An Alternative to SAT-based Approaches for Bit-Vectors*

Sébastien Bardin, Philippe Herrmann, and Florian Perroud

CEA LIST, Software Safety Laboratory,
Point Courrier 94, Gif-sur-Yvette, F-91191 France
first.name@cea.fr

Abstract. The theory BV of bit-vectors, i.e. fixed-size arrays of bits equipped with standard low-level machine instructions, is becoming very popular in formal verification. Standard solvers for this theory are based on a bit-level encoding into propositional logic and SAT-based resolution techniques. In this paper, we investigate an alternative approach based on a word-level encoding into bounded arithmetic and Constraint Logic Programming (CLP) resolution techniques. We define an original CLP framework (domains and propagators) dedicated to bit-vector constraints. This framework is implemented in a prototype and thorough experimental studies have been conducted. The new approach is shown to perform much better than standard CLP-based approaches, and to considerably reduce the gap with the best SAT-based BV solvers.

1 Introduction

The first order theory of bit-vectors allows reasoning about variables interpreted over fixed-size arrays of bits equipped with standard low-level machine instructions such as machine arithmetic, bitwise logical instructions, shifts or extraction. An overview of this theory can be found in Chapter 6 of [27]. The bit-vector theory, and especially its quantifier-free fragment (denoted QFBV, or simply BV), is becoming increasingly popular in automatic verification of both hardware [4, 7, 36] and software [10, 11, 14, 15]. Most successful BV solvers (e.g. [3, 5, 24, 25, 40]) rely on encoding the BV formula into an equisatisfiable propositional logic formula, which is then submitted to a SAT solver. The encoding relies on *bit-blasting*: each bit of a bit-vector is represented as a propositional variable and BV operators are modelled as logical circuits. The main advantage of the method is to ultimately rely on the great efficiency of modern DPLL-based SAT solvers [19, 20, 32, 33]. However, this approach has a few shortcomings. First, bit-blasting may result in very large SAT formulas, difficult to solve for the best current SAT solvers. This phenomenon happens especially on "arithmetic-oriented" formulas. Second, the SAT-solving process cannot rely on any information about the word-level structure of the problem, typically missing simplifications such as arithmetic identities. State-of-the-art approaches complement optimised bit-blasting [6, 12, 34] with word-level preprocessing [9, 24] and dedicated SAT-solving heuristics [40].

* Work partially funded by Agence Nationale de la Recherche (grant ANR-08-SEGI-006).

Constraint Logic Programming. Constraint Logic Programming (CLP) over finite domains can be seen as a natural extension of the basic DPLL procedure to the case of finite but non boolean domains, with an interleaving of propagation and search steps [1, 18]. Intuitively, the search procedure explores exhaustively the tree of all partial valuations of variables to find a solution. Before each labelling step, a propagation mechanism narrows each variable domain by removing some inconsistent values. In the following, constraints over bounded arithmetic are denoted by $\mathbb{N}^{\leq M}$. Given a theory T, CLP(T) denotes CLP techniques designed to deal with constraints over T.

Alternative word-level (CLP-based) approach for BV. In order to keep advantage of the high-level structure of the problem, a BV constraint can be encoded into a $\mathbb{N}^{\leq M}$ constraint using the standard (one-to-one) encoding between bit-vectors of size k and unsigned integers less than or equal to $2^k - 1$. A full encoding of BV requires non-linear operators and case-splits [21, 39, 41]. At first sight, CLP($\mathbb{N}^{\leq M}$) offers an interesting framework for word-level solving of BV constraints, since non-linear operations and case-splits are supported. However, there are two major drawbacks leading to poor performance. Firstly, bitwise BV operators cannot be encoded directly and require a form of bit-blasting. Secondly the encoding introduces too many case-splits and non-linear constraints. Recent experiments show that the naive word-level approach is largely outperformed by SAT-based approaches [37]. In the following, we denote by $\mathbb{N}_{BV}^{\leq M}$ bounded integer constraints coming from an encoding of BV constraints.

The problem. Our longstanding goal is to design an efficient word-level CLP-based solver for BV constraints. In our opinion, such a solver could outperform SAT-based approaches on arithmetic-oriented BV problems typically arising in software verification. This paper presents a first step toward this goal. We design new efficient domains and propagators in order to develop a true CLP($\mathbb{N}_{BV}^{\leq M}$) solver, while related works rely on standard CLP($\mathbb{N}^{\leq M}$) techniques [21, 39, 41]. We also deliberately restrict our attention to the conjunctive fragment of BV in order to focus only on BV propagation issues, without having to consider the orthogonal issue of handling formulas with arbitrary boolean skeletons. Note that the conjunctive fragment does have practical interests of its own, for example in symbolic execution [10, 14].

Contribution. We rely on the CLP($\mathbb{N}^{\leq M}$) framework developed in COLIBRI, the solver integrated in the model-based testing tool GaTeL [31].

The main results of this paper are twofold. First, we set up the basic ingredients of a dedicated CLP($\mathbb{N}_{BV}^{\leq M}$) framework, avoiding both bit-blasting and non-linear encoding into $\mathbb{N}^{\leq M}$. The paper introduces two main features: (1) $\mathbb{N}_{BV}^{\leq M}$-propagators for existing domains (union of intervals with congruence [28], denoted Is/C), and (2) a new domain bit-list $\mathcal{BL}$ designed to work in combination with Is/C and $\mathcal{BL}$-propagators. While Is/C comes with efficient propagators on linear arithmetic constraints, $\mathcal{BL}$ is equipped with efficient propagators on "linear" bitwise constraints, i.e. bitwise operations with one constant operand. Second, these ideas have been implemented in a prototype on top of COLIBRI and thorough empirical evaluations have been performed. Experimental results prove that dedicated Is/C-propagators and $\mathcal{BL}$ allow a significant increase of performance compared to a direct CLP($\mathbb{N}^{\leq M}$) approach, as well as

considerably lowering the gap with state-of-the-art SAT-based approaches. Moreover, the CLP($\mathbb{N}_{BV}^{\leq M}$)-based approach scales better than the SAT-based approach with the size of bit-vector variables, and is superior on non-linear arithmetic problems.

Outline. The rest of the paper is structured as follows. Section 2 describes the relevant background on BV and CLP, Sections 4 and 5 presents dedicated propagators and domains, Section 6 presents experimental results and benchmarks. Section 7 discusses related work and Section 8 provides a conclusion.

2 Background

2.1 Bit-vector Theory

Variables in BV are interpreted over bit-vectors, i.e. fixed-size arrays of bits. Given a bit-vector a, its size is denoted by S_a and its i-th bit is denoted by a_i, a_1 being the least significant bit of a. A bit-vector a represents (and is represented by) a unique non-negative integer between 0 and $2^{S_a} - 1$ (power-two encoding) and also a unique integer between -2^{S_a-1} and $2^{S_a-1} - 1$ (two's complement encoding). The unsigned encoding of a is denoted by $[\![a]\!]_u$. Common operators consist of: bitwise operators "and" ($\&$), "or" ($|$), "xor" (xor) and "not" ($\sim$); bit-array manipulations such as left shift ($\ll$), unsigned right shift ($\gg_u$), signed right shift ($\gg_s$), concatenation ($::$), extraction ($a[i..j]$), unsigned and signed extensions ($ext_u(a, i)$ and $ext_s(a, i)$); arithmetic operators ($\oplus$, $\ominus$, $\otimes$, $\oslash_u$, modulo $\%_u$, $<_u$, $\leq_u$, $\geq_u$, $>_u$) with additional constructs for signed arithmetic ($\oslash_s$, $\%_s$, $<_s$, $\leq_s$, $\geq_s$, $>_s$); and a case-split operator $ite(cond, term_1, term_2)$. The exact semantics of all operators can be found in [27]. The following provides only a brief overview. Most operators have their intuitive meaning. Signed extension and signed shift propagate the sign-bit of the operand to the result. Arithmetic operations are performed modulo 2^N, with N the size of both operands. Unsigned (resp. signed) operations consider the unsigned (resp. signed) integer encoding.

Conjunctive fragment. This paper focuses on the conjunctive fragment of BV, i.e. no other logical connector than $\wedge$ is allowed.

2.2 Constraint Logic Programming

Let $\mathcal{U}$ be a set of values. A constraint satisfaction problem (CSP) over $\mathcal{U}$ is a triplet $\mathcal{R} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where the domain $\mathcal{D} \subseteq \mathcal{U}$ is a finite cartesian product $\mathcal{D} = d_1 \times \ldots \times d_n$, $\mathcal{X}$ is a finite set of variables $x_1, \ldots, x_n$ such that each variable x_i ranges over d_i and $\mathcal{C}$ is a finite set of constraints $c_1, \ldots, c_m$ such that each constraint c_i is associated with a set of solutions $L_{c_i} \subseteq \mathcal{U}$. In the following, we consider only the case of finite domains, i.e. $\mathcal{U}$ is finite. The set $L_{\mathcal{R}}$ of solutions of $\mathcal{R}$ is equal to $\mathcal{D} \cap \bigcap_i L_{c_i}$. A value of x_i participating in a solution of $\mathcal{R}$ is called a legal value, otherwise it is said to be spurious. In other words, the set $L_{\mathcal{R}}(x_i)$ of legal values of x_i in $\mathcal{R}$ is defined as the i-th projection of $L_{\mathcal{R}}$. Let us also define $L_c(x_i)$ as the i-th projection of L_c, and $L_{c,\mathcal{D}}(x_i) = L_c(x_i) \cap d_i$. The CLP approach follows a search-propagate scheme. Intuitively, propagation narrows the CSP domains, keeping all legal values of each variable

but removing some of the spurious values. Formally, a propagator P refines a CSP $\mathcal{R} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ into another CSP $\mathcal{R}' = \langle \mathcal{X}, \mathcal{D}', \mathcal{C} \rangle$ with $\mathcal{D}' \subseteq \mathcal{D}$. Only the current domain $\mathcal{D}$ is actually refined, hence we write $P(\mathcal{D})$ for $\mathcal{D}'$. A propagator P is correct (or ensures correct propagation) if $L_{\mathcal{R}}(x_1) \times \ldots \times L_{\mathcal{R}}(x_n) \subseteq P(\mathcal{D}) \subseteq \mathcal{D}$. The use of correct propagators ensures that no legal value is lost during propagation, which in turn ensures that no solution is lost, i.e. $L_{\mathcal{R}'} = L_{\mathcal{R}}$. Usually, propagators are defined locally to each constraint c. Such a propagator P_c is said to be locally correct over domain $\mathcal{D}$ if $L_{c,\mathcal{D}}(x_1) \times \ldots \times L_{c,\mathcal{D}}(x_n) \subseteq P_c(\mathcal{D}) \subseteq \mathcal{D}$. Local correctness implies correctness. A constraint c over domain $\mathcal{D}$ is locally arc-consistent if for all i, $L_{c,\mathcal{D}}(x_i) = \mathcal{D}_i$. This means that from the point of view of constraint c only, there is no spurious value in any d_i. A CSP $\mathcal{R}$ is globally arc-consistent if all its constraints are locally arc-consistent. A propagator is said to ensure local (global) arc-consistency if the resulting CSP is locally (globally) arc-consistent. Such propagators are considered as an interesting trade-off between large pruning and fast propagation.

2.3 Efficient CLP over bounded arithmetic

An interesting class of finite CSPs is the class of CSPs defined over bounded integers ($\mathbb{N}^{\leq M}$). $\mathbb{N}^{\leq M}$ problems coming from verification issues have the particularity to exhibit finite but huge domains. Specific CLP($\mathbb{N}^{\leq M}$) techniques have recently been developed for such problems.

Abstract domains. Domains are not represented concretely by enumeration, they are rather compactly encoded by a symbolic representation allowing efficient (but usually approximated) basic manipulations such as intersection and union of domains or emptiness testing. Even though primarily designed for static analysis, abstract interpretation [13] provides a convenient framework for abstract domains in CLP. An abstract domain ${d^{\#}}_x$ belonging to some complete lattice $(\mathcal{A}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$ is attached to each variable x. This abstract domain defines a set of integers $[\![{d^{\#}}_x]\!]$ that must over-approximate the set of legal values of x, i.e. $L_{\mathcal{R}}(x) \subseteq [\![{d^{\#}}_x]\!]$. The concretisation function $[\![\cdot]\!]$ must satisfy: $a \sqsubseteq b \implies [\![a]\!] \subseteq [\![b]\!]$ and $[\![\bot]\!] = \emptyset$. Given an arbitrary set of integers d, the minimal $\mathcal{A}$-abstraction of d, denoted $\langle d \rangle$, is defined as the least element $d^{\#} \in \mathcal{A}$ such that $d \subseteq [\![d^{\#}]\!]$. The existence of such an element follows from the lattice completeness property. Several abstract domains can be combined with (finite) cartesian product, providing that the concretisation of the cartesian product is defined as the intersection of concretisations of each abstract domain, and that abstract operations are performed in component-wise fashion. Intervals I are a standard abstract domain for $\mathbb{N}^{\leq M}$. The congruence domain C has been recently proposed [28].

In the context of CLP over abstract domains, it is interesting to consider new kinds of consistency. Given a certain class of abstract domains $\mathcal{A}$ and a CSP $\mathcal{R}$ over abstract domains ${d^{\#}}_1, \ldots, {d^{\#}}_n \in \mathcal{A}$, a constraint $c \in \mathcal{R}$ over domain $\mathcal{D}$ is locally $\mathcal{A}$-arc-consistent if for all i, $[\![{d^{\#}}_i]\!] = L_{c,\mathcal{D}}(x_i)$. Intuitively, a propagator ensuring local $\mathcal{A}$-arc-consistency ensures local arc-consistency only for domains representable in $\mathcal{A}$. The constraint c is locally abstract $\mathcal{A}$-arc-consistent if for all i, $[\![{d^{\#}}_i]\!] = [\![\langle L_{c,\mathcal{D}}(x_i) \rangle]\!]$. Intuitively, no more local propagation can be performed for c because of the limited expressiveness of $\mathcal{A}$.

Other features for solving large CLP($\mathbb{N}^{\leq M}$) problems. Other techniques for solving large $\mathbb{N}^{\leq M}$ problems include global constraints to quickly detect unsatisfiability (e.g. global difference constraint [23]) and restricted forms of rewriting rules *(simplification rules)* to dynamically perform syntactic simplifications of the CSP [22]. Note that in that case, the formal framework for propagation presented so far must be modified to allow propagators to add and delete constraints.

3 Encoding BV into Non-Linear Arithmetic

This section describes how to encode BV constraints into non-linear arithmetic problems. First, each bit-vector variable a is encoded as $[\![a]\!]_u$. Then BV constraints over bit-vectors a, b, etc. are encoded as $\mathbb{N}^{\leq M}$ constraints over integer variables $[\![a]\!]_u$, $[\![b]\!]_u$, etc. Unsigned relational operators correspond exactly to those of integer arithmetic, e.g. $a \leq_u b$ is equivalent to $[\![a]\!]_u \leq [\![b]\!]_u$. Unsigned arithmetic operators can be encoded into non-linear arithmetic using the corresponding integer operator and a modulo operation. For example, $[\![a \oplus b]\!]_u = ([\![a]\!]_u + [\![b]\!]_u)\ mod\ 2^N$, with $N = S_a = S_b$. Concatenation of a and b is encoded as $[\![a]\!]_u \times 2^{S_b} + [\![b]\!]_u$. Extraction can be viewed as a concatenation of three variables. Unsigned extension just becomes an equality between (integer) variables. Unsigned left and right shifts with a constant shift argument b are handled respectively like multiplications and divisions by $2^{[\![b]\!]_u}$. Signed operators can be encoded into unsigned operators, using case-splits (ite) based on operand signs (recall that $a \geq_s 0$ iff $a <_u 2^{S_a-1}$). For example, the signed extension $r = ext_s(a, k)$ is encoded as $ite([\![a]\!]_u < 2^{S_a-1}, [\![a]\!]_u, [\![a]\!]_u + 2^k - 2^{S_a})$. Except for the bitwise "not" operation $\sim$ which is efficiently encoded as $[\![\sim x]\!]_u = 2^{S_x} - 1 - [\![x]\!]_u$, encoding other bitwise operations requires a bit-blasting like method. For each BV variable a, this encoding introduces a new boolean variable per bit of a (denoted a_i for bit i), a N-ary consistency constraint relating the a_i to $[\![a]\!]_u$: $\sum_{i=1}^{N} a_i \times 2^{i-1} = [\![a]\!]_u$ and $3N$ ternary constraints over bits of operands and results modelling the bit operation. For example, the "and" operator on a single bit can be encoded with a $\times$ or a min operator.

This direct encoding suffers from at least two drawbacks. First, the size of the encoding of bitwise constraints depends on the number of bits, adding both a linear number of new variables, a linear number of ternary constraints and three N-ary constraints. Second, the encoding introduces many constructs which are not well handled by current CLP($\mathbb{N}^{\leq M}$) solvers, such as case-splits and non-linear operations. Actually, only a very small fragment of BV is encoded in an efficient manner for CLP($\mathbb{N}^{\leq M}$): concatenation, extraction, bitwise not, unsigned shifts and unsigned relational operators. Current state-of-the-art CLP domains and propagators for $\mathbb{N}^{\leq M}$ do not perform well for problems typically coming from BV. For example, considering the constraint $a \oplus 3 = b$ with a and b on 8 bits, domains $d_a = [251..255]$ and $d_b = [0..255]$, a perfect propagation would reduce d_b to $d'_b = [0..2] \cup [254..255]$, thus a perfect interval propagation cannot do better than $d''_b = [0..255]$, i.e. no spurious value is removed, keeping 250 spurious values out of 256 possible values. The same problem occurs with signed operations. It is thus not surprising that common CLP($\mathbb{N}^{\leq M}$) solvers perform very badly on $\mathbb{N}^{\leq M}_{BV}$ problems, as experimentally shown in [37] and confirmed in Section 6.

Our approach. Considering these different issues, we propose the following directions to design an efficient CLP($\mathbb{N}_{BV}^{\leq M}$) framework. First, it seems mandatory to rely on unions of intervals plus congruence (Is/C) rather than single intervals (plus congruence). This is an original point of view in CLP, since COLIBRI [31] is the only CLP solver based on unions of intervals. Second, we propose the two following improvements: (1) the use of original Is/C-propagators designed for BV-constraints instead of relying on combination of existing $\mathbb{N}^{\leq M}$ propagators; and (2) a new domain $\mathcal{BL}$ to efficiently propagate information of bitwise operations without relying on bit-blasting in order to complement Is/C, which is well suited for linear arithmetic. This CLP($\mathbb{N}_{BV}^{\leq M}$) framework works as follows: each variable x has a numerical domain Is/C and a $\mathcal{BL}$ domain, legal values for x being restricted to the intersection of the concretisations of the two domains; each constraint has two associated finite sets of propagators: one for Is/C and one for $\mathcal{BL}$; domains can be synchronised together, i.e. specific propagators are designed to propagate information from one domain to another.

4 Dedicated $\mathbb{N}_{BV}^{\leq M}$-Propagators for Is/C Domains

This section describes dedicated propagators for a CLP($\mathbb{N}_{BV}^{\leq M}$) framework over Is/C domains. The goal is to completely avoid bit-blasting and the introduction of additional case-splits and non-linear constraints at the CLP level.

4.1 Propagators for union of intervals

Propagators for unsigned BV constraints are based on performing modular arithmetic or integer arithmetic operations directly on single intervals, with forward and backward propagation steps. These operations are extended to unions of intervals by distribution over all pairs of intervals. Then, local propagators are defined by interleaving these propagation steps until a local fixpoint is reached. For example, for constraint $A \oplus B = R$ over N bits, the forward propagation step over single interval, denoted $\oplus_I$, is defined by ($\sqcup$ denotes union of intervals with normalisation, without any approximation):

$$[m_1..M_1] \oplus_I [m_2..M_2] = \begin{cases} [m_1+m_2..M_1+M_2] & \text{if } M_1+M_2 < 2^N \\ [m_1+m_2-2^N..M_1+M_2-2^N] & \text{if } m_1+m_2 \geq 2^N \\ [m_1+m_2..2^N-1] \sqcup [0..M_1+M_2-2^N] & \text{otherwise} \end{cases}$$

This definition is extended to unions of intervals $\oplus_{Is}$ by distribution and $\ominus_{Is}$ is defined similarly. Forward and backward propagation steps are defined as follows:

$$\rho_r : ({d^\#}_A, {d^\#}_B, {d^\#}_R) \mapsto ({d^\#}_A, {d^\#}_B, {d^\#}_A \oplus_{Is} {d^\#}_B)$$
$$\rho_a : ({d^\#}_A, {d^\#}_B, {d^\#}_R) \mapsto ({d^\#}_R \ominus_{Is} {d^\#}_B, {d^\#}_B, {d^\#}_R)$$
$$\rho_b : ({d^\#}_A, {d^\#}_B, {d^\#}_R) \mapsto ({d^\#}_A, {d^\#}_R \ominus_{Is} {d^\#}_A, {d^\#}_R)$$

The propagator for $\oplus$ is then defined as a greatest fixpoint of all propagation steps: $\nu X.(\rho_a(X) \sqcap \rho_b(X) \sqcap \rho_r(X) \sqcap X)(X_0)$. Existence follows from the Knaster-Tarski theorem, effective computability comes from Kleene fixed-point theorem and domain finiteness. It can be computed using the procedure presented in Figure 1.

Such propagators and domains are very well-suited to $\oplus$, $\ominus$, unsigned comparisons, unsigned extension and bitwise negation: they ensure local Is-arc consistency for these constraints. For signed operations, the main idea is to perform inside each propagation

procedure propagate-add-is(Is_A, Is_B, Is_R)
1: $(d^{\#}{}_A, d^{\#}{}_B, d^{\#}{}_R) := (Is_A, Is_B, Is_R)$
2: $d^{\#}{}_R := (d^{\#}{}_A \oplus_{Is} d^{\#}{}_B) \sqcap d^{\#}{}_R$;
3: $d^{\#}{}_A := (d^{\#}{}_R \ominus_{Is} d^{\#}{}_B) \sqcap d^{\#}{}_A$;
4: $d^{\#}{}_B := (d^{\#}{}_R \ominus_{Is} d^{\#}{}_A) \sqcap d^{\#}{}_B$;
5: if $(d^{\#}{}_A, d^{\#}{}_B, d^{\#}{}_R) \neq (Is_A, Is_B, Is_R)$ then
6: propagate-add-is($d^{\#}{}_A, d^{\#}{}_B, d^{\#}{}_R$)
7: else return $(d^{\#}{}_A, d^{\#}{}_B, d^{\#}{}_R)$

Fig. 1: Is-propagator for constraint `A ⊕ B = R`

step a case-split based on sign, compute interval propagation for each case and then join all the results. Note that all these computations are performed locally to the propagators, such that no extra variables nor constraints are added at the CLP level. Propagation steps for signed extension are depicted in Figure 2.

procedure Propagator for `exts(A,N') = R`
A: bit-vector of size N, R: bit-vector of size $N' > N$
Propagation steps
$\rho_r : (d^{\#}{}_A, d^{\#}{}_R) \mapsto ((d^{\#}{}_A \sqcap [0..2^{N-1}-1]) \sqcup (d^{\#}{}_A \sqcap [2^{N-1}..2^N-1]) +_{Is} (2^{N'}-2^N), d^{\#}{}_R)$
$\rho_a : (d^{\#}{}_A, d^{\#}{}_R) \mapsto (d^{\#}{}_A, (d^{\#}{}_R \sqcap [0..2^{N-1}-1]) \sqcup$
$(d^{\#}{}_R \sqcap [2^{N-1}+2^{N'}-2^N..2^{N'}-1]) -_{Is} (2^{N'}-2^N))$
propagator: $\nu X.(\rho_a(X) \sqcap \rho_r(X) \sqcap X)(Is_A, Is_R)$.

Fig. 2: Is-propagator for constraint `exts(A,N') = R`

Non-linear arithmetic, concatenation, extraction and shifts can be dealt with in the same way. However only correct propagation is ensured. Propagators for $\&$, $|$ and xor are tricky to implement without bit-blasting. Since $\mathcal{BL}$-propagators (see Section 5) are very efficient for linear bitwise constraints, only coarse but cheap Is-propagators are considered here and the exact computation is delayed until both operands are instantiated. Approximated propagation for $\&$ relies on the fact that $r = a \ \& \ b$ implies both $[\![r]\!]_u \leq [\![a]\!]_u$ and $[\![r]\!]_u \leq [\![b]\!]_u$. The same holds for $|$ by replacing $\geq$ with $\leq$. No approximate Is-propagator for xor is defined, relying only on $\mathcal{BL}$, simplification rules (see Section 4.2) and delayed exact computation.

Property 1 *Is-propagators ensure local Is-arc-consistency for $\oplus$, $\ominus$, comparisons, extensions and bitwise not. Moreover, correct propagation is ensured for non-linear BV arithmetic operators, shifts, concatenation and extraction.*

Efficiency. While unions of intervals are more precise than single intervals, they can in principle induce efficiency issues since the number of intervals could grow up to half of the domain sizes. Note that it is always possible to bound the number of intervals in a domain, adding an approximation step inside the propagators. Moreover, we did not observe any interval blow-up during our experiments (see Section 6).

4.2 Other issues

Simplification rules. These rules perform syntactic simplifications of the CSP [22]. It is different from preprocessing in that the rules can be fired at any propagation step. Rules can be local to a constraint (e.g. rewriting $A \otimes 1 = C$ into $A = C$) or global (syntactic equivalence of constraints, functional consistency, etc.). Moreover, simplification rules may rewrite signed constraints into unsigned ones (when signs are known) and $\mathbb{N}_{BV}^{\leq M}$-constraints into $\mathbb{N}^{\leq M}$-constraints (when presence or absence of overflow is known). The goal of this last transformation is to benefit both from the integer global difference constraint and better congruence propagation on integer constraints.

Congruence domain. Since the new $\mathcal{BL}$ domain can already propagate certain forms of congruence via the consistency propagators (see Section 5), only very restricted C-propagators are considered for BV-constraints, based on parity propagation. However, efficient C-propagation is performed when a BV-constraint is rewritten into a standard integer constraint via simplification. Consistency between congruence domains and interval domains (i.e. all bounds of intervals respect the congruence) is enforced in a standard way with an additional consistency propagator [28].

5 New Domain: BitList $\mathcal{BL}$

This section introduces the BitList domain $\mathcal{BL}$, a new abstract domain designed to work in synergy with intervals and congruences. Indeed, Is/C models well linear integer arithmetic while $\mathcal{BL}$ is well-suited to linear bitwise operations (except for xor), i.e. bitwise operations with one constant operand.

A $\mathcal{BL}$ is a fixed-size array of values ranging over $\{\bot, 0, 1, \star\}$: these values are denoted $\star$-bit in the following. Intuitively, given a $\mathcal{BL}$ $bl = (bl_1, \ldots, bl_N)$, $bl_i = 0$ forces bit i to be equal to 0, $bl_i = 1$ forces bit i to be equal to 1, $bl_i = \star$ does not impose anything on bit i and $bl_i = \bot$ denotes an unsatisfiable constraint. The set $\{\bot, 0, 1, \star\}$ is equipped with a partial order $\sqsubseteq$ defined by $\bot \sqsubseteq 0 \sqsubseteq \star$ and $\bot \sqsubseteq 1 \sqsubseteq \star$. This order is extended to $\mathcal{BL}$ in a bitwise manner. A non-negative integer k is in accordance with bl (of size N), denoted $k \sqsubseteq bl$, if its unsigned encoding on N bits, denoted $[\![k]\!]_{BV}^N$ satisfies $[\![k]\!]_{BV}^N \sqsubseteq bl$. The concretisation of bl, denoted $[\![bl]\!]$, is defined as the set of all (non-negative) integers k such that $k \sqsubseteq bl$. As such, the concretisation of a $\mathcal{BL}$ containing $\bot$ is the empty set. Join (resp. meet) operator $\sqcup$ (resp. $\sqcap$) are defined on $\star$-bits as min and max operations over the complete lattice $(\bot, 0, 1, \star, \sqsubseteq)$, and are extended in a component-wise fashion to $\mathcal{BL}$.

$\mathcal{BL}$-propagators. Precise and cheap propagators can be obtained for all constraints involving only local (bitwise) reasoning, i.e. bitwise operations, unsigned shifts, concatenation, extraction and unsigned extension. They can be solved with N independent fixpoint computation on $\star$-bit variables. $\mathcal{BL}$-propagator for constraint A & B = R is presented in Figure 3, where $\wedge_\star$ extends naturally $\wedge$ over $\star$-bits.

Signed shift and signed extension involve mostly local reasoning, however, non-local propagation steps must be added to ensure that all $\star$-bits of the result representing the sign take the same value, and that signs of operands and results are consistent. As

procedure Propagator for A & B = R
A, B, R bit-vectors of size N
At the $\star$-bit level (a_i, b_i, r_i being $\star$-bit values)
$\rho_r : (a_i, b_i, r_i) \mapsto (a_i, b_i, a_i \wedge_\star b_i)$
$\rho_a : (a_i, b_i, r_i) \mapsto (ite(r_i = 1, 1, ite(b_i = 1, r_i, a_i)), b_i, r_i)$
ρ_b : similar to ρ_a
propagator $\rho_\star$ for $\star$-bit: $\nu X.(\rho_a(X) \sqcap \rho_b(X) \sqcap \rho_r(X) \sqcap X)(X_0)$.
propagator for the constraint: perform $\rho_\star$ in a component-wise manner

Fig. 3: $\mathcal{BL}$-propagator for constraint A & B = R

$\mathcal{BL}$ cannot model equality constraints between unknown $\star$-bit values, these propagators ensure only local abstract $\mathcal{BL}$-arc-consistency. The same idea holds for comparisons. Propagators are simple and cheap: for A $\leq_u$ B, propagate the longest consecutive sequence of 1s (resp. 0s) starting from the most significant $\star$-bit from A to B (resp. B to A). Again, these propagators ensure only local abstract $\mathcal{BL}$-arc-consistency.

Arithmetic constraints involve many non-local reasoning and intermediate results. Moreover backward propagation steps are difficult to define. Thus, this work focuses only on obtaining cheap and correct propagation. Propagators for non-linear arithmetic use a simple forward propagation step (no fixpoint) based on a circuit encoding of the operations interpreted on $\star$-bit values. Propagators for $\oplus$ and $\ominus$ are more precise since they use a complete forward propagation and some limited backward propagation. The $\mathcal{BL}$-propagator for $\oplus$ is depicted in Figure 4. An auxiliary $\mathcal{BL}$ representing the carry is introduced locally to the propagator and the approach relies on the standard circuit encoding for $\oplus$: N local equations $r_i = a_i \; xor \; b_i \; xor \; c_i$ to compute the result, and N non-local equations for carries $c_{i+1} = (a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i)$. Note that the local equations are easy to invert thanks to properties of xor. Information in the $\mathcal{BL}$ is propagated from least significant bit to most significant bit (via the carry). A maximal propagation would require also a propagation in the opposite way. However, experiments show that this alternative is expensive without any clear positive impact. All these operations may appear to be a form of bit-blasting, but the encoding is used only locally to the propagator and no new variables are added.

Property 2 *$\mathcal{BL}$-propagators ensure local $\mathcal{BL}$-arc-consistency for bitwise constraints, unsigned shifts, unsigned extension, concatenation and restriction. $\mathcal{BL}$-propagators ensure local abstract $\mathcal{BL}$-arc-consistency for signed shift, signed extension and all comparisons. Finally, $\mathcal{BL}$-propagators are correct for all arithmetic constraints.*

Ensuring consistency between Is/C and $\mathcal{BL}$. Specific propagators are dedicated to enforce consistency between the numerical domain Is/C and the $\mathcal{BL}$ domain. Let us consider a variable x with domains bl, $I_s = \cup_j [m_j..M_j]$ and congruence (c, M) indicating that $x \equiv c \; mod \; M$. Information can be propagated from $\mathcal{BL}$ to Is/C in two ways, one for intervals and one for congruence. First, it is easy to compute an interval $I_b = [m_b..M_b]$ such that $[\![bl]\!]_u \subseteq I_b$, $m_b \sqsubseteq bl$ and $M_b \sqsubseteq bl$: to compute m (resp. M), just replace all $\star$ values in bl with a 0 (resp. 1). The domain I_s can then be refined to $I_s \sqcap I_b$. Second, if seq is the longest sequence of well-defined (i.e. 0

A, B, R: bitlist
let N be the size of A, B and B
1: $(A', B', R') := (A, B, R)$
2: $C := \star\star\star\ldots\star 0$ /* bit-vector of size N+1 */
3: for $i = 1$ to N do
4: $R'_i := (A'_i \; xor_\star \; B'_i \; xor_\star \; C'_i) \sqcap R'_i$
5: $A'_i := (R'_i \; xor_\star \; B'_i \; xor_\star \; C'_i) \sqcap A'_i$
6: $B'_i := (A'_i \; xor_\star \; R'_i \; xor_\star \; C'_i) \sqcap B'_i$
7: $C'_i := (A'_i \; xor_\star \; B'_i \; xor_\star \; R'_i) \sqcap C'_i$
8: $C'_{i+1} := ((A'_i \wedge_\star B'_i) \vee_\star (A'_i \wedge_\star C'_i) \vee_\star (B'_i \wedge_\star C'_i)) \sqcap C'_{i+1}$.
9: end for
10: return (A', B', R')

Fig. 4: $\mathcal{BL}$-propagator for constraint A $\oplus$ B = R

or 1) least significant $\star$-bits of bl, one can infer a congruence constraint on x such that $x \equiv [\![seq]\!]_u \; mod \, 2^{size(seq)}$. For example, if $bl = \star 1 \star 101$ (on 6 bits), then $x \equiv 5 \, mod \, 8$, and $x \in [21..61]$. Information can also be propagated from intervals and congruences to $\mathcal{BL}$: if (c, M) is such that M is equal to some 2^k then the k least bits of bl can be replaced by the encoding of c on k bits. Moreover, let k' be the smallest integer such that the maximal bound I_M of I satisfies $I_M \leq 2^{k'}$. Then the most significant bits of rank greater than k' of bl must be replaced by 0s. These consistency propagators do not impose that all interval bounds in Is satisfy the $\mathcal{BL}$ constraint. This situation can be detected and it is always possible to increment/decrement the min/max-bound values until a value suiting both Is/C and $\mathcal{BL}$ is reached. However, experiments (not reported in this paper) suggest that it is too expensive to be worthwhile.

6 Experiments

This section presents an empirical evaluation of the techniques developed so far. These experiments have two goals. The first goal (Goal 1) is to assess the practical benefit of the new CLP($\mathbb{N}_{BV}^{\leq M}$) framework, if any, compared to off-the-shelf CLP solvers and straightforward non-linear encoding. To this end, a comparison is performed between non-linear integer encoding for some well-known CLP solvers and a prototype implementing our results. All tools are compared on a common set of search heuristics to evaluate the stability of the results w.r.t. the search heuristic. The second goal (Goal 2) is to compare the current best SAT-based approaches and the best CLP-based approach identified above. We focus on quantifying the gap between the two approaches, comparing the benefits of each approach on different classes of constraints and evaluating scalability issues w.r.t. domain sizes (i.e. bit-width).

CLP($\mathbb{N}_{BV}^{\leq M}$) implementation. COLIBRI is a CLP($\mathbb{N}^{\leq M}$) solver integrated in the model-based testing tool GaTeL [30, 31]. It provides abstract numerical domains (unions of intervals, congruence), propagators and simplification rules for all common arithmetic constraints and advanced optimisations like global difference constraint [23]. COLIBRI is written in Eclipse [2], however it does not rely on the CLP($\mathbb{N}^{\leq M}$) library Eclipse/IC.

Our own prototype is written on top of COLIBRI (version v2007), adding the $\mathcal{BL}$ domain and all $\mathcal{BL}$- and Is/C-propagators described in sections 4 and 5. The following implementation choices have been made: (1) for Is domains the number of intervals is limited to 500; (2) the consistency propagator between Is/C and $\mathcal{BL}$ is approximated: only inconsistent singleton are removed from Is. Four different searches have been implemented (`min`, `rand`, `split`, `smart`). The three first searches are basic dfs with value selection based on the minimal value of the domain (`min`), a random value (`rand`) or splitting the domain in half (`split`). The `smart` search is an enhancement of `min`: the search selects at each step the most constrained variable for labelling ; after one unsuccessful labelling, the variable is put in *quarantine*: its domain is split and it cannot be labelled anymore until all non labelled variables are in quarantine.

Experimental setting. All problems are conjunctive QFBV formulas (including *ite* operators). There are two different test benches. The first one (T1) is a set of 164 problems coming from the standard SMT benchmark repository [38] or automatically generated by the test generation tool OSMOSE [10]. (T1) is intended to compare tool performance on a large set of medium-sized examples. Problems involve mostly 8-bit and 32-bit width bit-vectors and range from small puzzles of a few dozen operators to real-life problems with 20,000 operators and 1,700 variables. (T1) is partitioned into a roughly equal number of bitwise problems, linear arithmetic problems and non-linear arithmetic problems. There are also roughly as many SAT instances as UNSAT instances. The second test bench (T2) is a set of 87 linear and non-linear problems taken from (T1) and automatically extended to bit-width of 64, 128, 256 and 512 (difficulty of the problem may be altered). (T2) is intended to compare scalability on arithmetic constraints w.r.t. the domain size.

Competing tools are described hereafter. Our own prototype comes in 3 versions, depending on domains and propagators used: COL (COLIBRI version v2007 with non-linear encoding), COL-D (COLIBRI v2007 with dedicated Is/C-propagators) and COL-D-BL (COL-D with $\mathcal{BL}$). A new version of COLIBRI (v2009) with better support for non-linear arithmetic is also considered (COL-2009). The other CLP solvers are the standard tools GNU Prolog [17], Eclipse/IC [2], Choco [26] and Abscon [29]. GNU Prolog and Eclipse/IC use single interval domains while Choco and Abscon represent domains by enumeration. GNU Prolog and Eclipse/IC are used with built-in dfs-min, dfs-random and dfs-split heuristics. Choco and Abscon are used with settings of the CLP competition [16]. Selected SAT-based solvers are STP [24] (winner of the 2006 SMT-BV competition [38]), Boolector [3] (winner 2008) and MathSat [5] (winner 2009). We take the last version of each tool.

All experiments were performed on a PC Intel 2Ghz equipped with 2GBytes of RAM. Time out is set up to 20s for (T1) and 50s for (T2).

Results. A problem with all the CLP solvers we have tried except COLIBRI is that they may report overflow exception when domain values are too large: integer values are limited to 2^{24} in GNU Prolog, between 2^{24} and 2^{32} in Choco and Abscon and 2^{53} in Eclipse/IC. In particular, GNU Prolog and ABSCON report many bugs due to overflows in internal computations. Moreover, Choco and Abscon are clearly not designed for large domains and perform very poorly on our examples, confirming previous experimental results [37]. Thus, we report in the following only results of Eclipse/IC. Results

are presented in Table 1 (a) (T1) and (c) (T2). A detailed comparison of COLIBRI-D-BL-smart, STP, Boolector and MathSat can be found in Table 1 (b).

A few remarks about the results. First, Eclipse/IC performs surprisingly better than the standard version of COLIBRI. Actually, the non-linear encoding of BV problems prevents most of the optimisations of COLIBRI to succeed, since they target linear integer arithmetic. However, COLIBRI v2009 with optimised propagators for non-linear arithmetic performs much better than Eclipse/IC. Second, MathSat appears to be less efficient than Boolector and STP, which is rather surprising since it won the 2009 SMT competition. Recall that we consider only conjunctive problems and that our test bench exhibits a large proportion of (non-linear) arithmetic problems.

A few remarks about our implementation. (1) We did not observe any interval blow-up during computation, even when setting up a larger limit (2000 intervals per domain). (2) We have implemented a full consistency propagation between domains Is/C and $\mathcal{BL}$ as described in Section 5: it appears to be less efficient than the restricted consistency propagation described earlier in this section.

Comments. *Goal 1*. It is clear from Table 1 that the CLP($\mathbb{N}_{BV}^{\leq M}$) framework developed so far allows a significant improvement compared to the standard CLP($\mathbb{N}^{\leq M}$) approach with non-linear encoding. Actually, our complete CLP($\mathbb{N}_{BV}^{\leq M}$) solver with `smart` search is able to solve 1.7x more examples in 2.4x less time than Eclipse/IC, and 3x more examples in 3.5x less time than standard COLIBRI. Additional interesting facts must be highlighted:

- Each new feature allows an additional improvement: COL-D-BL performs better than COL-D which performs better than COL. Moreover, this improvement is observed for each of the four heuristics considered here.
- The `smart` search permits an additional gain only when dedicated propagators are used. It does not add anything to the standard version of COLIBRI.
- Every enhanced version of COLIBRI (v2007) performs better than Eclipse/IC and COLIBRI v2009.

Goal 2. According to (T1), global performance of our prototype lies within those of MathSat and STP in both number of successes and computation time, Boolector being a step ahead of the other three tools. Surprisingly, our prototype performs better than the BV-winner 2009, but worse than the BV-winner 2006. We can then conclude that, at least for medium-sized conjunctive problems, CLP can compete with current SAT-based approaches. Considering results by category (Table 1 (b)), our prototype is the best on non-linear UNSAT problems and very efficient on non-linear SAT problems (Boolector solves one more example, but takes 1.5x more time). Finally, considering results from T2 and Table 1 (c), CLP($\mathbb{N}_{BV}^{\leq M}$) scales much better than SAT-based approaches on arithmetic problems: the number of time outs and computation time is almost stable between 64-bit and 512-bit. STP reports very poor scalability. Here, MathSat both performs and scales much better than the other SAT-based tools. Note that due to the automatic scaling of examples, many LA SAT problems are turned into LA UNSAT problems where MathSat is much better.

Tool	Category	Time	# success
Eclipse/IC-min	$\mathbb{N}^{\leq M}$	1760	78/164
Eclipse/IC-rand	$\mathbb{N}^{\leq M}$	2040	72/164
Eclipse/IC-split	$\mathbb{N}^{\leq M}$	1750	79/164
COL-min	$\mathbb{N}^{\leq M}$	2436	43/164
COL-rand	$\mathbb{N}^{\leq M}$	2560	36/164
COL-split	$\mathbb{N}^{\leq M}$	2550	40 /164
COL-smart	$\mathbb{N}^{\leq M}$	2475	40/164
COL-2009-min	$\mathbb{N}^{\leq M}$	1520	89/164
COL-2009-rand	$\mathbb{N}^{\leq M}$	1513	89/164
COL-2009-split	$\mathbb{N}^{\leq M}$	1682	85/164
COL-2009-smart	$\mathbb{N}^{\leq M}$	1410	95/164
COL-D-min	$\mathbb{N}_{BV}^{\leq M}$	1453	94/164
COL-D-rand	$\mathbb{N}_{BV}^{\leq M}$	1392	96/164
COL-D-split	$\mathbb{N}_{BV}^{\leq M}$	1593	89/164
COL-D-smart	$\mathbb{N}_{BV}^{\leq M}$	893	125 /164
COL-D-BL-min	$\mathbb{N}_{BV}^{\leq M}$	1174	108/164
COL-D-BL-rand	$\mathbb{N}_{BV}^{\leq M}$	1116	111/164
COL-D-BL-split	$\mathbb{N}_{BV}^{\leq M}$	1349	103/164
COL-D-BL-smart	$\mathbb{N}_{BV}^{\leq M}$	712	138/164
MathSat	SAT	794	128/164
STP	SAT	618	144/164
Boolector	SAT	291	157/164

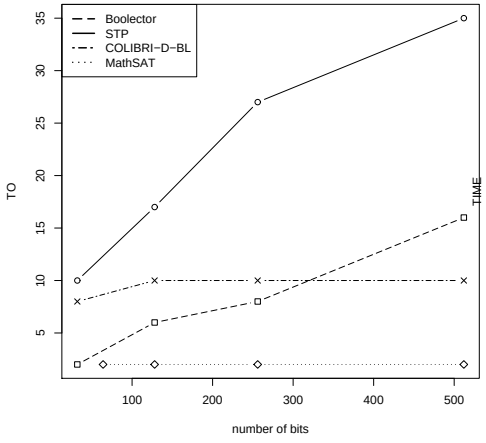
(a) T1: Time and #successes
Time out = 20s

category	COL-D-BL smart	STP	Boolect	MathSat
BW SAT	30 (30/30)	2 (30/30)	0 (30/30)	2 (30/30)
BW UNSAT	3 (30/30)	12 (30/30)	0 (30/30)	4 (30/30)
LA SAT	164 (28/30)	88 (30/30)	9 (30/30)	303 (15/30)
LA UNSAT	360 (7/25)	68 (25/25)	42 (23/25)	223 (16/25)
NLA SAT	148 (23/29)	357 (13/29)	220 (24/29)	221 (18/29)
NLA UNSAT	7 (20/20)	82 (16/20)	20 (20/20)	41 (19/20)
Total	712 (138/164)	589 (145/164)	291 (157/164)	794 (128/164)

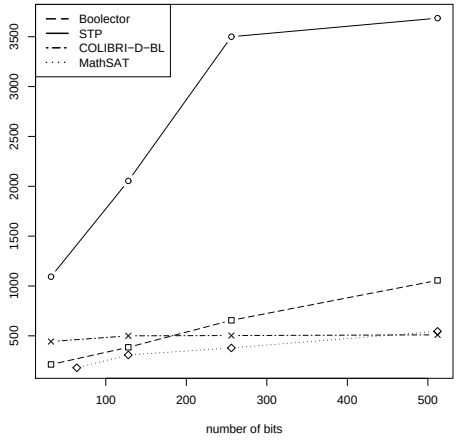
(b) T1: Time and # successes for Time out=20s
(BW: bitwise LA: linear arith. NLA: non-linear arith.)

bit-width	64	128	256	512
COL-D-BL-smart	8 TO, 443s	10 TO, 500s	10 TO, 503s	10 TO, 510s
STP	10 TO, 1093s	17 TO, 2054s	27 TO, 3500s	35 TO, 3686s
Boolector	2 TO, 213s	6 TO, 385s	8 TO, 656s	16 TO, 1056s
MathSat	2 TO, 180s	2 TO, 308s	2 TO, 379s	2 TO, 545s

(c) T2: #TO and time, Time out = 50s



T2: #TO w.r.t. bit-width



T2: Total time w.r.t. bit-width

Table 1. Experimental results

7 Related Work

Word-level BV solving has already been investigated through translations into linear arithmetic with disjunctions [8, 35, 42] or non-linear arithmetic [21, 39, 41]. On the one hand, none of these works consider specific resolution techniques: they all rely on standard approaches for integer arithmetic, i.e. linear integer programming or CLP($\mathbb{N}^{\leq M}$). On the other hand, these encodings require bit-blasting at least for bitwise operations which leads to large formulas. Experiments are performed only with very low bit-width (4 or 8) and no experimental comparison with SAT-based solvers is conducted. The work reported in [7] presents many similarities with this paper. In particular, the authors describe a dedicated domain similar to $\mathcal{BL}$ and they advocate the use of dedicated propagators for domain I (single interval). There are several significant differences with our own work. First, our experiments demonstrate that more elaborated domains are necessary to gain performance. Second, their dedicated domains and propagators are not described, they do not seem to handle signed operations and it is not clear whether or not they rely on bit-blasting for bitwise operations. Moreover, issues such as consistency or efficiency are not discussed. Third, there is no empiric evaluation against other approaches. Finally, experimental results reported in [37] confirm our own experiments concerning SAT-based approaches and traditional CLP($\mathbb{N}^{\leq M}$)-based approaches.

8 Conclusion

Ideas presented in this paper allow a very significant improvement of word-level CLP-based BV solving, considerably lowering the gap with SAT-based approaches and even competing with them on some particular aspects (non-linear BV arithmetic, scalability w.r.t. the domain size). Considering that our implementation relies only on basic searches, we think that this work is a significant step toward the longstanding goal of designing an efficient word-level CLP-based BV solver able to compete with the best SAT-based tools. There is still room for improvement on both the search aspect (learning, intelligent backtracking, etc.) and the propagation aspect (deeper understanding of the trade-off for local propagators, dedicated global propagators, etc.). And there remain many challenging issues: the best SAT-based approaches are still ahead on arbitrary conjunctive QFBV formulas, and formulas with arbitrary boolean skeletons and array operations should be investigated as well. The maturity of our framework is summarised in Table 2.

Acknowledgements. We are very grateful to Bruno Marre and Benjamin Blanc for designing, developing and maintaining the COLIBRI solver, as well as for many insightful comments and advices.

characteristics		SAT-based BV	COLIBRI-D-BL
reasoning level		bit	word
propagation	basic propagation	yes	yes
	propagation trade-off	yes	no
search	variable selection	yes	moderate
	value selection	yes	moderate
	learning	yes	no
	intelligent backtrack	yes	no
supported formulas	array operations	yes	no
	arbitrary boolean connectors	yes	no

Table 2. Maturity of our CLP-based framework for BV

References

1. Apt, K. R.: Principles of Constraint Programming. Cambridge University Press, New York (2003)
2. Apt, K. R., Wallace, M.: Constraint Logic Programming using Eclipse. Cambridge University Press, New York (2007)
3. Brummayer, R., Biere, A.: Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In: TACAS 2009. LNCS, vol. 5505, pp. 174-177. Springer, Heidelberg (2009)
4. Biere, A., Cimatti, A., Clarke, E. M., Zhu, Y.: Symbolic model checking without BDDs. In: TACAS 1999. LNCS, vol. 1579, pp. 193-207. Springer, Heidelberg (1999)
5. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MathSAT 4 SMT Solver. In: CAV 2008. LNCS, vol. 5123, pp. 299-303. Springer, Heidelberg (2008)
6. 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: CAV 2007. LNCS, vol. 4590, pp. 547-560. Springer, Heidelberg (2007)
7. Barray, F., Codognet, P., Diaz, D., Michel, H.: Code-based test generation for validation of functional processor descriptions. In: TACAS 2003. LNCS, vol. 2619, pp. 569-584. Springer, Heidelberg (2003)
8. R. Brinkmann and R. Drechsler: RTL-datapath verification using integer linear programming. In: 15th Int. Conf. on VLSI Design, pp. 741-746. IEEE Computer Society (2002)
9. Barret, C., Dill, D. L., Levitt, J.: A decision procedure for bit-vector arithmetic. In: 35th Design Automation Conf., pp. 522-527. ACM (1998)
10. Bardin, S., Herrmann, P.: Structural Testing of Executables. In: 1st Int. Conf. on Software Testing, Verification, and Validation, pp. 22-31. IEEE Computer Society (2008)
11. Babic, D., Hu, A. J.: Calysto: scalable and precise extended static checking, In: 30th Int. Conf. on Software Engineering, pp. 211-220. ACM (2008)
12. Bryant, R. E., Kroening, D., Ouaknine, J., Seshia, S. A., Strichman, O., Brady, B.: Deciding bit-vector arithmetic with abstraction. In: TACAS 2007. LNCS, vol. 4424, pp. 358-372. Springer, Heidelberg (2007)
13. Cousot, P., Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: 4th ACM Symposium on Principles of Programming Languages, pp. 238-252. ACM (1977)
14. Cadar, C., Ganesh, V., Pawlowski, P. M., Dill, D. L., Engler, D. R.: EXE: automatically generating inputs of death. In: 13th ACM Conf. on Computer and Communications Security, pp. 322-335. ACM (2006)

15. Clarke, E. M., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: TACAS 2004. LNCS, vol. 2988, pp. 168-176. Springer, Heidelberg (2004)
16. CLP competition, http://www.cril.univ-artois.fr/CPAI08/
17. Diaz, D., Codognet, P.: Design and Implementation of the GNU Prolog System. J. Functional and Logic Programming, 2001. EAPLS (2001)
18. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
19. Davis, M., Logemann, G., Loveland, D.: A Machine Program for Theorem Proving. Communications of the ACM 5(7), pp. 394-397 (1962)
20. Davis, M., Putnam, H.: A Computing Procedure for Quantification Theory. Journal of the ACM 7(3), pp. 201-215 (1960)
21. Ferrandi, F., Rendine, M., Sciuto, D.: Functional verification for SystemC descriptions using constraint solving. In: 5th Conf. on Design, Automation and Test in Europe, pp. 744-751. IEEE Computer Society (2002)
22. Frühwirth, T.: Theory and Practice of Constraint Handling Rules. In J. Logic Programming 37(1-3), 95-138 (1998)
23. Feydy, T., Schutt, A., Stuckey, P. J.: Global difference constraint propagation for finite domain solvers. In: 10th Int. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming, pp. 226-236. ACM (2008)
24. Ganesh, V., Dill, D. L.: A Decision Procedure for Bit-Vectors and Arrays. In: CAV 2007. LNCS, vol. 4590, pp. 519-531. Springer, Heidelberg (2007)
25. Jha, S., Limaye, R., Seshia, S. A.: Beaver: Engineering an Efficient SMT Solver for Bit-Vector. In: CAV 2009. LNCS, vol. 5643, pp. 668-674. Springer, Heidelberg (2009)
26. Jussien, N., Rochart, G., Lorca, X.: The CHOCO constraint programming solver. In: CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming.
27. Kroening, D., Strichman, O.: Decision Procedures: An Algorithmic Point of View. Springer, Heidelberg (2008)
28. Leconte, M., Berstel, B.: Extending a CP Solver With Congruences as Domains for Software Verification. In: CP'06 Workshop on Constraints in Software Testing, Verification and Analysis (2006)
29. Lecoutre, C., Tabary, S.: Abscon 112: Toward more Robustness. In CSP Solver Competition, held with CP'08 (2008)
30. Marre, B., Arnould, A.: Test sequences generation from LUSTRE descriptions: GATeL. In: 15th IEEE Inter. Conf. on Automated Software Engineering, pp. 229-240. IEEE Computer Society (2000)
31. Marre, B., Blanc, B.: Test selection strategies for Lustre descriptions in GATeL. Electr. Notes Theor. Comput. Sci. 111, 93-111 (2005)
32. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: 38th Design Automation Conf., pp. 530-535. ACM (2001)
33. Marques-Silva, J., Sakallah, K.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computing, 48(5), pp. 506-521 (1999)
34. Manolios, P., Vroon, D.: Efficient circuit to CNF conversion. In: SAT 2007. LNCS, vol. 4501, pp. 4-9. Springer, Heidelberg (2007)
35. G. Parthasarathy, M. K. Iyer, K. T. Cheng and L. C. Wang: An efficient finite-domain constraint solver for circuits. In: 41th Design Automation Conf., pp. 212-217. ACM (2004)
36. Singerman; E.: Challenges in making decision procedures applicable to industry. In: 3rd Workshop on Pragmatics of Decision Procedures in Automated Reasoning (2005)
37. Sülflow, A., Kühne, U., Wille, R., Große, D., Drechsler, R.: Evaluation of SAT like proof techniques for formal verification of word level circuits. In: 8th IEEE Workshop on RTL and High Level Testing, pp. 31-36. IEEE Computer Society (2007)
38. SMT competition, http://www.smtcomp.org/

39. Vemuri, R., Kalyanaraman, R.: Generation of design verification tests from behavioral VHDL programs using path enumeration and constraint programming. IEEE Transactions on VLSI Systems, 3(2), pp. 201-214 (1995)
40. Wille, R., Fey, G., Große, D., Eggersglüß, S., Drechsler, R.: SWORD: A SAT like prover using word level information. In: 18th Int. Conf. on Very Large Scale Integration of Systems-on-Chip, pp. 88-93. IEEE (2007)
41. Zeng, Z., Ciesielski, M., Rouzeyre, B.: Functional test generation using Constraint Logic Programming. In: 11th Int. Conf. on Very Large Scale Integration of Systems-on-Chip, pp. 375-387. Kluwer (2001)
42. Z. Zeng, P. Kalla and M. Ciesielski: LPSAT: a unified approach to RTL satisfiability. In: 4th Conf. on Design, Automation and Test in Europe, pp. 398-402. ACM (2001)

A Propagators for A $\oplus$ B = R

Simplification rules and parity propagators for A $\oplus$ B = R are presented in Figure 5 and Figure 6. Some of the symmetric cases for A and B are omitted.

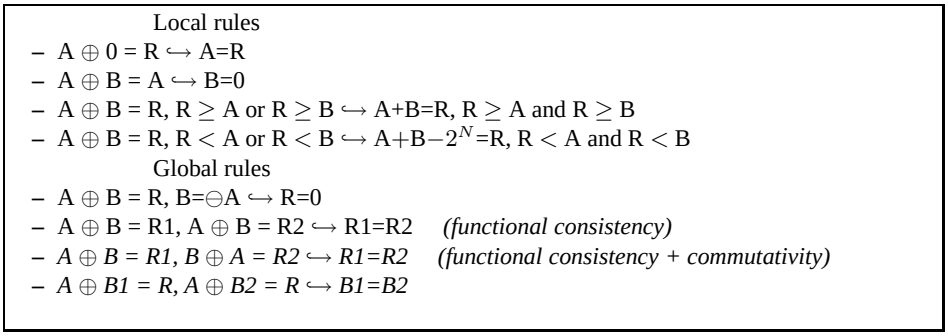


Fig. 5: Simplification rules for constraint A $\oplus$ B = R

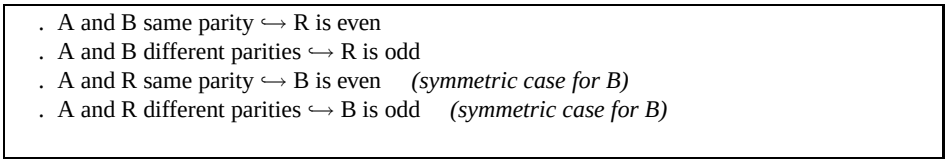


Fig. 6: C-propagators for constraint A $\oplus$ B = R

B Operations on $\star$-bits

The $\wedge_\star$ operation is defined by (q, q_1, q_2 denote $\star$-bit values): $\bot \wedge_\star q = \bot$, $0 \wedge_\star q = 0$, $1 \wedge_\star q = q$, $\star \wedge_\star \star = \star$, $q_1 \wedge_\star q_2 = q_2 \wedge_\star q_1$.

C Experiments with Choco, Abscon and GNU Prolog on small examples

This section reports experimental results obtained with the CLP solvers Abscon [29], Choco [26] and GNU Prolog [17] on a small set of 6 examples parametrised with various bit-vector sizes. Versions considered are the CSP-COMP 2006 version of Abscon (Abscon 109), the CSP-COMP 2008 version of Choco (Choco 2) and GNU Prolog version 1.3.1. The first two tools are launched with the same setting as indicated on the CSP-COMP web site. GNU Prolog is launched with a depth-first search (min value) labelling procedure. Exprimental results are reported in Table 3. Note that when integer domains become too large, overflows or other bugs may happen. In particular, for N=32, these three tools do not manage to answer successfully to any of the constraints.

CLP solver	N=4	N=8	N=12	N=16	N=24
CHOCO	102 (6/7)	112.8 (6/7)	260 (6/7)	418 (3/7)	- (0/7)
ABSCON	1.8 (7/7)	6.1 (7/7)	162 (6/7)	- (0/7)	- (0/7)
GNU Prolog	300 (4/7)	300 (4/7)	300 (4/7)	400 (3/7)	400 (3/7)
Eclipse/IC	0.1 (7/7)	0.04 (7/7)	0.24 (7/7)	90 (7/7)	364 (4/7)
COLIBRI-min	0 (7/7)	0.1 (7/7)	1 (7/7)	28 (7/7)	392 (4/7)

T (x/y): T time in seconds, x: #successful answer, y: total # problems
N: size of the bit-vector variables (in bits)
Time out: 100s

Table 3. Comparison of different CLP solver on integer encoding of BV constraints