mathcal{ME}_{\mathrm{E}}$ Calculus

In this section we show that, for each given clause set, the $\mathcal{ME}_{\mathrm{E}}$ calculus eventually builds a refutation tree if and only if the clause set is unsatisfiable.

7.1. Soundness

To show that the $\mathcal{ME}_{\mathrm{E}}$ calculus is sound we use an adaptation of the notion of *a*-satisfiability from [9] to the equational case. A sequent $\Lambda \vdash \Phi$ is *a-(un)satisfiable* iff the clause set $\Lambda^a \cup \Phi^{\mathrm{c}}$ is E-(un)satisfiable.

The idea behind the soundness proof is to replace, conceptually, in a refutation tree every parameter in every context literal by the same constant a, and then treat context literals as unit clauses and constrained clauses as ordinary clauses by considering their clause form. This results in a refutation tree where all inferences are sound in the conventional sense. Technically, the soundness proof relies on the fact that the derivation rules of the calculus preserve a-satisfiability, as made precise in the following lemma.

Lemma 7.1 *For every application of an $\mathcal{ME}_{\mathrm{E}}$ derivation rule, if the sequent in the premise is a-satisfiable, so is one of the sequents in the conclusion.*

Proposition 7.2 (Soundness) *For all sets Ψ of clauses, if Ψ has a refutation tree then Ψ is E-unsatisfiable.*

7.2. Completeness

As usual, the completeness of $\mathcal{ME}_{\mathrm{E}}$ relies on a suitable notion of fairness, which is defined in terms of *exhausted* branches. Recall that we distinguish between the *mandatory* derivation rules, which are Deduce, U-Split, P-Split and Close, and the optional ones, Assert, Simp and Compact.

Definition 7.3 (Exhausted Branch) *Let $\mathbf{T}$ be a limit tree and $\mathbf{B} = (N_i)_{i<\kappa}$ a branch in $\mathbf{T}$ with κ nodes. For all $i < \kappa$, let $\Lambda_i \vdash \Phi_i$ be the sequent labeling node N_i. The branch $\mathbf{B}$ is* exhausted *iff*

(i) *for all $i < \kappa$, every $\mathcal{ME}_\mathrm{E}$ inference with a mandatory derivation rule with premise $\Lambda_i \vdash \Phi_i$ and persistent selected rewrite literal, if any, and persistent selected clause is redundant wrt. $\Lambda_j \vdash \Phi_j$, for some $j < \kappa$ with $j \geq i$, and*

(ii) $\Box \cdot \emptyset \notin \Phi_\mathbf{B}$.

A limit tree of a derivation is *fair* iff it is a refutation tree or it has an exhausted branch. A derivation is *fair* iff its limit tree is fair.

Proposition 7.4 (Exhausted Branches are Saturated) *If $\mathbf{B}$ is an exhausted branch in a limit tree of a fair derivation then $\Lambda_\mathbf{B} \vdash \Phi_\mathbf{B}$ is saturated.*

Proposition 7.4 is instrumental in the proof of our main result, which is the following.

Theorem 7.5 (Completeness) *Let Ψ be a clause set and let $\mathbf{T}$ be the limit tree of a fair derivation of Ψ. If $\mathbf{T}$ is not a refutation tree then Ψ is satisfiable; more specifically, for every exhausted branch $\mathbf{B}$ of $\mathbf{T}$ with limit sequent $\Lambda_\mathbf{B} \vdash \Phi_\mathbf{B}$ we have that $\Lambda_\mathbf{B}, R_{\Lambda_\mathbf{B}} \models (\Phi_\mathbf{B})^{\Lambda_\mathbf{B}}$ and $R^\star_{\Lambda_\mathbf{B}} \models \Psi$.*

Note 7.6 (Proof Procedures) Carrying out a Deduce inference makes that inference redundant wrt. the resulting sequent. Similarly, carrying out a U-Split or P-Split inference makes that inference redundant wrt. both resulting sequents. This indicates that a fair proof procedure for $\mathcal{ME}_\mathrm{E}$ indeed exists. (The Close rule is unproblematic.) See Section 8 for a more detailed description of the proof procedure implemented in our system. $\Box$

For sets consisting of clauses only with literals of the form $(\neg)P(t_1, \ldots, t_n) \approx \mathbf{t}$ or $(\neg)(t_1 \approx t_2)$ where each t_i is either a variable or a constant, every fair derivation is finite. This is due to the following reasons: contexts cannot grow indefinitely because no (non-contradictory) context can contain two p-variants of the same literal, and terms cannot grow in depth. Therefore, there are only finitely many U-Split and P-Split applications along each branch. As for constrained clauses, either their clause part is shortened by a Ref or Neg-Res inference, or their constants are replaced by smaller constants or variables by a Sup-Res or Neg-Res inference (recall that superposition into variables is not possible). It is easy to see that these operations cannot be repeated infinitely often along a branch. The same holds for the remaining rules. Altogether, this leads to the following decidability result:

Corollary 7.7 (Bernays-Schönfinkel Class with Equality) *The $\mathcal{ME}_\mathrm{E}$ calculus can be used as a decision procedure for the Bernays-Schönfinkel class, i.e., for sentences with the quantifier prefix $\exists^*\forall^*$.*

8. Implementation

The $\mathcal{ME}_\mathrm{E}$ calculus has been implemented within the E-Darwin theorem prover, which is a fork from the Darwin system [6] that implements the original $\mathcal{ME}$ calculus. E-Darwin is intended as a testbed for equality-based reasoning in conjunction with model evolution. For this purpose E-Darwin features a number of equality-related inferences in addition to the basic $\mathcal{ME}$, including those of the predecessor calculus [8]. Also, E-Darwin retains the original Darwin implementation, without equality. This enables the user to select which calculus to employ.

E-Darwin supports input in *TPTP* and *Protein* syntax. The system operates on clauses in clause normal form. Input formulas are clausified using the prover E [33]. Results can be returned in various forms, including the *SZS*-compliant result status used in the CASC [27, 36] competition for automated theorem provers. For satisfiable input E-Darwin can return a model if it terminates. More precisely, recall that the E-Interpretation I induced by a context is given as the rewrite system R_Λ obtained from the model construction (cf. Section 4.1). In order to effectively evaluate any ground literal L in I, by taking its normal form wrt. R_Λ, one needs to be able to identify all rewrite rules from R_Λ that can reduce L. This can be done effectively if the ordering is such that for any ground term there are only finitely many terms that are smaller than those occurring in L. Knuth-Bendix orderings satisfy this property.

Proofs are provided in the form of listings of the derivation steps taken, with the level of detail being selectable. E-Darwin is implemented in the functional/imperative language OCaml[21]. The prover is available under the GNU General Public License at the E-Darwin website[22].

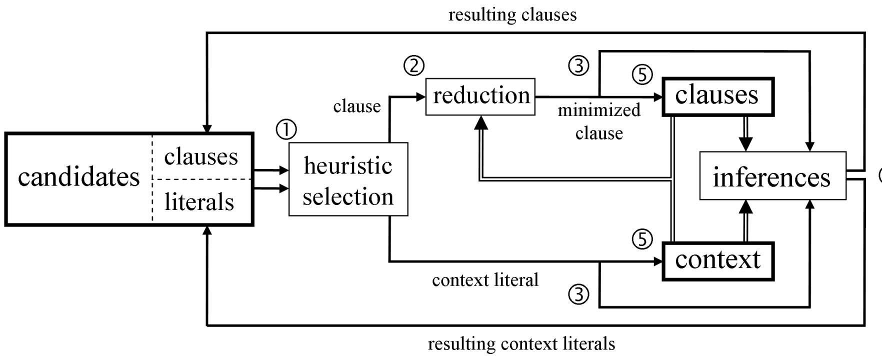

Figure 3: Main data structures and proving loop of E-Darwin. → indicates a transfer from one processing phase or data structure into another, and ⇒ indicates participation in inferences.

8.1. Proof Procedure

E-Darwin uses a uniform strategy for all classes of problems. Three principal data structures are maintained throughout the operation: the context and the set of clauses which together form the current sequent, and the set of *candidates*, the latter consisting of derived clauses and candidate split literals. The latter serves as an intermediate storage for inference results, each result being either a clause or a context literal. Candidates are not used in the reasoning before they have been selected for addition to

[21]`caml.inria.fr`

[22]`www.uni-koblenz.de/~bpelzer/edarwin`

the sequent and removed from the candidate set. Fig. 3 illustrates the relation between these structures. At first the context contains only the pseudo literal $\neg v$, while the clause set consists of the input. The candidate set is immediately populated by the inference results of the initial context and the input, i.e. clauses derived by Neg-Res using the $\neg v$ literal, and also new context literals derived by applying Assert to unit clauses. From then on the prover loop proceeds as follows.

In step ① the heuristic selection picks one element from the candidate set. Preference is given to candidates which can close the current derivation branch. Such candidates are detected by various lookahead functions which continuously compare candidates against each other as well as against the context and the clause set. Also, the selection avoids picking candidates which are subsumed by existing clauses and context literals, and currently non-productive clauses are avoided as well. Other selection criteria include preferring universal over parametric context literals, and clauses with an empty clause part over those clauses that have both constraint and non-constraint literals. Furthermore, an iterative deepening strategy, bounded by term and clause weight, favors the lighter candidates. More importantly, it also ensures that candidates derived by the Split inferences are selected only when all other candidate subsets have been exhausted within the weight bound. This mechanism generates a fair derivation strategy.

A Split candidate is essentially a context literal K, but upon its selection a choice point will be set. This allows backtracking: when all branches below K have been closed, the derivation continues with the right split alternative of K.

Once selected, a candidate can be used for inferencing. The first inferences applied to a clause candidate serve to reduce the clause to a minimal form. This happens in the clause reduction step labeled ②. Here the context and the clause set can demodulate the candidate and remove literals in accordance with the sub-rules covered by Simp. The Ref-rule makes additional reductions. This simplification phase is iterated exhaustively until a minimal clause has been derived which renders the original candidate clause and all intermediate clauses redundant. This means that for example a candidate clause C may be simplified to C' in the first iteration, which is then reduced to C'' in the next and final Simp-application. Only C'' will have to be added to the set of clauses after step ②.

While the goal of step ② is to produce a minimal clause for the clause set, the simplification process may also result in a tautological or redundant clause. In that case the prover returns to step ① for a new candidate. Another possibility is that the candidate clause gets reduced to the empty clause. This closes the branch immediately and initiates the backtracking procedure.

After the reduction the candidate is used as a premise in those inferences which derive new candidates, a step labeled ③ in the figure. Note that new context literals proceed to ③ right away after their selection in ①, skipping the simplification in ②. For a new clause with a non-empty clause part the derivation rules applied in step ③ are Sup-Neg, Sup-Pos and Neg-Res. The premise partners are taken from the context or the clause set as required. If a new clause has nothing but constraint literals, then it is subject to the Split rules as well as to Close. The optional Assert rule is applied to all selected clause candidates.

In principle these operations are accomplished by searches in discrimination tree indexes [23] over the sequent. Both imperfect and perfect indexing trees are used. The latter are an addition over the original Darwin, primarily serving to support the new superposition-based term rewriting. In practice these index searches are expensive, so E-Darwin keeps track of which pairings of context literals and clause positions unify and hence can be used as inference premises. Since terms are shared between literals and literals between clauses, this caching minimizes expensive index lookup operations, allowing a context literal to find all occurrences of a matching subterm in the set of clauses with a single index search and vice versa.

If the selected candidate is a new context literal, E-Darwin first attempts to Close with any existing clause and other context literals as required, triggering the backtracking mechanism if successful. Otherwise all Sup-Neg, Sup-Pos, Neg-Res and Assert inferences are computed in conjunction with the current sequent.

Any selected candidate may be able to simplify previously derived clauses, so Simp is computed. If a clause C is determined to be simplifiable by a candidate into C', C' is treated as an inference result and added to the candidate set. A redundancy mechanism then essentially deactivates C for the current sequent. Nevertheless this clause is still kept by E-Darwin, as its simplification may become invalid during backtracking. In this case the deactivation is quickly reversible, returning the original clause to the reasoning process.

All inference results passing some sanity checks are stored in the appropriate candidate set in step ④. In the final step ⑤ the selected candidate is inserted into the sequent, either into the context, applying Compact in the process, or into the clause set. Then the cycle starts anew.

From the implementation perspective, the possibility to derive new clauses is a significant difference between the $\mathcal{ME}$ and the $\mathcal{ME}_{\mathrm{E}}$ calculus. The implementation of $\mathcal{ME}$ in Darwin maintains a dynamic context and candidate set, while the set of clauses remains static, as no clause is ever added beyond the input. In E-Darwin, however, all sets must be dynamic in order to allow the derivation of new clauses. In comparison to the basic Darwin prover most existing modules had to be modified for E-Darwin. Combined with the additions this has resulted in an increase of the code size by approximately 50%.

8.2. *Experimental Evaluation*

We have tested E-Darwin over those problems of the TPTP library Version 4.0.1 [35] that are given in clause normal form (CNF) or as first order formulas (FOF). Overall the test set consisted of 13783 problems, constituting 83% of the whole TPTP with 16512 problems. (The remaining 17% of TPTP consist of higher-order logic problems, which are inappropriate for most current first-order automated theorem provers, including E-Darwin.) E-Darwin implements different selectable calculi, so for the test the prover was configured to use the $\mathcal{ME}_{\mathrm{E}}$ calculus as described in this paper. The test systems featured an Intel Q9950 quad processors at 2.83GHz. E-Darwin does not support parallelization at this point, so each problem was handled by one process utilizing one CPU core. The memory was limited to 1GB for each problem, and the maximum time allowed per problem was 300 seconds. Under these conditions E-Darwin solved 5977 problems, corresponding to 43.4% of the tested problems, or 36.2% of the full TPTP. The test set contains 3010 problems which carry a rating of 1.00, which means

ATP ystem	UNS	SAT	THM	CSA	total
Bliksem 1.12	2821	0	1685	0	4506 (33%)
Darwin 1.4.5	2327	507	2180	301	5315 (39%)
E 1.1	4211	536	3610	333	8690 (63%)
Equinox 4.1	2079	0	2487	0	4566 (33%)
E-Darwin 1.3	**2765**	**521**	**2344**	**347**	**5977 (43%)**
E-KRHyper 1.1.3	2022	235	1828	303	4388 (32%)
Geo 2007f	2270	623	1703	460	5056 (37%)
iProver 0.7	2687	604	3031	369	6691 (49%)
Metis 2.2	2257	0	1926	28	4211 (31%)
Otter 3.3	1691	0	1518	0	3209 (23%)
Prover9 0908	2834	0	2189	0	5023 (36%)
SNARK 20080805	2843	0	2446	0	5289 (38%)
SPASS 3.01	3406	535	2893	321	7155 (52%)
Vampire 11.0	4265	0	3229	0	7494 (54%)

Table 1: Comparison of our E-Darwin test results with the results for other provers on the same test set of 13783 problems as stated on the TPTP-website. The table lists the number of TPTP problems solved which are unsatisfiable (UNS), satisfiable (SAT), theorems (THM, unsatisfiable FOF problems with a conjecture) and countersatisfiable (CSA, satisfiable FOF problems with a conjecture), and finally the total number of problems solved as well as the percentage in relation to the complete test set.

that no automated theorem prover solves these problems at the time of the release of the TPTP version 4.0.1. E-Darwin proves six of these problems.[23]

The TPTP subset we used for our tests is commonly used for evaluating automated theorem provers for first-oder logic. The TPTP organizers periodically test theorem provers on appropriate TPTP problem sets and list the results on the TPTP website.[24] In Table 1 we quote the official TPTP results for a number of provers which have been tested on the same subset of the TPTP we used. The TPTP testing conditions are comparable to ours. The provers in the list are Bliksem [15], Darwin [6], E [33], Equinox [13], E-KRHyper [28], Geo [16], iProver [20], Metis [19], Otter [24], Prover9[25], SNARK [34], SPASS [37] and Vampire [31]. The table also includes our own E-Darwin results for comparison.

Overall our system occupies a middle ground, solving more problems than several established provers, but it does not rank among the top-rated systems. However, for a first implementation of the new calculus we believe this is a promising start. Generally E-Darwin offers an improvement over the original Darwin, but it must be noted that Darwin outperforms its successor in specialized problem classes. The original system excels at problems with a finite Herbrand universe - in 2006 and 2007 it won the EPR disivion (effectively propositional problems) of CASC. On the other hand it

[23]These six problems are: ALG035+1, GRP197-1, NUM378+1.020.015, PRO016+1, SWV527-1.040, and SWV527-1.050.

[24]`http://www.cs.miami.edu/~tptp/TPTP/Results.html`

[25]http://www.cs.unm.edu/~mccune/prover9/

was surpassed by the Otter system in several other categories (Otter serves as a benchmark in CASC as it has remained stable and unchanged for many years). The new calculus as implemented in E-Darwin is more generalized in its capabilities. Having participated in CASC in 2010, E-Darwin did not reach Darwin's positions for effectively propositional problems, but unlike Darwin it exceeded the Otter benchmark in the divisions CNF (clause normal form problems), FNE (first-order formula problems without equality), HNE (Horn without equality), NEQ (non-Horn with equality) and PEQ (purely equational). Only in the HEQ category (Horn with equality) did E-Darwin position below the benchmark, but this was the case for Darwin as well.

The uniform strategy of E-Darwin is a limitation at this point. At its core it remains the strategy used in the original Darwin for the non-equational $\mathcal{ME}$-calculus, and which had to decide between new context literals and splits. For E-Darwin this was extended by the selection of derived clauses, but the current scheme may be too rigid to account for all different problem classes. Our testing has shown that minor changes to the selection heuristics of E-Darwin can have a significant effect on the time it takes to solve a problem. The heuristics settings used to achieve the test results above were chosen to maximize the number of proofs. While other settings would result in slightly less problems solved in total, these would nevertheless include some problems that are not solved under the optimal settings. One noteworthy such parameter concerns the selection frequency of clauses with at least one constraint literal and at least one non-constraint literal. Such clauses cannot be used for closing, they can add great complexity to the derivation as superposition premises, and their actual relevance for a proof is difficult to estimate beforehand. A minor adjustment to their selection rate can mean a difference between solving a problem in less than five seconds and the same problem requiring over five minutes. More elaborate lookahead functions could provide some guidance here. Also, when the effects can be so large, an approach based on time slices may be appropriate, with the prover testing several strategies during the time allowed for a proof.

9. Conclusions

We have presented the $\mathcal{ME}_{\mathrm{E}}$ calculus, an extension of the Model Evolution calculus [7, 9] by superposition-based inference rules for equality. The $\mathcal{ME}_{\mathrm{E}}$ calculus as presented here is an extensively revised and improved version of an earlier extension of Model Evolution by equality inference rules [8]. The differences are numerous and include a, we think, simpler presentation and much more powerful redundancy criteria.

Our main theoretical result is the correctness of $\mathcal{ME}_{\mathrm{E}}$, in particular its completeness in combination with redundancy criteria. Our main practical result is the implementation in the E-Darwin system, described here for the first time. E-Darwin is a non-trivial extension of our earlier Darwin implementation. It performs reasonably well on the TPTP problem library and is able to solve six previously unsolved problems from that library.

As for future work, on the theoretical side, we plan to investigate how $\mathcal{ME}_{\mathrm{E}}$ can be exploited to obtain decision procedures for certain fragments of first-order logic that are beyond the scope of current superposition or instance-based methods. A key idea is to consider alternative formalisms to denote interpretations that are able to represent a larger class of infinite models, and adapt the derivation rules accordingly.

More on the applied side, it would be useful to investigate translations from practically interesting problems to fragments that can be decided by $\mathcal{ME}_{\mathrm{E}}$. In particular, $\mathcal{ME}_{\mathrm{E}}$ is a decision procedure for function-free clause logic (like other instance-based methods) with equality, a class with a NEXPTIME-complete satisfiability problem. Problems from that class include satisfiability of SHOIQ knowledge bases, first-order model expansion (a certain kind of constraint satisfaction problems), satisfiability of formulas of the Ackermann class with equality, Satisfiability of DQBF (Dependency Quantified Boolean Formulas, a generalization of QBF), first-order logic with two variables and counting quantifiers, and more. The challenge here is to find *practically* useful reductions into function-free clause logic.

The E-Darwin implementation, although already quite sophisticated, could still be improved. One of the most promising options is perhaps to look into more refined heuristics and strategies for search space exploration. In particular, the currently used iterative deepening on term weights is often too inflexible for refutation finding. Sometimes it is easy to find a refutation by starting with a certain weight bound n, but iterative deepening will get stuck on exploring all smaller bounds within a set time limit.

Acknowledgements

We thank the anonymous reviewers for their helpful comments, which let us improve the paper considerably.

Appendix A. Proofs

This appendix contains auxiliary lemmas, their proofs, and proofs of the results stated in the main part of this paper.

Lemma 4.8 *Let l and r be ground terms with $l \succ r$.*

(i) *If $l \to r \in R_\Lambda$ then Λ produces $l \to r$.*

(ii) *If l and r are irreducible wrt. R_Λ then Λ strongly produces $l \not\to r$.*

Proof. The statement (i) follows immediately from the definition of R_Λ. Concerning (ii), suppose that l and r are irreducible wrt. R_Λ. If Λ produces $l \to r$ we distinguish two cases. If R_Λ generates $l \to r$ then l is reducible by $l \to r \in R_\Lambda$. If R_Λ does not generate $l \to r$ then, by definition of R_Λ, l or r must be reducible wrt. $(R_\Lambda)_{l \to r}$, hence reducible wrt. R_Λ. Both cases thus contradict the assumption that l and r are irreducible wrt. R_Λ. It follows that Λ does not produce $l \to r$.

Thanks to the presence of the pseudo-literal $\neg x$ in every context, it is not difficult to see that every context produces K or $\overline{K}$, for every literal K. Thus, with Λ not producing $l \to r$ we can conclude that Λ strongly produces $l \not\to r$. □

Definition Appendix A.2 (Relevant Closures wrt. $\Lambda \vdash \Phi$) *Let $\Lambda \vdash \Phi$ be a sequent and $\mathcal{D}$ a ground closure. Define*

$$\Phi^\Lambda = \{(C \cdot \Gamma, \gamma) \mid C \cdot \Gamma \in \Phi \text{ and } (C \cdot \Gamma, \gamma) \text{ is a relevant closure wrt. } \Lambda\}, \text{ and}$$

$$\Phi^\Lambda_{\mathcal{D}} = \{C \in \Phi^\Lambda \mid \mathcal{D} \succ C\} \ .$$

In words, $\Phi^{\Lambda}_{\mathcal{D}}$ is the set of relevant closures wrt. Λ of all constrained clauses from Φ that are all smaller wrt. $>$ than $\mathcal{D}$.

Lemma Appendix A.3 *If (i) $(C \cdot \Gamma, \gamma)$ is redundant wrt. $\Lambda \vdash \Phi$ and $\mathcal{D}$, (ii) $(C \cdot \Gamma, \gamma)$ is a relevant closure of $C \cdot \Gamma$ wrt. Λ, and (iii) $(\Lambda, R_\Lambda) \models \Phi^{\Lambda}_{\mathcal{D}}$ then $(\Lambda, R_\Lambda) \models (C \cdot \Gamma, \gamma)$.*

Proof. Assume (*i*), (*ii*) and (*iii*). We have to show $(\Lambda, R_\Lambda) \models (C \cdot \Gamma, \gamma)$.

If $\Lambda \not\models (\Gamma, \gamma)$ then the conclusion follows trivially. Hence assume $\Lambda \models (\Gamma, \gamma)$ from now on. From (*ii*) conclude $R_\Lambda \models \Gamma\gamma$ by definition of relevance. With $\Lambda \models (\Gamma, \gamma)$, Definition 6.3 gives us ground closures $(C_i \cdot \Gamma_i, \gamma_i)$ of constrained clauses $C_i \cdot \Gamma_i \in \Phi$ that satisfy conditions (*i*) – (*iii*) in Definition 6.3

With $\Lambda \models (\Gamma, \gamma)$ from property (*i*) in Definition 6.3 it follows $R_\Lambda \models \Gamma_i\gamma_i$. Thus, each $(C_i \cdot \Gamma_i, \gamma_i)$ is a relevant closure wrt. Λ. Likewise, with $R_\Lambda \models \Gamma\gamma$ from property (*i*) in Definition 6.3 it follows $\Lambda \models (\Gamma_i, \gamma_i)$.

By condition (*ii*) in Definition 6.3, $(C_i \cdot \Gamma_i, \gamma_i)$ is smaller than $\mathcal{D}$. More formally, thus, $(C_i \cdot \Gamma_i, \gamma_i) \in \Phi^{\Lambda}_{\mathcal{D}}$, and with (*iii*) conclude $(\Lambda, R_\Lambda) \models (C_i \cdot \Gamma_i, \gamma_i)$. With $\Lambda \models (\Gamma_i, \gamma_i)$ from above it follows $R^{\star}_{\Lambda} \models C_i\gamma_i$.

By property (*iii*) of redundancy, $C_1\gamma_1, \ldots, C_n\gamma_n \models C\gamma$ or $C_1\gamma_1, \ldots, C_n\gamma_n \models l \approx r$ for some ground terms l and r with $l > r$ and such that $\Gamma\gamma$ is reducible by $l \to r$.

In the first case conclude $R^{\star}_{\Lambda} \models C\gamma$, and from that, trivially, $(\Lambda, R_\Lambda) \models (C \cdot \Gamma, \gamma)$, and nothing remains to be shown.

In the second case conclude $R^{\star}_{\Lambda} \models l \approx r$ with l and r as stated above. Because R_Λ is a convergent rewrite system, the normal forms of l and of r wrt. R_Λ are the same. Because l is greater wrt. $>$ than r, l must be reducible by some rule in $l' \to r' \in R_\Lambda$. But then, as $\Gamma\gamma$ is reducible by $l \to r$, it is straightforward to see that $\Gamma\gamma$ is reducible by $l' \to r'$ as well. In other words, $R_\Lambda \not\models \Gamma\gamma$, contradicting the assumption (*ii*). Hence the second case is impossible. □

Proposition Appendix A.4 *Let $\Lambda \vdash \Phi$ be a sequent and $(C \cdot \Gamma, \gamma)$ a ground closure. If (i) $C \cdot \Gamma$ is redundant wrt. $\Lambda \vdash \Phi$, (ii) $(C \cdot \Gamma, \gamma)$ is a relevant ground closure wrt. Λ, and (iii) $(\Lambda, R_\Lambda) \models \Phi^{\Lambda}_{(C \cdot \Gamma, \gamma)}$ then $(\Lambda, R_\Lambda) \models (C \cdot \Gamma, \gamma)$.*

Proof. Immediate from Lemma Appendix A.3 by setting $\mathcal{D} = (C \cdot \Gamma, \gamma)$ in Definition 6.3. □

Lemma Appendix A.5 *If $C \cdot \Gamma$ is redundant wrt. $\Lambda \vdash \Phi$, Λ' is obtained from Λ by deleting p-instances of other rewrite literals in Λ and/or by adding non-contradictory rewrite literals, and Φ' is obtained from Φ by deleting constrained clauses that are redundant wrt. $\Lambda \vdash \Phi$ and/or by adding arbitrary constrained clauses, then $C \cdot \Gamma$ is redundant wrt. $\Lambda' \vdash \Phi'$.*

Proof. It is obvious from Def. 6.3 that a clause that is redundant wrt. $\Lambda \vdash \Phi$ remains redundant if an arbitrary constrained clause is added to Φ; if a p-instance of another literal in Λ is deleted (by Compact) or a non-contradictory literal is added to Λ one needs to prove that, in terms of Def. 6.3, $(C \cdot \Gamma, \gamma)$ remains redundant wrt. $\Lambda \vdash \Phi$ and $\mathcal{D}$. The non-trivial case is when $\Lambda \not\models (\Gamma, \gamma)$ holds, but it is straightforward to check that $\Lambda \not\models (\Gamma, \gamma)$ is preserved even then.

To prove that a clause that is redundant wrt. $\Lambda \vdash \Phi$ remains redundant if redundant clauses are deleted from Φ, it suffices to show that the clauses $C_i \cdot \Gamma_i \in \Phi$ in Definition 6.3 can always be chosen in such a way that they are not themselves redundant or their deletion does not affect redundancy of $C \cdot \Gamma$: Suppose that a ground closure $(C \cdot \Gamma, \gamma)$ is redundant wrt. $\Lambda \vdash \Phi$ and $\mathcal{D}$. Let $\{ (C_i \cdot \Gamma_i, \gamma_i) \mid 1 \leq i \leq n \}$ be a minimal set of ground closures of clauses in Φ (wrt. the multiset extension of the clause ordering) that satisfies the conditions of Definition 6.3. Suppose that one of the $(C_i \cdot \Gamma_i, \gamma_i)$, say $(C_1 \cdot \Gamma_1, \gamma_1)$, is redundant itself. Then either $\Lambda \not\models (\Gamma_1, \gamma_1)$ and by condition (i) in Definition 6.3 it follows $\Lambda \not\models (\Gamma, \gamma)$, and so $(C \cdot \Gamma, \gamma)$ remains to be redundant, or there exist ground closures $(C_{1i} \cdot \Gamma_{1i}, \gamma_{1i})$ of constrained clauses $C_{1i} \cdot \Gamma_{1i} \in \Phi$ that satisfy the conditions of Definition 6.3 for $(C_1 \cdot \Gamma_1, \gamma_1)$. But then $\{ (C_i \cdot \Gamma_i, \gamma_i) \mid 2 \leq i \leq n \} \cup \{ (C_{1i} \cdot \Gamma_{1i}, \gamma_{1i}) \mid 1 \leq i \leq m \}$ would also satisfy the conditions of Definition 6.3 for $(C \cdot \Gamma, \gamma)$, contradicting the minimality of $\{ (C_i \cdot \Gamma_i, \gamma_i) \mid 1 \leq i \leq n \}$. □

Lemma Appendix A.6 *If a* Deduce *inference is redundant wrt. $\Lambda \vdash \Phi$, Λ' is obtained from Λ by deleting p-instances of other rewrite literals in Λ and/or by adding non-contradictory rewrite literals, and Φ' is obtained from Φ by deleting constrained clauses that are redundant wrt. $\Lambda \vdash \Phi$ and/or by adding arbitrary constrained clauses, then this* Deduce *inference is redundant wrt. $\Lambda' \vdash \Phi'$.*

Proof. Analogously to the proof of Lemma Appendix A.5. □

Lemma Appendix A.7 (ι_{Base}-inferences Preserve Relevant Closures) *Let $\Lambda \vdash \Phi$ be a sequent and assume an ι_{Base} inference with right (or only) premise $C \cdot \Gamma$, conclusion $C' \cdot \Gamma'$, and a ground instance via γ of the ι_{Base} inference such that*

(i) $(C \cdot \Gamma, \gamma)$ is a relevant closure of $C \cdot \Gamma$ wrt. Λ, and $\Lambda \models (\Gamma, \gamma)$,

(ii-a) in case of Sup-Neg *or* Sup-Pos*, where $l \to r$ is the left premise and σ is the mgu used, $l \to r$ produces $(l \to r)\sigma$ in Λ, $l \to r$ produces $(l \to r)\gamma$ in Λ, and $(l \to r)\gamma$ generates the rule $(l \to r)\gamma$ in R_Λ, and*

(ii-b) in case of Neg-Res*, where $l \to r$ is the left premise and σ is the mgu used, $\neg A$ produces $(s \not\to t)\sigma$ in Λ, $\neg A$ produces $(s \not\to t)\gamma$ in Λ, and $s\gamma$ and $t\gamma$ are irreducible wrt. R_Λ.*

Then, $(C' \cdot \Gamma', \gamma)$ is a relevant closure of $C' \cdot \Gamma'$ wrt. Λ, and $\Lambda \models (\Gamma', \gamma)$

Proof. For convenience we abbreviate $R := R_\Lambda$ below.

With (i), by Definitions 5.7 and 5.5 we have $\Lambda \models (C \cdot \Gamma, \gamma)$, i.e., $\Gamma\gamma$ is ordered and for every $K \rhd L \in \Gamma$, K produces L in Λ and K produces $L\gamma$ in Λ, and if $l \to r \in \Gamma\gamma$ then $l \to r \in R$, and if $l \not\to r \in \Gamma\gamma$ then l and r are irreducible wrt. R. We have to show

(1) $\Gamma'\gamma$ is ordered,

(2) for every $K' \rhd L' \in \Gamma'$, K' produces L' in Λ and K' produces $L'\gamma$ in Λ,

(3) if $l \to r \in \Gamma'\gamma$ then $l \to r \in R$, and

(4) if $l \not\to r \in \Gamma'\gamma$ then l and r are irreducible wrt. R.

The property (1) is easily obtained from inspection of the ι_{Base} inference rules. It remains to show (2), (3) and (4).

Let σ be the mgu used in the ι_{Base} inference, as mentioned in case (ii-a) and (ii-b). Assume σ is idempotent, which is the case with usual unification algorithms. Because γ gives a ground instance of the given ι_{Base} inference, γ must be a unifier for the same terms as σ. Because σ is a most general unifier, there is a substitution δ such that $\gamma = \sigma\delta$. With the idempotency of σ we get $\gamma = \sigma\delta = \sigma\sigma\delta = \sigma\gamma$.

For later use we prove some simple *facts*:

(i) if $K \rhd L\sigma \in \Gamma\sigma$ then K produces $L\sigma$ in Λ and K produces $L\sigma\gamma$ in Λ.
Proof: Assume $K \rhd L\sigma \in \Gamma\sigma$. We already know that K produces L in Λ and K produces $L\gamma$ in Λ. If K didn't produce $L\sigma$ in Λ then there would be a $K' \in \Lambda_{\geq}$ with $K \gtrsim\!\!\!\!/ \ \overline{K'} \gtrsim L\sigma$. With $\gamma = \sigma\delta$ and by transitivity of $\gtrsim$ we would get $K \gtrsim\!\!\!\!/ \ \overline{K'} \gtrsim L\gamma$, and so K would not produce $L\gamma$ either. With $\gamma = \sigma\gamma$ obtained above the second claim is trivial.

(ii) if $l \to r \in \Gamma\sigma\gamma$ then $l \to r \in R$.
Proof: we already know that if $l \to r \in \Gamma\gamma$ then $l \to r \in R$. The claim then follows immediately with $\gamma = \sigma\gamma$.

(iii) if $l \not\to r \in \Gamma\sigma\gamma$ then l and r are irreducible wrt. R.
Proof: we already know that if $l \not\to r \in \Gamma\gamma$ then l and r are irreducible wrt. R. The claim then follows immediately with $\gamma = \sigma\gamma$.

To prove (2), (3) and (4) we carry out a case analysis with respect to the ι_{Base} inference rule applied.

In case of a Ref inference let the premise be $s \not\approx t \vee C'' \cdot \Gamma$ and the conclusion $C' \cdot \Gamma' = (C'' \cdot \Gamma)\sigma$. Recall that σ is not applied to context literals of constraints, and so the context literals of Γ and Γ' are the same. With $\Gamma' = \Gamma\sigma$, (2) follows directly from fact (i), (3) follows immediately from fact (ii), and (4) follows immediately from fact (iii).

In case of a Sup-Neg inference let the left premise be $l \to r$, the right premise $C \cdot \Gamma = s[u]_p \approx t \vee C'' \cdot \Gamma$ and the conclusion $C' \cdot \Gamma' = (s[r]_p \approx t \vee C'' \cdot \Gamma, l \to r \rhd l \to r)\sigma$. The proofs of (2), (3) and (4) for the subset $\Gamma\sigma$ of Γ' follows immediately from facts (i), (ii) and (iii), respectively. Now consider the sole additional element $(l \to r)\sigma$ that is in Γ' but not in $\Gamma\sigma$. Recall we are given that $l \to r$ produces $(l \to r)\sigma$ in Λ and that $l \to r$ produces $(l \to r)\gamma = (l \to r)\sigma\gamma$ in Λ, which proves (2). Regarding (3), recall we are given that $(l \to r)\gamma$ generates $(l \to r)\gamma$ in R, which entails $(l \to r)\gamma = (l \to r)\sigma\gamma \in R$.

The proof for the case of a Sup-Pos inference is the same, and the proof for the case of a Neg-Res is similar and is omitted. □

Theorem 6.6 (Static Completeness) *If $\Lambda \vdash \Phi$ is a saturated sequent with a non-contradictory context Λ and $\Box \cdot \emptyset \notin \Phi$ then (Λ, R_Λ) satisfies all relevant instances of all clauses in Φ wrt. Λ , i.e., $(\Lambda, R_\Lambda) \models \Phi^\Lambda$. Moreover, if Ψ is a clause set and Φ includes Ψ, i.e., $\{D \cdot \emptyset \mid D \in \Psi\} \subseteq \Phi$, then $R_\Lambda^\star \models \Psi$.*

Proof. Let $\Lambda \vdash \Phi$ be a saturated sequent with a non-contradictory context and suppose $\Box \cdot \emptyset \notin \Phi$.

To complete the proof of the first statement we show that every relevant closure $(C \cdot \Gamma, \gamma)$ wrt. Λ, of every constrained clause $C \cdot \Gamma \in \Phi$ is canonically satisfied, i.e., satisfies the property

(P) $(\Lambda, R_\Lambda) \models (C \cdot \Gamma, \gamma)$.

Once $(\Lambda, R_\Lambda) \models \Phi^\Lambda$ is established we get the second statement $R^\star_\Lambda \models \Psi$ by the following argumentation. Let $C\gamma$ be a ground instance of a clause $C \in \Psi$. It suffices to show $R^\star_\Lambda \models C\gamma$. With Definition 5.7 it follows that every ground closure of a constrained clause with empty constraint is always relevant, for every pair (Λ, R). Hence, and more formally, $(C \cdot \emptyset, \gamma) \subseteq \{D \cdot \emptyset \mid D \in \Psi\}^\Lambda$. With $\{D \cdot \emptyset \mid D \in \Psi\} \subseteq \Phi$ conclude trivially $(C \cdot \emptyset, \gamma) \subseteq \Phi^\Lambda$. With $(\Lambda, R_\Lambda) \models \Phi^\Lambda$ we get $(\Lambda, R_\Lambda) \models (C \cdot \emptyset, \gamma)$, which means $\Lambda \not\models \emptyset, \gamma$ or $R^\star_\Lambda \models C\gamma$, equivalently $R^\star_\Lambda \models C\gamma$.

We prove (P) by contradiction. Every counterexample, that is, every closure $(C \cdot \Gamma, \gamma)$ of a constrained clause $C \cdot \Gamma \in \Phi$ that is relevant wrt. Λ and that does not satisfy (P) must satisfy the following properties:

(i) $R_\Lambda \models \Gamma\gamma$, by relevancy.
(ii) $\Lambda \models (\Gamma, \gamma)$, and
(iii) $R^\star_\Lambda \not\models C\gamma$, from $(C \cdot \Gamma, \gamma)$ not satisfying (P) by Definition 5.2.

Among all counterexamples, by well-foundedness of the ordering $\succ$ on ground closures, there is a minimal counterexample (minimal wrt. $\succ$). From now on let $(C \cdot \Gamma, \gamma)$ be such a minimal counterexample.

By minimality of $(C \cdot \Gamma, \gamma)$, every relevant closure of a constrained clause in Φ that is smaller wrt. $\succ$ than $(C \cdot \Gamma, \gamma)$ satisfies (P). More formally, $(\Lambda, R_\Lambda) \models \Phi^\Lambda_{(C \cdot \Gamma, \gamma)}$. Let us consider all possible cases.

(1) $(C \cdot \Gamma, \gamma)$ is redundant wrt. $\Lambda \vdash \Phi$.
If $(C \cdot \Gamma, \gamma)$ is redundant wrt. $\Lambda \vdash \Phi$, then by Lemma Appendix A.3, setting $\mathcal{D} = (C \cdot \Gamma, \gamma)$ there, (P) follows immediately, contradicting our assumption. Hence, $(C \cdot \Gamma, \gamma)$ cannot be redundant wrt. $\Lambda \vdash \Phi$.

(2) $\mathcal{V}ar(C)\gamma$ is reducible wrt. R_Λ.
The $\mathcal{ME}_\mathrm{E}$ calculus does not paramodulate into or below variables. To explain the completeness of this restriction we need to know that $\mathcal{V}ar(C)\gamma$ is irreducible wrt. R_Λ.

From (i) we know $R_\Lambda \models \Gamma\gamma$. First we show that every term in $(\mathcal{V}ar(C) \cap \mathcal{V}ar(\Gamma))\gamma$ is irreducible wrt. R_Λ. If there were such a term, reducible wrt. R_Λ, occurring in a negative rewrite literal $l \not\to r \in \Gamma\gamma$ then we would immediately get a contradiction to $R_\Lambda \models \Gamma\gamma$. If there were such a term occurring in a positive rewrite literal $l \to r \in \Gamma\gamma$ then $l \to r$ is reducible by a smaller rule from R_Λ, and hence $l \to r$ cannot be generated in R_Λ, again contradicting $R_\Lambda \models \Gamma\gamma$. That that rule is indeed smaller than $l \to r$ follows from the fact that by construction, as superposition into variables is not possible, Γ cannot contain rewrite literals of the form $x \to t$, where x is a variable. Thus, if $x \in \mathcal{V}ar(C) \cap \mathcal{V}ar(\Gamma)$ then $x\gamma$ is a *proper* subterm of l (or a subterm of r).

If $x\gamma$ is reducible for some $x \in \mathcal{V}ar(C) \setminus \mathcal{V}ar(\Gamma)$, then a term in the range of γ can be replaced by a smaller yet congruent term wrt. $R^\star_\Lambda$. Observe that this results in a smaller (wrt. $\succ$) relevant counterexample, thus contradicting the choice of $(C \cdot \Gamma, \gamma)$.

In summary, thus, $\mathcal{V}ar(C)\gamma$ is irreducible wrt. R_Λ.

(3) $C = s \not\approx t \vee D$ with selected literal $s \not\approx t$.
Suppose that none of the preceding cases holds and $C \cdot \Gamma = s \not\approx t \vee D \cdot \Gamma$ and $s \not\approx t$ is selected in $s \not\approx t \vee D$.

(3.1) $s\gamma = t\gamma$.
If $s\gamma = t\gamma$ then there is a ground Deduce inference with premise $C\gamma = (s \not\approx t \vee D \cdot \Gamma)\gamma$ and conclusion $(D \cdot \Gamma)\gamma$, which is an instance of a Deduce inference with an underlying Ref inference applied to $C \cdot \Gamma$ inference with selected clause $s \not\approx t \vee D \cdot \Gamma$ and conclusion $(D \cdot \Gamma)\sigma$. It is safe to assume that σ is idempotent, which gives us $\sigma\gamma = \gamma$.

By saturation, that Deduce inference is redundant wrt. $\Lambda \vdash \Phi$. Because the closure $(C \cdot \Gamma, \gamma)$ of the premise $C \cdot \Gamma$ is not redundant wrt. $\Lambda \vdash \Phi$, the derived clause, taken as the closure $((D \cdot \Gamma, l' \to r' \rhd l' \to r')\sigma, \gamma)$ must be redundant wrt. $\Lambda \vdash \Phi$ and $(C \cdot \Gamma, \gamma)$ by definition of redundant inferences. Furthermore, with Lemma Appendix A.7 it is a relevant closure wrt. Λ, hence, by Lemma Appendix A.3, $(\Lambda, R_\Lambda) \models (D \cdot \Gamma, l' \to r' \rhd l' \to r')\sigma, \gamma)$. By definition, this means $\Lambda \not\models ((\Gamma \cup \{l' \to r' \rhd l' \to r'\})\sigma, \gamma)$ or $R_\Lambda^\star \models D\sigma\gamma$. However, Lemma Appendix A.7 gives us additionally $\Lambda \models ((\Gamma \cup \{l' \to r' \rhd l' \to r'\})\sigma, \gamma)$, and so the former case is impossible. But then, from $R_\Lambda^\star \models D\sigma\gamma$ and with $\sigma\gamma = \gamma$ it follows $R_\Lambda^\star \models D\gamma$, and so, trivially, $R_\Lambda^\star \models C\gamma$, a plain contradiction to (*iii*) above.

(3.2) $s\gamma \neq t\gamma$.
If $s\gamma \neq t\gamma$ then without loss of generality assume $s\gamma \succ t\gamma$. The property (*iii*) above is $R_\Lambda^\star \not\models (s \not\approx t \vee D)\gamma$, and so $R_\Lambda^\star \models (s \approx t)\gamma$. Because R_Λ is a convergent (ordered) rewrite system, $s\gamma$ and $t\gamma$ must have the same normal forms. In particular, thus, $s\gamma$ is reducible wrt. R_Λ. Suppose $s\gamma = (s\gamma)[l]_p$ for some position p and rule $l \to r \in R_\Lambda$. With Lemma 4.8-(*i*) it follows that Λ produces $l \to r$. For later use let $l' \to r'$ be a fresh p-variant of a rewrite literal in Λ that produces $l \to r$ in Λ and assume that γ has already been extended so that $(l' \to r')\gamma = l \to r$.

The conclusions so far give that Deduce is applicable with underlying ground Sup-Neg inference with left premise $(l' \to r')\gamma$, right premise $s\gamma[l'\gamma]_p \not\approx t\gamma \vee D\gamma \cdot \Gamma\gamma$ and conclusion $s\gamma[r'\gamma]_p \not\approx t\gamma \vee D\gamma \cdot \Gamma\gamma, l' \to r' \rhd (l' \to r')\gamma$. The next step is to show that this ground inference is a ground instance via γ of a Sup-Neg inference with premises $l' \to r'$ and $C \cdot \Gamma = s[u]_p \not\approx t \vee D \cdot \Gamma$ and conclusion $(s[r']_p \not\approx t \vee D \cdot \Gamma, l' \to r' \rhd l' \to r')\sigma$, where σ is an mgu of l' and u.

The position p in $s\gamma$ cannot be at or below a variable position in s, because otherwise we had $x\gamma[l'\gamma]_p$ for some variable x occuring in s, and so $x\gamma$ would be reducible by $(l' \to r')\gamma = l \to r)$, which is impossible by case (2) above. Hence, the position p exists in s, and the term u at that position is not a variable. Then it follows easily that the mgu σ of l' and u exists. It is safe to assume that σ is idempotent, which gives us $\sigma\gamma = \gamma$.

By saturation, the Deduce inference is redundant wrt. $\Lambda \vdash \Phi$. Because the closure $(C \cdot \Gamma, \gamma)$ of the premise $C \cdot \Gamma$ is not redundant wrt. $\Lambda \vdash \Phi$, the derived clause, taken as the closure $((s[r']_p \not\approx t \vee D \cdot \Gamma, l' \to r' \rhd l' \to r')\sigma, \gamma)$ must be redundant wrt. $\Lambda \vdash \Phi$ and $(C \cdot \Gamma, \gamma)$ by definition of redundant inferences. Furthermore, with Lemma Appendix A.7 it is a relevant closure wrt. Λ, hence, by Lemma Appendix A.3, $(\Lambda, R_\Lambda) \models ((s[r']_p \not\approx t \vee D \cdot \Gamma, l' \to r' \rhd l' \to r')\sigma, \gamma)$. By definition, this means $\Lambda \not\models ((\Gamma \cup \{l' \to r' \rhd l' \to r'\})\sigma, \gamma)$ or $R_\Lambda^\star \models (s[r']_p \not\approx t \vee D)\sigma\gamma$. However, Lemma Appendix A.7 gives us additionally $\Lambda \models ((\Gamma \cup \{l' \to r' \triangleright l' \to r'\})\sigma, \gamma)$, and $R^\star_\Lambda \models (s[r']_p \not\approx t \vee D)\sigma\gamma$ follows. With $\sigma\gamma = \gamma$ we get a plain contradiction to (*iii*) above.

With $(l' \to r')\gamma \in R_\Lambda$ by congruence and $\sigma\gamma = \gamma$ it follows $R^\star_\Lambda \models (s \not\approx t \vee D)\gamma$, however $R^\star_\Lambda \not\models (s \not\approx t \vee D)\gamma$ was assumed for case (3.2) above, a plain contradiction.

(4) $C = s \approx t \vee D$ with selected literal $s \approx t$.
Suppose $C \cdot \Gamma = s \approx t \vee D \cdot \Gamma$ and $s \approx t$ is selected in $s \approx t \vee D$. With $(C \cdot \Gamma, \gamma)$ being a counterexample it follows $\Lambda \models (\Gamma, \gamma)$ but $R^\star_\Lambda \not\models (s \approx t \vee D)\gamma$. From the latter conclude immediately $R^\star_\Lambda \not\models (s \approx t)\gamma$, and so $s\gamma = t\gamma$ is impossible. Hence suppose $s\gamma \neq t\gamma$. We distinguish two further cases.

(4.1) $s\gamma$ or $t\gamma$ is reducible wrt. R_Λ.
If $s\gamma$ or $t\gamma$ is reducible wrt. R_Λ then there is a rule $l \to r \in R_\Lambda$ such that $s\gamma = s\gamma[l]_p$ or $t\gamma = t\gamma[l]_p$, for some position p. But then the same argumentation as in case (3.2) applies. The only changes are that instead of a (ground instance of a) Sup-Neg inference now a (ground instance of a) Sup-Pos inference is considered, and that $s \succ t$ does not apply.

(4.2) $s\gamma$ and $t\gamma$ are irreducible wrt. R_Λ.
If $s\gamma$ and $t\gamma$ are irreducible wrt. R_Λ then assume, w.l.o.g., $s\gamma \succ t\gamma$. With Lemma 4.8-(*ii*) then conclude that some literal $\neg A \in \Lambda$ produces $(s \not\to t)\gamma$ in Λ. This indicates that a Deduce inference with an underlying ground Neg-Res inference exists. More precisely, the left premise of that ground inference is $(s \not\to t)\gamma$, the right premise is $(s \approx t \vee D \cdot \Gamma)\gamma$ and the conclusion is $(D \cdot \Gamma, \neg A \triangleright s \not\to t)\gamma$. It is routine by now to check that this ground Neg-Res inference is a ground instance via γ of a Neg-Res inference with a right premise from Φ that is not redundant wrt. $\Lambda \vdash \Phi$, and the left premise $\neg A$.

The rest of the proof uses the same arguments as in case (3.2) and is omitted (we can show that the Deduce inference with the latter underlying Neg-Res inference exists, which will yield a contradiction to the conclusion $R^\star_\Lambda \not\models (s \approx t \vee D)\gamma$ drawn for case (4) above).

(5) $C = \square$.
Suppose $C \cdot \Gamma = \square \cdot \Gamma$. By assumption $\square \cdot \emptyset \notin \Phi$, and so $\Gamma \neq \emptyset$. First we are going to show that Split is applicable to $\Lambda \vdash \Phi$ with selected clause $\square \cdot \Gamma \in \Phi$. With property (*ii*), Λ produces every literal in Γ. More specifically, L' produces L in Λ and L' produces $L\gamma$ in Λ, for every $L' \triangleright L \in \Gamma$. (*)

If Close were applicable, then, by saturation, this Close inference would be redundant, which is the case only if $\square \cdot \emptyset \in \Phi$, which we have already excluded. Hence, Close is not applicable, and there is a literal $K \in \Gamma$ such that (*i*) K is variable-disjoint with $\Gamma \setminus \{K\}$ and there is no $K' \in \Lambda_\geq$ with $K \gtrsim K'$, or otherwise (*ii*) there is no $K' \in \Lambda_\geq$ with $K' \sim K$. In case (*i*) conclude that $\overline{K}$ is not contradictory with Λ, and in case (*ii*) conclude that $\overline{L}$ is not contradictory with Λ, where L is a variable-free variant of K. This conclusion will be needed below when we consider a Split inference.

In order to show that Split is applicable we still need to know in case (*ii*) that L is not contradictory with Λ. This follow easily from Definition 4.4 and the conclusion above that Λ produces every literal in Γ. In case (*i*), thus, U-Split is applicable, and in case (*ii*) P-Split is applicable with selected clause $\square \cdot \Gamma$ and split literal $\overline{K}$ and $\overline{L}$, respectively.

By redundancy then, (a) there is a $L' \rhd L \in \Gamma$ such that L' does not produce L in Λ, or (b) in the P-Split case the split literal is contradictory with Λ. The case (a) plainly contradicts (*), and case (b) plainly contradicts an earlier conclusion that $\overline{L}$ is not contradictory with Λ. $\square$

Lemma 7.1 *For every application of an $\mathcal{ME}_{\mathrm{E}}$ derivation rule, if the sequent in the premise is a-satisfiable, so is one of the sequents in the conclusion.*

Proof. We prove the claim for each derivation rule of $\mathcal{ME}_{\mathrm{E}}$.

Deduce) Let $\Lambda \vdash \Phi$ be the premise sequent and $\Lambda \vdash \Phi, D$ the conclusion sequent, and let I be an E-model of $\Lambda^a \cup \Phi^{\mathrm{c}}$.

If D is the conclusion of a Ref inference, then $D = (C \cdot \Gamma)\sigma$ for some constrained clause $s \not\approx t \vee C \cdot \Gamma \in \Phi$ and mgu σ of s and t. Observing that $D^{\mathrm{c}} = (C \cdot \Gamma)^{\mathrm{c}}\sigma$ and $I \not\models (s \not\approx t)\sigma$, it is easy to see that I satisfies D^{c} as well.

If D is the conclusion of a Sup-Pos or a Sup-Neg inference, then it has the form $(L[r]_p \vee C \cdot \Gamma, l \to r)\sigma$ for some constrained clause $L[l']_p \vee C \cdot \Gamma \in \Phi$, rewrite literal $l \to r \in \Lambda$, and mgu σ of l and l'. By elementary satisfiability arguments, we have that since I is a model of $l \approx r$ and of $(L[l']_p \vee C \cdot \Gamma)^{\mathrm{c}}$, it is also a model of $(L[l]_p \vee C \cdot \Gamma, l \to r)^{\mathrm{c}}\sigma$, and so of D.

If D is the conclusion of a Neg-Res inference, then $D = (C \cdot \Gamma, s \not\to t)\sigma$ for some constrained clause $s \not\approx t \vee C \cdot \Gamma \in \Phi$ and substitution σ. In that case, I satisfies D^{c} simply because D^{c} is equivalent to $(s \not\approx t \vee C \cdot \Gamma)^{\mathrm{c}}\sigma$, and I satisfies $(s \not\approx t \vee C \cdot \Gamma)^{\mathrm{c}}$ by assumption.

P-Split) Let $\Lambda \vdash \Phi$ be the premise sequent and let $\Lambda, \overline{L} \vdash \Phi$ and $\Lambda, L \vdash \Phi$ be the two conclusion sequents. Suppose that $\Lambda \vdash \Phi$ is a-satisfiable and note that L^a is ground. Clearly, one of the two clause sets

$$\Lambda^a \cup \{\overline{L}^a\} \cup \Phi^{\mathrm{c}} \quad \text{and} \quad \Lambda^a \cup \{L^a\} \cup \Phi^{\mathrm{c}}$$

must be E-satisfiable. By definition, this means that either $\Lambda, \overline{L} \vdash \Phi$ or $\Lambda, L \vdash \Phi$ is a-satisfiable.

U-Split) Let $\Lambda \vdash \Phi, \square \cdot K_1, \dots, K_n$ be the premise sequent where $n \geq 1$ and K_1 is variable disjoint with $K_2, \dots, K_n$. Let $\Lambda, \overline{K_1} \vdash \Phi, \square \cdot K_1, \dots, K_n$ and $\Lambda \vdash \Phi, \square \cdot K_2, \dots, K_n$ be the two conclusion sequents. Suppose the premise is a-satisfiable. Then, the clause set

$$\Lambda^a \cup \Phi^{\mathrm{c}} \cup \{\overline{K_1} \vee \overline{K_2} \vee \cdots \vee \overline{K_n}\} \tag{A.1}$$

is E-satisfiable. Consider any E-interpretation that satisfies (A.1) as well as the unit clause K_1. Observing that $K_1^a = K_1$, we can conclude that $\Lambda, \overline{K_1} \vdash \Phi, \square \cdot K_1, \dots, K_n$ is a-satisfiable. Now consider any E-interpretation that satisfies (A.1) but falsifies some ground instance of K_1. Since K_1 is variable disjoint with $\overline{K_2} \vee \cdots \vee \overline{K_n}$, such an interpretation must satisfy the latter clause. It follows that $\Lambda \vdash \Phi, \square \cdot K_2, \dots, K_n$ is a-satisfiable.

Close) Let $\Lambda \vdash \Phi, \square \cdot K_1, \dots, K_n$ be the premise sequent and $\Lambda \vdash \Phi, \square \cdot \emptyset$ the conclusion sequent. Since, trivially, $\Lambda^a \cup (\Phi, \square \cdot \emptyset)^{\mathrm{c}}$ is E-unsatisfiable, we must show that $\Lambda^a \cup (\Phi, \Box \cdot K_1, \ldots, K_n)^c$ is too. For that, it is enough to show that $\Lambda^a \cup \{\overline{K}_1 \vee \cdots \vee \overline{K}_n\}$ is E-unsatisfiable.[26]

By the rule's definition for every $i = 1, \ldots, n$ there is a $L_i \in \Lambda_{\geq}$ such that $K_i \gtrsim L_i$ if K_i is variable-disjoint with $\Gamma \setminus K_i$ and $K_i \sim L_i$ otherwise. This means that there exist $L_1, \ldots, L_n \in \Lambda_{\geq}$ such that the sets

$$\{L_1^a, K_1\},\ \{L_2^a, K_2\},\ \ldots,\ \{L_n^a, K_n\}$$

admit a simultaneous unifier σ. As a consequence, the set $\{L_1^a, \ldots,\ L_n^a,\ \overline{K}_1 \vee \cdots \vee \overline{K}_n\}$ is unsatisfiable, and hence E-unsatisfiable. To see that $\Lambda^a \cup \{\overline{K}_1 \vee \cdots \vee \overline{K}_n\}$ is E-unsatisfiable it is enough to observe that $\Lambda^a \models L_i^a$ for each $i = 1, \ldots, n$.

Compact) Immediate.

Assert) Let $\Lambda \vdash \Phi$ be the premise sequent and $\Lambda, L \vdash \Phi$ the conclusion sequent where $\Lambda^a \cup \Phi^c \models L^a$. Clearly, every E-interpretation that satisfies $\Lambda^a \cup \Phi^c$ is also a model of $\Lambda^a \cup \{L^a\} \cup \Phi^c$ which is the same as $(\Lambda \cup \{L\})^a \cup \Phi^c$.

Simp) Let $\Lambda \vdash \Phi,\ C \cdot \Gamma$ be the premise sequent and $\Lambda \vdash \Phi,\ C' \cdot \Gamma'$ the conclusion sequent where $\Lambda^a \cup (\Phi \cup \{C \cdot \Gamma\})^c \models (C' \cdot \Gamma')^c$. The argument is analogous to the previous case. □

Proposition 7.2 *For all sets Ψ of clauses, if Ψ has a refutation tree then Ψ is E-unsatisfiable.*

Proof. Let $\mathbf{T}_\Psi$ be a refutation tree of a set $\Psi = \{C_1, \ldots, C_n\}$ of parameter-free clauses. We prove below by structural induction that the root of any subtree of $\mathbf{T}_\Psi$ is a-unsatisfiable. This will entail in particular that $\neg v \vdash \Phi_0$, the root of $\mathbf{T}_\Psi$ itself, is a-unsatisfiable, where $\Phi_0 = \{C_1 \cdot \emptyset, \ldots, C_n \cdot \emptyset\}$. The claim will then follow from the immediate fact that the sequent $\neg v \vdash \Phi_0$ is a-unsatisfiable iff $\{\neg v\}^a \cup \Phi_0^c$, which coincides with Ψ, has no satisfying E-interpretation.

Let $\mathbf{T}$ be a subtree of $\mathbf{T}_\Psi$ and let N be its root. If $\mathbf{T}$ is a one-node tree, since $\mathbf{T}_\Psi$ is a refutation tree, N can only have the form $\Lambda \vdash \Phi, \Box \cdot \emptyset$, which is clearly a-unsatisfiable. If $\mathbf{T}$ has more than one node, we can assume by induction that all the children nodes of N are a-unsatisfiable. But then we can conclude that N is a-unsatisfiable as well by the contrapositive of Lemma 7.1. □

Lemma Appendix A.11 *Let $C \cdot \Gamma$ be a constrained clause. If $C \cdot \Gamma$ is redundant wrt. $\Lambda_j \vdash \Phi_j$, for some $j < \kappa$, then $C \cdot \Gamma$ is redundant wrt. $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$.*

Proof. The proof works in essentially the same way as in [3]. As a convenience, we denote the union of all context literals or all clauses of a branch $\mathbf{B} = (N_i)_{i<\kappa}$ by $\Lambda_{\mathbf{B}}^+ = \bigcup_{i<\kappa} \Lambda_i$ and $\Phi_{\mathbf{B}}^+ = \bigcup_{i<\kappa} \Phi_i$, respectively.

Suppose that $C \cdot \Gamma$ is redundant wrt. $\Lambda_j \vdash \Phi_j$. Since $\Lambda_{\mathbf{B}}^+ \supseteq \Lambda_j$ and $\Phi_{\mathbf{B}}^+ \supseteq \Phi_j$, Lemma Appendix A.5 implies that $C \cdot \Gamma$ is redundant wrt. $\Lambda_{\mathbf{B}}^+ \vdash \Phi_{\mathbf{B}}^+$ (observe that each

[26]By a light abuse of notation, we use each K_i's to denote both a rewrite literal of the form $(\neg)s_i \to t_i$ and its corresponding equational literal $(\neg)s_i \approx t_i$. Similarly for L_i later.

derivation rule can add only literals to a context that are non-contradictory with the context.) Now observe that every constrained clause in $\Phi^+_{\mathbf{B}} \setminus \Phi_{\mathbf{B}}$ has been deleted at some node of the branch **B**, which is only possible if it was redundant wrt. some $\Lambda_k \vdash \Phi_k$ with $k < \kappa$. Again using Lemma Appendix A.5, we see that every constrained clause in $\Phi^+_{\mathbf{B}} \setminus \Phi_{\mathbf{B}}$ is redundant wrt. $\Lambda^+_{\mathbf{B}} \vdash \Phi^+_{\mathbf{B}}$. Hence $\Phi_{\mathbf{B}}$ is obtained from $\Phi^+_{\mathbf{B}}$ by deleting redundant clauses, and $\Lambda_{\mathbf{B}}$ is obtained from $\Lambda^+_{\mathbf{B}}$ by deleting rewrite literals by Compact that satisfy the conditions of Lemma Appendix A.5. By using Lemma Appendix A.5 a third time, we conclude that $C \cdot \Gamma$ is redundant wrt. $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$. □

Lemma Appendix A.12 *Every* Deduce *inference that is redundant wrt.* $\Lambda_j \vdash \Phi_j$, *for some* $j < \kappa$, *is redundant wrt.* $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$.

Proof. Analogously to the proof of Lemma Appendix A.11 using Lemma Appendix A.6. □

Proposition 7.4 (Exhausted Branches are Saturated) *If* **B** *is an exhausted branch of a limit tree of a fair derivation then* $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$ *is saturated.*

Proof. Suppose **B** is an exhausted branch of a limit tree of some fair derivation. We have to show that every $\mathcal{ME}_{\mathrm{E}}$ inference with a mandatory derivation rule with premise $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$ is redundant wrt. $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$. We do this by assuming such an inference and carrying out a case analysis wrt. the derivation rule applied.

By Definition 7.3 there is no Close inference with premise $\Lambda_i \vdash \Phi_i$, for no $i < \kappa$, with a persistent closing clause and persistent closing literals. But then there is no Close inference with premise $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$ either. (Because if there were, for a large enough i there would be Close inference with premise $\Lambda_i \vdash \Phi_i$, which we excluded.) Thus there is nothing to show for Close.

If the derivation rule is Split then let $\Box \cdot \Gamma$ be the selected clause. There are only finitely many literals K, modulo renaming and modulo sign, that are more general than a given literal or set of literals such as Γ. The applicability conditions of the derivation rules makes sure that from some time k onwards, no more such literal K will be added to or removed from $\Lambda_k, \Lambda_{k+1}, \ldots$. (See Lemma 4.14 in [9] for a proof). We are given that $\Box \cdot \Gamma$ is persistent. Therefore suppose also $\Box \cdot \Gamma \in \Phi_k, \Phi_{k+1}, \ldots$, or choose k big enough. Together this shows that a Split inference with premise $\Lambda_i \vdash \Phi_i$ exists (i could be k or smaller). By Definition 7.3 then, the Split inference is redundant wrt. $\Lambda_j \vdash \Phi_j$, for some $j < \kappa$ with $j \geq i$. By redundancy, this means that the selected clause $\Box \cdot \Gamma$ is redundant wrt. $\Lambda_j \vdash \Phi_j$. Now use Lemma Appendix A.11 to conclude that $\Box \cdot \Gamma$ is redundant wrt. $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$, and so the Split inference with selected clause $\Box \cdot \Gamma$ is redundant wrt. $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$.

If the derivation rule is Deduce then by Definition 7.3 it is redundant wrt. $\Lambda_j \vdash \Phi_j$, for some $j \geq i$, and by Lemma Appendix A.12 it is redundant wrt. $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$. □

Theorem 7.5 (Completeness) *Let* Ψ *be a clause set and* **T** *be the limit tree of a fair derivation of* Ψ. *If* **T** *is not a refutation tree then* Ψ *is satisfiable; more specifically, for every exhausted branch* **B** *of* **T** *with limit sequent* $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$ *it holds* $\Lambda_{\mathbf{B}}, R_{\Lambda_{\mathbf{B}}} \models (\Phi_{\mathbf{B}})^{\Lambda_{\mathbf{B}}}$ *and* $R^{\star}_{\Lambda_{\mathbf{B}}} \models \Psi$.

Proof. Suppose $\mathbf{T}$ is not a refutation tree and let $\mathbf{B}$ an exhausted branch of $\mathbf{T}$. By Proposition 7.4 the limit sequent $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$ is saturated. It is easy to see that $\Lambda_{\mathbf{B}}$ is non-contradictory (the context in the initial sequent of the derivation is non-contradictory, and all derivation rules preserve this property.) By Theorem 6.6 then $(\Lambda_{\mathbf{B}}, R_{\Lambda_{\mathbf{B}}}) \models (\Phi_{\mathbf{B}})^{\Lambda_{\mathbf{B}}}$.

To show $R^{\star}_{\Lambda_{\mathbf{B}}} \models \Psi$, let $C \in \Psi$ be any clause from Ψ, and it suffices to show $R^{\star}_{\Lambda_{\mathbf{B}}} \models C$. By definition of derivation, $C \cdot \emptyset \in \Phi_1$. If $C \cdot \emptyset \in \Phi_{\mathbf{B}}$ then the second part of Theorem 6.6 gives $R^{\star}_{\Lambda_{\mathbf{B}}} \models C$ immediately. Otherwise assume $C \cdot \emptyset \notin \Phi_{\mathbf{B}}$. Hence $C \cdot \emptyset$ has been removed at some time $k < \kappa$ from the clause set Φ_k of the sequent $\Lambda_k \vdash \Phi_k$ by an application of the Simp rule. By definition of Simp, $C \cdot \emptyset$ is redundant wrt. $\Lambda_{k+1} \vdash \Phi_{k+1}$. By Lemma Appendix A.11, $C \cdot \emptyset$ is redundant wrt. $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$. Since $C \cdot \emptyset$ has an empty constraint, all its ground closures are relevant, and by Proposition Appendix A.4, all relevant closures wrt. $\Lambda_{\mathbf{B}}$ are redundant wrt. $\Lambda_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$, hence they are entailed wrt. $\Lambda_{\mathbf{B}}$ by clauses in $(\Phi_{\mathbf{B}})^{\Lambda_{\mathbf{B}}}$. With $\Lambda_{\mathbf{B}}, R_{\Lambda_{\mathbf{B}}} \models (\Phi_{\mathbf{B}})^{\Lambda_{\mathbf{B}}}$, the first part of the theorem, which is already proved, we get $\Lambda_{\mathbf{B}}, R_{\Lambda_{\mathbf{B}}} \models C \cdot \emptyset$. With the constraint being empty, $R^{\star}_{\Lambda_{\mathbf{B}}} \models C$ follows immediately. □

References

[1] Baader, F., Nipkow, T., 1998. Term Rewriting and All That. Cambridge University Press.

[2] Bachmair, L., Ganzinger, H., 1998. Chapter 11: Equational Reasoning in Saturation-Based Theorem Proving. In: Bibel, W., Schmitt, P. H. (Eds.), Automated Deduction. A Basis for Applications. Vol. I: Foundations. Calculi and Refinements. Kluwer Academic Publishers, pp. 353–398.

[3] Bachmair, L., Ganzinger, H., Waldmann, U., April 1994. Refutational theorem proving for hierarchic first-order theories. Applicable Algebra in Engineering, Communication and Computing 5 (3/4), 193–212.

[4] Baumgartner, P., 2000. FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure. In: McAllester, D. (Ed.), CADE-17 – The 17th International Conference on Automated Deduction. Vol. 1831 of Lecture Notes in Artificial Intelligence. Springer, pp. 200–219.

[5] Baumgartner, P., July 2007. Logical engineering with instance-based methods. In: Pfenning, F. (Ed.), CADE-21 – The 21st International Conference on Automated Deduction. Vol. 4603 of Lecture Notes in Artificial Intelligence. Springer, pp. 404–409.

[6] Baumgartner, P., Fuchs, A., Tinelli, C., 2006. Implementing the model evolution calculus. International Journal of Artificial Intelligence Tools 15 (1), 21–52.

[7] Baumgartner, P., Tinelli, C., 2003. The Model Evolution Calculus. In: Baader, F. (Ed.), CADE-19 – The 19th International Conference on Automated Deduction. Vol. 2741 of Lecture Notes in Artificial Intelligence. Springer, pp. 350–364.

[8] Baumgartner, P., Tinelli, C., 2005. The model evolution calculus with equality. In: Nieuwenhuis, R. (Ed.), CADE-20 – The 20th International Conference on Automated Deduction. Vol. 3632 of Lecture Notes in Artificial Intelligence. Springer, pp. 392–408.

[9] Baumgartner, P., Tinelli, C., 2008. The Model Evolution Calculus as a First-Order DPLL Method. Artificial Intelligence 172 (4-5), 591–632.

[10] Baumgartner, P., Waldmann, U., July 2009. Superposition and model evolution combined. In: Schmidt, R. (Ed.), CADE-22 – The 22nd International Conference on Automated Deduction. Vol. 5663 of Lecture Notes in Artificial Intelligence. Springer, pp. 17–34.

[11] Bernays, P., Schönfinkel, M., 1928. Zum Entscheidungsproblem der Mathematischen Logic. Mathematische Annalen 99, 342–372.

[12] Billon, J.-P., 1996. The Disconnection Method. In: Miglioli, P., Moscato, U., Mundici, D., Ornaghi, M. (Eds.), Theorem Proving with Analytic Tableaux and Related Methods. No. 1071 in Lecture Notes in Artificial Intelligence. Springer, pp. 110–126.

[13] Claessen, K., October 2005. Equinox, a new theorem prover for full first-order logic with equality, presentation at Dagstuhl Seminar 05431 on Deduction and Applications.

[14] Davis, M., Logemann, G., Loveland, D., Jul. 1962. A machine program for theorem proving. Communications of the ACM 5 (7), 394–397.

[15] de Nivelle, H., 1999. The Bliksem theorem prover.

[16] de Nivelle, H., Meng, J., 2006. Geometric resolution: A proof procedure based on finite model search. In: Furbach, U., Shankar, N. (Eds.), IJCAR. Vol. 4130 of Lecture Notes in Computer Science. Springer, pp. 303–317.

[17] Ganzinger, H., Korovin, K., 2003. New directions in instantiation-based theorem proving. In: Proc. 18th IEEE Symposium on Logic in Computer Science,(LICS'03). IEEE Computer Society Press, pp. 55–64.

[18] Ganzinger, H., Korovin, K., 2004. Integrating equational reasoning into instantiation-based theorem proving. In: Computer Science Logic (CSL'04). Vol. 3210 of Lecture Notes in Computer Science. Springer, pp. 71–84.

[19] Hurd, J., 2003. First-order proof tactics in higher-order logic theorem provers. In: Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003-212448 in NASA Technical Reports. pp. 56–68.

[20] Korovin, K., 2008. iProver — an instantiation-based theorem prover for first-order logic (system description). In: IJCAR '08: Proceedings of the 4th international joint conference on Automated Reasoning. Springer-Verlag, Berlin, Heidelberg, pp. 292–298.

[21] Korovin, K., 2009. Instantiation-based automated reasoning: From theory to practice. In: Schmidt, R. A. (Ed.), Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings. Vol. 5663 of Lecture Notes in Computer Science. Springer, pp. 163–166.

[22] Letz, R., Stenz, G., 2002. Integration of Equality Reasoning into the Disconnection Calculus. In: Egly, U., Fermüller, C. G. (Eds.), TABLEAUX. Vol. 2381 of Lecture Notes in Computer Science. Springer, pp. 176–190.

[23] McCune, W., 1992. Experiments with Discrimination-Tree Indexing and Path Indexing for Term Retrieval. Journal of Automated Reasoning 9 (2), 147–167.

[24] McCune, W., 2003. OTTER 3.3 Reference Manual. Argonne National Laboratory, Argonne, Illinois.

[25] Nieuwenhuis, R., Rubio, A., 1995. Theorem Proving with Ordering and Equality Constrained Clauses. Journal of Symbolic Computation 19, 321–351.

[26] Nieuwenhuis, R., Rubio, A., 2001. Paramodulation-based theorem proving. In: [32], pp. 371–443.

[27] Pelletier, F. J., Sutcliffe, G., Suttner, C., 2002. The Development of CASC. AI Communications 15 (2-3), 79–90.

[28] Pelzer, B., Wernhard, C., 2007. System Description: E-KRHyper. In: Pfenning, F. (Ed.), Automated Deduction - 21st International Conference on Automated Deduction (CADE-21). Vol. 4603 of Lecture Notes in Computer Science. Springer, pp. 508–513.

[29] Plaisted, D. A., Zhu, Y., 2000. Ordered Semantic Hyper Linking. Journal of Automated Reasoning 25 (3), 167–217.

[30] Ramsey, F. P., 1930. On a problem in formal logic. Proceedings of the London Mathematical Society 30, 264–286.

[31] Riazanov, A., Voronkov, A., 2002. The design and implementation of Vampire. AI Communications 15 (2-3), 91–110.

[32] Robinson, J. A., Voronkov, A. (Eds.), 2001. Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press.

[33] Schulz, S., 2002. E - a brainiac theorem prover. AI Communications 15 (2-3), 111–126.

[34] Stickel, M. E., Waldinger, R. J., Chaudhri, V. K., 2000. A guide to SNARK. Technical report. SRI International.

[35] Sutcliffe, G., Suttner, C., 1998. The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning 21 (2), 177–203.

[36] Sutcliffe, G., Suttner, C., 2006. The State of CASC. AI Communications 19 (1), 35–48.

[37] Weidenbach, C., Schmidt, R., Hillenbrand, T., Rusev, R., Topic, D., 2007. System description: Spass version 3.0. In: In: Pfenning, F. (Ed.), Automated Deduction - 21st International Conference on Automated Deduction (CADE-21). Vol. 4603 of Lecture Notes in Computer Science. Springer, pp. 514–520.