Deciding Fixed and Non-fixed Size Bit-vectors

Nikolaj S. Bjørner and Mark C. Pichora

Computer Science Department,
Stanford University, Stanford, California 94305
nikolaj|mpichora@cs.stanford.edu

Abstract. We develop a new, efficient, and compact decision procedure for fixed size bit-vectors with bit-wise boolean operations. The algorithm is designed such that it can also decide some common cases of parameterized (non-fixed) size. To handle even more parameterized cases for bit-vectors without bit-wise boolean operations we devise a unification based algorithm which invokes the first algorithm symbolically on parameters of the form $aN + b$, where a and b are integers and N is the only unknown.
Our procedures are designed to be integrated in the Shostak combination of decision procedures. This allows them to be tightly integrated with decision procedures for other theories in STeP's (the Stanford Temporal Prover) simplifier and validity checker.

1 Introduction

Bit-vectors are the natural data-type for hardware descriptions. To handle bit-vectors in computer-aided verification, it is convenient to have specialized decision procedures to solve constraints involving bit-vectors and their operations.

STeP [3], supports the computer-aided verification of reactive, real-time and hybrid systems. STeP provides the capability of verifying properties of parameterized systems with parameterized control and data domains. While model checking techniques are available for finite-state systems, deductive rules for linear-time temporal formulas are available for establishing correctness of very large finite-state and parameterized systems. STeP even supports a diagram-based deductive model checking procedure [18] which can verify infinite-state systems using STeP's deductive tools. The deductive verification methods are based on checking the validity of first-order verification conditions [4] which arise from applications of proof rules.

To verify industrial hardware designs, we are developing a compiler from the Verilog hardware description language to fair transition systems, which are STeP's computational model. Since bit-vectors are pervasive in Verilog we have found it useful to develop the decision procedures for bit-vectors described in

* This research was supported in part by the National Science Foundation under grant CCR-95-27927, the Defense Advanced Research Projects Agency under NASA grant NAG2-892, ARO under grant DAAH04-95-1-0317, ARO under MURI grant DAAH04-96-1-0341, and by Army contract DABT63-96-C-0096 (DARPA).

this paper. The presented procedure is easy to integrate tightly with decision procedures for other theories, which fits well into the wide scope of STeP.

1.1 Contributions

This work arose from the need for an efficiently supported theory of bit-vectors to prove basic verification conditions. An algorithm that addresses bit-vectors from a perspective similar to ours has been reported in [11]. In an effort to use that algorithm we found that it had to be extended in a nontrivial way to handle bit-wise boolean operations properly. In contrast, with our algorithm we offer a compact and efficient procedure that readily handles bitwise operations. The key feature of the procedure is that it only splits contiguous bit-vectors on demand. Its performance is often independent of the length of the bit-vectors in the input. We also discuss non-equational bit-vector constraints, which we think have not received proper attention elsewhere.

Legal inputs to the STeP-Verilog verification tool include *parameterized* hardware designs where the bit-vector size is not fixed at verification time. The need then arises for a method that can handle both fixed and non-fixed size bit-vectors. In certain cases our simple procedure for fixed size bit-vectors can be used directly for non-fixed size bit-vectors. To handle more cases, we first present an optimized decision procedure for equations $s = t$, where s and t do not contain bit-wise boolean operations, and then extend it to handle bit-vectors whose sizes are parameterized (still without supporting boolean operations). To our knowledge this is the first reported decision procedure that handles concatenation of a non-trivial class of non-fixed size bit-vectors.

1.2 Bit-vectors

Bit-vector terms are of the form

$$\begin{array}{lll} t & ::= & x \mid t[i:j] \mid t_1 \otimes t_2 \mid c_{[m]} \mid t_1 \; op \; t_2 \\ op & ::= & \& \text{ (bitwise and)} \mid \;\hat{}\; \text{ (bitwise xor)} \mid \text{“}|\text{” (bitwise or)} \\ c & ::= & 1 \mid 0 \end{array}$$

$t[i : j]$ denotes subfield extraction, and $\otimes$ concatenates two bit-vectors. The constant 0 is synonymous with false and 1 with true. For clarity, a term may be annotated by a length, such that $t_{[m]}$ indicates that t has length m.

Terms are well-formed when for every subterm $t_{[m]}[i : j]$, $0 \leq i \leq j < m$, and for every $s_{[m]}$ op $t_{[n]}$, $n = m$. Terms without occurrences of op are called ***basic bit-vector terms.***

Bit-vectors can be interpreted as finite functions from an initial segment of the natural numbers to booleans. Hence, if η is a mapping from bit-vector variables $x_{[m]}$ to an element of the function space $\{0, \ldots, m-1\} \longrightarrow \mathcal{B}$ we interpret composite terms as follows:

$$\begin{array}{ll}
[\![x]\!]_\eta & = \eta(x) \\
[\![t[i:j]]\!]_\eta & = \lambda k \in \{0,\ldots,j-i\}.[\![t]\!]_\eta(i+k) \\
[\![s_{[m]} \otimes t_{[n]}]\!]_\eta & = \lambda k \in \{0,\ldots,m{+}n{-}1\}.\textbf{if } k<m \textbf{ then } [\![s]\!]_\eta(k) \textbf{ else } [\![t]\!]_\eta(k{-}m) \\
[\![s_{[m]}\ op\ t]\!]_\eta & = \lambda k \in \{0,\ldots,m-1\}.[\![s]\!]_\eta(k)\ [\![op]\!]\ [\![t]\!]_\eta(k) \\
[\![c_{[m]}]\!]_\eta & = \lambda k \in \{0,\ldots,m-1\}.c=1
\end{array}$$

Bit-vector terms from the above grammar appear, for instance, throughout the system description and verification conditions from a split-transaction bus design from SUN Micro-systems [13]. A sample proof obligation encountered during STeP's verification of a safety property of the bus (processes are granted exclusive and non-interfering access to the bus) takes the form

$$\begin{array}{l}
l\_wires = 4 \ \wedge\ request \neq 0_{[8]} \ \rightarrow \\
\quad (request\_h \ \hat{}\ request) \neq 0_{[8]} \\
\quad \vee\ (request \neq 0_{[8]}) \wedge request = request\_h
\end{array} \tag{1}$$

where *request* and *request_h* are bit-vector variables of length 8. While this proof obligation is evidently valid, a simple encoding of bit-vectors as tuples causes examination of multiple branches when establishing the verification condition. The procedure developed here avoids this encoding and its potential case splitting. This and similar verification conditions can then be established independently of the bit-vector length (and in a fraction of a second). Thus, our procedure is able to establish this verification condition when the length 8 is replaced by an arbitrary parameter N.

While other logical operations like shifting can easily be encoded in the language of bit-vectors we analyze, the arithmetical (signed, unsigned and IEEE-compliant floating point) operations are not treated in an original way here.

1.3 Alternative approaches

This paper investigates decision methods for bit-vectors that can easily be integrated in the Shostak combination [17], which allows a tight integration of decision procedures for several theories. The strength of this approach is that verification conditions involving a mixture of bit-vector expressions, and uninterpreted function symbols, as in $x[1:6] = y[0:5] \ \rightarrow\ R(x[1:5]) = R(y[0:4])$, can be decided efficiently. Our algorithms can then be used in heterogeneous verification conditions. The Shostak combination is also targeted by the procedure in [11]. Naturally, the same verification conditions may be established directly using an axiomatization of bit-vectors and equality. Better than a raw axiomatization, proof assistants like ACL2 and PVS provide sophisticated libraries containing relevant bit-vector lemmas. Although highly useful, libraries do not provide a decision method.

In the symbolic model checking community, BDDs [6] (binary decision diagrams) are used to efficiently represent and reason about bit-vectors. Purely

BDD based representation of bit-vectors requires allocating one variable for every position in a bit-vector. (Just two bit-vector variables each of length 64 require allocation of 128 variables, pushing the limits of current BDD technology). A BMD-based (binary moment diagram) representation [7] optimizes on this while being able to efficiently perform arithmetical operations on bit-vectors. Unfortunately it is nontrivial to combine BMDs efficiently into the Shostak combination.

Since the values of bit-vectors range over strings of 0's and 1's it is possible to use regular automata to constrain the possible values of bit-vectors. Using this approach the MONA tool [2] can effectively represent addition of parameterized bit-vectors using M2L (Monadic Second-Order Logic). The expressive power of M2L also allows a direct and practical decision procedure of fixed size bit-vectors encoded either as tuples of boolean variables or as unary predicates with a constant domain. Furthermore M2L allows quantification over bit-vectors (with non-elementary complexity as the price). The approach based on regular automata, however does not admit an encoding of concatenations of parameterized bit-vectors. For suppose the regular language R_x (say 10^*1) encodes evaluations of bit-vector x that satisfy constraint $\varphi(x)$. Then the pumping lemma tells us that the evaluations of y consistent with $\varphi(x) \wedge y = x \otimes x$ is not in general (certainly $\{ww \mid w \in 10^*1\}$ is not) a regular language. Automata with constraints [8] (see chapter 4) is a possible remedy, but this imposes even more challenges in obtaining a direct ground integration with other decision procedures, which we seek here. Our procedure addresses this problem and solves satisfiability of ground equalities, a problem which is "only" NP-complete.

2 A decision procedure for fixed size bit-vectors

We present a normalization function $\mathcal{T}$, which takes a bit-vector term $t_{[m]}$ and a subrange (initially $[0 : m-1]$) and normalizes it to a bit-vector term $F_1 \otimes F_2 \otimes \ldots \otimes F_n$ where each F_i is of the form

$$F ::= F \; op \; F \mid x \mid c_{[m]} \;\; .$$

A normalization routine with a similar scope can be found in [5]. In words, $\mathcal{T}$ produces a term without occurrences of subfield extraction where every $\otimes$ is above every op. The translation furthermore maps every original variable $x_{[m]}$ to a concatenation $x_1 \otimes x_2 \otimes \ldots \otimes x_n$, and maintains a decoding of the auxiliary variables into subranges $decode([i_k : j_k])$, such that $i_1 = 0$, $j_n = m-1$, and $j_k + 1 = i_{k+1}$ for $k = 1 \ldots n-1$.

The normalization function shown in Figure 3 is designed to satisfy the basic correspondence

$$[\![t_{[n]}]\!]_\eta = [\![\mathcal{T}(t, [0 : n-1])]\!]_{\eta'}$$

for every η, where η' coincides with η on the free variables in t and, furthermore, if $\mathcal{T}$ rewrites x to $x_1 \otimes \ldots \otimes x_k \otimes \ldots \otimes x_n$, with $decode(x_k) = [i : j]$, then $\eta'(x_k) = \lambda k \in \{0, \ldots, j-i\}.\eta(x)(k+i)$.

Normalization works by recursive descent on the syntax tree of t, pushing a subfield extraction $[i : j]$ downwards. By maintaining only one copy of each variable, the procedure may update a variable occurrence x to a concatenation $x_1 \otimes x_2 \otimes x_3$ globally in the cases where only the subfield $[3 : 5]$ needs to be extracted from $x_{[8]}$. The result of normalizing $x[3 : 5]$ then becomes x_2, such that $decode(x_2) = [3 : 5]$. Since the variable x may occur in a different subterm under the scope of a boolean operator $x \;\&\; y$ the cutting of x rewrites this to $(x_1 \otimes x_2 \otimes x_3) \;\&\; y$. The auxiliary procedure *cut* (that takes a term and a cut-point as argument) shown in Figure 1 recursively cuts y in the same proportions as x, and forms the normalized concatenation $x_1 \;\&\; y_1 \;\otimes\; x_2 \;\&\; y_2 \;\otimes\; x_3 \;\&\; y_3$. It uses a set *parents* associated with each variable x to collect the maximal boolean subterms involving x that have already been normalized. Initially $parents(x) = \emptyset$ for each variable. Subterms can also be marked. By default (and initially) they are unmarked. To avoid cluttering the pseudocode we have suppressed variable dereferencing. To normalize boolean operators, $\mathcal{T}$ uses the auxiliary procedure *slice* shown in Figure 2, which aligns the normalized terms s and t into concatenations of equal length boolean subterms. Operator application can then be distributed over each of the equally sized portions. The auxiliary symbol ϵ is used for the empty concatenation.

The proper functioning of $\mathcal{T}$ relies on the precondition that every time $\mathcal{T}(t_{[n]}, [i : j])$ is invoked, then $0 \leq i \leq j < n$. This ensures that whenever $cut(t_{[n]}, m)$ is invoked then $m < n$.

```
cut(F, m) =
    mark(F);
    let
        (F_1(x̄^1), F_2(x̄^2)) = dice(F, m)
    in
        for each j = 1, 2, x^j ∈ x̄^j do
            parents(x^j) := parents(x^j) ∪ {F_j}
        return (F_1(x̄^1), F_2(x̄^2))
```

```
dice(s op t, m) =
    let
        (s_1, s_2) = dice(s, m);
        (t_1, t_2) = dice(t, m);
    in
        return (s_1 op t_1, s_2 op t_2)
dice(c_[l], m) = return (c_[m], c_[l-m])
dice(x^1_[m] ⊗ x^2_[n], m) = return (x^1, x^2)
dice(x, m) =
    let
        [i : j] = decode(x)
        x_1, x_2 be fresh variables with ∅ parents
    in
        decode(x_1) := [i : i + m - 1];
        decode(x_2) := [i + m : j];
        x := x_1 ⊗ x_2
        for each unmarked s ∈ parents(x) do
            s := s_1 ⊗ s_2 where (s_1, s_2) = cut(s, m)
        return (x_1, x_2)
```

Fig. 1. Basic cutting and dicing

Example: As an example of the translation of an bit-vector expression, consider:

$$\begin{array}{rl} s: & w_{[7]} \;\;\&\;\; (y_{[7]}[0:3] \;\otimes\; x_{[3]}) \\ t: & y_{[7]} \;\;|\;\; (x_{[3]} \;\otimes\; 1_{[1]} \;\otimes\; w_{[7]}[0:2]) \end{array}$$

We first apply $\mathcal{T}(s, [0 : 6])$ which results in cutting y into $y_1 \otimes y_2$, where $decode(y_1) = [0 : 3], decode(y_2) = [4 : 6]$. w is cut similarly. The translation

```
$apply(op, F(\overline{x}), G(\overline{y})) =$
    for each $x \in \overline{x} \cup \overline{y}$ do
        $parents(x) := parents(x) \setminus \{F, G\} \cup \{F(\overline{x})\ op\ G(\overline{y})\}$
    return $F(\overline{x})\ op\ G(\overline{y})$

$slice(op, \epsilon, \epsilon) = \epsilon$
$slice(op, F(\overline{x})_{[n]} \otimes s, G(\overline{y})_{[m]} \otimes t) =$
        if $m = n$ then
            $apply(op, F(\overline{x}), G(\overline{y})) \otimes slice(op, s, t)$
        else if $m > n$ then
            $(G_1(\overline{y}_1), G_2(\overline{y}_2)) := cut(G(\overline{y}), n);$
            $apply(op, F(\overline{x}), G_1(\overline{y}_1)) \otimes slice(op, s, G_2(\overline{y}_2) \otimes t)$
        else
            $(F_1(\overline{x}_1), F_2(\overline{x}_2)) := cut(F(\overline{y}), m);$
            $apply(op, F_1(\overline{x}), G(\overline{y})) \otimes slice(op, F_2(\overline{x}_2) \otimes s, t)$
```

Fig. 2. Slicing and operator application

$$
\begin{array}{ll}
\mathcal{T}(s\ op\ t, [i:j]) = & slice(op, \mathcal{T}(s,[i:j]), \mathcal{T}(t,[i:j])) \\
\mathcal{T}(s[k:l], [i:j]) = & \mathcal{T}(s, [k+i : k+j]) \\
\mathcal{T}(s_{[n]} \otimes t_{[m]}, [i:j]) = & \textbf{if } n \le i \textbf{ then } \mathcal{T}(t, [i-n : j-n]) \textbf{ else} \\
 & \textbf{if } n > j \textbf{ then } \mathcal{T}(s,[i:j]) \textbf{ else } \mathcal{T}(s,[i:n-1]) \otimes \mathcal{T}(t,[0:j-n]) \\
\mathcal{T}(x_{[m]}, [i:j]) = & \textbf{if } 0 < i \textbf{ then } \mathcal{T}(\text{second}(dice(x,i)), [0:j-i]) \textbf{ else} \\
 & \textbf{if } j < m-1\ (i=0) \textbf{ then } \text{first}(dice(x, j+1)) \textbf{ else } x \\
\mathcal{T}(c_{[m]}, [i:j]) = & c_{[j-i+1]}
\end{array}
$$

Fig. 3. Normalization function $\mathcal{T}$

of t results in further cutting y_1 into $y_3 \otimes y_4$, where $decode(y_3) = [0:2]$, in order to align with $x_{[3]} \otimes 1_{[1]}$. The variable $w_{[7]}$ is also cut into $w_1 \otimes w_2 \otimes w_3$ overing the same intervals as the parts of y, namely $[0:2], [3:3], [4:6]$. The result of translation is then:

$$s : w_1\ \&\ y_3 \otimes\ w_2\ \&\ y_4 \otimes\ w_3\ \&\ x \qquad t : y_3 \mid x \otimes\ y_4 \mid 1_{[1]} \otimes\ y_2 \mid w_1 \quad \blacksquare$$

2.1 Interfacing to the Shostak combination

Shostak's method of combining decision procedures allows integrating decision procedures for theories, such as arrays, linear arithmetic, records, inductive data-types, and sets inside Shostak's congruence closure algorithm [17, 10]. The method requires each decision procedure to provide (1) a canonizer (σ), which satisfies $\sigma(s) = \sigma(t)$ whenever the equality $s = t$ holds in the theory supported by the decision procedure; and (2) a solver, which rewrites an equation $s = t$ to either `false` (if it is unsatisfiable) or into an equivalent form $\exists V_{aux} \cdot \bigwedge_{i=1}^{n} x_i = t_i$, where each x_i is a variable from s or t, each t_i is canonized, and no x_i occurs in t_j or is equal to an x_k, when $k \neq i$. V_{aux} is the collection of auxiliary variables that occur in the t_j's but not in the original equation $s = t$. We will use the equivalent form as a substitution $\theta = [x_i \mapsto t_i \mid i = 1, \ldots, n]$ and call it a *Shostak* *substitution.* The substitution can be used to decide verification conditions of the form $s_1 = t_1 \wedge s_2 = t_2 \rightarrow s_3 = t_3$ by extracting θ_1 from $s_1 = t_1$, extracting θ_2 from $\theta_1(s_2 = t_2)$ and check if $\sigma(\theta_2(\theta_1(s_3)))$ is identical to $\sigma(\theta_2(\theta_1(t_3)))$. ·

To canonize a term $t_{[m]}$ we first obtain $F^1 \otimes \ldots \otimes F^n = \mathcal{T}(t, [0 : m-1])$. We will identify a free variable x_k in F^i with $x[i : j]$, where $decode(x_k) = [i : j]$. Each F^i is represented in a canonical form (for instance an ordered BDD) based on a total order of the variables. A consecutive pair F^i and F^{i+1} can now be combined whenever F^i is equivalent to the boolean expression obtained from F^{i+1} by replacing each variable $x[k : l]$ by $x[k-n : l-1]$, where n is the length of F^i.

To decide the satisfiability of an equality $s_{[n]} = t_{[n]}$ and extract a canonized substitution θ we notice that $s = t$ is equivalent to $s \,\hat{}\, t = 0_{[n]}$. Hence the equality is satisfiable if and only if $\mathcal{T}(s \,\hat{}\, t, [0 : n-1]) = F^1 \otimes \ldots \otimes F^m$ and $\bigwedge_{i=1}^m \neg F^i$ is satisfiable. At this point we can apply the technique used in [11], which extract equalities from BDDs using equivalence preserving transformations of the form $\mathbf{ite}(x, H, G) \equiv (H \vee G) \wedge \exists \delta . x = H \wedge (\neg G \vee \delta)$. This produces a substitution θ_0 with subranges of the original variables in the domain and auxiliary δ's in the range. The resulting Shostak substitution can then be extracted by generating θ as follows:

$$\theta_1 : [x \mapsto \theta_0(x_1) \otimes \ldots \otimes \theta_0(x_n) \mid x_i \in \mathbf{dom}(\theta_0) \;\wedge\; x = x_1 \otimes \ldots \otimes x_n]$$

$$\theta_2 : \left[x_k \mapsto x[i : j] \;\middle|\; \begin{array}{l} x = x_1 \otimes \ldots \otimes x_n,\; k \le n, \\ decode(x_k) = [i : j],\; \forall i : [1..n].x_i \notin \mathbf{dom}(\theta_0) \end{array} \right]$$

$$\theta : \;[x \mapsto \sigma(\theta_2(\theta_1(x))) \mid x \in \mathbf{dom}(\theta_1)]$$

Example: Continuing with the translated versions of our example terms s and t we will extract a Shostak substitution from the equality constraint $s = t$. We therefore complete the translation to get:

$$s \,\hat{}\, t : (w_1 \,\&\, y_3) \,\hat{}\, (y_3 \mid x) \otimes (w_2 \,\&\, y_4) \,\hat{}\, (y_4 \mid 1_{[1]}) \otimes (w_3 \,\&\, x) \,\hat{}\, (y_2 \mid w_1)$$

By negating the concatenations, the constraints needed to extract a substitution are obtained. The second constraint is easiest as it simply imposes $w_2 = y_4 = 1_{[1]}$. The conjunction of the first and third constraint is transformed:

$$\begin{array}{l}
\neg((w_1 \,\&\, y_3) \,\hat{}\, (y_3 \mid x)) \,\&\, \neg((w_3 \,\&\, x) \,\hat{}\, (y_2 \mid w_1)) = 1_{[3]} \\
\leftrightarrow \mathbf{ite}(x,\; w_1 \,\&\, y_3 \,\&\, w_3,\; \neg y_2 \,\&\, \neg w_1 \,\&\, \neg y_3) = 1_{[3]} \\
\leftrightarrow (x = w_1 \,\&\, y_3 \,\&\, w_3) \wedge ((w_1 \,\&\, y_3 \,\&\, w_3) \mid (\neg y_2 \,\&\, \neg w_1 \,\&\, \neg y_3)) = 1_{[3]} \\
\leftrightarrow (x = w_1 \,\&\, y_3 \,\&\, w_3) \wedge (w_1 = y_3 \,\&\, w_3) \wedge ((y_3 \,\&\, w_3) \mid (\neg y_2 \,\&\, \neg y_3)) = 1_{[3]} \\
\leftrightarrow (x = w_1 \,\&\, y_3 \,\&\, w_3) \wedge (w_1 = y_3 \,\&\, w_3) \wedge (y_3 = w_3) \wedge (w_3 \mid \neg y_2) = 1_{[3]} \\
\leftrightarrow (x = w_1 \,\&\, y_3 \,\&\, w_3) \wedge (w_1 = y_3 \,\&\, w_3) \wedge (y_3 = w_3) \wedge \exists \delta . y_2 = w_3 \,\&\, \delta
\end{array}$$

The composition of the extracted equalities gives an idempotent substitution:

$$\theta : [w_1 \mapsto w_3 \,\&\, \delta,\quad x \mapsto w_3 \,\&\, \delta,\quad y_2 \mapsto w_3 \,\&\, \delta,\quad y_3 \mapsto w_3]$$

From this we generate a Shostak substitution, where $V_{aux} = \{w_3, \delta\}$.

$$[x \mapsto w_3 \,\&\, \delta,\quad w \mapsto (w_3 \,\&\, \delta) \otimes 1_{[1]} \otimes w_3,\quad y \mapsto w_3 \otimes 1_{[1]} \otimes (w_3 \,\&\, \delta)] \;\;. \;\blacksquare$$

2.2 Equational running time

For input $s = t_{[n]}$ not involving *op* subterms (basic bit-vectors) the presented algorithm can be tuned to run in time: $\mathcal{O}(m + n\log(n))$, where m is the number of $\otimes$ and subfield extraction occurrences in s and t. First subfield extraction is pushed to the leaves in time $\mathcal{O}(m)$, then the $\otimes$ subterms are arranged in a balanced tree and $\mathcal{T}$ is applied to the balanced terms while maintaining balance in the tree. The translated equality $s = t$ is processed in a style similar to *slice*, but the auxiliary function *apply* has been replaced by one that builds a graph by connecting vertices representing the equated constants or variables. If some connected component contains two different constants there is a contradiction and the equality is unsatisfiable. Otherwise an equivalence class representative is appointed for each connected component, choosing a constant if one is present, or an arbitrary variable vertex otherwise. The extracted Shostak substitution then maps every variable to a concatenation of equivalence class representatives.

A canonized solution for satisfiable equalities can be extracted in time $\mathcal{O}(n)$ (which is dominated by the running time of $\mathcal{T}$). An algorithm with the same functionality is presented in [11]. That algorithm has running time $\mathcal{O}(m\log(m)+n^2)$, but offers some shortcuts that we don't address. Both procedures may still depend heavily on the parameter n. For instance, the equality

$$0_{[1]} \otimes 1_{[1]} \otimes x_{[m]} = x_{[m]} \otimes 0_{[1]} \otimes 1_{[1]} \tag{2}$$

requires (the maximal) m cuts of x, and is only satisfiable if m is even. The same functionality can, as [11] noticed, be achieved in $\mathcal{O}(m + n)$ time, but at the expense of having this as the minimal running time as well.

Another advantage of our algorithm is that it can be extended (with a few modifications) to the case where bit-vectors of parameterized length are either exclusively on the right or exclusively on the left of every concatenation. This excludes cases like (2), which we will address in Section 3.

2.3 Beyond Equalities

The satisfiability problem for constraints involving disequalities is NP-complete in the case of basic bit-vectors. Membership in NP follows from the fact that we can easily check in polynomial time that a given instantiation of bit-vector variables satisfies prescribed constraints. NP-hardness follows from a reduction from 3-SAT to conjunctions of disequality constraints: take an instance of 3-SAT $\bigwedge_i(l_i \vee k_i \vee m_i)$ where l_i, k_i and m_i are literals over the vocabulary $\mathcal{V}$ of boolean variables. Translate this into $\bigwedge_i(l_i \otimes k_i \otimes m_i \neq 000) \wedge \bigwedge_{x\in\mathcal{V}}(\overline{x} \neq x)$, where for each boolean variable x we associate two bit-vector variables $x_{[1]}$ representing x and $\overline{x}_{[1]}$ representing the negation of x.

We therefore settle here by handling $t \neq s$ as $|(t\ \hat{}\ s)$, and converting $|t_{[n]}$ to $t[0:0] \mid \ldots \mid t[n-1:n-1] = 1_{[1]}$. The connectives $<$ and $\leq$, as well as operations like $+$ and $*$ can be handled similarly, though the advantages of this approach are questionable. Naturally these constraints are only analyzed when all equational constraints have been processed and the resulting Shostak substitutions have been applied to the non-equational constraints.

Verification conditions of the form $f(A) \neq f(B) \wedge f(A) \neq f(C) \rightarrow f(B) = f(C)$, where f is an uninterpreted function symbol, are handled using a complete case analysis on bit-vectors A, B and C (it is valid only when A, B and C are bit-vectors of length 1). Shostak's approach to combining equational theories misses cases like this as it is originally designed for theories admitting infinite models (see for example [15]).

3 Unification of basic bit-vectors

In this section we focus on the problem of finding unifiers for basic bit-vector terms s and t. The restriction to basic bit-vector terms allows us to develop a more efficient procedure and at the same time widen its scope to bit-vectors whose lengths are parameterized.

3.1 *ext*-terms

To more compactly represent solutions to equations like (2) we introduce a new bit-vector term construct $ext(t_{[n]}, m)$ (the extension of t up to length m), which is well-formed whenever $m > 0$. The meaning of ext is given by the equation

$$[\![ext(t_{[n]}, m)]\!]_\eta = [\![\underbrace{t \otimes .. \otimes t}_{k} \otimes t[0 : l{-}1]]\!]_\eta \text{ where } (k{+}1)n \geq m > kn \text{ and } l = m{-}kn$$

Thus, $ext(t_{[n]}, m)$ repeats t up to the length m. To map ext-terms to terms in the base language we use the unfolding function unf

$$unf(t_{[n]}, m) = \underbrace{t \otimes .. \otimes t}_{k} \otimes \mathcal{T}(t, [0 : l{-}1]) \text{ where } (k{+}1)n \geq m > kn \text{ and } l = m{-}kn$$

A solution to equation (2) can now be given compactly when m is even as $x = ext(0_{[1]} \otimes 1_{[1]}, m)$.

3.2 Unification with *ext*-terms

To decide the satisfiability of equalities $s = t$ of basic bit-vector terms extended with ext-subterms we will develop a Martelli-Montanari style unification algorithm [14] which takes the singleton set $\mathcal{E}_0 : \{s = t\}$ as input and works by transforming $\mathcal{E}_0$ to intermediary sets $\mathcal{E}_1, \mathcal{E}_2, \ldots$ by equivalence preserving transformations which simplify, delete or propagate equalities. It ultimately produces either FAIL, when $s = t$ is unsatisfiable, or a substitution $\mathcal{E}_{final} : \{x_1 = t_1, \ldots, x_n = t_n\}$.

Since our procedure uses $\mathcal{T}$ to decompose terms, every auxiliary variable in $\mathcal{E}_{final}$ furthermore corresponds to a unique disjoint subrange of one of the original variables. The obviously satisfiable conjunction of equalities is equivalent to the original equality.

A canonizer can be obtained by first eliminating the *ext*-terms by using unfold and then using the canonizer of Section 2.1.

Example: Anticipating the algorithm we will present, consider the following equality assertion:

$$y_{[3]} \otimes x_{[16]} \otimes x_{[16]} \otimes z_{[2]} = x_{[16]} \otimes w_{[4]} \otimes 0_{[1]} \otimes x_{[16]} \ .$$

In processing the implied equality $y_{[3]} \otimes x_{[16]} = x_{[16]} \otimes \dots$ we obtain $x_{[16]} = ext(y_{[3]}, 16)$ as a solution for $x_{[16]}$. Continuing with the remaining equalities we get the intermediate set of equations:

$$\begin{array}{llll} x_{[16]} = ext(y_{[3]}, 16), & y_{[3]}[1:2] \otimes y_{[3]}[0:0] & = w_{[4]}[0:2], \\ z_{[2]} \ = w_{[4]}[3:3] \otimes 0_{[1]}, & ext(w_{[4]}[3:3] \otimes 0_{[1]}, 16) = x_{[16]} \ . \end{array}$$

The two equations involving x are combined to produce the implied constraint

$$ext(y_{[3]}, 16) = ext(w_{[4]}[3:3] \otimes 0_{[1]}, 16) \ .$$

This equality is evidently equivalent to its *unf*-unfolding, but as we will later formulated in a general setting, we can do better and only need to assert:

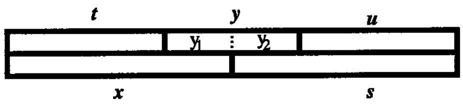
$$y_{[3]} \otimes y_{[3]}[0:0] = w_{[4]}[3:3] \otimes 0_{[1]} \otimes w_{[4]}[3:3] \otimes 0_{[1]} \ .$$

In fact this implies $y[0:0] = y[1:1] = y[2:2] = w[3:3] = 0_{[1]}$. After propagating the resulting constraints we obtain the final result:

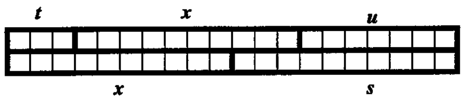
$$w_{[4]} = 0_{[4]}, \quad x_{[16]} = 0_{[16]}, \quad y_{[3]} = 0_{[3]}, \quad z_{[2]} = 0_{[2]} \ . \ ■$$

While the full unification algorithm is given in Figure 4 we highlight and explain the more delicate cases below.

$x_{[n]} \otimes s = t_{[m]} \otimes y_{[l]} \otimes u$ when $m+l > n > m$, $x \not\equiv y$. The situation is described in the picture below, which suggests that the equality is equivalent to $x = t \otimes y_1$ and $s = y_2 \otimes u$ for suitable splits y_1 and y_2 of y. We use $\mathcal{T}$ to cut y into the appropriate pieces. This replaces y everywhere in $\mathcal{E}$ by $y_1 \otimes y_2$.



$x_{[n]} \otimes s = t_{[m]} \otimes x_{[n]} \otimes u$ when $n > m$. For example we are given the configuration:



Thus, the original equality constraint is equivalent to $x = t \otimes t \otimes t \otimes t[0:0]$ and $t[1:2] \otimes t[0:0] \otimes u = s$. To more compactly describe the first equality we use the *ext*-construct to obtain $x = ext(t, 10)$.

$ext(s_{[m]}, l) = ext(t_{[n]}, l)$ The effect of replacing x by s in the variable elimination step may introduce equality constraints between *ext*-terms. Although the equality constraint is by definition equivalent to $unf(s_{[m]}, l) = unf(t_{[n]}, l)$, we can be even more economical in the unfolding as the following lemma suggests.

Lemma 1. *Assume $2n \le l$ and $2m \le l$ and let $g = \gcd(m, n)$ then*

$$ext(s_{[m]}, l) = ext(t_{[n]}, l) \quad \leftrightarrow \quad unf(s, m+n-g) = unf(t, m+n-g)$$

Thus, we will ensure that our algorithm maintains the invariant $2n \le l$ for every $ext(t_{[n]}, l)$ term, and the equality constraint $ext(s_{[m]}, l) = ext(t_{[n]}, l)$ is replaced by $unf(s, m+n-g) = unf(t, m+n-g)$.

Other simpler cases are summarized in Figure 4. It omits cases that can be obtained using symmetry of equality.

Constructor elimination

R1 $\{s_{[m]} \otimes u = t_{[m]} \otimes v\} \cup \mathcal{E} \;\rightarrow\; \{s = t, u = v\} \cup \mathcal{E}$

R2 $\{c_{[m]} \otimes s = c'_{[n]} \otimes t\} \cup \mathcal{E} \;\rightarrow\;$ FAIL **where** $c \neq c'$

R3 $\{c_{[m]} \otimes s = c_{[n]} \otimes t\} \cup \mathcal{E} \;\rightarrow\; \{s = c_{[n-m]} \otimes t\} \cup \mathcal{E}$ **where** $n > m$

R4 $\{x_{[n]} \otimes s = t_{[m]} \otimes y_{[l]} \otimes u\} \cup \mathcal{E} \;\rightarrow\; \{x = t \otimes y_1, s = y_2 \otimes u\} \cup \mathcal{E}$
where $m + l > n > m > 0$, $x \neq y$,
$y_1 = \mathcal{T}(y, [0 : m-n-1])$, $y_2 = \mathcal{T}(y, [m-n : l-1])$,

R5 $\{x_{[n]} \otimes s = t_{[m]} \otimes x_{[n]} \otimes u\} \cup \mathcal{E} \;\rightarrow\; \{s = wrap(t, n) \otimes u, x = mk\text{-}ext(t, n)\} \cup \mathcal{E}$
where $n > m > 0$

R6 $\{x_{[n]} \otimes s = t_{[m]} \otimes c_{[l]} \otimes u\} \cup \mathcal{E} \;\rightarrow\; \{x = t \otimes c_{[m-n]}, s = c_{[l+n-m]} \otimes u\} \cup \mathcal{E}$
where $m + l > n > m \ge 0$

R7 $\{s_{[m]} \otimes t = ext(u_{[l]}, n) \otimes v\} \cup \mathcal{E} \;\rightarrow\; \left\{\begin{array}{l} s_{[m]} = mk\text{-}ext(u, m), \\ t = mk\text{-}ext(wrap(u, m), n - m) \otimes v \end{array}\right\} \cup \mathcal{E}$
where $m < n$

R8 $\{ext(s_{[l_1]}, m) = ext(t_{[l_2]}, m)\} \cup \mathcal{E} \rightarrow \{unf(s, l) = unf(t, l)\} \cup \mathcal{E}$
where $l = l_1 + l_2 - \gcd(l_1, l_2)$

Equality and variable elimination

R9 $\{t = t\} \cup \mathcal{E} \;\rightarrow\; \mathcal{E}$

R10 $\{x = s\} \cup \mathcal{E} \;\rightarrow\; \{x = s\} \cup \mathcal{E}[x \mapsto s]$

Fig. 4. Rules for unification with *ext*-terms

The auxiliary function *wrap* splits the term t at position k and swaps the two pieces. The function *mk-ext* produces either an *ext*-term when the length of t is sufficiently small or unfolds t. It ensures that every $ext(t_{[n]}, m)$ term generated by the algorithm satisfies $2n \leq m$. These are defined more precisely below:

$$\begin{aligned} wrap(t_{[n]}, m) = &\ \textbf{let}\ \ k = m \bmod n\ \textbf{in} \\ &\ \textbf{if}\ \ k = 0\ \textbf{then}\ t\ \textbf{else}\ \mathcal{T}(t, [k : n-1]) \otimes \mathcal{T}(t, [0 : k-1]) \end{aligned}$$

$$mk\text{-}ext(t_{[n]}, m) = \textbf{if}\ \ 2n \leq m\ \textbf{then}\ \ ext(t, m)\ \textbf{else}\ \ unf(t, m)$$

The unification algorithm terminates since the variable elimination step removes duplicate constraints involving x and every other step produces equalities of smaller size (in terms of the number of bitwise comparisons) than the one eliminated. For instance, in the R8 rule we rely on $m \geq 2 \cdot \max(l_1, l_2) > l$.

3.3 Nonfixed size bit-vectors

The most prominent feature of the unification algorithm in Figure 4 is that it can be used to decide bit-vector equality constraints $s = t$, where lengths and projections are not restricted to fixed naturals, but are of the form $aN + b$, where a and b are integers and N is a parameter (where we assume without loss of generality that $N > 0$). This allows us to apply the algorithm in the Shostak combination for deciding verification conditions with non-fixed bit-vector equalities. The unification problem for non-fixed bit-vectors is also reminiscent of the word unification problem, originally solved by Makanin and later solved using a unification procedure in [12]. The main difference is that variables ranging over words in that problem do not have associated size constraints which bit-vectors have. By performing comparisons and arithmetic on these lengths symbolically and allowing admissible answers to be paired with accumulated constraints (as explained later), we can deal with the following example:

Example: By performing the unification of

$$\{w_{[2]} \otimes 0_{[1]} \otimes x_{[N+6]} \otimes y_{[N+7]} = x_{[N+6]} \otimes 1_{[1]} \otimes z_{[3]} \otimes x_{[N+6]}\} \tag{3}$$

we obtain as an intermediate step

$$\left\{ \begin{array}{rl} x_{[N+6]} = ext(w_{[2]} \otimes 0_{[1]}, N+6), & y_{[N+7]} = z_{[3]}[2:2] \otimes x_{[N+6]} \\ 1_{[1]} \otimes z_{[3]}[0:1] = wrap(w_{[2]} \otimes 0_{[1]}, N+6) & \end{array} \right\}$$

and finally two cases:

$$\left.\begin{aligned} x_{[N+6]} &= ext(1_{[1]} \otimes \beta_{[1]} \otimes 0_{[1]}, N+6), \\ y_{[N+7]} &= \alpha_{[1]} \otimes ext(1_{[1]} \otimes \beta_{[1]} \otimes 0_{[1]}, N+6), \\ z_{[3]} &= \beta_{[1]} \otimes 0_{[1]} \otimes \alpha_{[1]}, \\ w_{[2]} &= 1_{[1]} \otimes \beta_{[1]} \end{aligned}\right. \quad \text{if } N \equiv 0 \pmod 3$$

$$\left.\begin{aligned} x_{[N+6]} &= ext(\beta_{[1]} \otimes 1_{[1]} \otimes 0_{[1]}, N+6), \\ y_{[N+7]} &= \alpha_{[1]} \otimes ext(\beta_{[1]} \otimes 1_{[1]} \otimes 0_{[1]}, N+6), \\ z_{[3]} &= 0_{[1]} \otimes \beta_{[1]} \otimes \alpha_{[1]}, \\ w_{[2]} &= \beta_{[1]} \otimes 1_{[1]} \end{aligned}\right. \quad \text{if } N \equiv 1 \pmod 3$$

When $N \equiv 0 \pmod 3$, the evaluation of the *wrap* function simplifies the second equation of the intermediate result to $1_{[1]} \otimes z_{[3]}[0:1] = w_{[2]} \otimes 0_{[1]}$. The case that corresponds to $N \equiv 2 \pmod 3$ requires $1_{[1]} \otimes z_{[3]}[0:1] = 0_{[1]} \otimes w_{[2]}$ which results in an inconsistency. The $\beta_{[1]}, \alpha_{[1]}$ are auxiliary variables that are introduced to represent unknown segments of the bit-vector variables. ∎

Thus, the result produced by the unification algorithm will now be a *set* of constraints, each of the form

$$(ax + b > c, [N \mapsto ax + b] \circ [x_i \mapsto t_i \mid i = 1, \ldots, n])$$

where x is a fresh variable and the first constraint is passed on to decision procedures for linear arithmetic, and the second constraint is a Shostak substitution. We are thus faced with a finitary as opposed to unitary unification problem (see [1] for a survey on unification theory).

The crucial observation that allows lifting the algorithm to parameterized bit-vector expressions is that all operations and tests on the lengths and projections are of the form

$$m + n, \quad m - n, \quad m > n, \quad m \geq n, \quad m = n, \quad m \bmod n.$$

Since terms of the form $aN + b$ are closed under addition and subtraction, the first two operations can be performed directly in a symbolic way.

The comparison $m > n$ is rewritten to $m - n > 0$, $m \geq n$ to $m - n + 1 > 0$, and $n = m$ to $n - m + 1 > 0 \;\wedge\; m - n + 1 > 0$. This reduces the evaluation of comparisons to $aN + b > 0$. Since

$$\begin{aligned} aN + b > 0 \;\leftrightarrow\; (a = 0 \wedge b > 0 \;\vee\; a > 0) \;\text{ iff} \\ (a > 0 > b \vee a < 0 < b) \;\rightarrow N \geq |b| \text{ div } |a| \end{aligned} \tag{4}$$

tests can be evaluated using $a = 0 \wedge b > 0 \;\vee\; a > 0$ and accumulating auxiliary lower bounds on N for a separate treatment. Our algorithm then produces answers for all N greater than the largest accumulated lower bound. For values of N smaller than the accumulated bounds we instantiate N and run the fixed size version.

The auxiliary function *wrap* requires us to compute m mod n. To simplify this case our algorithm will maintain the invariant that m mod n is only invoked when n is a constant b', whereas m may be of the form $N+b$. The case $N \geq b' - b$ causes case-splitting on each of the possible solutions $k = 0, \ldots, b' - 1$.

We could represent each case in Presburger arithmetic as $\exists x \geq 0 . xb' = N + b - k$ and use a Presburger decision procedure [9] to check satisfiability of conjunctions of such constraints. However, in order to manage these constraints more efficiently we can use the Chinese Remainder Theorem (see [16]). If $\Pi_i p_i^{\alpha_i}$ is a prime factorization of b' (with $p_1, p_2, \ldots$ the sequence of all primes), then

$$N + b \equiv k \pmod{b'} \quad \text{iff} \quad N + b \equiv k \pmod{p_i^{\alpha_i}} \text{ for every } i.$$

Let $D(p, \beta, l)$ be the predicate that $N \equiv l \pmod{p^\beta}$ is true. Let $\mathcal{C}_{\text{mod}} = \bigwedge_i D(p_i, \beta_i, b_i)$ be the conjunction of divisibility constraints imposed on the current system. Only one predicate is needed for each p_i, since:

$$D(p, \beta, l') \wedge D(p, \alpha, l) \wedge \beta \geq \alpha \quad \text{iff} \quad D(p, \beta, l') \wedge l' \equiv l \pmod{p^\alpha} \ . \qquad (5)$$

In order to split on the case $N + b \equiv k \pmod{b'}$ for different values of $k = 0, \ldots, (b' - 1)$ we can form the product of the case splits on $N + b \equiv k_i \pmod{p_i^{\alpha_i}}$ for $k_i = 0, \ldots, (p_i^{\alpha_i} - 1)$ (the product is over $i = 1, 2, \ldots$). The situation is not as bad as it seems, since we can use the existing $\mathcal{C}_{\text{mod}}$ to merge the new constraints in an optimal way:

$$\mathcal{C}'_{\text{mod}} = \bigwedge_i P(i) \quad \text{where} \quad P(i) = \begin{cases} \bigvee_{j=0}^{p_i^{\alpha_i - \beta_i} - 1} D(p_i, \alpha_i, b_i + j p_i^{\beta_i}) & \text{if } \alpha_i \geq \beta_i \\ D(p_i, \beta_i, b_i) & \text{if } \alpha_i < \beta_i \end{cases}$$

The predicate $P(i)$ represents the enumeration of valid congruences modulo a power of p_i. Statement (5) suggests the form of the enumeration for each case in the definition of $P(i)$. Expressing $\mathcal{C}'_{\text{mod}}$ in disjunctive normal form $\bigvee_i \mathcal{C}^i_{\text{mod}}$ the constraints for the different cases are obtained. The value of k for a particular case of $\mathcal{C}_{\text{mod}}$ can be reconstructed using the congruence $k \equiv (\sum_i n_i b_i) - b \pmod{b'}$ where $n_i = z_i \overline{z}_i$, $z_i = \Pi_{j \neq i} p_j^{\alpha_i}$, and $\overline{z}_i$ satisfies $z_i \overline{z}_i \equiv 1 \pmod{p_i^{\alpha_i}}$ (it exists since $\gcd(p_i^{\alpha_i}, z_i) = 1$).

Given expressions s and t our algorithm now engages in the following steps: (1) Apply $\mathcal{T}$ to both s and t, i.e., let $(s, t) := (\mathcal{T}(s, [0 : m-1]), \mathcal{T}(t, [0 : m-1]))$. This generates bit-vector expressions without subfield extraction and an assignment to each original variable x to a concatenation $x_1 \otimes x_2 \otimes \ldots \otimes x_n$ of distinct variables, where $decode(x_i)$ cover disjoint intervals of x. Using equivalence (4) the tests in $\mathcal{T}$ are evaluated unambiguously, and possibly generating a new lower bound on N. The cases where N is smaller than this bound are processed later. (2) Every variable $x_{[aN+b]}$ remaining in s or t, where $a > 0$, is replaced by a concatenation of a fresh variables: $x^{(1)}_{[N]} \otimes x^{(2)}_{[N]} \otimes \ldots \otimes x^{(a)}_{[N+b]}$. Constants are cut in a similar way[2]. If b is negative the lower bound $1 - b$ on N is added.

Every variable occurring in s and t now has length $N + k$ or k, where k is an integer. (3) The algorithm in Figure 4 is invoked on the equality $\{s = t\}$. Each comparison accumulates a lower bound on N and each invocation of mod may cause a multi-way case split while accumulating modulus constraints on N. The unification algorithm therefore generates constraints of the form $(\mathcal{E}_1, \mathcal{C}_1), \ldots, (\mathcal{E}_n, \mathcal{C}_n)$, where the $\mathcal{E}_i$ are equalities and $\mathcal{C}_i$ is a conjunction of $N \geq k$ and $D(p_i, \alpha_i, a_i)$ constraints.

We need to ensure that every step is well defined: in particular that $unf(t, m)$ and, as we assumed, $n \bmod m$ are only invoked when m is a constant. This is a consequence of the following invariant:

[2] This step is not strictly necessary, but simplifies the further presentation of the algorithm.

Invariant 2 *For every occurrence of $ext(t_{[aN+b]}, n)$: $a = 0 \wedge 2b \leq n$.*

This holds as *ext* terms are only generated when $mk\text{-}ext(t_{[aN+b]}, a'N + b')$ is invoked and $2(aN + b) \leq a'N + b'$. Since both a and a' are either 0 or 1, this inequality can only hold if $a = 0$ or N is bounded above by $(b' - 2b)$ div $(2a - a')$. The cases where N is bounded above by a constant are treated separately.
(4) The solved form can now be extracted. For each $(\mathcal{E}, \mathcal{C})$ generated from the previous step let $\mathcal{C}$ be of the form $N \geq k \wedge \bigwedge_{i=1}^{l} D(p_i, \alpha_i, a_i)$. The Chinese Remainder Theorem tells us how to find n_i such that the constraints can be rewritten to the equivalent form

$$N \geq k \wedge \exists x. N = Ax + B \text{ where } A = \prod_{i=1}^{l} p_i^{\alpha_i} \; B = \left(\sum_{i=1}^{l} n_i a_i\right) \bmod A$$

Since we extract the Shostak substitution θ from $\mathcal{E}$ as in the fixed-length case the combined constraint returned for this case is

$$(Ax + B \geq k, [N \mapsto Ax + B] \circ \theta).$$

For each k less than the least lower bound accumulated above we instantiate N by k and extract θ_k by running the fixed-size version of the algorithm (that is, running $\{s = t\}[N \mapsto k]$). For these cases the returned constraints have the form

$$(\mathtt{true}, [N \mapsto k] \circ \theta_k).$$

The algorithm now concludes by returning the entire set of the constraints extracted above.

As we have argued above we now have

Theorem 3. *(Correctness) When the non-fixed unification algorithm terminates on the input constraint $s = t$ with a set of constraints $\{(\varphi_i(x), \theta_i) \mid i = 0, \ldots n\}$ then* $s = t \quad \leftrightarrow \quad \bigvee_{i=0}^{n} \exists x, V_{\text{aux}}. \varphi_i(x) \wedge \theta_i$.

Finally we must ensure that we can make the unification algorithm modified for parameterized lengths terminate. To this end we apply the transformation rules from Figure 4 by preferring the variable and equality elimination rules to the other rules.

We will proceed to prove the termination by induction on the number of distinct non-fixed variables k in $\mathcal{E}$ that participate in some equality where rule R1-R8 can be applied. The base case ($k = 0$) operates only on fixed-size variables, and so it terminates.

Whenever a variable x has been isolated using one of the rules R4-R6, it is eliminated from the rest of $\mathcal{E}$. Indeed it is eliminated as x cannot be a proper subterm of t in the equality constraint $x = t$, since the length of t is the sum of the lengths of its variable and constant subterms, which equals the length of x. Since rules R1-R8 produce equalities between smaller bit-vectors we cannot repeatedly apply these rules without eventually eliminating a non-fixed size variable. Rule R4 may split a non-fixed length variable y into two parts y_1 and y_2, but only one of these parts will have non-fixed length, so the overall number of non-fixed length variables is constant.

We therefore have

Theorem 4. *(Termination) The non-fixed unification algorithm terminates.*

A reduction from the problem of simultaneous incongruences [19] can establish that the unification problem for non-fixed bit-vectors is NP-hard. A more careful analysis of the termination argument can establish that a satisfying unifier can be verified in time polynomial in the constant parameter sizes and number of subterms, hence establishing NP-completeness of the non-fixed bit-vector unification problem.

The unification algorithm finally needs to be supplied also with a canonizer that works on *ext*-terms of non-fixed length to enable an integration with other decision procedures. While simple unfoldings cannot be performed this time our implementation normalizes terms into a concatenation of variables, constants and *ext*-terms whose arguments are fixed size terms in canonical form. The occurrences of *ext* in the resulting expression are then shifted as much as possible to the left. This step cannot be performed unambiguously without asserting congruence constraints on the parameter and hence also leads to case splits.

The table below summarizes a few benchmarks presented to our prototype implementation (coded in SML/NJ, executed on a 200Mhz SUN Ultra II).

1.	equation (3)	satisfiable	0.06 s
2.	$0_{[1]} \otimes 1_{[1]} \otimes 0_{[1]} \otimes x_{[N+7]} \otimes 1_{[1]} \otimes 0_{[1]} \otimes 1_{[1]} \otimes y_{[N+1]}$ $= x_{[N+7]} \otimes x_{[N+7]}$	unsatisfiable	0.06 s
3.	$x_{[N+4]} \otimes 0_{[1]} \otimes 1_{[1]} \otimes 0_{[1]} \otimes y_{[N+9]}$ $= y_{[N+9]} \otimes 1_{[1]} \otimes 0_{[1]} \otimes 1_{[1]} \otimes x_{[N+4]}$	unsatisfiable	0.09 s
4.	$(3) \rightarrow z_{[3]}[0:0] = 0_{[1]} \vee z_{[3]}[1:1] = 0_{[1]}$	valid	0.07 s

4 Conclusion

This paper presented two algorithms: one algorithm handles boolean operations on fixed-size bit-vectors, the other handles equational constraints in the absence of boolean operations on parameterized bit-vectors. A completed picture would combine the algorithms to handle boolean operations on parameterized bit-vectors. Encouraged by the presented results we are currently trying to extend the algorithms to handle parameterized boolean operations, and to address efficient integration of arithmetical operations on bit-vectors. The fixed-size algorithm is presently integrated into STeP's simplifier and validity checker where it has been used in hardware verification. Simultaneously we are experimenting with our prototype implementation of the non-fixed bit-vector decision procedure on verification conditions from parameterized hardware designs.

Acknowledgement: We are grateful for the kind advice from the anonymous referees as well as Michael Colón, Henny B. Sipma and Tomás E. Uribe.

References

1. Baader, F., and Siekmann, J. Unification theory. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. Gabbay, C. Hogger, and J. Robinson, Eds. Oxford University Press, Oxford, UK, 1993.
2. Basin, D., and Klarlund, N. Hardware verification using monadic second-order logic. In *CAV'95* (1995), vol. 939 of *LNCS*, pp. 31–41.
3. Bjørner, N. S., Browne, A., Chang, E. S., Colón, M., Kapur, A., Manna, Z., Sipma, H. B., and Uribe, T. E. STeP: Deductive-algorithmic verification of reactive and real-time systems. In *CAV'96* (1996), vol. 1102 of *LNCS*, pp. 415–418.
4. Bjørner, N. S., Stickel, M. E., and Uribe, T. E. A practical integration of first-order reasoning and decision procedures. In *CADE'97* (1997), vol. 1249 of *LNCS*, pp. 101–115.
5. Bratsch, A., Eveking, H., Färber, H.-J., and Schellin, U. LOVERT - A Logic Verifier of Register-Transfer Level Descriptions. In *IMEC-IFIP* (1989), L. Claesen, Ed., Elsevier.
6. Bryant, R. E. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers C-35*, 8 (1986), 677-691.
7. Bryant, R. E., and Chen, Y.-A. Verification of arithmetic circuits with binary moment diagrams. In *DAC'95* (1995).
8. Comon, H., Dauchet, M., Gilleron, R., Lugiez, D., Tison, S., and Tommasi, M. *Tree Automata Techniques and Applications*. Obtainable from `http://l3ux02.univ-lille3.fr/tata/`, 1998.
9. Cooper, D. C. Theorem proving in arithmetic without multiplication. In *Machine Intelligence*, vol. 7. American Elsevier, 1972, pp. 91–99.
10. Cyrluk, D., Lincoln, P., and Shankar, N. On Shostak's decision procedure for combinations of theories. In *CADE'96* (1996), vol. 1104 of *LNCS*, pp. 463–477.
11. Cyrluk, D., Möller, O., and Ruess, H. An efficient decision procedure for the theory of fixed-sized bit-vectors. In *CAV'97* (1997), vol. 1254 of *LNCS*, pp. 60–71.
12. Jaffar, J. Minimal and complete word unification. *J. ACM 37*, 1 (1990), 47–85.
13. Kamerer, J. Bus scheduler verification using STeP. Unpublished report, 1996.
14. Martelli, A., and Montanari, U. An efficient unification algorithm. *ACM Trans. Prog. Lang. Sys. 4*, 2 (1982), 258–282.
15. Nelson, G., and Oppen, D. C. Simplification by cooperating decision procedures. *ACM Trans. Prog. Lang. Sys. 1*, 2 (1979), 245–257.
16. Niven, I., Zuckerman, H., and Montgomery, H. *An Introduction to the Theory of Numbers*. John Wiley & Sons, New York, 1991.
17. Shostak, R. E. Deciding combinations of theories. *J. ACM 31*, 1 (1984), 1–12.
18. Sipma, H. B., Uribe, T. E., and Manna, Z. Deductive model checking. In *CAV'96* (1996), vol. 1102 of *LNCS*, pp. 208–219.
19. Stockmeyer, L. J., and Meyer, A. R. Word problems requiring exponential time. In *Proc. 5rd ACM Symp. Theory of Comp.* (1973), pp. 1–9.