Solving Bit-Vector Equations*

M. Oliver Möller[1] and Harald Rueß[2]

[1] University of Århus
Department of Computer Science
Ny Munkegade, building 540
DK - 8000 Århus C, Denmark
omoeller@brics.dk

[2] SRI International
Computer Science Laboratory
333 Ravenswood Ave.
Menlo Park, CA, 94025, USA
ruess@csl.sri.com

Abstract. This paper is concerned with solving equations on fixed and non-fixed size bit-vector terms. We define an equational transformation system for solving equations on terms where all sizes of bit-vectors and extraction positions are known. This transformation system suggests a generalization for dealing with bit-vectors of unknown size and unknown extraction positions. Both solvers adhere to the principle of splitting bit-vectors only on demand, thereby making them quite effective in practice.

1 Introduction

Efficient automation of bit-vector reasoning is essential for the effective mechanization of many hardware verification proofs. It has been demonstrated, for example, in [SM95] that the lack of such a capability forms one of the main impediments to cost-effective verification of industrial-sized microprocessors.

Here we are specifically concerned with the problem of solving equational bit-vector constraints. Given an equation on bit-vector terms, a *solver* yields *true* if this equation is valid, *false* if it is unsatisfiable, and an equivalent system of solved equations otherwise. A solver can therefore not only be used to decide formulas but also to compute a compact representation of the set of all interpretations (witnesses) for a given formula. Consider, for example, the formula $x_{[6]} \otimes y_{[2]} = y_{[2]} \otimes x_{[6]}$, where $x_{[6]}$, $y_{[2]}$ are variables for bit-vectors of length 6 and 2, respectively, and $.\otimes.$ denotes concatenation of bit-vectors. A solved form for this formula is given by $x_{[6]} = y_{[2]} \otimes y_{[2]} \otimes y_{[2]}$ without further restrictions on $y_{[2]}$. Moreover, solving is the central concept for deciding formulas in the combination of theories, since Shostak's algorithm [Sho84] consists of a composition of solvers for the individual theories. Thus, solving bit-vector equations opens the door for deciding many hardware verification problems including operations from other fundamental theories like arithmetic, lists, or arrays.

* This research was supported in part by the Deutsche Forschungsgemeinschaft (DFG) project *Verifix* and by the Deutscher Akademischer Austauschdienst (DAAD). The work undertaken at SRI was partially supported by the National Science Foundation under grant No. CCR-9509931. Accepted for publication in: Proc. of Formal Methods in Computer-Aided Design (FMCAD'98), G. Gopalakrishnan, Ph. Windley (Eds.), Lecture Notes in Computer Science, Springer Verlag, Palo Alto, November, 1998.

It is our basic premise that the peculiarities of bit-vectors and the particular combination of bit-vector operations like concatenation, extraction of contiguous parts, bitwise Boolean operations, and finite arithmetic require specialized reasoners to effectively deal with bit-vector problems instead of simply encoding bit-vectors and operations there upon in, say, arithmetic. Such a specialized solver has been described, for example, in [CMR96, CMR97]. The key design feature of this algorithm is that it only splits bit-vectors on demand. In this way, run times of the solver are—in extreme cases—independent of the lengths of the data paths involved. The solver in [CMR97] includes a number of optimizations to improve usability. Unfortunately, these low-level details also tend to distract from the underlying, basically simple concepts of this algorithm.

In this paper we remedy the situation and reconstruct the solver in [CMR97] in terms of an equational transformation system. Besides the advantage of separating the conceptual ideas of the algorithm from low-level efficiency issues, the description of solving as an equational transformation system suggests a generalization to deal with equations on non-fixed size bit-vector terms, i.e., terms of unknown size or with integer variables as extraction positions.

We proceed as follows. In Section 2 we fix the notation used throughout this paper, review basic concepts of bit-vectors, and define the problem of solving. Hereby, we restrict ourselves to the theory of bit-vectors with the fundamental operations of concatenation and extraction only, since other bit-vector operations, like bitwise Boolean operations, can be added in a conceptually clean way using the notion of bit-vector OBDDs [CMR97, Möl98]. Section 3 forms the core of this paper and describes a rule-based algorithm for solving fixed size bit-vector equations. In Section 4 we extend the rule-based algorithm for solving equations on non-fixed size bit-vector terms. Section 5 concludes with some remarks.

Prototypical implementations of the fixed size and the non-fixed size bit-vector solvers are available from:

`http://www.informatik.uni-ulm.de/ki/Bitvector/`

2 Preliminaries

This section contains background material on Shostak's procedure for deciding combinations of quantifier-free theories and on the theory of bit-vectors.

Canonization. Shostak's procedure [Sho84] operates over a subclass of certain unquantified first-order theories called σ-theories. Informally, these theories have a computable canonizer function σ from terms (in the theory) to terms, such that an equation $t = u$ is valid in the theory if and only if $\sigma(t) \doteq \sigma(u)$, where $\doteq$ denotes syntactic equality. More precisely, the full set of requirements on canonizers—as stated in [CLS96]—are as follows.

Definition 1. *Let T be a set of terms, possibly containing subterms not in the theory. Then $\sigma : T \to T$ is called a* canonizer *if it fulfills the properties:*

1. *An equation $t = u$ in the theory is valid if and only if $\sigma(t) \doteq \sigma(u)$.*

2. *If* $t \notin T$ *then* $\sigma(t) \doteq t$.
3. $\sigma(\sigma(t)) \doteq \sigma(t)$
4. *If* $\sigma(t) \doteq f(t_1, \ldots, t_n)$ *for a term* $t \in T$ *then* $\sigma(t_i) \doteq t_i$ *for* $1 \leq i \leq n$.
5. $vars(\sigma(t)) \subseteq vars(t)$

Algebraic Solvability. To construct a decision procedure for equality in a combination of σ-theories, Shostak's method requires that the σ-theories have the additional property of *algebraic solvability.* A σ-theory is algebraically solvable if there exists a computable function *solve*, that takes an equation $s = t$ and returns *true* for a valid equation, *false* for an unsatisfiable equation, and, otherwise, an equivalent conjunction of equations of the form $x_i = t_i$, where the x_i's are distinct variables of $s = t$ that do not occur in any of the t_i's; notice that the t_i's may be constructed using some variables not occurring in the source equation $s = t$.

Definition 2. *Let* $E = solve(e)$, t_i *denote terms in some algebraic theory, and* $x_i \in vars(e)$*; then* $solve(.)$ *is called a* solver *for this theory if the following requirements are fulfilled [CLS96].*

1. $\vdash e \quad \Leftrightarrow \quad solve(e)$ *is in the theory.*
2. $E \in \{\text{true}, \text{false}\}$ *or* $E = \bigwedge_i x_i = t_i$.
3. *If* e *contains no variables, then* $E \in \{\text{true}, \text{false}\}$.
4. *If* $E = \bigwedge_i x_i = t_i$ *then* $x_i \in vars(e)$ *and for all* i, j*:* $x_i \notin vars(t_j)$, $x_i \neq x_j$, *and* $t_i \doteq \sigma(t_i)$.

Systems of equations E *satisfying these requirements are also said to be in* solved form *with respect to* e.

Solving an equation $t = u$ yields an explicit description of all satisfying models $\mathcal{I}$ of $t = u$, i.e., assignments of the variables to terms such that $\mathcal{I} \models t = u$ where semantic entailment $\models$ is defined in the usual way.

Example 1. A solved form for the equation $(x \vee y) = \neg z$ on Boolean terms is given by $\{x = a,\ y = b,\ z = \neg a \wedge \neg b\}$, where a, b are fresh variables.

Notice that solving is a rather powerful concept, since it can be used to decide fully-quantified equations by inspecting dependencies between the terms on the right-hand side of solved forms. The formula $\forall x, y.\ \exists z.\ (x \vee y) = \neg z$, for example, is valid since the solved form of the equation in the body (see Example 1) does not put any restrictions on the $\forall$-quantified variables. This argument holds in general.

Bit-vectors. Fixed size bit-vector terms have an associated (positive) length n, and the bits in a bit-vector of length n are indexed from 0 (leftmost) to $n-1$ (rightmost). In the following, $s_{[n]}$, $t_{[m]}$, $u_{[k]}$ denote bit-vector terms of lengths n, m, k, respectively, and i, j are positions in bit-vectors; whenever possible we omit subscripts. Bit-vectors are either variables $x_{[n]}$, constants $\mathcal{C} := \{1_{[n]} | n \in$

$$
\begin{array}{rll}
t_{[n]}[0:n-1] & \rightarrow t_{[n]} & \\
t[i:j][k:l] & \rightarrow t[k+i:l+i] & \\
(t_{[n]} \otimes u_{[m]})[i:j] & \rightarrow t_{[n]}[i:j] & \text{IF } j < n \\
 & \rightarrow u_{[m]}[i-n:j-n] & \text{IF } n \leq i \\
 & \rightarrow t_{[n]}[i:n-1] \otimes u_{[m]}[0:j-n] & \text{IF } i < n \leq j \\
 & & \\
c_{[n]} \otimes c_{[m]} & \rightarrow c_{[n+m]} & \\
x[i:j] \otimes x[j+1:k] & \rightarrow x[i:k] &
\end{array}
$$

Fig. 1. Canonizing Bitvecor Terms

$\mathbb{N}^+\} \cup \{0_{[n]} | n \in \mathbb{N}^+\}$ (bit-vectors filled with 1's and 0's, respectively), subfield extractions $t_{[n]}[i:j]$ of $j-i+1$ many bits i through j from bit-vector t, or concatenations $t \otimes u$; sometimes we use t^k, for $k \in \mathbb{N}^+$, to denote the k successive copies $t \otimes \ldots \otimes t$ of t. A bit-vector term $t_{[n]}$ is said to be *well-formed* if the corresponding set $wf(t_{[n]})$ of arithmetic constraints is satisfiable.

$$
\begin{aligned}
wf(p_{[n]}) &:= \{1 \leq n\}, \quad \text{if } p_{[n]} \text{ is a constant or a variable} \\
wf(t_{[n]} \otimes u_{[m]}) &:= wf(t_{[n]}) \cup wf(u_{[m]}) \\
wf(t_{[n]}[i:j]) &:= wf(t_{[n]}) \cup \{0 \leq i, i \leq j, j < n\}
\end{aligned}
$$

The set of well-formed terms of size n are collected in BV_n and BV denotes the union of all BV_n. Likewise, an equation $t_{[n]} = u_{[m]}$ is well-formed if

$$
wf(t_{[n]} = u_{[m]}) := wf(t_{[n]}) \cup wf(u_{[m]}) \cup \{n = m\}
$$

is satisfiable. Furthermore, $t \preceq u$ denotes the subterm relation on bit-vector terms, and the set of variables in term t (equation $t = u$) is denoted by $vars(t)$ ($vars(t = u)$).

Concatenation and extraction form a core set of operations that permits encoding other bit-vector operations like rotation or shift. Other extensions involve bitwise Boolean operations on bit-vectors (e.g. $t\ XOR\ u$) and finite arithmetic (e.g. addition modulo 2^n, $t +_{[n]} u$). Bit-vector terms of unknown size or terms with extraction at unknown positions are referred to as *non-fixed size* terms in the following.

Canonizing Bit-vector Terms. A bit-vector term t is called *atomic* if it is a variable or a constant, and *simple terms* are either atomic or of the form $x_{[n]}[i:j]$ where at least one of the inequalities $i \neq 0, j \neq n-1$ holds. Moreover, terms of the form $t_1 \otimes t_2 \otimes \ldots \otimes t_k$ (modulo associativity), where t_i are all *simple*, are referred to as being in *composition normal form.* If, in addition, none of the neighboring simple terms denote the same constant (modulo length) and a simple term of the form $t[i:j]$ is not followed by a simple term of the form $t[j+1:k]$, then a term in composition normal form is called *maximally connected.* Maximally connected terms are a *canonical form* for bit-vector terms. A canonical form for a term $t_{[n]}$, denoted by $\sigma(t_{[n]})$, is computed using the term rewrite system in Figure 1 (see [CMR97]).

Solvability. Fixed size bit-vector equations can readily be solved by comparing corresponding bits on the left-hand side and the right-hand side of the equation followed by propagating the resulting bit equations; such an algorithm is described, for example, in [CMR96]. This overly-eager bitwise splitting yields, not too surprisingly, exorbitant run times in many cases and the resulting solved forms are usually not succinct. The key feature of the solvers described below is to avoid case splits whenever possible.

3 Solving Fixed Size Bit-Vector Equations

In this section an equational transformation system for solving equations on fixed size bit-vector terms with concatenation and extraction is presented. Hereby, it is assumed that both the left-hand side and the right-hand side are canonized.

The rules in Figure 2 describe (conditional) transformations on sets of equations. They are of the form $E,\ \alpha \rightarrowtail E'$, where E, E' are sets of equations and α is a side condition. Such a rule is applicable if E matches a subset of the equations to be transformed and if the given side condition(s) are fulfilled. The formulation of the transformation rules in Figure 2 relies on the concept of chunks.

Definition 3. *A* chunk *is either a bit-vector variable $x_{[n]}$, an extraction from a bit-vector variable $x_{[n]}[i:j]$, or a constant $c_{[n]}$.*

In the sequel we use the convention that $c_{[n]}$, $d_{[m]}$ denote constants, $p_{[n]}$, $q_{[m]}$, $r_{[k]}$ denote chunks, and $s_{[n]}$, $t_{[m]}$, $u_{[k]}$ denote arbitrary bit-vector terms.

Rules (1)–(1") in Figure 2 split equations on terms with a chunk at the first position of a concatenation into several equations on subterms; recall from Section 2 that the canonizer σ computes maximally connected composition normal forms. Furthermore, rule (2) detects inconsistencies, rule (3) deletes trivial equations, and the remaining rules are mostly used to propagate equalities; $t[q/u]$ in rule (4) denotes replacement of all occurrences of q with u in t. Finally, rule (9) flips non-structural equations in order to make them applicable for further processing with rule (1"). Notice that none of the rules in Figure 2 introduces fresh variables.

Example 2. Let $x := x_{[16]}$, $y := y_{[8]}$, and $z := z_{[8]}$; then:

$$\begin{array}{ll} & \{x[0:7]\otimes y = y\otimes z,\ x = x[0:7]\otimes x[8:15]\} \\ \overset{(1)}{\rightarrowtail} & \{x[0:7] = y,\ y = z,\ x = x[0:7]\otimes x[8:15]\} \\ \overset{(5)}{\rightarrowtail} & \{x[0:7] = z,\ y = z,\ x = x[0:7]\otimes x[8:15]\} \\ \overset{(4)}{\rightarrowtail} & \{x[0:7] = z,\ y = z,\ x = z\otimes x[8:15]\} \end{array}$$

This derivation exemplifies the importance of so-called *structural* equations like $x = x[0:7]\otimes x[8:15]$. Intuitively, these equations are used to represent necessary splits of a variable. On the other hand, structural equations $p = s$ do not carry any semantic information, since $p \doteq \sigma(s)$.

$$
\begin{array}{lll}
(1) & p_{[n]}\otimes t = q_{[n]}\otimes u & \rightarrowtail \left\{ \begin{array}{l} p_{[n]} = q_{[n]} \\ t = u \end{array} \right\} \\
(1') & p_{[n]}\otimes t = q_{[m]}\otimes u,\ n < m & \rightarrowtail \left\{ \begin{array}{l} p_{[n]} = \sigma(q_{[m]}[0:n-1]) \\ \sigma(q_{[m]}[n:m-1])\otimes u = t \\ q_{[m]} = \sigma(q_{[m]}[0:n-1])\otimes\sigma(q_{[m]}[n:m-1]) \end{array} \right\} \\
(1'') & p_{[n]}\otimes t = q_{[m]} & \rightarrowtail \left\{ \begin{array}{l} p_{[n]} = \sigma(q_{[m]}[0:n-1]) \\ \sigma(q_{[m]}[n:m-1]) = t \\ q_{[m]} = \sigma(q_{[m]}[0:n-1])\otimes\sigma(q_{[m]}[n:m-1]) \end{array} \right\}
\end{array}
$$

$$
\begin{array}{llll}
(2) & c = d\ , & c \not\doteq d & \rightarrowtail \textbf{FAIL} \\
(3) & t = t & & \rightarrowtail \{\} \\
(4) & \left\{ \begin{array}{l} p = t \\ q = u \end{array} \right\}\ , & q \preceq t & \rightarrowtail \left\{ \begin{array}{l} p = t[q/u] \\ q = u \end{array} \right\} \\
(5) & \left\{ \begin{array}{l} p = q \\ q = r \end{array} \right\} & & \rightarrowtail \left\{ \begin{array}{l} p = r \\ q = r \end{array} \right\} \\
(6) & \left\{ \begin{array}{l} p = q \\ q = p \end{array} \right\} & & \rightarrowtail \{\ p = q\ \} \\
(7) & \left\{ \begin{array}{l} p = t \\ p = u \end{array} \right\} & & \rightarrowtail \left\{ \begin{array}{l} p = t \\ u = t \end{array} \right\} \\
(8) & c = t, & t \notin \mathcal{C} & \rightarrowtail \{\ t = c\ \} \\
(9) & p = q\otimes t, & p \not\doteq \sigma(q\otimes t) & \rightarrowtail \{\ q\otimes t = p\ \}
\end{array}
$$

Fig. 2. Equational Transformation System $\mathcal{C}_{\Re}$.

Definition 4. *Given an equation $t = u$, the set of* cuts *for $x_{[n]} \in vars(t = u)$, denoted by $cuts(x_{[n]})$, is defined as follows:*

$$cuts(x_{[n]}) := \{-1, n-1\} \cup \{i-1, j \mid x_{[n]}[i:j] \preceq t = u\}$$

Now, the set of structural equations *for $t = u$ is defined as:*

$$
\begin{aligned}
\mathcal{SE}(t = u) := \bigcup_{x \in vars(t=u)} \{\ & \sigma(x[i_0+1:i_k]) = x[i_0+1:i_1]\otimes\cdots\otimes x[i_{k-1}+1:i_k] \mid \\
& i_j \in cuts(x),\ i_j < i_{j+1} \\
& \textit{and there is no } i' \in cuts(x),\ i' \neq i_j \textit{ with } i_0 < i' < i_k\ \}
\end{aligned}
$$

In the Example 2 above, $cuts(x_{[16]}) = \{-1, 7, 15\}$ and therefore

$$\mathcal{SE}(t = u) = \{\ x = x[0:7]\otimes x[8:15],\ y = y[0:7],\ z = z[0:7]\ \}.$$

The last two equations in this set of structural equations can be discarded, since they do not represent proper splits. Notice also that, in the worst case, the cardinality of $\mathcal{SE}(.)$ may grow quadratically with the size of the bit-vectors involved. Some simple implementation techniques for handling this kind of blow-up are listed in [Möl98].

Definition 5. *Let $t = u$ be a bit-vector equation. A set of equations $\Upsilon_\perp$ is called a* solved set *for $t = u$ if: 1. $\Upsilon_\perp$ is equivalent with $\{t = u\}$. 2. no rule of $\mathcal{C}_\Re$ is applicable for $\Upsilon_\perp$. 3. for each $x \in vars(t = u) \cap \Upsilon_\perp$, $\Upsilon_\perp$ contains an equation of the form $x = s$.*

A solved set does not constitute a solved form in the sense of Definition 2, since some variable might occur on both sides of an equation. Given a solved set, however, it is straightforward to construct an equivalent solved form by omitting equations not of the form $x_i = u$ and replacing occurrences of extractions right-hand sides with fresh variables. Given the solved set $\{x[0:7] = z,\ y = z,\ x = z \otimes x[8:15]\}$ of Example 2, the set $\{x = z \otimes a,\ y = z\}$, where a is a fresh variable, is an equivalent solved form. This can be done in general.

Lemma 1. *For each solved set $\Upsilon_\perp$ of equations there is an equivalent solved form $\Upsilon'_\perp$.*

Now we are in the position to state a soundness and completeness result for the equational transformation system in Figure 2. This theorem, together with the construction for proving Lemma 1, determines a solver in the sense of Definition 2.

Theorem 1. *Let $t = u$ be an equation on fixed size, canonized bit-vector terms. Starting with the initial set of equations $\{t = u\} \cup \mathcal{SE}(t = u)$, the equational transformation system $\mathcal{C}_\Re$ in Figure 2 terminates with an (equivalent) solved set for $t = u$.*

It can easily be checked that the transformation rules (1)–(9) in Figure 2 are equivalence-preserving. The transformation process terminates, since, first, rules (1)–(1") decrease the lengths of bit-vector terms, second, rules (2),(3),(6) decrease the number of equations in the respective target sets, and third, the rules (4),(5),(7),(8) do not enlarge target sets and, together with rule (6), they construct a unique representation of every chunk. This process of equality propagation is terminating. Finally, rule (9) flips an equation as a preprocessing step for applying rule (1"); it cannot be applied repeatedly. The form of the rules together with the initial set $\mathcal{SE}(t = u)$ guarantees that the terminal set is a solved set. In particular, the set $\mathcal{SE}(t = u)$ includes for each (non-arbitrary) variable x in $vars(t = u)$ an equation of the form $x = s$ in order to match the third requirement of Definition 5.

4 Solving Bit-Vector Equations of Non-Fixed Size

In this section we develop a conceptual generalization of the equational transformation system $\mathcal{C}_\Re$ that is capable of solving equations on non-fixed size bit-vector terms like the word problems below.

Example 3.

$$x_{[n]} \otimes 0_{[1]} \otimes y_{[m]} = z_{[2]} \otimes 1_{[1]} \otimes w_{[2]} \qquad (1)$$

$$x_{[l]} \otimes 1_{[1]} \otimes 0_{[1]} = 1_{[1]} \otimes 0_{[1]} \otimes x_{[l]} \qquad (2)$$

$$
(1)\ \Big(\{p_{[n]}\otimes t = q_{[m]}\otimes u\}\cup E,\ \Psi\Big) \rightarrowtail \left[\begin{array}{l}
\left(\begin{array}{l}\left\{\begin{array}{l} p_{[n]} = \sigma(q_{[m]}[0:n-1]),\\ \sigma(q_{[m]}[n:m-1])\otimes u = t,\\ q_{[m]} = q_{[m]}[0:n-1]\otimes q_{[m]}[n:m-1]\end{array}\right\}\cup E,\\ \quad \{n<m\}\cup\Psi\end{array}\right)\\
\left(\begin{array}{l}\left\{\begin{array}{l} p_{[n]} = q_{[m]},\\ t = u\end{array}\right\}\cup E,\\ \quad \{n=m\}\cup\Psi\end{array}\right)\\
\left(\begin{array}{l}\left\{\begin{array}{l} q_{[m]} = \sigma(p_{[n]}[0:m-1]),\\ \sigma(p_{[n]}[m:n-1])\otimes t = u,\\ p_{[n]} = p_{[n]}[0:m-1]\otimes p_{[n]}[m:n-1]\end{array}\right\}\cup E,\\ \quad \{n>m\}\cup\Psi\end{array}\right)
\end{array}\right]
$$

$$
\begin{array}{c}(1)^*\ \Big(\{x_{[n]}[i:j] = x_{[n]}[k:l]\}\cup E, \Psi\Big) \\ \text{where } i<k\end{array} \rightarrowtail \left[\begin{array}{l}
\left(\begin{array}{l}\left\{x_{[n]}[i:l] = a_{[k-i]}^{\frac{l-i+1}{k-i}}\right\}\cup E,\\ \quad \{(l-i+1)\mid(k-i)\}\cup\Psi\end{array}\right)\\
\left(\begin{array}{l}\left\{x_{[n]}[i:l] = a_{[h]}\otimes(b_{[h']}\otimes a_{[h]})^{\frac{l-i-h+1}{k-i}}\right\}\cup E,\\ \quad \{(l-i+1)\nmid(k-i)\}\cup\Psi\end{array}\right)\\
\text{where } \begin{array}{l} h = (l-i+1)\mathrm{MOD}(k-i),\\ h' = k-i-h,\\ a,\, b \text{ fresh variables.}\end{array}
\end{array}\right]
$$

Fig. 3. Split Rules of $\mathcal{S}_{\Re}$.

Equation (1) is solvable if and only if $n = 1$, $m = 3$ or $n = 3, m = 1$, while Equation (2) is solvable if and only if l is even. Thus, both equations can not be solved in the strict sense of Definition 2. Instead, they motivate not only the need for representing all solutions as a disjunction of solved forms but also for integrating integer reasoning into the process of solving bit-vector equations.

Definition 6. *A* frame *is a pair (Υ,Ψ) consisting of a set of bit-vector equations Υ and a set Ψ of integer constraints of the form $n<m$, $n=m$, $n\mid m$ (divisibility), or $n \nmid m$ (non-divisibility) for $n,m \in \mathbb{N}^+$. Let V_{BV}, $V_{\mathbb{N}^+}$ be sets that include the bit-vector variables and the natural number variables of some frame (Υ,Ψ), respectively; then an* interpretation *of (Υ,Ψ) is a function $\mathcal{I} : V_{BV}\cup V_{\mathbb{N}^+} \to BV \cup \mathbb{N}^+$ such that $\mathcal{I}(x_{[n]}) \in BV_n$ and $\mathcal{I}(n)\in\mathbb{N}^+$. This determines notions like satisfiability or consistency of a frame in the usual way.*

A set of frames is called satisfiable *if there is at least one satisfiable frame and two sets of frames are* equivalent *if their sets of satisfying interpretations coincide. Furthermore, a set Ξ of frames is called* disjoint*, if for each (Υ_i,Ψ_i), $(\Upsilon_j,\Psi_j)\in\Xi$, $i\neq j$, the conjunction of their integer constraints is inconsistent; i.e., if $\Psi_i\wedge\Psi_j \models \bot$.*

$$
\begin{array}{llll}
(1'') & \left(\{p_{[n]}\otimes t = q_{[m]}\} \cup E,\ \Psi\right) & \rightarrowtail & \left(\left\{\begin{array}{l} p_{[n]} = \sigma(q_{[m]}[0:n\text{-}1]), \\ \sigma(q_{[m]}[n:m\text{-}1]) = t, \\ q_{[m]} = q_{[m]}[0:n\text{-}1]\otimes q_{[m]}[n:m\text{-}1] \end{array}\right\} \cup E,\ \Psi\right)
\end{array}
$$

$$
\begin{array}{llll}
(2) & \left(\{c_{[n]} = d_{[n]}\} \cup E,\ \Psi\right),\ c \not\equiv d & \rightarrowtail & \mathbf{FAIL} \\
(3) & (\{t = t\} \cup E,\ \Psi) & \rightarrowtail & (E,\ \Psi) \\
(4) & \left(\left\{\begin{array}{l} p = t \\ q = u \end{array}\right\} \cup E,\ \Psi\right),\ q \preceq t & \rightarrowtail & \left(\left\{\begin{array}{l} p = t[q/u] \\ q = u \end{array}\right\} \cup E,\ \Psi\right) \\
(5) & \left(\left\{\begin{array}{l} p = q \\ q = r \end{array}\right\} \cup E,\ \Psi\right),\ a \text{ fresh} & \rightarrowtail & \left(\left\{\begin{array}{l} p = a \\ q = a \\ r = a \end{array}\right\} \cup E,\ \Psi\right) \\
(6) & \left(\left\{\begin{array}{l} p = q \\ q = p \end{array}\right\} \cup E,\ \Psi\right),\ a \text{ fresh} & \rightarrowtail & \left(\left\{\begin{array}{l} p = a \\ q = a \end{array}\right\} \cup E,\ \Psi\right) \\
(7) & \left(\left\{\begin{array}{l} p = t \\ p = u \end{array}\right\} \cup E,\ \Psi\right) & \rightarrowtail & \left(\left\{\begin{array}{l} p = t \\ u = t \end{array}\right\} \cup E,\ \Psi\right) \\
(8) & (\{c = t\} \cup E,\ \Psi),\quad t \notin \mathcal{C} & \rightarrowtail & (\{t = c\} \cup E,\ \Psi) \\
(9) & (\{p = q\otimes t\} \cup E,\ \Psi),\quad p \not\equiv \sigma(q\otimes t) & \rightarrowtail & (\{q\otimes t = p\} \cup E,\ \Psi) \\
(10) & (\Upsilon,\ \Psi),\quad \Psi \models \bot & \rightarrowtail & \mathbf{FAIL}
\end{array}
$$

Fig. 4. Simple Rules of $\mathcal{S}_\Re$.

A frame transformation system called $\mathcal{S}_\Re$ is presented in Figures 3 and 4. The form of these rules is largely motivated by the equational transformation system in Figure 2 with the additional provision of case splits in rule (1) and (1)*. Application of rule (1), for example, replaces a source frame (Υ, Ψ) by three target frames (Υ_i, Ψ_i). It is easily seen that the property of disjointness is preserved by the frame transformation system $\mathcal{S}_\Re$. The rules (1")–(9) in Figure 4 only operate on sets of bit-vector equations and do not alter any integer constraints, while rule (10) is used to delete frames with inconsistent integer constraints. Hereby, the side condition $\Psi \models \bot$ of rule (10) is decidable. This can be shown by a simple reduction to the Diophantine problem for addition and divisibility that was proven decidable, for example, in [Lip78, Lip81].[1] Altogether, $\mathcal{S}_\Re$ yields $\{(\emptyset, \emptyset)\}$ for tautologies, whereas unsatisfiable formulae are eventually reduced to $\{\}$. Notice also that, in contrast to the rule system $\mathcal{C}_\Re$ in Figure 2, the frame transformation system $\mathcal{S}_\Re$ may introduce fresh variables a and b.

[1] The Diophantine problem for addition and divisibility is equivalent to the decidability of the class of formulas of the form $\exists x_1, \ldots, x_n. \wedge_{i=1}^{k} A_i$ in natural numbers, where the A_i have the form $x_m = x_j + x_k$, $x_m | x_j$, or $x_m = p$ and p is a natural number [DMV95].

Theorem 2 states a correctness result for this frame transformation system that can be proved similarly to Theorem 1; in contrast to Theorem 1, however, it does not imply termination of the transformation process.

Theorem 2. *Let $t = u$ be an equation on canonized and (possibly) non-fixed size bit-vector terms. Define the* initial frame (Υ_0, Ψ_0) *by*

$$(\{t = u\} \cup \mathcal{SE}(t = u),\ wf(t = u)).$$

If the process of applying rules $\mathcal{S}_{\Re}$ to the singleton set $\Xi_0 := \{(\Upsilon_0, \Psi_0)\}$ terminates with a set of frames Ξ_f, then:

1. *Ξ_f contains only solved forms (for $t = u$) in the sense of Definition 2.*
2. *Ξ_f is disjoint.*
3. *Ξ_0 and Ξ_f are equivalent.*

The rules in Figures 3, 4 and the overall structure of the solver may be best explained by means of an example. Consider, for example, solving the word problem (2) in Example 3. In this example we freely simplify integer constraints by omitting weakest constraints.

Example 4. There are no structural equations for

$$x_{[l]} \otimes \mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]} = \mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]} \otimes x_{[l]}$$

and the corresponding well-formedness constraints are given by the singleton set $\{1 \leq l\}$. Thus, solving starts with the initial frame

$$\Xi_0 ::= \left\{ (\{x_{[l]} \otimes \mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]} = \mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]} \otimes x_{[l]}\}, \{1 \leq l\}) \right\}.$$

Rule (1) in Figure 3 matches with the leftmost chunks $x_{[l]}$ and $\mathbf{1}_{[1]}$ of the equation e, and application of this rule yields the following three frames:

$$\Xi_1 ::= \left\{ \begin{array}{ll} (\{\ldots\}, & \{1 \leq l, l < 1\}), \\ (\{x_{[l]} = 1_{[1]},\ \mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]} = \mathbf{0}_{[1]} \otimes x_{[l]}\}, & \{1 \leq l, l = 1\}), \\ (\{x_{[l]}[0:0] = 1_{[1]},\ x_{[l]}[1:l\text{-}1] \otimes \mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]} = \mathbf{0}_{[1]} \otimes x_{[l]}\}, & \{1 \leq l, l > 1\}) \end{array} \right\}$$

The first frame is inconsistent and can be deleted via rule (10). Likewise, the second frame in Ξ_1 vanishes, since rule (2) eventually triggers for the equation

$$\mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]} = \mathbf{0}_{[1]} \otimes x_{[1]}.$$

It remains to process

$$(\Upsilon_3, \Psi_3) ::= (\{x_{[l]}[0:0] = 1_{[1]},\ x_{[l]}[1:l\text{-}1] \otimes \mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]} = \mathbf{0}_{[1]} \otimes x_{[l]}\}, \{l > 1\}).$$

In an attempt to match the second equation of Υ_3 with rule (1), three new frames $(\Upsilon_{31}, \Psi_{31})$, $(\Upsilon_{32}, \Psi_{32})$ and $(\Upsilon_{33}, \Psi_{33})$ are generated with additional constraints

$l < 2$, $l = 2$ and $l > 2$, respectively. Now, Ψ_{31} yields $false$ and $(\Upsilon_{32}, \Psi_{32})$ terminates, after simplifying the integer constraints, with the frame

$$\big(\{x_{[2]} = \mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]}\},\ \{l = 2\}\big).$$

In frame $(\Upsilon_{33}, \Psi_{33})$, the rule $(1)^*$ matches equation $x_{[l]}[0 : l-3] = x_{[l]}[2 : l-1]$. This yields two frames $(\Upsilon_{331}, \Psi_{331})$ and $(\Upsilon_{332}, \Psi_{332})$ such that l is required to be even in the first frame and odd in the second frame. In Υ_{331}, the equation $x_{[l]} = (a_{[2]})^{\frac{l}{2}}$ is added, thereby terminating with the frame

$$\big(x_{[l]} = (\mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]})^{\frac{l}{2}}, \{l > 2,\ 2 \mid l\}\big).$$

Since Υ_{332} contains the equation $x_{[l]} = b_{[1]} \otimes (d_{[1]} \otimes b_{[1]})^{\frac{l-1}{2}}$, together with the equations $x_{[l]}[0 : 0] = \mathbf{1}_{[1]}$ and $x_{[l]}[l-1 : l-1] = \mathbf{0}_{[1]}$ this frame reduces to $false$. Altogether, solving terminates with the two frames:

$$\left\{ \begin{array}{ll} (\{x_{[2]} = \mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]}\}, & \{l = 2\}) \\ (\{x_{[l]} = (\mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]})^{\frac{l}{2}}\}, & \{l > 2, 2 \mid l\}) \end{array} \right\}$$

Thus, the set of solutions is characterized by $x_{[l]} = (\mathbf{1}_{[1]} \otimes \mathbf{0}_{[1]})^{\frac{l}{2}}$ for all even l's greater or equal to 2. In special situations like the one above, a refinement of the solving algorithm could try to merge 'compatible' frames as soon as possible.

5 Conclusions

We have presented two specialized algorithms for solving equations on fixed size bit-vectors and for non-fixed size bit-vectors built up from concatenation and extraction operators. Both solvers adhere to the principle of splitting bit-vectors only on demand.

In the case of the fixed size bit-vector solver, this feature leads to moderate run times for large data paths. Even better, for some equations, solving time is independent of the size of data paths. However, the restriction to concatenation and extraction causes bit-wise splitting in situations where the regularity of solved forms cannot be expressed succinctly in terms of these operators only (for examples see [BDL98] or [BP98]). An extension of the fixed size solver with an iteration operator as proposed in [BP98] handles these cases well. More importantly, a useful bit-vector solver should support a rich set of operators including bitwise Boolean operators and finite arithmetic. We have demonstrated in [CMR97] that bitwise Boolean operators can be added in a conceptually clean way using the notion of *bit-vector OBDDs*. Again, splits are only performed on demand. OBDDs can also be used to encode finite arithmetic operations as Boolean functions. Using a ripple-carry adder for adding (modulo 2^n) two bit-vectors of length n in a naive way, however, yields overly-eager bitwise splitting and the resulting solver is, in our experience, not useful in practice. Barrett et al [BDL98] approach this problem by introducing a normal form geared to support arithmetic directly. In order to perform splits lazily, they introduce an overflow operator for ripple-carry additions. Their normalizer, however, does not possess the property of canonicity, and their algorithm does not seem to directly support bitwise Booleans operators.

The solver for non-fixed size bit-vector equations has been developed as a generalization of the algorithm for fixed size bit-vector equations. With a similar motivation in mind, Bjørner and Pichora [BP98]—independently—developed an algorithm for solving special cases of bit-vector equations of non-fixed size; their approach, however, is restricted to processing equations including one unknown size only, while our solver permits processing equations containing several unknowns. On the other hand, the solver in [BP98] is known to be terminating on the given fragment, while it is unknown if the algorithm described in Section 4 terminates for all input equations. If it is indeed terminating then it may be used to decide word equations [Mak92, PR98]. It has been shown, however, that any non-fixed size solver that supports a richer set of operators—as required for most hardware applications—is necessarily incomplete, since the halting problem can be reduced to solve non-fixed size equations on bit-vectors built up from concatenation, extraction, and bitwise Boolean operators only [Möl98].

Acknowledgements. We would like to thank Nikolaj Bjørner and David Cyrluk for their invaluable suggestions. In particular, Nikolaj pointed us to [Lip78]. Furthermore we thank the anonymous referees and Holger Pfeifer for their comments which helped to improve the presentation.

References

[BDL98] C.W. Barrett, D.L. Dill, and J.R. Levitt. A Decision Procedure for Bit-Vector Arithmetic. In *Proceedings of the 35th Design Automation Conference*, June 1998. San Francisco, CA.

[BP98] N.S. Bjørner and M.C. Pichora. Deciding Fixed and Non-Fixed Size Bit-Vectors. volume 1384 of *Lecture Notes in Computer Science LNCS*, pages 376–392, Heidelberg - New York - Berlin, April 1998. Springer.

[CLS96] D. Cyrluk, P. Lincoln, and N. Shankar. On Shostak's Decision Procedure for Combinations of Theories. In M. A. McRobbie and J. K. Slaney, editors, *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE-96)*, volume 1104 of *LNAI*, pages 463–477, Berlin;Heidelberg;New York, 1996. Springer.

[CMR96] D. Cyrluk, M.O. Möller, and H. Rueß. An Efficient Decision Procedure for a Theory of Fixed-Size Bitvectors with Composition and Extraction. Ulmer Informatik-Berichte 96-08, Fakultät für Informatik, Universität Ulm, 1996.

[CMR97] D. Cyrluk, M.O. Möller, and H. Rueß. An Efficient Decision Procedure for the Theory of Fixed-Sized Bit-Vectors. In O. Grumberg, editor, *Computer Aided Verification. 9th International Conference (CAV97). Haifa, Israel, June 22-25, 1997: Proceedings*, volume 1254 of *Lecture Notes in Computer Science LNCS*, pages 60–71, Berlin - Heidelberg - New York, 1997. Springer.

[DMV95] A. Degtyarev, Yu. Matiyasevich, and A. Voronkov. Simultaneous Rigid E-Unification is not so Simple. UPMAIL Technical Report 104, Uppsala University, Computing Science Department, 1995.

[Lip78] L. Lipshitz. The Diophantine Problem for Addition and Divisibility. *Transactions of the American Mathematical Society*, 235:271–283, January 1978.

[Lip81] L. Lipshitz. Some Remarks on the Diophantine Problem for Addition and Divisibility. *Bull. Soc. Math. Belg. Sér. B*, 33(1):41–52, 1981.

[Mak92] G. S. Makanin. Investigations on equations in a free group. In K.U. Schulz, editor, *Proceedings of Word Equations and Related Topics (IWWERT '90)*, volume 572 of *LNCS*, pages 1–11, Berlin;Heidelberg;New York, 1992. Springer.

[Möl98] M.O. Möller. Solving Bit-Vector Equations—A Decision Procedure for Hardware Verification, 1998. Diploma Thesis, available from `http://www.informatik.uni-ulm.de/ki/Bitvector/`.

[PR98] W. Plandowski and W. Rytter. Application of Lempel-Ziv Encodings to the Solution of Words Equations. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming, ICALP'98*, pages 731–742, July 1998. Aalborg, Denmark.

[Sho84] R.E. Shostak. Deciding Combinations of Theories. *Journal of the ACM*, 31(1):1–12, January 1984.

[SM95] M.K. Srivas and S.P. Miller. Formal Verification of the AAMP5 Microprocessor. In M.G. Hinchey and J.P. Bowen, editors, *Applications of Formal Methods*, International Series in Computer Science, chapter 7, pages 125–180. Prentice Hall, Hemel Hempstead, UK, 1995.