$\mathcal{ME}$(LIA) - Model Evolution With Linear Integer Arithmetic Constraints

Peter Baumgartner
NICTA, Canberra, Australia
Peter.Baumgartner@nicta.com.au

Cesare Tinelli
Department of Computer Science
The University of Iowa, USA
tinelli@cs.uiowa.edu

Alexander Fuchs
Department of Computer Science
The University of Iowa
fuchs@cs.uiowa.edu

September 27, 2008

Abstract

Many applications of automated deduction require reasoning modulo some form of integer arithmetic. Unfortunately, theory reasoning support for the integers in current theorem provers is sometimes too weak for practical purposes. In this paper we propose a novel calculus for a large fragment of first-order logic modulo Linear Integer Arithmetic (LIA) that overcomes several limitations of existing theory reasoning approaches. The new calculus — based on the *Model Evolution* calculus, a first-order logic version of the propositional DPLL procedure — supports restricted quantifiers, requires only a decision procedure for LIA-validity instead of a complete LIA-unification procedure, and is amenable to strong redundancy criteria. We present a basic version of the calculus and prove it sound and (refutationally) complete.

1 Introduction

Many applications of automated deduction require reasoning modulo some form of integer arithmetic. Unfortunately, theory reasoning support for the integers in current theorem provers is sometimes too weak for practical purposes. In particular, the family of *Satisfiability Modulo Theories* solvers lack support for quantifiers and resort to incomplete or inefficient heuristics to deal with quantified formulas [GBT07, e.g.]. Also, theory reasoning techniques developed within first-order theorem proving are often impractical as they require the enumeration of complete sets of theory unifiers (in particular those in the tradition of Stickel's Theory Resolution [Sti85]) or feature only weak or no redundancy criteria (e.g., Bürckert's Constraint Resolution [Bür90]).

In this paper we propose a novel refutation calculus for a fragment of first-order logic modulo Linear Integer Arithmetic (LIA) that overcomes these problems. The fragment is a clause logic with restricted quantifiers whose language extends that of LIA with free constant and predicate symbols, and whose models are arbitrary expansions of the integers structure to those free symbols.

The quantifier restrictions are as follows: every (universal) variable is restricted to range over a bounded below interval of $\mathbb{Z}$ (such as, for instance, $\mathbb{N}$), while every free constant is restricted to range over a finite interval of $\mathbb{Z}$.

The exclusion of free function symbols of non-zero arity and the restriction of free constant symbols to a finite range make (entailment in) the logic recursively enumerable, and semi-decided by the calculus we present here. The restriction of the variables on the other hand is not essential. It is used here only because it simplifies the treatment of the calculus. Further restricting the variables to finite intervals makes however the logic decidable, and our calculus terminating.

In spite of the restrictions, the logic is quite powerful. For instance, functions with a finite range can be easily encoded into it. This makes the logic particularly well-suited for applications that deal with bounded domains, such as, for instance, bounded model checking and planning. SAT-based techniques, based on clever reductions of BMC and planning to SAT, have achieved considerable success in the past, but they do not scale very well due to the size of the propositional formulas produced. It has been argued and shown by us and others [BFdNT07, NP07] that this sort of applications could benefit from a reduction to a more powerful logic for which efficient decision procedures are available. That work had proposed the function-free fragment of clause logic as a candidate. This paper takes that proposal a step further by adding integer constraints to the picture. The ability to reason natively about the integers can provide a reduction in search space even for problems that do not originally contain integer constraints. The following simple example from finite model reasoning demonstrates this:[1]

$$a : [1..100] \qquad P(a) \qquad \neg P(x) \leftarrow 1 \dot{\leq} x \wedge x \dot{\leq} 100 \ .$$

The clause set above is unsatisfiable because the interval declaration $a : [1..100]$ for the constant a together with the unit clause $P(a)$ permit only models that satisfy one of $P(1), \ldots, P(100)$. Such models however falsify the third clause. Finite model finders, e.g., need about 100 steps to refute the clause set, one for each possible value of a. Our $\mathcal{ME}$(LIA) calculus, on the other hand, reasons directly with integer intervals and allows a refutation in $O(1)$ steps. See Section 2 for an in-depth discussion of another example.

The calculus we propose relies on a decision procedure for the full fragment of LIA instead of a complete enumerator of LIA-unifiers. It is derived from the *Model Evolution* calculus ($\mathcal{ME}$) [BT03], a first-order logic version of the propositional DPLL procedure. In $\mathcal{ME}$, the main data structure is the *context*, a finite set of literals providing a compact representation of certain Herbrand interpretations serving as candidate models for the input clause set. The new calculus, $\mathcal{ME}$(LIA), extends $\mathcal{ME}$'s contexts to sets of literals

[1] The predicate symbol $\dot{\leq}$ denotes less than or equal on integers.

with LIA constraints, representing expansions of the integers structure by free predicate and constant symbols. The crucial insight that leads from $\mathcal{ME}$ to $\mathcal{ME}$(LIA) lies in the use of the ordering $<$ on integers in $\mathcal{ME}$(LIA) instead of the instantiation ordering on terms in $\mathcal{ME}$. This then allows $\mathcal{ME}$(LIA) to work with concepts over integers that are similar to concepts used in $\mathcal{ME}$ over free terms. For instance, it enables a strong redundancy criteria that is formulated, ultimately, as certain constraints over LIA expressions.

In this paper we introduce the main ideas of the new calculus and discuss its theoretical properties. For simplicity and space constraints, we present only a basic version of the calculus, which is designed to use a minimal number of inference rules and treat the LIA decision procedure as an oracle, disregarding its (high) computational complexity. We are currently working on an enhanced version with additional rules that relies on a customized quantifier elimination procedure for LIA and efficient solvers for its universal fragment to reduce the cost of solving LIA constraints. The enhanced calculus will be described in a follow-up paper.

Related work. Most of the related work has been carried out in the framework of the resolution calculus. One of the earliest related calculi is theory resolution [Sti85]. In our terminology, theory resolution requires the enumeration of a complete set of solutions of constraints. The same applies to various "theory reasoning" calculi introduced later [Bau98, GK06]. In contrast, in $\mathcal{ME}$(LIA) all background reasoning tasks can be reduced to *satisfiability checks* of (quantified) constraint formulas. This weaker requirement facilitates the integration of a larger class of solvers (such as quantifier elimination procedures) and leads to potentially far less calls to the background reasoner. For an extreme example, the clause $\neg(0 < x) \vee P(x)$ has, by itself, infinitely many most general LIA-unifiers (the theory reasoning analogous of most general unifiers), namely $\{x \mapsto 1\}, \{x \mapsto 2\}, \ldots$, the most general solutions of the constraint $(0 < x)$ with respect to the term instantiation ordering. Thus, any calculus based on the computation of complete sets of (most general) solutions of LIA-constraints may need to consider all of them. In contrast, in $\mathcal{ME}$(LIA), or in other calculi based on *satisfiability* alone, notably Bürckert's *constrained resolution* [Bür90], it is enough just to check that a constraint like $(0 < x)$ is LIA-satisfiable.

Constrained resolution is actually more general than $\mathcal{ME}$(LIA), as it admits background theories with (infinitely, essentially enumerable) many models, as opposed to the single fixed model that $\mathcal{ME}$(LIA) works with.[2] On the other hand, constraint resolution does not admit free constant or function symbols—unless they are considered as part of the background theory, which is pointless since specialized background theory reasoners do not accept free symbols. The most severe drawback of constraint resolution, however, is the lack of redundancy criteria.

The importance of powerful redundancy criteria has been emphasized in the development of the modern theory of resolution in the 1990s [NR01]. With slight variations they carry over to *hierarchical superposition* [BGW94], a calculus that is related to constraint resolution. The recent calculus in [KV07] integrates dedicated inference rules for Linear

[2] Extending $\mathcal{ME}$(LIA) correspondingly is future work.

Rational Arithmetic into superposition. In [BT03, e.g.] we have described conceptual differences between $\mathcal{ME}$, further *instance based methods* [Bau07] and other (resolution) calculi. Many of the differences carry over to the constraint-case, possibly after some modifications. For instance, $\mathcal{ME}$(LIA) *explicitly*, like $\mathcal{ME}$, maintains a candidate model, which gives rise to a redundancy criterion different to the ones in superposition calculi. Also it is known that instance-based methods decide different fragments of first-order logic, and the same holds true for the constraint-case.

Over the last years, *Satisfiability Modulo Theories* has become a major paradigm for theorem proving modulo background theories. In one of its main approaches, DPLL(T), a DPLL-style SAT-solver is combined with a decision procedure for the quantifier-free fragment of the background theory T [NOT06]. DPLL(T) is essentially limited to the ground case. In fact, addressing this intrinsic limitation by lifting DPLL(T) to the first-order level is one of the main motivations for the $\mathcal{ME}$(LIA) calculus (much like $\mathcal{ME}$ was motivated by the goal of lifting the propositional DPLL procedure to the first-order level while preserving its good properties). At the current stage of development the core of the procedure—the Split rule—and the data structures are already lifted to the first-order level. We are working on an enhanced version with additional rules, targeting efficiency improvements. With these rules then $\mathcal{ME}$(LIA) can indeed be seen as a proper lifting of DPLL(T) to the first-order level (within recursion-theoretic limitations).

2 Calculus Preview

It is instructive to discuss the main ideas of the $\mathcal{ME}$(LIA) calculus with a simple example before defining the calculus formally. Consider the following two unit *constrained clauses* (formally defined in Section 3):[3]

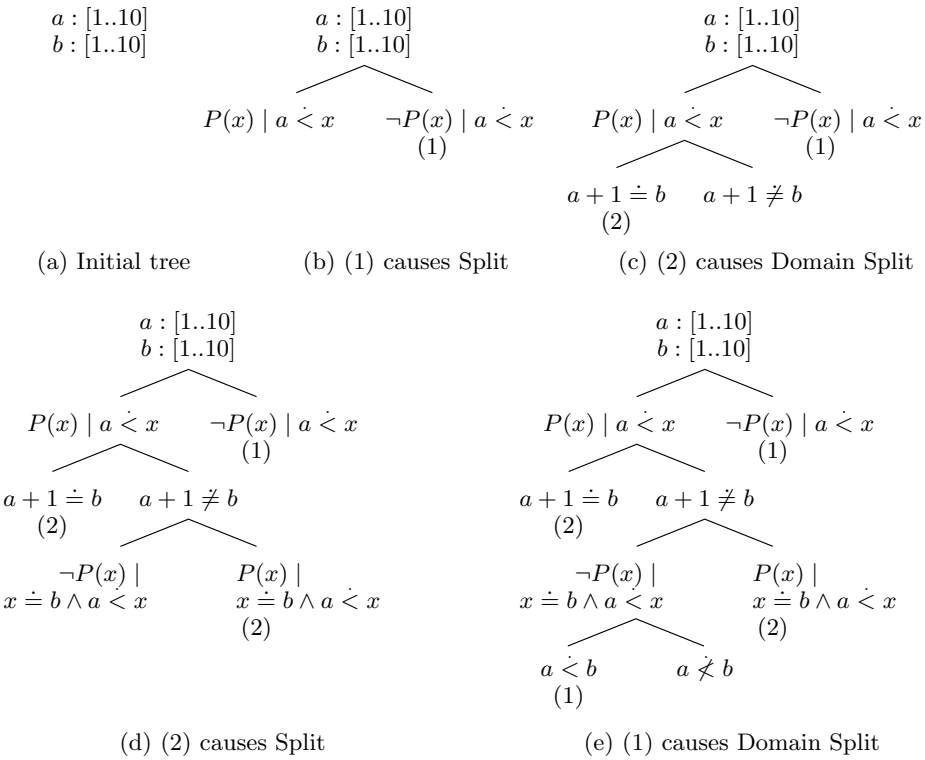
$$P(x) \leftarrow a \dot{<} x \tag{1}$$

$$\neg P(x) \leftarrow x \doteq b \tag{2}$$

where a, b are free constants, which we call *parameters*, x, y are (implicitly universally quantified) variables, and $a \dot{<} x$ and $x \doteq b$ are the respective constraints of clause (1) and (2). The restriction that all parameters range over some finite integer domain is achieved with the global constraints $a : [1..10]$, $b : [1..10]$. Informally, clause (1) states that there is a value of a in $\{1, \ldots, 10\}$ such that $P(x)$ holds for all integers x greater than a. Similarly for clause (2).

The clause set above is satisfiable in any expansion of the integers structure $\mathcal{Z}$ to $\{a, b, P\}$ that maps a, b into $\{1, \ldots, 10\}$ with $a \geq b$. The calculus will discover that and compute a data structure that denotes exactly all these expansions. To see how this works, it is best to describe the calculus' main operations using a semantic tree construction, illustrated in Figure 1. Each branch in the semantic tree denotes a finite set of first-order interpretations that are expansions of $\mathcal{Z}$. These interpretations are the

[3]The predicate symbol $\doteq$ denotes integer equality and $\not\doteq$ stands for $\neg(\cdot \doteq \cdot)$; similarly for $\dot{<}$.



(a) Initial tree (b) (1) causes Split (c) (2) causes Domain Split

(d) (2) causes Split (e) (1) causes Domain Split

Figure 1: Derivation example. Closed branches are marked with the number of the clause used to close them.

key to understanding the working of the calculus. The calculus' goal is to construct a branch denoting a set of interpretations that are each a model of the given clause set and the global parameter constraints, or to show that there is no such model.

In the example in Figure 1a, the initial single-node tree denotes all interpretations that interpret a and b over $\{1, \ldots, 10\}$ and falsify *by default* all ground atoms of the form $P(n)$ where n is an integer constants (e.g., $P(-1), P(4), \ldots$). Each of these (100) interpretations falsifies clause (1). The calculus detects that and tries to fix the problem by changing the set of interpretations in two essentially complementary ways. It does that by computing a *context unifier* and applying the *Split* inference rule (both defined later) which extends the tree as in Figure 1b. With the addition of the *constrained literal* $P(x) \mid a \dot{<} x$, the left branch of the new tree now denotes all interpretations that interpret a and b as before but satisfy $P(n)$ only for values of n greater than a.

The right branch in Figure 1b still denotes the same set of interpretations as in the original branch. However, the presence of $\neg P(x) \mid a \dot{<} x$ now imposes a restriction on later extensions of the branch. To explain how, we must observe first that in the calculus the set of solutions of any constraint (which are integer tuples) is a well-founded poset. Hence, each satisfiable constraint has *minimal solutions*. Now, if a branch in the

semantic tree contains a literal $L(x_1, \ldots, x_k) \mid c$ where c is a satisfiable constraint over the variables $x_1, \ldots, x_n$, each associated interpretation I satisfies $L(n_1, \ldots, n_k)$ where $(n_1, \ldots, n_k)$ is one of the minimal solutions of c in I. Further extensions of the branch must maintain $L(n_1, \ldots, n_k)$ satisfied. This minimal solution is commited to at the time the literal is added to the semantic tree. In the right branch of Figure 1b a (unique) minimal solution of $a \dot{<} x$ is $a + 1$ for all interpretations. This entails that $\neg P(a + 1)$ is *permanently valid* in the branch in the sense that (i) $\neg P(a + 1)$ holds in every interpretation of the branch *and* (ii) no extensions of the branch are allowed to change that. As a consequence, the right branch permanently falsifies clause (1), and so it can be closed.

Similarly, $P(a + 1)$ is permanently valid in the left branch of Figure 1b.[4] In interpretations of the branch where $a + 1 = b$ this is a problem because there clause (2) is falsified. Since the branch also has interpretations where $a + 1 \neq b$, the calculus makes progress by splitting on $a + 1 \doteq b$. This is done with the *Domain Split* rule, leading to the tree in Figure 1c. The leftmost branch there denotes only interpretations where $a + 1 = b$. That branch can be closed because it permanently falsifies clause (2). It is worth pointing out that domain splits like the above, identifying "critical" cases of parameter assignments, can be computed deterministically. They do not need not be guessed.

The branch ending in $a+1 \not\doteq b$ still contains interpretations that falsify the clause set. For instance, those that map a to 2 and b to 4, say, will satisfy $P(4)$, and so falsify clause (2). This situation is identified by conjoining the constraint $a \dot{<} x$ in $P(x) \mid a \dot{<} x$ and the constraint $x \doteq b$ in clause (2). The obtained constraint $x \doteq b \wedge a \dot{<} x$ is satisfiable under the parameter valuation $\{a \mapsto 2, b \mapsto 4\}$. Moreover, its minimal solution differs from the minimal solution of the constraint in $P(x) \mid a \dot{<} x$. This makes the Split rule applicable with the literal $\neg P(x) \mid x \doteq b \wedge a \dot{<} x$, which yields the tree in Figure 1d. The branch ending in $P(x) \mid x \doteq b \wedge a \dot{<} x$ can be closed, with clause (2), for permanently satisfying $P(b)$.[5]

Moving to the branch ending in $\neg P(x) \mid x \doteq b \wedge a \dot{<} x$, let us consider its interpretations where $a < b$. As defined later, those interpretations satisfy $P(a + 1), \ldots, P(b - 1)$ and falsify, among others, $P(b), P(b + 1)$ and so on. This is a consequence of the fact that when $a < b$ the minimal solution of the constraint in $P(x) \mid a \dot{<} x$, namely $a + 1$, is smaller than the minimal solution of the constraint in $\neg P(x) \mid x \doteq b \wedge a \dot{<} x$, namely b. If the branch had only such interpretations, it would permanently falsify clause (1) and could then be closed. This situation is achieved by applying Domain Split with the literal $a \dot{<} b$, resulting in the tree of Figure 1e. As for the branch ending in $a \not\dot{<} b$, all its interpretations satisfy $P(n)$ for all $n > a$ (because the constraint in $\neg P(x) \mid x \doteq b \wedge a \dot{<} x$ is now unsatisfiable) and falsify $P(b)$ (by default, because $a \not< b$). It follows that they

[4] In DPLL terms, the split with $P(x) \mid a \dot{<} x$ and $\neg P(x) \mid a \dot{<} x$ is akin to a split on the complementary literals $P(a + 1)$ and $\neg P(a + 1)$. The calculus soundness proof relies in essence on this observation.

[5] Because b is a minimal solution of $x \doteq b \wedge a \dot{<} x$ in all interpretations of the branch where $x \doteq b \wedge a \dot{<} x$ has a solution.

all satisfy the clause set. The calculus recognizes that and stops. Had the clause set been unsatisfiable, the calculus would have generated a tree with closed branches only.

Note how the calculus found a model, in fact a set of models, for the input clause set without having to enumerate all possible values for the parameters a and b, resorting instead to much more course-grained domain splits. In its full generality, the calculus still works as sketched above. Its formal description is, however, more complex because the calculus handles constraints with more than one (free) variable, and does not require the computation of explicit, symbolic representations of minimal solutions.

3 Constraints and Constrained Clauses

The new calculus works with clauses containing *parametric linear integer constraints*, which we call here simply *constraints*. These are *any first-order formulas* over the signature $\Sigma_{\mathcal{Z}}^{\Pi} = \{\doteq, \dot{<}, +, -, 0, \pm 1, \pm 2, \ldots\} \cup \Pi$, where Π is a finite set of constant symbols not in $\Sigma_{\mathcal{Z}} = \Sigma_{\mathcal{Z}}^{\Pi} \setminus \Pi$. The symbols of $\Sigma_{\mathcal{Z}}$ have the expected arity and usage. Following a common math terminology, we will call the elements of Π *parameters*. We will use, possibly with subscripts, the letters m, n to denote the *integer constants* (the constants in $\Sigma_{\mathcal{Z}}$); a, b to denote parameters; x, y to denote variables (chosen from an infinite set X); s, t to denote terms over $\Sigma_{\mathcal{Z}}^{\Pi}$, and l to denote literals.

We write $t : [m \,..\, n]$ as an abbreviation of $m \dot{\leq} t \wedge t \dot{\leq} n$. We denote by $\bar{\exists}\, c$ (resp. $\bar{\forall}\, c$) the existential (resp. universal) closure of the constraint c, and by $\pi\, \mathbf{x}\, c$ the *projection of c on* $\mathbf{x}$, i.e., $\exists \mathbf{y}\, c$ where $\mathbf{y}$ is a tuple of all the free variables of c that are not in the variable tuple $\mathbf{x}$.

A constraint is *ground* if it contains no variables, *closed* if it contains no free variables.[6] We define a satisfaction relation $\models_{\mathcal{Z}}$ for closed parameter-free constraints as follows: $\models_{\mathcal{Z}} c$ if c is satisfied in the standard sense in the structure $\mathcal{Z}$ of the integers—the one interpreting the symbols of $\Sigma_{\mathcal{Z}}$ in the usual way over the universe $\mathbb{Z}$. A *parameter valuation* α, a mapping from Π to $\mathbb{Z}$, determines an expansion $\mathcal{Z}_\alpha$ of $\mathcal{Z}$ to the signature $\Sigma_{\mathcal{Z}}^{\Pi}$ that interprets each $a \in \Pi$ as $\alpha(a)$. For each parameter valuation α and closed constraint c we write $\alpha \models_{\mathcal{Z}} c$ to denote that c is satisfied in $\mathcal{Z}_\alpha$. A (possibly non-closed) constraint c is *α-satisfiable* if $\alpha \models_{\mathcal{Z}} \bar{\exists}\, c$.

For finite sets Γ of closed constraints we denote by $\mathit{Mods}(\Gamma)$ the set of all valuations α such that $\alpha \models_{\mathcal{Z}} \Gamma$. We write $\Gamma \models_{\mathcal{Z}} c$ to denote that $\alpha \models_{\mathcal{Z}} c$ for all $\alpha \in \mathit{Mods}(\Gamma)$. For instance, $a : [1 \,..\, 10] \models_{\mathcal{Z}} \exists x\ x \dot{<} a$ but $a : [1 \,..\, 10] \not\models_{\mathcal{Z}} \exists x\ (5 \dot{<} x \wedge x \dot{<} a)$.

If e is a term or a constraint, $\mathbf{y} = (y_1, \ldots, y_k)$ is a tuple of distinct variables containing the free variables of e, and $\mathbf{t} = (t_1, \ldots, t_k)$, we denote by $e[\mathbf{t}/\mathbf{y}]$ the result of simultaneously replacing each free occurrence of y_i in e by t_i, possibly after renaming e's bound variables as needed to avoid variable capturing. We will write just $e[\mathbf{t}]$ when $\mathbf{y}$ is clear from context. With a slight abuse of notation, when $\mathbf{x}$ is a tuple of distinct variables, we will write $e[\mathbf{x}]$ to denote that the free variables of e are included in $\mathbf{x}$.

[6] Note that a ground or closed constraint can contain parameters.

For any valuation α, a tuple $\mathbf{m}$ of integer constants is *an α-solution of* a constraint $c[\mathbf{x}]$ if $\alpha \models_{\mathcal{Z}} c[\mathbf{m}]$. For instance, $\{a \mapsto 3\} \models_{\mathcal{Z}} c[4,1]$, where $c[x,y] = (a \doteq x - y)$.

With tuples $\mathbf{s} = (s_1, \ldots, s_n)$ and $\mathbf{t} = (t_1, \ldots, t_n)$ of terms, we will use the following abbreviations:

$$\mathbf{s} \doteq \mathbf{t} \stackrel{\text{def}}{=} s_1 \doteq t_1 \wedge \cdots \wedge s_n \doteq t_n$$

$$\mathbf{s} \dot{\leq} \mathbf{t} \stackrel{\text{def}}{=} s_1 \dot{\leq} t_1 \wedge \cdots \wedge s_n \dot{\leq} t_n \qquad \mathbf{s} \dot{<} \mathbf{t} \stackrel{\text{def}}{=} \mathbf{s} \dot{\leq} \mathbf{t} \wedge \neg(\mathbf{s} \doteq \mathbf{t})$$

$$\mathbf{s} \dot{<}_\ell \mathbf{t} \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } n = 0 \\ s_1 \dot{<} t_1 \vee (s_1 \doteq t_1 \wedge (s_2, \ldots, s_n) \dot{<}_\ell (t_2, \ldots, t_n)) & \text{if } n > 0 \end{cases}$$

$$\mathbf{s} \dot{\leq}_\ell \mathbf{t} \stackrel{\text{def}}{=} \mathbf{s} \doteq \mathbf{t} \vee \mathbf{s} \dot{<}_\ell \mathbf{t}$$

It should be clear that $\mathbf{s} \dot{\leq} \mathbf{t}$ denotes the component-wise extension of the integer ordering $\leq$ to integer tuples, $\mathbf{s} \dot{\leq}_\ell \mathbf{t}$ denotes the lexicographic extension of $\leq$, and $\mathbf{s} \dot{<} \mathbf{t}$ and $\mathbf{s} \dot{<}_\ell \mathbf{t}$ the strict versions of those.

The example in the introduction demonstrated the role of minimal solutions of (satisfiable) constraints. However, minimal solutions need not always exist—consider e.g. the constraint $x \dot{<} 0$. We say that a constraint c is *admissible* iff for all parameter valuations α, if c is α-satisfiable then the set of α-solutions of c contains finitely many minimal elements with respect to $\dot{\leq}$, each of which we call a *minimal α-solution of c*. *From now on we always assume that all constraints are admissible.* Note that admissibility can be easily enforced by conjoining a given constraint $c[\mathbf{x}]$ with the constraint $\mathbf{n} \dot{\leq} \mathbf{x}$ for some tuple $\mathbf{n}$ of integer constants.

As indicated in Section 2, the calculus needs to analyse constraints and their minimal solutions. We stress that for the calculus to be effective, it need not actually *compute* minimal solutions. Instead, it is enough for it to work with constraints that *denote* each of the minimal α-solutions $m_1, ..., m_n$ of an α-satisfiable constraint $c[\mathbf{x}]$. This can be done with the formulas $\mu_k\, c$ defined below, where $\mathbf{y}$ is a tuple of fresh variables with the same length as $\mathbf{x}$ and $k \geq 1$.[7]

$$\mu\, c \stackrel{\text{def}}{=} c \wedge \forall \mathbf{y}\, (c[\mathbf{y}] \rightarrow \neg(\mathbf{y} \dot{<} \mathbf{x})) \qquad \mu_\ell\, c \stackrel{\text{def}}{=} c \wedge \forall \mathbf{y}\, (c[\mathbf{y}] \rightarrow \mathbf{x} \dot{\leq}_\ell \mathbf{y})$$

$$\mu_k\, c \stackrel{\text{def}}{=} \mu_\ell\, (\neg(\mu_1\, c) \wedge \cdots \wedge \neg(\mu_{k-1}\, c) \wedge (\mu\, c))$$

Recalling that c is admissible, it is easy to see that for any valuation α, $\mu\, c$ has at most n α-solutions (for some n): the n minimal α-solutions of c, if any. If c is α-satisfiable, let $m_1, ..., m_n$ be the enumeration of these solutions in the lexicographic order $\dot{\leq}_\ell$. Observing that $\dot{\leq}_\ell$ is a linearization of $\dot{\leq}$, it is also easy to see that $\mu_\ell\, c$ has exactly one α-solution: m_1. Similarly, for $k = 1, \ldots, n$, $\mu_k\, c$ has exactly one α-solution: m_k (this is thanks to the additional constraint $\neg(\mu_1\, c) \wedge \cdots \wedge \neg(\mu_{k-1}\, c)$, which excludes the previous minimal α-solutions, denoted by $\mu_1\, c, \ldots, \mu_{k-1}$. For $k > n$, $\mu_k\, c$ is never α-satisfiable. This is a formal statement of these claims:

[7] The notations $\forall \mathbf{x}\, c$ and $\exists \mathbf{x}\, c$ stand just for c when $\mathbf{x}$ is empty.

Lemma 3.1 *Let α be an assignment and c an admissible constraint. Then, there is an $n \geq 0$ such that $\mu_1\, c, \ldots, \mu_n\, c$ have unique, pairwise different α-solutions, which are all minimal α-solutions of c. Furthermore, for all $k > n$, $\mu_k\, c$ is not α-satisfiable.*

For example, if $c[(x,y)] = a \mathrel{\dot{\leq}} x \wedge a \mathrel{\dot{\leq}} y \wedge \neg(x \doteq y)$ then $\mu_\ell c$ is semantically equivalent ($\equiv$) to $x \doteq a \wedge y \doteq a+1$, $\mu\, c \equiv (x \doteq a \wedge y \doteq a+1) \vee (x \doteq a+1 \wedge y \doteq a)$, $\mu_1\, c \equiv (x \doteq a \wedge y \doteq a+1)$, $\mu_2\, c = (x \doteq a+1 \wedge y \doteq a)$ and $\mu_3\, c$ is not α-satisfiable, for any α.

In Section 2 we introduced the (informal) notion of a *permanently valid* literal. In the current terminology, a (ground) literal, say $P(a+1,a)$ is permanently valid if there is a literal in the semantic tree $P(x,y) \mid c_k$ such that the tuple $(a+1,a)$ is $c[x,y]$'s least α-solution wrt. $\dot{<}_{\mu_\ell}$, for all $\alpha \in \Gamma$, where c_k admits only α-solutions that are greater or equal wrt. $\dot{\leq}$ than $\mu_k\, c$, for some constraint c.

As we will see later, the calculus compares lexicographically minimal α-solutions of constraints that have a *single* minimal solution. With such constraints it is enough to compare their least α-solutions with respect to $\dot{\leq}_\ell$. This is done with the following comparison operators over constraints, where $\mathbf{x}$ and $\mathbf{y}$ are disjoint vectors of variables of the same length:

$$c \mathrel{\dot{<}_{\mu_\ell}} d \quad \stackrel{\text{def}}{=} \quad \exists \mathbf{x}\, \exists \mathbf{y}\, (\mu_\ell\, c[\mathbf{x}] \wedge \mu_\ell\, d[\mathbf{y}] \wedge \mathbf{x} \mathrel{\dot{<}_\ell} \mathbf{y}) \ .$$

In words, for every α, the formula $c \mathrel{\dot{<}_{\mu_\ell}} d$ is α-satisfiable iff the least α-solutions of c and d exist, and the former is $\dot{<}_\ell$-smaller than the latter. Similarly, we write

$$c \mathrel{\dot{=}_{\mu_\ell}} d \quad \stackrel{\text{def}}{=} \quad \exists \mathbf{x}\, (\mu_\ell\, c[\mathbf{x}] \wedge \mu_\ell\, d[\mathbf{x}])$$

to denote the formula expressing that $c[\mathbf{x}]$ and $d[\mathbf{x}]$ have the same least solution.

From the above, it is not difficult to show the following.

Lemma 3.2 (Total ordering) *Let α be a parameter valuation, and $c[\mathbf{x}]$ and $d[\mathbf{x}]$ two α-satisfiable (admissible) constraints. Then, exactly one of the following cases applies: (i) $\alpha \models_{\mathcal{Z}} c \mathrel{\dot{<}_{\mu_\ell}} d$, (ii) $\alpha \models_{\mathcal{Z}} c \mathrel{\dot{=}_{\mu_\ell}} d$, or (iii) $\alpha \models_{\mathcal{Z}} d \mathrel{\dot{<}_{\mu_\ell}} c$.*

We stress that the restriction to α-satisfiable constraints is essential here. If c or d is not α-satisfiable, then none of the listed cases applies.

3.1 Constrained Clauses

We now expand the signature $\Sigma^{\Pi}_{\mathcal{Z}}$ with a finite set of free predicate symbols, and denote the resulting signature by Σ. The language of our logic is made of sets of *admissible constrained Σ-clauses*, defined below. The semantics of the logic consists of all the expansions of the integer structure to the signature Σ, the *Σ-expansions* of $\mathcal{Z}$.

A *normalized literal* is an expression of the form $(\neg)p(\mathbf{x})$ where p is a n-ary free predicate symbol of Σ and $\mathbf{x}$ is an n-tuple of *distinct* variables. We write $L(\mathbf{x})$ to denote that L is a normalized literal whose argument tuple is *exactly* $\mathbf{x}$.

A *normalized clause* is an expression $C = L_1(\mathbf{x}_1) \vee \cdots \vee L_n(\mathbf{x}_n)$ where $n \geq 0$ and each $L_i(\mathbf{x}_i)$ is a normalized literal, called a *literal in* C. We write $C(\mathbf{x})$ to indicate that C is a normalized clause whose variables are exactly $\mathbf{x}$. We denote the empty clause by $\Box$.

A *(constrained Σ-)clause* $D[\mathbf{x}]$ is an expression of the form $C(\mathbf{x}) \leftarrow c$ with the free variables of c included in $\mathbf{x}$. When C is $\Box$ we call D a *constrained empty clause*. A clause $C(\mathbf{x}) \leftarrow c$ is *LIA-(un)satisfiable* if there is an (no) Σ-expansion of the integer structure $\mathcal{Z}$ that satisfies $\forall \mathbf{x}\,(c \rightarrow C(\mathbf{x}))$. A set S of clauses and constraints is *LIA-(un)satisfiable* if there is an (no) Σ-expansion of $\mathcal{Z}$ that satisfies every element of S.

We will consider only *admissible clauses*, i.e., constrained clauses $C(\mathbf{x}) \leftarrow c$ where (i) $C \neq \Box$ and (ii) c is an admissible constraint. Condition (i) above is motivated by purely technical reasons. It is, however, no real restriction, as any clause $\Box \leftarrow c$ in a clause set S can be replaced by $\mathit{false} \leftarrow c$, where *false* is a 0-ary predicate symbol not in S, once S has been extended with the clause $\neg \mathit{false} \leftarrow \top$.[8] Condition (ii) is the real restriction, needed to guarantee the existence of minimal solutions, as explained earlier. To simplify the presentation, we will further restrict ourselves to clauses with (trivially admissible) constrains of the form $c[\mathbf{x}] \wedge \mathbf{0} \dot{\leq} \mathbf{x}$, where $\mathbf{0}$ is the tuple of all zeros. For brevity, in our examples we will sometimes leave the constraint $\mathbf{0} \dot{\leq} \mathbf{x}$ implicit.

4 Constrained Contexts

A *context literal* K is a pair $L(\mathbf{x}) \mid c$ where $L(\mathbf{x})$ is a normalized literal and c is an (admissible) constraint with free variables included in $\mathbf{x}$. We denote by $\overline{K}$ the constrained literal $\overline{L}(\mathbf{x}) \mid c$, where $\overline{L}$ is the complement of L.

A *(constrained) context* is a pair $\Lambda \cdot \Gamma$ where Γ is a finite set of closed constraints and Λ is a finite set of context literals. We will implicitly identify the sets Λ with their closure under renamings of a context literal's free variables.

In terms of the semantic tree presentation in Figure 1, each branch there corresponds (modulo a detail explained below) to a context $\Lambda \cdot \Gamma$, where Γ are the parameter constraints along the branch and Λ are the constrained literals. In the discussion of Figure 1 we explained informally the meaning of parameter constraints and constrained literals. The purpose of this section is to provide a formal account for that. We start with some preliminary definitions.

Definition 4.1 (α-Covers, α-Extends) Let α be a parameter valuation. A context literal $L(\mathbf{x}) \mid c_1$ *α-covers* a context literal $L(\mathbf{x}) \mid c_2$ if $\alpha \models_{\mathcal{Z}} \bar{\exists}\, c_2$ and $\alpha \models_{\mathcal{Z}} \bar{\forall}\, (c_2 \rightarrow c_1)$.

The literal $L(\mathbf{x}) \mid c_1$ *α-extends* $L(\mathbf{x}) \mid c_2$ if $L(\mathbf{x}) \mid c_1$ α-covers $L(\mathbf{x}) \mid c_2$ and $\alpha \models_{\mathcal{Z}} c_1 \doteq_{\mu_\ell} c_2$. If Γ is a set of closed constraints, $L(\mathbf{x}) \mid c_1$ *Γ-extends* $L(\mathbf{x}) \mid c_2$ if it α-extends it *for all* $\alpha \in \mathit{Mods}(\Gamma)$. $\Box$

[8] We will use $\top$ and $\bot$ respectively for the universally true and the universally false constraint.

For an unnormalized literal $L(\mathbf{t})$ we say that $L(\mathbf{x}) \mid c_1[\mathbf{x}]$ *α-covers* $L(\mathbf{t})$ if $L(\mathbf{x})$ covers the normalized version of $L(\mathbf{t})$, i.e., the literal $L(\mathbf{x}) \mid \pi\, \mathbf{x}\, (\mathbf{x} \doteq \mathbf{t}[\mathbf{z}/\mathbf{x}])$ where $\mathbf{z}$ is a tuple of fresh variables.

The intention of the previous definition is to compare context literals with respect to their set of solutions for a fixed valuation α. This is expressed basically by the second condition in the definition of α-covers. For example, $P(x) \mid a \dot{<} x$ α-covers $P(x) \mid a + 1 \dot{<} x$, for any α. The first condition ($\alpha \models_{\mathcal{Z}} \bar{\exists}\ c_2$) is needed to exclude α-coverage for trivial reasons, because c_2 is not α-satisfiable. Without it, for example, $P(x) \mid x \doteq 2$ would α-cover $P(x) \mid x \doteq a \wedge a \doteq 5$ when, say, $\alpha(a) = 3$, which is not intended. But note that $\alpha \not\models_{\mathcal{Z}} \exists x\, (x \doteq a \wedge a \doteq 5)$ in this case. Also note that the two conditions $\alpha \models_{\mathcal{Z}} \bar{\exists}\ c_2$ and $\alpha \models_{\mathcal{Z}} \bar{\forall}\ (c_2 \rightarrow c_1)$ in combination enforce that c_1 is α-satisfiable as well.

The notion of α-extension is similar to that of α-coverage, but applies to literals with the same least solutions only. For instance, $P(x) \mid 0 \dot{\leq} x \wedge x \dot{<} 7$ α-extends $P(x) \mid 0 \dot{\leq} x \wedge x \dot{<} 3$, and α-covers it, for any α (the least solution being 0 for both literals), and $P(x) \mid 3 \dot{<} x$ α-covers $P(x) \mid 7 \dot{<} x$ but does not α-extend it.

The concepts introduced in the next three definitions allow us to associate a set of structures to each context satisfying certain well-formedness conditions.

Definition 4.2 (α-Produces) Let Λ be a set of constrained literals and α a parameter valuation. A context literal $L(\mathbf{x}) \mid c_1$ *α-produces* a context literal $L(\mathbf{x}) \mid c_2$ *wrt.* Λ if

1. $L(\mathbf{x}) \mid c_1$ α-covers $L(\mathbf{x}) \mid c_2$, and
2. there is no $\overline{L}(\mathbf{x}) \mid d$ in Λ that α-covers $\overline{L}(\mathbf{x}) \mid c_2$ and such that $\alpha \models_{\mathcal{Z}} c_1 \dot{<}_{\mu_\ell} d$.

The set Λ *α-produces* a context literal K if some literal in Λ α-produces K wrt. Λ. A context $\Lambda \cdot \Gamma$ *produces* K if there is an $\alpha \in \mathit{Mods}(\Gamma)$ such that Λ α-produces K. □

Note that a context literal $L(\mathbf{x}) \mid c_1$ can α-produce a context literal $L(\mathbf{x}) \mid c_2$ only if both c_1 and c_2 are α-satisfiable.

As an example, if $\alpha(a) = 3$ then $P(x) \mid 2 \dot{<} x$ α-produces $P(5)$ wrt. $\Lambda = \{\neg P(x) \mid x \doteq a \wedge a \doteq 5\}$. Observe that neither $\alpha \models_{\mathcal{Z}} (2 \dot{<} x) \dot{<}_{\mu_\ell} (x \doteq a \wedge a \doteq 5)$ holds nor does $\neg P(x) \mid x \doteq a \wedge a \doteq 5$ α-cover $\neg P(5)$, as $x \doteq a \wedge a \doteq 5$ is not α-satisfiable. However, if $\alpha(a) = 5$ then $P(x) \mid 2 \dot{<} x$ no longer α-produces $P(5)$ wrt. Λ, because now $\alpha \models_{\mathcal{Z}} (2 \dot{<} x) \dot{<}_{\mu_\ell} (x \doteq a \wedge a \doteq 5)$ and $\neg P(x) \mid x \doteq a \wedge a \doteq 5$ α-covers $\neg P(5)$.

Definition 4.3 (α-Contradictory) Let $\Lambda \cdot \Gamma$ be a context and $\alpha \in \mathit{Mods}(\Gamma)$. A context literal $L(\mathbf{x}) \mid c$ is *α-contradictory with* Λ if there is a context literal $\overline{L}(\mathbf{x}) \mid d$ in Λ such that $\alpha \models_{\mathcal{Z}} c \doteq_{\mu_\ell} d$. It is *$\Gamma$-contradictory with* Λ if there is a $\overline{L}(\mathbf{x}) \mid d$ in Λ such that $\Gamma \models_{\mathcal{Z}} c \doteq_{\mu_\ell} d$.

The literal $L(\mathbf{x}) \mid c$ is *contradictory with* the context $\Lambda \cdot \Gamma$ if it is α-contradictory with Λ for some $\alpha \in \mathit{Mods}(\Gamma)$. The context $\Lambda \cdot \Gamma$ itself is *contradictory* if some context literal in Λ is contradictory with it. □

The notion of Γ-contradictory is based on equality of the least α-solutions of the involved constraints for all $\alpha \in \mathit{Mods}(\Gamma)$. It underlies the abandoning of candidate models due to permanently falsified clauses in Section 2, which is captured precisely as *closing literals* in Definition 4.8 below.

We require our contexts not only to be non-contradictory but also to constrain each parameter to a finite subset of $\mathbb{Z}$. Furthermore, they should guarantee that the associated Σ-expansions of $\mathcal{Z}$ are total over tuples of natural numbers. All this is achieved with *admissible* contexts.

Definition 4.4 (Admissible Γ, Admissible Context) A context $\Gamma \cdot \Lambda$ is *admissible* if

1. Γ is *admissible*, that is, Γ is satisfiable, and, for each parameter a in Π, there are integer constants $m, n \geq 0$ such that $\Gamma \models a : [m \,..\, n]$.

2. For each free predicate symbol P in Σ, the set Λ contains $\neg P(\mathbf{x}) \mid -\mathbf{1} \dot{\leq} \mathbf{x}$.

3. $\Lambda \cdot \Gamma$ is not contradictory.[9]

□

Thanks to Condition 2 in the above definition, an admissible context α-produces a literal $\neg P(\mathbf{n})$ with $\mathbf{n}$ consisting of non-negative integer constants, if no other literal in the context α-produces $P(\mathbf{n})$. Observe that admissible contexts $\Lambda \cdot \Gamma$ may contain context literals whose constraint is not α-satisfiable for some (or even all) $\alpha \in \mathit{Mods}(\Gamma)$. For those α's, such literals simply do not matter as their effect is null.

However, admissible contexts are always consistent in the following sense.

Lemma 4.5 (Consistent α-Productivity) *Let $\Lambda \cdot \Gamma$ be an admissible context and $\alpha \in \mathit{Mods}(\Gamma)$. For any context literal $L(\mathbf{x}) \mid c$, Λ cannot α-produce both $L(\mathbf{x}) \mid c$ and its complement $\overline{L}(\mathbf{x}) \mid c$.*

The following definition provides the formal account of the meaning of contexts announced at the beginning of this section.

Definition 4.6 (Induced Structure) Let $\Gamma \cdot \Lambda$ be an admissible context and let $\alpha \in \mathit{Mods}(\Gamma)$. The Σ-*structure $\mathcal{Z}_{\Lambda,\alpha}$ induced by Λ and α* is the expansion of $\mathcal{Z}$ to all the symbols in Σ that agrees with α on the parameters[10] and satisfies a positive ground literal $L(\mathbf{s})$ iff Λ α-produces $L(\mathbf{s})$. □

Lemma 4.7 *Let $\Lambda \cdot \Gamma$ be an admissible context and $\alpha \in \mathit{Mods}(\Gamma)$. For any ground literal $L(\mathbf{s})$ such that $\alpha \models_{\mathcal{Z}} \mathbf{0} \dot{\leq} \mathbf{s}$, $\mathcal{Z}_{\Lambda,\alpha}$ satisfies $L(\mathbf{s})$ if and only if Λ α-produces $L(\mathbf{s})$.*

[9] Equivalently, for every $\alpha \in \mathit{Mods}(\Gamma)$ and every pair of context literals $L(\mathbf{x}) \mid c$ and $\overline{L}(\mathbf{x}) \mid d$ in Λ, it is *not* the case that $\alpha \models_{\mathcal{Z}} c \doteq_{\mu_\ell} d$.

[10] That is, $\mathcal{Z}_{\Lambda,\alpha}$ interprets each parameter a as $\alpha(a)$.

Proof. This is a consequence of Lemma 4.5 and the presence of the literals $\neg P(\mathbf{x}) \mid -\mathbf{1} \dot{\leq} \mathbf{x}$ in admissible contexts.

Thus, Definition 4.6 connects syntax (α-productivity) to semantics (truth) in a one-to-one way.

In Section 2 we explained the derivation in Figure 1 as being driven by semantic considerations, to construct a model by successive branch extensions. The calculus' inference rules achieve that in their core by computing *context unifiers*.

Definition 4.8 (Context Unifier) Let $\Lambda \cdot \Gamma$ be an admissible context and $D[\mathbf{x}] = L_1(\mathbf{x}_1) \vee \cdots \vee L_k(\mathbf{x}_k) \leftarrow c[\mathbf{x}]$ a constrained clause with free variables $\mathbf{x}$. A *context unifier of D against $\Lambda \cdot \Gamma$* is a constraint

$$d[\mathbf{x}] = d'[\mathbf{x}] \wedge \exists \mathbf{y}\, (\mathbf{y} \dot{\leq} \mathbf{x} \wedge \mu_j\, d'[\mathbf{y}]), \quad \text{where } d'[\mathbf{x}] = c[\mathbf{x}] \wedge c_1[\mathbf{x}_1] \wedge \cdots \wedge c_k[\mathbf{x}_k] \qquad (1)$$

with each c_i coming from a literal $\overline{L_i}(\mathbf{x}_i) \mid c_i$ in Λ, and $j \geq 1$.

For each $i = 1, \ldots, k$, the context literal

$$L_i(\mathbf{x}_i) \mid d_i, \text{ with } d_i = \pi\, \mathbf{x}_i\, d \qquad (2)$$

is a *literal of* the context unifier. The literal $L_i(\mathbf{x}_i) \mid d_i$ is *closing* if $\Gamma \models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$. Otherwise, it is a *($\alpha$-)remainder literal (of d)* if there is an $\alpha \in \mathit{Mods}(\Gamma)$ such that $\alpha \models_{\mathcal{Z}} c_i \dot{<}_{\mu_\ell} d_i$ (equivalently, such that $\alpha \not\models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$ and d_i is α-satisfiable)[11].

The context unifier d is *closing* if each of its literals is closing. It is *(α-)productive* if for each $i = 1, \ldots, k$, the context literal $\overline{L_i}(\mathbf{x}_i) \mid c_i$ α-produces $\overline{L_i}(\mathbf{x}_i) \mid d_i$ wrt. Λ for some $\alpha \in \mathit{Mods}(\Gamma)$. □

The constraint d in (1) can be perhaps best understood as follows. Its component $d' = c[\mathbf{x}] \wedge c_1[\mathbf{x}_1] \wedge \cdots \wedge c_k[\mathbf{x}_k]$ denotes any simultaneous solution of D's constraint and the constraints coming from pairing each of D's literal with a context literal with same predicate symbol but opposite sign. The component $\mu_j d'[\mathbf{y}]$ denotes the j^{th} minimal solution of d', which bounds from below the solutions of d. A simple, but important consequence (for completeness) is that for any α and concrete solution $\mathbf{m}$ of d', j can be always chosen so that $d[\mathbf{m}]$ is α-satisfied. As a special case, when $\mathbf{m}$ is the j-th minimal solution of d', it is also the least solution of d. Regarding d_i in (2), for any α, the set of α-solutions of d_i is the projection over the vector $\mathbf{x}_i$ of the solutions of d.

A formal statement of the above is expressed by the following lemma.

Lemma 4.9 (Lifting) *Let $\Lambda \cdot \Gamma$ be an admissible context, $\alpha \in \mathit{Mods}(\Gamma)$, $D[\mathbf{x}] = L_1(\mathbf{x}_1) \vee \cdots \vee L_k(\mathbf{x}_k) \leftarrow c[\mathbf{x}]$ with $k \geq 1$ a constrained clause, and $\mathbf{m}$ a vector of constants from $\mathbb{Z}$. If $\mathcal{Z}_{\Lambda,\alpha}$ falsifies $D[\mathbf{m}]$, then there is an α-productive context unifier d of D against $\Lambda \cdot \Gamma$ where $\mathbf{m}$ is an α-solution of d.*

[11] Observe that if d_i is α-satisfiable so are d and c_i.

As an example (with no parameters, for simplicity), let $d' = c[x_1, x_2] \wedge c_1[x_1] \wedge c_2[x_2]$ where

$$c = \neg(x_1 \doteq x_2), \quad c_1 = 1 \dot{\leq} x_1, \quad c_2 = 1 \dot{\leq} x_2 \ .$$

Then, the (unique) solution of $\mu_j\, d'$ for $j = 1$ is $(1, 2)$; for $j = 2$ it is $(2, 1)$. By fixing $j = 1$ now let us commit to $(1, 2)$. Then the solutions of d_1 are $(1), (2), \ldots$ and the solutions of d_2 are $(2), (3), \ldots$ The least solution of d_1, (1), coincides with the projection over x_1 of the commited minimal solution $(1, 2)$. Similarly for d_2. This is no accident and is crucial in proving the soundness of the calculus. It relies on the property that the least (individual) solutions of all the d_i's are, in combination, the least solution of d—which is in turn the first minimal solution of d'. In the example, the least solutions of d_1 and d_2 are 1 and 2, respectively, and combine into $(1, 2)$, the least solution of d.

We stress that all the notions in the above definition are effective thanks to the decidability of LIA. A subtle point here is the choice of j in (1), as j is not bounded *a priori*. However, all these notions hold only if d_i is α-satisfiable for some or all (finitely) many choices of $\alpha \in \mathit{Mods}(\Gamma)$, and that d_i becomes α-unsatisfiable if j exceeds the number of minimal α-solutions of d_i. By this argument, the possible values for j are effectively bounded.

Example 4.10 Consider the context

$$\{P(x) \mid a \dot{<} x\} \cdot \{a : [1\,..\,10], b : [1\,..\,10]\}$$

and the input clause $\neg P(x) \leftarrow b \dot{<} x$. The context corresponds to the left branch in Figure 1b. There is a context unifier, for any $j \geq 1$, $d = a \dot{<} x \wedge b \dot{<} x \wedge \exists y\, (y \dot{\leq} x \wedge \mu_j\, (a \dot{<} y \wedge b \dot{<} y))$. Its literal is $K' = \neg P(x) \mid d_1$, where $d_1 = \pi\, x\, d(= d)$. The constraint $(a \dot{<} y \wedge b \dot{<} y)$ has a unique minimal α-solution, which is also its least α-solution. Thus, d is equivalent to $a \dot{<} x \wedge b \dot{<} x$, obtained with $j = 1$. Let us analyze if d is closing, that is, if $\Gamma \models_{\mathcal{Z}} (a \dot{<} x) \doteq_{\mu_\ell} d$. That is true iff

$$\Gamma \models_{\mathcal{Z}} \exists x\, \mu_\ell\, (a \dot{<} x) \wedge \mu_\ell\, (a \dot{<} x \wedge b \dot{<} x) \ . \tag{3}$$

By quantifier elimination we can show that checking (3) reduces to checking if $\Gamma \models_{\mathcal{Z}} \neg(a \dot{<} b)$. Since that entailment does not hold, K' is not closing, and neither is d.

Now take any $\alpha \in \mathit{Mods}(\Gamma)$ that falsifies $\neg(a \dot{<} b)$, and hence satisfies $a \dot{<} b$. Clearly, d_1 is α-satisfiable. According to Definition 4.8 then, K' is a remainder literal of d. Let us verify then that, as stated in the definition,

$$\alpha \models_{\mathcal{Z}} (a \dot{<} x) \dot{<}_{\mu_\ell} (a \dot{<} x \wedge b \dot{<} x) \ . \tag{4}$$

This can be done by quantifier elimination again, reducing the formula in (4) to $a \dot{<} b$. □

5 The Calculus

The inference rules of the calculus are defined over triples, *sequents*, of the form $\Lambda \cdot \Gamma \vdash \Phi$ where $\Lambda \cdot \Gamma$ is an admissible context and Φ is a set of constrained clauses.

The completeness of the calculus guarantees that each of its branches terminates with failure, i.e., contains the clause $\square \leftarrow \top$, when Φ is LIA-unsatisfiable. Contrapositively, from any unfailing branch it is possible to extract (possibly in the limit) a set of models for Φ. When the branch is finite and ends with a sequent $\Lambda_n \cdot \Gamma_n \vdash \Phi$, these models are precisely those denoted by $\Lambda_n \cdot \Gamma_n$.

Context unifiers play a crucial role in the evolution of $\Lambda \cdot \Gamma$. To illustrate their use, consider a sequent $\Lambda \cdot \Gamma \vdash \Phi$. If for some $\alpha \in \mathit{Mods}(\Gamma)$ the structure $\mathcal{Z}_{\Lambda,\alpha}$ induced by Λ and α falsifies Φ, it must falsify a "ground" instance $D[\mathbf{m}]$ of some clause D in Φ. As shown in the appendix, this implies the existence of an α-productive context unifier d of D against $\Lambda \cdot \Gamma$ where $\mathbf{m}$ is an α-solution of d.

If d has an α-remainder literal $K'_i = L(\mathbf{x}_i) \mid d_i$ not contradictory with the context, the problem with $D[\mathbf{m}]$ can be fixed by adding K'_i to Λ. In fact, if $\mathbf{m}_i$ is the projection of $\mathbf{m}$ over $\mathbf{x}_i$, then K'_i will α-produce $L_i(\mathbf{m}_i)$ in the new context, as its least solution is no greater than $\mathbf{m}_i$.[12] That will make the new $\mathcal{Z}_{\Lambda,\alpha}$ satisfy $L_i(\mathbf{m}_i)$ and so $D[\mathbf{m}]$ as well. This is essentially what the calculus does to $\Lambda \cdot \Gamma \vdash \Phi$ with the rules $\mathsf{Split}(d)$ or $\mathsf{Extend}(d)$ introduced below. If each α-remainder literal of d is contradictory with the context, it will be β-contradictory with Λ for one or more $\beta \in \mathit{Mods}(\Gamma)$. Then, it is necessary to strengthen Γ to eliminate the offending β's, which is achieved with the $\mathsf{Domain\ Split}(d)$ rule. Strengthening Γ either makes $\mathsf{Split}(d)$ or $\mathsf{Extend}(d)$ applicable to an α-remainder literal of d or turns all literals of d into closing ones. In the latter case, the calculus will close the corresponding branch with the $\mathsf{Close}(d)$ rule.

5.1 Derivation Rules

The $\mathcal{ME}(\mathrm{LIA})$ calculus has the following five derivation rules, where the last is optional. Applications of the other rules are subject to certain fairness criteria, explained later. In the rules, the notation Φ, D abbreviates $\Phi \cup \{D\}$. (Similarly for Λ, K and Γ, c.)

$$\mathsf{Close}(d) \quad \frac{\Lambda \cdot \Gamma \vdash \Phi, D}{\Lambda \cdot \Gamma \vdash \Phi, D, \square \leftarrow \top} \quad \text{if } \begin{cases} (\square \leftarrow \top) \notin \Phi \cup \{D\}, \text{ and} \\ d \text{ is a closing context unifier of } D \text{ against } \Lambda \cdot \Gamma. \end{cases}$$

This rule recognizes that the context not only falsifies some input clause D but is also unfixable, and adds the empty clause as a marker for that.

$$\mathsf{Split}(d) \quad \frac{\Lambda \cdot \Gamma \ \vdash \ \Phi, D}{(\Lambda, L_i \mid d_i) \cdot \Gamma \ \vdash \ \Phi, D \qquad (\Lambda, \overline{L_i} \mid d_i) \cdot \Gamma \ \vdash \ \Phi, D} \quad \text{if } \star$$

[12] This is the analogous of "lifting" in a Herbrand-based theorem proving.

where

$$\star = \begin{cases} d \text{ is a context unifier of } D \text{ against } \Lambda \cdot \Gamma, \\ L_i \mid d_i \text{ is a remainder literal of } d, \text{ and} \\ \text{neither } L_i \mid d_i \text{ nor } \overline{L_i} \mid d_i \text{ is contradictory with } \Lambda \cdot \Gamma. \end{cases}$$

This rule, analogous to the main rule of the DPLL procedure, derives one of two possible sequents non-deterministically. The left-hand side conclusion chooses to fix the context by adding $L_i \mid d_i$ to Λ. The right-hand side branch is needed for soundness, in case the left-hand side fix leads to an application of Close. It causes progress in the derivation by making $L_i \mid d_i$ Γ-contradictory with the context, which forces the calculus to consider other alternatives to $L_i \mid d_i$.

$$\mathsf{Extend}(d) \qquad \frac{\Lambda \cdot \Gamma \qquad \vdash \Phi, D}{(\Lambda, L_i \mid d_i) \cdot \Gamma \vdash \Phi, D} \quad \text{if} \quad \begin{cases} d \text{ is a context unifier of } D \text{ against } \Lambda \cdot \Gamma, \\ L_i \mid d_i \text{ is a remainder literal of } d, \\ \overline{L_i} \mid d_i \text{ is } \Gamma\text{-contradictory with } \Lambda, \text{ and} \\ \text{there is no } K \text{ in } \Lambda \text{ that } \Gamma\text{-extends } L_i \mid d_i. \end{cases}$$

This rule can be seen as a one-branched Split. If $\overline{L_i} \mid d_i$ is Γ-contradictory with Λ, the only way to fix the context is to add $L_i \mid d_i$ to it. Its last precondition is a redundancy test—which also prevents a repeated application of the rule with the same literal.

To illustrate the need of Extend, suppose $\Lambda = \{\neg P(x) \mid -1 \dot{\leq} x,\ P(x) \mid x : [1\,..\,5]\}$, $\Gamma = \emptyset$ and $D = P(x) \leftarrow x : [1\,..\,7]$. The clause D is falsified in the (single) induced interpretation[13]. Adding $P(x) \mid x : [1\,..\,7]$ to Λ will fix the problem. However, Split cannot be used for that since $\neg P(x) \mid x : [1\,..\,7]$ is Γ-contradictory with Λ—for having the same least solution, 1, as the constraint of $P(x) \mid x : [1\,..\,5]$. Extend will do instead.

$$\mathsf{Domain\ Split}(d) \qquad \frac{\Lambda \cdot \Gamma \ \vdash \ \Phi, D}{\Lambda \cdot (\Gamma, c \doteq_{\mu_\ell} d_i) \ \vdash \ \Phi, D \qquad \Lambda \cdot (\Gamma, \neg(c \doteq_{\mu_\ell} d_i)) \ \vdash \ \Phi, D} \quad \text{if } \star$$

where

$$\star = \begin{cases} d \text{ is a context unifier of } D \text{ against } \Lambda \cdot \Gamma, \\ \text{there is a literal } L_i \mid d_i \text{ of } d, \text{ and} \\ \text{there is } \overline{L_i} \mid c \text{ or } L_i \mid c \text{ in } \Lambda \text{ s.t.} \\ \quad \alpha \models_{\mathcal{Z}} c \doteq_{\mu_\ell} d_i, \text{ for some } \alpha \in \mathit{Mods}(\Gamma), \text{ and} \\ \quad \Gamma \not\models_{\mathcal{Z}} c \doteq_{\mu_\ell} d_i. \end{cases}$$

The purpose of this rule is to enable later applications of the other rules that are not applicable to the current context. It does that by partitioning the current $\mathit{Mods}(\Gamma)$ in two non-empty parts.

Observe that adding parameters with finite domains to the input language, as we did, does not increase its expressivity; in principle, they can be eliminated by an exhaustive

[13] Because, for instance, $\neg P(6)$ is true in it.

case analysis over all the possible values. In practice, however, doing that can be prohibitively expensive, depending on the size of the parameters' domains. In contrast, this calculus treats constants much more efficiently, by doing case analysis on parameters only *on demand*, as manifested in the applicability conditions of Domain Split.

$$\textsf{Ground Split} \quad \frac{\Lambda \cdot \Gamma \ \vdash \ \Phi}{\Lambda \cdot (\Gamma, l) \ \vdash \ \Phi \quad \Lambda \cdot (\Gamma, \bar{l}) \ \vdash \ \Phi} \quad \text{if } \star$$

where

$$\star = \begin{cases} l \text{ is a ground atomic constraint over the parameters } \Pi, \\ \alpha \models_{\mathcal{Z}} l, \text{ for some } \alpha \in \mathit{Mods}(\Gamma), \text{ and} \\ \Gamma \not\models_{\mathcal{Z}} l. \end{cases}$$

This optional rule adds another, more flexible, way to do case analyses on the parameters. It can improve efficiency in particular when paired with a suitable quantifier elimination procedure for LIA. In that case, one can replace each application of Domain Split, adding a constraint $[\neg](c \doteq_{\mu_\ell} d_i)$ to Γ, with one application of Ground Split where the chosen ground atomic constraint l is computed from $[\neg](c \doteq_{\mu_\ell} d_i)$ by the QE procedure, and is so that either it or its complement $\bar{l}$ entails $c \doteq_{\mu_\ell} d_i$. The net effect is that Γ grows only with ground literals, making tests involving it potentially cheaper.

It is not too difficult to see that the non-optional derivation rules are mutually exclusive, in the sense that for a given sequent at most one of them is applicable to the same clause D, context unifier d, and literal of d.

5.2 Derivations

Derivations in the $\mathcal{ME}$(LIA) calculus are defined in terms of *derivation trees*, where each node corresponds to a particular application of a derivation rule, and each of the node's children corresponds to one of the conclusions of the rule. More precisely, a derivation tree is a labeled tree inductively defined as follows.

Let Φ be an admissible clause set and Γ an admissible set of closed constraints. A one-node tree is a derivation tree (of Φ and Γ) iff its root is labeled with an *initial sequent for* Φ *and* Γ, that is, a sequent of the form $\Lambda \cdot \Gamma \vdash \Phi$, where Λ contains (only) the constraint literal $\neg p(\mathbf{x}) \mid -\mathbf{1} \dot{\leq} \mathbf{x}$ for each free predicate symbol p in Σ. It is easy to see that the context $\Lambda \cdot \Gamma$ is admissible.

A tree $\mathbf{T}'$ is a derivation tree iff it is obtained from a derivation tree $\mathbf{T}$ by adding to a leaf node N in $\mathbf{T}$ new children nodes $N_1, \ldots, N_m$ so that the sequents labeling $N_1, \ldots, N_m$ can be derived by applying a rule of the calculus to the sequent labeling N. In this case, we say that $\mathbf{T}'$ *is derived from* $\mathbf{T}$. When it is convenient and it does not cause confusion, we will identify the nodes of a derivation tree with their labels.[14]

[14]The formal framework of derivation trees over sequents could be simplified to some degree, as the clause set Φ does not change and both Λ and Γ only grow with the derivation, but never shrink. The current form can be justified with a view to adding simplification rules later, which will allow to modify or remove elements in Φ, Λ and Γ. Also, the $\mathcal{ME}$(LIA) calculus will be easier to grasp for readers familiar with the $\mathcal{ME}$ calculus, which is presented in the same style.

We say that a branch in a derivation tree is *closed* if its leaf is labeled by a sequent of the form $\Lambda \cdot \Gamma \vdash \Phi, \square \leftarrow \top$; otherwise, the branch is *open*. A derivation tree is *closed* if each of its branches is closed, and it is *open* otherwise. We say that a derivation tree (of Φ and Γ) is a *refutation tree (of Φ and Γ)* iff it is closed.

Definition 5.1 (Derivation) Let Φ be an admissible clause set and Γ an admissible set of closed constraints. A *derivation (in $\mathcal{ME}$(LIA)) of Φ and Γ* is a possibly infinite sequence of derivation trees $\mathcal{D} = (\mathbf{T}_i)_{i<\kappa}$, such that $\mathbf{T}_0$ is a one-node tree whose root is labeled with an initial sequent for Φ and Γ, and for all i with $0 < i < \kappa$, $\mathbf{T}_i$ is derived from $\mathbf{T}_{i-1}$. □

We say that $\mathcal{D}$ is a *refutation of Φ and Γ* iff $\mathcal{D}$ is finite and ends with a refutation tree of Φ and Γ.

We show below that the $\mathcal{ME}$(LIA) calculus is sound and (strongly) complete in the following sense: for all admissible clause sets Φ and admissible sets of closed constraints Γ, $\Phi \cup \Gamma$ is unsatisfiable iff every fair derivation of Φ and Γ is a refutation of Φ and Γ.

6 Correctness of the Calculus

6.1 Soundness

Proposition 6.1 (Soundness) *For all admissible clause sets Φ and admissible sets of closed constraints Γ, if there is a refutation tree of Φ and Γ, then $\Gamma \cup \Phi$ is LIA-unsatisfiable.*

In essence, and leaving Γ aside, the proof is by first deriving a binary tree over ground, parameter-free literals that reflects the applications of the derivation rules in the construction of the given refutation tree. For instance, a Split application with its new constraint literal $L(\mathbf{x}) \mid c$ in the left context gives rise to the literal $L(\mathbf{m})$, where $\mathbf{m}$ is the least α-solution of c for a given α. In the resulting tree neighbouring nodes will be labelled with complementary literals, like $L(\mathbf{m})$ and $\neg L(\mathbf{m})$. In the second step it is shown that this binary tree is closed by ground instances from the input set. It is straightforward then to argue that $\Phi \cup \Gamma$ is LIA-unsatisfiable.

6.2 Fairness

To prove the calculus' completeness we will introduce the notion of an *exhausted branch*, in essence, a (limit) derivation tree branch that need not be extended any further by the calculus and that is obtained by a fair derivation.

The specific notion of fairness that we adopt is defined formally in the following. For that, it will be convenient to describe a tree $\mathbf{T}$ as the pair $(\mathbf{N}, \mathbf{E})$, where $\mathbf{N}$ is the set of the nodes of $\mathbf{T}$ and $\mathbf{E}$ is the set of the edges of $\mathbf{T}$. In the rest of the section, we will use κ to denote a countable (possibly infinite) cardinal, and i, j to denote finite cardinals.

Each derivation $\mathcal{D} = (\mathbf{T}_i)_{i<\kappa} = ((\mathbf{N}_i, \mathbf{E}_i))_{i<\kappa}$ in the calculus determines a *limit tree* $\mathbf{T} := (\bigcup_{i<\kappa} \mathbf{N}_i, \bigcup_{i<\kappa} \mathbf{E}_i)$. It is easy to show that a limit tree of a derivation $\mathcal{D}$ is indeed a tree. But it will not be a derivation tree unless $\mathcal{D}$ is finite.

Definition 6.2 (Limit Context and Clause Set) Let $\mathbf{T}$ be the limit tree of some derivation, and let $\mathbf{B} = (N_i)_{i<\kappa}$ be a branch in $\mathbf{T}$ with κ nodes. Let $\Lambda_i \cdot \Gamma_i \vdash \Phi_i$ be the sequent labeling node N_i, for all $i < \kappa$. We define the *limit context* of $\mathbf{B}$ as $\Lambda_\mathbf{B} \cdot \Gamma_\mathbf{B} := (\bigcup_{i<\kappa} \Lambda_i) \cdot (\bigcup_{i<\kappa} \Gamma_i)$ and the *limit clause set* of $\mathbf{B}$ as $\Phi_\mathbf{B} := \bigcup_{i<\kappa} \Phi_i$. □

Although, strictly speaking, $\Lambda_\mathbf{B} \cdot \Gamma_\mathbf{B}$ is not a context because $\Lambda_\mathbf{B}$ may be infinite, for the purpose of the completeness proof we treat it as one. This is possible because all relevant definitions (in particular Definition 4.8) can be applied without change to $\Lambda_\mathbf{B} \cdot \Gamma_\mathbf{B}$ as well.

One of the main technical notions needed to prove the calculus' completeness is that of an *exhausted branch*, in essence, a (limit) derivation tree branch that need not be extended any further. It is based on the notion of *redundant context unifiers*.

Definition 6.3 (Redundant Context Unifier) Let $\Lambda_1 \cdot \Gamma_1$ and $\Lambda_2 \cdot \Gamma_2$ be admissible contexts, $\alpha \in \mathit{Mods}(\Gamma_1)$ and D a clause. A context unifier d of D against $\Lambda_1 \cdot \Gamma_1$ is *α-redundant in $\Lambda_2 \cdot \Gamma_2$* if

1. Λ_2 α-produces some literal of d, or
2. $\mathit{Mods}(\Gamma_2) \subsetneq \mathit{Mods}(\Gamma_1)$

We say that d is *redundant in $\Lambda_2 \cdot \Gamma_2$* if it is α-redundant in $\Lambda_2 \cdot \Gamma_2$ for all $\alpha \in \mathit{Mods}(\Gamma)$. □

If condition (1) applies then the interpretation induced by Λ_2 and α will already satisfy D.[15] There is no point then considering a derivation rule application based on that d. Condition (2) allows us to discard an existing derivation rule application when the constraints in Γ are strengthened.

Now, an *exhausted (limit) branch* (i) satisfies whenever Split, Extend or Domain Split is applicable to some of its sequents, based on an α-productive context unifier, then this context unifier is α-redundant in the context of some later sequent (a sequent more distant from the root), (ii) cannot be applied Close, and (iii) does not contain $\square \leftarrow \top$.

Definition 6.4 (Exhausted branch) Let $\mathbf{T}$ be a limit tree, and let $\mathbf{B} = (N_i)_{i<\kappa}$ be a branch in $\mathbf{T}$ with κ nodes. For all $i < \kappa$, let $\Lambda_i \cdot \Gamma_i \vdash \Phi_i$ be the sequent labeling node N_i. The branch $\mathbf{B}$ is *exhausted* if for all $D \in \Phi_\mathbf{B}$ and for all $i < \kappa$ all of the following hold:

(i) For all $\alpha \in \mathit{Mods}(\Gamma_\mathbf{B})$, if Split($d$), Extend($d$) or Domain Split($d$) is applicable to $\Lambda_i \cdot \Gamma_i \vdash \Phi_i$ with selected clause $D \in \Phi_i$, where d is an α-productive context unifier of D against $\Lambda_i \cdot \Gamma_i$, then there is $j \geq i$ with $j < \kappa$ such that d is α-redundant in $\Lambda_j \cdot \Gamma_j$.

[15] Lemmas 4.5 and 4.7 in the Appendix provide a formal explanation for that.

(ii) $\mathsf{Close}(d)$ is not applicable to $\Lambda_i \cdot \Gamma_i \vdash \Phi_i$, for selected clause $D \in \Phi_i$ and any productive context unifier d of D against $\Lambda_i \cdot \Gamma_i$.

(iii) $D \neq (\Box \leftarrow \top)$.

□

As a consequence of the fact that any (admissible) context never α-produces both a context literal and its complement (Lemma 4.5) it follows that d is an α-productive context unifier if and only if it is not α-redundant in $\Lambda \cdot \Gamma$. Definition 6.4 requires us to consider (α)-productive context unifiers only, where $\alpha \in \mathit{Mods}(\Gamma)$. It follows that these (and only these) context unifiers are indeed non-redundant in the current context and need consideration.

Definition 6.5 (Fairness) A limit tree of a derivation is *fair* if it is a refutation tree or it has an exhausted branch. A derivation is *fair* if its limit tree is fair. □

We point out that fair derivations in the sense above exist and are computable for any set of Σ-clauses. A naive fair proof procedure, for instance, grows a branch until the conditions (ii) and (iii) in Definition 6.4 are violated, and turns to another branch to work on, if any, or otherwise applies the next Split, Extend or Domain Split taken from a FIFO queue, unless its context unifier is redundant. A similar proof procedure has been described for the ME calculus in [BFT06].

6.3 Completeness

For the rest of the section, let Φ be an admissible clause set, Γ a finite set of admissible closed constraints, and assume that $\mathcal{D}$ is a fair derivation of Φ and Γ that is not a refutation. Observe that $\mathcal{D}$'s limit tree must have at least one exhausted branch. We denote this branch by $\mathbf{B} = (N_i)_{i<\kappa}$. Then, by $\Lambda_i \cdot \Gamma_i \vdash \Phi_i$, we will always mean the sequent labeling the node N_i in $\mathbf{B}$, for all $i < \kappa$, and $\Lambda_0 \cdot \Gamma_0 \vdash \Phi_0$ is an initial sequent for Φ and Γ.

Lemma 6.6 *For all $i < \kappa$, Γ_i is satisfiable, and there is an $i < \kappa$ such that for all $j \geq i$, $\Gamma_i = \Gamma_j = \Gamma_{\mathbf{B}}$.*

The following proposition is the main result for proving the calculus complete.

Proposition 6.7 (Model Construction) *If $(\Box \leftarrow \top) \notin \Phi_{\mathbf{B}}$ then, for every $\alpha \in \mathit{Mods}(\Gamma_{\mathbf{B}})$, $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ is a model of $\Phi_{\mathbf{B}}$.*

It is worth noting that Proposition 6.7 never holds for the trivial reason that $\Gamma_{\mathbf{B}}$ is unsatisfiable.

The completeness of the calculus is a consequence of Proposition 6.7 and Lemma 6.6. We state it here in its contrapositive form to underline the model computation ability of $\mathcal{ME}$(LIA).

Theorem 6.8 (Completeness) *Let $\mathcal{D}$ be a fair derivation of Φ and Γ with limit tree $\mathbf{T}$. If $\mathbf{T}$ is not a refutation tree, then $\Phi \cup \Gamma$ is satisfiable; more specifically, for every exhausted branch $\mathbf{B}$ of $\mathbf{T}$ and for every $\alpha \in \mathit{Mods}(\Gamma_{\mathbf{B}})$, $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ is a model of $\Phi \cup \Gamma$.*

Note that the theorem includes a proof convergence result, that *every* fair derivation of an unsatisfiable clause set is a refutation. In practical terms, it implies that as long as a derivation strategy guarantees fairness, the order of application of the rules of the calculus is irrelevant for proving an input clause set unsatisfiable, giving to the $\mathcal{ME}$(LIA) calculus the same flexibility enjoyed by the DPLL calculus at the propositional level.

Proof. Assume that $\mathbf{T}$ is not a refutation tree and let $\mathbf{B}$ be an exhausted branch of $\mathbf{T}$. By definition, $(\Box \leftarrow \top) \notin \Phi_{\mathbf{B}}$.

By Lemma 6.6, $\Gamma_{\mathbf{B}}$ is satisfiable. Chose any $\alpha \in \mathit{Mods}(\Gamma_{\mathbf{B}})$ arbitrarily. With Proposition 6.7 conclude that $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ is a model of $\Phi_{\mathbf{B}}$.

The only derivation rule that manipulates the Φ_i's is Close, by adding $\Box \leftarrow \top$, but Close was not applied, as $(\Box \leftarrow \top) \notin \Phi_{\mathbf{B}}$. It follows $\Phi_{\mathbf{B}} = \Phi_i$, for all $i < \kappa$, and so $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ is a model of Φ_0 ($= \Phi$), too. (*)

From the definition of the derivation rules it follows that $\Gamma_0 \subseteq \Gamma_{\mathbf{B}}$. With $\alpha \in \mathit{Mods}(\Gamma_{\mathbf{B}})$ it follows that $\alpha \in \mathit{Mods}(\Gamma_0)$ ($= \mathit{Mods}(\Gamma)$). By definition, $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ agrees with α on the parameters. With (*) it follows that $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ is a model of $\Phi \cup \Gamma$, as desired. □

When the branch $\mathbf{B}$ in Theorem 6.8 is finite, $\Lambda_{\mathbf{B}}$ coincides with the context $\Lambda_n \cdot \Gamma_n$, say, in $\mathbf{B}$'s leaf. From a model computation perspective, this is a very important fact because it means that a model of the original clause set—or rather, a finite representation of it, $\Lambda_n \cdot \Gamma_n$—is readily available at the end of the derivation; it does not have to be computed from the branch, as in other model generation calculi.

7 Conclusions and Further Work

We have presented a basic version of $\mathcal{ME}$(LIA), a new calculus for a logic with restricted quantifiers and linear integer constraints. The calculus allows one to reason with certain useful extensions of linear integer arithmetic with relations and finite domain constants. With the restriction of variables to finite domains, implementations of the calculus have potential applications in formal methods and in planning, where they can scale better than current decision procedures based on weaker logics, such as propositional logic or function-free clause logic.

We are working on extending the set of derivation rules with rules analogous to the unit-propagation rule of DPLL, which are crucial for producing efficient implementations. With that goal, we are also working on refinements of the calculus that reduce the cost of processing LIA-constraints. We stress though that the basic version presented here is already geared toward efficiency for featuring a (semantically justified) redundancy criterion, by reduction to LIA's ordering constraints, that allows one to avoid inferences with clause instances satisfied by one of the current candidate models.

References

[Bau98] Peter Baumgartner. *Theory Reasoning in Connection Calculi*, volume 1527 of *Lecture Notes in Artificial Intelligence*. Springer, 1998.

[Bau07] Peter Baumgartner. Logical engineering with instance-based methods. In Frank Pfenning, editor, *CADE-21 – The 21st International Conference on Automated Deduction*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 404–409. Springer, 2007.

[BFdNT07] Peter Baumgartner, Alexander Fuchs, Hans de Nivelle, and Cesare Tinelli. Computing finite models by reduction to function-free clause logic. *Journal of Applied Logic*, 2007. In Press.

[BFT06] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal of Artificial Intelligence Tools*, 15(1):21–52, 2006.

[BGW94] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Refutational theorem proving for hierachic first-order theories. *Appl. Algebra Eng. Commun. Comput*, 5:193–212, 1994.

[BT03] Peter Baumgartner and Cesare Tinelli. The Model Evolution Calculus. In Franz Baader, editor, *CADE-19 – The 19th International Conference on Automated Deduction*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 350–364. Springer, 2003.

[Bür90] H.J. Bürckert. A Resolution Principle for Clauses with Constraints. In Mark E. Stickel, editor, *10th International Conference on Automated Deduction*, LNAI 449, pages 178–192, Kaiserslautern, FRG, July 24–27, 1990. Springer-Verlag.

[GBT07] Yeting Ge, Clark Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction (CADE-21), Bremen, Germany*, Lecture Notes in Computer Science. Springer, 2007.

[GK06] H. Ganzinger and K. Korovin. Theory Instantiation. In *Proceedings of the 13 Conference on Logic for Programming Artificial Intelligence Reasoning (LPAR'06)*, volume 4246 of *Lecture Notes in Computer Science*, pages 497–511. Springer, 2006.

[KV07] K. Korovin and A. Voronkov. Integrating linear arithmetic into superposition calculus. In *Computer Science Logic (CSL'07)*, volume 4646 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2007.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare 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, November 2006.

[NP07] Juan Antonio Navarro Peréz. *Encoding and Solving Problems in Effectively Propositional Logic*. PhD thesis, The University of Manchester, 2007.

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

[Sti85] M.E. Stickel. Automated Deduction by Theory Resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.

A Proofs

A.1 Section 3

Lemma 3.1 *Let α be an assignment and c an admissible constraint. Then, there is an $n \geq 0$ such that $\mu_1\, c, \ldots \mu_n\, c$ have unique, pairwise different α-solutions, which are all minimal α-solutions of c. Furthermore, for all $k > n$, $\mu_k\, c$ is not α-satisfiable.*

Proof. Let $\mathbf{m}_1, \ldots, \mathbf{m}_n$ be all minimal α-solutions of c, for some $n \geq 0$. They exist because c is admissible. Without loss of generality assume they are ordered lexicographically, i.e. $\mathbf{m}_i \leq_\ell \mathbf{m}_j$[16] whenever $1 \leq i \leq j \leq n$.

To prove the first part of the lemma it suffices to show that $\mathbf{m}_k$ is the unique α-solution of $\mu_k\, c$, for all $k = 1, \ldots, n$. It then follows that these solutions are pairwise different, because $\mathbf{m}_1, \ldots, \mathbf{m}_n$ are all different.

The case $n = 0$ being trivial, we assume $n > 0$ and prove the statement by induction on k, for all $k = 1, \ldots, n$.

If $k = 1$ then recall that by assumption $\mathbf{m}_1$ is the least of all minimal α-solutions of c. This fact is expressed in our constraint language as the constraint $\mu_\ell(\mu\, c)$ $(= \mu_1\, c)$, which has exactly one α-solution, $\mathbf{m}_1$.

If $k > 1$ then assume by induction that $\mathbf{m}_1, \ldots, \mathbf{m}_{k-1}$ are the unique α-solution of $\mu_1\, c, \ldots \mu_{k-1}\, c$, respectively. We have to prove this for $\mu_k\, c$.

By assumption, $\mathbf{m}_k$ is a minimal α-solution of c (the k-th one). Thus, $\mathbf{m}_k$ is an α-solution of $\mu\, c$. Because $\mathbf{m}_1, \ldots, \mathbf{m}_n$ are all pairwise different, it follows with the induction hypothesis (and $k \leq n$) that $\mathbf{m}_k$ is not an α-solution of any of the constraints $\mu_1\, c, , ..., \mu_{k-1}\, c$. Thus $\mathbf{m}_k$ is an α-solution of $\neg(\mu_1\, c) \wedge \cdots \wedge \neg(\mu_{k-1}\, c)$. It follows that $\mathbf{m}_k$ is an α-solution of the conjunction $c' = \neg(\mu_1\, c) \wedge \cdots \wedge \neg(\mu_{k-1}\, c) \wedge (\mu c)$.

For the induction step it suffices to show that $\mathbf{m}_k$ is the least α-solution of c'. Because then, with $\mu_k\, c = \mu_\ell\, c'$ it follows with the definition of the μ_ℓ-operator that $\mathbf{m}_k$ is the unique α-solution of $\mu_k\, c$.

By way of contradiction, assume there is a least α-solution $\mathbf{m}$ of c' with $\mathbf{m} \neq \mathbf{m}_k$. We consider two (exhaustive) cases.

In the first case $\mathbf{m}_k <_\ell \mathbf{m}$. This is a direct contradiction to the assumption that $\mathbf{m}$ is the least α-solution of c.

In the second case $\mathbf{m} <_\ell \mathbf{m}_k$. Because $\mathbf{m}$ is a (least) α-solution of c', $\mathbf{m}$ is in particular an α-solution of its conjunct $(\mu\, c)$. In other words, $\mathbf{m}$ is a minimal α-solution of c'. Further, recall that $\mathbf{m}_1, \ldots, \mathbf{m}_k, \ldots, \mathbf{m}_n$ are all minimal α-solutions and that they are lexicographically ordered. Altogether, with $\mathbf{m} \dot{<} \mathbf{m}_k$ it follows that $\mathbf{m} = \mathbf{m}_j$, for some $1 \leq j < k$. By the induction hypothesis, $\mathbf{m}_j$ is an α-solution of $\mu_j\, c$. On the other hand, with $\mathbf{m}$ and thus also $\mathbf{m}_j$ being a solution of $\mu_k\, c$, $\mathbf{m}_j$ is also an α-solution of $\mu_k\, c$'s conjunct $\neg(\mu_j\, c)$. A plain contradiction.

From the contradictions in both cases conclude $\mathbf{m} = \mathbf{m}_k$, which remained to be shown for the first part.

[16] $\leq_\ell$ denotes the lexicographic ordering on integer tuples, and $<_\ell$ its strict subset.

It remains to show that $\mu_k\, c$ is not α-satisfiable, for all $k > n$. This, however, is clear with the first part, because any such α-solution $\mathbf{m}$ would provide a minimal α-solution for c that is different to each of $\mathbf{m}_1, \ldots, \mathbf{m}_n$. However $\mathbf{m}_1, \ldots, \mathbf{m}_n$ was assumed to consist of *all* minimal α-solutions. □

A.2 Section 4

Lemma 4.5 (Consistent α-Productivity) *Let $\Lambda \cdot \Gamma$ be an admissible context and $\alpha \in Mods(\Gamma)$. For any context literal $L(\mathbf{x}) \mid c$, Λ cannot α-produce both $L(\mathbf{x}) \mid c$ and its complement $\overline{L}(\mathbf{x}) \mid c$.*

Proof. Suppose, by contradiction, Λ α-produces both $L(\mathbf{x}) \mid c$ and its complement $\overline{L}(\mathbf{x}) \mid c$, for some context literal $\overline{L}(\mathbf{x}) \mid c$. Then there need to be two context literals $K = L(\mathbf{x}) \mid c_1$ and $K' = \overline{L}(\mathbf{x}) \mid c_2$ in Λ that α-produce $L(\mathbf{x}) \mid c$ and $\overline{L}(\mathbf{x}) \mid c$ wrt. Λ, respectively. By definition of α-productivity, K and K' α-cover these literals and so c_1 and c_2 must be α-satisfiable. Now, with Lemma 3.2 three cases apply: if $\alpha \models_{\mathcal{Z}} c_1 \doteq_{\mu_\ell} c_2$ then $\Lambda \cdot \Gamma$ would be contradictory, which is impossible by admissibility; if $\alpha \models_{\mathcal{Z}} c_1 \dot{<}_{\mu_\ell} c_2$ then K cannot α-produce $L(\mathbf{x}) \mid c$, and if otherwise $\alpha \models_{\mathcal{Z}} c_2 \dot{<}_{\mu_\ell} c_1$ then K' cannot α-produce $\overline{L}(\mathbf{x}) \mid c$. Both cases contradict the assumption made. □

Lemma 4.9 (Lifting) *Let $\Lambda \cdot \Gamma$ be an admissible context, $\alpha \in Mods(\Gamma)$, $D[\mathbf{x}] = L_1(\mathbf{x}_1) \vee \cdots \vee L_k(\mathbf{x}_k) \leftarrow c[\mathbf{x}]$ with $k \geq 1$ a constrained clause, and $\mathbf{m}$ a vector of constants from $\mathbb{Z}$. If $\mathcal{Z}_{\Lambda,\alpha}$ falsifies $D[\mathbf{m}]$, then there is an α-productive context unifier d of D against $\Lambda \cdot \Gamma$ where $\mathbf{m}$ is an α-solution of d.*

Proof. Suppose that $\mathcal{Z}_{\Lambda,\alpha}$ falsifies $D[\mathbf{m}]$. This means:

1. $\mathcal{Z}_{\Lambda,\alpha}$ satisfies $c[\mathbf{m}]$, and, as $\mathcal{Z}_{\Lambda,\alpha}$ agrees with α on the parameters, $\alpha \models_{\mathcal{Z}} c[\mathbf{m}]$.
2. $\mathcal{Z}_{\Lambda,\alpha}$ satisfies $\overline{L_i}(\mathbf{m}_i)$, for all $i = 1, \ldots, k$, where $\mathbf{m}_i$ is the projection of $\mathbf{m}$ over the variables $\mathbf{x}_i$.

From (2) and with Lemma 4.7, it follows that Λ α-produces $\overline{L_i}(\mathbf{m}_i)$. That is, there are context literals $\overline{L_i}(\mathbf{x}_i) \mid c_i$ in Λ that α-produce $\overline{L_i}(\mathbf{m}_i)$, or, better said, the normalized versions $\overline{L_i}(\mathbf{x}_i) \mid \mathbf{x}_i \doteq \mathbf{m}_i$. With Definition 4.2 it follows $\alpha \models_{\mathcal{Z}} \bar{\forall}\, (\mathbf{x}_i \doteq \mathbf{m}_i \rightarrow c_i)$. Equivalently, $\alpha \models_{\mathcal{Z}} c_i[\mathbf{m}_i]$. Together with (1) above it follows $\alpha \models_{\mathcal{Z}} d'[\mathbf{m}]$, where

$$d'[\mathbf{x}] \stackrel{\text{def}}{=} c[\mathbf{x}] \wedge c_1[\mathbf{x}_1] \wedge \cdots \wedge c_k[\mathbf{x}_k] \ .$$

Because $\mathbf{m}$ is an α-solution of d' there is also a minimal α-solution $\mathbf{m}'$ of d' that is less or equal (in the component-wise ordering) than $\mathbf{m}$. With Lemma 3.1 this minimal

solution can be characterized by a constraint. More precisely, there is a $k \geq 1$ such that $\mathbf{m}'$ is the only α-solution of $\mu_k\, d'$. But then, it follows that $\mathbf{m}$ is an α-solution of

$$d[\mathbf{x}] = d'[\mathbf{x}] \wedge \exists \mathbf{y}\, (\mu_k\, d'[\mathbf{y}]) \wedge \mathbf{y} \dot{\leq} \mathbf{x}\ .$$

In other words, d is a context unifier of D against $\Lambda \cdot \Gamma$ where $\mathbf{m}$ is an α-solution of d.

It remains to show that d is α-productive and that $\mathbf{m}$ is an α-solution of d. Let, for all $i = 1, \ldots, k$,

$$\begin{aligned} K_i &= \overline{L_i}(\mathbf{x}_i) \mid c_i, \text{ and} \\ K_i' &= L_i(\mathbf{x}_i) \mid d_i, \text{ where } d_i = \pi\, \mathbf{x}_i\, d. \end{aligned}$$

We have to show that K_i α-produces $\overline{K_i'} = \overline{L_i}(\mathbf{x}_i) \mid d_i$ wrt. Λ.

First we have to show that K_i α-covers $\overline{K_i'}$, i.e., (i) $\alpha \models_{\mathcal{Z}} \bar{\exists} d_i$, and (ii) $\alpha \models_{\mathcal{Z}} d_i \rightarrow c_i$. Regarding (i), $\alpha \models_{\mathcal{Z}} d[\mathbf{m}]$ from above entails that d is α-satisfiable. Its projection over the variables $\mathbf{x}_i$ is an α-solution of d_i, by definition. The claim (ii) follows immediately from the fact that d_i is stronger than c_i (c_i is an element in the conjunction d).

It remains to show that there is no $L_i(\mathbf{x}_i) \mid e_i$ in Λ that α-covers $L_i(\mathbf{x}_i) \mid d_i$ and such that $\alpha \models_{\mathcal{Z}} c_i \dot{<}_{\mu_\ell} e_i$. It is not difficult to see that Λ cannot contain such a literal $L_i(\mathbf{x}_i) \mid e_i$, because, if it did, $\overline{L_i}(\mathbf{x}_i) \mid c_i$ would not α-produce $\overline{L_i}(\mathbf{m}_i)$ wrt. Λ as concluded above. □

A.3 Section 6.1

To prove soundness we have to show that whenever there is a refutation of Φ and Γ then $\Gamma \cup \Phi$ is LIA-unsatisfiable.

To prove the result, we need to take the context literals Λ into account as they evolve in the refutation. To this end, for a given parameter evaluation α we define a set $\Lambda^{\mu_\ell(\alpha)}$ of parameter-free, ground unit clauses as follows.

$$\begin{aligned} (L(\mathbf{x}) \mid c)^{\mu_\ell(\alpha)} &\stackrel{\text{def}}{=} \begin{cases} L(\mathbf{m}) & \text{if } \alpha \models_{\mathcal{Z}} \mu_\ell\, c[\mathbf{m}/\mathbf{x}] \\ \top & \text{otherwise} \end{cases} \\ \Lambda^{\mu_\ell(\alpha)} &\stackrel{\text{def}}{=} \{K^{\mu_\ell(\alpha)} \leftarrow \top \mid K \in \Lambda\} \end{aligned}$$

As noted earlier, if a constraint $c[\mathbf{x}]$ is α-satisfiable then its least solution $\mathbf{m}$, a vector of integer constants, is uniquely defined. Then, and only then it holds $\alpha \models_{\mathcal{Z}} \mu_\ell\, c[\mathbf{m}]$. This guarantees that $(L(\mathbf{x}) \mid c)^{\mu_\ell(\alpha)}$ is well-defined. In words then, the clause set $\Lambda^{\mu_\ell(\alpha)}$ ist obtained from the context literals by instantiating these with their least solution for a given parameter valuation α, if existent.

Let α be a parameter valuation. We say that a sequent $\Lambda \cdot \Gamma \vdash \Phi$ is *μ_ℓ-satisfiable* iff for some parameter valuation α, $\Lambda^{\mu_\ell(\alpha)} \cup \Gamma \cup \Phi$ is LIA-satisfiable in some Σ-interpretation that agrees with α on the parameters.

Recall that derivations always start with an initial sequent $\Lambda \cdot \Gamma \vdash \Phi$ for Φ and Γ, where Λ contains (only) the constraint literal $\neg P(\mathbf{x}) \mid -\mathbf{1} \dot{\leq} \mathbf{x}$ for each predicate symbol P in Σ. It follows that the initial sequent $\Lambda \cdot \Gamma \vdash \Phi$ is μ_ℓ-satisfiable iff $\Gamma \cup \Phi$ is LIA-satisfiable. This is immediate, as by admissibility the minimal solution of each constrained literal in Λ is of the form $\neg P(-\mathbf{1})$, while each variable of a clause in Φ is restricted to solutions ≥ 0.

To prove soundness it therefore suffices to show that whenever there is a refutation of Φ and Γ, then the initial sequent $\Lambda \cdot \Gamma \vdash \Phi$ is μ_ℓ-unsatisfiable (not μ_ℓ-satisfiable).

Lemma A.4 *For each rule of the $\mathcal{ME}$(LIA) calculus, if the premise of the rule is μ_ℓ-satisfiable, then one of its conclusions is μ_ℓ-satisfiable as well.*

Proof. We carry out a case analysis wrt. the derivation rule applied.
Close) The premise of $\mathsf{Close}(d)$ has the form $\Lambda \cdot \Gamma \vdash \Phi, D$, while its conclusion has the form $\Lambda \cdot \Gamma \vdash \Phi, D, \Box \leftarrow \top$, where d is a closing context unifier of D against $\Lambda \cdot \Gamma$. As $\Lambda \cdot \Gamma \vdash \Phi, D, \Box \leftarrow \top$ is μ_ℓ-unsatisfiable, we must show that $\Lambda \cdot \Gamma \vdash \Phi, D$ is μ_ℓ-unsatisfiable as well. Equivalently, we must show that for all parameter valuations α, $\Lambda^{\mu_\ell(\alpha)} \cup \Gamma \cup \Phi \cup \{D\}$ is LIA-unsatisfiable in all Σ-interpretations that agree with α on the parameters.

Let α be any parameter valuation. If $\alpha \notin \mathit{Mods}(\Gamma)$ then there is no Σ-interpretation that agrees with α on the parameters and that satisfies Γ. In this case the claim follws trivially. Hence assume from now on $\alpha \in \mathit{Mods}(\Gamma)$.

As $d[\mathbf{x}] = d'[\mathbf{x}] \wedge \exists \mathbf{y}\, (\mu_j\, d'[\mathbf{y}]) \wedge \mathbf{y} \dot{\leq} \mathbf{x}$, for some $j \geq 1$, where $d' = c \wedge c_1 \wedge \cdots \wedge c_k$ is a closing context unifier of $D[\mathbf{x}] = L_1(\mathbf{x}_1) \vee \cdots \vee L_k(\mathbf{x}_k) \leftarrow c[\mathbf{x}]$ against $\Lambda \cdot \Gamma$, there are context literals $\overline{L_i}(\mathbf{x}_i)|c_i$ for $1 \leq i \leq k$, with $\Gamma \models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$. With $\alpha \in \mathit{Mods}(\Gamma)$, thus, $\alpha \models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$.

With the definition of $\doteq_{\mu_\ell}$ it follows that d_i is α-satisfiable, and thus so is d. Because d is stronger than d' and d' is stronger than c_i, c_i is α-satisfiable, too. It follows $(\overline{L_i}(\mathbf{x}_i) \mid c_i)^{\mu_\ell(\alpha)} = \overline{L_i}(\mathbf{m}_i)$, where $\mathbf{m}_i$ is the least α-solution of c_i. That is, $\Lambda^{\mu_\ell(\alpha)}$ includes the clauses $\overline{L_i}(\mathbf{m}_i) \leftarrow \top$, for all $i = 1, \ldots, k$.

Consider the definition of d_i,

$$d_i = \pi\, \mathbf{x}_i\, (d'[\mathbf{x}] \wedge \exists \mathbf{y}\, (\mu_j\, d'[\mathbf{y}]) \wedge \mathbf{y} \dot{\leq} \mathbf{x}) \ .$$

Let $\mathbf{n}$ be the least α-solution of d. The constraint $\mu_\ell\, d_i$ then has exactly one α-solution which projects out $\mathbf{n}_i$ from $\mathbf{n}$. With $\alpha \models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$ it follows in current terminology that $\alpha \models_{\mathcal{Z}} \mathbf{m}_i \doteq \mathbf{n}_i$.

Consider the clause instance $D[\mathbf{n}] = L_1(\mathbf{n}_1) \vee \cdots \vee L_k(\mathbf{n}_k) \leftarrow c[\mathbf{n}]$. Because $\mathbf{n}$ is an α-solution of d (the least one), and d is stronger than c, $\mathbf{n}$ is an α-solution of c, too. But then, any Σ-model $\mathcal{Z}$ of $\Lambda^{\mu_\ell(\alpha)} \cup \Gamma \cup \Phi \cup \{D\}$ (and in particular of D) that agrees with α on the parameters must assign true to $L_1(\mathbf{n}_1) \vee \cdots \vee L_k(\mathbf{n}_k)$. However, with $\alpha \models_{\mathcal{Z}} \mathbf{m}_i \doteq \mathbf{n}_i$ and the fact that $\Lambda^{\mu_\ell(\alpha)}$ includes the clauses $\overline{L_i}(\mathbf{m}_i) \leftarrow \top$, for all $i = 1, \ldots, k$ such a model $\mathcal{Z}$ cannot exist, which remained to be shown.

Split) The premise of $\mathsf{Split}(d)$ has the form $\Lambda \cdot \Gamma \vdash \Phi$, while its conclusions have respectively the form $(\Lambda, L_i \mid d_i) \cdot \Gamma \vdash \Phi$ and $(\Lambda, \overline{L_i} \mid d_i) \cdot \Gamma \vdash \Phi$. Suppose that $\Lambda \cdot \Gamma \vdash \Phi, D$ is μ_ℓ-satisfiable. Then there must be a parameter valuation α such that $\Lambda^{\mu_\ell(\alpha)} \cup \Gamma \cup \Phi \cup \{D\}$ is LIA-satisfiable in some Σ-interpretation $\mathcal{Z}$ that agrees with α on the parameters.

Now, if d_i is not α-satisfiable then $(L_i \mid d_i)^{\mu_\ell(\alpha)} = \top = (\overline{L_i} \mid d_i)^{\mu_\ell(\alpha)}$ and it is easy to see that μ_ℓ-satisfiability is preserved even for both conclusions.

If on the other hand d_i is α-satisfiable then $(L_i \mid d_i)^{\mu_\ell(\alpha)} = L_i(\mathbf{m}_i)$ and $(\overline{L_i} \mid d_i)^{\mu_\ell(\alpha)} = \overline{L_i}(\mathbf{m}_i)$, for some integer vector $\mathbf{m}_i$. Thus $\mathcal{Z}$ must satisfy either $\Lambda^{\mu_\ell(\alpha)} \cup \{L_i(\mathbf{m}_i) \leftarrow \top\} \cup \Gamma \cup \Phi \cup \{D\}$ or $\Lambda^{\mu_\ell(\alpha)} \cup \{\overline{L_i}(\mathbf{m}_i) \leftarrow \top\} \cup \Gamma \cup \Phi \cup \{D\}$.

In conclusion in any case one of the consequences is μ_ℓ-satisfiable.

Extend) The premise of $\mathsf{Extend}(d)$ has the form $\Lambda \cdot \Gamma \vdash \Phi, D$, while its conclusion has the form $(\Lambda, L_i \mid d_i) \cdot \Gamma \vdash \Phi, D$, where $\overline{L_i} \mid d_i$ is Γ-contradictory with Λ.

Suppose that $\Lambda \cdot \Gamma \vdash \Phi, D$ is μ_ℓ-satisfiable. Then there must be a parameter valuation α such that $\Lambda^{\mu_\ell(\alpha)} \cup \Gamma \cup \Phi \cup \{D\}$ is LIA-satisfiable in some Σ-interpretation $\mathcal{Z}$ that agrees with α on the parameters. It follows that $\alpha \in \mathit{Mods}(\Gamma)$ (because otherwise $\mathcal{Z}$ cannot satisfy Γ).

As $\overline{L_i} \mid d_i$ is Γ-contradictory with Λ, with the same argumentation as for $\mathsf{Close}(d)$ above it follows that (in particular) $\mathcal{Z}$ cannot LIA-satisfy $\Lambda^{\mu_\ell(\alpha)} \cup \{\overline{L_i}(\mathbf{m}_i) \leftarrow \top\} \cup \Gamma \cup \Phi \cup \{D\}$. Hence $\mathcal{Z}$ must LIA-satisfy $\Lambda^{\mu_\ell(\alpha)} \cup \{L_i(\mathbf{m}_i) \leftarrow \top\} \cup \Gamma \cup \Phi \cup \{D\}$. Equivalently, $(\Lambda, L_i \mid d_i) \cdot \Gamma \vdash \Phi, D$ is μ_ℓ-satisfiable, which was to be shown.

Domain Split) The premise of Domain Split has the form $\Lambda \cdot \Gamma \vdash \Phi, D$, while its conclusions have respectively the form $\Lambda \cdot (\Gamma, c \doteq_{\mu_\ell} d_i) \vdash \Phi, D$ and $\Lambda \cdot (\Gamma, \neg(c \doteq_{\mu_\ell} d_i)) \vdash \Phi, D$, where $\alpha \models_{\mathcal{Z}} c \doteq_{\mu_\ell} d_i$ for some $\alpha \in \mathit{Mods}(\Gamma)$.

Suppose that $\Lambda \cdot \Gamma \vdash \Phi, D$ is μ_ℓ-satisfiable. Then there must be a parameter valuation α such that $\Lambda^{\mu_\ell(\alpha)} \cup \Gamma \cup \Phi \cup \{D\}$ is LIA-satisfiable in some Σ-interpretation $\mathcal{Z}$ that agrees with α on the parameters.

As $\mathcal{Z}$ agrees with α on the parameters, if $\alpha \models_{\mathcal{Z}} c \doteq_{\mu_\ell} d_i$ then $\mathcal{Z}$ satisfies $c \doteq_{\mu_\ell} d_i$. And if $\alpha \not\models_{\mathcal{Z}} c \doteq_{\mu_\ell} d_i$ then $\alpha \models_{\mathcal{Z}} \neg(c \doteq_{\mu_\ell} d_i)$ and so $\mathcal{Z}$ satisfies $\neg(c \doteq_{\mu_\ell} d_i)$. Corresponding to the case that applies, it follows that one of the consequences is μ_ℓ-satisfiable.

Ground Split) The proof for this case is very similar to the case of Domain Split and is omitted. □

Proposition 6.1 (Soundness) *For all admissible clause sets Φ and admissible sets of closed constraints Γ, if there is a refutation tree of Φ and Γ, then $\Gamma \cup \Phi$ is LIA-unsatisfiable.*

Proof. Let $\mathbf{T}$ be a refutation tree of Φ and Γ. As observed in the beginning of Section 6.1, to prove that $\Gamma \cup \Phi$ is LIA-unsatisfiable it suffices to prove that the initial sequent $\Lambda \cdot \Gamma \vdash \Phi$ is μ_ℓ-unsatisfiable, i.e. the sequent in the root of $\mathbf{T}$.

If $\mathbf{T}$ has only one node N, the root, then the clause set Φ of the initial sequent must contain $\square \leftarrow \top$, and hence $\Gamma \cup \Phi$ is trivially LIA-unsatisfiable.[17]

If $\mathbf{T}$ has more than one node N, we can assume by induction that all the children nodes of N are μ_ℓ-unsatisfiable. But then we can conclude that N is μ_ℓ-unsatisfiable as well by the contrapositive of Lemma A.4. □

A.4 Section 6.3

We assume the same setup as in Section 6.3. That is, Φ is an admissible clause set, Γ a finite set of admissible closed constraints, $\mathcal{D}$ is a fair derivation of Φ and Γ that is not a refutation, $\mathbf{B} = (N_i)_{i<\kappa}$ is an exhausted branch in its limit tree, and $\Lambda_i \cdot \Gamma_i \vdash \Phi_i$ means the sequent labeling the node N_i in $\mathbf{B}$, for all $i < \kappa$.

Lemma 6.6 *For all $i < \kappa$, Γ_i is satisfiable, and there is an $i < \kappa$ such that for all $j \geq i$, $\Gamma_i = \Gamma_j = \Gamma_{\mathbf{B}}$.*

Proof. First we show that Γ_i is satisfiable, for all $i < \kappa$. This however follows easily from the facts that, by definition, Γ_0 ($= \Gamma$) is satisfiable, and that the only derivation rules that can manipulate the Γ_i's, Domain Split and Ground Split, preserve satisfiability (and admissibility) in both conclusions. This is guaranteed by the last condition in the definition of each rule.

Note that the constraint added by Domain Split or Ground Split to its context has the effect of strictly strengthening a finite range constraint for one or more parameters (recall that by admissibility every parameter is restricted to a finite range). Because the signature Π for parameters is finite, and satisfiability of the Γ_i's is preserved, these strengthenings can occur only finitely many times. Thus, for some $i < \kappa$ and all $j \geq i$, $\Gamma_i = \Gamma_j = \Gamma_{\mathbf{B}}$ and it follows that $\Gamma_{\mathbf{B}}$ is satisfiable. □

We need an additional lemma.

Lemma A.7 (Persistent Productivity) *Let K_1 be a context literal in $\Lambda_{\mathbf{B}}$, K_2 a context literal and $\alpha \in \mathit{Mods}(\Gamma_{\mathbf{B}})$. If K_1 α-produces K_2 wrt. $\Lambda_{\mathbf{B}}$ then there is an $i < \kappa$ such that for all $j \geq i$, K_1 is in Λ_j and K_1 α-produces K_2 wrt. Λ_j. Furthermore, $\alpha \in \mathit{Mods}(\Gamma_j)$.*

Proof. That K_1 is in $\Lambda_{\mathbf{B}}$ entails that K_1 is in Λ_i, for some $i < \kappa$. As derivations only grow contexts (but never shrink them), it follows that K_1 is in Λ_j, for all $j \geq i$. For the same reason, from $\alpha \in \mathit{Mods}(\Gamma_{\mathbf{B}})$ it follows $\alpha \in \mathit{Mods}(\Gamma_j)$.

Suppose, ad absurdum, that K_1 does not α-produce K_2 wrt. Λ_k, for some $k \geq j$. This means that there is a context literal $\overline{L}(\mathbf{x}) \mid d$ in Λ_k that α-covers $\overline{L}(\mathbf{x}) \mid c_2$ and such that $\alpha \models_{\mathcal{Z}} c_1 \dot{<}_{\mu_\ell} d$. That K is in Λ_k entails, by construction, that K is in $\Lambda_{\mathbf{B}}$, too. But then, K_1 does not α-produce K_2 wrt. $\Lambda_{\mathbf{B}}$ either, a plain contradiction to what was concluded above. □

[17] *Admissible* clause sets cannot contain $\square \leftarrow \top$, but the proof holds true even if they did.

As said, the following proposition is the main result for proving the calculus complete.

Proposition 6.7 (Model Construction) *If* $(\square \leftarrow \top) \notin \Phi_{\mathbf{B}}$ *then, for every* $\alpha \in \mathit{Mods}(\Gamma_{\mathbf{B}})$, $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ *is a model of* $\Phi_{\mathbf{B}}$.

Proof. Assume that $(\square \leftarrow \top) \notin \Phi_{\mathbf{B}}$. Chose any $\alpha \in \mathit{Mods}(\Gamma_{\mathbf{B}})$ arbitrarily.

Clearly, $\Lambda_{\mathbf{B}} \cdot \Gamma_{\mathbf{B}}$ is admissible and hence $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ is indeed defined. Suppose ad absurdum that $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ is not a model of $\Phi_{\mathbf{B}}$. This means that there is a constrained clause $D[\mathbf{x}] = L_1(\mathbf{x}_1) \vee \cdots \vee L_k(\mathbf{x}_k) \leftarrow c[\mathbf{x}]$ with $k \geq 1$ from $\Phi_{\mathbf{B}}$ that is falsified by $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ (it holds $k \geq 1$ because the clause set Φ is admissible). That is, there is a vector $\mathbf{m}$ of constants from $\mathbb{Z}$ such that $D[\mathbf{m}]$ is falsified by $\mathcal{Z}_{\Lambda,\alpha}$. By Lemma 4.9 then there is an α-productive context unifier d of D against $\Lambda_{\mathbf{B}} \cdot \Gamma_{\mathbf{B}}$ where $\mathbf{m}$ is an α-solution of d. For later use note that thus d is α-satisfiable.

The next step in the proof is to argue with Definition 6.4 (exhausted branch) and arrive at a contradiction in each case of an exhaustive case analysis. This case analysis has to be prepared a little, though.

That D is in $\Phi_{\mathbf{B}}$ entails that $D \in \Phi_i$, for all $i < \kappa$. From the fact that d is an α-productive context unifier of D against $\Lambda_{\mathbf{B}} \cdot \Gamma_{\mathbf{B}}$ it follows with Lemma A.7, by taking the maximum of the indices ι obtained from it, that there is an $\iota < \kappa$ such that for all $j \geq \iota$, d is an α-productive context unifier of D against $\Lambda_j \cdot \Gamma_j$. More specifically, in current notation, the context literals K_i, for $i = 1, \ldots, k$, all are in Λ_j and K_i α-produces $\overline{K_i'}$ wrt. Λ_j. Furthermore, with $\alpha \in \mathit{Mods}(\Gamma_j)$ and Lemma 4.5 it follows that Λ_j does not α-produce any literal K_i' of d.

Now fix any such $j \geq \iota$ big enough that it also satisfies $\Gamma_{j'} = \Gamma_{\mathbf{B}}$, for all $j' \geq j$. Such an index j must exist by Lemma 6.6.

Together, in terms of Definition 6.3 we have now shown that,

$$\text{for all } j' \geq j,\ d \text{ is not } \alpha\text{-redundant in } \Lambda_{j'} \cdot \Gamma_{j'}. \tag{5}$$

The conclusion (5) will lead to various contradictions below.

From Definition 6.4-(ii) it follows that $\mathsf{Close}(d)$ is not applicable to $\Lambda_j \cdot \Gamma_j \vdash \Phi_j$ with selected clause D. Thus there is a literal $L_i(\mathbf{x}_i) \mid d_i$ of d that is not Γ_j-contradictory with Λ_j. (Such a literal must exist because $k \geq 1$.) In particular, thus, $\Gamma_j \not\models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$, where c_i and d_i are as in Definition 4.8.

We analyze this situation wrt. the parameter valuation α from above, by a case analysis whether $\alpha \models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$ holds, or not.

1. $\alpha \models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$**.** Recall that j was chosen big enough so that (in particular) $\Gamma_j = \Gamma_{\mathbf{B}}$. From $\alpha \in \mathit{Mods}(\Gamma_{\mathbf{B}})$ it follows $\alpha \in \mathit{Mods}(\Gamma_j)$. Then it is easy to see that $\mathsf{Domain\ Split}(d)$ is applicable to $\Lambda_j \cdot \Gamma_j \vdash \Phi_j$ with selected clause D (by virtue of the context unifier literal $L_i(\mathbf{x}_i) \mid d_i$). By Definition 6.4-(i) then, d must be α-redundant in $\Lambda_{j'} \cdot \Gamma_{j'}$, for some $j' \geq j$. This is a direct contradiction to (5). Hence case (1) is impossible.

2. $\alpha \not\models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$**.** From the application of Lemma 4.9 above we know that d and hence d_i is α-satisfiable. In the terminology of Definition 4.8 then, the literal $L_i(\mathbf{x}_i) \mid d_i$ of d is a remainder literal. Various subcases apply.

2.1. $L_i(\mathbf{x}_i) \mid d_i$ **is contradictory with** $\Lambda_j \cdot \Gamma_j$**.** This means that there is a context literal $\overline{L_i}(\mathbf{x}_i) \mid c$ in Λ_j and a parameter evaluation $\alpha' \in \mathit{Mods}(\Gamma_j)$ such that $\alpha' \models_{\mathcal{Z}} c \doteq_{\mu_\ell} d_i$. Recall from above $\Gamma_j \not\models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$. But then, $\mathsf{Domain\ Split}(d)$ is applicable to $\Lambda_j \cdot \Gamma_j$ with context unifier literal $L_i(\mathbf{x}_i) \mid d_i$. Again by Definition 6.4-(i) then, d must be α-redundant in $\Lambda_{j'} \cdot \Gamma_{j'}$, for some $j' \geq j$. This is a direct contradiction to (5). Hence case (2.1) is impossible.

2.2. $L_i(\mathbf{x}_i) \mid d_i$ **is not contradictory with** $\Lambda_j \cdot \Gamma_j$**.**

2.2.1. $\overline{L_i}(\mathbf{x}_i) \mid d_i$ **is contradictory with** $\Lambda_j \cdot \Gamma_j$**.**

2.2.1.1. $\overline{L_i}(\mathbf{x}_i) \mid d_i$ **is** Γ_j**-contradictory with** Λ_j**.** Suppose that there is a $L_i(\mathbf{x}_i) \mid e_i$ in Λ_j that, for all $\alpha' \in \mathit{Mods}(\Gamma_j)$, α'-extends $L_i(\mathbf{x}_i) \mid d_i$. (This will lead to a contradiction.)

Thus, in particular, $L_i(\mathbf{x}_i) \mid e_i$ α-extends $L_i(\mathbf{x}_i) \mid d_i$. By Definition 4.1 then, $L_i(\mathbf{x}_i) \mid e_i$ α-covers $L_i(\mathbf{x}_i) \mid d_i$ and $\alpha \models_{\mathcal{Z}} e_i \doteq_{\mu_\ell} d_i$. The assumption for case (2) is $\alpha \not\models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$. From above we know that d_i is α-satisfiable. Because d_i is stronger than c_i, c_i is α-satisfiable, too. By Lemma 3.2 then, $\alpha \models_{\mathcal{Z}} c_i \dot{<}_{\mu_\ell} d_i$ or $\alpha \models_{\mathcal{Z}} d_i \dot{<}_{\mu_\ell} c_i$. However, d_i is stronger than c_i, which makes the latter case impossible. Thus, $\alpha \models_{\mathcal{Z}} c_i \dot{<}_{\mu_\ell} d_i$. With $\alpha \models_{\mathcal{Z}} e_i \doteq_{\mu_\ell} d_i$ it follows $\alpha \models_{\mathcal{Z}} c_i \dot{<}_{\mu_\ell} e_i$.

Alltogether then, by virtue of $L_i(\mathbf{x}_i) \mid e_i$ in Λ_j, this means that $\overline{L_i}(\mathbf{x}_i) \mid c_i$ cannot α-produce $\overline{L_i}(\mathbf{x}_i) \mid d_i$ wrt. Λ_j. This is a contradiction to the conclusion above that d is an α-productive context unifier of D against $\Lambda_j \cdot \Gamma_j$, for all $j \geq \iota$.

Thus, there is no $L_i(\mathbf{x}_i) \mid e_i$ in Λ_j that, for all $\alpha' \in \mathit{Mods}(\Gamma_j)$, α'-extends $L_i(\mathbf{x}_i) \mid d_i$. But then, $\mathsf{Extend}(d)$ is applicable to $\Lambda_j \cdot \Gamma_j$ with remainder literal $L_i(\mathbf{x}_i) \mid d_i$. Again by Definition 6.4-(i) then, d must be α-redundant in $\Lambda_{j'} \cdot \Gamma_{j'}$, for some $j' \geq j$, a direct contradiction to (5). Hence case (2.2.1.1) is impossible.

2.2.1.2. $\overline{L_i}(\mathbf{x}_i) \mid d_i$ **is not** Γ_j**-contradictory with** Λ_j**.** The assumption of case (2.2.1), that $\overline{L_i}(\mathbf{x}_i) \mid d_i$ is contradictory with $\Lambda_j \cdot \Gamma_j$, means that there is a context literal $L_i(\mathbf{x}_i) \mid c$ in Λ_j and a parameter evaluation $\alpha' \in \mathit{Mods}(\Gamma_j)$ such that $\alpha' \models_{\mathcal{Z}} c \doteq_{\mu_\ell} d_i$. That $\overline{L_i}(\mathbf{x}_i) \mid d_i$ is not Γ_j-contradictory with Λ_j entails in particular $\Gamma_j \not\models_{\mathcal{Z}} c_i \doteq_{\mu_\ell} d_i$. But then, $\mathsf{Domain\ Split}(d)$ is applicable to $\Lambda_j \cdot \Gamma_j$ with remainder literal $L_i(\mathbf{x}_i) \mid d_i$, and the same argumentation as in case (1) applies.

2.2.2. $\overline{L_i}(\mathbf{x}_i) \mid d_i$ **is not contradictory with** $\Lambda_j \cdot \Gamma_j$**.** Here, neither $L_i(\mathbf{x}_i) \mid d_i$ nor $\overline{L_i}(\mathbf{x}_i) \mid d_i$ is contradictory with $\Lambda_j \cdot \Gamma_j$. But then, $\mathsf{Split}(d)$ is applicable to $\Lambda_j \cdot \Gamma_j$ with remainder literal $L_i(\mathbf{x}_i) \mid d_i$. Once more by Definition 6.4-(i) then, d must be

α-redundant in $\Lambda_{j'} \cdot \Gamma_{j'}$, for some $j' \geq j$. This is a direct contradiction to (5). Hence case (2.2.2) is impossible.

In sum, each case has led to a contradiction now. Consequently, the assumption that $\mathcal{Z}_{\Lambda_{\mathbf{B}},\alpha}$ is not a model of $\Phi_{\mathbf{B}}$ is false, and the proof is complete. □