Precise and Complete Propagation Based Local Search for Satisfiability Modulo Theories

Aina Niemetz(✉), Mathias Preiner, and Armin Biere

Johannes Kepler University, Linz, Austria
aina.niemetz@jku.at

Abstract. Satisfiability Modulo Theories (SMT) is essential for many applications in computer-aided verification. A recent SMT solving approach based on stochastic local search for the theory of quantifier-free fixed-size bit-vectors proved to be quite effective on hard satisfiable instances, particularly in the context of symbolic execution. However, it still relies on brute-force randomization and restarts to achieve completeness. In this paper we simplify, extend, and formalize the propagation-based variant of this approach. We introduce a notion of essential inputs to lift the well-known concept of controlling inputs from the bit-level to the word-level, which allows to prune search. Guided by a formal completeness proof for our propagation-based variant we obtain a clean, simple and more precise algorithm, which yields a substantial gain in performance, as shown in our experimental evaluation.

1 Introduction

In many applications of hardware and software verification, such as (constrained random) test case generation [1–3] or white box fuzz testing [4], the vast majority of problems is satisfiable. Hence, for this kind of problems local search procedures for deciding satisfiability are useful even though they do not allow to determine unsatisfiability. They were further shown to be orthogonal to other approaches [5,6] and proved to be particularly beneficial in a portfolio setting [6].

Applications as above require bit-precise reasoning as provided by Satisfiability Modulo Theories (SMT) solvers for the theory of fixed-size bit-vectors. Current state-of-the-art SMT solvers [7–11] employ the so-called bit-blasting approach (e.g., [12]), where a given formula is eagerly translated to propositional logic (SAT) while heavily relying on rewriting techniques [13–21] to simplify the input during preprocessing.

Since the SAT Challenge 2012 [22], a new generation of (stochastic) local search (SLS) solvers with very simple architecture [23] achieved remarkable results not only in the random but also in the combinatorial tracks of recent SAT competitions [22,24,25]. In an attempt to reproduce these successful applications of SLS in SAT, in [5], Fröhlich et al. lifted SLS to SMT and proposed a

Supported by Austrian Science Fund (FWF) under NFN Grant S11408-N23 (RiSE).

© Springer International Publishing Switzerland 2016

DOI: 10.1007/978-3-319-41528-4_11

procedure to solve bit-vector formulas on the theory level (word-level) without bit-blasting. In contrast to previous attempts of utilizing SLS techniques in SMT by integrating a bit-level SLS solver in the SMT solver MathSAT [9], the SLS for SMT approach in [5] showed promising initial results. However, it does not fully exploit the word-level structure but rather simulates bit-level local search by focusing on single bit flips. As a consequence, in [6] we proposed a propagation-based extension of [5], which introduced an additional strategy to the original approach to propagate assignments from the outputs to the inputs. This significantly improves performance. We further showed that these techniques are beneficial in a sequential portfolio setting [26] in combination with bit-blasting.

Down propagation of assignments along a single path as in [6] utilizes inverse value computation. This process, however, can get stuck, in which case [6] falls back to either brute force randomization or SLS moves to achieve completeness, as does [5]. Further, inverse value computation as presented in [6] is, as we show in this paper, too restrictive for some operators. Actually, focusing on inverse values only is already too restrictive and may inadvertently prune the search.

In this paper, we simplify and extend the approach presented in [6]. Guided by a formal completeness proof, we present a precise and complete variant of the procedure proposed in [6]. In contrast to [6], our variant only uses propagation and neither relies on restarts nor SLS techniques to achieve completeness. We extend the concept of controlling inputs, used in [6] to determine propagation paths for bit-level operators, to the word-level by introducing the notion of essential inputs. This allows to further prune the search. To overcome the problem of too restrictive inverse value computation, we lift the ATPG [27] concept of "backtracing", which goes back to the PODEM algorithm [28], to the word-level and provide a formalization for both the bit-level and the word-level. Our experiments show that our techniques yield a substantial gain in performance.

Note that in contrast to backtracing in ATPG, our algorithm works with complete assignments. Existing algorithms for word-level ATPG [29,30] are based on branch-bound, use neither backtracing nor complete assignments, are further focused on HW models, and in general lack formal treatment. Other related work on structural bit-level solving from the AI community, e.g., [31–33], has not found actual applications yet, as far we know.

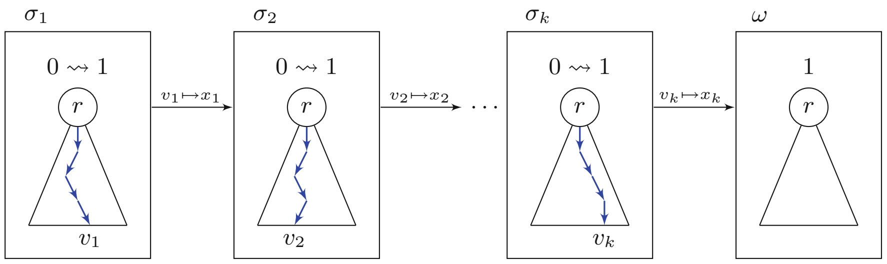

Fig. 1. Basic idea of our propagation-based (local search) strategy.

2 Overview

The propagation-based local search strategy presented in this paper follows the basic idea of the approach presented in [6] as illustrated in Fig. 1. It is applied to both Boolean formulas (bit-level) and quantifier-free fixed-size bit-vector formulas (word-level) and (in contrast to [6]) neither relies on restarts nor requires to fall back on SLS moves (as introduced in [5]) in order to achieve completeness but employs down propagation of assignments only.

Assume that formula ϕ is a directed acyclic graph (DAG) with root r, and σ_1 is a complete but non-satisfying initial assignment, i.e., $\sigma_1(r) = 0$. Our goal is to reach a satisfying assignment ω by changing the value of some primary inputs such that $\omega(r) = 1$. Therefore, we first force root r to assume its target value $\omega(r) = 1$ (denoted as $0 \rightsquigarrow 1$ in Fig. 1), and then propagate this information along a path towards the primary inputs. This process is also known as "backtracing" [28]. Recursively propagating the target value $\omega(r) = 1$ from the root to a primary input (e.g., v_1 in Fig. 1) yields a new assignment $x_i \neq \sigma_i(v_i)$ for a primary input v_i. We then move from assignment σ_i to assignment σ_{i+1} by updating σ_i on v_i to $\sigma_{i+1}(v_i) = x_i$ without changing the value of other primary inputs but recomputing consistent values for inner nodes. Figure 2 describes this strategy more precisely in pseudo code. On the bit-level, as in [6], during path propagation selection of controlling inputs is prioritized w.r.t. the current assignment σ, a well-known concept from ATPG. On the word-level, in contrast to [6], we introduce our new notion of essential inputs, which lifts the bit-level concept of controlling inputs to the word-level while trying to maximally reduce non-deterministic choices without sacrificing completeness.

```
function sat ($r$, $\sigma$)                 // returns Boolean value $\mathbb{B} = \{0, 1\}$
    while $\sigma(r) \neq 1$                  // while not satisfied
        $g = r$, $t = 1$                      // initialize path as root path
        while $\neg$leaf($g$)                 // while current node is an operator
            $n$ = child($\sigma, t, g$)       // select backtracing node
            $x$ = value($\sigma, t, g, n$)    // select backtracing value
            $g = n$, $t = x$                  // backtracing step (propagation)
        if $\neg$constant($g$)                // check if leaf is variable $v = g$
            $\sigma$ = update($\sigma, g, t$) // apply move to variable $v = g$
    return 1                                  // return with 1 (*true*) if satisfied
```

Fig. 2. The core sat procedure in pseudo-code.

As expected for local search, our propagation strategy is not able to determine unsatisfiability. When determining satisfiability, however, our strategy is complete in the sense that if there is a solution, then there exists a non-deterministic choice of moves according to the strategy that leads to a solution.

In the following, we first introduce and formalize our propagation-based approach on the bit-level and prove its completeness. We then lift it to the word-level, again prove its completeness, and show in our experimental evaluation that our techniques yield substantial performance improvements.

3 Bit-Level

In order to simplify the exposition we consider a fixed Boolean formula ϕ and restrict the set of Boolean operators to $\{\wedge, \neg\}$. We can interpret ϕ as a single-rooted And-Inverter-Graph (AIG) [34], where an AIG is a DAG represented as a 5-tuple (r, N, G, V, E). The set of nodes $N = G \cup V$ contains the single root node $r \in N$, and is further partitioned into a set of *gates* G and a set of *primary inputs* (or *variables*) V. We require that $V \neq \emptyset$ and assume that the Boolean *constants* $\mathbb{B} = \{0, 1\}$, i.e., *true* (1) and *false* (0), do not occur in N. This assumption is without loss of generality since every occurrence of *true* and *false* as input to a gate $g \in G$ can be eliminated through rewriting. The set of gates $G = A \cup I$ consists of a set of *and*-gates A and a set of *inverter*-gates I. We write $g = n \wedge m$ if $g \in A$, and $g = \neg n$ if $g \in I$, and refer to the children of a node $g \in G$ as *its* (gate) inputs (e.g., n or m). Let $E = E_A \cup E_I$ be the edge relation between nodes, with $E_A \colon A \rightarrow N^2$ and $E_I \colon I \rightarrow N$ describing edges from *and*-resp. *inverter*-gates to its input(s). We write $E(g) = (n, m)$ for $g = n \wedge m$ and $E(g) = n$ for $g = \neg n$ and further introduce the notation $g \rightarrow n$ for an edge between a gate g and one of its inputs n.

We define a *complete assignment* σ of the given fixed ϕ as a *complete* function $\sigma \colon N \rightarrow \mathbb{B}$, and a *partial assignment* α of ϕ as a *partial* function $\alpha \colon N \rightarrow \mathbb{B}$. We say that a complete assignment σ is *consistent* on a gate $g \in I$ with $g = \neg n$ iff $\sigma(g) = \neg\sigma(n)$, and consistent on a gate $g \in A$ with $g = n \wedge m$ iff $\sigma(g) = \sigma(n) \wedge \sigma(m)$. A complete assignment σ is *globally consistent* on ϕ (or just *consistent*) iff it is consistent on all gates $g \in G$. An assignment σ is *satisfying* if it is consistent (thus complete) and satisfies the root, i.e., $\sigma(r) = 1$. We use the letter ω to denote a satisfying assignment. A formula ϕ is satisfiable if it has a satisfying assignment. Let $\mathcal{C}$ be the set of *consistent* assignments, and let $\mathcal{W}$ with $\mathcal{W} \subseteq \mathcal{C}$ be the set of *satisfying* assignments of formula ϕ.

Given two consistent assignments σ and σ', we say that σ' is obtained from σ by *flipping* the (assignment of a) variable $v \in V$, written as $\sigma \xrightarrow{v} \sigma'$, iff $\sigma(v) = \neg\sigma'(v)$ and $\sigma(u) = \sigma'(u)$ for all $u \in V \backslash \{v\}$. We also refer to flipping a variable as a *move*. Note that $\sigma'(g)$ for gates $g \in G$ is defined implicitly due to consistency of σ' after fixing the values for the primary inputs V.

Given as a set of variables V that can be flipped non-deterministically, let $S \colon \mathcal{C} \rightarrow \mathbb{P}(\mathcal{M})$ be a (local search) *strategy* that maps a consistent assignment to a set of possible moves $\mathcal{M} = V$. The move $v \in V$ is *valid* under strategy S for a consistent assignment $\sigma \in \mathcal{C}$ if $v \in S(\sigma)$. Similarly, a sequence of moves $\mu = (v_1, \ldots, v_k) \in V^*$ of length $k = |\mu|$ with $v_1, \ldots, v_k \in V$ is *valid* under strategy S iff there exists a sequence of consistent assignments $(\sigma_1, \ldots, \sigma_{k+1}) \in \mathcal{C}^*$ such that $\sigma_i \xrightarrow{v_i} \sigma_{i+1}$ and $v_i \in S(\sigma_i)$ for $1 \leq i \leq k$. In this case σ_{k+1} can be *reached* from σ_1 under S (with k moves), also written $\sigma_1 \rightarrow^* \sigma_{k+1}$.

Definition 1. *If formula ϕ is satisfiable, then a strategy S is called* complete *iff for all consistent assignments $\sigma \in \mathcal{C}$ there exists a satisfying assignment $\omega \in \mathcal{W}$ such that ω can be reached from σ under S, i.e., $\sigma \rightarrow^* \omega$.*

Given a satisfiable assignment $\omega \in \mathcal{W}$ and a consistent assignment $\sigma \in \mathcal{C}$, let $\Delta(\sigma, \omega) = \{v \in V \mid \sigma(v) \neq \omega(v)\}$. Thus $|\Delta(\sigma, \omega)|$ is the Hamming Distance between σ and ω on V. We say that a strategy S is (non-deterministically) *distance reducing*, if for all $\sigma \in \mathcal{C} \backslash \mathcal{W}$ and $\omega \in \mathcal{W}$ there exists a move $\sigma \xrightarrow{v} \sigma'$ valid under S which reduces the Hamming Distance, i.e., move v is in $\Delta(\sigma, \omega)$ thus $|\Delta(\sigma, \omega)| - |\Delta(\sigma', \omega)| = 1$. Obviously, any distance reducing strategy can reach a satisfying assignment (though not necessarily ω) within at most $|\Delta(\sigma, \omega)|$ moves. This first observation is the key argument in the completeness proofs for our propagation based strategies later on (both on the bit-level and word-level).

Proposition 2. *A distance reducing strategy is also complete.*

The ultimate goal of this paper is to define a strategy that maximally reduces non-deterministic choices without sacrificing completeness. In the algorithm shown in Fig. 2, selecting the backtracing node (line 5) and its value (line 6) constitute the only source of non-determinism. This non-determinism can be reduced by using the notion of controlling inputs from ATPG [27], which was already considered in [6], but only for Boolean expressions.

Definition 3 (Controlling Input). *Let $n \in N$ be an input of a gate $g \in G$, i.e., $g \rightarrow n$, and let σ be a complete assignment consistent on g. We say that input n is* controlling *under σ, if for all complete assignments σ' consistent on g with $\sigma(n) = \sigma'(n)$ we have $\sigma(g) = \sigma'(g)$.*

In other words, gate input n is controlling, if the assignment of g, i.e., its output value, remains unchanged as long as the assignment of n does not change. Given an assignment σ consistent on a gate $g \in G$, we denote a *target value* t for g as $\sigma(g) \rightsquigarrow t$. On the bit-level, t is *implicitly* given through σ as $t = \neg\sigma(g)$, i.e., t can not be reached as long as the controlling inputs of g remain unchanged. On the word-level, t may be any value $t \neq \sigma(g)$.

Example 4. Figure 3 shows all possible assignments σ consistent on a gate $g \in G$. At the outputs we denote current assignment $\sigma(g)$ and target value t as $\sigma(g) \rightsquigarrow t$ with $t = \neg\sigma(g)$, e.g., $0 \rightsquigarrow 1$. At the inputs we show their assignment under σ. All controlling inputs are indicated with an underline. Note that for $\sigma(g) = 1$, *and*-gate $g = n \wedge m$ has *no* controlling inputs.

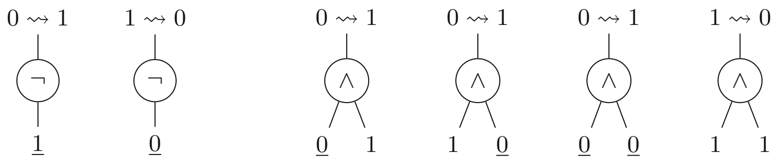

Fig. 3. An *inverter* and an *and*-gate and their controlling (underlined) inputs. Given output values indicate the transition from current to target value.

A sequence of nodes $\pi = (n_1, \ldots, n_k) \in N^+$ is a *path* of length $k = |\pi|$ iff $n_i \rightarrow n_{i+1}$ for $0 < i < k$, also written as $n_1 \rightarrow \ldots \rightarrow n_k$. A path π is *rooted* if $n_1 = r$, and (fully) *expanded* if $n_k \in V$. We call $n_k \in V$ the *leaf* of π in this case. As a restriction on ϕ, we require all paths to be acyclic, i.e., for all $n \in N$ there is no path $n \rightarrow^+ n$, and all nodes $n \in N$ to be reachable from the root, i.e., there exists a path $\pi = (r, \ldots, n)$. Note that as a consequence of representing ϕ as a DAG, any path in ϕ is acyclic. Given a path $\pi = (\ldots, g)$ with $g \in G$ and $g \rightarrow n$, we say that $\pi.n = (\ldots, g, n)$ is an *extension* of path π with node n.

Definition 5 (Path Selection). *Let $\sigma \in \mathcal{C}$ be a (complete) consistent assignment, and let $\pi = (\ldots, g)$ be a path as above. Then* input n can be selected *w.r.t. σ to extend π to $\pi.n$ if n is controlling or if g has no controlling input.*

Path selection based on the notion of controlling inputs as introduced above exploits observability don't cares as defined in the context of ATPG [27]. Similarly, we adopt the ATPG idea of backtracing [27,28], as follows.

Definition 6 (Backtracing Step). *Let $\sigma \in \mathcal{C}$ be a (complete) consistent assignment. Given a gate $g \in G$ with $g \rightarrow n$, a* backtracing step *w.r.t. σ selects input n of g w.r.t. σ as in* Definition 5*, and determines a* backtracing value *x for n as follows. If $g = \neg n$, then $x = \sigma(g)$. Else, if $g = n \wedge m$, then $x = \neg\sigma(g)$.*

As an important observation it turns out that performing a (bit-level) backtracing step always flips the value of the selected input under σ. For a selected input the backtracing value is therefore always unique. This can be checked easily by considering all possible scenarios shown in Fig. 3.

Proposition 7. *A backtracing step always yields as backtracing value $x = \neg\sigma(n)$.*

Example 8. Consider $g = n \wedge m$ and the assignment $\sigma = \{g \mapsto 0, n \mapsto 0, m \mapsto 1\}$ consistent on g as depicted in Fig. 3. Assume that $t = \neg\sigma(g) = 1$ is the target value of g, i.e., $\sigma(g) \rightsquigarrow t$ with $0 \rightsquigarrow 1$. We select n as the single controlling input of g (underlined), which yields backtracing value $x = \neg\sigma(n) = 1$ for n.

A *trace* $\tau = (\pi, \alpha)$ is a rooted path $\pi = (n_1, \ldots, n_k)$ labelled with a partial assignment α, where α is defined exactly on $\{n_1, \ldots, n_k\}$. A trace (π, α) is (fully) *expanded*, if π is a fully expanded path, i.e., node $n_k \in V$ is the *leaf* of π and τ.

Let $\sigma \in \mathcal{C} \backslash \mathcal{W}$ be a consistent but non-satisfying assignment. Then the notion of extension is lifted from paths to traces as follows. A trace $\tau' = (\pi', \alpha')$ *extends* $\tau = (\pi, \alpha)$ by a *propagation step* (or backtracing step) w.r.t. σ, denoted $\tau \rightarrow \tau'$, if $\pi' = \pi.n$ is an extension of $\pi = (\ldots, g)$ with $g \in G$, $\alpha'(m) = \alpha(m)$ for all nodes m in π, and $x = \alpha'(n)$is the (unique) backtracing value of this backtracing step on g w.r.t. σ and target value $t = \neg\sigma(g)$. The *root trace* $\rho = ((r), \{r \mapsto 1\})$ isa trace that maps r to its target value $\omega(r) = 1$.A *propagation trace* τ w.r.t. σ is a (partial) trace that starts from the root trace ρ and is extended by propagation steps w.r.t. σ, i.e., $\rho \rightarrow^* \tau$. Note that α is redundant on the bit-level, given π and σ. We use the same notation on the word-level though, where α captures updates to σ along π, which (in contrast to the bit-level) are not uniquely defined.

Definition 9 (Propagation Strategy). *Given a non-satisfying but consistent assignment $\sigma \in \mathcal{C} \backslash \mathcal{W}$, the set of valid moves $\mathcal{P}(\sigma)$ for σ under propagation strategy $\mathcal{P}$ contains exactly the leafs of all expanded propagation traces w.r.t σ.*

In the following, we present and prove the main Lemma of this paper, which then immediately gives completeness of $\mathcal{P}$ in Theorem 11. It is further reused for proving completeness of the extension of $\mathcal{P}$ to the word-level in Theorem 26.

Lemma 10 (Propagation Lemma). *Given a non-satisfying but consistent assignment $\sigma \in \mathcal{C} \backslash \mathcal{W}$, then for any satisfying assignment $\omega \in \mathcal{W}$, used as oracle, there exists an expanded propagation trace τ w.r.t. σ with leaf $v \in \Delta(\sigma, \omega)$.*

Proof. The idea is to inductively extend the root trace ρ to traces $\tau = (\pi, \alpha)$ through propagation steps, i.e., $\rho \rightarrow^* \tau$, which all satisfy the (key) invariant

$$\alpha(n) = \omega(n) \neq \sigma(n) \quad \text{for all nodes } n \text{ in } \pi. \tag{1}$$

The root trace $\rho = ((r), \{r \mapsto \omega(r)\})$ obviously satisfies this invariant. Now, let $\tau = (\pi, \alpha)$ be a trace that satisfies the invariant but is *not fully expanded*, i.e., $\pi = (r, \ldots, g)$ with $g \in G$ and $\alpha(g) = \omega(g) \neq \sigma(g)$. Since $\sigma(g) \neq \omega(g)$ it follows that g has at least one input n with $\sigma(n) \neq \omega(n)$. If g has no controlling input, then by Definition 5 it is allowed to select n as an input with $\sigma(n) \neq \omega(n)$. Otherwise, input n is selected as any controlling input. In both cases we select $x = \omega(n) \neq \sigma(n)$ as backtracing value using Proposition 7. Trace τ is now extended by n with backtracing value x to τ', i.e., $\tau \rightarrow \tau'$, which in turn concludes the inductive proof of Eq. (1). Any fully expanded propagation trace $\tau = (\pi, \alpha)$ with leaf $v \in V$, as generated above, also satisfies the invariant in Eq. (1). Thus, we have $\alpha(v) = \omega(v) \neq \sigma(v)$ with $v \in \Delta(\sigma, \omega)$. □

In essence, given σ and ω as above, our propagation strategy propagates target value $\omega(r)$ from root r towards the primary inputs, ultimately producing a fully expanded propagation trace τ. In case of non-deterministic choices for extending the trace we use ω as an oracle to pick an input n with $\sigma(n) \neq \omega(n)$, which can be selected according to Definition 5. The oracle allows us to ensure that $v \in \Delta(\sigma, \omega)$ for leaf v of τ. Thus, using Lemma 10, our propagation strategy turns out to be distance reducing, and therefore, according to Proposition 2, complete. This important contribution of this paper serves in the following as a basis for lifting our approach from the bit-level to the word-level.

Theorem 11. *Under the assumptions of the previous* Lemma 10 *we also get $v \in \mathcal{P}(\sigma)$. Thus $\mathcal{P}$ is distance reducing and, as a consequence, complete.*

Our notion of completeness follows the traditional notion of non-deterministic computation of Turing machines. For the purpose of this paper, it is equivalent to the more established property of *probabilistically approximately complete* (PAC) [35], commonly used in the AI community to discuss completeness properties of local search algorithms. This holds as long as probabilistic choices are treated as non-deterministic choices, which actually also is the case in [35].

4 Word-Level

We only consider bit-vector expressions of *fixed* bit-width $w \in \mathbb{N}$, and denote a bit-vector expression n of width w as $n_{[w]}$. We will not explicitly state the bit-width of an expression if the context allows. We refer to the i-th bit of $n_{[w]}$ as $n[i]$ for $1 \leq i \leq w$, and further interpret $n[1]$ as the least significant bit (LSB) and $n[w]$ as the most significant bit (MSB). Similarly, we denote bit ranges over n from bit index j down to index i as $n[j:i]$. Note that for the sake of simplicity, bit indices start from 1 rather than 0. Further, in string representations of bit-vectors we interpret the bit at the far left index as the MSB.

To simplify the exposition and w.l.o.g. we consider a fixed single-rooted quantifier-free bit-vector formula ϕ with Boolean expressions interpreted as bit-vector expressions of bit-width one. The set of bit-vector operators is restricted to $\mathcal{O} = \{\&, \sim, =, <, \ll, \gg, +, \cdot, \div, \text{mod}, \circ, [:], \text{if-then-else}\}$ and interpreted according to Table 1. The selection of operators in $\mathcal{O}$ is rather arbitrary but provides a good compromise between effective and efficient word-level rewriting and compact encodings for bit-blasting approaches. It is complete, though, in the sense that all operators defined in SMT-LIB [36] (in particular signed operators) can be modeled in a compact way. Note that our methods are not restricted to single-rootedness or this particular selection of operators, and can easily be lifted to the full set of operators in SMT-LIB as well as the multi-rooted case.

We interpret formula ϕ as a single-rooted DAG represented as an 8-tuple $(r, N, \kappa, O, F, V, B, E)$. The set of nodes $N = O \cup V \cup B$ contains the single root node $r \in N$ of bit-width one, and is further partitioned into a set of operator nodes O, a set of primary inputs (or bit-vector variables) V, and a set of bit-vector constants $B \subseteq \mathbb{B}^*$, which are denoted in either decimal or binary notation if the context allows. The bit-width of a node is given by $\kappa \colon N \to \mathbb{N}$, thus $\kappa(r) = 1$. Operator nodes are interpreted as bit-vector operators via $F : O \to \mathcal{O}$, which in turn determines their arity and input and output bit-widths as defined in Table 1. The edge relation between nodes is given as $E = E_1 \cup E_2 \cup E_3$, with $E_i \colon O \to N^i$ describing the set of edges from unary, binary, and ternary operator nodes to its input(s), respectively. We again use the notation $o \to n$ for an edge between an operator node o and one of its inputs n.

We only consider well-formed formulas, where the bit-widths of all operator nodes and their inputs conform to the conditions imposed via interpretation F as defined in Table 1. For instance, we denote a bit-vector addition node o with inputs n and m as $o = n + m$, where $o \in O$ of arity 2 with $F(o) = +$, and therefore $\kappa(o) = \kappa(n) = \kappa(m)$. In the following, if more convenient we will use the functional notation $o = \diamond(n_1, \ldots, n_k)$ for operator node $o \in O$ of arity k with inputs $n_1, \ldots, n_k$ and $F(o) = \diamond$, e.g., $+(n, m)$. Note that the semantics of all operators in $\mathcal{O}$ correspond to their SMT-LIB counterparts listed in Table 1, with three exceptions. Given a logical shift operation $n \ll m$ or $n \gg m$, w.l.o.g. and as implemented in our SMT solver Boolector [7], we restrict bit-width $\kappa(n)$ to $2^{\kappa(m)}$. Further, as implemented by Boolector and other state-of-the-art SMT solvers, e.g., Mathsat [9] Yices [10] and Z3 [11], we define an unsigned division by

Table 1. Considered bit-vector operators – bit-widths and indices $w, p, q, i, j \in \mathbb{N}$.

		Output bit-width	Arity	Input bit-width			
Operator	SMT-LIB			1st	2nd	3rd	
$[j:i]$	`extract`	$j-i+1$	1	w	–	–	extraction $(1 \leq i \leq j \leq w)$
$\sim$	`bvnot`	w	1	w	–	–	bit-wise negation
$\&$	`bvand`	w	2	w	w	–	bit-wise conjunction
$=$	`=`	1	2	w	w	–	equality
$<$	`bvult`	1	2	w	w	–	unsigned less than
$\ll$	`bvshl`	w	2	w	q	–	logical shift left $(w = 2^q)$
$\gg$	`bvshr`	w	2	w	q	–	logical shift right $(w = 2^q)$
$+$	`bvadd`	w	2	w	w	–	addition
$\cdot$	`bvmul`	w	2	w	w	–	multiplication
$\div$	`bvudiv`	w	2	w	w	–	unsigned division
mod	`bvurem`	w	2	w	w	–	unsigned remainder
$\circ$	`concat`	w	2	p	q	–	concatenation $(w = p + q)$
if-then-else	`ite`	w	3	1	w	w	conditional

zero to return the greatest possible value rather than introducing uninterpreted functions, i.e., $x \div 0 = \sim 0$. Similarly, $x \bmod 0 = x$.

A complete function $\sigma \colon N \rightarrow \mathbb{B}^*$ with $\sigma(n) \in \mathbb{B}^{\kappa(n)}$ of a given fixed ϕ is called a *complete assignment* of ϕ, and a partial function $\alpha \colon N \rightarrow \mathbb{B}^*$ with $\alpha(n) \in \mathbb{B}^{\kappa(n)}$ a *partial assignment*. Given an operator node $o \in O$ with $o = \diamond(n_1, \ldots, n_k)$ and $\diamond \in \mathcal{O}$, a complete assignment σ is *consistent* on o if $\sigma(o) = f(\sigma(n_1), \ldots, \sigma(n_k))$, where $f \colon \mathbb{B}^{\kappa(n_1)} \times \cdots \times \mathbb{B}^{\kappa(n_k)} \rightarrow \mathbb{B}^{\kappa(o)}$ is determined by the standard semantics of bit-vector operator $\diamond$ as defined in SMT-LIB [36] (with the exceptions discussed above). A complete assignment is (globally) *consistent* on ϕ (or just *consistent*), iff it is consistent on all bit-vector operator nodes $o \in O$ and $\sigma(b) = b$ for all bit-vector constants $b \in B$. A *satisfying* assignment ω is a complete and consistent assignment that satisfies the root, i.e., $\omega(r) = 1$. In the following, we will again use the letter $\mathcal{C}$ to denote the set of complete and consistent assignments, and the letter $\mathcal{W}$ with $\mathcal{W} \subseteq \mathcal{C}$ to denote the set of satisfying assignments of formula ϕ.

Given a bit-vector variable $v \in V$ with $\kappa(v) = w$, we adopt the notion of obtaining an assignment σ' from an assignment σ by assigning a new value x to v with $x \in \mathbb{B}^w$ where $x \neq \sigma(v)$, written as $\sigma \xrightarrow{v \mapsto x} \sigma'$, which we refer to as a *move*. Thus, the set of word-level moves is defined as $\mathcal{M} = \{(v, x) \mid v \in V, x \in \mathbb{B}^{\kappa(v)}\}$, and accordingly, a word-level strategy is defined as a function $S \colon C \mapsto \mathbb{P}(\mathcal{M})$ mapping a consistent assignment to a set of moves.

Our propagation strategy $\mathcal{P}$ is lifted from the bit-level to the word-level starting with our new notion of essential inputs for word-level operators, which lifts and extends the corresponding notion of controlling inputs.

Definition 12 (Essential Inputs). *Let $n \in N$ be an input of a bit-vector operator node $o \in O$, i.e., $o \rightarrow n$, and let σ be a complete assignment consistent* *on o. Further, let t be the* target value *of o, i.e., $\sigma(o) \rightsquigarrow t$, with $t \neq \sigma(o)$. We say that n is an* essential input *under σ w.r.t. target value t, if for all complete assignments σ' consistent on o with $\sigma(n) = \sigma'(n)$ we have $\sigma'(o) \neq t$.*

In other words, an operator node input n is essential w.r.t. target value t, if o can not assume t as long as the assignment of n does not change. As an example, consider the bit-vector operators and their essential inputs under some consistent assignment σ w.r.t. some target value t as in Fig. 4.

Example 13. Consider the bit-vector operators &, $\ll$, $\cdot$, $\div$, mod, and $\circ$ of bit-width 2 in Fig. 4. For an operator node o, at the outputs we denote given assignment $\sigma(o)$ and target value t as $\sigma(o) \rightsquigarrow t$, e.g., $10 \rightsquigarrow 01$. At the inputs we show their assignment under σ. Essential inputs are indicated with an underline. Consider, e.g., operator node $o := n \,\&\, m$ with $t = 01$ and $\sigma = \{o \mapsto 10,\ n \mapsto 10,\ m \mapsto 11\}$. Since $t \,\&\, \sigma(n) \neq t$, it is not possible to find a value x for m such that $\sigma(n) \,\&\, x = t$. Hence, input n is essential w.r.t. target value t. Input m, however, is obviously not essential since $t \,\&\, \sigma(m) = t$.

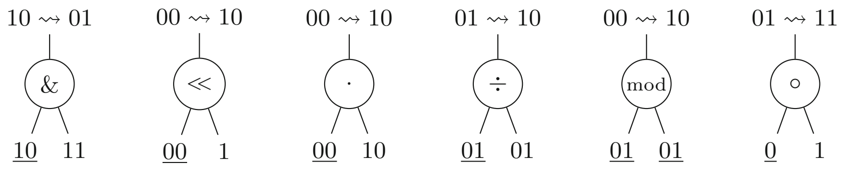

Fig. 4. Bit-vector operator nodes and examples for essential (underlined) inputs. Given output values indicate the transition from current to target value.

Note that AIGs can be represented by bit-vectors of bit-width one, which can be interpreted as Boolean expressions. In this sense, as in [6], the notion of controlling inputs can also be applied to word-level Boolean expressions.

Proposition 14. *When applied to bit-level or Boolean expressions, the notion of* essential inputs *exactly matches the notion of* controlling inputs.

Proof. For applying the notion of essential inputs to the bit-level, consider the operator set $\{\neg, \wedge\}$ for $o \in G$ with $o \rightarrow n$. Further, target value $t \neq \sigma(o)$ on the bit-level implies $t = \neg\sigma(o)$, which exactly matches the corresponding implicit definition of the target value of o on the bit-level. Now assume that input n is essential w.r.t. t. Then, if $\sigma(n) = \sigma'(n)$, by Definition 12 we have that $\sigma'(o) \neq t$, and therefore $\sigma'(o) = \neg t = \sigma(o)$. The other direction works in the same way. $\square$

The definition of a (rooted and expanded) *path* as a sequence of nodes $\pi = (n_1, \ldots, n_k) \in N^*$ is lifted from the bit-level to the word-level in the natural way. Corresponding restrictions and implications of Sect. 3 apply. The notions of *path selection* and *path extension* are lifted to the word-level as follows.

Definition 15 (Path Extension). *Given a path $\pi = (\dots, o)$ with $o \in O$ and $o \rightarrow n$, we say that $\pi.n = (\dots, o, n)$ is an extension of path π with node n.*

Definition 16 (Path Selection). *Given a (complete) consistent assignment $\sigma \in \mathcal{C}$, a path $\pi = (\dots, o)$ as in* Definition 15 *above, and $\sigma(o) \rightsquigarrow t$, i.e., $t \neq \sigma(o)$, then* input n can be selected *w.r.t. σ and t to extend π to $\pi.n$ if n is essential or if o has no essential input (in both cases essential under σ w.r.t. t).*

In contrast to the bit-level, where a *backtracing step* always flips the assigned value of a selected input (and the output), on the word-level we also have to select a *backtracing value*. We consider three variants of value selection under the following assumptions. Let t be the target value of an operator node $o \in O$, and let σ be a complete assignment such that $\sigma(o) \neq t$. Further, assume that input n with $o \rightarrow n$ is selected w.r.t. t and σ as in Definition 16 above.

Definition 17 (Random Value). *Any value x with $\kappa(x) = \kappa(n)$ is called a* random value *for n.*

Definition 18 (Consistent Value). *A random value x is a* consistent value *for n w.r.t. t, if there is a complete assignment σ' consistent on o with $\sigma'(n) = x$ and $\sigma'(o) = t$.*

In other words, a value is consistent for an input, if it allows to produce the target value after changing values of other inputs if necessary.

Example 19 (Consistent Value Computation). Consider operator node $o := n \cdot m$, and assume that input n is selected w.r.t. t and σ as in Definition 16. Let *ctz* be defined as the function *counting the number of trailing zeroes* of a given bit-vector, i.e., the number of consecutive 0-bits starting from the LSB, e.g., $ctz(0101) = 0$, $ctz(111100) = 2$. Then any random value x with $ctz(t) \geq ctz(x)$ is a consistent value for n, except that $t = 0$ if $x = 0$.

Restricting the notion of consistent values even further, however, may be beneficial in some cases. Consider the following motivating example from [6].

Example 20 (From [6]). Let $\phi := 274177_{[65]} \cdot v = 18446744073709551617_{[65]}$. Computing $x = 18446744073709551617_{[65]} \div 274177_{[65]} = 67280421310721_{[65]}$ immediately concludes with a satisfying assignment for ϕ.

The chances to select $x = 67280421310721$ if consistent values for the multiplication operator are chosen as in Example 19 are arbitrarily small. Hence, as in [6], we use the notion of *inverse values* which utilize the inverse of a given operator.

Definition 21 (Inverse Value). *A consistent value x is an* inverse value *for n w.r.t. t and σ, if there exists a complete assignment σ' consistent on o with $\sigma'(n) = x$, $\sigma'(o) = t$ and $\sigma'(m) = \sigma(m)$ for all m with $o \rightarrow m$ and $m \neq n$.*

In other words, we define an inverse value as a consistent value for an input n such that operator node o assumes target value t without changing the assignment of its other inputs.

Example 22 (Inverse Value Computation). Consider operator node $o := n \cdot m$ of bit-width w. Assume that input n is selected w.r.t. t and σ as in Definition 16. If $t = \sigma(m) = 0$, any x is an inverse value. However, this contradicts assumption $\sigma(o) \neq t$. If $t \neq 0$ and $\sigma(m) = 0$, or if $\sigma(m) \neq 0$ with $ctz(t) < ctz(\sigma(m))$, there exists no inverse value. Otherwise, $ctz(t) \geq ctz(\sigma(m))$ and $\sigma(m) \neq 0$. Let $y = m \gg ctz(\sigma(m))$, thus y is odd. We compute y^{-1} as its multiplicative inverse modulo 2^w via the Extended Euclidean algorithm (similar to word-level rewriting techniques that require solving for a variable, e.g. [14]) and determine x as $(t \gg ctz(\sigma(m))) \cdot y^{-1}$ except that all bits in $x[w : w - ctz(\sigma(m)) + 1]$ are set arbitrarily, with $w = \kappa(n)$.

In contrast to [6], we require inverse value computation for all operators in $\mathcal{O}$ to generate all possible inverse values. This holds for the rule for inverse computation for multiplication discussed in the example above, while it does not hold for the corresponding rule in [6]. The same applies for operators $\ll$, $\gg$, $\div$, and mod.

Definition 23 (Backtracing Step). *Let $\sigma \in \mathcal{C}$ be a (complete) consistent assignment. Given an operator node $o \in O$ with $o \rightarrow n$ and a target value $t \neq \sigma(o)$, then a* backtracing step *selects input n of o w.r.t. σ as in* Definition 16 *and selects a* backtracing value *x for n as a* consistent *(and optionally* inverse*) value w.r.t. σ and t, if such a value exists, and a* random value *otherwise.*

Note that it is not always possible to find an inverse value for n, e.g., $o := n \,\&\, m$ with $\sigma = \{o \mapsto 00, n \mapsto 00, m \mapsto 00\}$ and $t = 01$. Further, even for operators that allow to always produce inverse values, e.g., operator $+$, doing so may lead to inadvertently pruning the search space, see Example 24 below.

Example 24. Given $r := p_2 = 0_{[2]}$ with $p_2 := v + p_1$ and $p_1 := v + 2_{[2]}$ and a complete consistent assignment $\sigma_1 = \{v \mapsto 00,\, p_1 \mapsto 10,\, p_2 \mapsto 10, r \mapsto 0\}$, as shown in Fig. 5. Let $t = 1$ be the target value of root node r, i.e., we want to find a value for bit-vector variable v such that $p_2 = 00$. Assume that only inverse values are selected for $+$ operators during propagation. Propagating along the path indicated by blue arrows in Fig. 5 the move $v \mapsto 10 = \alpha_1(v)$ is generated, which yields assignment $\sigma_2 = \{v \mapsto 10,\, p_1 \mapsto 00,\, p_2 \mapsto 10, r \mapsto 0\}$. Selecting the other possible propagation path, the same move is produced. Thus, σ_2 is independent of which of the two paths is selected. Since $\sigma_2(r) \neq t$, target value t is again propagated down, which generates move $v \mapsto 00 = \alpha_2(v)$, again independently of which path is selected. With this, we move back to the initial assignment σ_1. Consequently, a satisfying assignment, e.g., $\omega(v) = 01$ or $\omega'(v) = 11$, can not be reached by only selecting inverse values. However, selecting a consistent but non-inverse value for p_1 generates move $v \mapsto 01 = \alpha'_1(v)$, which yields ω.

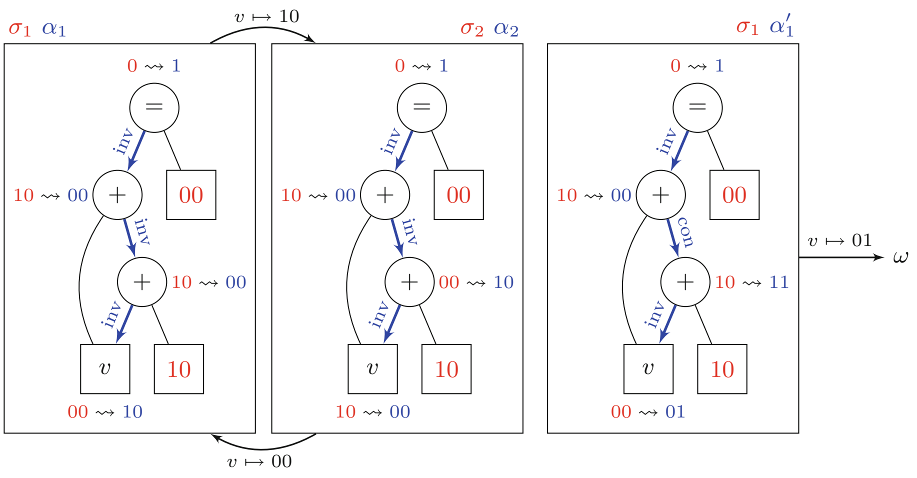

Fig. 5. Example illustrating the necessity of choosing between random and inverse values. The output values indicate the (desired) transition from current to target value. Other values indicate the transition from current value to the value yielded by down propagating the target output value. Thus a propagation based strategy without further randomization using only inverse values as in [6] is incomplete. (Color figure online)

As a consequence, when performing a backtracing step we in general select some consistent non-inverse value, if no inverse value exists, and otherwise non-deterministically choose between consistent (but not necessarily inverse) and inverse values. Since all operators in $\mathcal{O}$ are *surjective* (i.e., they can produce any target value) for our selected semantics (e.g., $\sim 0 \text{mod} 0 = \sim 0$), it is not necessary to select inconsistent random values. For other sets of operators, however, this might be necessary. For the sake of completeness we therefore included the selection of random values in the formal definition of backtracing steps.

Note that on the bit-level, the backtracing value for a selected input is uniquely determined (see Proposition 7). Thus, the issue of value selection is specific to the word-level. Further, when interpreting AIGs as word-level expressions, the notion of backtracing steps on the bit-level as in Definition 6 exactly matches the word-level notion as in Definition 23 using Proposition 14.

The *word-level propagation strategy* $\mathcal{P}$ is defined in exactly the same way as for the bit-level (see Definition 9) except that the word-level notion of backtracing based on essential inputs and consistent value selection (Definition 23) replaces bit-level backtracing based on controlling inputs (Definition 6), and the set of valid moves $\mathcal{P}(\sigma)$ contains not only the leafs of all expanded propagation traces but also their updated assignments, i.e., $(v, \alpha(v))$ for a leaf v. Further important concepts defined on the bit-level in Sect. 3 can be extended naturally to the word-level. These concepts include (expanded) paths and traces, leafs, and trace extension. We omit formal definitions accordingly.

The completeness proof on the word-level is almost identical to the bit-level proof, except that the bit-level Proposition 7, which is substantial for the bit-level proof of Lemma 10, requires more attention due to the more sophisticated selection of backtracing values on the word-level.

Proposition 25. *Let $\sigma \in \mathcal{C}$ be a (complete) consistent assignment, and let ω be a satisfying assignment $\omega \in \mathcal{W}$. Given operator node $o \in O$ and target value $t = \omega(o) \neq \sigma(o)$, i.e., $\sigma(o) \rightsquigarrow t$. Then there exists a backtracing step w.r.t. σ and t, which selects input n with $o \to n$ and backtracing value $x = \omega(n) \neq \sigma(n)$.*

Proof. First, assume o has an essential input w.r.t. σ. Then we select an arbitrary essential input n of o. Since target value $t = \omega(o) \neq \sigma(o)$, we get $\sigma(n) \neq \omega(n)$ by contraposition of Definition 12. Similarly, if o has no essential inputs, then we select n as an arbitrary input with $\sigma(n) \neq \omega(n)$, which has to exist since $\omega(o) \neq \sigma(o)$. In both cases, we can select $x = \omega(n) \neq \sigma(n)$ as backtracing value, which is consistent for o w.r.t. σ and t since ω is consistent. Picking a random value as backtracing value, which is the last case in Definition 23, can not occur under the given assumptions since, as already discussed, ω is consistent on o. □

With Proposition 25, the proof of the key Lemma 10 for completeness on the bit-level can be lifted to the word-level by replacing each occurrence of gate g with operator node o and the notion of "controlling" with "essential" input, and further using Proposition 25 instead of Proposition 7, as discussed above.

Theorem 26. Theorem 11 *and* Lemma 10 *also apply to the word-level setting, and thus, propagation strategy $\mathcal{P}$ is also complete for the word-level.*

As a side note, the problem of value selection during word-level backtracing and subsequent word-level propagation is similar to the problem of making a theory decision and propagating this decision in MCSat [37,38].

5 Experimental Evaluation

We implemented our approach within our SMT solver Boolector and consider two configurations Paig and Pw to be evaluated against [6] on the same set of benchmarks[1] (16436 total). Configuration Paig works on AIGs obtained after bit-blasting by Boolector. Configuration Pw works directly on the bit-vector formula, with inverse values prioritized over consistent values during backtracing with a probability of 99 to 1. As base case, we use the *best* configurations Bb, Bprop and Bb+Bprop of the version of Boolector used in [6], where Bb is its bit-blasting engine and Bprop implements the propagation-based approach of [6]. As in [6], Bb+Bprop serves as a "virtual" sequential portfolio, i.e., Bprop is run for a certain time limit as a preprocessing step prior to invoking Bb. The best portfolio setting in [6] used random walks for recovery and is called Bb+Bprop+frw in [6]. We only compare against this best version, but call it

[1] All experimental data of this evaluation can be found at http://fmv.jku.at/cav16.

Bb+Bprop. Corresponding to Bb+Bprop, we introduce a "virtual" sequential portfolio configuration Bb+Pw, which combines Bb and Pw as above.

The benchmark set of [6] was compiled from all benchmarks with status sat and unknown in the QF_BV category of the SMT-LIB[2] benchmark library, where all benchmarks proved by Bb to be unsatisfiable within a time limit of 1200 s where excluded. Further excluded where all benchmarks solved by Boolector via rewriting only as well as all benchmarks from the *Sage2* family that were not SMT-LIB v2 compliant due to the use of non-compliant operators.

Table 2. Numbers in parentheses indicate the number of instances solved by a configuration within the given time limit, but not solved by Bb in 1200 s.

	Bprop	Paig	Pw	Bb	Bb+Bprop	Bb+Pw
time limit	10 sec	10 sec	10 sec	1200 sec	1200 sec	1200 sec
# solved	7539 (56)	6789 (14)	**8012** (67)	14806	14862	**14873**
total time	90462	99442	**86061**	2611840	2575700	**2562108**
time limit	1 sec	1 sec	1 sec	1200 sec	1200 sec	1200 sec
# solved	7268 (38)	6016 (2)	**7632** (60)	14806	14844	**14866**
total time	9544	11151	**9106**	2611840	2534471	**2513348**

All experiments were performed on a cluster with 30 nodes of 2.83 GHz Intel Core 2 Quad machines with 8 GB of memory using Ubuntu 14.04.3 LTS. Each run is limited to use 7 GB of main memory. In terms of run time we consider CPU time only and apply the same time limits as in [6], i.e., a limit of 1 and 10 s for the propagation-based approaches and a limit of 1200 s for the bit-blasting configuration and its combinations. In case of a time out or a memory out, the time limit is used as run time.

Table 2 summarizes the results, where the numbers given in parentheses indicate the number of uniquely solved instances solved by a configuration within the given time limit but not solved by Bb within 1200 s.

For propagation-based configurations, the two considered time limits are 1 and 10 s for both scenarios, i.e., within and without a sequential portfolio setting. With a time limit of 10 s, our word-level configuration Pw outperforms Bprop by 473 instances. Considering only instances solved within 1 s, Pw solves 364 more instances than Bprop.

Combining Bb with Pw into Bb+Pw with a time limit of 10 s for the propagation-based configuration Pw improves the overall performance by 67 instances and outperforms Bb+Bprop by 11 instances. Restricting the time limit for Pw to 1 second, Bb+Pw still improves performance by 60 instances, with an even larger gap of 22 instances compared to Bb+Bprop.

In contrast to Bprop, Pw utilizes our new word-level notion of essential inputs for path selection. Restricting Pw to select paths as in [6], i.e., based

[2] http://www.smt-lib.org.

on controlling inputs of bit-level operators only, solves 118 instances less. This shows that the notion of essential inputs indeed allows to prune the search space. Restricting Pw to select inverse values only solves 41 instances less. Selecting consistent values only, on the other hand, performs even worse, with 1456 less solved instances. Prioritizing inverse values shows the best results.

Configuration Pw does not rely on restarts. Introducing restarts as employed in [5,6] does not improve performance and even solves 19 instances less. In case of multi-rooted DAGs, Pw randomly selects a yet unsatisfied root for path propagation, instead of selecting roots based on the heuristics suggested in [5] (and also employed in [6]). Introducing this heuristic in Pw, substantially decreases performance with 323 less solved instances. These observations are consistent with results obtained in SAT competitions by SAT solvers based on the Walk-SAT [39,40] architecture, such as [23]. In terms of Paig, neither restarts nor the root selection heuristic mentioned above have an impact on the number of solved instances or the run time of Paig.

We experimented using different seeds for the random number generator of Pw, where the number of solved instances with a time limit of 10 (1) seconds ranged from 8012 (7596) to 8054 (7649). With a time limit of 10 (1) seconds for Pw the number of solved instances of configuration Bb+Pw ranges from 14869 (14864) to 14874 (14869). Due to space constraints, a statistical analysis on randomization effects is left to future work.

Overall, our new propagation-based technique Pw significantly outperforms all other propagation-based configurations. Further, combined with a bit-blasting engine it considerably improves performance in terms of the number of solved instances, run time, and particularly w.r.t. instances solved within 1 second that Bb was not able to solve even within 1200 s. The number of these uniquely solved instances increased by a factor of 1.6 compared to previous work.

6 Conclusion

We extended and formalized a propagation-based local search procedure for SMT, instantiated for bit-vector logics. Guided by a completeness proof, we avoid brute-force randomization as in earlier work. This not only simplifies the whole approach dramatically, but also improves performance. We further formalized the notion of controlling inputs and backtracing from ATPG and extended it to the word-level, by introducing the new concept of essential inputs.

Our procedure was evaluated on bit-level as well as word-level problems, but can be applied to other than bit-vector logics as well, which is probably the most interesting future work. Extending our techniques by introducing strategies for conflict detection and resolution during backtracing as well as lemma generation in order to obtain an algorithm that is able to prove unsatisfiability and not just satisfiability is another intriguing direction for future work.

Finally we would like to thank Andreas Fröhlich for helpful comments, and Holger Hoos for fruitful discussions on the relation between non-deterministic completeness and the notion of probabilistically approximately complete (PAC).

References

1. Tillmann, N., Schulte, W.: Parameterized unit tests. In: Proceedings of ESEC/SIGSOFT FSE 2005, pp. 253–262. ACM (2005)
2. Yuan, J., Pixley, C., Aziz, A.: Constraint-Based Verification. Springer, Heidelberg (2006)
3. Naveh, Y., Rimon, M., Jaeger, I., Katz, Y., Vinov, M., Marcus, E., Shurek, G.: Constraint-based random stimuli generation for hardware verification. AI Mag. **28**(3), 13–30 (2007)
4. Godefroid, P., Levin, M.Y., Molnar, D.A.: Automated whitebox fuzz testing. In: Proceedings of NDSS 2008. The Internet Society (2008)
5. Fröhlich, A., Biere, A., Wintersteiger, C.M., Hamadi, Y.: Stochastic local search for satisfiability modulo theories. In: Proceedings of AAAI 2015, pp. 1136–1143. AAAI Press (2015)
6. Niemetz, A., Preiner, M., Biere, A., Fröhlich, A.: Improving local search for bit-vector logics in SMT with path propagation. In: Proceedings of DIFTS 2015, pp. 1–10 (2015)
7. Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0. JSAT **9**, 53–58 (2015)
8. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011)
9. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013)
10. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Heidelberg (2014)
11. de Moura, L., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
12. Kroening, D., Strichman, O.: Decision Procedures - An Algorithmic Point of View. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2008)
13. Ganesh, V.: Decision procedures for bit-vectors, arrays and integers. Ph.D. thesis, Stanford University (2007)
14. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 519–531. Springer, Heidelberg (2007)
15. Bruttomesso, R.: RTL verification: from SAT to SMT(BV). Ph.D. thesis, University of Trento (2008)
16. Brummayer, R.: Efficient SMT solving for bit-vectors and the extensional theory of arrays. Ph.D. thesis, Johannes Kepler University Linz (2009)
17. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Hanna, Z., Nadel, A., Palti, A., Sebastiani, R.: A lazy and layered *SMT*(BV) solver for hard industrial verification problems. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 547–560. Springer, Heidelberg (2007)
18. Franzen, A.: Efficient solving of the satisfiability modulo bit-vectors problem and some extensions to SMT. Ph.D. thesis, University of Trento (2010)
19. Hansen, T.A.: A constraint solver and its application to machine code test generation. Ph.D. thesis, University of Melbourne (2012)

20. Hadarean, L., Bansal, K., Jovanović, D., Barrett, C., Tinelli, C.: A tale of two solvers: eager and lazy approaches to bit-vectors. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 680–695. Springer, Heidelberg (2014)
21. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The OpenSMT solver. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 150–153. Springer, Heidelberg (2010)
22. Balint, A., Belov, A., Järvisalo, M., Sinz, C.: Overview and analysis of the SAT challenge 2012 solver competition. Artif. Intell. **223**(2015), 120–155 (2012)
23. Balint, A., Schöning, U.: Choosing probability distributions for stochastic local search and the role of make versus break. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 16–29. Springer, Heidelberg (2012)
24. Balint, A., Belov, A., Heule, M.J.H., Järvisalo, M. (eds.): Proceedings of SAT Competition 2013. Volume B-2013-1 of Department of Computer Science Series of Publications B., University of Helsinki (2013)
25. Belov, A., Heule, M.J.H., Järvisalo, M. (eds.): Proceedings of SAT Competition 2014. Volume B-2014-2 of Department of Computer Science Series of Publications B., University of Helsinki (2014)
26. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. (JAIR) **32**, 565–606 (2008)
27. Kunz, W., Stoffel, D.: Reasoning in Boolean Networks: Logic Synthesis and Verification Using Testing Techniques. Kluwer Academic Publishers, Norwell (1997)
28. Goel, P.: An implicit enumeration algorithm to generate tests for combinational logic circuits. IEEE Trans. Comput. **30**(3), 215–222 (1981)
29. Huang, C., Cheng, K.: Assertion checking by combined word-level ATPG and modular arithmetic constraint-solving techniques. In: Proceedings of DAC 2000, pp. 118–123 (2000)
30. Iyer, M.A.: Race: a word-level atpg-based constraints solver system for smart random simulation. In: Proceedings of ITC 2003, pp. 299–308. IEEE Computer Society (2003)
31. Järvisalo, M., Junttila, T.A., Niemelä, I.: Unrestricted vs restricted cut in a tableau method for boolean circuits. Ann. Math. Artif. Intell. **44**(4), 373–399 (2005)
32. Drechsler, R., Junttila, T.A., Niemelä, I.: Non-clausal SAT and ATPG. In: Handbook of Satisfiability, vol. 185. Frontiers in Artificial Intelligence and Applications, pp. 655–693. IOS Press (2009)
33. Belov, A., Järvisalo, M., Stachniak, Z.: Depth-driven circuit-level stochastic local search for SAT. In: IJCAI, IJCAI/AAAI, pp. 504–509 (2011)
34. Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust boolean reasoning for equivalence checking and functional property verification. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **21**(12), 1377–1394 (2002)
35. Hoos, H.H.: On the run-time behaviour of stochastic local search algorithms for SAT. In: Proceedings of AAAI/IAAI 1999, pp. 661–666. AAAI Press/The MIT Press (1999)
36. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa (2015). www.SMT-LIB.org
37. de Moura, L., Jovanović, D.: A model-constructing satisfiability calculus. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) VMCAI 2013. LNCS, vol. 7737, pp. 1–12. Springer, Heidelberg (2013)

38. Jovanović, D., Barrett, C., de Moura, L.: The design and implementation of the model constructing satisfiability calculus. In: FMCAD, pp. 173–180. IEEE (2013)
39. Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: Proceedings of AAAI 1994, pp. 337–343. AAAI Press/The MIT Press (1994)
40. McAllester, D.A., Selman, B., Kautz, H.A.: Evidence for invariants in local search. In: Proceedings of AAAI/IAAI 1997, pp. 321–326. AAAI Press/The MIT Press (1997)