Model Evolution with Equality Modulo Built-in Theories

Peter Baumgartner[1] and Cesare Tinelli[2]

[1] NICTA* and Australian National University, Canberra, Australia
[2] The University of Iowa, USA

Abstract. Many applications of automated deduction require reasoning modulo background theories, in particular some form of integer arithmetic. Developing corresponding automated reasoning systems that are also able to deal with quantified formulas has recently been an active area of research. We contribute to this line of research and propose a novel instantiation-based method for a large fragment of first-order logic with equality modulo a given complete background theory, such as linear integer arithmetic. The new calculus is an extension of the Model Evolution Calculus with Equality, a first-order logic version of the propositional DPLL procedure, including its ordering-based redundancy criteria. We present a basic version of the calculus and prove it sound and (refutationally) complete under certain conditions.[3]

1 Introduction

Many applications of automated deduction require reasoning modulo background theories, in particular some form of integer arithmetic. Developing sophisticated automated reasoning systems that are also able to deal with quantified formulas has recently been an active area of research [6, 8, 10, 3, 1]. We contribute to this line of research and propose a novel instantiation-based method for a large fragment of first-order logic with equality modulo a given complete background theory, such as linear integer arithmetic. The new calculus, $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$, is an extension of the Model Evolution calculus with equality [4], a first-order logic version of the propositional DPLL procedure, including its ordering-based redundancy criteria as recently developed in [5]. At the same time, $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$ is a generalization wrt. these features of the earlier $\mathcal{ME}(\mathrm{LIA})$ calculus [3].

Instantiation based methods, including Model Evolution, have proven to be a successful alternative to classical, saturation-based automated theorem proving methods. This then justifies attempts to develop theory-reasoning versions of them, even if their input logic or their associated decidability results are not new. As one of these extensions, we think $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$ is relevant in particular for its versatility since it combines powerful techniques for first-order equational logic with equality, based on an adaptation of the Bachmair-Ganzinger theory of superposition, with a black-box theory reasoner. In this sense, $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$ is similar to the hierarchic superposition calculus [1, 2].

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

[3] The full version of this paper, which includes all proofs, is available as a NICTA research report from `http://www.nicta.com.au/research/research_publications`.

$\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$ also relates to DPLL(T) [9], a main approach for theorem proving modulo theories. DPLL(T) is essentially limited to the ground case and resorts to incomplete or inefficient heuristics to deal with quantified formulas [7, e.g.]. In fact, addressing this intrinsic limitation by lifting DPLL(T) to the first-order level is one of the main motivations for $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$, much like lifting the propositional DPLL procedure to the first-order level while preserving its good properties was the main motivation for Model Evolution.

One possible application of $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$ is in finite model reasoning. For example, the three formulas $1 \leq a \leq 100$, $P(a)$ and $\neg P(x) \leftarrow 1 \leq x \wedge x \leq 100$ together are unsatisfiable, when a is a constant and T is a theory of the integers. Finite model finders, e.g., need about 100 steps to refute the clause set, one for each possible value of a. Our calculus, on the other hand, can reason directly with integer intervals and allows a refutation in $O(1)$ steps. See Section 7 for further discussion of how this is achieved, variations of the example, and considerations on $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$ as a decision procedure.

The most promising applications of $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$ could be in software verification. Quite frequently, proof obligations arise there that require quantified formulas to define data structures with *specific* properties, e.g., ordered lists or ordered arrays, and to prove that these properties are preserved under certain operations, e.g., when an element is inserted at an appropriate position. In the array case, one could define ordered arrays with an axiom of the form "for all i, j with $0 \leq i < j \leq m$, $a[i] \leq a[j]$", where i and j are variables and m is a parameter, all integer-valued. Our calculus natively supports parameters like m and is well suited to reason with bounded quantification like the one above. In general, parameters like m must be additionally constrained to a finite domain for the calculus to be effective, see again Section 7.

The general idea behind our calculus with respect to theory reasoning is to use *rigid* variables to represent individual, but not yet known, elements of the background domain, and instantiate them as needed to carry the derivation forward. As a simple example without parameters, consider the clauses $f(x) \approx g(x) \leftarrow x > 5$ and $\neg(f(y + y) \approx g(8))$. These clauses will be refuted, essentially, by checking satisfiability of the set $\{v_1 = v_2 + v_2, v_1 > 5, v_1 = 8\}$ of constraints over rigid variables and (ordered) paramodulation inferences for reasoning with the equations in these clauses.

2 Preliminaries

We work in the context of standard many-sorted logic with first-order signatures comprised of sorts and operators (i.e., function symbols and predicate symbols) of given arities over these sorts. We rely on the usual notions of structure, (well-sorted) term/formula, satisfiability, and so on. If Σ is a sorted signature and X a set of sorted variables we will call $\Sigma(X)$-*term (resp. -formula)* a well-sorted term (resp. formula) built with symbols from Σ and variables from X. The notation $\Sigma(X_1, X_2)$ is a shorthand for $\Sigma(X_1 \cup X_2)$.

Syntax. For simplicity, we consider here only signatures with at most two sorts: a *background* sort B and a *foreground* sort F. We assume a *background signature* Σ_{B} having B as the only sort and an at most countable set of operators that includes an (infix) equality predicate symbol = of arity $\mathsf{B} \times \mathsf{B}$. We will write $s \neq t$ as an abbreviation of $\neg(s = t)$. We fix an infinite set X_{B} of B-*variables*, variables of sort B.

We assume a complete first-order *background theory* T of signature Σ_{B} all of whose models interpret = as the identity relation. Since T is complete and we do not extend Σ_{B} in any essential way with respect to T, we can specify it with no loss of generality simply as a Σ_{B}-structure. We call the set $|\mathsf{B}|$ that T associates to the sort B the *background domain*. We assume, again with no loss of generality, that $|\mathsf{B}|$ is at most countably infinite and all of its elements are included in Σ as B-constant symbols.Our running example for T will be the theory of linear integer arithmetic (LIA). For that example, Σ_{B}'s operators are $\leq$, $+$ and all the integer constants, all with the expected arities, T is the structure of the integer numbers with those operators, and $|\mathsf{B}| = \{0, \pm 1, \pm 2, \ldots\}$.

We will consider formulas over an expanded signature $\Sigma_{\mathrm{B}}^{\Pi}$ and expanded set of variables $X_{\mathrm{B}} \cup V$ where $\Sigma_{\mathrm{B}}^{\Pi}$ is obtained by adding to Σ_{B} an infinite set Π of *parameters*, free constants of sort B, and V is a set of B-variables not in X_{B}, which we call *rigid variables*. The function and predicate symbols of $\Sigma_{\mathrm{B}}^{\Pi}$ are collectively referred to as the *background operators*. We call *(background) constraint* any formula in the closure of the set of $\Sigma_{\mathrm{B}}^{\Pi}(X_{\mathrm{B}}, V)$-atoms under conjunction, negation and existential quantification of variables.[4] A *closed constraint* is a constraint with no free variables (but possibly with rigid variables).

Note that rigid variables always occur free in a constraint. We will always interpret distinct rigid variables in a constraint as distinct elements of $|\mathsf{B}|$. Intuitively, in the calculus presented here, a rigid variable v will stand for a specific, but unspecified, background domain element, and will be introduced during proof search similarly to rigid variables in free-variable tableaux calculi. In contrast, parameters will be free constants in input formulas, standing for arbitrary domain values.

The *full signature* Σ for our calculus is obtained by adding to $\Sigma_{\mathrm{B}}^{\Pi}$ the foreground sort F, function symbols of given arities over B and F, and one infix equality predicate symbol, $\approx$, of arity $\mathsf{F}\times\mathsf{F}$. The new function symbols and $\approx$ are the *foreground operators*. As usual, we do not consider additional foreground predicate symbols because they can be encoded as function symbols, e.g., an atom of the form $P(t_1, \ldots, t_n)$ can be encoded as $P(t_1, \ldots, t_n) \approx \mathsf{tt}$, where tt is a new, otherwise unused, foreground constant. For convenience, however, in examples we will often write the former and mean the latter. Since $\approx$ will always denote a congruence relation, we will identify $s \approx t$ with $t \approx s$.

Let X_{F} be an infinite set of F-*variables*, variables of sort F, disjoint from X_{B} and V, and let $X = X_{\mathrm{B}} \cup X_{\mathrm{F}}$. When we say just "variable" we will always mean a variable in X, not a rigid variable.

The calculus takes as input $\Sigma(X)$-formulas of a specific form, defined later, and manipulate more generally $\Sigma(X, V)$ formulas, i.e., formulas possibly containing rigid variables. We use, possibly with subscripts, the letters $\{x, y\}, \{u, v\}$, $\{a, b\}$, and $\{f, e\}$ to denote respectively regular variables (those in X), rigid variables, parameters, and foreground function symbols.

To simplify the presentation here, *we restrict the return sort of all foreground function symbols to be* F . This is a true restriction for non-constant function symbols (foreground constant symbols of sort B can be supplied as parameters instead). For example,

[4] The calculus needs a decision procedure only for the validity of the $\forall\exists$-fragment over the class of constraints used in input formulas. When such formulas contain no parameters, a decision procedure for the $\exists$-fragment is sufficient.

if Σ is the signature of lists of integers, with T being again LIA and F being the list sort, our logic allows formulas like $cdr(cons(x, y)) \approx y$ but not $car(cons(x, y)) \approx x$, as *car* would be integer-sorted. To overcome this limitation somewhat, one could turn *car* into a predicate symbol and use $car(cons(x, y), x)$ instead, together with the (universal) functionality constraint $\neg car(x, y) \vee \neg car(x, z) \leftarrow y \neq z$. This solution is however approximate as it does not include a totality restriction on the new predicate symbols.

A *term* is a (well-sorted) $\Sigma(X, V)$-term, a *formula* is a (well-sorted) $\Sigma(X, V)$-formula. A *foreground term* is a term with no operators from $\Sigma_{\mathrm{B}}^{\Pi}$. Foreground atoms, literals, and formulas are defined analogously. An *ordinary foreground clause* is a multiset of foreground literals, usually written as a disjunction. A *background term* is a (well-sorted) $\Sigma_{\mathrm{B}}^{\Pi}(X_{\mathrm{B}}, V)$-term. Note that background terms are always B-sorted and vice versa. Foreground terms are made of foreground symbols, variables and rigid variables; they are all F-sorted unless they are rigid variables. A *ground term* is a term with no variables and no rigid variables. A *Herbrand term* is a ground term whose only background subterms are background domain elements. Intuitively, Herbrand terms do not contain symbols that need external evaluation, i.e., they contain no parameters, no variables, and no rigid variables. For example, $f(e, 1)$ and 1 are Herbrand terms, but $f(v, 1)$ and $f(a, 1)$ are not.

A *substitution* is a mapping σ from variables to terms that is sort respecting, that is, maps each variable $x \in X$ to a term of the same sort. We write substitution application in postfix form and extend the notation to (multi)sets S of terms or formulas as expected, that is, $S\sigma = \{F\sigma \mid F \in S\}$. The *domain* of a substitution σ is the set $\mathrm{dom}(\sigma) = \{x \mid x \neq x\sigma\}$. We work with substitutions with finite domains only. A *Herbrand substitution* is a substitution that maps every variable to a Herbrand term. We denote by $fvar(F)$ the set of non-rigid variables that occur free in F, where F is a term or formula.

Semantics. An *interpretation* I is any Σ-structure augmented to include an injective, possibly partial, mapping from the set V of rigid variables to the domain of B in I. We will be interested primarily in Herbrand interpretations, defined below.

Definition 2.1 (Herbrand interpretations). *A* (T-based) Herbrand interpretation *is any interpretation I that* (*i*) *is identical to T over the symbols of Σ^{B},* (*ii*) *interprets every foreground n-ary function symbol f as itself, i.e., $f^I(d_1, \ldots, d_n) = f(d_1, \ldots, d_n)$ for every tuple $(d_1, \ldots, d_n)$ of domain elements from the proper domain, and* (*iii*) *interprets $\approx$ as a congruence relation on F-sorted Herbrand terms.*[5]

A *(parameter) valuation* π is a mapping from Π to $|\mathsf{B}|$. An *assignment* α is an *injective* mapping from a (finite or infinite) subset of V to $|\mathsf{B}|$. The range of α is denoted by $\mathrm{ran}(\alpha)$. Since T is fixed, a Herbrand interpretation I is completely characterized by a congruence relation on the Herbrand terms, a valuation π and an assignment α.

An assignment α is *suitable* for a formula or set of formulas F if its domain includes all the rigid variables occurring in F. Since all the elements of $|\mathsf{B}|$ are constants of Σ_{B} we will often treat assignments and valuations similarly to substitutions. For any Herbrand interpretation I, valuation π and assignment α, we denote by $I[\pi]$ the interpretation that agrees with π on the meaning of the parameters (that is, $a^I = a\pi$ for all $a \in \Pi$) and is

[5] Note that Condition (*iii*) is well defined because, by Condition (*ii*), the interpretation of the sort F is the set of all F-sorted Herbrand terms.

otherwise identical to I; we denote by $I[\alpha]$ the interpretation that agrees with α on the meaning of the rigid variables in α's domain and is otherwise identical to I. We write $I[\pi, \alpha]$ as a shorthand for $I[\pi][\alpha]$.

The symbols I, α and π we will always denote respectively Herbrand interpretations, assignments and valuations. Hence, we will often use the symbols directly, without further qualification. We will do the same for other selected symbols introduced later. Also, we will often implicitly assume that α is suitable for the formulas in its context.

Definition 2.2 (Satisfaction of constraints). *Let c be a closed constraint. For all π and all α suitable for c, the pair (π, α)* satisfies *c, written as $(\pi, \alpha) \models c$, if $T \models c\pi\alpha$ in the standard sense.*[6] *If α is suitable for a set Γ of closed constraints, (π, α)* satisfies *Γ, written $(\pi, \alpha) \models \Gamma$, iff (π, α) satisfies every $c \in \Gamma$.*

The set Γ above is *satisfiable* if $(\pi, \alpha) \models \Gamma$, for some π and α. Since constraints contain no foreground symbols, for any interpretation $I[\pi, \alpha]$, $I[\pi, \alpha] \models c$ iff $(\pi, \alpha) \models c$.

The satisfiability of arbitrary closed constraints, which may contain rigid variables, reduces in a straightforward way to the satisfiability of Σ_{B}-constraints without rigid variables, and so can be decided by any decision procedure for the latter. It requires only to read parameters and rigid variables as variables in the usual sense, and to conjoin disequality constraints $u \neq v$ for all distinct rigid variables u and v that occur in c.

Finally, we assume a reduction ordering $\succ$ that is total on the Herbrand terms.[7] We also require that $\succ$ is stable under assignments, i.e., if $s \succ t$ then $s\alpha \succ t\alpha$, for every suitable assignment α for s and t. The ordering $\succ$ is extended to literals over Herbrand terms by identifying a positive literal $s \approx t$ with the multiset $\{s, t\}$, a negative literal $\neg(s \approx t)$ with the multiset $\{s, s, t, t\}$, and using the multiset extension of $\succ$. Multisets of literals are compared by the multiset extension of that ordering, also denoted by $\succ$.

3 Contexts and Constrained clauses

Our calculus maintains two data structures for representing Herbrand interpretations: a *foreground context* Λ, a set of foreground literals, for the foreground operators; and a *background context* Γ, a set of closed constraints, for valuations and assignments. The elements of Λ are called *context literals*. We identify every foreground context Λ with its closure under renaming of (regular) variables, and assume it contains a pseudo-literal of the form $\neg x$. A foreground literal K is *contradictory with* Λ if $\overline{K} \in \Lambda$, where $\overline{K}$ denotes the complement of K. Λ itself is *contradictory* if it contains a literal that is contradictory with Λ. We will work only with non-contradictory contexts.

For any foreground literals K and L, we write $K \gtrsim L$ iff L is an instance of K, i.e., iff there is a substitution σ such that $K\sigma = L$. We write $K \sim L$ iff K and L are variants, equivalently, iff $K \gtrsim L$ and $L \gtrsim K$. We write $K \gnsim L$ iff $K \gtrsim L$ but $L \not\gtrsim K$.

[6] Observe that the test $T \models c\pi\alpha$ is well formed because $c\pi\alpha$ is closed and contains neither parameters nor rigid variables.

[7] A *reduction ordering* is a strict, well-founded ordering on terms that is compatible with contexts, i.e., $s \succ t$ implies $f[s] \succ f[t]$, and stable under substitutions, i.e., $s \succ t$ implies $s\sigma \succ t\sigma$.

Definition 3.1 (Productivity). *Let K, L be foreground literals. We say that K* produces *L in Λ if (i) $K \gtrsim L$, and (ii) there is no $K' \in \Lambda$ such that $K \gtrsim \overline{K'} \gtrsim L$.*

Since foreground contexts contain the pseudo-literal $\neg x$, it is not difficult to see that Λ produces at least one of K and $\overline{K}$, for every Λ and literal K.

The calculus works with *constrained clauses*, expressions of the form $C \leftarrow R \cdot c$ where R is a multiset of foreground literals, the set of *context restrictions*, C is an ordinary foreground clause, and c is a (background) constraint with $fvar(c) \subseteq fvar(C) \cup fvar(R)$. When C is empty we write it as $\Box$. When R is empty, we write the constrained clause more simply as $C \leftarrow c$. The calculus takes as input only clauses of the latter form, hence we call such clauses *input constrained clauses*. Below we will often speak of (input) clauses instead of (input) constrained clauses when no confusion can arise.

We can turn any expression of the form $C \leftarrow c$ where C is an arbitrary ordinary Σ-clause and c a constraint into an input clause by abstracting out offending subterms from C, moving them to the constraint side of $\leftarrow$, and existentially quantifying variables in the constraint side that do not occur in the clause side. For example, $P(a, v, x + 5) \leftarrow x > v$ becomes $P(x_1, v, x_2) \leftarrow \exists x\,(x > v \wedge x_1 = a \wedge x_2 = x + 5)$. As will be clear later, this transformation preserves the semantics of the original expression.

The variables of input clauses are implicitly universally quantified. Because the background domain elements (such as, e.g., $0, 1, -1, \ldots$) are also background constants, we can define the semantics of input clauses in terms of Herbrand interpretations. To do that, we need one auxiliary definition first.

If γ is a Herbrand substitution and $C \leftarrow c$ an input clause, the clause $(C \leftarrow c)\gamma = C\gamma \leftarrow c\gamma$ is a *Herbrand instance* of $C \leftarrow c$. For example, $(P(v, x, y) \leftarrow x > a)\gamma$ is $P(v, 1, f(1, e)) \leftarrow 1 > a$ if $\gamma = \{x \mapsto 1, y \mapsto f(1, e), \ldots\}$. A Herbrand instance $C \leftarrow c$ can be evaluated directly by an interpretation $I[\alpha]$, for suitable α: we say that $I[\alpha]$ *satisfies* $C \leftarrow c$, written $I[\alpha] \models C \leftarrow c$ if $I[\alpha] \models C \vee \neg c$. For input clauses $C \leftarrow c$ we say that $I[\alpha]$ *satisfies* $C \leftarrow c$ iff $I[\alpha]$ satisfies every Herbrand instance of $C \leftarrow c$.

Definition 3.2 (Satisfaction of sets of formulas). *Let Δ be a set of input clauses and closed constraints. We say that $I[\alpha]$* satisfies *Δ, written as $I[\alpha] \models \Delta$, if $I[\alpha] \models F$, for every $F \in \Delta$.*

We say that *Δ is satisfiable* if some $I[\alpha]$ satisfies F. Let G be an input clause or closed constraint. We say that *Δ entails* G, written as $\Delta \models G$, if for every suitable assignment α for Δ and G, every interpretation $I[\alpha]$ that satisfies Δ also satisfies G.

The definition of satisfaction of general constrained clauses $C \leftarrow R \cdot c$, with a non-empty restriction R, is more complex because in our completeness argument for the calculus C is evaluated *semantically*, with respect to Herbrand interpretations induced by a context, whereas R is evaluated syntactically, with respect to productivity in a context. Moreover, constrained clause satisfaction is not definable purely at the ground level but requires a suitable notion of *Herbrand closure*.

Definition 3.3 (Herbrand closure). *Let γ be a Herbrand substitution. The pair $(C \leftarrow R \cdot c,\ \gamma)$ is a* Herbrand closure (of $C \leftarrow R \cdot c$).

Context restrictions are evaluated in terms of productivity by applying an assignment to the involved rigid variables first. To this end, we will use *evaluated contexts* $\Lambda\alpha = \{K\alpha \mid$

$K \in \Lambda\}$. By the injectivity of α, the notions above on contexts apply *isomorphically* after evaluation by α. For instance, K produces L in Λ iff $K\alpha$ produces $L\alpha$ in $\Lambda\alpha$.

Definition 3.4 (Satisfaction of context restrictions). *Let R be a set of context restrictions and γ a Herbrand substitution. The pair* (Λ, α) satisfies (R, γ)*, written as* $(\Lambda, \alpha) \models (R, \gamma)$*, if*

(i) $R\alpha\gamma$ contains no trivial literals*, of the form $t \approx t$ or $\neg(t \approx t)$, and for every $l \approx r \in R\alpha\gamma$, if $l \succ r$ then l is not a variable, and*
(ii) for every $K \in R\alpha$ there is an $L \in \Lambda\alpha$ that produces both K and $K\gamma$ in $\Lambda\alpha$.

Point (*i*) makes paramodulation into variables unnecessary for completeness in the calculus.

Definition 3.5 (Satisfaction of Herbrand closures). *A triple* (Λ, α, I) satisfies $(C \leftarrow R \cdot c,\ \gamma)$*, written as* $(\Lambda, \alpha, I) \models (C \leftarrow R \cdot c,\ \gamma)$*, iff* $(\Lambda, \alpha) \not\models (R, \gamma)$ *or* $I \models (C \leftarrow c)\gamma$.

We will use Definition 3.5 always with $I = I[\alpha]$. The component Λ in the previous definition is irrelevant for input clauses (where $R = \emptyset$), and satisfaction of Herbrand closures and Herbrand instances coincide then. Formally, $(\Lambda, \alpha, I[\alpha]) \models (C \leftarrow \emptyset \cdot c,\ \gamma)$ if and only if $I[\alpha] \models (C \leftarrow c)\gamma$.

In our soundness arguments for the calculus a constrained clause $C \leftarrow R \cdot c$ will stand for the Σ-formula $C \vee (\bigvee_{L \in R} \overline{L}) \vee \neg c$. We call the latter the *clause form* of $C \leftarrow R \cdot c$ and denote it by $(C \leftarrow R \cdot c)^{\mathrm{c}}$. If Φ is a set of clauses, $\Phi^{\mathrm{c}} = \{F^{\mathrm{c}} \mid F \in \Phi\}$.

4 Core Inference Rules

The calculus works on sequents of the form $\Lambda \cdot \Gamma \ \vdash\ \Phi$, where $\Lambda \cdot \Gamma$ is a context and Φ is a set of constrained clauses. It has five core inference rules: Ref, Para, Pos-Res, Split and Close. In their description, if S is a set and a is an element, we will write S, a as an abbreviation of $S \cup \{a\}$.

The first two inference rules perform equality reasoning at the foreground level.

$$\textsf{Ref}\ \frac{\Lambda \cdot \Gamma \ \vdash\ \Phi}{\Lambda \cdot \Gamma \ \vdash\ \Phi, (C \leftarrow R \cdot c)\sigma}$$

if Φ contains a clause $\neg(s \approx t) \vee C \leftarrow R \cdot c$, the *selected clause*, and σ is an mgu of s and t. The new clause in the conclusion is the *derived clause*.

The next inference rule is a variant of ordered paramodulation.

$$\textsf{Para}\ \frac{\Lambda \cdot \Gamma \ \vdash\ \Phi}{\Lambda \cdot \Gamma \ \vdash\ \Phi, (L[r] \vee C \leftarrow (R \cup \{l \approx r\}) \cdot c)\sigma}$$

if $l \approx r \in \Lambda$ and Φ contains a clause $L[s] \vee C \leftarrow R \cdot c$, the *selected clause*, such that (*i*) σ is an mgu of l and s, (*ii*) s is neither a variable nor a rigid variable, (*iii*) $r\sigma \not\succeq l\sigma$, and (*iv*) $l \approx r$ produces $(l \approx r)\sigma$ in Λ. The context literal $l \approx r$ is the *selected context equation*, and the new clause in the conclusion is the *derived clause*.

We can afford to not paramodulate into rigid variables s, as these are B-sorted, and the resulting unifier with (an F-sorted variable) l would be ill-sorted. The equation $l \approx r$ is added to R to preserve soundness.

For example, if $\Lambda = \{f(x, y, e) \approx x\}$ then the clause $P(f(x, e, y)) \vee y \approx e \leftarrow \emptyset \cdot x > 5$ paramodulates into $P(x) \vee e \approx e \leftarrow f(x, e, e) \approx x \cdot x > 5$.

Let $C = L_1 \vee \cdots \vee L_n$ be an ordinary foreground clause with $n \geq 0$. We say that a substitution σ is a *context unifier of C against Λ* if there are literals $K_1, \ldots, K_n \in \Lambda$ such that σ is a simultaneous most general unifier of the sets $\{K_1, \overline{L}_1\}, \ldots, \{K_n, \overline{L}_n\}$. We say that σ is *productive* iff K_i produces $\overline{L}_i\sigma$ in Λ, for all $i = 1, \ldots, n$.

For any ordinary foreground clause, let $\overline{C} = \{\overline{L} \mid L \in C\}$.

$$\textsf{Pos-Res}\ \frac{\Lambda \cdot \Gamma \vdash \Phi}{\Lambda \cdot \Gamma \vdash \Phi, (\Box \leftarrow (R \cup \overline{C}) \cdot c)\sigma}$$

if Φ contains a clause of the form $C \leftarrow R \cdot c$, the *selected clause*, such that (*i*) $C \neq \Box$ and C consists of positive literals only, and (*ii*) σ is a productive context unifier of C against Λ. The new clause in the conclusion is the *derived clause*.

For example, if $\Lambda = \{\neg P(e)\}$, from $f(x, y, z) \approx g(y) \vee P(x) \leftarrow \emptyset \cdot y > 5$ one gets $\Box \leftarrow \{\neg(f(e, y, z) \approx g(y)), \neg P(e)\} \cdot y > 5$. (Recall that Λ implicitly contains $\neg x$.)

Intuitively, Pos-Res is applied when all literals in the ordinary clause part of a clause have been sufficiently processed by the equality inference rules Para and Ref and turns them into context restrictions. Deriving an empty constrained clause this way does not necessary produce a contradiction, as the clause could be satisfied, in an interpretation that falsifies its context restriction or falsifies its constraint. The Split rule below considers this possibility.

The rule has side conditions that treat context literals as constrained clauses. Formally, let $\Lambda^{(e,n)} = \{K^{(e,n)} \leftarrow \top \mid K \in \Lambda\}$ be the *clause form of* Λ, where $K^{(e,n)}$ is the context literal obtained from K by replacing every foreground variable by a fixed foreground constant e and replacing every background variable by a fixed background domain element n. We say that $(C \leftarrow R \cdot c)\delta$ is a *domain instance* of a clause $C \leftarrow R \cdot c$ if δ moves every B-sorted variable of $\mathit{fvar}(c)$ to a rigid variable and does not move the other variables of $\mathit{fvar}(c)$.

$$\textsf{Split}\ \frac{\Lambda \cdot \Gamma \vdash \Phi}{\Lambda, \overline{K} \cdot \Gamma, c \vdash \Phi \qquad \Lambda, K \cdot \Gamma^\star \vdash \Phi}$$

if there is a domain instance $(\Box \leftarrow R \cdot c)$ of some clause in Φ such that (*i*) $K \in R$ and neither $\overline{K}$ nor K is contradictory with Λ, (*ii*) for every $L \in R$, Λ produces L, (*iii*) $\Gamma \cup \{c\}$ is satisfiable, and (*iv*) $\Gamma^\star$ is any satisfiable background context such that $\Gamma \cup \{c\} \subseteq \Gamma^\star$ and $(\Lambda \cup \overline{K})^{(e,n)} \cup \Phi^{\mathrm{c}} \cup \Gamma^\star$ is not satisfiable, if such a $\Gamma^\star$ exists, or else $\Gamma^\star = \Gamma \cup \{c\}$. The clause $\Box \leftarrow R \cdot c$ is the *selected clause*, and the literal $\overline{K}$ is the *split literal*.

For example, if $\Lambda = \{\neg P(e)\}$ and Φ contains $\Box \leftarrow \{\neg(f(e, y, z) \approx g(y)), \neg P(e) \cdot y > 5$, where y is B-sorted and the sort of z is irrelevant, the domain instance could be $\Box \leftarrow \{\neg(f(e, v_1, z) \approx g(v_1)), \neg P(e)\} \cdot v_1 > 5$, and the split literal then is $f(e, v_1, z) \approx g(v_1)$.

The set Φ can also be seen to implicitly contain with each clause all its domain instances, and taking one of those as the selected clause for Split.

While splitting is done in a complementary way, as in earlier $\mathcal{ME}$ calculi, background contexts are global to derivations. Moreover, all constraints added to Γ in the course of the further derivation of the left branch need to be present in the right branch as well. This is modeled by Condition (*iv*). The branch $\Gamma^\star$ can be obtained in a constructive way by trying to extend the left branch to a refutation sub-tree, which, if successful, gives the desired $\Gamma^\star$. If not successful, no matter if finite or infinite, the input clause set is satisfiable, and the derivation need not return to the right branch anyway. We remark that extending background constraints, as done by Split (and Close and Restrict below) causes no soundness problems, as our soundness theorem applies relative to derived background contexts only. See Section 7 for details and how soundness in the usual sense is recovered.

$$\textsf{Close}\ \frac{\Lambda \cdot \Gamma \ \vdash\ \Phi}{\Lambda \cdot \Gamma, c \ \vdash\ \Phi, (\square \leftarrow \emptyset \cdot \top)}$$

if Φ contains a clause $\square \leftarrow R \cdot c$ such that (*i*) $R \subseteq \Lambda$, and (*ii*) $\Gamma \cup \{c\}$ is satisfiable. The clause $\square \leftarrow R \cdot c$ is the *selected clause*.

5 Model Construction, Redundancy and Static Completeness

In this section we show how to derive from a sequent $\Lambda \cdot \Gamma \ \vdash\ \Phi$ an intended interpretation $I[\Lambda, \pi, \alpha]$ as a canonical candidate model for Φ. Its components π and α will be determined first by Γ, and its congruence relation will be presented by a convergent ground rewrite system $\mathcal{R}_{\Lambda,\alpha}$ extracted from Λ and α. The general technique for defining $\mathcal{R}_{\Lambda,\alpha}$ is borrowed from the completeness proof of the Superposition calculus and the earlier $\mathcal{ME}_{\mathrm{E}}$ calculus.

A *rewrite rule* is an expression of the form $l \to r$ where l and r are F-sorted Herbrand terms. A *rewrite system* is a set of rewrite rules. The rewrite systems constructed below will be ordered, that is, consist of rules of the form $l \to r$ such that $l \succ r$. For a given Λ and suitable assignment α, we define by induction on the term ordering $\succ$ sets ϵ_K and $\mathcal{R}_K$ for every ground equation K between F-sorted Herbrand-terms. Assume that ϵ_L has already been defined for all such L with $K \succ L$. Let $\mathcal{R}_K = \bigcup_{K \succ L} \epsilon_L$, where

$$\epsilon_{l \approx r} = \begin{cases} \{l \to r\} & \text{if } \Lambda\alpha \text{ produces } l \approx r,\ l \succ r, \text{ and } l \text{ and } r \text{ are irreducible wrt } \mathcal{R}_{l \approx r} \\ \emptyset & \text{otherwise} \end{cases}$$

Finally define $\mathcal{R}_{\Lambda,\alpha} = \bigcup_K \epsilon_K$. If $\epsilon_{l \approx r} = l \to r$ we say that $l \approx r$ *generates* $l \to r$ *in* $\mathcal{R}_{\Lambda,\alpha}$.

For example, if $\Lambda = \{P(x), \neg P(v)\}$ and $\alpha = \{v \mapsto 1\}$ then $\mathcal{R}_{\Lambda,\alpha}$ contains $P(0) \to \mathsf{tt}$, $P(-1) \to \mathsf{tt}$, $P(-2) \to \mathsf{tt}$, $P(2) \to \mathsf{tt}$, $P(-3) \to \mathsf{tt}$, $P(3) \to \mathsf{tt}, \ldots$ but not $P(1) \to \mathsf{tt}$, which is irreducible, but $P(1) \to \mathsf{tt}$ is not produced by $\Lambda\alpha$.

Definition 5.1 (Induced interpretation). *Let Λ be a context, π a valuation, and α a suitable assignment for Λ. The* interpretation induced by Λ, π and α, *written as $I[\Lambda, \pi, \alpha]$, is the Herbrand interpretation $I[\pi, \alpha]$ that interprets foreground equality as $\mathcal{R}^\star_{\Lambda,\alpha}$, the congruence closure of $\mathcal{R}_{\Lambda,\alpha}$ (as a set of equations) over the Herbrand terms.*

The rewrite system $\mathcal{R}_{\Lambda,\alpha}$ is fully reduced by construction (no rule in $\mathcal{R}_{\Lambda,\alpha}$ rewrites any other rule in it). Since $\succ$ is well-founded on the Herbrand terms, $\mathcal{R}_{\Lambda,\alpha}$ is convergent. It

follows from well-known results that equality of Herbrand terms in $\mathcal{R}^{\star}_{\Lambda,\alpha}$ can be decided by reduction to normal form using the rules in $\mathcal{R}_{\Lambda,\alpha}$.

The rewrite system $\mathcal{R}_{\Lambda,\alpha}$ will also be used to evaluate evaluated context restrictions:

Definition 5.2 (Satisfaction of variable-free foreground literals). *Let R be a set of literals over Herbrand terms. We say that $\mathcal{R}_{\Lambda,\alpha}$* satisfies *$R$, and write $\mathcal{R}_{\Lambda,\alpha} \models R$, iff*

(i) for every $l \approx r \in R$, if $l \succ r$ then $l \to r \in \mathcal{R}_{\Lambda,\alpha}$, and
(ii) for every $\neg(l \approx r) \in R$, l and r are irreducible wrt. $\mathcal{R}_{\Lambda,\alpha}$.

For example, if $\Lambda = \{f(v) \approx e_2\}$, $\alpha = \{v \mapsto 1\}$, and $f(1) \succ e_1 \succ e_2 \succ 1$ then $\mathcal{R}_{\Lambda,\alpha} = \{f(1) \to e_2\}$ and $\mathcal{R}_{\Lambda,\alpha} \not\models \{\neg(f(1) \approx e_1), e_2 \approx e_1\}$ because the left-hand side of $\neg(f(1) \approx e_1)$ is reducible wrt. $\mathcal{R}_{\Lambda,\alpha}$, and because $e_1 \to e_2$ is not in $\mathcal{R}_{\Lambda,\alpha}$.

Our concepts of redundancy require comparing Herbrand closures. To this end, define $(C_1 \leftarrow R_1 \cdot c_1,\ \gamma_1) \succ (C_2 \leftarrow R_2 \cdot c_2,\ \gamma_2)$ iff $C_1\gamma_1 \succ C_2\gamma_2$, or else $C_1\gamma_1 = C_2\gamma_2$ and $R_1\gamma_1 \succ R_2\gamma_2$. Note that even if it ignores constraints, this ordering is not total, as constrained clauses may contain rigid variables.

Definition 5.3 (Redundant clause). *Let $\Lambda \cdot \Gamma \vdash \Phi$ be a sequent, and $\mathcal{D}$ and $(C \leftarrow R \cdot c,\ \gamma)$ Herbrand closures. We say that $(C \leftarrow R \cdot c,\ \gamma)$ is* redundant wrt $\mathcal{D}$ and $\Lambda \cdot \Gamma \vdash \Phi$ *iff (a) there is a $K \in R$ that is contradictory with Λ, (b) $\Gamma \cup \{c\gamma\}$ is not satisfiable, or (c) there exist Herbrand closures $(C_i \leftarrow R_i \cdot c_i,\ \gamma_i)$ of clauses in Φ, such that all of the following hold:*

(i) for every $L \in R_i$ there is a $K \in R$ such that $L \sim K$ and $L\gamma_i = K\gamma$,
(ii) $\Gamma \cup \{c\gamma\} \models c_i\gamma_i$,
(iii) $\mathcal{D} \succ (C_i \leftarrow R_i \cdot c_i,\ \gamma_i)$, and
(iv) $\{C_1\gamma_1, \ldots, C_n\gamma_n\} \models C\gamma$.

We say that a Herbrand closure $(C \leftarrow R \cdot c,\ \gamma)$ is *redundant wrt $\Lambda \cdot \Gamma \vdash \Phi$* iff it is redundant wrt $(C \leftarrow R \cdot c,\ \gamma)$ and $\Lambda \cdot \Gamma \vdash \Phi$, and that a clause $C \leftarrow R \cdot c$ is *redundant wrt $\Lambda \cdot \Gamma \vdash \Phi$* iff every Herbrand closure of $C \leftarrow R \cdot c$ is redundant wrt. $\Lambda \cdot \Gamma \vdash \Phi$.

If case (a) or (b) in the previous definition applies then $(C \leftarrow R \cdot c,\ \gamma)$ is trivially satisfied by $(\Lambda, \alpha, I[\alpha])$, for every suitable α that satisfies Γ and every $I[\alpha]$. Case (c) provides with (ii) and (iv) conditions under which $(C \leftarrow c)\gamma$ follows from the $(C_i \leftarrow c_i)\gamma_i$'s (in the sense of Definition 3.2). The context restrictions are taken into account by condition (i), which makes sure that evaluation of the pairs (R_i, γ_i) in terms of Definition 3.4 is the same as for (R, γ). In condition (iv), entailment $\models$ is meant as entailment in equational clause logic between sets of ordinary ground clauses and an ordinary ground clause.

Given a Pos-Res, Ref or Para inference with premise $\Lambda \cdot \Gamma \vdash \Phi$, selected clause $C \leftarrow R \cdot c$, selected context equation $l \approx r$ in case of Para, and a Herbrand substitution γ. If applying γ to $C \leftarrow R \cdot c$, the derived clause, and $l \approx r$ satisfies all applicability conditions of that inference rule, except $(C \leftarrow R \cdot c)\gamma \in \Phi$ and $(l \approx r)\gamma \in \Lambda$, we call the resulting ground inference a *ground instance via γ (of the given inference)*. This is not always the case, as, e.g., ordering constraints could be unsatisfied after application of γ.

Definition 5.4 (Redundant inference). *Let $\Lambda \cdot \Gamma \vdash \Phi$ and $\Lambda' \cdot \Gamma' \vdash \Phi'$ be sequents. An inference with premise $\Lambda \cdot \Gamma \vdash \Phi$ and selected clause $C \leftarrow R \cdot c$ is* redundant wrt $\Lambda' \cdot \Gamma' \vdash \Phi'$ *iff for every Herbrand substitution γ, $(C \leftarrow R \cdot c,\ \gamma)$ is redundant wrt. $\Lambda' \cdot \Gamma' \vdash \Phi'$ or the following holds, depending on the inference rule applied:*

Pos-Res, Ref, Para: *Applying γ to that inference does not result in a ground instance via γ, or $(C' \leftarrow R' \cdot c', \gamma)$ is redundant wrt. $(C \leftarrow R \cdot c, \gamma)$ and $\Lambda' \cdot \Gamma' \vdash \Phi'$, where $C' \leftarrow R' \cdot c'$ is the derived clause of that inference.*

Split ($C = \square$): *(a) there is a literal $K \in R$ such that Λ' does not produce K or (b) the split literal is contradictory with Λ'.*

Close ($C = \square$): $\square \leftarrow \emptyset \cdot \top \in \Phi'$.

Definition 5.5 (Saturated sequent). *A sequent $\Lambda \cdot \Gamma \vdash \Phi$ is* saturated *iff every inference with a core inference rule and premise $\Lambda \cdot \Gamma \vdash \Phi$ is redundant wrt. $\Lambda \cdot \Gamma \vdash \Phi$.*

We note that actually carrying out an inference makes it redundant wrt. the (all) conclusion(s), which already indicates that saturated sequents, although possibly infinite in each of its components, can be effectively computed.

Our first completeness result holds only for saturated sequents with respect to *relevant* closures. We say that a clause $(C \leftarrow R \cdot c, \gamma)$ is *relevant wrt. Λ and α* iff $\mathcal{R}_{\Lambda,\alpha} \models R\alpha\gamma$. All Herbrand closures of input clauses are always relevant.

Theorem 5.6 (Static completeness). *Let $\Lambda \cdot \Gamma \vdash \Phi$ be a saturated sequent, π a valuation and α a suitable assignment for $\Lambda \cdot \Gamma \vdash \Phi$. If $(\pi, \alpha) \models \Gamma$, $\mathrm{ran}(\alpha) = |\mathsf{B}|$ and $(\square \leftarrow \emptyset \cdot \top) \notin \Phi$ then the induced interpretation $I[\Lambda, \pi, \alpha]$ satisfies all Herbrand closures of all clauses in Φ that are relevant wrt. Λ and α. Moreover, $I[\Lambda, \pi, \alpha] \models C \leftarrow c$, for every $C \leftarrow c \in \Phi$.*

The stronger statement $I[\Lambda, \pi, \alpha] \models \Phi$ does in general not follow, as $I[\Lambda, \pi, \alpha]$ possibly does not satisfy a *non-relevant* closure of a clause in Φ. See [5] for a discussion why.

6 The $\mathcal{ME}_{\mathrm{E}}(\mathbf{T})$ Calculus

We now turn to the process of deriving saturated sequents. First, we introduce two more inference rules. The first one, Simp, is a generic simplification rule.

$$\mathsf{Simp}\ \frac{\Lambda \cdot \Gamma \vdash \Phi, C \leftarrow R \cdot c}{\Lambda \cdot \Gamma \vdash \Phi, C' \leftarrow R' \cdot c'}$$

if (1) $C \leftarrow R \cdot c$ is redundant wrt. $\Lambda \cdot \Gamma \vdash \Phi, C' \leftarrow R' \cdot c'$, and (2) $\Gamma \cup \Lambda^{(e,n)} \cup (\Phi \cup \{C \leftarrow R \cdot c\})^{\mathrm{c}} \models (C' \leftarrow R' \cdot c')^{\mathrm{c}}$. The first condition is needed for completeness, the second for soundness.

For example, if Λ contains a ground literal K, then every constrained clause of the form $C \leftarrow (\{\overline{K}\} \cup R) \cdot c$ can be deleted, and every constrained clause of the form $C \leftarrow (\{K\} \cup R) \cdot c$ can be replaced by $C \leftarrow R \cdot c$. The Simp rule encompasses various additional forms of simplification of the literals in C based on rewriting and subsumption, see [5].

$$\mathsf{Restrict}\ \frac{\Lambda \cdot \Gamma \vdash \Phi}{\Lambda \cdot \Gamma, c \vdash \Phi}$$

if c is a closed constraint such that $\Gamma \cup \{c\}$ is satisfiable.

For example, by 10-fold application of restrict one can construct a background context $\{1 \le v_1 \le 10, \ldots, 1 \le v_{10} \le 10\}$ that represents the numbers $1, \ldots, 10$ in a "nondeterministic" way. The purpose of Restrict is to construct finitely committed branches, as formally introduced below.

We are now ready to introduce derivation formally. In the following, we will use κ to denote an at most countably infinite ordinal. Let Ψ be a set of input clauses and Γ a satisfiable set of closed constraints, both rigid variable-free. A *derivation from Ψ and Γ* is a sequence $((N_i, E_i))_{0 \le i < \kappa}$ of trees of sequents (called derivation trees) with nodes N_i and edges E_i, such that $\mathbf{T}_0$ consists of the root-only tree whose sequent is $\neg x \cdot \Gamma \vdash \Psi$, and $\mathbf{T}_i$ is obtained by one single application of one of the core inference rules, Simp or Restrict to a leaf of $\mathbf{T}_{i-1}$, for all $1 \le i < \kappa$.

A *refutation* is a derivation that contains a *refutation tree*, that is, a derivation tree that contains in each leaf a sequent with $\square \leftarrow \emptyset \cdot \top$ in its clauses.

Every derivation determines a possibly infinite *limit tree* $\mathbf{T} = (\bigcup_{i<\kappa} N_i, \bigcup_{i<\kappa} E_i)$. In the following, let $\Lambda_i \cdot \Gamma_i \vdash \Phi_i$ be the sequent labeling the node i in some branch $\mathbf{B}$ with κ nodes of a limit tree $\mathbf{T}$, for all $i < \kappa$. Let

- $\Gamma_\mathbf{B} = \bigcup_{i<\kappa} \Gamma_i$ the *limit background context*,
- $\Lambda_\mathbf{B} = \bigcup_{i<\kappa} \Lambda_i$ be the *limit foreground context*, and
- $\Phi_\mathbf{B} = \bigcup_{i<\kappa} \bigcap_{i \le j<\kappa} \Phi_i$ be the *persistent clauses*.

The tuple $\Lambda_\mathbf{B} \cdot \Gamma_\mathbf{B} \vdash \Phi_\mathbf{B}$ is the *limit sequent (of* $\mathbf{B}$*)*. To prove a completeness result, derivations in $\mathcal{ME}_\mathrm{E}(\mathrm{T})$ need to construct limit sequents with certain properties:

Definition 6.1 (Exhausted branch). *We say that* $\mathbf{B}$ *is* exhausted *iff for all $i < \kappa$:*

(i) every Pos-Res, Ref, Para, Split *and* Close *inference with premise $\Lambda_i \cdot \Gamma_i \vdash \Phi_i$ and a persistent selected clause is redundant wrt. $\Lambda_j \cdot \Gamma_j \vdash \Phi_j$ for some j with $i \le j < \kappa$.*
(ii) $(\square \leftarrow \emptyset \cdot \top) \notin \Phi_i$.

While the above notion is similar to the one already used in $\mathcal{ME}_\mathrm{E}$, $\mathcal{ME}_\mathrm{E}(\mathrm{T})$ has additional requirements on the limit background context $\Gamma_\mathbf{B}$, introduced next.

Definition 6.2 (Finitely committed branch). *We say that* $\mathbf{B}$ *is* finitely committed *iff (a) $\Gamma_\mathbf{B}$ is finite or (b) for all $i < \kappa$, there are π_i and α_i such that $(\pi_i, \alpha_i) \models \Gamma_i$, and*

(i) $\bigcup_{i<\kappa} \bigcap_{i \le j<\kappa} \mathrm{ran}(\alpha_j) = |\mathsf{B}|$,
(ii) for every $n \in |\mathsf{B}|$, the set $\{v \mid \alpha_i(v) = n$, for some $i < \kappa\}$ is finite,
(iii) for every rigid variable v occuring in $\Gamma_\mathbf{B}$, the set $\{\alpha_i(v) \mid v \in \mathrm{dom}(\alpha_i)$, for some $i < \kappa\}$ is finite, and
(iv) for every parameter a occuring in $\Gamma_\mathbf{B}$, the set $\{\pi_i(a) \mid i < \kappa\}$ is finite.

The set in condition (i) consists of those background domain elements that are represented by some (not necessarily the same) rigid variable from some point on forever. The condition requires that this must be the case for all background domain elements. Condition (ii) says that only finitely many rigid variables can be used for that. Condition (iii) says that no rigid variable occuring in $\Gamma_\mathbf{B}$ can be assigned infinitely many values as the context evolves. Condition (iv) is similar, but for parameters. (Recall that parameter valuations are total, hence $\pi_i(a)$ is defined for every parameter a.)

The purpose of Definition 6.2 is to make sure that a valuation π and a suitable assignment α for Γ always exists, and moreover, that (π, α) satisfies $\Gamma_\mathbf{B}$:

Proposition 6.3 (Compactness of finitely committed branches). *If* $\mathbf{B}$ *is finitely committed then there is a* π *and an* α *such that* $\mathrm{ran}(\alpha) = |\mathsf{B}|$ *and* $(\pi, \alpha) \models \Gamma_{\mathbf{B}}$.

To see one of the issues that Proposition 6.3 addresses consider $\Gamma_i = \bigcup_{n \leq i}\{v_1 > n\}$, then $\Gamma_{\mathbf{B}}$ is not satisfiable, although every finite subset is satisfiable. On the other hand, condition (iii) in Definition 6.2 is not satisfied.

With enough Restrict applications finitely committed limit branches can be constructed in a straightforward way if the input background constraints confine every parameter to a finite domain. In the LIA case, e.g., one could "slice" the integers in intervals of, say, 100 elements and enumerate, with Restrict, declarations like $1 \leq v_1 \leq 100, \ldots, 1 \leq v_{100} \leq 100$ before any rigid variable v_i is used for the first time (in Split), and do that for all intervals. In certain cases it is possible to determine *a priori* that limit background contexts will be finite, and then Restrict is not required at all, see Section 7.

Definition 6.4 (Fairness). *A derivation is fair iff it is a refutation or its limit tree has an exhausted and finitely committed branch.*

Theorem 6.5 (Completeness). *Let* Ψ *be a set of input clauses and* Γ *a satisfiable set of closed constraints, both rigid variable-free. Suppose a fair derivation from* Ψ *and* Γ *that is not a refutation. Let* $\mathbf{B}$ *be any exhausted and finitely committed branch of its limit tree, and let* $\Lambda_{\mathbf{B}} \cdot \Gamma_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$ *be the limit sequent of* $\mathbf{B}$.

Then there is a valuation π *and a suitable assignment* α *for* $\Gamma_{\mathbf{B}}$ *such that* $\mathrm{ran}(\alpha) = |\mathsf{B}|$ *and* $(\pi, \alpha) \models \Gamma_{\mathbf{B}}$*, and it holds* $I[\Lambda_{\mathbf{B}}, \pi] \models \Gamma \cup \Psi$*, where* $I[\Lambda_{\mathbf{B}}, \pi, \alpha]$ *is the interpretation induced by* $\Lambda_{\mathbf{B}}$*,* π*, and* α.

The proof exploits Proposition 6.3 to show that π and α exist as claimed. It then proceeds by showing that $\Lambda_{\mathbf{B}} \cdot \Gamma_{\mathbf{B}} \vdash \Phi_{\mathbf{B}}$ is saturated, as a link with Theorem 6.5.

7 Soundness and Special Cases

Theorem 7.1 (Relative refutational soundness). *Let* Ψ *be a set of input clauses and* Γ *a satisfiable set of closed constraints, both rigid variable-free. Suppose a refutation from* Ψ *and* Γ *and let* $\Gamma_{\mathbf{B}}$ *be its limit background context. Then,* $\Gamma_{\mathbf{B}} \supseteq \Gamma$*,* $\Gamma_{\mathbf{B}}$ *is satisfiable, and* $\Gamma_{\mathbf{B}} \cup \Psi$ *is not satisfiable.*

Here, by the limit background context $\Gamma_{\mathbf{B}}$ of a refutation we mean the background context of the sequent in the leaf of the rightmost branch in its refutation tree.

Suppose the conditions of Theorem 7.1 hold, and let $I[\pi, \alpha]$ be such that $(\pi, \alpha) \models \Gamma_{\mathbf{B}}$, as claimed. It follows $I[\pi, \alpha] \not\models \Psi$ and, as Ψ is rigid variable-free, $I[\pi] \not\models \Psi$. (If additionally Ψ is parameter-free then $\Gamma_{\mathbf{B}}$ and Ψ are independent, and so Ψ alone is not satisfiable.) For example, if $\Psi = \{P(x) \leftarrow x = a,\ \neg P(x) \leftarrow x = 5\}$ and $\Gamma = \{a > 2\}$ then there is a refutation with, say, $\Gamma_{\mathbf{B}} = \{a > 2,\ v_1 = a,\ v_1 = 5\}$. Notice that $(\pi, \alpha) \models \Gamma_{\mathbf{B}}$ entails $\pi = \{a \mapsto 5\}$, and, obviously, $I[\pi] \not\models \Psi$. But of course $\Psi \cup \Gamma$ is satisfiable, take, e.g., $\pi = \{a \mapsto 3\}$. A usual soundness result can thus be not based on *single* refutations, and this is why we call the soundness result above "relative". To fix that, we work with *sequences* of refutations whose limit background contexts collectively

cover the initially given Γ. In the example, the next derivation starts with (essentially) $\Gamma = \{a > 2, \neg(a = 5)\}$, which leads to a derivation that provides the expected model.

Define $\mathrm{mods}(\Gamma) = \{\pi \mid (\pi, \alpha) \models \Gamma$, for some suitable $\alpha\}$. Then, the intuition above leads to the following general procedure:

1: **D** $\leftarrow$ a fair derivation from Ψ and Γ {both Ψ and Γ assumed rigid-variable free}
2: **while D** is a refutation **do**
3: $\quad\Gamma_{\mathbf{B}} \leftarrow$ the limit background context of **D**
4: $\quad$**if** $\mathrm{mods}(\Gamma_{\mathbf{B}}) = \mathrm{mods}(\Gamma)$ **then**
5: $\quad\quad$**return** unsatisfiable
6: $\quad$**else**
7: $\quad\quad\Gamma' \leftarrow$ any background context s.t. $\mathrm{mods}(\Gamma') = \mathrm{mods}(\Gamma) \setminus \mathrm{mods}(\Gamma_{\mathbf{B}})$
8: $\quad\quad\Gamma \leftarrow \Gamma'$; **D** $\leftarrow$ a fair derivation from Ψ and Γ
9: **return** satisfiable

At line 4, $\mathrm{mods}(\Gamma)$ is the set of parameter valuations under which the unsatisfiability of Ψ is yet to be established. If the current refutation **D** from Ψ and Γ does not further constrain Γ, i.e., if $\mathrm{mods}(\Gamma_{\mathbf{B}}) = \mathrm{mods}(\Gamma)$, then nothing remains to be done. Otherwise $\mathrm{mods}(\Gamma) \setminus \mathrm{mods}(\Gamma_{\mathbf{B}})$ is non-empty, and in the next iteration Γ is taken to stand for exactly those parameter valuations that are not sanctioned by the current refutation, i.e., those that satisfy the current Γ but not $\Gamma_{\mathbf{B}}$. It follows easily with Theorem 7.1 that if the procedure terminates with "unsatisfiable" on line 5 then $\Psi \cup \Gamma$ is indeed unsatisfiable, the desired standard soundness result. If on the other hand **D** is not a refutation then the procedure returns "satisfiable", which is sanctioned by Theorem 6.5.

Notice that the test in line 4 can be made operational by checking the validity of the formula $(\exists \wedge_{c\in\Gamma} c) \equiv (\exists \wedge_{c\in\Gamma_{\mathbf{B}}} c)$, where $\exists F$ denotes the existential quantification over all rigid variables occurring in F. Similarly, Γ' on line 7 can be taken as $\Gamma \cup \{\neg\exists \wedge_{c\in\Gamma_{\mathbf{B}}} c\}$. If the background theory admits quantifier elimination (e.g., LIA extended with divisibility predicates) the existential quantifiers can be removed and further simplifications may take place. In the example, the background context Γ' computed in the first iteration is $\Gamma' = \{a > 2, \neg\exists v_1\ (a > 2 \wedge v_1 = a \wedge v_1 = 5)\} \equiv \{a > 2, \neg(a = 5)\}$.

The derivation **D** might not be finite. In this case the procedure does not terminate, but this is acceptable as by $\Psi \cup \Gamma$ is satisfiable then. Another source of non-termination comes from growing the sets Γ' without bound. This is theoretically acceptable, as our logic is not even semi-decidable. In practice, one could add to Γ finite domain declarations for all parameters involved, such as $1 \leq a \leq 100$. This leads to finitely many Γ' only. Moreover, the sets Γ' can then be computed in a conflict-driven way. For example, if $\Psi = \{P(x) \leftarrow x = a,\ \neg P(x) \leftarrow 1 \leq x \leq 50\}$ and $\Gamma = \{1 \leq a \leq 100\}$, the procedure will derive in the first iteration a refutation (in $O(1)$ time). The second iteration will then result in a derivation (a non-refutation) that restricts a to the range $[51, 100]$ and the procedure will stop with "satisfiable".

Another special case is when all clauses are of the form $C \leftarrow R \cdot (c \wedge x_1 = t_1 \wedge \cdots \wedge x_n = t_n)$, where $\{x_1, \ldots, x_n\} = \mathit{fvar}(C) \cup \mathit{fvar}(R)$, and c and the t_i's are ground. Such clauses, where initially $R = \emptyset$, are obtained from abstraction of formulas of the form $C \leftarrow c$, where C is an ordinary ground Σ-clause and c is a ground constraint. (This is the fragment over which $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$ overlaps with typical SMT methods.) It is not too difficult to argue that all derivable clauses then have that form as well. As a conse-

quence, (*i*) all split literals are variable-free, and hence so are all derivable foreground contexts, and (*ii*) there is only one instantiation of the x_i's in Split, since no (satisfiable) background context can contain $v_1 = t$ and $v_2 = t$ for different rigid variables v_1 and v_2. It follows that the limit background contexts are finite *for any input background context*, hence no finite domain declarations for parameters are needed. Moreover, as the set of (non-rigid variable) background terms t_i is fixed a priori, there are only finitely many non-equivalent background contexts. Therefore, the procedure above cannot grow Γ indefinitely. Furthermore, all derivations are guaranteed to be final because context literals are variable-free and can use only finitely-many rigid variables. As a consequence, $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$ provides a decision procedure for ground problems in the combination of the background theory and uninterpreted (F-sorted) function symbols with equality.

8 Conclusions

We presented the new $\mathcal{ME}_{\mathrm{E}}(\mathrm{T})$ calculus, which properly generalizes the essentials of two earlier Model Evolution calculi, $\mathcal{ME}_{\mathrm{E}}$ [4], and $\mathcal{ME}(\mathrm{LIA})$ [3], one with equational inference rules but without theory reasoning, and the other with theory reasoning by without equality over non-theory symbols.

Much remains to be done. Further work includes extending the calculus with "universal variables" and additional simplification rules. A further extension, which could be done along the lines of [2], would allow also B-sorted (non-constant) function symbols. Another important question is how to strengthen the model-building capabilities of the calculus, to guarantee termination in more cases of practical relevance.

References

1. E. Althaus, E. Kruglov, and C. Weidenbach. Superposition modulo linear arithmetic SUP(LA). In S. Ghilardi and R. Sebastiani, eds., *Proc. FroCos*, LNCS 5749, Springer, 2009.
2. L. Bachmair, H. Ganzinger, and U. Waldmann. Refutational theorem proving for hierachic first-order theories. *Appl. Algebra Eng. Commun. Comput*, 5:193–212, 1994.
3. P. Baumgartner, A. Fuchs, and C. Tinelli. ME(LIA) – Model Evolution With Linear Integer Arithmetic Constraints. In I. Cervesato, H. Veith, and A. Voronkov, eds., *Proc. LPAR'08*, LNAI 5330, Springer, 2008.
4. P. Baumgartner and C. Tinelli. The Model Evolution Calculus With Equality. In R. Nieuwenhuis, ed., *Proc. CADE-20*, LNAI 3632, Springer, 2005.
5. P. Baumgartner and U. Waldmann. Superposition and Model Evolution Combined. In R. Schmidt, ed., *Proc. CADE-22*, LNAI 5663, Springer, 2009.
6. H. Ganzinger and K. Korovin. Theory Instantiation. In *Proc. LPAR'06*, LNCS 4246, Springer, 2006.
7. Y. Ge, C. Barrett, and C. Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In F. Pfenning, eds., *Proc. CADE-21*, LNCS, Springer, 2007.
8. K. Korovin and A. Voronkov. Integrating linear arithmetic into superposition calculus. In *Proc. CSL'07*, LNCS 4646, Springer, 2007.
9. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, Nov. 2006.
10. P. Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In I. Cervesato, H. Veith, and A. Voronkov, eds., *Proc. LPAR'08*, LNAI 5330, Springer, 2008.