Evaluation of Cardinality Constraints on SMT-based Debugging

Andre Sülflow Robert Wille Görschwin Fey Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{*suelflow,rwille,fey,drechsle*}@*informatik.uni-bremen.de*

Abstract

For formal verification of hardware Satisfiability Modulo Theory *(SMT) solvers are increasingly applied. Today's state-of-the-art SMT solvers use different techniques like term-rewriting, abstraction, or bit-blasting. The performance does not only depend on the underlying decision problem but also on the encoding of the original problem into an SMT instance.*

In this work, encodings for cardinality constraints in SMT are investigated. Three different encodings are considered: an adder network, an encoding with multiplexors, and a newly proposed encoding with shifters. The encodings are analyzed with respect to size and complexity. The experimental evaluation on debugging instances that contain cardinality constraints shows the strong influence of the encoding on the resulting run-times.

1. Introduction

Due to the increasing complexity of nowadays circuit designs, there is a need to check the correctness of hardware by formal verification. Today, often *Boolean satisfiability* (Boolean SAT) solvers are applied, that perform quite well on the Boolean level. But when complex data paths and arithmetic operations are considered, the solvers reach their limits.

As an alternative, *Satisfiability Modulo Theory* (SMT) solvers have been developed (see e.g. [4, 7, 15, 3, 9]). In contrast to Boolean SAT, SMT solvers work on a more abstract level, e.g. on arithmetic bit-vectors, that may be used to describe circuits on *Register Transfer Level* (RTL). The problem is given in a mixture of Boolean and word level constraints instead of pure Boolean logic. Internally, this higher level is exploited by optimization techniques like term-rewriting, bit-blasting, or by so called theory solvers, that check the validity of (partial) assignments to a formula. The application of SMT solvers to formal hardware verification problems already showed promising results [25].

For Boolean SAT it is known, that finding a good encoding for a given problem may significantly speed-up the verification process [17]. Much attention has been spent to *cardinality constraints* of the form $s_0+s_1+\ldots+s_{n-1} \leq k$, forcing the arithmetic sum of the Boolean variables s_i to be less or equal than the integer k. For Boolean SAT there exist several studies on the optimal representation of cardinality constraints in general (see e.g. [22, 11, 18]) and with an application to debugging in [23]. These studies show, that the encoding of cardinality constraints is crucial as it may decrease or increase the performance of the solver. However, no similar studies for the SMT domain are available. In particular, the different proof techniques of SMT solvers let suppose a high degree of performance differences.

In this work, encodings for cardinality constraints in SMT instances are considered, theoretically analyzed, and experimentally evaluated on debugging problems. We evaluate two of the commonly known encodings, one is based on multiplexors the other one on an adder structure. These are compared to a newly introduced shifter representation. All encodings are defined in terms of word level operations. Significant trade-offs in the sizes of the different encodings are identified.

Since isolated cardinality constraints are always satisfiable, a concrete application is required to evaluate different encodings. Automated debugging is one particular problem, where a cardinality constraint defines how many failures are suspected in a circuit. The experimental evaluation on debugging instances shows, that the chosen encoding is crucial to achieve high performance. No single encoding is the best for all SMT solvers, but usually one encoding most suitable for a particular SMT solver can be identified. Choosing the wrong encoding may increase the run-time by more than a factor of 2500.

The paper is structured as follows: The next two sections introduce Boolean Satisfiability, SAT Modulo Theories, as well as SMT-based debugging and therewith provide the basis of this work. In Section 4, the SMT encodings of the cardinality constraints are proposed and discussed. An experimental comparison is given in Section 5. The last section concludes the paper.

2. Proof Engines

Due to the recent improvements, *Boolean satisfiability* (Boolean SAT) solvers are quite effective for formal hard-

ware verification. However, arithmetic operations (like addition and multiplication) are known to be a hard problem for Boolean SAT solvers. Thus, new solve techniques are required for the future. *Satisfiability Modulo Theory* (SMT) provides SAT solving on a higher level of abstraction and may cope with the limitations. In the following, we give an overview of Boolean SAT and compare it to SMT in the context of formal hardware verification.

2.1. Boolean Satisfiability (SAT)

The *Boolean Satisfiability problem* (SAT problem) is to determine an assignment for a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ that evaluates to 1 or to prove that no such assignment exists. A SAT problem is called *satisfiable* if a satisfying assignment exists, otherwise it is *unsatisfiable*.

The function f is given in *Conjunctive Normal Form* (CNF). A CNF consists of clauses, each clause consists of literals, and a literal is a variable or its negation. A CNF is satisfied, if all clauses are satisfied. A clause is satisfied if at least one literal evaluates to 1. Circuits can be easily encoded in CNF [26]. Signals are represented by Boolean variables while gates (e.g. AND, OR) or arithmetic operations (ADD, MUL) are encoded in terms of clauses.

Algorithms for solving the SAT problem are called SAT solvers. Most Boolean SAT solvers get a Boolean formula in CNF as input and rely on the DPLL algorithm [6] extended by conflict analysis [19]. Here, (1) a decision heuristic assigns values to free variables, (2) the propagation procedure deduces further assignments, and (3) the conflict analysis determines conflicts and backtracks to resolve a conflict, if necessary.

2.2. Satisfiability Modulo Theory (SMT)

SAT solving on a higher level of abstraction is provided by SMT solvers. A set of assumptions over high level (e.g. bit-vector) constraints represents the satisfiability problem.

Constraints are defined over a set of variables and operations. A variable may be a Boolean predicate, a fixed bitwidth variable, an array or an uninterpreted function. The common input format for SMT [21] defines several theories. In this work, we consider the theory for quantifier free bit-vectors (QF_BV), a suitable format for circuits given on Boolean and word level [25]. Here, Boolean operations (e.g. AND, OR), arithmetic operations (e.g. ADD, MUL), or relations (e.g. LEQ, EQ, GEQ) are supported.

In general, SMT solvers combine a Boolean SAT algorithm with a theory solver resulting in DPLL($\mathcal{T}$) [16, 8]. Conceptually, a SAT solver is called to determine a (partial) satisfiable assignment to an abstraction at the Boolean level that is afterwards checked by the underlying theory solver. If the theory solver determines a conflict, the conflict is passed to the SAT engine, which blocks the assignment and backtracks. However, state-of-the-art SMT solvers do not strictly implement this concept.

Today, several state-of-the-art SMT solvers for QF_BV [4, 7, 15, 3, 9] exist. The most common technique for QF_BV is to simplify the instance in a preprocessing step by using e.g. term rewriting or abstraction, followed by "bit-blasting" (i.e. encoding in CNF and passing it) to a standard SAT solver to determine the satisfiability.

3. Debugging using SMT

Usually, debugging is carried out manually to locate faults in circuits. But using a proof engine this step can be partially automated. Given a circuit and failure traces with expected correct output behavior, the debugging engine automatically determines fault candidates to aid the debugging process.

Before introducing SMT-based debugging, the basics of SAT-based debugging for combinational circuits [23] are briefly revisited.

The input for the debugging algorithm is a faulty circuit and one or more counterexamples (failure traces). Additionally, each counterexample contains information on the correct responses that have to be fulfilled at the primary outputs.

In a first step, the circuit is divided into n components, representing possible fault candidates. The choice of the components controls the granularity of the debugging result. Typical choices are operations (e.g. AND, OR, ADD, MUL) or expressions, but also hierarchical or structural information can be used [12, 13]. For each component, additional *correction logic* is inserted to change the output behavior of the component.

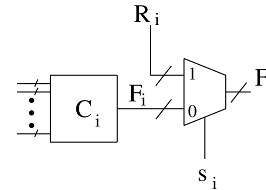


Figure 1. Correction Logic

The correction logic is depicted in Figure 1. The original output F_i of a component C_i is replaced by F_i'. A select line s_i of a multiplexor controls F_i': If s_i is assigned to one, then $F_i' = R_i$, otherwise $F_i' = F_i$. The new variable R_i enables C_i to behave non-deterministically. A correction may be injected at C_i. The variable s_i is called *abnormal predicate* and enables or disables the correction logic, respectively.

Given a combinational circuit $\mathcal{G}$ and a set of m failure traces $X_1, \cdots, X_m$ with corresponding correct output responses $O_1, \cdots, O_m$, the instance is created as shown in Figure 2: For each failure trace, the circuit is duplicated, the correction logic is added, the inputs are restricted to the failure trace, and the outputs are assigned to the correct output response. For each component the same abnormal predicate over all duplicated circuits is used. Thus, if s_i is assigned to one, non-deterministic behavior of component C_i occurs in each duplication simultaneously.

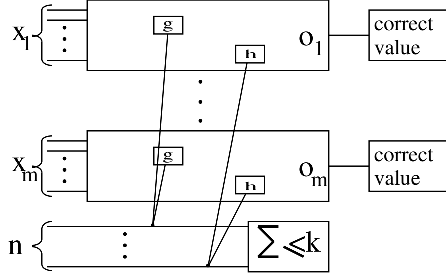


Figure 2. Combinational Debugging

To restrict the number k of enabled abnormal predicates, a *cardinality constraint* is used. Setting k to zero, i.e. disabling the correction logic of all components, leads to an unsatisfiable instance: the correct output responses $O_1, \cdots, O_m$ contradict the failure trace applied to the inputs. Thus, the debugging algorithm increments k by 1, as long as the instance remains unsatisfiable. If the instance becomes satisfiable, a fault candidate can be extracted. A fault candidate is the set of components, where enabling the respective correction logic leads to the correct primary output responses. If exactly one component has to be enabled (i.e. if $k = 1$) a *single fault* is considered. Correcting multiple components ($k > 1$) corresponds to *multiple faults*. The set of all fault candidates can be extracted by incrementally adding constraints that block all already found fault candidates. The algorithm terminates if the instance becomes unsatisfiable. Thus, all fault candidates with minimal cardinality k are extracted.

The debugging instance can be solved using SAT [23] by encoding the described debugging formulation in CNF, i.e. into a set of clauses. But recently also word level debugging has been introduced [20, 24]. Here, the SAT-based debugging approaches for combinational debugging [23] and property debugging [14] are applied to the bit-vector level without modifications. That is, the SMT instance is created on a set of bit-vector constraints instead of translating the circuit into CNF. For word level problems, SMT-based debugging leads to significant run-time improvements in comparison to SAT-based debugging.

4. Cardinality Constraints

While SMT-encodings for circuit elements have already been evaluated in [25] and – for debugging problems – in [24], respectively, representations for cardinality constraints in SMT have not been considered so far. In debugging, a cardinality constraint restricts the maximum number of enabled components to k, i.e. $s_0 + s_1 + \cdots + s_{n-1} \leq k$, with s_i being Boolean variables and k a positive integer. Typically, other types of cardinality constraints like equality, greater, etc. are also used in practice. But these constraints are not further considered here, as they can be encoded analogously.

To ensure an efficient algorithm, "good" encodings of cardinality constraints require good scalability for a large number of components (n) as well as for an increasing k. Especially in case of multiple faults ($k > 1$), the run-time grows significantly and requires efficient encodings.

In the domain of (*Pseudo*) *Boolean SAT*, several studies about finding good encodings for cardinality constraints in *Conjunctive Normal Form* (CNF) exists (see e.g. [1, 11, 22, 2, 18]). Here, encodings based on sequential and parallel counters [22], derived from *Binary Decision Diagrams* (BDD) [5], or adder structures as well as sorter structures [11] have been proposed, respectively. However, the encodings are mostly given on the Boolean level, that may not be efficiently supported by an SMT solver. Thus, the encodings have to be generalized to a set of word level constraints for optimized SMT solving.

In the following we propose three SMT encodings: An adder encoding, a multiplexor encoding (both based on [11]), and a new shifter encoding. Afterwards, the encodings are compared and discussed with respect to their sizes, the scalability for increasing k and n, as well as their arc-consistency.

4.1. Adder Encoding

A straight-forward encoding for a cardinality constraint is the subsequent usage of adder operations for summing up all abnormal predicates s_i and a comparator for restricting the resulting sum to k. In total, this requires $(n-1)$ adders (ADD) and an additional less-equal (LEQ) operation.

However, subsequently using these adders leads to large bit-widths which may degrade the performance of the underlying solve engine (because more values have to be considered). Thus, the respective adders are organized in a tree structure as depicted in Figure 3(a). As a result, $\frac{n}{2}$ adders with a bit-width of 1, $\frac{n}{4}$ adders with a bit-width of 2, etc. are needed. The maximum bit-width using this tree structure is $\lceil \log_2 n \rceil + 1$.

4.2. ITE Encoding

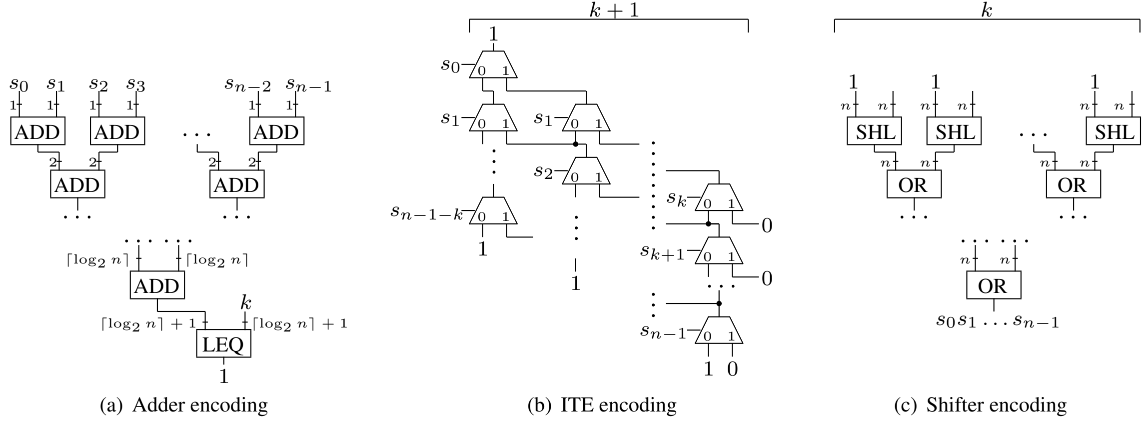
A compact encoding with multiplexors was found by building a BDD [5] of a cardinality constraint [11]. Since each BDD can be easily converted to a multiplexor circuit, this circuit is used to encode the cardinality using multiplexor, i.e. if-then-else (ITE), operations.

Figure 3(b) shows the resulting structure. Starting at the root multiplexor cell, each s_i variable is checked by an ITE operation. If an s_i variable is assigned to 1, multiplexors in the next column are considered. Furthermore, in the rightmost column all data-1 inputs of the multiplexors are assigned to the constant 0. Since additionally the output of the root multiplexor is assigned to 1, a contradiction occurs if more than k variables s_i are assigned to 1.

To construct this structure, for each column $n - k$ ITE operations are required. In total, this leads to $(k+1) \cdot (n-k)$ operations. However, since ITE works with Boolean variables, no word level information can be exploited by the search engine.

4.3. Shifter Encoding

Besides the adder and ITE encoding, which are already known from Boolean satisfiabiliy, finally we introduce a



(a) Adder encoding (b) ITE encoding (c) Shifter encoding

Figure 3. Encoding of Cardinality Constraints

new shifter encoding. The corresponding circuit structure is presented in Figure 3(c). It consists of k shifters of bit-width n. Each shifter shifts a constant 1 within an n-bit bit-vector. ORing the outputs of all shift operations leads to a bit-vector of size n with at most k bits assigned to 1. Constraining the resulting bit-vector to $s_0, \cdots, s_{n-1}$ completes the cardinality constraint. Setting the ith bit of the resulting bit-vector is equivalent to setting s_i to 1. Since at most k constant 1 values can be set using the shifters, the maximum number of s_i variables allowed to be assigned to 1 is restricted to k.

In total, this leads to k shift operations as well as $k - 1$ OR operations. Shifters as well as bit-vector ORs are directly supported in SMT. Similar to the adder encoding, a tree structure is used for the OR operations. Although this has no effect on the bit-width in this case, it is the best encoding with respect to the depth of the structure.

4.4. Discussion

Summing up the worst case numbers of operations needed for the respective encodings leads to the following instance sizes:

1. Adder encoding: $\frac{n}{2} + \frac{n}{4} + \cdots + 1$ adders with bit-widths from 1 to $\lceil \log_2 n \rceil + 1$ (in total $n - 1$ adders) plus an additional LEQ operation.

2. ITE encoding: $(k + 1) \cdot (n - k)$ multiplexors with Boolean inputs and outputs, respectively.

3. Shifter encoding: k shifters and $k - 1$ bit-vector OR operations with bit-width n.

With respect to the number of operations, the shifter encoding is the most compact representation of cardinality constraints (followed by the adder encoding). In contrast, a significantly larger number of operations is required for the ITE encoding. However, this might lead to a stronger implicative power due to *arc-consistency* which is – in principle – possible using the ITE structure.

Arc-consistency means that whenever an assignment could be propagated on the original constraint, the solver‘s implication engine should find that assignment as well. The notion of arc-consistency has been introduced in the CSP domain and applied to Boolean satisfiability [11]. In [11], it has been proven that the ITE encoding is arc-consistent (when additional clauses are added) while e.g. the adder encoding is not. However, these proofs require knowledge about the reasoning engine. In case of Boolean SAT solvers *unit propagation* is the implication procedure. But for SMT solvers knowledge about the implication engine or the bit-blasting procedure is usually not publicly available. Thus, a statement regarding the arc-consistency cannot be given for SMT solvers.

The next section gives an experimental evaluation of the proposed SMT encodings.

5. Experimental Evaluation

The debugging approach has been evaluated on a set of gate level circuits (taken from the LGSynth93 package) as well as on word level benchmarks (taken from SMT-COMP'08[1]). To this end, single as well as multiple faults have been randomly injected into the instances. Counterexamples describing the errors have been generated using a SAT-based equivalence checker.

In this section, the results of the evaluation are reported. The documented run-times are given in CPU seconds. All experiments have been conducted on an AMD Athlon 64 4200+ (2.2GHz) with 2GB main memory running Linux. An MO in the tables denotes a memory out, i.e. requiring more than 2GB main memory.

5.1. Constraint Size

In the following the resulting instance sizes are evaluated. A comparison of the encoding overhead in SAT and

[1]The instances were modified to explicitly model primary in- and outputs (see [24] for details).

Table 1. Encoding Overhead

				SAT				SMT			
	n	#c	k	w/o c. (1.00)	ITE	ADD	SHIFT	w/o c. (1.00)	ITE	ADD	SHIFT
Gate Level Benchmarks											
i7	1190	3	2	39,879	**1.36**	2.17	*4.03*	7,347	1.49	*1.65*	**1.01**
i8	2063	3	3	104,922	**1.31**	1.77	*4.24*	12,627	*1.65*	*1.65*	**1.01**
i9	857	5	2	62,170	**1.17**	1.54	*2.27*	8,895	1.29	*1.39*	**1.01**
k2	676	18	1	430,254	**1.01**	1.06	*1.07*	25,146	1.05	*1.11*	**1.01**
misex3	6261	6	2	449,598	**1.17**	1.54	*2.62*	75,228	1.25	*1.33*	**1.01**
pair	3021	9	5	221,067	**1.33**	1.53	*4.71*	55,611	*1.33*	1.22	**1.01**
rot	1267	2	2	30,730	**1.49**	2.61	*5.17*	5,284	1.72	*1.96*	**1.01**
t481	1647	1	1	18,125	**1.73**	4.55	*5.57*	3,295	2.00	*3.00*	**1.01**
table5	1246	11	3	160,666	**1.12**	1.30	*2.18*	27,577	*1.18*	*1.18*	**1.01**
too_large	2190	3	2	343,605	**1.08**	1.25	*1.70*	13,149	1.50	*1.67*	**1.01**
x1	776	2	2	38,204	**1.24**	1.80	*2.88*	3,174	1.73	*1.98*	**1.01**
x4	1051	3	3	34,923	**1.48**	2.18	*5.60*	6,525	1.64	*1.65*	**1.01**
Word Level Benchmarks											
c-jpg2gif-2138	15474	1	1	4,943,854	**1.03**	1.12	*1.19*	35,891	1.86	*2.72*	**1.01**
c-jpg2gif-2139	15506	1	1	4,946,110	**1.03**	1.12	*1.19*	35,963	1.86	*2.72*	**1.01**
c-jpg2gif-2140	15688	1	1	4,975,819	**1.03**	1.12	*1.20*	36,401	1.86	*2.72*	**1.01**
c-jpg2gif-2141	15739	1	1	4,980,608	**1.03**	1.12	*1.20*	36,516	1.86	*2.72*	**1.01**
c-jpg2gif-1139	38896	1	1	6,956,114	**1.04**	1.22	*1.40*	93,830	1.83	*2.66*	**1.01**
c-jpg2gif-1140	38904	1	1	6,956,744	**1.04**	1.22	*1.40*	93,848	1.83	*2.66*	**1.01**
c-jpg2gif-1141	38947	1	1	7,048,733	**1.04**	1.22	*1.39*	93,950	1.83	*2.66*	**1.01**

SMT is given in Table 1. The columns denote the number of abnormal predicates (n), number of applied counterexamples ($\#c$), the cardinality (k), and the respective encodings. Here, the overhead is given as a factor to the number of clauses (SAT) or number of assumptions (SMT)[2] of the plain instance without cardinality constraint (column w/o c.). The minimal encoding overhead is marked bold, whereas the maximum is marked italic. As in the rest of this section, the respective encodings are denoted by *ADD*, *ITE*, and *SHIFT*.

The syntax of the CNF format and the standard SMT format [21] require additional operations for each of the encodings, causing a larger number of operations compared to the theoretical analysis in Section 4. For example, a constant value or bit accesses have to be encoded in SAT and SMT. Also notable are the additional constraints for the *ADD* encoding. Here, n additional ITE operations of bit-size one are used to transform a Boolean predicate s_i into a bit-vector of size one needed for the ADD operations.

For Boolean SAT, *ITE* requires the least and *SHIFT* the maximum number of additional clauses to represent the cardinality constraint. Especially for *SHIFT*, the overhead is significant and increases the instance size with a factor of up to 5.6. For the word level benchmarks, with large plain instance sizes and large values for n, the overhead becomes more moderate for all encodings.

In SMT the *SHIFT* encoding requires the least and *ADD* the maximum number of additional assumptions. For *ADD* the run-time increases up to 3 times. Here, the overhead for the gate and the word level benchmarks with an increasing n is similar.

The number of counterexamples also influences the overhead. The more counterexamples are applied, the smaller is the overhead of the cardinality constraint. The plain instance size grows with each counterexample, but the number of abnormal predicates n and the cardinality k does not.

Thus, it can be concluded, that the choice of cardinality constraint significantly influences the number of assumptions. Depending on the system used, this might become an issue due to memory limitations.

[2] An assumption in SMT may consists of a set of operations, e.g. an bit-extract in combination with an ADD.

5.2. Run-time

The obtained run-times for the respective encodings are given in Table 2. Here, for each benchmark, the name, the number of components (n), the number of applied counterexamples ($\#c$), and the minimal k are listed in the first four columns. The remaining columns give the run-times obtained by a Boolean SAT solver and three SMT solvers using the described encodings (i.e. *ADD*, *ITE*, and *SHIFT*). As Boolean SAT solver MiniSAT [10] has been applied, while as SMT solver STP [15], Boolector [4], and Z3 (version 2) [7] have been used. All these solvers achieved good results in the respective competitions (i.e. SATCOMP and SMTCOMP) during the last years.

The *ITE* encoding is the best choice if the Boolean SAT solver is applied to the gate level benchmarks. However, for word level instances, with a large number of abnormal predicates, the newly introduced *SHIFT* encoding performed best. In fact, improvements of up to one order of magnitude can be observed in comparison to the *ITE* encoding.

For the SMT solvers, the efficiency of the respective encoding strongly depends on the solver. In more detail, e.g. Z3 degrades up to a factor of 2500 (benchmark *c-jpg2gif-1141*) if the *ITE* encoding is replaced by an *ADD* encoding. If the *SHIFT* encoding is used, the machine may even run out of memory. The solvers apply different proof techniques and pre-processing techniques which explains the performance differences. Nevertheless for each solver one particular encoding is quite robust. STP and Z3 achieve the highest performance when the *ITE* encoding is applied. Boolector handles the encoding with adders most effectively.

The best results for the word level benchmarks are obtained by an SMT solver: Z3 using the *ITE* encoding. In the best case (i.e. for *c-jpg2gif-1141*), an improvement of almost a factor of 20 in comparison to the Boolean SAT solver is achieved.

6. Conclusion

In this work, we evaluated encodings for cardinality constraints in the context of SMT-based debugging. Besides the straight-forward adder encoding and the more optimized ITE encoding (known from SAT-based debugging) also a new shift encoding has been proposed.

The shift encoding is a good alternative for SAT-based debugging on word level benchmarks. For SMT solvers, the efficiency has been observed to be strongly solver dependent. For each SMT solver one particular encoding performs best. The results clearly showed, that it is worthwhile to consider different SMT representations for cardinality constraints as for the word level benchmarks improvements of more than a factor of 2500 can be achieved.

Table 2. Influence on Run-time

	n	#c	k	Minisat ITE	Minisat ADD	Minisat SHIFT	STP ITE	STP ADD	STP SHIFT	Boolector ITE	Boolector ADD	Boolector SHIFT	Z3 ITE	Z3 ADD	Z3 SHIFT
Gate Level Benchmarks															
i7	1190	3	2	**2.27**	8.65	10.70	13.48	54.58	**12.78**	17.79	**4.90**	7.13	**5.54**	38.72	628.07
i8	2063	3	3	**10.36**	37.13	116.76	84.69	288.52	**58.47**	101.37	**12.20**	15.72	**17.74**	262.07	997.18
i9	857	5	2	**3.60**	10.67	23.81	17.19	50.00	**16.69**	21.00	**7.72**	12.19	**16.30**	56.45	614.17
k2	676	18	1	3.88	7.58	**1.74**	**21.45**	103.99	22.22	16.80	**5.64**	5.88	**5.17**	60.76	6.06
misex3	6261	6	2	**177.12**	379.73	412.64	**197.47**	3326.29	283.09	796.40	**35.41**	72.76	**269.42**	2544.89	MO
pair	3021	9	5	**384.59**	848.18	52582.57	**1052.89**	2855.22	4375.90	1061.19	**236.56**	2375.62	**560.83**	1011.39	MO
rot	1267	2	2	**1.75**	9.43	6.34	**11.55**	60.31	12.43	15.32	**3.87**	4.53	**2.53**	35.68	118.23
t481	1647	1	1	0.91	0.96	**0.33**	**2.56**	37.16	5.91	3.07	**1.42**	3.42	**0.39**	18.65	46.53
table5	1246	11	3	**23.33**	28.57	68.02	81.20	231.73	**70.44**	77.64	**19.71**	24.91	**71.98**	224.82	1427.09
too_large	2190	3	2	33.02	92.65	**15.99**	98.46	542.25	**50.88**	205.78	**14.59**	18.93	**279.49**	704.92	MO
x1	776	2	2	**1.23**	3.63	2.00	**7.18**	29.72	7.70	11.11	**3.35**	5.31	**3.41**	14.74	66.88
x4	1051	3	3	**3.37**	5.27	9.03	**18.52**	73.33	19.33	27.13	7.78	**7.45**	**8.74**	32.87	50.90
Word Level Benchmarks															
c-jpg2gif-2138	15474	1	1	182.32	1066.36	**36.72**	**73.29**	MO	MO	1845.98	**28.17**	46.31	**7.82**	2035.40	MO
c-jpg2gif-2139	15506	1	1	349.27	1495.37	**22.47**	**73.65**	MO	MO	1628.84	**29.04**	51.23	**6.67**	2074.50	MO
c-jpg2gif-2140	15688	1	1	232.96	2059.92	**87.97**	**74.80**	MO	MO	1002.77	**28.39**	50.11	**7.20**	1932.68	MO
c-jpg2gif-2141	15739	1	1	219.21	1898.09	**31.65**	**74.53**	MO	MO	2959.95	**29.70**	51.72	**7.31**	2146.10	MO
c-jpg2gif-1139	38896	1	1	**259.27**	10309.90	398.33	MO	MO	MO	11553.50	**76.27**	MO	**19.45**	28796.40	MO
c-jpg2gif-1140	38904	1	1	481.81	5579.12	**132.70**	MO	MO	MO	6562.28	**77.16**	MO	**19.36**	26864.40	MO
c-jpg2gif-1141	38947	1	1	736.51	6421.14	**399.52**	MO	MO	MO	12836.40	**74.81**	MO	**20.19**	50918.20	MO

References

[1] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming*, number 2833 in LNCS, pages 108–122, 2003.

[2] O. Bailleux, Y. Boufkhad, and O. Roussel. A translation of pseudo-boolean constraints to SAT. In *Journal on Satisfiability, Boolean Modeling and Computation*, volume 2, pages 191–200, 2006.

[3] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. The MathSAT 3 System. In *Int'l Conference on Automated Deduction (CADE)*, volume 3632, pages 315–321, 2005.

[4] R. Brummayer and A. Biere. Boolector 0.4. In *SMT-COMP: Satisfiability Modulo Theories Competition*, 2008. Available at http://fmv.jku.at/boolector/.

[5] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[6] M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5:394–397, 1962.

[7] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008. Available at http://research.microsoft.com/projects/Z3.

[8] B. Dutertre and L. Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Computer Aided Verification*, volume 4114, pages 81–94, 2006.

[9] B. Dutertre and L. Moura. The YICES SMT Solver. In *SMT-COMP: Satisfiability Modulo Theories Competition*, 2006. Available at http://yices.csl.sri.com/.

[10] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[11] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. In *Journal on Satisfiability, Boolean Modeling and Computation*, volume 2, pages 1–26, 2006.

[12] M. Fahim Ali, S. Safarpour, A. Veneris, M. Abadir, and R. Drechsler. Post-verification debugging of hierarchical designs. In *Int'l Conf. on CAD*, pages 871–876, 2005.

[13] G. Fey and R. Drechsler. Efficient hierarchical system debugging for property checking. In *IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pages 41–46, 2005.

[14] G. Fey, S. Staber, R. Bloem, and R. Drechsler. Automatic fault localization for property checking. *IEEE Trans. on CAD*, 27(6):1138–1149, 2008.

[15] V. Ganesh and D. L. Dill. A decision procedure for bit-vectors and arrays. In *Computer Aided Verification*, number 4590 in LNCS, pages 524–536, 2007.

[16] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *Computer Aided Verification*, volume 3114 of *LNCS*, pages 175–188, 2004.

[17] A. Hertel, P. Hertel, and A. Urquhart. Formalizing dangerous SAT encodings. In *SAT*, volume 4501 of *LNCS*, pages 159–172, 2007.

[18] J. Marques-Silva and I. Lynce. Towards robust CNF encodings of cardinality constraints. In *Principles and Practice of Constraint Programming*, number 4741 in LNCS, pages 483–497, 2007.

[19] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Comp.*, 48(5):506–521, 1999.

[20] S. Mirzaeian, F. Zheng, and K.-T. T. Cheng. RTL error diagnosis using a word-level SAT-solver. In *Int'l Test Conf.*, pages 1–8, 2008.

[21] S. Ranise and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2006.

[22] C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming*, number 3709 in LNCS, pages 827–831, 2005.

[23] A. Smith, A. Veneris, M. Fahim Ali, and A.Viglas. Fault diagnosis and logic debugging using boolean satisfiability. *IEEE Trans. on CAD*, 24(10):1606–1621, 2005.

[24] A. Sülflow, G. Fey, and R. Drechsler. Experimental studies on SMT-based debugging. In *IEEE Workshop on RTL and High Level Testing*, pages 93–98, 2008.

[25] A. Sülflow, U. Kühne, R. Wille, D. Große, and R. Drechsler. Evaluation of SAT like proof techniques for formal verification of word level circuits. In *IEEE Workshop on RTL and High Level Testing*, pages 31–36, 2007.

[26] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125, 1968. (Reprinted in: J. Siekmann, G. Wrightson (Ed.), Automation of Reasoning, Vol. 2, Springer, Berlin, 1983, pp. 466-483.).