An Abstract Decision Procedure for a Theory of Bit-vectors

Clark Barrett
barrett@cs.nyu.edu
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University

Cesare Tinelli
cesare-tinelli@uiowa.edu
Department of Computer Science
University of Iowa

December 3, 2009

1 Introduction

[CT: to do]

2 Preliminaries

[CT: to do. Usual stuff]

3 The Algebra of Bit Vectors

We are interested in the satisfiability of conjunction of literals in the many-sorted algebra $\mathcal{BV}$ defined as follows.

The signature Σ of $\mathcal{BV}$ consists of sorts of the form $[n]$ for each $n \geq 0$, and function and predicate symbols from the table below (where :: separates the function symbol from its type).

constants	$\epsilon :: [0]$, $\mathbf{0} :: [1]$, $\mathbf{1} :: [1]$
concat	$\_ \circ \_ :: [m], [n] \to [m+n]$ for all $m, n \geq 0$
extract	$\_[i:j] :: [m] \to [i-j+1]$ for all $m > i, j \geq 0$ with $i - j \geq -1$
and	$\_ \,\&\, \_ :: [n], [n] \to [n]$ for all $n \geq 0$
or	$\_ \mid \_ :: [n], [n] \to [n]$ for all $n \geq 0$
exclusive or	$\_ \oplus \_ :: [n], [n] \to [n]$ for all $n \geq 0$
not	$\sim\_ :: [n] \to [n]$ for all $n \geq 0$
plus	$\_ + \_ :: [n], [n] \to [n]$ for all $n \geq 0$
times	$\_ \cdot \_ :: [n], [n] \to [n]$ for all $n \geq 0$
shift left	$\_ \ll \_ :: [n], [n] \to [n]$ for all $n \geq 0$
shift right	$\_ \gg \_ :: [n], [n] \to [n]$ for all $n \geq 0$
equal	$\_ \approx \_ :: [n], [n]$ for all $n \geq 0$
less	$\_ < \_ :: [n], [n]$ for all $n \geq 0$

Note that except for ϵ, $\mathbf{0}$ and $\mathbf{1}$ all the symbols above are overloaded. So, in the following, when we speak of "the symbol" $+$, say, we actually mean the infinite family $\{+ :: [n], [n] \to [n]\}_{n>0}$. Similarly for the other symbols.

Intuitively, $[n]$ is the sort of bit vectors of length n. The sort $[0]$ and its corresponding operators are not really needed in practice. We have them here for notational convenience with some of the derivation rules described later.

For each $n \geq 0$, we will use an infinite set X_n of variables of sort $[n]$. The set of well-sorted Σ-terms and quantifier-free formulas over $X = \{X_n\}_{n \geq 0}$ is defined as expected. Note that the extract operators i, j are indices on the name of the operator, not arguments. From this it follows, that each well-sorted term x has a unique sort $[n]$. We may write $x_{[n]}$ to specify that x has sort $[n]$. Also, we may write $+_n$ instead of $+$, say, to specify the $+$ with type $[n], [n] \to [n]$. (Similarly, for the other arithmetic and Boolean operators.)

To define $\mathcal{BV}$ formally, we start by defining a core algebra $\mathcal{BV}_\mathrm{c}$ over the signature $\Sigma_\mathrm{c} = \{\epsilon, \mathbf{0}, \mathbf{1}, \circ\} \cup \{[i{:}j]\}_{i,j}$. The algebra $\mathcal{BV}_\mathrm{c}$ is the *initial model* of the following set E_c of universal equations, for all m, n, k, i, j satisfying the given side constraints (in addition to the wellsortedness ones).[1]

$$E_{\mathrm{c}} = \begin{cases} (x_{[m]} \circ y_{[n]}) \circ z_{[k]} \approx x_{[m]} \circ (y_{[n]} \circ z_{[k]}) & \\ x_{[n]} \circ \epsilon \approx x_{[n]} & \\ \epsilon \circ x_{[n]} \approx x_{[n]} & \\ x_{[n]}[i:j] \approx \epsilon & \text{if } i-j=-1 \\ x_{[n]}[i:j] \circ x_{[n]}[j-1:k] \approx x_{[n]}[i:k] & \\ x_{[n]}[n-1:0] \approx x_{[n]} & \\ x_{[n]}[i:j][k:l] \approx x_{[n]}[k+j:l+j] & \\ (x_{[m]} \circ y_{[n]})[i:j] \approx x_{[m]}[i-n:j-n] & \text{if } j \geq n \\ (x_{[m]} \circ y_{[n]})[i:j] \approx y_{[n]}[i:j] & \text{if } i < n \\ (x_{[m]} \circ y_{[n]})[i:j] \approx x_{[m]}[i-n:0] \circ y_{[n]}[n-1:j] & \text{if } i \geq n > j \end{cases}$$

Using standard rewriting arguments it is possible to show [CT: to be double-checked] that $\mathcal{BV}_{\mathrm{c}}$ is constructed by the symbols $\epsilon, \mathbf{0}, \mathbf{1}$ and $\circ$; in other words, every element of $\mathcal{BV}_{\mathrm{c}}$ is denoted by a ground term over those symbols.

We will call a (possibly non-ground) term over the signature $\{\circ, \mathbf{0}, \mathbf{1}\}$ a *concat term*.

Since the operator $\circ$ satisfies the associative property in the first equation of E_{c}, we will treat it as multiarity symbol. Also, we will use non-empty strings over $\{\mathbf{0}, \mathbf{1}\}$ as an abbreviation of the concatenation of their elements (e.g., **001011** as a short-hand for $\mathbf{0} \circ \mathbf{0} \circ \mathbf{1} \circ \mathbf{0} \circ \mathbf{1} \circ \mathbf{1}$). We will treat such strings as if they were actual constants of Σ_{c}, defined in $\mathcal{BV}_{\mathrm{c}}$ in the obvious way. We'll call them and ϵ *(bit vector) values.* We will write $\mathbf{0}^n$ for the concatenation of $\mathbf{0}$ n times when $n > 0$ and for ϵ when $n \leq 0$. Similarly for $\mathbf{1}^n$.

We now expand $\mathcal{BV}_{\mathrm{c}}$ to the signature $\Sigma_{\mathrm{b}} = \Sigma_{\mathrm{c}} \cup \{\sim, \,\&\,, |, \oplus\}$ by interpreting the new operators respectively as Boolean negation, conjunction, disjunction and exclusive disjunction over the sort $[1]$ and as bitwise extensions of those operators over sort $[n]$ with $n > 0$. Over the sort $[0]$, which contains a single element, all operators are, trivially, constant functions. Let $\mathcal{BV}_{\mathrm{b}}$ be the resulting Σ_{b}-model.

It is possible to show that for each $n \geq 0$ the reduct of $\mathcal{BV}_{\mathrm{b}}$ to the sort $[n]$ and the symbols $\{\mathbf{0}^n, \mathbf{1}^n, \sim, \,\&\,, |, \oplus\}$ is isomorphic to the n-fold direct product of the standard model of the Booleans with itself. Also worth noting is that, for $n > 0$, the reduct of $\mathcal{BV}_{\mathrm{b}}$ to the sort $[n]$ and the symbols $\{\mathbf{0}^n, \mathbf{1}^n, \,\&\,, \oplus\}$ is a (one-sorted) Boolean ring, while the reduct of $\mathcal{BV}_{\mathrm{b}}$ to the sort $[n]$ and the symbols $\{\mathbf{0}^n, \mathbf{1}^n, \sim, \,\&\,, |\}$ is a Boolean algebra.

[1]Since we work with many-sorted first-order logic with equality here, the equality symbol $\approx$ is treated as a logical constant and always interpreted as the identity relation over $[n]$ for all $n \geq 0$.

We further expand $\mathcal{BV}_\mathrm{b}$ to the signature $\Sigma_\mathrm{b} \cup \{+, \cdot\}$ by interpreting the new operators respectively as base-2 integer addition and multiplication modulo 2^n, for each $n > 0$, and as constant functions for $n = 0$. Finally, we expand the resulting algebra to the signature $\Sigma_\mathrm{b} \cup \{+, \cdot\} \cup \{<, \ll, \gg\}$ by interpreting the new symbols as expected over $[n]$ for all $n \geq 0$. [CT: More details here?]

Let $\mathcal{BV}$ be the resulting Σ-model and let B_n be the set of all bit vector values of length n. It can be shown that, for $n > 0$, the reduct of $\mathcal{BV}$ to the sort $[n]$ and the symbols $B_n \cup \{+, \cdot\}$ is isomorphic to the residue class ring $\mathbb{Z}_{2^n}$. Also note that in the reduct of $\mathcal{BV}$ to the sort $[n]$ the relation denoted by $<$ is the lexicographic extension to bit vectors of size $[n]$ of the relation $\{(\mathbf{0}, \mathbf{1})\}$ over $[1]$ if $n > 0$; for $n = 0$, it is the empty relation.

It should be clear that every ground Σ-term t is equivalent in $\mathcal{BV}$ to one and only one bit vector value c. We say, that t *evaluates to* c.

In the following sections we define derivations systems for a number of fragments of the language of sets of Σ-literals. Each of these derivation systems are terminating and sound and complete with respect to satisfiability in $\mathcal{BV}$.

The various fragments are obtained simply by restricting the signature to some subset of Σ.

[Fix meta-notation: use x, y, x for variables, a, b, c, d for values, w, u, v for concat terms, m for monomials, p, q for polynomials, s, t for terms in general. :

]

4 Reasoning in the core algebra $\mathcal{BV}_\mathrm{c}$

We start with sets of Σ_c literals. Clearly, any such set is satisfiable in $\mathcal{BV}$ if and only if it is satisfiable in $\mathcal{BV}_\mathrm{c}$.

We first define a rewrite system $\longrightarrow_\mathrm{c}$ for Σ_c-terms which is terminating and normalizing modulo associativity of $\circ$. The rewrite system consists of the following rule schemas where x and y are meta-variables and the expressions involving i, j, n, k, l are meta-expressions standing for concrete numerals. The rewrite rules apply modulo associativity of $\circ$.

LeftUnit

$\epsilon \circ x \;\longrightarrow_\mathrm{c}\; x$

RightUnit

$x \circ \epsilon \;\longrightarrow_\mathrm{c}\; x$

ConcatMerge

$x[i : j] \circ x[j - 1 : k] \;\longrightarrow_\mathrm{c}\; x[i : k]$

Empty

$x_{[0]} \longrightarrow_{\mathrm{c}} \epsilon \quad \text{if} \;\; x \neq \epsilon$

ExtractWhole

$x_{[n]}[n-1:0] \longrightarrow_{\mathrm{c}} x_{[n]}$

ExtractExtract

$x[i:j][k:l] \longrightarrow_{\mathrm{c}} x[k+j:l+j]$

ExtractConcat1

$(x \circ y_{[n]})[i:j] \longrightarrow_{\mathrm{c}} x[i-n:j-n] \quad \text{if} \;\; j \geq n$

ExtractConcat2

$(x \circ y_{[n]})[i:j] \longrightarrow_{\mathrm{c}} y_{[n]}[i:j] \quad \text{if} \;\; i < n$

ExtractConcat3

$(x \circ y_{[n]})[i:j] \longrightarrow_{\mathrm{c}} x[i-n:0] \circ y_{[n]}[n-1:j] \quad \text{if} \;\; i \geq n > j$

Lemma 4.1. *The rewrite system $\longrightarrow_c$ is confluent and normalizing modulo associativity of $\circ$.*

Proof. [CT: to do] □

By virtue of the lemma above, every Σ_{c}-term t has a normal form modulo associativity of $\circ$. Let $t{\downarrow_{\mathrm{c}}}$ denote such a normal form. We write $t_1 =_A t_2$ to denote that two Σ_{c}-terms t_1 and t_2 are identical modulo associativity of $\circ$.

Proposition 4.2. *For all Σ_c-terms s and t,*

$$\mathcal{BV}_c \models s \approx t \quad \textit{iff} \quad s{\downarrow_c} =_A t{\downarrow_c} \tag{1}$$

Proof. [CT: to do] □

We now define a derivation system for finite sets E of normalized $\Sigma_{\rm c}$-equalities, that is, equalities between $\Sigma_{\rm c}$-terms in normal form. Note that in such sets, terms of sort $[0]$ can occur in E on in equalities of the form $\epsilon \approx \epsilon$.

The derivation system consists of rules that apply to pairs of the form $\sigma \vdash E$ where σ is a substitution from variables to (possibly non-ground) terms over the signature $\{\mathbf{0}, \mathbf{1}, \circ\}$. Each rule replaces a pair $\sigma \vdash E$ into a pair of the same form or into the symbol $\bot$ standing for the universally false formula.

In the rules below, $\sigma\{x \mapsto y\}$ denotes the composition of the substitution σ with the substitution $\{x \mapsto y\}$.[2] Also, $\sigma{\downarrow_{\rm c}} = \{x \mapsto (x\sigma){\downarrow_{\rm c}} \mid x \in \mathbf{dom}(\sigma)\}$ and $E{\downarrow_{\rm c}} = \{s{\downarrow_{\rm c}} \approx t{\downarrow_{\rm c}} \mid s \approx t \in E\}$.

The rules apply modulo associativity of $\circ$ and symmetry of $\approx$.

FailEq

$$\frac{\sigma \vdash E,\ c \approx d}{\bot} \quad \text{if } \ c, d \text{ distinct values}$$

SimplifyEq

$$\frac{\sigma \vdash E,\ x \approx y}{\sigma \vdash E} \quad \text{if } \ x =_A y$$

DecomposeEq

$$\frac{\sigma \vdash E,\ x_{[n]} \circ z_1 \approx y_{[n]} \circ z_2}{\sigma \vdash E,\ x \approx y,\ z_1 \approx z_2}$$

AlignEq

$$\frac{\sigma \vdash E,\ x_{[n+k]} \circ y \approx z_{[n]} \circ u}{\sigma \vdash E,\ x_1 \circ x_2 \circ y \approx z_{[n]} \circ u,\ x \approx x_1 \circ x_2} \quad \text{if } \begin{cases} k > 0 \\ x \text{ variable} \\ z \text{ variable or value of maximal length in } z \circ u \\ x_1 :: [n], x_2 :: [k] \text{ fresh variables} \end{cases}$$

EliminateExtract

$$\frac{\sigma \vdash E,\ l[y_{[n]}[i : j]]}{\sigma \vdash E,\ l[y_2],\ y \approx (y_1 \circ y_2 \circ y_3){\downarrow_{\rm c}}} \quad \text{if } \begin{cases} y \text{ variable} \\ y_1 :: [n-i], y_2 :: [i-j+1], y_3 :: [j] \text{ fresh variables} \end{cases}$$

Observe that in the rules above n is necessarily positive.

Solve

[2] We write substitution applications in postfix format: $x\sigma$. So $x(\sigma_1\sigma_2) = (x\sigma_1)\sigma_2$.

$$\frac{\sigma \vdash E,\ x \approx y}{(\sigma\{x \mapsto y\})\!\downarrow_{\mathrm{c}} \vdash (E\{x \mapsto y\})\!\downarrow_{\mathrm{c}}} \quad \text{if} \begin{cases} x \text{ variable} \\ y \text{ concat term} \end{cases}$$

[CT: Note that, with the given premises, x cannot occur in y otherwise the equality would be either ill-sorted or reducible.]

The rules are applied starting with an initial pair $\varepsilon \vdash E\!\downarrow_{\mathrm{c}}$ where ε is the empty substitution and E is a set of Σ_{c}-equalities whose satisfiability one is interested in.

In the following, let $\sigma_{\approx}$ denote the set $\{x \approx x\sigma \mid x \in \mathbf{dom}(\sigma)\}$.

Lemma 4.3. *If $\sigma' \vdash E'$ is the result of applying a derivation rule to $\sigma \vdash E$ and $\bar{y}$ are all the new variables in $\sigma' \vdash E'$, then $\exists \bar{y}\,(\sigma'_{\approx} \wedge E')$ is equivalent to $\sigma_{\approx} \wedge E$ in $\mathcal{BV}_c$.*

Proof. (Sketch) For most rules the claim is immediate. For **Solve** it is a consequence of Proposition 4.2. Similarly for **EliminateExtract**[3]. □

Proposition 4.4 (Termination)**.** *The derivation system terminates for any input of the form $\varepsilon \vdash E\!\downarrow_c$, and reduces it to one of the form $\bot$ or $\sigma \vdash \emptyset$.*

Proof. [CT: to do] □

Proposition 4.5 (Soundness)**.** *Let E be a set of Σ_c-equalities. If $\varepsilon \vdash E\!\downarrow_c$ reduces to $\bot$, then E is unsatisfiable in $\mathcal{BV}_c$.*

Proof. [CT: to do] □

Proposition 4.6 (Completeness)**.** *Let E be a set of Σ_c-equalities. If $\varepsilon \vdash E\!\downarrow_c$ reduces $\sigma \vdash \emptyset$, then E is satisfiable in $\mathcal{BV}_c$. Moreover, for any substitution θ mapping the variables of E to bit vector values, θ is a solution of E iff it is an instance of σ.*

Proof. [CT: to do] □

[3]The normalization there is needed for when y' or y'' has sort $[0]$. In practice, it might be better to have three rules depending on whether $i = n$ or not and $j = 0$ or not.

We now extend the previous solver to finite sets L of normalized Σ_1-literals by extending the applicability of **EliminateExtract** to disequalities and adding the remaining rules below.

For convenience, we assume that L always contains the disequality $\mathbf{0} \not\approx \mathbf{1}$. Let G_L be the undirected graph whose nodes are the terms in L and whose links connect two terms s and t iff $s \not\approx t \in L$. The *connected component in* L of a term t is the subgraph of G_L containing exactly all terms reachable from t in G_L.

A disequality in L is marked by a Boolean flag, which is initially false. We write $s \not\approx t$ for an disequality whose flag is false and $s \not\approx^{\surd} t$ for one whose flag is true.

FailDis

$$\frac{\sigma \vdash L,\ x \not\approx y}{\bot} \quad \text{if}\ \ x =_A y$$

ImplicitEq

$$\frac{\sigma \vdash L,\ x_{[1]} \not\approx z, y \not\approx z}{\sigma \vdash L,\ x_{[1]} \approx y, y \not\approx z}$$

DecomposeDis

$$\frac{\sigma \vdash L,\ x \not\approx y_{[n]}}{\sigma \vdash L,\ x \not\approx^{\surd} y,\ x_1 \not\approx y_1 \quad || \quad \sigma \vdash L,\ x \not\approx^{\surd} y,\ x_1 \approx y_1,\ x_2 \not\approx y_2} \quad \text{if}\ \ (*)$$

$$(*) = \begin{cases} \text{the connected component of } x \text{ in } L \cup \{x \not\approx y\} \text{ has size} > 2^n \\ x_1 = (x[n-1:i])\downarrow_{\mathrm{c}},\ y_1 = (y[n-1:i])\downarrow_{\mathrm{c}} \\ x_2 = (x[i-1:0])\downarrow_{\mathrm{c}},\ y_2 = (y[i-1:0])\downarrow_{\mathrm{c}} \\ n > i \geq 1 \end{cases}$$

Each connected component of G_L is used here as an over-approximation of the largest clique contained in that component. If a connected component of terms of sort $[n]$ contains more than 2^n elements, the corresponding set of disequalities might be unsatisfiable, which is the case exactly when it contains a clique of size $> 2^n$. For $n = 1$, that situation is recognized by an application of **ImplicitEq** and **FailDis**. For $n > 1$ (where recognizing a clique of size $> 2^n$ might be too expensive), **DecomposeDis** is used to reduce n.

DecomposeDis is the only branching rule of the derivation system. Note that the value of the index i in that rule is chosen in a *don't care* non-deterministic way.

An optimization is provided by the following optional rule which can be applied when a clique of size $2^n + 1$ in graph associated with the current set of disequalities can be computed cheaply.

FailDis2

$$\frac{\sigma \vdash L \cup \{x_i \not\approx x_j \mid 1 \leq i < j \leq 2^n + 1\}}{\bot} \quad \text{if } \ x_i :: [n] \text{ for all } i = 1, \ldots, 2^n + 1$$

Lemma 4.7. *If $\sigma' \vdash L'$ is the result of applying a derivation rule to $\sigma \vdash L$ and $\bar{y}$ are all the new variables in $\sigma' \vdash L'$, then $\exists \bar{y}\, (\sigma'_{\approx} \wedge L')$ is equivalent to $\sigma_{\approx} \wedge L$ in $\mathcal{BV}_c$.*

Proof. [CT: to do] □

We say that a pair $\sigma \vdash L$ is *(ir)reducible* with respect to all the derivation rules above if (n)one of the rules applies to it. A *derivation tree* for a set L_0 of Σ_c-literals is a finite tree with root $\varepsilon \vdash L_0 \downarrow_c$ such that each internal node of the tree is a reducible pair of the form $\sigma \vdash L$ and its children are the conclusions of a derivation rule that applies to $\sigma \vdash L$. A *refutation tree* (for L_0) is a derivation tree all of whose leaves are $\bot$. A *derivation* of L_0 is a sequence of derivation trees starting with the single-node tree containing $\varepsilon \vdash L_0 \downarrow_c$, where each tree is derived from the previous one by the application of a derivation rule to one of its leaves. A *refutation* of L_0 is a finite derivation of L_0 ending with a refutation tree.

Proposition 4.8 (Termination)**.** *The derivation system has no infinite derivations.*

Proof. [CT: to do. Each new rule reduces the number of unmarked disequalities of sort $[n]$ and introduces new disequalities only of smaller sorts.] □

Proposition 4.9 (Soundness)**.** *A set L_0 of Σ_c-literals is unsatisfiable in $\mathcal{BV}_c$ if it has a refutation.*

Proof. [CT: to do. Simple inductive argument based on Lemma 4.7.] □

Proposition 4.10 ((Strong) Completeness)**.** *If a set L_0 of Σ_c-literals is unsatisfiable in $\mathcal{BV}_c$, every derivation of L_0 extends to a refutation.*

Proof. [CT: to do. Take a derivation with an irreducible leaf $\sigma \vdash L$ and show that the σ is a solution of L. Then appeal to Lemma 4.7.] □

5 Reasoning in the algebra $\mathcal{BV}_\mathrm{a}$

We now consider the restriction of $\mathcal{BV}$ to the signature $\Sigma_\mathrm{a} = \Sigma_\mathrm{c} \cup \{+, \cdot\}$.

Let B_n be again the set of all bitvector values of sort $[n]$. Recall that for every $n > 0$, there is an isomorphism between the residue class ring $\mathbb{Z}_{2^n}$ and the reduct $\mathcal{BV}_n$ of $\mathcal{BV}$ to $[n]$ and the symbols $B_n \cup \{+, \cdot\}$. This isomorphism maps B_n to the set $\{0, \ldots, 2^n - 1\}$ as expected: every value in B_n is the encoding in base 2 of a value in $\{0, \ldots, 2^n - 1\}$. When convenient then, and when the sort $[n]$ is understood, we will not distinguish between $\mathcal{BV}_n$ and $\mathbb{Z}_{2^n}$ and will use natural numbers and expressions to denote their corresponding value in B_n. For instance, for a given $n > 2$, say, we may use 0 to denote the bit vector value $\mathbf{0}^n$, 5 to denote the value $\mathbf{0}^{n-3}\mathbf{101}$, 2^k for some $k < n$ to denote the value $\mathbf{0}^{n-k-1}\mathbf{1}\mathbf{0}^k$, and so on.

If c is a bit vector value of sort $[n]$, $-c$ denotes the additive inverse of c, i.e., the unique value c' such that $c + c'$ evaluates to 0. If c is also odd (equivalently, if $c[0:0]$ evaluates to $\mathbf{1}$), c^{-1} denotes the multiplicative inverse of c,[4] i.e., the unique value c' such that $c \cdot c'$ evaluates to 1. We write $x - c \cdot y$ as a shorthand for $x + -c \cdot y$.

We say that a term is a $\mathbf{0}$-*prefix term* [CT: For want of a better term] if it is a variable or has the form $\mathbf{0}^m \circ x_{[n]}$ where $m, n > 0$ and x is a variable.

A *monomial* is a term of the form $y_1 \cdot y_2 \cdot \cdots \cdot y_n$, modulo AC of $\cdot$, where $n \geq 1$ and each y_i is a $\mathbf{0}$-prefix term. We call each y_i an *indeterminate.* A $\cdot$-*term* [CT: come up with a better name] is a term of the form $c \cdot m$ where c is a constant and m is a monomial. In the following, we will not distinguish between monomials that are equal modulo AC of $\cdot$. Similarly for $\cdot$-terms.

A *polynomial* is a term of the form $c_1 \cdot m_1 + c_2 \cdot m_2 + \cdots + c_k \cdot m_k + c$, modulo AC of $+$, where $k \geq 0$, each c_i is a non-zero value, c is value, and $m_1, \ldots, m_k$ are pairwise distinct monomials of $\cdot$. The polynomial is *constant* if $k = 0$; it is *linear* if each of its monomials consists of one indeterminate.

Remark 5.1. The point of using $\mathbf{0}$-prefix terms as indeterminates in polynomials is to normalize concat terms as much as possible so that certain implicit shared constraints become explicit. For example, normally, if x is a bit vector variable for sort [3], say, in the (linear) equality

$$1 \cdot (x \circ \mathbf{0}) + 3 \cdot (\mathbf{0} \circ x) \approx \mathbf{0}^4$$

the terms $x \circ \mathbf{0}$ and $\mathbf{0} \circ x$ would be two distinct indeterminates of the left-hand side polynomial. This polynomial has only one solution for those indeterminates: $(\mathbf{0}^4, \mathbf{0}^4)$. But discovering that requires a number of arithmetic and core inferences. Putting the equation in its prefix form, on the other hand, yields a polynomial in one indeterminate:

$$2 \cdot (\mathbf{0} \circ x) + 3 \cdot (\mathbf{0} \circ x) \approx \mathbf{0}^4$$

making the equation easily solvable with basic arithmetic rules. $\square$

[4]Such inverse always exits because if c is odd then it is coprime with 2^n.

5.1 Approach A

For every constant c of sort $[n]$, let $\mathit{rank}(c) = r$ if $c = k \cdot 2^r$ for some odd natural k and $0 \leq r < n$. Equivalently, $\mathit{rank}(c) = r$ if c, as a bit vector, has the form $d \circ \mathbf{1} \circ \mathbf{0}^r$. Observe that $\mathit{rank}(c) = 0$ iff c itself is odd.

We fix a well-founded strict ordering $\sqsupset$ over Σ_{a}-terms that is total within each sort $[n]$. [CT: The ordering satisfies a number of restrictions, to be added.] The ordering is such that, for any contants c_1, c_2 and monomials m_1, m_2, if $c_1 \cdot m_1 \sqsupset c_2 \cdot m_2$ then $\mathit{rank}(c_1) \leq \mathit{rank}(c_2)$.

We assume that each literal l in the input set L_0 is either

1. a Σ_{c}-literal or
2. a *polynomial equality* of the form
$$p \approx_n 0$$
where p is a polynomial of sort $[n]$ irreducible with respect to the rewrite system $\longrightarrow_{\mathrm{a}}$ below. We say the the polynomial equalities are of sort $[n]$.

Polynomials will be put in a normal form with respect to the rewrite system $\longrightarrow_{\mathrm{a}}$ obtained by extending the rewrite system $\longrightarrow_{\mathrm{c}}$ with the rules below, which apply modulo AC of $+$ and $\cdot$.

EvalConst

$$x \;\longrightarrow_{\mathrm{a}}\; c \quad \text{if} \begin{cases} x \neq c \\ x \text{ contains no variables and evaluates to value } c \end{cases}$$

TimesZero

$$\mathbf{0}^n \cdot x \;\longrightarrow_{\mathrm{a}}\; \mathbf{0}^n$$

Add

$$c \cdot y \,+\, d \cdot y \;\longrightarrow_{\mathrm{a}}\; (c + d) \cdot y \quad \text{if } \; c, d \text{ non-zero values}$$

Distribute

$$(x + y) \cdot z \;\longrightarrow_{\mathrm{a}}\; x \cdot z + y \cdot z$$

[CT: For now, for simplicity we apply distributivity eagerly, even if this can cause an exponential explosion in the worst case. We can see later if we can add an abstraction rule that allows us to apply distributivity in a more selective manner.]

Prefix

$$(x_{[m]} \circ y_{[n]}) \cdot z \ \longrightarrow_{\mathrm{a}} \ 2^n \cdot (\mathbf{0}^n \circ x_{[m]}) \cdot z + (\mathbf{0}^m \circ y_{[n]}) \cdot z \quad \text{if} \ \ (*)$$

$$(*) = \begin{cases} x \circ y \text{ not a value} \\ x \text{ maximal length non-zero value or } \mathbf{0}\text{-prefix term} \end{cases}$$

ExtractArith

$$(x_{[m]} \bowtie y_{[m]})[n-1:0] \ \longrightarrow_{\mathrm{a}} \ x_{[m]}[n-1:0] \bowtie y_{[m]}[n-1:0] \quad \text{if} \ \begin{cases} \bowtie \in \{+, \cdot\} \\ m > n \end{cases}$$

It is possible to show that a is confluent and normalizing modulo AC of $+$ and $\cdot$. [CB: Actually, this system is not normalizing because of the problem of general extracts applied to arithmetic operations.] [CT: We need to see then if we can relax the normalization assumption.]

For each Σ_{a}-term t, let $t{\downarrow_{\mathrm{a}}}$ denote t's normal form with respect of a.

The derivation system in the previous section is extended by the rules below, which apply modulo AC of $\cdot$ and $+$. In the rules, we omit writing the σ component of the pair $\sigma \vdash L$, when it is neither accessed nor modified.

ExtractPlus

$$\frac{L,\ z \approx (x_{[m]} + y_{[m]})[i:j]}{\begin{array}{l} L,\ z \approx (x_{[m]}[i:j] + y_{[m]}[i:j]),\ \mathbf{0} \circ x_{[m]}[j-1:0] +_{[j]} \mathbf{0} \circ y_{[m]}[j-1:0] < \mathbf{10}^j \qquad || \\ L,\ z \approx (x_{[m]}[i:j] + y_{[m]}[i:j] + \mathbf{0}^{i-j}\mathbf{1}),\ \mathbf{01}^j < \mathbf{0} \circ x_{[m]}[j-1:0] +_{[j]} \mathbf{0} \circ y_{[m]}[j-1:0] \end{array}}$$

if $j > 0$

Eliminate

$$\frac{L,\ c \cdot x + y \approx_n 0,\ d \cdot x + z \approx_n 0}{L,\ c \cdot x + y \approx_n 0,\ (c' \cdot z - 2^k \cdot d' \cdot y){\downarrow_{\mathrm{a}}} \approx_n 0} \quad \text{if} \ \begin{cases} c \cdot x \ \sqsupset\text{-largest } \cdot\text{-term in } c \cdot x + y \\ c = c' \circ \mathbf{0}^m \text{ with } c' \text{ odd} \\ d = d' \circ \mathbf{0}^{m+k} \text{ with } d' \text{ odd} \end{cases}$$

FailAr

$$\frac{L,\ c_1 \cdot x_1 + \ldots + c_k \cdot x_k + c \approx 0}{\bot} \quad \text{if} \ \begin{cases} c_1 \cdot x_1 \ \sqsupset\text{-largest } \cdot\text{-term in} \\ \quad c_1 \cdot x_1 + \ldots + c_k \cdot x_k + c, \\ \mathit{rank}(c_1) > \mathit{rank}(c) \end{cases}$$

When the input set is a system of linear polynomial equalities of the same sort all of whose indeterminates are variables, the derivation system defined so far can determine the satisfiability of the system using only the rules **Eliminate**, **FailAr**, **FailEq**, and **SimplifyEq** (the last two from the core derivation system).

Definition 5.2. *A set S of polynomial equalities of sort $[n]$ is in* triangular form *if $S = \emptyset$ or $S = \{c \cdot x + p \approx_n 0\} \cup T$ where $c \cdot x$ is the $\sqsupset$-largest $\cdot$-term in S, x does not occur in T, and T is in triangular form.*

The set S is in solved triangular form *if it is in triangular form and for all of its equalities $c_1 \cdot x_1 + \ldots + c_k \cdot x_k + c \approx_n 0$, if $c_1 \cdot x_1$ is the $\sqsupset$-largest $\cdot$-term in the polynomial then $rank(c_1) \leq rank(c)$.*

Applying **Eliminate**, **FailAr** and **FailEq** to completion to a set of polynomial equalities of the same sort derives either $\bot$ or a set in solved triangular form. [CT: Not true right now. More ordering restrictions on **Eliminate** are needed to prevent reintroducing eliminated indeterminates by combining again already processed equations with others.]

Lemma 5.3. *Let S be a set of linear equations in solved triangular form whose indeterminates are all variables. Then, S is satisfiable.*

Proof. We prove the claim, by induction on the size of S.

Base) $S = \emptyset$. Immediate.

Step) $S = \{c \cdot x + p \approx_n 0\} \cup T$ where $c \cdot x$ is the $\sqsupset$-largest $\cdot$-term in S. By definition, the set T is in triangular form and so it is satisfiable by induction hypothesis. Let α be any satisfying valuation of T's variables extended arbitrarily to the variables of p that do not occur in T. Clearly, α satisfies T. We further extend α to x_1 so that it satisfies $c \cdot x + p \approx_n 0$ as well.

Let $p = c_1 \cdot x_1 + \cdots + c_k \cdot x_h + c_{h+1}$ for some $h \geq 0$. Let $r = rank(c)$. Then $c = k \cdot 2^r$ for some odd k. Since $c \cdot x$ is the $\sqsupset$-largest $\cdot$-term in S and S is in solved triangular form, for each $i = 1, \ldots, h+1$ there is a k_i and an $n_i \geq 0$ such that $c_i = k_i \cdot 2^{r+n_i}$. It follows that $\alpha(p) = 2^r \cdot a$ for some natural number a. Let $\alpha(x) = -k^{-1} \cdot a$ (where $-k^{-1}$ is the additive inverse of the multiplicative inverse of k in $\mathcal{BV}_n$.) Then,

$$\alpha(c \cdot x + p) = k \cdot 2^r \cdot \alpha(x) + \alpha(p) = k \cdot 2^r \cdot (-k^{-1} \cdot a) + 2^r \cdot a = 0 \ .$$

It follows that α satisfies S. □

[CT: Add observation that unless r in the proof above is 0, the value of x is not uniquely determined by α: other values for $\alpha(x)$ than the given one are possible. For instance if $S = \{2 \cdot x - 4 \approx_3 0\}$, the possible values for x are 2 and 6.]

We now extend the derivation system further with rules for processing polynomial equalities that are non-linear or contain non-variable prefix-terms.

Propagate1

$$\frac{L,\ M,\ c \cdot x_{[m+n]} + c' \approx 0}{L,\ M,\ (x[m-1:0] \circ \mathbf{0}^n \approx -c' \cdot d^{-1})\downarrow_{\mathrm{a}}} \quad \text{if } \begin{cases} n \geq 0 \\ c = 2^n \cdot d \text{ with } d \text{ odd} \\ c' \text{ value} \end{cases}$$

Propagate2

$$\frac{L,\ c \cdot x - c \cdot y \approx 0}{L,\ x \approx y} \quad \text{if } \begin{cases} c \text{ odd} \\ x, y\ \mathbf{0}\text{-prefix terms} \end{cases}$$

Propagate1 and **Propagate2** propagate entailed equalities between **0**-prefix terms to the core solver. Note however that they are actually reduction rules.

SplitMul

$$\frac{L}{L,\ z \approx \mathbf{0}^n \qquad L,\ z \approx \mathbf{1}^n \qquad L,\ z \not\approx \mathbf{0}^n,\ z \not\approx \mathbf{1}^n} \quad \text{if } \ z_{[n]} \text{ variable in a non-linear monomial of } L$$

RefineMul

$$\frac{L}{L,\ z \approx z_1 \circ z_2} \quad \text{if } \begin{cases} z_{[n]} \text{ variable in a non-linear monomial of } L \\ z_1, z_2 \text{ fresh variables} \end{cases}$$

The system above should be sound and complete. [CT: To check if completeness does indeed hold given the non-linearity restriction on **SplitMul** and **RefineMul**. Also, we may still need the old **Slice** rule.]

[CT: To be added: processing of non-linear monomials (counterpart of the old **Binomial** and **Superpose** rules).]

5.2 Approach B

We fix a well-founded strict ordering $\sqsupset$ over Σ_a-terms that is total within each sort $[n]$. [CT: The ordering satisfies a number of restrictions, to be added.]

We assume that each literal l in the input set L_0 is either

1. a Σ_c-literal or
2. a *polynomial equality* of the form
$$p \approx q$$
where p and q are polynomials of the same sort and $p \approx q$ is irreducible with respect to the rewrite system $\longrightarrow_a$ defined below.

Polynomials will be put in a normal form with respect to the rewrite system $\longrightarrow_{\mathrm{a}}$ obtained by extending the rewrite system $\longrightarrow_{\mathrm{c}}$ with the rules below, which apply modulo AC of $+$ and $\cdot$.

EvalConst

$$x \ \longrightarrow_{\mathrm{a}} \ c \quad \text{if} \ \begin{cases} x \neq c \\ x \text{ contains no variables and evaluates to value } c \end{cases}$$

PlusZero

$$x + \mathbf{0}^n \ \longrightarrow_{\mathrm{a}} \ x$$

TimesZero

$$\mathbf{0}^n \cdot x \ \longrightarrow_{\mathrm{a}} \ \mathbf{0}^n$$

Add

$$c \cdot y \ + \ d \cdot y \ \longrightarrow_{\mathrm{a}} \ (c + d) \cdot y \quad \text{if} \ \ c, d \text{ non-zero values}$$

Distribute

$$(x + y) \cdot z \ \longrightarrow_{\mathrm{a}} \ x \cdot z + y \cdot z$$

[CT: For now, for simplicity we apply distributivity eagerly, even if this can cause an exponential explosion in the worst case. We can see later if we can add an abstraction rule that allows us to apply distributivity in a more selective manner.]

Prefix

$$(x_{[m]} \circ y_{[n]}) \cdot z \ \longrightarrow_{\mathrm{a}} \ 2^n \cdot (\mathbf{0}^n \circ x_{[m]}) \cdot z + (\mathbf{0}^m \circ y_{[n]}) \cdot z \quad \text{if} \ \ (*)$$

$$(*) = \begin{cases} x \circ y \text{ not a value} \\ x \text{ maximal length non-zero value or } \mathbf{0}\text{-prefix term} \end{cases}$$

ExtractArith

$$(x_{[m]} \bowtie y_{[m]})[n-1:0] \ \longrightarrow_{\mathrm{a}} \ x_{[m]}[n-1:0] \bowtie y_{[m]}[n-1:0] \quad \text{if} \ \begin{cases} \bowtie \in \{+, \cdot\} \\ m > n \end{cases}$$

It is possible to show that a is confluent and normalizing modulo AC of $+$ and $\cdot$. For each Σ_{a}-term t, let $t{\downarrow_{\mathrm{a}}}$ denote t's normal form with respect of a.

We extend the rewrite system from terms to polynomial equalities as follows.

Isolate

$$y \approx x \ \longrightarrow_{\mathrm{a}} \ 1 \cdot z \approx u \quad \text{if} \begin{cases} c \cdot z \ \sqsupset\text{-largest odd-coefficient monomial in } (y - x){\downarrow_{\mathrm{a}}} \\ u = (1 \cdot z - c^{-1} \cdot (y - x))){\downarrow_{\mathrm{a}}} \\ y \text{ is not } 1 \cdot z \end{cases}$$

Group

$$y_{[n]} \approx x \ \longrightarrow_{\mathrm{a}} \ \mathbf{0}^n \approx (x - y){\downarrow_{\mathrm{a}}} \quad \text{if} \begin{cases} y \neq \mathbf{0}^n \\ (y - x){\downarrow_{\mathrm{a}}} \ \text{ contains no odd-coefficient monomials} \end{cases}$$

The derivation system in the previous section is extended by the rules below, which apply modulo AC of $\cdot$ and $+$. The **Solve** rule from the core theory is modified to normalize its conclusion with respect to $\longrightarrow_{\mathrm{a}}$ instead of $\longrightarrow_{\mathrm{c}}$.

In the rules below we omit writing the σ component of the pair $\sigma \vdash L$, when it is neither accessed nor modified.

Eliminate

$$\frac{L,\ M,\ 1 \cdot x \approx y}{L,\ (M\{x \mapsto y\}){\downarrow_{\mathrm{a}}},\ 1 \cdot x \approx y} \quad \text{if} \begin{cases} M \text{ set of all pol. equalities with monomial } x \\ M \text{ non-empty} \end{cases}$$

EqDownsize

$$\frac{L,\ \mathbf{0}^{m+n} \approx (c_0 \circ \mathbf{0}^n) \cdot x_0 + \cdots + (c_k \circ \mathbf{0}^n) \cdot x_k + c_{k+1} \circ \mathbf{0}^n}{L,\ (\mathbf{0}^m \approx c_0 \cdot x_0[m-1:0] + \cdots + c_k \cdot x_k[m-1:0] + c_{k+1}){\downarrow_{\mathrm{a}}}}$$

FailAr

$$\frac{L,\ \mathbf{0}^{m+n} \approx (c_0 \circ \mathbf{0}^n) \cdot x_0 + \cdots + (c_k \circ \mathbf{0}^n) \cdot x_k + c_{k+1}}{\bot} \quad \text{if } \ c \text{ odd}$$

Slice

$$\frac{L,\ 1 \cdot x \approx y}{L,\ 1 \cdot x \approx y,\ (1 \cdot z \approx y[n:0]){\downarrow_{\mathrm{a}}}} \quad \text{if} \begin{cases} x = \mathbf{0}^m \circ z_{[n+1]} \\ z \text{ occurs as an indeterminate in } L \end{cases}$$

The four rules above are used to process polynomial equalities and put them in a "solved form" where each maximal term in an polynomial equality occurs with coefficient 1 and does not occur in other polynomial equalities.

Binomial

$$\frac{L,\ 1 \cdot x \approx z,\ 1 \cdot y \approx z}{L,\ 1 \cdot x \approx 1 \cdot y,\ 1 \cdot y \approx z} \quad \text{if } \ x \sqsupset y$$

Propagate1

$$\frac{L,\ 1 \cdot x \approx c}{L,\ x \approx c} \quad \text{if } \begin{cases} x\ \mathbf{0}\text{-prefix term} \\ c \text{ value} \end{cases}$$

Propagate2

$$\frac{L,\ 1 \cdot x \approx 1 \cdot y}{L,\ x \approx y} \quad \text{if } \ x, y\ \mathbf{0}\text{-prefix terms}$$

Propagate1 and **Propagate2** propagate entailed equalities between **0**-prefix terms to the core solver. Note however that they are actually reduction rules.

SplitMul

$$\frac{L}{L,\ z \approx \mathbf{0}^n \qquad L,\ z \approx \mathbf{1}^n \qquad L,\ z \not\approx \mathbf{0}^n,\ z \not\approx \mathbf{1}^n} \quad \text{if } \ z_{[n]} \text{ variable in a non-linear monomial of } L$$

RefineMul

$$\frac{L}{L,\ z \approx z_1 \circ z_2} \quad \text{if } \begin{cases} z_{[n]} \text{ variable in a non-linear monomial of } L \\ z_1, z_2 \text{ fresh variables} \end{cases}$$

The system above should be sound and complete. [CT: To check if completeness does indeed hold given the non-linearity restriction on **SplitMul** and **RefineMul**.]

The rules above are all simplification rules. One can also imagine adding (judicious applications of) the following Groebner bases-style propagation rule as well.

Superpose

$$\frac{L,\ x \cdot y_1 \approx z_1,\ x \cdot y_2 \approx z_2}{L,\ x \cdot y_1 \approx z_1,\ x \cdot y_2 \approx z_2,\ (z_1 \cdot y_2 \approx z_2 \cdot y_1){\downarrow_{\mathrm{a}}}} \quad \text{if } \begin{cases} y_1 \text{ not a submonomial of } y_2 \\ y_2 \text{ not a submonomial of } y_1 \\ z_1, z_2 \text{ monomials} \end{cases}$$

Note that by normalization we have $x \cdot y_1 \sqsupset z_1$ and $x \cdot y_2 \sqsupset z_2$.

[CT: Rule below to be revised] **Slice2**

$$\frac{L,\ c \cdot x + z \approx \mathbf{0}^{m+n}}{L,\ c \cdot x + z \approx \mathbf{0}^{m+n},\ ((c \cdot x + z)[n + k - 1 : 0]){\downarrow_{\mathrm{a}}} \approx \mathbf{0}^n} \quad \text{if } \begin{cases} c = d \circ \mathbf{0}^k \text{ with } d \text{ odd} \\ x = \mathbf{0}^m \circ y \\ y_{[n]} \text{ variable} \end{cases}$$

6 Reasoning in the algebra $\mathcal{BV}_{\mathrm{o}}$

[CT: Rule set to be completed.]

We now consider the restriction of $\mathcal{BV}$ to the signature $\Sigma_{\mathrm{o}} = \Sigma_{\mathrm{c}} \cup \{<\}$ and assume that each literal l in the input set L_0 is a Σ_{o}-literal irreducible wrt the rewrite system $\longrightarrow_{\mathrm{c}}$.

If c is a bit vector value other than ϵ, we write $\mathit{num}(c)$ for the (positive) integer value denoted by c as a number in base 2. We write $\mathit{num}(\epsilon)$ for -1.

Since none of the derivation rules presented in this section modifies the σ component of the pair $\sigma \vdash L$, we streamline their notation by omitting that component.

The expression $x_1 <^k y$ abbreviates the chain $x_1 < x_2,\ x_2 < x_3,\ \ldots,\ x_k < y$ for some $x_2, \ldots, x_k$ with $k \geq 1$. The expression $x <^+ y$ abbreviates $x <^k y$ for some $k \geq 1$.

Cycle

$$\frac{L,\ x <^+ x}{\bot}$$

Chain1

$$\frac{L,\ x <^k c}{\bot} \quad \text{if } \begin{cases} c \text{ value} \\ k > \mathit{num}(c) \end{cases}$$

Chain2

$$\frac{L,\ c <^k x_{[n]}}{\bot} \quad \text{if } \begin{cases} c \text{ value} \\ k \geq 2^n - \mathit{num}(c) \end{cases}$$

Chain3

$$\frac{L,\ c <^k d}{\bot} \quad \text{if } \begin{cases} c, d \text{ values} \\ k \geq \mathit{num}(d) - \mathit{num}(c) \end{cases}$$

Note that **Chain3** also applies, for any k, when d is not greater than c.

Chain4

$$\frac{L,\ x_{[n]} <^k y}{\bot} \quad \text{if } k \geq 2^n$$

LowerBound

$$\frac{L,\ x < \mathbf{0}^n\mathbf{1}}{L,\ x < \mathbf{0}^n\mathbf{1},\ x \approx \mathbf{0}^{n+1}}$$

UpperBound

$$\frac{L,\ \mathbf{1}^n\mathbf{0} < x}{L,\ \mathbf{1}^n\mathbf{0} < x,\ x \approx \mathbf{1}^{n+1}}$$

Instantiate

$$\frac{L,\ b < x,\ x < d}{L,\ b < x,\ x < d,\ x \approx c} \quad \text{if} \begin{cases} b, c, d \text{ values} \\ num(c) = num(b) + 1 \\ num(d) = num(c) + 1 \\ x \text{ non-value} \end{cases}$$

Eliminate

$$\frac{L,\ x <^{+} y,\ x < y}{L,\ x <^{+} y}$$

DownTighten

$$\frac{L,\ x < y,\ y < d}{L,\ x < y,\ y < d,\ x < c} \quad \text{if} \begin{cases} c, d \text{ values} \\ num(c) = num(d) - 1 \\ x, y \text{ non-values} \end{cases}$$

UpTighten

$$\frac{L,\ c < y,\ y < z}{L,\ c < y,\ y < z,\ d < z} \quad \text{if} \begin{cases} c, d \text{ values} \\ num(d) = num(c) + 1 \\ y, z \text{ non-values} \end{cases}$$

NegOrd

$$\frac{L,\ x \not< y}{L,\ x \approx y \qquad L,\ y < x}$$

DownPropag

$$\frac{L,\ x_{[m]} \circ y_{[n]} < \mathbf{0}^m \circ z}{L,\ x_{[m]} \circ y_{[n]} < \mathbf{0}^m \circ z,\ x \approx \mathbf{0}^m,\ y < z} \quad \text{if } L \text{ contains constraints over } [m] \text{ or } [n]$$

UpPropag

$$\frac{L,\ \mathbf{1}^m \circ z < x_{[m]} \circ y_{[n]}}{L,\ \mathbf{1}^m \circ z < x_{[m]} \circ y_{[n]},\ x \approx \mathbf{1}^m,\ z < y} \quad \text{if } L \text{ contains constraints over } [m] \text{ or } [n]$$

Split

$$\frac{L,\ x_{[m]} \circ y_{[n]}\ <\ z_{[n]} \circ v}{L,\ x_{[m]} \circ y_{[n]}\ <\ z_{[n]} \circ v,\ x < z \qquad L,\ x_{[m]} \circ y_{[n]}\ <\ z_{[n]} \circ v,\ x \approx z,\ y < v} \quad (*)$$

$(*) = L$ contains constraints over $[m]$ or $[n]$

[CT: It looks like a refinement rule is not needed for completeness. But we might need an alignment for the last three rules above. To be checked.]

7 Reasoning in the algebra $\mathcal{BV}_{\mathrm{b}}$

We now consider the restriction of $\mathcal{BV}$ to the signature $\Sigma_{\mathrm{b}} = \Sigma_{\mathrm{c}} \cup \{\sim, |, \ \&\ , \oplus\}$.

This section presents two alternative approaches, the first based on a lazy reduction to Boolean CNF formulas and the second on direct reasoning in Boolean rings.

Since none of the derivation rules presented in this section modifies the σ component of the pair $\sigma \vdash L$, we streamline their notation by omitting that component.

In the following, we say that $x_{[n]}$ *is a uniform prefix of* $x \circ y$ if x is a variable, $\mathbf{0}^n$ or $\mathbf{1}^n$.

7.1 First approach

We assume that each literal l in the input set L_0 is either a Σ_{c}-literal or has the form

$$t_1 \mid \cdots \mid t_k \approx \mathbf{1}^n$$

modulo associativity and commutativity of $|$ where $n, k > 1$ and each t_i is either a Σ_{c}-term s or a negated Σ_{c}-term $\sim s$. This assumption can be made without loss of generality, as we can apply common purification and CNF conversion rules to convert an arbitrary set of Σ_{b}-literals into an $\mathcal{BV}$-equisatisfiable set of the form above.[5]

We expand the signature with the following family of functions symbols

$$\{\mathsf{hi} :: [n] \to [1]\}_{n>0}$$

and expand $\mathcal{BV}$ to interpret hi as the function that maps a bit vector to its highest bit. Semantically then, the term $\mathsf{hi}(x_{[n]})$ is equivalent to $x_{[n]}[n-1:n-1]$. Operationally, however, it will be treated differently.

We first extend the rewrite system $\longrightarrow_{\mathrm{c}}$ to the rewrite system $\longrightarrow_{\mathrm{b}}$ with the addition of the following rules.

The first rule is a "big-step" rule that applies to any ground Σ_{b}-terms. Note that the rule is used also to simplify terms of the form $t_1 \mid \cdots \mid t_k$.

EvalConst

$$x \ \longrightarrow_{\mathrm{b}}\ c \quad \text{if} \ \begin{cases} x \neq c \\ x \text{ contains no variables and evaluates to value } c \end{cases}$$

HighConcat

$$\mathsf{hi}(x \circ y) \ \longrightarrow_{\mathrm{b}}\ \mathsf{hi}(x)$$

[5]As usual, care must be taken if one needs to preserve equivalence (modulo extra fresh variables) instead of just satisfiability.

HighBit

$$\mathsf{hi}(x_{[1]}) \ \longrightarrow_{\mathrm{b}} \ x$$

HighNeg

$$\mathsf{hi}({\sim}x) \ \longrightarrow_{\mathrm{b}} \ {\sim}\mathsf{hi}(x)$$

The extended rewrite system $\longrightarrow_{\mathrm{b}}$ is again confluent and normalizing—although distinct equivalent terms may now have distinct normal forms.

We extend the derivation system in the previous section by starting with sets of literals in normal form wrt $\longrightarrow_{\mathrm{b}}$. The old rules now normalize wrt $\longrightarrow_{\mathrm{b}}$. Then we add the rules below as well.

In the new rules, $x_1 \mid \cdots \mid x_k$ denotes a term with each x_i of the form t or ${\sim}t$ for some (normal-form) Σ_{c}-term t. Somewhat ambiguously, we also use $({\sim})x_1 \mid \cdots \mid ({\sim})x_k$ to mean that each x_i is a possibly negated Σ_{c}-term. Individual negation symbols carry from the premise to the conclusions of a rule as expected. [CT: This is imprecise but making it rigorous would complicate the rules, perhaps unnecessarily.] Similarly to disequalities in the previous derivation systems, now equalities can be marked (with $\surd$) too.

AlignDisj

$$\frac{L,\ ({\sim})(y_{[m+n]} \circ z) \mid ({\sim})x_1 \mid \cdots \mid ({\sim})x_k \approx \mathbf{1}^m}{L,\ ({\sim})(y_1 \circ y_2 \circ z) \mid ({\sim})x_1 \mid \cdots \mid ({\sim})x_k \approx \mathbf{1}^m,\ y \approx y_1 \circ y_2} \quad \text{if } \ (*)$$

$$(*) = \begin{cases} m, n, k > 0 \\ y \text{ variable} \\ \text{longest uniform prefix of } x_1 \text{ with size} < m+n \text{ has sort } [m] \\ y_1 :: [m],\ y_2 :: [n] \text{ fresh variables} \end{cases}$$

Reduce

$$\frac{L,\ x_1 \mid \cdots \mid x_k \approx \mathbf{1}^n}{L,\ x_1 \mid \cdots \mid x_k \approx^{\surd} \mathbf{1}^n,\ (\mathsf{hi}(x_1) \mid \cdots \mid \mathsf{hi}(x_k)){\downarrow_{\mathrm{b}}} \approx \mathbf{1}}$$

DecomposeDisj

$$\frac{L,\ ({\sim})(x_1 \circ y_1) \mid \cdots \mid ({\sim})(x_k \circ y_k) \bowtie \mathbf{1}^{m+n}}{L,\ ({\sim})x_1 \mid \cdots \mid ({\sim})x_k \approx \mathbf{1}^m,\ \ ({\sim})y_1 \mid \cdots \mid ({\sim})y_k \approx \mathbf{1}^n} \quad \text{if } \begin{cases} k > 0 \\ \bowtie \in \{\approx, \approx^{\surd}\} \\ x_1 :: [n], \ldots x_k :: [n] \end{cases}$$

Split

$$\frac{L,\ x_1 \mid \cdots \mid x_k \approx \mathbf{1}}{L,\ x_1 \approx \mathbf{1} \qquad \cdots \qquad L,\ x_k \approx \mathbf{1}} \quad \text{if } \ x_i \approx \mathbf{1} \notin L \text{ for all } i = 1, \ldots, k$$

ExtendOne

$$\frac{L,\ \mathsf{hi}(x_{[m+n]}) \approx \mathbf{1}}{L,\ x \approx \mathbf{1}^{m+n} \qquad L,\ \mathsf{hi}(x) \approx \mathbf{1},\ x \approx y_{[m]} \circ z_{[n]}} \quad \text{if } \begin{cases} m, n > 0 \\ x \text{ variable} \\ y, z \text{ fresh variables} \end{cases}$$

ExtendZero

$$\frac{L,\ {\sim}\mathsf{hi}(x_{[m+n]}) \approx \mathbf{1}}{L,\ x \approx \mathbf{0}^{m+n} \qquad L,\ {\sim}\mathsf{hi}(x) \approx \mathbf{1},\ x \approx y_{[m]} \circ z_{[n]}} \quad \text{if } \begin{cases} m, n > 0 \\ x \text{ variable} \\ y, z \text{ fresh variables} \end{cases}$$

Note that the new rules interact with the old ones dynamically because some of them make **Solve** applicable.

Intuitively, the derivation system processes the Σ_c-constraints as before while processing the Boolean algebra constraints $(\sim)x_1 \mid \cdots \mid (\sim)x_k \approx \mathbf{1}^n$ as follows.

If any x_i is a concatenation $y_1 \circ z_2$, the other x_j's are *aligned* with $y_1 \circ z_2$ and $(\sim)x_1 \mid \cdots \mid (\sim)x_k \approx \mathbf{1}^n$ is replaced by two corresponding constraints over the components of the concatenation. Once all constraints $(\sim)x_1 \mid \cdots \mid (\sim)x_k \approx \mathbf{1}^n$ are over variables only, it first tries to find values for the high bit of each variable that satisfy all the constraints. It that succeeds, it tries to extend such a 0-1 solution to the entire variable, for each variable. If none of the high bit solutions can be extended this way, it guesses a decomposition $y_i \circ z_i$ of some variable x_i of size greater than 1 and repeats the process. In the worst case, all variables are decomposed to the bit level.

An important feature of this system is that when a variable x_i is decomposed as $y_i \circ z_i$ and then solved, all the constraints on the high bit of x_i are recast in terms of y_i via the **HighConcat** rule. This allows the recycling of bit level constraints and their partial solutions.

[CT: It should be possible to show that for sets of equalities only the system is complete without the **Extend*** rules whenever the input contains no disequalities. More generally, bit level solutions are guaranteed to be extensible to word level solutions whenever the current set of literals has no (more) disequalities between variables of size > 1.]

The main strength of this approach is that the overhead of processing Boolean algebra constraints is minimal as it involves very simple simplification rules. The approach however has a major weakness and it is its poor treatment of disequalities. The reason is essentially that it does not recognize terms that are equivalent (under some partial solution σ).

For instance, with input

$$\varepsilon\ \vdash\ {\sim}x \mid y \approx \mathbf{1}^n,\ {\sim}y \mid x \approx \mathbf{1}^n,\ x \not\approx y$$

the system needs to decompose x and y to the bit level before it can conclude that it is unsatisfiable. This represents a serious scalability problem.

The next approach tries to address that, but at the cost of (much?) more expensive simplification rules.

7.2 Second Approach

This approach is inspired by Dershowitz *et al.* [1] who describe a Boolean ring-based procedure for propositional satisfiability. We adapt their idea to $\mathcal{BV}$-satisfiability.

We assume that each literal l in the input set L_0 is either

1. a Σ_c-literal or
2. a linear $\oplus$-*polynomial* equality of the form
$$x_1 \oplus \cdots \oplus x_p \ \approx \mathbf{0}^n$$
(modulo AC of $\oplus$) where $p > 0$ and each x_i is a concat term, or
3. a $\&$-*binomial* equality of the form
$$x_1 \,\&\, \cdots \,\&\, x_p \ \approx\ y_1 \,\&\, \cdots \,\&\, y_q$$
(modulo AC of $\&$) where $p, q > 0$ and each x_i and y_j are concat terms.

Furthermore, each literal is irreducible wrt the rewrite system $\longrightarrow_\mathrm{b}$ below. Note that the three cases above are, intentionally, not mutually exclusive. For instance, for any Σ_c-term x, the equation $x \approx \mathbf{0}^n$ is a Σ-literal, a $\oplus$-polynomial equality and $\&$-binomial equality.

Any initial set of Σ_b-literals can be converted in the format above thanks to a satisfiability preserving transformation like the one in [1].

The main point of the reduced form above is that, contrary to the standard polynomial form for Boolean rings, it can be computed to have always size polynomial in the size of the initial set. At the same time, it allows a lot of cheap simplifications in the theory of Boolean rings when combined with a splitting rule that guesses values for its variables.

We first extend the rewrite system $\longrightarrow_\mathrm{c}$ to the rewrite system $\longrightarrow_\mathrm{b}$ with the addition of the following rules. The rewrite rules apply modulo AC of $\&$ and $\oplus$.

EvalConst

$$x \ \longrightarrow_\mathrm{b}\ c \quad \text{if} \begin{cases} x \neq c \\ x \text{ contains no variables and evaluates to value } c \end{cases}$$

AndOne

$\mathbf{1}^n \mathbin{\&} x \;\longrightarrow_{\mathrm{b}}\; x$

AndZero

$\mathbf{0}^n \mathbin{\&} x \;\longrightarrow_{\mathrm{b}}\; \mathbf{0}^n$

AndIdem

$x \mathbin{\&} x \;\longrightarrow_{\mathrm{b}}\; x$

XorZero

$\mathbf{0}^n \oplus x \;\longrightarrow_{\mathrm{b}}\; x$

XorInv

$x_{[n]} \oplus x_{[n]} \;\longrightarrow_{\mathrm{b}}\; \mathbf{0}^n$

The rewrite system $\longrightarrow_{\mathrm{b}}$ is again confluent and normalizing.

We extend the derivation system in the previous section by starting with sets of literals in normal form wrt $\longrightarrow_{\mathrm{b}}$. The old rules now normalize wrt $\longrightarrow_{\mathrm{b}}$. The **Solve** rule is restricted to apply only to equalities $x \approx y$ where y is a Σ_{c}-term.

The rules below apply modulo AC of $\&$ and $\oplus$ and symmetry of $\approx$.

They assume some total ordering $\succ_{[n]}$ on concat terms of sort $[n]$ for each n. We will write just $\succ$ when n is clear from context or not important. [CT: We might not be able to find a suitable *total* ordering. Things below might have to be adjusted if $\succ$ is only partial. In any case, it'll probably be necessary for $\succ$ to be such that $x \succ y$ whenever y is a proper instance of x.]

The ordering is extended lexicographically (modulo AC) to $\oplus$-polynomials by first sorting the concat terms in a polynomial with respect to $\succ$. The ordering is extended lexicographically to $\&$-monomials in a similar way.

[CT: The **Solve** rule from the core solver may interfere with the purposes of the ordering above, by turning a concat term in a monomial or a polynomial into a smaller concat term. This should be checked out and the rules below tweaked as needed to maintain termination and avoid unnecessary recomputation of solved forms.]

[CT: Implementation note: Both $\&$-monomials and $\oplus$-polymomials could be implemented as sorted lists of Σ_{c}-terms ordered wrt $\succ$. Binomial equations could be kept oriented, with lhs $\succ$ rhs.]

Eliminate

$$\frac{L,\ x \oplus y \approx \mathbf{0}^n,\ x \oplus z \approx \mathbf{0}^n}{L,\ x \oplus y \approx \mathbf{0}^n,\ (y \oplus z){\downarrow_{\mathrm{b}}} \approx \mathbf{0}^n} \quad \text{if} \begin{cases} x\ \succ\text{-maximal concat term in } x \oplus y \\ z \succ y \end{cases}$$

[CT: Implementation note: With polynomials implemented as sorted lists, to obtain $(y \oplus z){\downarrow_{\mathrm{b}}}$ it is enough to merge $x \oplus y$ and $x \oplus y$ eliminating any element that occurs in both lists.]

Equate1

$$\frac{L,\ x \oplus z \approx \mathbf{0}^n,\ y \oplus z \approx \mathbf{0}^n}{L,\ x \approx y,\ y \oplus z \approx \mathbf{0}^n} \quad \text{if} \begin{cases} x \succ\text{-maximal concat term in } x \oplus z \\ y \succ\text{-maximal concat term in } y \oplus z \\ x \succ y \end{cases}$$

[CT: Implementation note: With perfect sharing of *lists* the cost of Equate1 is only quadratic.]

Equate2

$$\frac{L,\ x \oplus y \approx \mathbf{0}^n}{L,\ x \approx y} \quad \text{if} \ \ x, y \text{ concat terms}$$

[CT: Equate1 and Equate2 propagate to the core solver equalities entailed by the polynomials.]

Interreduce

$$\frac{L,\ x \approx z,\ x \,\&\, y \approx u}{L,\ x \approx z,\ (z \,\&\, y){\downarrow_{\mathrm{b}}} \approx u} \quad \text{if} \begin{cases} x \approx z \text{ binomial equality} \\ x \succ z \\ x \,\&\, y \succ u \end{cases}$$

[CT: Implementation note: This rule may be expensive to implement (cubic cost?) because it requires one to check that the elements of x are contained in the left side of some other binomial. (After that, to get $(z \,\&\, y){\downarrow_{\mathrm{b}}}$ it is enough to merge x, z and $x \,\&\, y$ together as in Eliminate.)]

DecomposeAnd

$$\frac{L,\ x \,\&\, y \approx \mathbf{1}^n}{L,\ x \approx \mathbf{1}^n,\ y \approx \mathbf{1}^n}$$

AlignBool

$$\frac{L,\ (y_{[m+n]} \circ z) \diamond x_1 \diamond \cdots \diamond x_p \approx u_1 \diamond \cdots \diamond u_q}{L,\ (y_1 \circ y_2 \circ z) \diamond x_1 \diamond \cdots \diamond x_k \approx u_1 \diamond \cdots \diamond u_q,\ y \approx y_1 \circ y_2} \quad \text{if} \ \ (*)$$

$$(*) = \begin{cases} \diamond \in \{\,\&\,,\oplus\} \\ p + q > 0 \\ m, n > 0 \\ y \text{ variable, } x_1, \ldots x_p, u_1, \ldots, u_q \text{ concat terms} \\ \text{longest uniform prefix of } x_1 \text{ or of } u_1 \text{ with size} < m + n \text{ has sort } [m] \\ y_1 :: [m],\ y_2 :: [n] \text{ fresh variables} \end{cases}$$

[CT: The preconditions of AlignBool try to minimize the break up of variables, subject to the requirement that values should be broken up into subconcatenations of zeros or ones only.]

DecomposeBool

$$\frac{L,\ (x_1 \circ y_1) \diamond \cdots \diamond (x_p \circ y_p) \approx (u_1 \circ v_1) \diamond \cdots \diamond (u_q \circ v_q)}{L,\ x_1 \diamond \cdots \diamond x_p \approx u_1 \diamond \cdots \diamond u_q,\quad y_1 \diamond \cdots \diamond y_p \approx v_1 \diamond \cdots \diamond v_q} \quad \text{if} \begin{cases} \diamond \in \{\,\&\,,\ \oplus\} \\ p, q > 0 \\ x_1, \ldots x_p, u_1, \ldots u_p :: [n] \end{cases}$$

[CT: Implementation notes: It is not clear at the moment whether it would be better to apply RefineBool and DecomposeBool as soon as possible (so that the previous rules operate only on XOR's and AND's of variables only) or, instead, only when the previous rules do not apply anymore (to minimize the proliferation of constraints caused by DecomposeBool).]

SplitAnd

$$\frac{L,\ x_{[n]} \approx y \,\&\, z}{L,\ x_{[n]} \approx \mathbf{0}^n,\ y \approx \mathbf{0}^n \qquad L,\ x_{[n]} \approx z,\ y \approx \mathbf{1}^n \qquad L,\ x_{[n]} \approx y \,\&\, z,\ y \not\approx \mathbf{0}^n,\ y \not\approx \mathbf{1}^n} \quad \text{if}\ \ (*)$$

$(*) = y$ variable

Split can be optimized when $n = 1$ by removing the then unsatisfiable third conclusion.

RefineAnd

$$\frac{L,\ x_{[n]} \approx y \,\&\, z}{L,\ x_{[n]} \approx y \,\&\, z,\ y \approx y_1 \circ y_2} \quad \text{if}\ \ y \text{ variable}$$

[CT: RefineAnd is needed for completeness. One can imagine different strategies where variables are refined enough before SplitAnd is applied.

Applying SplitAnd and RefineAnd only to variables occurring in conjunctions should be enough for completeness.]

The systems above should be sound and complete. [CT: I think completeness is preserved if one stops splitting as soon as none of the other rules applies and the set of $\oplus$-equalities is solved in this sense: each equality has the form $x \oplus y \approx \mathbf{0}^n$ where x is a variable not occurring in anywhere else.]

The rules above are all simplification rules. One can also imagine adding (judicious applications of) the following Groebner bases-style propagation rule as well.

Superpose

$$\frac{L,\ x \,\&\, y_1 \approx z_1,\ x \,\&\, y_2 \approx z_2}{L,\ x \,\&\, y_1 \approx z_1,\ x \,\&\, y_2 \approx z_2,\ (z_1 \,\&\, y_2)\!\downarrow_{\mathrm{b}} \approx (z_2 \,\&\, y_1)\!\downarrow_{\mathrm{b}}} \quad \text{if} \begin{cases} x \,\&\, y_1 \succ z_1 \\ x \,\&\, y_2 \succ z_2 \\ y_1 \text{ not a submonomial of } y_2 \\ y_2 \text{ not a submonomial of } y_1 \\ z_1, z_2 \neq \mathbf{1}^n \end{cases}$$

[CT: Implementation notes: Superpose candidates could be found by computing the differences $l_1 \setminus l_2$ and $l_2 \setminus l_1$ of two lhs lists of two binomials (which yield respectively y_1 and y_2 in the rule when the length of y_1 is smaller then that of l_1). With ordered lists, the set difference computation is linear, so the total cost is quadratic.]

Another propagation rule could be the following.

XorAnd

$$\frac{L,\ x_1 \oplus \cdots \oplus x_{2p+1} \approx \mathbf{0}^n}{L,\ x_1 \oplus \cdots \oplus x_{2p+1} \approx \mathbf{0}^n,\ (x_1 \ \&\ \cdots\ \&\ x_{2p+1}){\downarrow_{\mathrm{b}}} \approx \mathbf{0}^n} \quad \text{if } \begin{cases} p > 0 \\ x_i \text{ concat term} \end{cases}$$

8 Reasoning in the algebra $\mathcal{BV}_{\mathrm{s}}$

[CT: Rule set to be completed.]

We now consider the restriction of $\mathcal{BV}$ to the signature $\Sigma_{\mathrm{s}} = \Sigma_{\mathrm{c}} \cup \{\ll, \gg\}$.

Since none of the derivation rules presented in this section modifies the σ component of the pair $\sigma \vdash L$, we streamline their notation by omitting that component.

We assume that each literal l in the input set L_0 is either

- a Σ_{c}-literal or
- an equality of the form

$$z \approx\ x \ll y \quad \text{or} \quad z \approx\ x \gg y$$

  where x, y, z are concat terms.

Furthermore, each literal is irreducible wrt the rewrite system $\longrightarrow_{\mathrm{s}}$ below. Any set of Σ_{a}-literals can be converted in the format above by standard flattening operations.

If c is a bit vector value, we write $\mathit{num}(c)$ for the natural number represented by c in base 2. We extend the rewrite system $\longrightarrow_{\mathrm{c}}$ to the rewrite system $\longrightarrow_{\mathrm{s}}$ with the addition of the following rules.

LeftShiftZero

$$x \ll \mathbf{0}^{n+1} \ \longrightarrow_{\mathrm{s}}\ x$$

LeftShiftAll

$$x_{[n+1]} \ll y \ \longrightarrow_{\mathrm{s}}\ \mathbf{0}^{n+1} \quad \text{if} \ \begin{cases} y =_U u \circ c_{[p]} \circ v_{[q]} \\ c \text{ maximal length non-zero value in } y \\ p > 0 \\ \mathit{num}(c \circ \mathbf{0}^q) > n \end{cases}$$

[CT: Derivation rules.]

LeftShift

$$\frac{L,\ z_{[n+1]} \approx x \ll y}{L,\ (z[n:i] \approx x[n-i:0] \ll u \circ \mathbf{0}^p \circ v)\!\downarrow_{\mathrm{s}},\ (z[i-1:0] \approx \mathbf{0}^i)\!\downarrow_{\mathrm{s}}} \quad \text{if } (*)$$

$$(*) = \begin{cases} y =_U u \circ c_{[p]} \circ v_{[q]} \\ c \text{ maximal length non-zero value in } y \\ p > 0 \\ i = \mathit{num}(c \circ \mathbf{0}^q) \leq n \end{cases}$$

References

[1] Nachum Dershowitz, Jieh Hsiang, Guan-Shieng Huang, and Daher Kaiss. Boolean rings for intersection-based satisfiability. In Miki Hermann and Andrei Voronkov, editors, *13th International Conference on Logic for Programming and Artificial Intelligence and Reasoning (Phnom Penh, Cambodia*, volume 4246 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2006.