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

Peter Baumgartner[1], Alexander Fuchs[2], and Cesare Tinelli[2]

[1] National ICT Australia (NICTA), Peter.Baumgartner@nicta.com.au
[2] The University of Iowa, USA, {fuchs,tinelli}@cs.uiowa.edu

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.

We propose a novel refutation calculus for a suitable clause logic modulo Linear Integer Arithmetic (LIA) that overcomes these problems. To obtain a complete calculus, we disallow free function symbols of arity > 0 and restrict every free constant to range over a finite interval of $\mathbb{Z}$. For simplicity, we also restrict every (universal) variable to range over a bounded below interval of $\mathbb{Z}$ (such as, for instance, $\mathbb{N}$),

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 [4, 12] that this sort of applications can 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 not originally contain integer constraints. The following trivial example

from finite model reasoning demonstrates this.[3]

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

In effect, the finite interval declaration $a : [1..100]$ for the constant a together with the unit clause $P(a)$ enforces that any model must satisfy one of $P(1), \ldots, P(100)$. However, the third clause contradicts that. Finite model finders, e.g., need about 100 steps for the refutation, one for each case of the domain of a. Our $\mathcal{ME}(\mathrm{LIA})$ calculus, on the other hand, reasons directly with finite interval declarations and allows a refutation in $O(1)$ steps. See Section 2 for an in-depth discussion of another example.

The calculus we propose is derived from the *Model Evolution* calculus ($\mathcal{ME}$) [7], a first-order logic version of the propositional DPLL procedure. The new calculus, $\mathcal{ME}(\mathrm{LIA})$, shares with $\mathcal{ME}$ the concept of *evolving* interpretations in search for a model for the input clause set. The crucial insight that leads from $\mathcal{ME}$ to $\mathcal{ME}(\mathrm{LIA})$ lies in the use of the ordering $<$ on integers in $\mathcal{ME}(\mathrm{LIA})$ instead of the instantiation ordering on terms in $\mathcal{ME}$. This then allows $\mathcal{ME}(\mathrm{LIA})$ 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 criterion that is formulated, ultimately, as certain constraints over LIA expressions. All that requires (only) a decision procedure for the full fragment of LIA instead of a complete enumerator of LIA-unifiers.

For space constraints, we present only a basic version of the calculus We refer the reader to a longer version of this paper [6] for extensions and improvements.

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 [15]. 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 [2, 9]. In contrast, in $\mathcal{ME}(\mathrm{LIA})$ 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 instance, alone for the clause $\neg(0 < x) \vee P(x)$ there are infinitely many LIA-unifiers, $\{x \mapsto 1\}, \{x \mapsto 2\}, \ldots$, as solutions of the literal $\neg(0 < x)$, and each of them is (LIA-)most general. Thus, calculi based on complete sets of (most general) solutions of constraints will have to consider all of them. Theory resolution, for instance, will generate the infinitely many clauses $P(1), P(2), \ldots$ Calculi based on *satisfiability* alone, such as $\mathcal{ME}(\mathrm{LIA})$ or Bürckert's *constrained resolution* [8] avoid that by only checking satisfiability of constraints wrt. the background theory.

On the one hand, constrained resolution is more general than $\mathcal{ME}(\mathrm{LIA})$, as it admits background theories with (infinitely, essentially denumerable) many models, as opposed to the single fixed model that $\mathcal{ME}(\mathrm{LIA})$ works with.[4] On the other hand, constraint resolution does not admit free constant or function symbols.[5] The most severe drawback of constraint resolution, however, is the lack of redundancy criteria.

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

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

[5] Unless they are part of the background theory, which would be pointless, as typically background reasoners do not admit them.

The importance of powerful redundancy criteria has been emphasized in the development of the modern theory of resolution in the 1990s [14]. With slight variations they carry over to *hierarchical superposition* [1], a calculus that is related to constraint resolution. The recent calculus in [11] integrates dedicated inference rules for LIA into superposition. In [7, e.g.] we have described conceptual differences between $\mathcal{ME}$, further *instance based methods* [3] and other (resolution) calculi. Many of the differences carry over to the constraint-case, possibly after some modifications. For instance, $\mathcal{ME}(\mathrm{LIA})$ *explicitly*, like $\mathcal{ME}$, maintains a candidate model, which gives rise to a redundancy criteria 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 [13]. 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 $\mathcal{ME}(\mathrm{LIA})$ 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 currently working on an enhanced version with additional rules, targeting efficiency improvements. With these rules then $\mathcal{ME}(\mathrm{LIA})$ can indeed be seen as a proper lifting of DPLL(T) to the first-order level (within recursion-theoretic limitations).

2 Calculus Preview

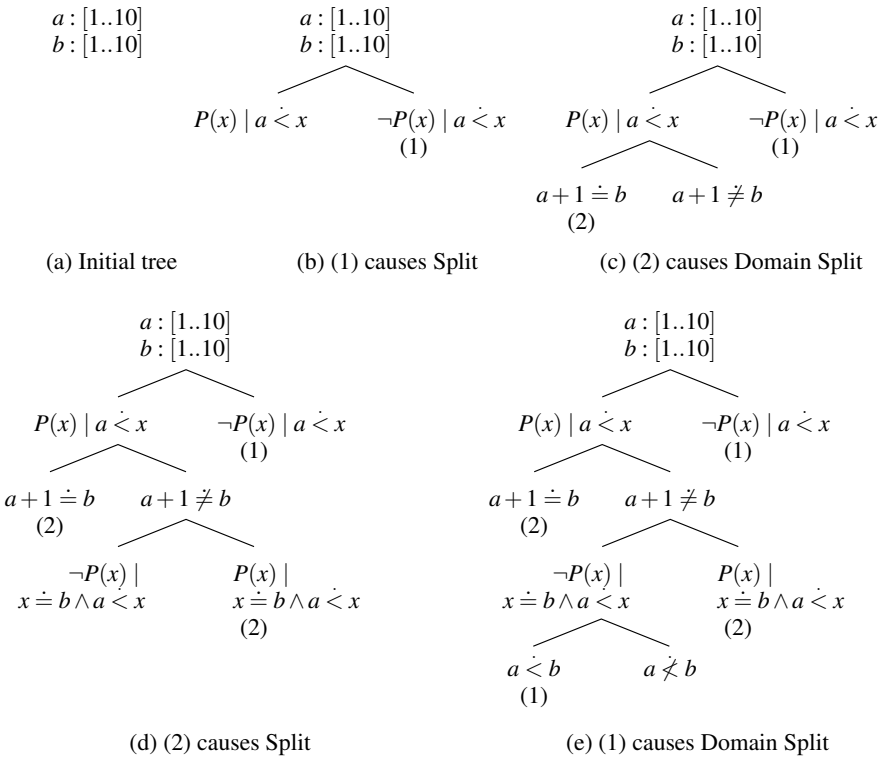
It is instructive to discuss the main ideas of the $\mathcal{ME}(\mathrm{LIA})$ calculus with a simple example before defining the calculus formally. Consider the following two unit *constrained clauses* (formally defined in Section 3):[6]

$$P(x) \leftarrow a \dot{<} x \qquad (1) \qquad\qquad \neg P(x) \leftarrow x \doteq b \qquad (2)$$

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

[6] 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

Fig. 1: Derivation example.

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 $P(n)$ for every integer $n > a$.

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 (partially) ordered and bounded below. 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 preserve this property. This minimal solution is commited to at the time the literal is added to the semantic tree and can always be denoted by a closed constraint over the parameters. 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 Figure 1b.[7] 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.

We skip the rest of the derivation, and just note that the trees in Figure 1d and Figure 1e are obtained by applying Split and Domain Split, respectively. 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 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 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.

[7] In DPLL terms, is akin to having performed a split on the complementary literals $P(a+1)$ and $\neg P(a+1)$. The calculus soundness proof relies in essence on this observation.

We write $t : [m..n]$ as an abbreviation of $m \dot{\leq} t \wedge t \dot{\leq} n$. We denote by $\bar{\exists}\, c$ the existential 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 $\mathbf{x}$. We use the predicate symbol $\dot{\leq}$ also to denote the component-wise extension of the integer ordering $\dot{\leq}$ to integer tuples (for any tuple size), $\dot{\leq}_\ell$ to denote the lexicographic extension of $\dot{\leq}$ to integer tuples, and $\dot{<}$ and $\dot{<}_\ell$ to denote their respective strict version.[8]

A constraint is *ground* if it contains no variables, *closed* if it contains no free variables.[9] We define a satisfaction relation $\models_{\mathcal{Z}}$ for closed parameter-free constraints as follows: $\models_{\mathcal{Z}} c$ if c is satisfiable 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 constraint c we write $\alpha \models_{\mathcal{Z}} c$ to denote that the *universal closure* of c is satisfiable in $\mathcal{Z}_\alpha$. A 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 $\mathbf{x}$.

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 $c[(x,y)] = (a \doteq x - y)$.

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*. Admissibility of a constraint $c[\mathbf{x}]$ can always be enforced by conjoining it 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 $m_1, \ldots, m_n$ 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 $k \geq 1$.[10]

$$\mu c \stackrel{\text{def}}{=} c \wedge (\forall \mathbf{y}\, d[\mathbf{y}] \rightarrow \neg(\mathbf{y} \dot{<} \mathbf{x})) \qquad \mu_\ell c \stackrel{\text{def}}{=} c \wedge \forall \mathbf{y}\, (c[\mathbf{y}] \rightarrow \mathbf{x} \dot{\leq}_\ell \mathbf{y})$$
$$\mu_k c \stackrel{\text{def}}{=} \mu_\ell\, (\neg(\mu_1 c) \wedge \cdots \wedge \neg(\mu_{k-1} c) \wedge (\mu c))$$

[8] We remark that each of the new symbols is definable in the given constraint language.

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

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

Recalling that c is admissible, it is easy to see that for any valuation α, μc has at most m α-solutions: the m minimal α-solutions of c, if any. If c is α-satisfiable, let $m_1, ..., m_n$ be an 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:

Lemma 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 $c[(x,y)] = a \dot{\leq} x \wedge a \dot{\leq} y \wedge \neg(x \doteq y)$ 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.

As we will see later, the calculus compares lexicographically minimal α-solutions of certain constraints. Since these constraints will have a *single* minimal solution, 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:

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

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

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

Lemma 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) $\alpha \models_{\mathcal{Z}} c \dot{<}_{\mu_\ell} d$, (ii) $\alpha \models_{\mathcal{Z}} c \doteq_{\mu_\ell} d$, or (iii) $\alpha \models_{\mathcal{Z}} d \dot{<}_{\mu_\ell} c$.*

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 in all of 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 $\square$.

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) there is a integer tuple $\mathbf{n}$ such that $\alpha \models_{\mathcal{Z}} c \rightarrow \mathbf{n} \dot{\leq} \mathbf{x}$ for all parameter valuations α. Condition (i) above is motivated by 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 $\neg \mathit{false} \leftarrow \top$.[11] Condition (ii) is the real restriction, forcing clauses to have admissible constraints, which is needed to guarantee the existence of least solutions, as explained above. To simplify our presentation, we will restrict consideration only to admissible clauses with a lower bound of $\mathbf{0}$, the tuple of all zeros, for each variable. For readability, in our examples we will always implicitly conjoin a clause constraint with $\mathbf{0} \dot{\leq} \mathbf{x}$ without explicitly writing the latter down.

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.

Definition 3 (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* $\alpha \models_{\mathcal{Z}} c_2 \rightarrow c_1$.

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 $\alpha \in \mathit{Mods}(\Gamma)$.

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

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

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 $\alpha \models_{\mathcal{Z}} \dot{\exists}\, c_2$ and $\alpha \models_{\mathcal{Z}} c_2 \rightarrow c_1$ 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 following notion of α-production is the one that allows us to associate a set of structures with each context.

Definition 4 (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.

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 5 (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 Γ-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 model candidates due to permanently falsified clauses in Section 2, which is captured precisely as *closing literals* in Definition 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 6 (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.*[12]

[12] 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 $\alpha \models_{\mathcal{Z}} c \doteq_{\mu_\ell} d$.

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 sense that they cannot produce both a constraint literal $L(\mathbf{x}) \mid c$ 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 7 (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 interprets each parameter a as $\alpha(a)$, and satisfies a positive ground literal $L(\mathbf{s})$ iff Λ α-produces $L(\mathbf{s})$.*

The above consistency property and the presence of literals $\neg P(\mathbf{x}) \mid -\mathbf{1} \dot{\leq} \mathbf{x}$ in admissible contexts entails that, for every $\alpha \in \mathit{Mods}(\Gamma)$, $\mathcal{Z}_{\Lambda,\alpha}$ satisfies a literal $L(\mathbf{n})$ if and only if Λ α-produces $L(\mathbf{n})$, where $\mathbf{n}$ is a tuple of non-negative integer constants. Thus, Definition 7 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 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*

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

with each c_i coming from a literal $\overline{K_i} = \overline{L_i}(\mathbf{x}_i) \mid c_i$ in Λ, and $j \geq 0$.

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

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

is a literal *of the context unifier. The literal K_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)*[13].

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 $\overline{K_i} = \overline{L_i}(\mathbf{x}_i) \mid c_i$ α-produces $\overline{K_i'} = \overline{L_i}(\mathbf{x}_i) \mid d_i$ 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 D's literals with context literals. The component $\mu_j d'[\mathbf{y}]$ denotes some minimal solution of d', the j-th one, which bounds from below the solutions of d. A simple, but import consequence (for completeness) is that for given

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

α and concrete solution $\mathbf{m}$ of d', j can always be chosen in such a way 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.

This is a formal statement of the just said.

Lemma 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 $I_{\Lambda,\alpha}$ 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.*

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 $c = \neg(x_1 \doteq x_2)$, $c_1 = 1 \dot{\leq} x_1$, and $c_2 = 1 \dot{\leq} x_2$. Then, the (unique) solution of $\mu_j d'$ for $j = 1$ is $(1,2)$ and 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 also the j-th 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 $\alpha \in Mods(\Gamma)$, 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 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 $j \geq 1$, $d = a \dot{<} x \wedge b \dot{<} x \wedge \exists y\,(\mu_i\,(a \dot{<} y \wedge b \dot{<} y)) \wedge y \dot{\leq} x$. 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 $j = 1$. It is closing if $\Gamma \models_{\mathcal{Z}} (a \dot{<} x) \doteq_{\mu_\ell} d_1$, which is equivalent to $\Gamma \models_{\mathcal{Z}} \neg(a \dot{<} b)$. That is not the case, i.e. there is an $\alpha \in Mods(\Gamma)$ that satisfies $a \dot{<} b$. According to Definition 8 then, K' is a remainder literal of d. Indeed, it can be verified then that $\alpha \models_{\mathcal{Z}} (a \dot{<} x) \dot{<}_{\mu_\ell} (a \dot{<} x \wedge b \dot{<} x)$.

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. Intuitively, an antecedent $\Lambda \cdot \Gamma$ corresponds to a branch in the semantic tree presentation in Section 2 and always denotes *a set* of candidate models for Φ, the Σ-structures induced by Λ and $\alpha \in Mods(\Gamma)$ (Def. 7) .

The calculus derives a tree of sequents with the goal of *evolving* the set of candidate models into a set of models of Φ. More precisely, a *derivation* of Γ and Φ starts with a tree with a root node only, which is labeled with the sequent $\Lambda_0 \cdot \Gamma \vdash \Phi$, where Λ_0 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 then applies the derivation rules defined below to grow that tree, by applying a rule at a time to a leaf of the tree and extending it with the conclusions in the expected way. See [6] for a formal definition.

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 [6], 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 $\mathbf{m}_i$.[14] 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 literals 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.

The $\mathcal{ME}(\mathrm{LIA})$ calculus has four derivation rules. The application of these rules is 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.)

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

This rule recognizes that the context is unfixable and adds the empty clause as a marker for that.

$$\mathsf{Split}(d)\quad \frac{\Lambda \cdot \Gamma \ \vdash \ \Phi, D}{(\Lambda, L_i \mid d_i) \cdot \Gamma \ \vdash \ \Phi, D \qquad (\Lambda, \overline{L_i} \mid d_i) \cdot \Gamma \ \vdash \ \Phi, D} \text{ if } \begin{cases} d \text{ is a context unifier of } D \text{ against } \Lambda \cdot \Gamma, \\ L_i \mid d_i \text{ is a remainder literal of } d, \text{ and} \\ \text{neither } L_i \mid d_i \text{ nor } \overline{L_i} \mid d_i \text{ is contradictory} \\ \qquad \text{with } \Lambda \cdot \Gamma. \end{cases}$$

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 $L_i \mid d_i$.

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

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

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 $\Lambda = \{P(x) \mid -1 \dot{\leq} x,\ P(x) \mid x \dot{<} 5\}$. Then, the clause, say, $P(x) \leftarrow x \dot{<} 7$ is falsified in the induced interpretation[15]. Adding $P(x) \mid x \dot{<} 7$ to Λ will fix that. However, Split is of no avail, as $\neg P(x) \mid x \dot{<} 7$ is contradictory with Λ for having the same least solution, 0, as $P(x) \mid x \dot{<} 5$. Extend will do instead.

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

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.

It is not too difficult to see that the 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.

In [6] we introduce an additional, but optional rule Ground Split that 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. Ground Split splits on Γ by adding to add a ground constraint l and $\bar{l}$, respectively, where 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.

5.1 Soundness and Completeness

Proposition 11 (Soundness). *For all admissible clause sets Φ and admissible sets of closed constraints Γ, if there is a derivation of Φ and Γ that ends in a tree containing $\square \leftarrow \top$ in each of its leaf nodes, 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

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

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.

To prove the calculus' completeness requires to introduce several technical notions. Again we refer to the long version of this paper [6] for that, and provide a brief summary here only. One of these notions 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 12 (Redundant Context Unifier). *Let $\Lambda_1 \cdot \Gamma_1$ and $\Lambda_2 \cdot \Gamma_2$ be admissible contexts, $\alpha \in Mods(\Gamma_1)$ 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. *$Mods(\Gamma_2) \subsetneq Mods(\Gamma_1)$*

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 $\alpha \in Mods(\Gamma)$.*

If condition (1) applies then the interpretation induced by Λ_2 and α will already satisfy D, and there is no point 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 that 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 to, and (iii) does not contain $\Box \leftarrow \top$. Finally, in *fair derivations* each leaf node of some derived tree contains $\Box \leftarrow \top$ or its limit tree has an exhausted branch.

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 above conditions (ii) and (iii) 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 $\mathcal{ME}$ calculus in [5].

The following is our main result (see [6] for a more precise statement and proof).

Theorem 13 (Completeness). *For every fair derivation of Φ and Γ, the (limit) context of every exhausted branch of its limit tree induces a LIA-model of $\Phi \cup \Gamma$.*

Note that this result 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 $\mathcal{ME}(\mathrm{LIA})$ calculus the same flexibility enjoyed by the DPLL calculus at the propositional level.

An interesting special case arises when the exhausted branch in Theorem 13 is finite. The branch then readily provides a model of the input clause set.

6 Conclusions and Further Work

We have presented a basic version of $\mathcal{ME}(\mathrm{LIA})$, 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 currently 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. But already the basic version presented here features a (semantically justified) redundancy criterion, essentially by reduction to LIA's ordering constraints, which allows to avoid inferences with clause instances that are already satisfied in the current candidate model.

References

1. L. Bachmair, H. Ganzinger, U. Waldmann. Refutational Theorem Proving for Hierachic First-Order Theories. *Appl. Algebra Eng. Commun. Comput*, 5:193–212, 1994.
2. P. Baumgartner. *Theory Reasoning in Connection Calculi*, LNAI 1527. Springer, 1998.
3. P. Baumgartner. Logical Engineering with Instance-Based Methods. In Frank Pfenning, ed., *Proc. CADE-21*, LNAI 4603, pp. 404–409. Springer, 2007.
4. P. Baumgartner, A. Fuchs, H. de Nivelle, C. Tinelli. Computing Finite Models by Reduction to Function-Free Clause Logic. *Journal of Applied Logic*, 2007. In Press.
5. P. Baumgartner, A. Fuchs, C. Tinelli. Implementing the Model Evolution Calculus. *International Journal of Artificial Intelligence Tools*, 15(1):21–52, 2006.
6. P. Baumgartner, A. Fuchs, C. Tinelli. $\mathcal{ME}(\mathrm{LIA})$ – Model Evolution With Linear Integer Arithmetic Constraints. `http://users.rsise.anu.edu.au/~baumgart/publications/MELIA.pdf`
7. P. Baumgartner, C. Tinelli. The Model Evolution Calculus. In Franz Baader, ed., *Proc. CADE-19*, LNAI 2741, pp. 350–364. Springer, 2003.
8. H.J. Bürckert. A Resolution Principle for Clauses with Constraints. In Mark E. Stickel, ed., *Proc. CADE-10*, LNAI 449, pp. 178–192. Springer, 1990.
9. H. Ganzinger, K. Korovin. Theory Instantiation. In *Proc. LPAR'06*, LNAI 4246, pp. 497–511. Springer, 2006.
10. Y. Ge, C. Barrett, C. Tinelli. Solving Quantified Verification Conditions Using Satisfiability Modulo Theories. In F. Pfenning, ed., *Proc. CADE-21*, LNCS 4603. Springer, 2007.
11. K. Korovin, A. Voronkov. Integrating Linear Arithmetic Into Superposition Calculus. In *Proc. CSL'07*, LNCS 4646, pp. 223–237. Springer, 2007.
12. J. Antonio N. Peréz. *Encoding and Solving Problems in Effectively Propositional Logic*. PhD thesis, The University of Manchester, 2007.
13. R. Nieuwenhuis, A. Oliveras, C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *J. of the ACM*, 53(6):937–977, 2006.
14. R. Nieuwenhuis, A. Rubio. Paramodulation-Based Theorem Proving. In J. A. Robinson and A. Voronkov, eds., *Handbook of Automated Reasoning*, pp. 371–443. Elsevier and MIT press, 2001.
15. M.E. Stickel. Automated Deduction by Theory Resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.