JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

Faculty of Engineering
and Natural Sciences

PBoolector: A Parallel SMT Solver for QF_BV by Combining Bit-Blasting with Look-Ahead

MASTER'S THESIS

submitted in partial fulfillment of the requirements
for the academic degree

Diplom-Ingenieur

in the Master's Program

COMPUTER SCIENCE

Submitted by

Christian Reisenberger, BSc.

At the

Institute for Formal Models and Verifcation

Advisor

Univ.-Prof. Dr. Armin Biere

Co-advisor

Dipl-Ing Aina Niemetz, Bsc
Dipl-Ing Mathias Preiner, Bsc

Linz, November 2014

Abstract

The development of parallel algorithms for the Satisfiability Modulo Theory (SMT) problem is a quite novel research field. Although there are already several parallel approaches for the related Satisfiability (SAT) problem out there, few implementations of parallel SMT solvers exist. In recent years parallelization has grown to an important topic throughout all fields of computer science. The clock frequency increase for new processor generations has reached a limit. Current processor generations are improved by embedding multiple cores using shared memory architecture. In this thesis a parallel SMT algorithm for the quantifier free theory of fixed-size bit-vectors (QF_BV) is proposed and implemented in the tool PBoolector. The Cube and Conquer approach, which was already shown to be a successful parallel SAT solving algorithm, is lifted from bit-level to the SMT term-level. The idea of Cube and Conquer is to split a formula into independent and simpler subproblems which are solved in parallel by a sequential decision procedure. The PBoolector implementation splits a bit-vector formula and solves the subproblems in parallel with the bit-blasting procedure. The search space splitting is based on the so-called look-ahead methods. Look-ahead methods are well researched methods from the SAT area, which are adapted in this thesis for SMT bit-vector formulas. We show that the implemented algorithm can achieve clear improvements (up to super-linear speedup) on a set of hard benchmark files, which are a bottleneck for sequential SMT solvers. The formula structures for which the parallel SMT approach is appropriate will be identified and evaluated in detail.

Zusammenfassung

Die Entwicklung von Parallelen Algorithmen für das Satisfiability Modulo Theory (SMT) Entscheidungsproblem ist ein relativ neues Themengebiet in der heutigen Forschung. Obwohl bereits einige Ansätze zur parallelen Lösung des verwandten Erfüllbarkeitsproblems für Aussagenlogik (SAT) existieren, gibt es wenige Parallelisierungsansätze für SMT. In den letzten Jahren wurde Parallelisierung zu einem wichtigen Thema in sämtlichen Bereichen der Informatik. Der kontinuierliche Anstieg von Taktraten neuer Prozessorgenerationen hat eine Obergrenze erreicht. Prozessoren werden heutzutage verbessert indem mehrere Rechenkerne auf einem Chip integriert werden, welche sich den verfügbaren Speicher teilen. In dieser Arbeit wird ein paralleler SMT Algorithmus für die quantorenfreie Theorie von Bit-Vektoren mit fixer Bitbreite (QF_BV) vorgestellt und im Tool PBoolector implementiert. Der Cube and Conquer Algorithmus, ein erfolgreicher paralleler SAT Lösungsansatz, wird von Bitebene auf die SMT Termebene angehoben. Die Idee von Cube and Conquer besteht darin eine Formel in unabhängige und einfachere Teilprobleme zu teilen welche mit einem sequentiellen Eintscheidungsalgorithmus parallel gelöst werden. Die PBoolector Implementierung teilt eine Bit-Vektor Formel und löst die Teilprobleme parallel mittels Bit-Blasting. Die Teilung des Suchraumes erfolgt über sogenannte Look-Ahead Methoden. Look-Ahead Methoden sind gut erforschte Methoden aus dem SAT Umfeld, welche in dieser Arbeit für SMT Bit-Vektor Formeln adaptiert werden. Wir zeigen, dass der implementierte Algorithmus für eine Menge von harten Aufgabenstellungen, welche Probleme für sequentielle SMT Solver darstellen, klare Verbesserungen (bis hin zu einem superlinearen Speedup) erreicht. Formelstrukturen, welche für den parallelen SMT Ansatz geeignet sind, werden identifziert und im Detail erörtert.

Contents

Chapter 1

Introduction

1.1 Motivation

The Satisfiability (SAT) problem for propositional logic is a very well researched problem in computer science. It was the first problem that was proven to be NP-complete [25]. Therefore, all problems in NP can be transformed in polynomial time into a SAT problem. The SAT problem is to decide for a given propositional formula F whether it is satisfiable (*sat*), or unsatisfiable (*unsat*). If F is *sat*, there exists an assignment (also called interpretation or model) of all variables in F such that F evaluates to *true*. However, if F is *unsat*, there does not exist an assignment for F such that F evaluates to *true*. A first efficient algorithm for solving the problem, the Davis–Putnam–Logemann–Loveland (DPLL) procedure, was proposed in the 1960's [27]. With increasing hardware capabilities, in the recent two decades the SAT problem became applicable for practical applications like hardware and software verification tasks. The ability to solve real world problems lead to a renaissance in this research field, where great improvements have been achieved over the last 20 years [14]. The most powerful approach is the state-of-the-art Conflict Driven Clause Learning (CDCL) algorithm [56], which is based on the DPLL procedure from the 60's .

Propositional logic has a very limited expressive power. For reasoning on many application domains more expressiveness is desirable. A first choice for more expressiveness would be to use first order logic, which is undecidable in general and therefore hard for automated reasoning. First order logic can be restricted by first order theories [7]. First order theories give enough expressiveness to be appropriate for modelling real world application and problems. The theories are useful especially in computer science for verification tasks. Examples of first order theories are the theories of equality, integers, reals, bit-vectors and data structures like lists or arrays. Some examples for bit-vector verification applications are shown in the evaluation chapter (Chap. 5). One of the advantages of some first order theories or fragments of theories, is that satisfiability is

(efficiently) decidable. The Satisfiability Modulo Theory (SMT) problem is the decision problem whether a first order logic formula is satisfiable with respect to some background theory T, i.e., it decides if F is satisfiable modulo theory T. In this thesis, a parallel algorithm for solving the quantifier free fragment of fixed-size bit-vectors (QF_BV) is implemented.

In recent years, the need of parallel algorithms has increased. The increase of processor clock rates every year has reached a limit through physical constraints of the silicon chips. Modern processor generations are improved by embedding multiple cores using a shared memory architecture. This leads to new challenges for SAT and SMT solvers. The Question is how the highly optimized and often inherently sequential algorithms can be adapted to modern multicore architectures. In the SAT field there already exist some approaches [43, 50], whereas parallel SMT solving is a quite novel research field. Only few approaches for parallel SMT solvers have been proposed, yet.

In this thesis a parallel SMT solving approach for QF_BV formulas is implemented. The common solving algorithm for bit-vector formulas, the bit-blasting algorithm, is combined with look-ahead methods, which help to split the formula into several independent subproblems. The resulting subproblems are simpler and can be solved by the sequential bit-blasting algorithm in parallel. To give an idea about the algorithm's basic idea, an example is given in the following section.

1.2 Introductory Example

The basic idea of the solving approach implemented in this thesis is illustrated with the following example. All notations used in the example are informally described. In the preliminary chapter more formal definitions are given. Consider Equ. 1.1 (lazy-if-then-else example motivated by a similar example in [36]). The formula is a first order formula in the background theory of quantifier-free bit-vectors. Using a background theory denotes that a meaning is given to the first order uninterpreted variables, functions and predicates. Variables and constants are numbers of fixed-size bit-vectors, similar to data types in programming languages. For instance the variable z can be an unsigned integer, a bit-vector variable with bit-width 32 representing numbers from 0 to $2^{32} - 1$. Common functions on bit-vectors like "$\cdot$" (arithmetic multiplication) or "$p\ ?\ a : b$" (a so-called if-then-else conditional or ite: if predicate p holds then a else b is returned) are defined.

$$z \cdot (x_0 = y_0\ ?\ y_0 \cdot (x_1 = y_1\ ?\ y_1 : 1) : 1) = \\ (x_0 = y_0\ ?\ x_0 \cdot (x_1 = y_1\ ?\ x_1 \cdot z : z) : z) \quad (1.1)$$

The left part and the right part of the equality in Equ. 1.1 represent two

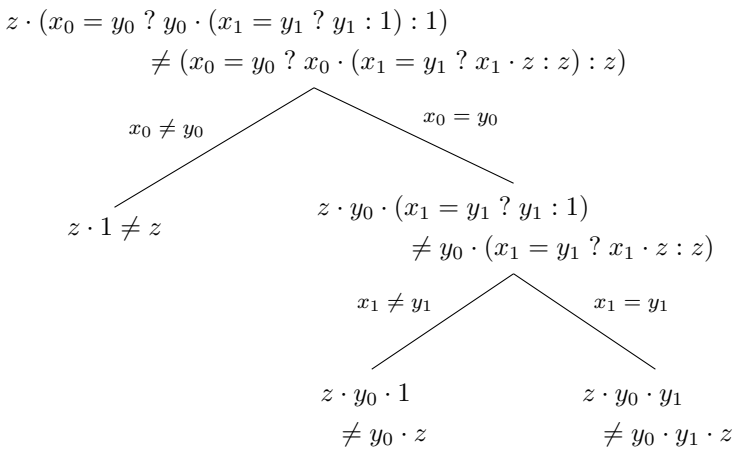


Figure 1.1: Splitting the satisfiability version of Equ. 1.1 into simpler subformulas

different models of implementations in some programming language. The implementation on the right side is an optimization of the implementation on the left side, where the outermost multiplication is propagated to the innermost terms. If we can prove that the two models are equal we can replace the original implementation with the optimized one. This verification method is commonly called equivalence checking. To prove that the two implementations are equal, the validity of Equ. 1.1 has to be shown. A formula is valid iff for all possible variable assignments it evaluates to *true* $(\top, 1)$. Since in practice Satisfiability Modulo Theory solvers are used, the problem needs to be converted to a satisfiability problem. A formula is valid iff the negation of the formula is *unsat*. Thus, to check the validity of Equ. 1.1, it is negated and given to an SMT solver supporting QF_BV. As shown in this thesis, formulas containing multiplications with high bit-width are often very hard to solve in reasonable time for state-of-the-art SMT solvers.

In this thesis an approach is presented to split such hard formulas into several more easy to solve subformulas, which can be solved independently. The generation of smaller and more easy subformulas is shown in Fig. 1.1. On the top of the tree is the original formula to solve which is the negation of Equ. 1.1. First, the formula is split into two subformulas by asserting the predicate $x_0 = y_0$ to be *false* on the left side, and *true* on the right side. Finding the best predicate on which the formula is split is determined by means of a look-ahead method. On the left side, with the knowledge that $x_0 \neq y_0$, the outermost conditionals can be replaced with their "else" branch and the formula can be reduced to $z \cdot 1 \neq z$. This subformula is trivially *unsat*: There is no variable assignment for z such that $z \cdot 1 \neq z$ evaluates to *true*. In other words, any variable $z \cdot 1$ is always

equal to itself. On the right hand side the positive assertion allows to reduce the formula by removing the conditionals with their "if" branch. Further, x_0 can be replaced with y_0 because we asserted $x_0 = y_0$. The resulting reduced subformula can again be split into two smaller subproblems by asserting the remaining predicate $x_1 = y_1$. Applying conditional and variable elimination again, analogously to how it was done after the first split, leads to two trivial subproblems. On the left side, there can be no assignment for z and y_0 such that $z \cdot y_0 \cdot 1 \neq y_0 \cdot z$ evaluates to *true* (*unsat*). The multiplication is commutative. Same argument applies on the right side with result *unsat*. All generated subproblems are *unsat*. Therefore it can be concluded that the original formula is *unsat* and Equ. 1.1 is valid. The two models are equal and the non-optimised implementation can be replaced with the optimized implementation. By making some assumptions and applying some rewriting rules the hard to solve multiplication operations can be removed and trivial subformulas are generated which can be solved quite efficiently.

1.3 Outlook

Chap. 2 briefly introduces SAT algorithm's with a focus on look-ahead solving approaches. The theory of the quantifier free fragment of fixed-size bit-vectors is defined, followed by an explanation of the basic concepts of Boolector[1], an SMT Solver for the quantifier free theories of fixed-size bit-vectors and arrays (QF_ABV), on which the parallel SMT solver is based on. The subsequent Chap. 3 gives an overview for state-of-the-art parallel SMT and SAT solver approaches. Chap. 4 introduces the algorithm implemented in this thesis, beginning from a high level view down to several implementation details. Chap. 5 contains the evaluation results for different benchmark sets. We show for which type of benchmarks the algorithm works and for which not. The last chapter Chap. 6 concludes the thesis and gives an outlook for possible future extensions to improve the algorithm's performance.

Tables and figures which are too big for floating text integration are added in the appendix.

[1] `http://www.fmv.jku.at/boolector`

1.4 Contributions

- In this thesis we show a parallel SMT approach for solving QF_BV formulas by combining conventional bit-vector solving techniques with look-ahead methods. We lift the Cube and Conquer [39] approach from bit-level to the SMT term-level.
- We propose look-ahead methods for bit-vector formulas. With the aid of look-ahead a formula is split into a set of more easy to solve subformulas, which can be solved in parallel.
- A new class of rewriting could be introduced, namely the AIG level rewriting, with the help of the parallel implementation. By a closer investigation of the parallel evaluation results the first AIG rewriting rule was found for Boolector (see Sect. 5.4.3 *problem_14*).
- The algorithm's limitation will be shown. In practice the approach works well for a certain kind of bit-vector formulas. The possible speedup is heavily related to the input formula's structure.
- We propose several optimisation methods to improve the algorithm.
- Formula structures on which the approach works well will be identified and examined in more detail.

Chapter 2

Preliminaries

This chapter will give a definition for bit-vector formulas and a brief introduction into solving concepts on which our parallel algorithm is based on.

2.1 Satisfiability for Propositional Logic

To understand the terms used in the latter sections, here the basic concepts for propositional SAT solving are shown. We will take slightly closer look to look-ahead based SAT solvers. For more details about the topic the reader is referenced to [14].

Given a propositional formula, a SAT solver checks for a propositional formula whether it is satisfiable or unsatisfiable. The majority of state-of-the-art SAT solvers take a propositional formula in Conjunctive Normal Form (CNF) as input, which is defined as follows:

- A formula in CNF is a conjunction of clauses
  $C_1 \wedge C_2 \wedge \ldots \wedge C_n$
- A clause C is a disjunction of literals
  $C = l_1 \vee l_2 \vee \ldots \vee l_m$
- Each literal l is either a Boolean variable in positive phase x or a Boolean variable in negative phase $\neg x$
- A clause is called *unit* if it consists of one literal only.
- A clause C is satisfied if at least one literal in C is assigned to be *true* (for example in clause $\neg x \vee y$, if variable x assigned to be *false* or variable y is assigned to be *true*).

CNF restricts general propositional logic by just using conjunctive and disjunctive connectives. A general propositional formula can be transformed into

an equivalent CNF formula with exponential increase in terms of size in the worst case. To avoid this exponential increase of, a method called Tseitin transformation [57] is used. Any propositional formula can be Tseitin transformed into an equisatisfiable CNF formula. Two formulas f and g are equisatisfiable, if g is satisfiable (unsatisfiable) if and only if f is satisfiable (unsatisfiable). The following equation shows an example of a satisfiable formula in CNF, e.i., it evaluates to *true* with variable assignments $a = true$, $b = false$, $c = false$:

$$F = (a \vee b \vee c) \wedge (\neg a \vee \neg b) \wedge (\neg a \vee \neg c) \tag{2.1}$$

2.1.1 DPLL

The basic DPLL algorithm [14,27] is based on the so called Boolean Constraint Propagation (BCP). If a clause is unit, i.e., it contains only one literal l, l has to be set to *true* to find a satisfying assignment. Then the formula can be reduced by applying following two rules. First, if the unit clause with l is satisfied all clauses containing l are also satisfied and can be removed. Second, the literal $\neg l$ can be removed in each clause where it occurs. The second rule is a consequence of the *unit resolution rule*. New unit clauses can appear in the reduced formula (implied unit clauses). BCP is the process of applying the two rules until fix-point. If no clauses are left (all clauses are removed by the first rule), the formula is satisfiable (*sat*), and if the empty clause is detected (all literals of a clause are removed by the second rule) the formula is unsatisfiable (*unsat*).

The DPPL algorithm starts with BCP on the input formula F. If no result (*sat*/*unsat*) is found by the reduction a decision variable x_{dec} is picked. Then DPLL is recursively called two times, first with x_{dec} assigned to be *true* ($F \wedge x_{dec}$) and second with x_{dec} assigned to be *false* ($F \wedge \neg x_{dec}$). If one recursive call returns *sat*, F is *sat*. If both recursive calls return *unsat* the previous decision is undone (backtracking) and set in the opposite phase. In the unsatisfiable case the algorithm stops after "all" combination of possible assignments where examined.

Consider Equ. 2.1 and assume DPLL makes a decision and selects variable a. In the first recursive call it is assumed to be *true* and added as unit to the formula $F \wedge a$. Then the clause $(a \vee b \vee c)$ is also *true* and can be removed (rule one). By applying rule two, $\neg a$ can be removed from the clauses $(\neg a \vee \neg b)$ and $(\neg a \vee \neg c)$, resulting in a reduced formula with two unit clauses $\neg b$ and $\neg c$. BCP on the two new implied unit clauses leads to the empty formula and we conclude that the formula is *sat*. The satisfying assignment is the combination of the decisions made (a) and the units implied by BCP ($\neg b, \neg c$).

2.1.2 Look-Ahead Based SAT Solvers

Look-ahead based SAT solvers implement a variant of the DPLL algorithm. In DPLL we need to select a variable after BCP and call DPLL recursively. To

make this decision, however, is subject to various heuristical methods. Look-ahead based SAT solvers select decision variables based on a so-called look-ahead procedure [40]. A look-ahead on a formula F with variable x is done as follows: x is added as unit first in positive and then in negative phase to F. BCP is performed for both phases and the reduction is measured. This is done for each variable in F. The variable which leads to the highest reduction is selected as decision variable in the current DPLL step. Finding a decision variable by look-ahead is a costly operation but the DPLL search tree can be pruned by concentrating on making "good" decisions instead of cheap guesses. Empirically, look-ahead solvers exhibit a good performance on hard combinatorial problems.

Heuristics

A *difference* or *distance heuristic* measures the reduction. Usually this is done by counting the number of reduced but not satisfied clauses after BCP. A method used in the OKsolver [52] called *clause reduction heuristic* weights the number of reduced clauses with the number of literals left: Reduced clauses with two literals left are more important and get a higher weight than clauses with three literals left and so forth. More on branching heuristics for selecting a decision variable can be found in [48]. The reduction of a variable in positive and in negative phase have to be combined: A variable is a "good" decision variable if the reduction can be maximised in both phases. An effective heuristic to combine the two reductions is done by multiplication: Select the variable which maximizes the score *positive reduction* $\cdot$ *negative reduction*.

The *direction heuristic* decides after a variable is picked whether the variable is set first in positive or in negative phase in the DPLL search tree. Making a right decision can improve performance on satisfiable instances significantly. If a theoretically perfect heuristic always chooses the right value, no backtracking needs to be done in the DPLL search tree. On unsatisfiable instances both branches would have to be visited anyhow. One strategy implemented in various solvers is to approximate the branch which is more likely to be satisfiable and select this direction first. A simple way to do this is to select the phase which more often occurs in the formula. Select the positive phase first if the formula contains the decision variable $x_{decision}$ more frequently than $\neg x_{decision}$, and otherwise select the negative phase first.

Failed Literals

If the look-ahead on variable x leads to a conflict (BCP leads to an empty clause), x is called a failed literal, and the unit clause $\neg x$ can be added to the formula. If x and $\neg x$ are failed literals the look-ahead procedure is aborted. In this case no solution can be found and the search can be stopped on the current search branch. DPLL backtracks in the search tree by undoing previously made decisions if possible.

Further Enhancements

To reduce the cost of the look-ahead, several extensions to the base algorithm are proposed. Most of these enhancements reduce the "good" decision quality. One enhancement is to use a preselection of variables on which the look-ahead is performed. Another improvement is the so-called Tree Based Look-Ahead [41], which builds up an implication graph and selects those variables first, on which implications are shared. This can efficiently reduce the cost of the look-ahead function.

2.1.3 Conflict Driven Clause Learning (CDCL) Solvers

For completeness sake, we also mention the Conflict Driven Clause Learning approach here (for details see [56]). The method has its strength in solving huge and real world application instances. In contrast look-ahead approaches, CDCL uses a rather cheap heuristic for selecting a decision variable. If a CDCL solver runs into a conflict, the conflict is analyzed and a conflict clause is learned. Conflict clause learning allows non-chronological backtracking, which can effectively prune the DPLL search tree.

2.2 The Theory of Bit-Vectors

In this thesis we consider the quantifier free fragment of the theory of fixed-size bit-vectors QF_BV. As the name implies, there are no (nested) quantifiers allowed for this restricted first order logic. First order theories are commonly defined by its signature Σ and a set of axioms A (see [16]). In the following we define a subset of bit-vector arithmetic by defining the grammar as in [47]:

$$
\begin{aligned}
\mathit{formula} &: \mathit{formula} \wedge \mathit{formula} \mid \neg \mathit{formula} \mid (\mathit{formula}) \mid \mathit{atom} \\
\mathit{atom} &: \mathit{term}\ \mathit{rel}\ \mathit{term} \mid \mathit{Boolean\text{-}variable} \mid \mathit{term}[\mathit{constant}] \\
\mathit{rel} &: < \mid = \\
\mathit{term} &: \mathit{term}\ \mathit{op}\ \mathit{term} \mid \mathit{variable} \mid \sim \mathit{term} \mid \mathit{constant} \mid \mathit{atom}\,?\,\mathit{term} : \mathit{term} \mid \\
&\quad\ \mathit{term}[\mathit{constant} : \mathit{constant}] \\
\mathit{op} &: + \mid - \mid \cdot \mid \gg \mid \ll \mid \& \mid \mid \mid \oplus \mid \circ
\end{aligned}
$$

On the top level, bit-vector formulas can be arbitrary connected by Boolean operators. Common, useful operators are obtained by using a combination of the shown operators (e.g., $a \vee b \equiv \neg(\neg a \wedge \neg b)$ or $a \geq b \equiv \neg(a < b)$). Atoms can be seen as bit-vectors with bit-width 1. Terms are bit-vectors with bit-width ≥ 1. Variables are bit-vector variables with bit-width n, representing natural numbers in a finite domain from 0 to $2^n - 1$. Constants are fixed natural numbers from 0 to $2^n - 1$. Terms can be produced, for instance, with binary functions (*term op term*). Binary arithmetic functions $(+, -, \cdot)$, denote the respective arithmetic operator on bit-vectors. Operator $+$ for instance is the bit-vector

addition with common overflow semantics. The operators $\ll$ and $\gg$ are logical shift left and right. Operators $\&, |, \oplus$ are the logical bit-wise operators AND, OR and XOR, respectively. The operator $\circ$ is the concatenation of two bit-vectors. For example the concatenation of two bit-vectors with bit-width 4 results in a bit-vector with bit-width 8. The unary operator $\sim$ is the logical bit-wise negation of a term. The if-then-else operator $p\ ?\ a : b$ returns term a if atom p is *true*, and term b otherwise. And finally there are slice operators defined such that $a[n]$ returns bit n of term a, and the slice over a range $a[m : n]$ returns a sub-term of a, containing the bits m down to n.

The set of operators defined defined above is a subset of all possible bit-vector operators. It can be extended with any arbitrary bit-vector operator (e.g., arithmetic division or remainder functions, reduction functions like an "AND reduce" which connects each bit of a bit-vector by logical AND and returns a Boolean result, (unsigned/signed) sign-extension functions, ...). More information for SMT theories and standards can be found at the SMT-LIB website[1].

2.3 Boolector

Boolector[2] [18] is an efficient SMT solver developed at the Institute of Formal Models and Verification at Johannes Kepler University Linz. Boolector supports formulas in the theory of QF_BV. Actually it was developed to show an efficient approach which solves bit-vector formulas in combination with the extensional theory of arrays QF_ABV [19]. The theory of arrays especially is interesting for hardware and software verification tasks. Arrays allow to model for instance software memory or hardware caches and do some reasoning on it. The most recent improvements in Boolector were to support lambda expressions [51, 54]. Lambda expressions have a higher expressiveness which allow to model parallel update function on arrays like it is done in the standard C library functions memset() or memcpy(). The latest version (v2.0) of Boolector also supports the theory of equality and uninterpreted functions EUF. Boolector has won several prices at frequently held SMT solving competitions[3]: In 2014 Boolector won the competition in the QF_BV and QF_ABV divisions. In 2012 and 2008 Boolector won the first place for QF_BV and QF_AUFBV divisions.

As input format Boolector supports the SMT-LIB v1 and v2 formats [6], and its own BTOR format [20] an easy to parse input format for bit-vectors and arrays. Boolector provides a C and Python API for direct tool integration.

Internally Boolector represents the formula as s Directed Acyclic Graph (DAG) with structural hashing. That means that nodes with identical sub-

[1] http://smt-lib.org/
[2] http://fmv.jku.at/boolector/
[3] http://smtcomp.org/

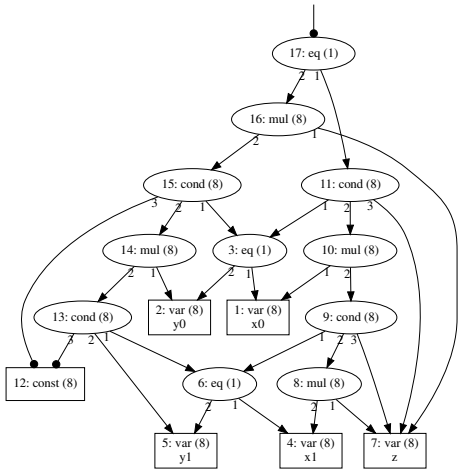


Figure 2.1: Formula 1.1 represented as DAG. Each node contains an unique id and its type. The node's bit-width is notated in the parenthesis. A negated node is indicated with a dot (node 17). The edge values represent the parameter ordering.

structures are shared. Fig. 2.1 shows formula 1.1 with bit-width 8 represented as a DAG. The root node is a Boolean node and represents the whole formula. If there are several roots, each root is an assertion which all have to be satisfied. This is semantically the same as if all root nodes where conjoined with the Boolean $\wedge$ operator. The roots are also called outputs. The leave nodes are variables or constants with bit-width ≥ 1. Variables are the inputs of the formula. The intermediate nodes are arbitrary nested operators defined in the theory. Note that, for instance, conditional 15 and 11 share the same substructure, the predicate $y0 = x0$ (node 3).

As in this thesis, a parallel algorithm for solving formulas in the theory of fixed-size bit-vectors is implemented, in the next sections, the bit-vector solving procedure implemented in Boolector, the so-called bit-blasting approach, will be introduced. More details for solving other SMT theories and their combinations can be found in [16] and [47].

2.3.1 Bit-Blasting

Bit-blasting is the state-of-the-art approach for solving bit-vector formulas. The approach benefits from nowadays available high-performance SAT solvers and is quite efficient in combination with term-level rewriting (see Sect. 2.3.2).

Each bit-vector operation in the formula is encoded as a Boolean circuit. In

n	variables	clauses
8	989	2802
16	4005	1168
32	16181	47874
64	65109	193986
128	261269	781122
256	1046805	3135042

Table 2.1: The size of the CNF problem of Equ. 1.1 with bit width n after Tseitin transformation.

Boolector this is implemented by introducing an intermediate layer, the And Inverter Graph (AIG) layer. An AIG represents a circuit (or Boolean formula) consisting of only AND ($\wedge$) and NOT ($\neg$) gates. In general, AIG implementations also uses structural sharing of isomorphic sub-graphs. In Boolector, each term-level operatior is first encoded as an AIG. For example, an equality node ($x = y$) can be encoded as a circuit with a Boolean output which is *true* iff each bit of x is equal to each bit of y. The circuit of the bit-vector equality node over variables with bit-width 2 is defined as follows (note that the subscript numbers denote the Boolean bit variables of the term at position i):

$$\begin{aligned} x_0 = y_0 \wedge x_1 = y_1 &\Rightarrow \neg(x_0 \oplus y_0) \wedge \neg(x_1 \oplus y_1) \\ &\xRightarrow{toAIG} (\neg(x_0 \wedge \neg y_0) \wedge \neg(\neg x_0 \wedge y_0)) \wedge (\neg(x_1 \wedge \neg y_1) \wedge \neg(\neg x_1 \wedge y_1)) \end{aligned}$$

Next, the AIG circuit is encoded to a propositional formula in CNF via Tseitin transformation. Finally, the resulting CNF formula is fed to the underlying CDCL SAT which determines it to be *sat* or *unsat*.

Arithmetic operators are transformed into circuits which are similar to hardware arithmetic circuits. For example, an add operator is encoded as a chain of full-adder circuits. A multiplication operator is encoded as a shift-and-add multiplier circuit. Fig. B.1 in the appendix shows Equ. 1.1 with bit-width 8 encoded as an AIG. The reason why multiplications are often a bottleneck for bit-blasting is shown in Tab. 2.1, which shows the number of clauses and variables of the CNF representation of Equ. 1.1 (generated by Tseitin transformation), which contains 4 multiplications with bit-width n. With increasing bit-width the multiplication circuits become huge and therefore hard for the SAT solver. For example, Equ. 1.1 in the 256 bit version, the CNF formula contains 1 million variables and 3 million clauses (note that in addition to the exploding CNF size, the symmetry and the connectivity of bit-blasted multiplications is a burden for the SAT solvers).

2.3.2 Term Rewriting / Simplification

As mentioned above bit-blasting can be very powerful in combination with term rewriting (and other simplification techniques). The goal is to reduce the formula already on term-level to keep the later generated CNF problem as small as possible, such that the SAT solver can solve it efficiently. In Boolector, rewriting is divided into 3 levels. Level 1 applies local rewriting rules during formula construction (e.g, $a \wedge \neg a \equiv \bot$). Level 2 and 3 apply global rewriting rules (e.g., term substitutions, arithmetic normalization). Rules of quadratic worst case complexity are bounded by recursion depth [17]. We will illustrate some local and global rewriting rules with the following example. Reconsider one of the subformula $(z \cdot y_0 \cdot (x_1 = y_1 \ ? \ y_1 : 1) \neq y_0 \cdot (x_1 = y_1 \ ? \ x_1 \cdot z : z)) \wedge x_1 = y_1$ produced in introductory example in Fig. 1.1.

$$
\begin{aligned}
(z \cdot y_0 \cdot (x_1 = y_1 \ ? \ y_1 : 1) \neq y_0 \cdot (x_1 = y_1 \ ? \ x_1 \cdot z : z)) \wedge x_1 = y_1 \quad & \textit{conditional elim.} \\
z \cdot (y_0 \cdot y_1) \neq y_0 \cdot (x_1 \cdot z) \wedge x_1 = y_1 \quad & \textit{variable elim.} \\
z \cdot (y_0 \cdot y_1) \neq y_0 \cdot (y_1 \cdot z) \wedge y_1 = y_1 \quad & \textit{equivalence} \\
z \cdot (y_0 \cdot y_1) \neq y_0 \cdot (y_1 \cdot z) \wedge \top \quad & \textit{Boolean rewr.} \\
z \cdot (y_0 \cdot y_1) \neq y_0 \cdot (y_1 \cdot z) \quad & \textit{normalisation} \\
z \cdot (y_0 \cdot y_1) \neq z \cdot (y_0 \cdot y_1) \quad & \textit{equivalence} \\
\bot &
\end{aligned}
$$

The first rule, conditional elimination was already shown in the introductory example: With the knowledge that the conditional's predicate is *true* ($x_1 = y_1$) the conditionals can be eliminated by replacing them with the "if" branch. The second rule, variable elimination, eliminates x_1 by replacing every occurrence of x_1 with y_1 through the knowledge that $x_1 = y_1$. The equivalence rule is implemented with the help of structural node sharing. If a substructure (here y_1) is equal to itself the equality can be replaced with $\top$. The expression $\top$ can be removed with the Boolean rewriting rule $x \wedge \top \equiv x$. In the next line, arithmetic normalisation is applied on the commutative multiplication operation by reordering the variables. And finally the equivalence rule solves the subproblems. "The same substructure is not equal to itself" is always $\bot$.

This example highlights the importance of rewriting. The formula proir to rewriting contains 3 multiplications. Assume that the bit-width of the variables is 256. Without rewriting, the formula would be bit-blasted to a very hard CNF formula containing millions of variables and clauses. Applying some "simple" arithmetic and logical rules, the formula can be solved with result *unsat* already on the term-level, without even generating a Boolean circuit and starting the SAT solver.

2.3.3 Boolector API

This section will introduce the most important functions of the Boolector C API, which in the parallel implementation will be used. The Btor structure represent

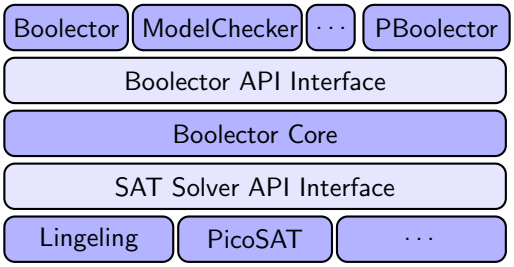


Figure 2.2: The Boolector architecture from a top level view.

an Boolector instance which contains formula in the internal representation.

- boolector_sat(Btor*):int
  First the formula is simplified by preprocessing (including term rewriting), then it is bit-blasted to SAT. It returns the result *sat* (10) if the formula is satisfiable and *unsat* (20) otherwise.

- boolector_limited_sat(Btor*,int):int
  This function behaves similar to boolector_sat(), but the underlying CDCL SAT solver stops after the given conflict limit is reached. The function returns *unknown* (0) if the number of conflicts is exceeded, else it returns the result *sat* or *unsat*.

- boolector_clone(Btor*):Btor*
  Returns an exact but disjunct copy (clone) of a given Boolector instance- This method is important for producing independent subproblems. The search can be independently (in parallel) continued on the clone.

A high-level view of Boolector's internal architecture is shown in Fig. 2.2. On the bottom, a set of SAT solvers can be plugged into Boolector via the SAT solver API interface. The back end SAT solver is exchangeable. Either Lingeling, PicoSAT or any other supported solver can be used. The core of Boolector implements the lemmas on demand approach for lambdas, it maintains the AIG layer and is further responsible for bit-blasting the formula to SAT (see [19, 51, 54])). On top of the Boolector API several tools are implemented. For instance the Boolector front end a standalone SMT solver tool and a model checker implementation are based on the API. The parallel approach implemented in this thesis, namely PBoolector, is a further tool based on the Boolector API.

Chapter 3

Related Work

3.1 Overview

Parallelization is a rather novel research field for SMT solving. For SAT solvers several parallelization approaches have been proposed. An overview on the state-of-the-art in parallel SAT solving is given in [43,50]. SMT solvers use SAT solvers as back end procedure (e.g., Boolector [19], Z3 [28], STP [35]). Simple parallel SMT solvers integrate parallel SAT solver implementations to run on multicore architectures (simplifyingSTP [5]). Parallel SAT/SMT solvers can be roughly classified into three types. The types differ in the level of abstraction where parallelization is introduced. The most abstract level 2 is a competitive approach, whereas level 1 and 0 are cooperative approaches:

- Level 2: The most abstract layer is called portfolio approach. Different sequential algorithms or the same algorithm with different parameters (a portfolio of solvers) are started on the same input formula. The solver instances are run in parallel. The fastest algorithm returns the result and all other running instances are stopped. This simple approach has be shown to be very successful. Different kinds of formulas exhibit quite different performance behaviours on different algorithms or parameter settings. Hence, overall it is likely that one of the started algorithms performs better than others on a given formula.

- Level 1: The second approach at an intermediate level of abstraction is based on search space partitioning. It tries to split the problem into smaller and more easy to solve portions. The subproblems can be solved in parallel by the sequential algorithms.

- Level 0: And finally, the third approach is an attempt to parallelize the solving algorithms. This is a more classical approach of parallelization. In practise, it is quite hard to find appropriate ways to distribute the work of the highly optimized and inherently sequential algorithms. For SAT

solvers, implementations have been proposed which share the work of the most intensive part, the Boolean Constraint Propagation (see Sect. 2.1.1).

Some implementations share information whereas others run completely independent. Sharing learned information introduces synchronisation overhead which possibly has negative influence to the parallelization capabilities. On the other hand, information sharing prevents another worker to perform work which was already done by someone else. Parallel CDCL SAT solver usually exchange learned conflict clauses to share information (see e.g., ManySat [37]).

3.2 Parallel SMT Solvers

There are few approaches which directly address parallelization for SMT algorithms. The introduction of the concurrent track in the SMT competition 2010 was designed to advance the research for parallel implementations [5]. In 2010, two parallel solvers where submitted: The solver test_pmathsat runs several instances of the sequential MathSat 5 [22] solver in parallel with different parameters (portfolio). The solver simplifyingSTP is a variant of the STP [35] solver. It supports QF_BV formulas. First the bit-vector formula is simplified on the term level, then it is converted to propositional logic which is solved by the portfolio based parallel SAT solver ManySat [37]. For the competition 2012 no solvers were submitted to the parallel track [24]. Two more advanced (and well analyzed) parallel SMT implementations using information sharing are introduced in the following sections.

3.2.1 Parallel Z3

One of the few parallel SMT solvers is a parallel version of Z3 [28]. Z3 is an SMT solver supporting a variety of theories. It integrates a DPLL-based SAT solver, different theory solvers and an E-matching abstract machine for quantifiers. The parallel implementation [59] follows a portfolio approach. Different portfolios are shown to solve formulas in the theory of QF_IDL (quantifier free integer difference logic). The first portfolio runs 4 different arithmetic theory solvers. Another portfolio chooses different parameters for the SAT solver. Best speedup was achieved by using the same theory solver but initialising them with different random seeds. Compared to sequential Z3 the portfolio approach achieves an average speedup of up to 3.5 on 4 cores on the selected benchmarks. In the implementation the produced lemmas of the theory solvers are shared over the parallel instances, which was shown to be crucial for good performance.

3.2.2 Parallel Interval Constraint Solver - Picoso

Picoso [45] is a parallel SMT solver for the undecidable theory of reals extended with transcendental functions. It is based on the iSat algorithm [32], a combination of the DPLL search algorithm with interval constraint propagation, enriched with common CDCL methods like conflict learning and non-chronological

backtracking. Initially the real variables are assigned to intervals. To find a satisfying assignment the search space is traversed by splitting a variable's interval into two disjoint intervals. This results into two, more constraint subproblems. New constraints are propagated by updating the remaining variable's intervals. The search space is split until either a solution is found, or each subproblem leads to a conflict. Picoso introduces parallelization by continuing the search (after generating the subproblems) on different machines or CPU cores (workers). Picoso is a typical application of the search space partitioning approach (Level 1). Learned conflicts are exchanged over different workers. Search space splitting is done on demand: If a worker becomes idle a running worker is asked to get one of its unresolved (already split) subproblems. Compared to sequential iSat, Picoso achieves an average speedup on selected benchmark files of ~ 3.8 on a 4 core machine. In [44] an extension for Picoso is proposed, where the search space partitioning approach is combined with portfolios.

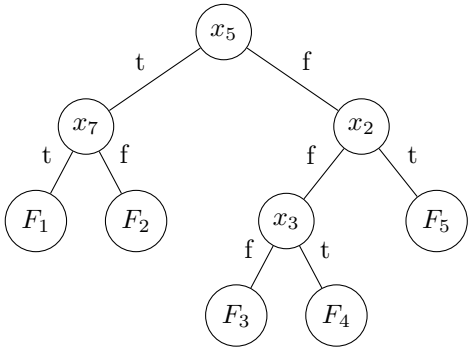
3.3 (Concurrent) Cube and Conquer

Cube and Conquer (CC) [39] is a parallel approach for solving propositional logic formulas. Parallelization is introduced by search space partitioning (level 1) without information sharing. As it is the base concept for the implementation presented in Chap. 4, we will give a more detailed description in the following.

CDCL SAT solvers nowadays are very successful for solving huge industrial instances. They are using inexpensive local heuristics to decide which variable is picked next in the search tree. Look-ahead SAT solvers, on the other hand, are able to solve small hard combinatorial problems. They try to make good and global decisions which variable is picked next for assignment by using more expensive heuristics. CC, introduced by Heule et al. [39] tries to combine the strengths of both methods.

A look-ahead solver is used to split the formula into smaller, more easy to solve subproblems (cube phase). This is done by adding look-ahead decisions in the form of cubes to the formula. A cube is a conjunction of literals. Each literal in the cube is a decision made by the look-ahead procedure. The formula is split into thousands of subproblems which are small enough that they can be solved efficiently by a CDCL procedure (conquer phase). Due to the independence of the subproblems the CDCL solver procedures can be run in parallel. The sum of the subproblem's runtimes should be similar to the runtime of the original problem. The runtime of individual subproblems should be comparable.

Fig. 3.1 shows a simple example of how a CNF formula F is split. Each node represents a subproblem on which the look-ahead procedure makes a decision to assign a variable. On the root node, the original formula F, it is decided to assign x_5 to *true*. In the next step variable x_7 is assigned and so on. If the problem is small enough that it efficiently can be solved with a CDCL procedure a cutoff



$$F_1 := F \wedge (x_5 \wedge \neg x_7)$$
$$F_2 := F \wedge (x_5 \wedge \neg x_7)$$
$$F_3 := F \wedge (\neg x_5 \wedge \neg x_2 \wedge \neg x_3)$$
$$F_4 := F \wedge (\neg x_5 \wedge \neg x_2 \wedge x_3)$$
$$F_5 := F \wedge (\neg x_5 \wedge x_2)$$

Figure 3.1: The lookahead solvers search tree on the left. The leaves shows the lookahead's cut off. The smaller subproblems $F_1 \ldots F_5$ are solved by a CDCL procedure

heuristic decides to stop the look-ahead phase. The resulting leaves F_1, F_2, ..., F_5 represent the conjunction of F with all previous made decisions, which can be solved in parallel by the CDCL solver. If one subproblem evaluates to *sat* the original formula is *sat*. If all subproblems are *unsat* the original formula is *unsat*.

There are two crucial points to make CC work well. First, a good look-ahead heuristic has to be chosen. The look-ahead procedure makes the decision on which variables the formula is split. For the decision heuristic a slightly modified version of the too costly standard look-ahead heuristic is used: Instead of weighting the reduced clauses, the number of assigned variables during the look-ahead is counted. The variable which leads to the most variable assignments during BCP in positive and negative phase is selected first.

Second, a suitable cutoff heuristic has to be chosen. This is a metric indicating that the problem is small enough to stop splitting. For the cutoff, a dynamic metric was proposed, which depends on the number of the made decisions and implied literal assignments during BCP. The solver ILingeling [39] which implements CC, employs a modified version of the march look-ahead solver in combination with the CDCL solver Lingeling [11]. In [39] it was shown that the CC approach can outperform other parallel solvers like ManySat especially on very hard (industrial) SAT instances.

An enhancement of CC was proposed by the Concurrent Cube and Conquer (CCC) approach [58]. One problem of CC is that on many benchmarks it had slower runtimes than a sequential CDCL solver. For CCC it was tried to improve the lack of inappropriate cutoff heuristics. This was done by running a CDCL solver concurrently to the cube phase. After each decision made by the look-ahead solver, a CDCL solver (assuming the new decision) is started in parallel. If the CDCL solver is able to solve the problem before the look-

ahead solver makes a new decision, the current search branch can be closed. Then, look-ahead can continue search on other open search branches. Using this method the search branch is implicitly cutoff if the problem is small enough for CDCL solving. The hard part here is to implement an appropriate backtracking mechanism to continue the look-ahead search in another branch. If the CDCL solver can not solve the problem within the time limit, look-ahead selects a variable to continue splitting. After each split the parallel CDCL solver is restarted with the respective look-ahead decisions. Further, in CCC a metric was introduced to predict if a formula is appropriate for cube and conquer. If the formula seems not appropriate, the predictor aborts the search procedure after a couple of seconds and runs the problem on a sequential CDCL solver. The CCC approach is implemented in the tool CLingeling [10].

The approaches shown in this section are implemented based on the CDCL solver Lingeling[1]. Some further (parallel) tools based on Lingeling [8–12] are mentioned in the following: A reduced version of CLingeling was done in the tool Flegel. Flegel recursively splits the search space with look-aheads and runs a limited CDCL search procedure before doing the next decision. There also exists a portfolio version of Lingeling called PLingeling. The tool Treengeling combines the positive aspects of Lingeling's parallel tool family: PLingeling, Flegel, CLingeling and ILingeling. Mainly the concepts of the Treengeling implementation are used in this work to develop a parallel SMT solver. The details are explained in Chap. 4. The tools from the Lingeling solver family are very efficient and have won several prices over the last years in the annually held SAT competitions[2].

[1] http://fmv.jku.at/lingeling/
[2] http://www.satcompetition.org/

Chapter 4

Implementation

The algorithm proposed in this thesis is mainly motivated by the (Concurrent) Cube and Conquer (CC) approach [39, 58] (see Sect. 3.3). CC is a parallel solving method for propositional formulas by combining look-ahead methods with a Conflict Driven Clause Learning (CDCL) solver. The method was implemented in a tool called Treengeling [12], which is a parallel SAT solver based on the CDCL solver Lingeling. The main goal of this thesis is to adapt the concepts of CC to the SMT world. It further evaluates if look-ahead methods are also applicable in combination with the bit-vector solving procedure of Boolector where powerful term-rewriting rules are combined with bit-blasting.

The implemented tool PBoolector is a parallel SMT solver for QF_BV formulas. It is written in C and build up on the top of the Boolector's C-API. This allows to keep the Boolector implementation unchanged. For missing functionalities needed in PBoolector the API had to be extended. The most important extension is the boolector_lookahead() method, which is explained in more detail in Sect. 4.3. Smaller implemented auxiliary API extensions, needed for the algorithm, are explained directly in the corresponding implementation sections.

A high level view of the basic idea of our algorithm is sketched as follows:

① Try to solve the formula(s) within some given limit.

② If this limit is exceeded, split the formula(s) into smaller subformulas and continue with ①

③ Terminate with *sat* if one of the subproblems is *sat*.

④ Terminate with *unsat* if all subproblems are *unsat*.

The main goal of our algorithm is to split the a formula into smaller and more easy to solve subformulas (subproblems). The decision on how the formula is split is made by boolector_lookahead(). Splitting is continued as long as the

subproblems can not be solved by Boolector's decision procedure within some given limit (c.f. cutoff heuristic in CC). Such a limit could be for instance a time limit, where a formula, which cannot be solved within a certain time, is split into several subproblems. In our actual implementation, we will use the underlying SAT solver's conflict limit as a limit for cutoff. Step ① and ② are repeated until one of the termination criterias ③ or ④ is fulfilled.

4.1 Splitting QF_BV SMT Formulas

To reduce a hard to solve formula into many smaller subproblems, the formula is split into easier to solve subproblems. For example, in the propositional case, Treengeling picks a Boolean variable and assumes it to be *true*/*false* to get two and more simpler subproblems. For SMT formulas in the theory of bit-vectors it is slightly different how partitioning is done. Instead of choosing only variables, PBoolector selects any arbitrary suitable node in the formula's DAG representation as follows.

4.1.1 Boolean Nodes

In the simplest case boolecor_lookahead() selects a Boolean node on which the formula is split by asserting it to be *true* and *false*. Boolean nodes are all operations resulting in a term with bit-width one. This includes all bit-vector operations $(+, \cdot, \sim, ..)$ having operands of bit-width one and all predicates with operands of bit-width ≥ 1 $(=, <, \dots)$. Further, all if-then-else nodes contain a Boolean condition node as first parameter on which the formula can be split. For example in the introductory example (Fig. 1.1) look-ahead decides to select the if-then-else node and split the formula by asserting its condition to be *true* $(x_0 = y_0)$ on one branch and to be *false* $(x_0 \neq y_0)$ on the second branch.

4.1.2 Defining New Predicates

Another option is to select any non-Boolean node and define new predicates on it. Variables or any other arbitrary term with bit-width > 1 can be used to split the formula. Reconsider formula F in the introductory example on top of Fig. 1.1. F could also be split by picking the bit-vector variable x_1 (with bit-width > 1) and asserting a newly defined predicate $x_1 = 0$ on one branch and its negation $x_1 \neq 0$ on the other branch. Note that not only equality, but any other predicate can be used (e.g. $x_1 < 0$ and $x_1 \geq 0$).

4.1.3 Setting Single Bits

Another approach for splitting is to set single bits of arbitrary bit-vector terms. This is especially interesting for hard operations like multiplication. In [49] it was shown that the solving effort can be reduced by setting the least significant

bit (LSB) of a multiplication to *true* in one branch, and to *false* in the second branch.

4.2 Main Algorithm

This section will show our implemented PBoolector algorithm on top of the Boolector API. First the implementation of the main loop will be introduced (Sect. 4.2.1) followed by some more detailed explanation of the procedures used in the main loop (Sect. 4.2.2, Sect. 4.2.3). Finally implementation details for the parallelization strategy are given (Sect. 4.2.4).

4.2.1 Main Loop

A more detailed view of the PBoolector's main loop (step ① - ④ sketched at the beginning of this chapter) is given in List. 4.1 as pseudo-code.

Listing 4.1: The main loop of PBoolector.

```
1  struct sp {
2    F, lkhd, res;
3  };
4
5  pboolector (F) {
6    res = 0;
7    sp[ ] = {(F,0,0)}; //list of subproblems initially containing F
8    search(sp[ ], lim); //initial search
9    while ((res = flush(sp[ ]))==UNKNOWN) {
10     lookahead(sp[ ]); //look−ahead phase
11     split(sp[ ]); //split phase
12     lim = upd(lim);
13     search(sp[ ], lim); //search phase
14   }
15   return res;
16 }
```

Subproblems are organized in a structure sp (line 1). The structure contains a formula instance F, a look-ahead node lkhd and an intermediate result res. The generated subproblems are maintained in the list sp[ ]. Initially the list contains one subproblem, which is the original formula F (line 7). The continuous splitting and searching is implemented in the while loop (line 9-14). The loop consists of following 3 phases: look-ahead (line 10), the split (line 11) and search phase (line 13). Each phase performs the according operation for each subproblem in the list. For instance the search-phase makes a boolector-limited_sat(F,lim) call for each subproblem F in the list sp[ ].

The algorithm starts with an initial limited search on the original formula (line 8). The initial search allows to solve easy instances immediately without splitting the formula. However, if the initial search cannot solve the original

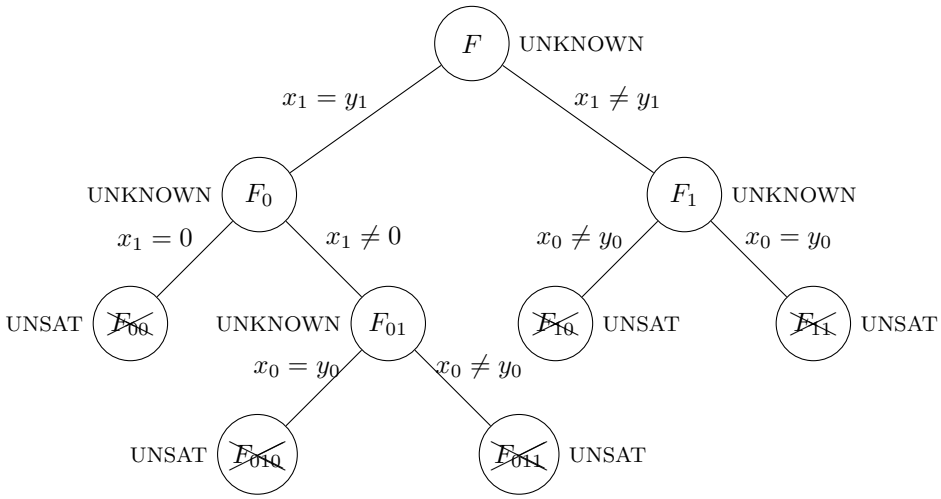


Figure 4.1: Sample execution represented as binary search tree

formula the main loop is entered. In each iteration (round), the subproblems are tried to be solved as follows: First a look-ahead is performed to split the subproblems. The split phase produces for each subproblem, on which look-ahead is performed two new subproblems using the boolector_clone() method. List sp[ ] is updated and now contains all new subproblems after splitting. Next, the limit is dynamically updated. The strategy for updating the limit is shown in Sect. 4.2.3. Finally, the actual work is performed in the search phase, where each subproblem is tried to be solved within a certain limit. The loop's exit condition is implemented in the flush method shown in List. 4.2.

Listing 4.2: The flush method implements the main loop's exit condition.

```
1  flush(sp[ ]) {
2    for (i = 0; i<length(sp[ ]); i++) {
3      if (sp[i].res == UNSAT)
4        removefromlist(sp[ ], i);
5        closed++;
6      else if (sp[i].res == SAT)
7        return SAT;
8    }
9    return length(sp[ ])==0 ? UNSAT : UNKNOWN;
10 }
```

Flush deletes (closes) each solved and unsatisfiable subproblem in list sp[ ]. If sp[ ] is empty it returns *unsat*. If one subproblem is evaluated to *sat* it returns *sat* and *unknown* otherwise to stay in the loop and perform the next round. Note that global variable closed counts the number of deleted nodes for each round. It is used for dynamically updating the limit shown in Sect. 4.2.3.

The execution of the algorithm can be visualized as a binary search tree. A possible example execution to solve the satisfiable version of Equ. 1.1 is shown in Fig. 4.1. The execution is similar to Fig. 1.1, but now with a closer look to the actual implementation. To visualize the expansion of the search tree, different nodes are selected by the look-ahead for splitting. Initially, the original formula F is tried to be solved within the given limit. The search results into *unknown*. No node can be closed by flush() and the loop is entered. The look-ahead on F returns the predicate $x_1 = y_1$. The formula is split by asserting the predicate to be *true* and *false*. The list of subproblems is updated and contains now F_0 and F_1. After updating the limit, in the search phase F_0 and F_1 are tried to be solved without success (result = *unknown*). The method flush() can not close any subproblem and the next loop iteration is performed. In the next iteration F_0 and F_1 are split into 4 subproblems F_{00}, F_{01}, F_{10} and F_{11}. Note that the look-ahead for each subproblem is independent: The look-ahead on F_1 decides to split on the next predicate $x_0 = y_0$ and the look-ahead on F_0 proposes to define a new predicate $x_0 = 0$ ($x_0 \neq 0$) to split the formula. The search phase with the limited sat call is now able to evaluate F_{00}, F_{10} and F_{11} to be *unsat*. The method flush() deletes these three subproblems from the list. There is still one unresolved subproblem F_{01} left. In the next round F_{01} is split into F_{010} and F_{011} which are tried to be solved with result *unsat*. The method flush() deletes F_{010} and F_{011}. The list is empty, flush() aborts the while loop and the result *unsat* is returned from PBoolector.

To keep the algorithm simple it is implemented in a way, such that the search tree is a binary tree. This gives better control over the tree generation (see next section). In general, it would be possible to split a problem into three or more subproblems. For instance on F_0, three new predicates ($x_1 = 0$, $x_1 = 1$ and $x_1 \neq 0 \wedge x_1 \neq 1$) could be defined to do a multiple split. In our current implementation, however, these three subproblems are generated by performing two binary splits resulting in four subproblem where one is trivially *unsat* ($F \wedge x_1 = 0 \wedge x_1 = 1 \equiv \bot$).

4.2.2 Search Tree Generation

In the previous section it was proposed that each subproblem is split into two new subproblems. Generating the search tree in such a breadth-first manner, the increase of subproblems would be exponential if subproblems can not be solved. In practice breadth-first tree generation can lead quite fast to a memory overflow: If an instance of a formula occupies 100MB memory (which is a not unusual value for big (bit-blasted) formulas), doing a binary split until tree depth 10 leads to a very high memory usage of $2^{10} \cdot 100MB = 102.4GB$. On the other hand, generating the tree in a less memory consuming depth-first manner is not appropriate for the algorithm neither. By splitting only one subproblem each round, less formulas are generated. The problem here is that the list of subproblems will contain formulas with very different workloads for solving. One formula would be split several times whereas other formulas are not split

at all. Different workload for solving would be bad for the later parallelization (Sect. 4.2.4). We want to produce several subproblems each round with similar solving effort which should be solved in parallel. Hence the actual implementation combines both, the breath-first and depth-first method. The search tree is generated in a depth-first manner but with some breadth. Each round a limited number (maxactive) of subproblems is produced. The following listings examine the methods used in the main algorithm (List. 4.1) in more detail. The search tree generation is controlled in the look-ahead and split phase.

Listing 4.3: Lookahead procedure.

```
1 maxactive = 8
2
3 lookahead(sp[ ]) {
4   sortlist(sp[ ]);
5   for (i = min(maxactive, length(sp)) − 1; i>=maxactive/2 && i>=0; i−−) {
6     sp[i].lkhd = boolector_lookahead(sp[i].F);
7   }
8 }
```

List. 4.3 shows the implementation of the look-ahead phase. Since the number of generated subproblems should be restricted to be not greater than maxactive (default value 8), a look-ahead is performed on maximum maxactive/2 subproblems. Doing a look-ahead on 4 instances results in 8 additional subproblems generated in the split phase. The question is, which subproblems in the list should be selected to split first. The chosen strategy is to select simple subproblems but not the simplest ones. It is assumed that the formula's hardness is proportional to its size (in reality this assumption is not always true but it will fit for our purposes). The Boolector API was extended with boolector_get_size() which returns the formula size (the number of unique nodes of the formula's DAG representation). In line 4 the subproblems are first sorted by increasing size. Simpler subproblems are at the beginning and harder subproblems at the end of the list. Next, simple formulas are selected to perform a look-ahead on. In the case that the number of current subproblems is ≥ 8, a look-ahead on sp[7] to sp[4] is performed. The idea behind this selection strategy is explained at the end of the current section.

Listing 4.4: Split procedure.

```
1  split(sp[ ]) {
2    for (i = 0; i<length(sp); i++) {
3      if (sp[i].lkhd != NULL) {
4        pred = defpredicate(sp[i].lkhd); //define a predicate
5        clone = boolector_clone(sp[i].F);
6        boolector_assert(sp[i].F, pred);
7        boolector_assert(clone, boolector_not(clone, pred));
8        addtolist(sp[ ], (clone,0,0))
9        added++;
10       sp[i].lkhd = 0 //reset lookahead node
11     }
12   }
13 }
```

Each formula with a valid look-ahead node is split (List. 4.4). The pseudocode uses C style pointers for look-ahead nodes. The lkhd field of structure sp contains a valid look-ahead node if the pointer is not NULL. First, a predicate has to be defined on the look-ahead node (line 4). In the case that sp[i].lkhd is a Boolean node the predicate is the node itself. In the case that sp[i].lkhd has bit-width > 1 a predicate is generated (e.g, $x = 0$). In line 5 the original formula is cloned. Then the predicate is asserted to be *true* in the original formula (subproblem 1) and to be *false* in its clone (subproblem 2). The clone is added to the list. The added counter is later used for the dynamic limit update (Sect. 4.2.3).

Listing 4.5: Search procedure.

```
1  search(sp[], lim) {
2    sortlist(sp[ ]);
3    for (i = 0; i < maxactive && i < length(sp); i++) {
4      sp[i].res = boolector_sat_limit(sp[i].F, lim);
5    }
6  }
```

In the search phase (List. 4.5) the subproblems are finally tried to be solved within the given limit lim. This is the most work intensive part. Usually 99% of the total runtime is spent in this phase. First, the list is again sorted by hardness (line 2). Next, the smallest maxactive subproblems are chosen and tried to be solved with boolector_limited_sat() (line 3-5). To recap, Boolector's SAT call internally first simplifies the formula on term-level and second solves the bit-blasted formula with the SAT solver.

Fig. 4.2 visualizes the abstract form of the search tree after some rounds. The tree is generated in an asymmetric way by selecting small but not the smallest subproblems for splitting (see List. 4.3). The small subproblems are split into smaller and smaller subproblems until they are solved. If they are solved, the next larger subproblems are chosen for splitting. Note that if look-ahead would

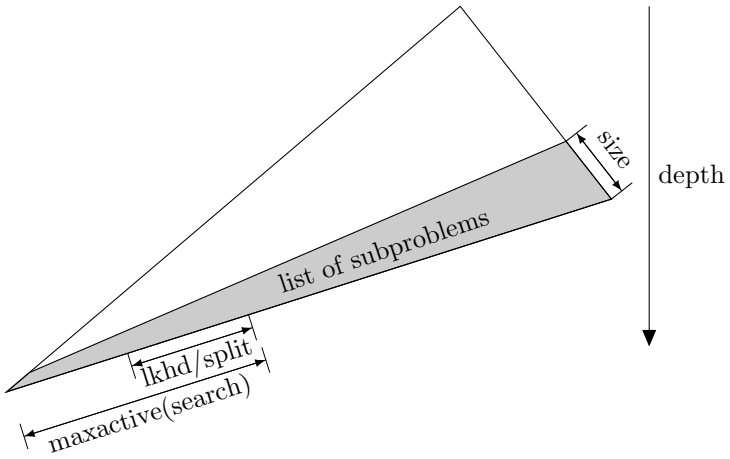


Figure 4.2: The abstract form of the search tree after some rounds. The filled area shows the current subproblems to be solved. The list is sorted by size (it is assumed that size is correlated to hardness). On the left side are small subproblems split several times (high depth in the search tree). On the right side are bigger subproblems which are split less (low depth in the search tree). The look-ahead phase selects the upper part of maxactive subproblems to split in the current round. The search phase tries to solve the maxactive smallest subproblems.

take the smallest subproblems (most left part of the list), the splitting would be too greedy, e.g., subproblems are produced which are too easy for search and the abstract sorted tree would look even more skewed. If the largest subproblems would be taken for splitting (most right part of the list) the tree would be generated in a balanced form. The algorithm would not be greedy enough to solve many subproblems as early as possible. As a consequence, we choose an intermediate, greedy but not too greedy, version. For *unsat* instances the whole tree has to be expanded anyhow, until all subnodes are closed. Hence, the selection strategy will rather influence *sat* instances, where the main loop can be immediately aborted by making the right look-ahead decisions.

4.2.3 Dynamic Limit

In the search phase of the main loop, subproblems are tried to be solved within some limit. If the limit exceeds, the subproblems are split until they are small enough to be solved within the limit. To abort the SAT call, the back end CDCL solver's conflict limit n is used. The SAT solver, and therefore also boolector_sat_limit(), returns with result *unknown* if n conflicts are exceeded without result. The abortion leaves the SAT solver in a consistent state. After incrementally adding some further constraints, the search can be continued

where it was aborted. Work already done does not need to be redone in the next round of the main loop.

In each round, upd(lim) updates the limit in a dynamic way. Changing the limit allows to control PBboolector's interplay between the split and the search phase. If the limit is low, less time is given to the SAT solver and the algorithm is forced to concentrate on the look-ahead and splitting. If the limit is high the algorithm is forced to concentrate on the search phase by giving more time to the SAT solver.

Listing 4.6: Update limit function.

```
1  closed, added; // number of closed and added nodes in current round
2  lim=5000; //intial limit
3  lmin=1000, lmax=50000; //min and max limit
4
5  upd(lim) {
6    if (!closed) lim *= 0.5;
7    else if (closed > added) lim *= 2;
8    else if (closed < added) lim *= 0.9:
9    if (lim < lmin) lim = lmin;
10   if (lim > lmax) lim = lmax;
11   added = closed = 0;
12   return lim;
13 }
```

List. 4.6 shows the update function. The variable added maintains the number of added subproblems in the current round. This value is updated by the split() procedure. The variable closed maintains closed subproblems in the current round updated by flush(). If no subproblem can be closed, it is assumed that the subproblems are too hard. Then the algorithm should concentrate on splitting. The limit is halved (line 6). If PBoolector is able to close more subproblems than added, subproblems may be split too much. It could be possible that the SAT solver would have been able to solve the problem more efficiently before splitting and with a higher limit. Therefore, more time is given to the search phase, by doubling the limit (line 7). If the algorithm is in an intermediate phase where more subproblems are added than closed the algorithm is forced to slightly concentrate more on splitting. The limit is reduced by 10% (line 8).

The look-ahead and split phases can be seen as the global search part of the algorithm. The search phase (bit-blasting and CDCL SAT solving) is the local search part. On hard to solve formulas we perform a global search and split into smaller subproblems. After enough reduction we switch to a local search. If the subproblems become hard again the algorithm is forced back to global search.

The limit is bound by lmin and lmax. A too small minimal limit can produce a lot of subproblems. A too high maximal limit can cause the SAT solver to

spend too much time for individual subproblems. The options "`-l <n>`" (initial limit), "`-lmin <n>`" (minimal limit) and "`-lmax <n>`" (maximal limit) can be set for PBoolector to configure the limits. The default values for the options are selected similar to the Treengeling default values.

4.2.4 Parallelization

Subproblem's are generated via the boolector_clone() interface, which makes a deep copy of a given solver instance. The resulting instance is an exact copy of the solver's current state. No memory is shared by the individual subproblems, which allows to run all operations in each phase in parallel. After finishing a phase the parallel operations are synchronized in the main loop and the operations of the next phase are started in parallel.

For parallelization the POSIX thread API *Pthreads* is used. The used API calls are briefly described in this section. We refer to the Linux man pages for a detailed API reference. Threads are created with pthread_create(). Create takes the function to be executed in the new thread and the corresponding arguments as parameters. For maintaining the thread, a thread instance of structure type pthread_t is returned. The pthread_join() method waits for a certain thread instance until termination. *Pthread* supports mutexes and conditional variables for synchronisation. Mutexes are of type pthread_mutex_t. To protect shared data. pthread_mutex_(un)lock() is used. Condition variables (type pthread_cond_t) allow to block the current thread until a certain condition is met. The pthread_cond_wait() method blocks the current thread. The waiting thread can be woken up by another thread with pthread_cond_signal() when the condition is fulfilled.

In List. 4.7 the parallelization of the most work intensive search procedure is shown. Parallelization of the other phases is done analogously by exchanging the Boolector operation (e.g., the look-ahead call is used instead of the limited SAT call). Note that the pseudocode only sketches the parallelization task and the *Pthread* API calls are shown in a simplified form.

The subproblems to solve are maintained in the initial empty list jobs[ ] (line 10). The job structure contains the thread, to which it is assigned, and the subproblem to solve. The number of maximal workers is by default the number of cores available on the machine (line 5). Each CPU core should try to solve 4 subproblems in each round (line 6). So on a 4 core machine in each round there are 16 active subproblems to be solved. First, maxactive jobs are scheduled to be run in parallel (line 13 -16). Second, the jobs are distributed and run as worker threads on the available cores (line 18-25). A created thread (line 24) executes the searchsp() method with the corresponding parameters (sp, lim), where the actual work (line 34) is performed. If all workers are busy (line 20) the thread creation is stopped by the conditional wait (line 21) and the main thread is blocked. If a thread finishes its work, the number of current workers is decre-

Listing 4.7: Parallelization of search phase.

```
1  struct job{
2    thread, sp
3  };
4  mutex, cond;
5  maxworkers = number of cores;
6  maxactive = maxworker * 4;
7  numworkers;
8
9  searchparallel(sp[ ], lim) {
10   jobs[ ]={}; numworkers=0;
11   sortlist(sp[ ]);
12   //schedule jobs;
13   for(i=(length(sp[ ])<maxactive)?length(sp[ ])-1:maxactive-1; i>=0; i--) {
14     job.sp = sp[i];
15     addtolist(jobs[ ], job);
16   }
17   //run jobs in worker threads
18   for (i=0; i<length(jobs[ ]); i++) {
19     pthread_lock(mutex);
20     while (numworkers>=maxworkers)
21       pthread_cond_wait(cond, mutex);
22     numworkers++;
23     pthread_unlock(mutex);
24     job.thread = pthread_thread(searchsp, jobs[i].sp, lim);
25   }
26   //join remaining workers
27   for(i=0; i<length(jobs); i++) {
28     pthread_join(jobs[i].thread);
29     deletefromlist(jobs, job[i]);
30   }
31 }
32 //limted search on subproblem sp
33 searchsp(sp, lim) {
34   boolector_limited_sat(sp.F, lim);
35   lock(mutex);
36   numworker--;
37   pthread_cond_signal(cond);
38   unlock(mutex);
39 }
```

mented (line 36) and the blocking main thread is woken up (line 37). In the main thread, the while condition (line 20) becomes false and the next job is distributed to a new worker thread on the free CPU slot. If all jobs are distributed the remaining running threads are synchronised (line 27-30). Shared access to the variable numworkers is avoided by locking. The method pthread_cond_wait() internally unlocks the mutex and then blocks the thread. Therefore, a finishing worker thread is able to enter the protected area (line 35-38) and wake up the conditional wait. Before pthread_cond_wait() returns, according to the *Pthread* API model, internally mutex is locked again and the main thread continues from line 21.

In short: The procedure searchparallel() schedules a certain number of jobs and executes them one after the other on the next free CPU slot. On each core at most one SAT call is done at the same time. This prevents thread switching. Modern architectures use hardware CPU virtualisation (hyper-threading). In the current implementation the default value of maxworkers is equal to the number of real cores plus number of virtual cores. The number of maximal workers can also be set with option "`-t <n>`" for PBoolector. Additionally it is possible to set the number of maximal active nodes with option "`-a <n>`". The number of maximal active nodes has to be always greater equal to the number of maximal workers.

By assigning more than one job to a worker (line 6), unbalanced workload can be compensated. In practice it is often impossible to split a formula into two parts with the same workload (even if the same limit is used for the SAT call). For instance assigning some variable $x = 0$, the formula may be solved much faster than by assigning $x \neq 0$ on the other branch. The predicate $x \neq 0$ is in general a very weak restriction and usually cannot speed up the SAT call. If there is unbalanced workload, searchparallel() does occupy one CPU core with the work intensive subproblem, whereas more and smaller subproblems are assigned to the other CPU slots. It is preferable to schedule the most work intensive jobs first, in order to be able to compensate them with less work intensive jobs. To do so, initially the subproblems are ordered by ascending hardness (line 11). Harder jobs are scheduled first by running the schedule loop from maxactive-1 to 0 (line 13).

4.3 Look-Ahead API Extension

Beside some utility functions the look-ahead method is the only implementation, which had to be integrated into the Boolector core implementation. Therefore, a new API extension for Boolector was implemented:

- boolector_lookahead(Btor*):BoolectorNode*
  Returns a node of the formula's DAG (BoolectorNode*) on which the formula can be split.

The look-ahead method decides on which node to split. The "quality" of that decision is crucial for the algorithm. This is demonstrated by the following example: Reconsider the search tree in Fig. 4.1. Look-ahead uses as first decision $x_1 = y_1 (x_1 \neq y_1)$. After splitting, look-ahead chooses for F_0 to split on $x_1 = 0$ ($x_1 \neq 0$) and for F_1 to split on $x_0 = y_0$ ($x_0 \neq y_0$). For this example "bad" decisions are made to demonstrate the expansion of the search tree. Anyhow, in the introductory example in Fig. 1.1 it was shown, that F can be solved faster by making "better" decisions. If $x_0 = y_0$ ($x_0 \neq y_0$) is selected first, the left branch can be closed immediately. The subformula gets trivially *unsat* by applying term rewriting rules. On the remaining subformula the next condition predicate $x_1 = y_1$ ($x_1 \neq y_1$) is selected. The two generated subproblems become trivially *unsat* too, which concludes the problem to be *unsat*. In the introductory example (Fig. 1.1) 4 subproblems are created, whereas in the execution example with the "bad" decisions (Fig. 4.1) 8 subproblems are generated for the same formula F.

The question is which node of the formula DAG should be selected by boolector_lookahead(). To give an idea which node would be the best, Sect. 4.3.2 shows an expensive implementation, the *Probing Method*. Note that the probing method uses the same concepts as methods which are usually associated in literature with the term **look-ahead** [40]. Look-ahead SAT solvers select a decision variable by probing the reduction. In this thesis, the term look-ahead is used in a slightly different way. If not explicitly stated otherwise, the term look-ahead is always associated with a faster and for PBoolector more appropriate implementation called the *Reduction Approximation Method*. This method does not measure the real reduction by costly simplification operations, but approximates reduction by means of cheaper heuristics. Implementation details for approximation are explained in Sect. 4.3.3.

The process of making a look-ahead decision is sketched in the following points, and explained in detail in the according sections:

(1) Find look-ahead candidates (Sect. 4.3.1).

(2) Approximate or measure reduction for each candidate (Sect. 4.3.2 and Sect. 4.3.3).

(3) Return the most appropriate node (Sect. 4.3.4).

4.3.1 Look-Ahead Candidates

The look-ahead method first searches for look-ahead candidates. A look-ahead candidate is a node on which the formula can be split. Candidates are all nodes which have an asserted (constrained) node as antecessor. If a node has no constrained parent it is no look-ahead candidate. Further, the constraint nodes are no candidates itself. If for example predicate p and formula G are constraint nodes ($F := G \wedge p$) the formula F can not be split on p: asserting p to be *true*

does not change F ($G \wedge p \wedge p \equiv G \wedge p$) and asserting the negation is trivially *unsat* ($G \wedge p \wedge \neg p \equiv \bot$). In the implementation the formula's DAG is traversed once by starting at the constraint root nodes and storing each child in a hash table lkhd_candidates.

4.3.2 Probing Method

The goal is to select a look-ahead candidate which maximizes the simplification reduction for both subproblems, which are generated in the main loop's split phase. The probing method implementation simply finds this candidate by trial (probing). For each candidate the real reduction is measured. To measure the positive reduction, the node (or a new defined predicate to split) is asserted to be *true* and the rewrite engine (simplification) is called. The positive reduction *pred* is the difference of the formula's size before and after the simplification. The negative reduction *nred* is measured by asserting the node or the predicate to be *false*. After measuring all nodes, the node with the highest score $pred \cdot nred$ is returned by the look-ahead method. The multiplication is used in the score to return nodes that have high reduction in positive **and** in negative phase. Splitting should produce two subproblems, which should be able to be simplified in the main loop. For example a node with $pred = 5$ and $nred = 5$ ($score = 25$) is preferable to a node which would produce unbalanced subproblems with $pred = 18$ and $nred = 1$ ($score = 18$).

The probing method is very cost intensive. Simplification is a time linear operation in number of nodes contained in the DAG. Calling the linear simplification function for each node in the DAG leads boolector_lookahead() to have a quadratic time complexity in number of nodes.

For testing and evaluation purposes the probing method was implemented in PBoolector. With option "`-pr`" probing is enabled. With the additional option "`-rp`" the look-ahead method returns the the node with highest score $pred \cdot nred$. If the simplification during probing directly leads to a result ($sat/unsat$) in positive and/or in negative phase the look-ahead is aborted and the node is directly returned for splitting. This is similar to find failed literals in look-ahead SAT solvers. In practise, if option "`-pr`" is enabled most of the runtime is spent in the look-ahead phase.

4.3.3 Reduction Approximation Method

Due to the overhead of the probing method, in PBoolector look-ahead was implemented without exactly measuring the reduction by using cheaper approximation heuristics. In the implementation the formula DAG is traversed ones and statistical features are calculated. The features are used by the heuristic to approximate the reduction.

It is assumed that the number of occurrences is proportional to the formula's real reduction. As the formula is internally represented as a DAG using structural hashing, nodes are referenced several times from their parents. The most basic statistical feature to approximate the reduction is to count the number of positive occurences (*pocc*) and negative occurrences (*nocc*) for each candidate. The node with the highest score $pocc \cdot nocc$ is returned. Equal to the probing method, this method returns the node with maximal (approximated) reduction in both subproblems.

In the implementation Boolector's unique_table is used to count the occurrences. The unique_table is a hash table containing each unique node of the DAG. For each unique node the children are traversed. If a child is referenced positive and it is contained in the lkhd_candidates table a corresponding positive reference counter is incremented. If the child is referenced negative analogously, its negative reference counter is incremented. Each node has to be visited only once. Therefore, this implementation is close to linear in the number of nodes. The term "close" is used because hashing with collision chains has only close to constant time access.

4.3.4 Look-Ahead Types and Heuristics

In the implementation several different look-ahead types are defined. For each look-ahead type a separate heuristic can be applied. The look-ahead method generates a table containing for each type the node with the maximal score. This section discusses in detail each look-ahead type, the corresponding heuristic and the expected reduction. All types are summarized in Tab. 4.1 at the end of this section. Sect. 4.3.5 shows the implementation of how the individual look-ahead types are combined. The desired look-ahead type is selected in PBoolector with option "`-lt <id>`".

Equality Nodes

First bit-vector equalities are examined. An equality is always a Boolean node on which the formula can be split by asserting it to be either *true* or *false*. The used heuristic is $pocc \cdot nocc$. Bit-vector equalities are separated into three look-ahead types:

BEQ1VN are bit-vector equalities containing at least one parameter which is a variable with bit-width > 1. By asserting an equality with a variable to be *true* ($x = term$) simplification can trigger basic Boolean rewriting and also variable elimination. By asserting $x = term$ every occurrence of x can be replaced with the node $term$, which results in a formula with x eliminated. On the second branch where the equality is asserted negatively ($x \neq term$) only Boolean rewriting rules can be triggered.

BEQ1V1 are equality nodes containing at least one variable with bit-width one. An inequality with Boolean parameters ($x \neq term$) is rewritten into an equality with one parameter negated ($x = \neg term$). Splitting leads on both

branches to Boolean rewriting and, different to BEQ1VN, also in both branches to variable elimination.

BEQ1OTHER are all remaining bit-vector equality nodes without variables. Here only Boolean rewriting rules can be triggered after splitting.

Inequality Nodes

Inequality (unsigned less than $<$) nodes have look-ahead type **ULT1N**. The disadvantage is that no powerful rewriting rules like variable elimination or conditional rewriting can be triggered by asserting the node. Asserting an inequality only Boolean rewriting rules can be applied to reduce the formula. The used heuristic is $pocc \cdot nocc$.

If-Then-Else Nodes

A good candidate for splitting are if-then-else nodes (look-ahead type **BCONDN**). If the conditional predicate is asserted to be *true* or *false* the conditional can always be rewritten by replacing it with the corresponding branch. Conditionals often occur uniquely in a formula but with high reduction potential. If for example a condition has only one positive reference $pocc = 1$ ($nocc = 0$). The score with heuristic $pocc \cdot nocc$ would be 0. Nodes with score 0 are not selected by the look-ahead procedure. In order to select also unique referenced conditionals the heuristic $pocc + nocc$ is used. In the evaluation chapter (Chap. 5) it is shown that some formulas containing hard operations (like multiplications) with conditional parameters can be solved with very good speedup by splitting on conditionals.

Other Boolean Nodes

Any other node with bit-width one can be chosen as look-ahead, which are defined as follows: Variables (**VAR1**), slice operations (**SLICE1**), and operations (**AND1**) and all remaining operations (except constants) with bit-with one (**OTHER1**). For AND1 and OTHER1 types the $pocc \cdot nocc$ heuristic is used. For VAR1 and SLICE1 the heuristic $pocc + nocc$ is used. Like for conditionals the "$\cdot$" heuristic would not select uniquely referenced variables or slices. The "+" heuristic is used, because in practice it was observed that fixing one-bit-slices or variables with only one reference, sometimes can speedup the solving procedure. An example was found in the SMT-LIB benchmarks, the *problem_19* from category *calypto* (see Sect. 5.4.3).

Defining New Predicates for Hard Operators

It is especially desired to be able to rewrite hard operators like multiplications or additions. Look-ahead type **MULNV0** selects multiplications that contain at least one variable ($x \cdot term$). The formula is split by defining a new predicate ($x = 0$) and asserting it to be *true* ($x = 0$) and *false* ($x \neq 0$), respectively. Setting $x = 0$ the multiplication can be reduced by variable elimination and

arithmetic rewriting to a zero constant ($x \cdot term \wedge x = 0 \equiv 0$). The problem is that on the second search branch with the constraint $x \neq 0$ nothing can be reduced or rewritten. The subproblem is mostly as hard as the original problem. The constraint $x = 0$ is very strong whereas $x \neq 0$ is a very weak constraint (especially if x has high bit-width). Therefore, to split on multiplications by defining new predicates on its variables can only speed up satisfiable instances. If a solution (*sat*) can be found in the easy partial search tree with reduced multiplications the algorithm immediately concludes with result *sat*. On the other hand, if there is no solution, a set of subformulas with similar hardness to the original formula is generated (all subformulas with constraints $x \neq 0, y \neq 0, \ldots$) and has to be solved.

Note that on all previously described look-ahead types the node on which is split becomes a constraint or is eliminated by rewriting. Thus, it is not possible that in a following round the look-ahead method returns the same node again, because a constraint node is not a look-ahead candidate. By asserting $x = 0$ the variable and the multiplication is eliminated too ($x \cdot term \equiv 0$). However, asserting $x \neq 0$ does not trigger any rewriting and can not elimate the multiplication ($x \cdot term$). In order to avoid returning the same node again in a subsequent round, it needs to be checked if variable x is not constrained with $x \neq 0$. If the constraint exists the node is not considered as look-ahead node.

The look-ahead type **MULNV1** is the same as MULNV0 except that its variable x is set to 1 to trigger the arithmetic rewriting rule $1 \cdot term \equiv term$. For additions the type **ADDNV0** is defined where the variable is set to 0 ($0 + term \equiv term$). Similar types can be defined for all other hard arithmetic operations (division, modulo, ...). It is left for future work to implement and evaluate the types for the remaining arithmetic operations.

Bit-Splitting on Hard Operations

Another way to speed up hard operations is to split on single bits. Bit-splitting is implemented for the look-ahead types **MULNBIT0** and **ADDNBIT0**. MULNBIT0 selects an arbitrary multiplication node in the DAG and splits on the LSB. ADDNBIT0 does the same for adders.

In the bit-blasted CNF encoding the term structure of the formula is lost. The SAT solver should examine in the DPLL search tree first the Boolean variable which corresponds to a multiplier's/adder's LSB. But without term structure knowledge it is not possible to select the right variable. In PBoolector the operator's LSB can be fixed already on term-level. Bit-splitting is different than other look-ahead types proposed in this section. For all other look-ahead types the aim was to facilitate formula reduction for the rewrite engine, prior to bit-blasting. However, the aim of bit-splitting is not a reduction of the formula but to guide the search of the SAT solver.

For bit-splitting a slice of the operation's LSB is made. The Boolean slice node is asserted one time in positive and one time in negative phase. Note that the assigned context bit is not only available in the CNF Encoding it is also available in the intermediate AIG layer, which gives future opportunities to use the information also for new AIG rewriting rules. Similar to MULNV* and ADDNV*, in the look-ahead method it needs to be checked if a operation's LSB is already asserted. If the bit is already asserted the respective node is not considered anymore as look-ahead.

4.3.5 Combination of Look-Ahead Types

PBoolector can be run for each look-ahead type independently. For example option "`-lt 4`" only considers nodes with look-ahead type BCONDN. This section shows how different look-ahead types are combined. After finding the node with maximal score for each look-ahead type one of the types has to be selected and returned by the look-ahead method.

Default Fixed Order Combination

The types are selected by a fixed order. Look-ahead types which are most appropriate for splitting are selected first. The defined order was a result of theoretical consideration and observed behaviour in the experiments. The look-ahead types are selected as follows:

(1) If a node of type BEQ1VN exists return node with maximal score.

(2) Else if a node of type BCONDN exists return node with maximal score.

(3) Else if a node of type SLICE1 exists return node with maximal score.

(4) Else if a node of type VAR1 exists return node with maximal score.

(5) Else if a node of type MULNBIT0 exists return node with maximal score.

(6) Else if a node of type BEQ1OTHER exists return node with maximal score.

(7) Else if a node of type AND1 exists return node with maximal score.

(8) Else if a node of type ULT1N exists return node with maximal score.

(9) Else if a node of type OTHER1 exists return node with maximal score.

(10) Else if a node of type BEQ1V1 exists return node with maximal score.

(11) Else return null (no look-ahead node found).

Id	Type	Description	Heuristic
0	BEQ1VN	Bit-vector equalities containing a variable with bit-width > 1.	$pocc \cdot nocc$
1	BEQ1V1	Bit-vector equalities containing a variable with bit-width $= 1$.	$pocc \cdot nocc$
2	BEQ1OTHER	Every bit-vector equality which is not in look-ahead type 0 or 1.	$pocc \cdot nocc$
3	ULT1N	Unsigned less than bit-vectors.	$pocc \cdot nocc$
4	BCONDN	Bit-vector conditions with bit-width > 1.	$pocc + nocc$
5	VAR1	Bit-vector variables bit-width $= 1$.	$pocc + nocc$
6	SLICE1	Bit-vector slices with bit-width $= 1$.	$pocc + nocc$
7	AND1	And bit-vector operations with bit-width $= 1$.	$pocc \cdot nocc$
8	OTHER1	All other nodes with bit-width $= 1$ (except constants) which are not in look-ahead type 0 - 7.	$pocc \cdot nocc$
9	MULNV0	Multiplication containing a variable x with bit-width > 1. Split by setting $x = 0$ and $x \neq 0$.	$pocc + nocc$
10	MULNV1	Multiplication containing a variable x with bit-width > 1. Split by setting $x = 1$ and $x \neq 1$.	$pocc + nocc$
11	MULNBIT0	Multiplication m with bit-width > 1. Split by setting $m[0] = true$ and $m[0] = false$.	$pocc + nocc$
12	ADDNV0	Addition containing a variable x with bit-width > 1. Split by setting $x = 0$ and $x \neq 0$.	$pocc + nocc$
11	ADDNBIT0	Addition a with bit-width > 1. Split by setting $a[0] = true$ and $a[0] = false$.	$pocc + nocc$

Table 4.1: Summary of defined look-ahead types.

First, nodes of type BEQ1VN and BCONDN are selected. They have high rewriting potential (variable elimination and conditional reduction) and have shown some good results in the experiments. Next, SLICE1 and VAR1 are selected. These 2 types are selected with some higher priority because on some input formulas very good speedup can be achieved. Next, MULNBIT0 is selected. Here again splitting on MULNBIT0 nodes achieved good performance on some experiments. Type ADDNBIT0 is not added, as no improvements could be achieved for this type. Also the remaining types could not achieve performance gain on any benchmark. Somehow interesting is the type BEQ1V1. For this type good reduction by variable elimination was expected. But experimental results have shown that splitting on this type the algorithm gets very slow on several benchmarks. Therefore, this type is ranked at the end of the selection.

In general it is hard to define a fixed order. It seems that the best order strategy depends on the input formula's structure. With the implemented solution it was tried to find a good general strategy. The fixed order combination is done by running PBoolector with look-ahead type option "`-lt 14`". Type 14 is PBoolector's default option.

Fixed Order Combination for Satisfiable Instances

Another combination for selecting look-ahead types is defined by selecting those look-ahead types first that should be able to improve satisfiable instances (MULNV0, MULNV1, ADDNV0). Again, a fixed order selection is done by first trying to reduce the hard operators. The strategy is as follows:

① If a node of type MULNV0 exists return node with maximal score.

② Else if a node of type MULNV1 exists return node with maximal score.

③ Else if a node of type ADDNV0 exists return node with maximal score.

④ Else if a node of type MULNBIT0 exists return node with maximal score.

⑤ Else select by the default fixed order combination.

This strategy is used by running PBoolector with look-ahead type option "`-lt 15`".

Other Combination Methods

Two other methods to combine look-ahead types are implemented: Option "`-lt 16`" selects the look-ahead type randomly. Option "`-lt 17`" uses probing for selecting the best look-ahead type. For these look-ahead types the reduction of the node with maximal (heuristic) score is measured. The look-ahead type with maximal reduction $pred \cdot nred$ is returned.

4.4 Implementation Extensions

The evaluation in Chap. 5 shows that on several benchmarks PBoolector performs slower than the sequential reference implementation. This section will present some extensions which aim to improve the results. Sect. 4.4.1 shows a method similar to portfolio approaches to make PBoolector competitive to the sequential solver. The optional features proposed in Sect. 4.4.2 and Sect. 4.4.3 are expected to have great impact on the results because similar (but more advanced) approaches improved the performance of the Treengeling solver. Sect. 4.4.4 shows some smaller implementation extensions to improve PBoolector.

4.4.1 Parallel Boolector Instance

To make PBoolector competitive to other solvers, the implementation was extended to run a parallel sequential Boolector instance. The parallel solver instance tries to solve the formula without a limit in a separate thread. If a solution is found by the parallel instance, the interleaving split and search in the main loop is aborted. However, if a solution is found by splitting and searching the parallel instance is aborted. Starting PBoolector with "`-sp`" (start parallel instance) option will always have faster or at least similar runtimes as plain Boolector.

The drawback with option "`-sp`" is that the parallel instance occupies one core and the main loop can utilise maximal *number of cores* $-$ 1 cores. The parallel instance has just similar runtime to Boolector if it is run on a separate physical core respectively. If the instance runs on a virtual core shared with other computations the parallel instance always has slower runtimes than Boolector. In practice, it turns out that the parallel instance is slightly slower than sequential Boolector, even if only physical cores are used. A similar problem arises by running two sequential Boolector processes on the same input formula on two cores in parallel. The two processes will need more wall clock time than just starting one process. For instance starting Boolector on the selected QF_BV benchmark *8moves_mti_8-Atd_00004_bmc.smt2*, it is solved in 80.6s. Running two identical processes on two cores with same input, the solving time increases for each process to $\sim 98s$. This effect seems similar to the challenges observed for the portfolio approach of Z3 [59]. Considerable overhead was observed running multiple identical solver instances in parallel. Cache and memory bus congestion have negative impact on the solver's performance. A closer investigation on this phenomena was done for SAT solvers by Aigner et al. [1]. Here it was shown that Lingeling (the default background solver of Boolector) has lowest slowdown compared to other SAT solvers, by running multiple instances in parallel.

To get a clean termination and stop a parallel running boolector_(limited)_sat() call the SAT solver's termination callback function is used. The callback can be set with the new API extension boolector_set_term_sat(). Termination is cur-

rently only supported if Lingeling is used as back end SAT solver. The boolector_(limited)_sat() method internally solves the formula after bit-blasting with Lingeling's lglsat() call. The method lglsat() continuously checks the termination function and aborts with result *unknown* if the termination function returns true. The termination function is implemented to return true if a solution is found either by the parallel instance or in the main loop. Since for bit-vector formulas boolector_(limited)_sat() spends most time in the lglsat() call, it is sufficient to abort the search by aborting the SAT solver.

4.4.2 Using Boolector in Non-Incremental Mode

Boolector's incremental mode allows to call boolector(_limited)_sat() (in the following abbreviated with bsat()) on the same formula instance several times. If Boolector is used in non-incremental mode bsat() can only be called once on an instance. Running Boolector (and therefore the back end SAT solver) in non-incremental mode allows to do several optimisations which speed up the solving process. However, PBoolector is a natural application for incremental solving. In the main loop after doing a limited search, constraints are added and in a subsequent round bsat() is called again. This section proposes an optional implemented feature, which allows to use Boolector in non-incremental mode. In the evaluation (Chap. 5) it is shown that using non-incremental mode on some instances perform better. On the other hand there are several instances, which perform better using the default incremental mode.

To be able to maintain non-incremental subproblems, in the search phase before calling bsat(), the instances are cloned. Next, on the cloned instances bsat() is called exactly one time. The result is stored and the cloned instances are deleted. For the original instances, which are split by adding constraints, bsat() is never called. There are two drawbacks using non-incremental mode. First, there is additional overhead by generating a clone before each bsat() call. The second drawback, which is even worse, is that already done work in the SAT solver needs to be redone in subsequent rounds. Running the SAT solver for a while, parts of the search tree are examined, e.g., some Boolean variables can be assigned to fixed values and conflict clauses are learned. If the SAT solver is in incremental mode in the next search call, the work can be continued. If the SAT solver is in non-incremental mode the intermediate learned results are lost and all previous work needs to be redone. However, in order to not loose the work of the rewriting engine a boolector_simplify() call is done on the original instance before cloning. Option "`-ni`" enables the non-incremental mode in PBoolector.

4.4.3 Failed Literal Reduction

One method to improve the performance of PBoolector is called Failed Literal Reduction (FLR). FLR is performed before the Cube and Conquer algorithm is started, and tries to initially reduce the formula by applying failed literals.

Failed literals are Boolean nodes which lead to a result in the simplification if they are assumed to be *true* or *false*. FLR is similar to the probing method introduced in Sect. 4.3.2. The input formula DAG is traversed from top to bottom with breadth first search. If a Boolean node (literal) is found the problem is cloned 2 times. Next, the node is asserted to be *true* on the first clone and to be *false* on the second clone. Each clone is tried to be reduced by calling boolector_simplify. If the simplification of one clone leads to the result UNSAT the Boolean node is a failed literal and it can be asserted in the opposite phase. After finding and asserting a failed literal the process is restarted again to find the next failed literal. The process is continued until no failed literal can be found anymore (until fixpoint). If the simplification of one clone leads to SAT a solution is found and FLR is aborted with result SAT. If the simplification of both clones leads to result UNSAT, FLR is aborted with the result UNSAT. For most benchmarks FLR can only reduce the formula but does not find a result.

FLR is very cost intensive. The probing method in Sect. 4.3.2 was stated to have quadratic time complexity in terms of the number of nodes n in the DAG. In the FLR the probing process is restarted n times (worst case) and has cubic time complexity. Therefore FLR, enabled with option "`-flr`", is only called once before the Cube and Conquer algorithm is started. For SAT solvers an advanced method called Tree Based Look-Ahead was proposed by Heule et.al. [38], where FLR was improved by using implication trees. In Treengeling the solver performance could be improved by doing an initial Tree Based Look-Ahead before Cube and Conquer is started.

Boolector has a feature to solve a problem under assumption. The API call boolector_assume() adds a constraint which is removed again after the problem is solved with the assumption (boolector_sat()). For FLR and also for the probing method a similar feature "simplification under assumption" would be desirable. Simplification under assumption would avoid the overhead of cloning the instance before probing is done. Unfortunately such a feature is not available in current SMT solvers nor SAT solvers actually.

4.4.4 Further Implementation Extensions

This section proposes some further small implementation extension to improve the algorithm. Similar features where implemented in the parallel SAT solver Treengeling.

Memory Limitation

To avoid that PBoolector runs out of memory a memory limit was implemented. If the memory limit is exceeded the look-ahead phase is aborted to not produce new subproblems. The implemented API extension

- boolector_bytes(Btor*):size_t
  returns the number allocated bytes of a Boolector instance.

The number of bytes for all subproblems are accumulated. If current memory consumption, plus the expected memory consumption which will be produced by splitting exceeds the memory limit, the look-ahead phase is aborted. By default the memory limit is set to *maximal available memory on the machine*/3. The option "`-m <limit>`" allows to set the memory limit in MB. Option "`-m -1`" runs PBoolector without memory limitations.

Dynamic Conflict Limit Extension

One extension was made to the upd(lim) function. There are two cases that zero subproblems are added in a round. The first case is if for all subproblems no look-ahead can be found. The second case is that the memory limit is reached and no new subproblems are produced. In both cases, the subproblems need to be solved in the search phase. This is done by adding a new update limit rule: If no node is added within a round force the algorithm to spend more time in the SAT solver by increasing the conflict limit (if (!added) lim = lim * 2;)

Finishing Subproblems

If no look-ahead node can be found no further splitting is possible. By default, then subproblems are solved without a limit (subproblems are finished). Finishing the subproblems sometimes leads to very high individual search times. To avoid finishing subproblems an optional flag no finish ("`-nf`") can be set for PBoolector. Using the option, subproblems where no look-ahead node can be found are not directly solved. They are tried to be solved within the current limit again. They are kept in the list of subproblems until they can be solved by continuously calling the limited version of the Boolector SAT call. The option is only allowed if PBoolector is used in incremental mode. If subproblems are not finished in the non-incremental mode, the main loop possibly does not terminate. This is the case if a subproblem, on which no further look-ahead can be found, is not solved within the maximal limit.

Full Search

Another extension made is to enable a full search each n'th round. Full search means that the limited search call is not only done on the maxactive smallest subproblems, but it is called on all open subproblems in the list. The assumption that a subproblem's size is proportional to its hardness does not always hold in practice. The occasionally started full search gives such "large" and "easy" subproblems, where the assumption does not hold, the chance that they can be solved and deleted from the list. The parameter n is set to be 10. Every 10th round all open subproblems are tried to be solved within the current limit.

Chapter 5

Evaluation

This chapter provides a detailed evaluation on different sets of benchmarks. The evaluation is separated into 3 parts. After the introductory section (Sect. 5.1) first a general evaluation is done on a big set of input formulas (Sect. 5.2). Second, the evaluation is done on a set of crafted benchmarks on which the algorithm is expected to work well (Sect. 5.3). The third part will take a closer look at the structure of successful SMT-LIB benchmarks to identify patterns which are appropriate for PBoolector (Sect. 5.4). Along the evaluation some few further optimisations are proposed to improve the performance.

5.1 Experimental Setup

All experiments are done on 2.83 GHz Intel Core 2 Quad machines with 8 GB of memory running Ubuntu 14.04. If not explicitly stated otherwise the default parameters of the PBoolector are chosen. The default parameter are chosen on one hand in an intuitive way (also motivated by default parameters used in Treengeling) and on the other hand by informal experiments on several benchmarks. The parameters which seemed to be best are chosen as default:

- Initial limit (`-l`): 5000
- Minimal limit (`-lmin`): 1000 - Using a value smaller 1000, experiments showed to produce too many subproblems at the intitial phase of the algorithm.
- Maximal limit (`-lmax`): 50000 - It was observed that if the limit is higher than 50000, on several benchmarks solving the individual subproblems get very slow.
- Number of workers (`-t`): Number of processor cores - For the 4 core machine used during evaluation (without hyper-threading support) the number of workers is 4.

- Maximal active subproblems (`-a`): 4 · number of workers - In Treengeling the constant value 8 was taken. Experiments showed that for PBoolector the constant value 4 achieves better performance. For the evaluation machine the number of active nodes is thus $4 \cdot 4 = 16$.
- Memory limit (`-m`): Maximal memory / 3 - For the machine used in the evaluation splitting is stopped if used memory exceeds the limit 8GB / 3 = 2.67GB.
- Look-ahead type (`-lt`): 14 - The default look-ahead is the combined order look-ahead type.

For the implementation Boolector version 2.0 with Lingeling version ayv as back end SAT solver is used. For all time measurements the wall-clock time is used. All experiments where run with a wall-clock time limit of 1800 seconds and a memory limit of 7GB. If a limit is reached the runtime status can be either out of time (oot) or out of memory (oom). All used benchmarks except the crafted ones are from the SMT benchmark LIB for QF_BV which contains approximately 33000 files.

5.1.1 Notations

To fully identify the denoted benchmarks they are notated by its name with the preceding category separated with an underscore. For example benchmark *brummayerbiere2_smulov1bw32* denotes benchmark *smulov1bw32* from category *brummayerbiere2*. To denote a set of benchmarks the "*" wildcard is used. The notation *smulov1bw** denotes for example the benchmarks *smulov1bw24*, *smulov1bw32* and so forth. Detailed information of the used table notation is found in App. A.

5.1.2 Implementation Issues

There are two issues left in the implementation which will appear in the results.

The first issue is related to the Runlim[1] tool which is used to control the execution of the experiments. Runlim stops the execution after the specified timeout is reached. The execution is aborted with the interrupt signal SIGINT. PBoolector have an installed signal handler for SIGINT where the statistical values are printed before termination. For some benchmarks the signal handler of PBoolector is not called when Runlim tries to abort the execution with SIGINT. The reason for this problem is unclear. If the execution does not stop after some seconds Runlim sends the kill signal. SIGKILL can not be handled by a signal handler. On benchmarks where this issue occurs no statistical values are printed and the according columns are empty in the detailed tables.

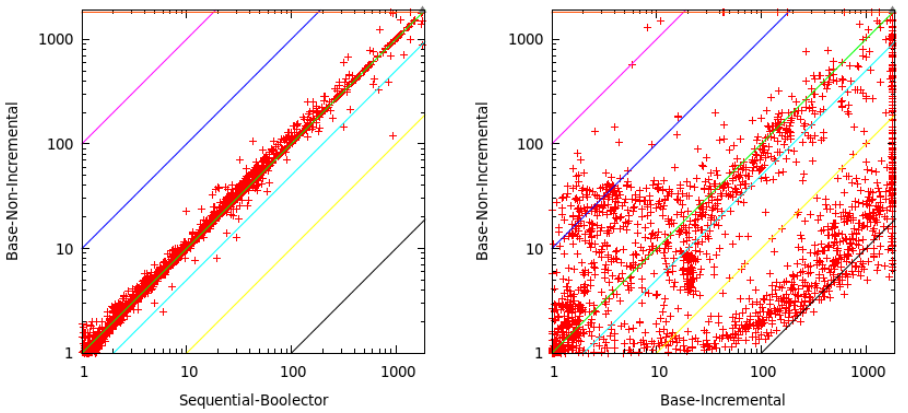
[1] http://fmv.jku.at/runlim/

Further there is still a non-deterministic bug in the implementation which could not be found. Because of the non-deterministic behaviour it is assumed that the bug is caused by some race condition. It was observed that the more threads are used the more frequent the bug is triggered. The Valgrind memory error detector does not find any issue in the implementation. It was tried to identify the problem with the Valgrind thread error detector tools Helgrind and DRD. Helgrind does not detect any error. DRD detects some errors but the issue could not be identified with the help of the error messages. It was also tried to use the Clang compiler which includes the optional data race detector ThreadSanitizer. No errors where detected in the implementation with the ThreadSanitizer either. PBoolector was adapted to use different parallelization APIs (OpenMP, cilk) where the bug also occurred. So it could be excluded that this issue lies somewhere in the locking mechanism implemented with the *Pthread* API.

A minimal version of PBoolector was implemented which uses a very simple look-ahead implementation and produces just 2 subproblems each round which are tried to be solved in parallel. For the parallelization only 2 threads are created which are joined after execution without any conditional wait or locking mechanism. The minimal implementation was made to limit the cause of the error. The problem was also reproducible in the simplified implementation. In the simplified implementation it "almost" can be excluded that it contains a race error, therefore it is assumed that the race error might be somewhere in the Boolector implementation. The issue occurs always during the boolector_clone() call where an assertion is not met. To debug the error, a set of files was selected on which the error frequently occurred. The implementation was adapted to add a sleep call with 50000 seconds if the assertion is not met. Then the selected benchmark files where executed continuously until the assertion was triggered and the execution went to sleep. With the GNU debugger it is possible to attach to the sleeping process and examine what is going wrong. However, the cause of the issue could not be found in the debugger either. The problem occurred only on very huge benchmarks on which debugging was nearly impossible. In the result tables there are some few benchmarks with status sf (segmentation fault) where this issue occurred.

5.2 SMT-LIB Competition Benchmarks 2014

First, a general evaluation is done on the SMT competition benchmarks 2014 (*sc14*) for QF_BV. This benchmark set contains 2488 files, which is a selection of the SMT benchmark LIB. The most promising PBoolector extensions (non-incremental mode, failed literal reduction and the combined look-ahead for satisfiable instances) are compared to the default configuration.



(a) Sequential-Boolector vs. Base-Non-Incremental reference times.

(b) Base-Incremental vs. Base-Non-Incremental reference times.

Figure 5.1: Runtime comparison of reference times.

5.2.1 Reference Runtimes

As reference times the solving times from PBoolector runs with limit -1 are taken. If the limit option is set to "`-l -1`" the problem is solved immediately in the first initial search call without any limit. For the evaluation of *sc14* benchmarks 2 different reference times are used. If parallel PBoolector is run in incremental mode the runtimes are compared to incremental reference times **Base-Incremental** (**base-inc**) with option "`-l -1`". If parallel PBoolector is run in non-incremental mode the runtimes are compared to non-incremental reference runtimes **Base-Non-Incremental** (**base-ninc**) with option "`-l -1 -ni`". The abbreviations in the parenthesis are used in the detailed tables where the space is sparse. The Reference-Non-Incremental runtimes should have the same runtime behaviour to the **Sequential-Boolector** (**sequ-btor**) tool, because Boolector solves QF_BV formulas in non-incremental mode.

To verify that Sequential-Boolector has equal runtimes to the Base-Non-Incremental runtimes, Fig. 5.1a shows a runtime comparison. The scatter-plot shows the runtimes of both implementations for each benchmark. Below the diagonal benchmarks are listed where Base-Non-Incremental is faster and above the diagonal Sequential-Boolector is faster. Most of the benchmarks exhibit similar runtime behaviour for both tools as expected. There are some outliers where Base-Non-Incremental is faster than Sequential-Boolector and vice versa. One remarkable outlier is the benchmark file *float_test_v5_r15_vr1_c1_s8236*. Base-Non-Incremental solves it in 120.26s whereas Sequential-Boolector takes 927.66s. Actually there is a slight difference in the implementation of Base-Non-Incremental compared to Sequential-Boolector. In the search phase of the non-incremental BPoolector first an explicit simplification is done, next the in-

stance is cloned and finally boolector_sat() is called on the clone. In the clone implicitly a second simplification is done during boolector_sat() (see Sect. 4.4.2). Running the simplification two times leads to a huge time difference of more than $800s$ compared to Boolector (where simplification is done once in the sat call) for this benchmark. Simplification is not done until fix-point in Boolector rather it is limited by some value. If the fix-point is not reached two simplification calls can lead to better reduction. Another reason for the more or less small jitter is that the IO operations (included in the measured wall-clock time) are not constant.

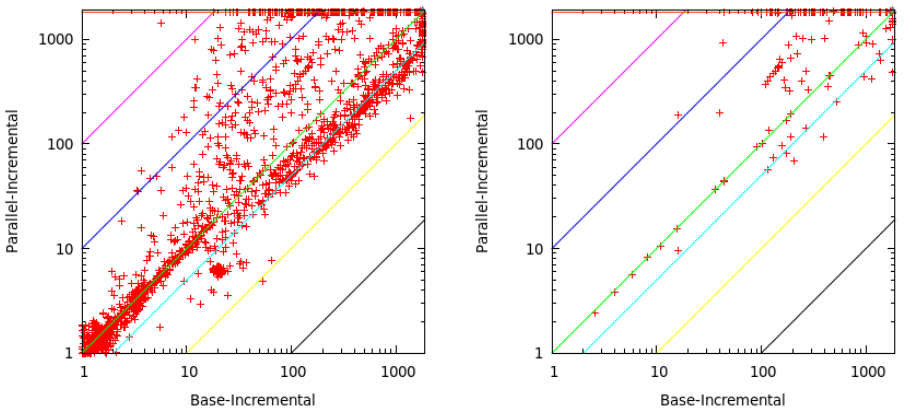
The second comparison shown in Fig. 5.1b should visualize why it is important to compare incremental PBoolector with the incremental reference times and non-incremental PBoolector with non-incremental reference times. There is a clear runtime difference for a lot of benchmarks turning incremental mode on or off. The back end SAT solver is differently configured depending on whether it is run in incremental mode or not. In the following experiments it should be compared if splitting can outperform an immediately solved instance. If the reference times would not be distinguished it would be not clear whether the speedup comes from the incremental mode setting or from splitting. The scatter plot shows that there is a cluster of a lot of "hard" benchmarks where the non-incremental mode is much faster (below the diagonal). On the other hand there is a cluster consisting of several "easy" benchmarks with contrary behaviour, where the incremental mode is faster (above the diagonal). Obviously sequential Boolector (which uses non-incremental mode for QF_BV formulas) and the SAT solver are configured to work best for hard benchmarks.

5.2.2 Results for PBoolector in Incremental Mode

For the incremental mode two different runs with different configurations are done. In the first run PBoolector is evaluated with the default configuration and in the second run PBoolector is evaluated with failed literal reduction (FLR) enabled.

Parallel PBoolector

PBoolector with the default options (**Parallel-Incremental/par-inc**) is compared to the reference implementation Base-Incremental. Fig. 5.2a shows the results for all *sc14* benchmarks. Below the diagonal the parallel implementation is faster, and above the diagonal the reference implementation is faster. There is a big set of benchmarks (especially hard ones) where the parallel implementation can not outperform the reference times. But there are several benchmarks which show an improvement. Most of the benchmarks with improvement are located near the diagonal where both implementations have similar runtimes. Tab. 5.1 row 1 shows this results in concrete numbers. From all 2488 benchmarks PBoolector is able to solve 47 files which are not solved by the reference implementation (++). 1395 benchmarks are solved faster (+), 610 are solved



(a) All benchmarks (2488).

(b) Hard Benchmarks (371), where Sequential-Boolector time is greater 100 sec.

Figure 5.2: Runtime comparison of PBoolector in incremental mode (Parallel-Incremental) vs. the reference implementation (Base-Incremental).

slower ($-$) and 196 benchmarks are not solved by PBoolector although they are solved by the reference implementation ($--$). 240 benchmarks are not solved by both implementations (o). To be able to give more detailed results the number of benchmarks is reduced. Only interesting input formulas, which are also hard for the sequential Boolector implementation are selected (runtime of Sequential-Boolector $>$ 100.0 sec). Using this filter 371 files are left. The results for the filtered benchmark set is shown in Fig. 5.2b respectively in Tab. 5.1 row 2. There are 9 benchmarks which achieve reasonable speedup whereas 93 benchmarks show a clear slowdown in the parallel implementation. Most of the benchmarks (201) are not solved in both implementations. The detailed results for the selected 371 benchmarks are shown in the appendix in Tab. C.1.

Tab. C.1 is sorted primary by its success state in ascending order and secondary by the time difference (to the reference implementation) in descending order. The ranking number consists of the success state ($++ = 1$, $+ = 2$, $- = 3$, $-- = 4$ and o $= 5$) and an increasing ranking number within each success state. The success states are separated by an empty line.

Before discussing the results, first some general notes are made about PBoolector's execution behaviour. There exist several benchmarks which are solved in the initial search call within the initial limit of 5000. These are all results where the number of rounds (*rnds*) is equal to zero. Another set of benchmarks is not appropriate for splitting. In all benchmarks where $rnds = 1$ and the number of total subproblems is equal to one, no look-ahead node was found for splitting. The work intensive part of the PBoolector is clearly the search phase. The look-

compare		filter	number of benchmarks					
ref	to		total	++	+	−	−−	o
base-inc	par-inc	-	2488	47	1395	610	196	240
base-inc	par-inc	sequ-btor> 100.0	371	9	30	38	93	201
base-inc	par-inc-flr	sequ-btor> 100.0	371	5	13	42	106	205

Table 5.1: Runtime comparison for parallel PBoolector runs in incremental mode (Detailed table notation in Sect. A.1 in the appendix).

rank	benchmark	sequ-btor[s]	base-inc[s]	par-inc[s]
1.1	brummayerbiere2_smulov1bw32	1056.25	oot	480.29
1.2	log-slicing_bvurem_16	oot	oot	1002.92
1.3	...iffyInterleavedModMult-8	oot	oot	1002.92
1.4	core_ext_con_056_002_1024	279.98	oot	1464.58
1.5	core_ext_con_060_002_1024	220.71	oot	1470.71
1.6	core_ext_con_052_002_1024	194.60	oot	1495.92
1.7	log-slicing_bvudiv_18	oot	oot	1554.09
1.8	core_ext_con_064_002_1024	344.29	oot	1769.72
1.9	log-slicing_bvsmod_15	oot	oot	1779.82

Table 5.2: The 9 benchmarks with success state ++ for the PBoolector (Parallel-Incremental) run.

ahead phase on almost all files have runtimes close to zero. The splitting phase has slight higher runtimes. For almost all files more than 99% of the time is spent in the search phase. In the search phase, almost for all benchmarks 99% of the CPU time is spent in the SAT solver (tot(cpu)∼sat(cpu)). The maximal and the mean search times are indicators for parallelization capability. The higher the maximal search time, the higher the parallel critical path and the less speedup can be obtained by increasing the number of processor cores.

There are 9 benchmarks with success state ++. An excerpt of the detailed result table is shown in Tab. 5.2. On 4 of these benchmarks (*core_ext_con**) even no look-ahead node is found. PBoolector first makes a limited initial search and solves the problem without any limit if no look-ahead is found in a second search call. Calling boolector_sat() twice solves some problems much faster than only calling it ones. This is probably the same behaviour as it was observed in Sect. 5.2.1. It was observed that solving times can be increased by running the simplification 2 times. Benchmarks which show a clear improvement (++,+) also compared to Boolector are from the benchmark sets *brummayerbiere2_smulov1bw**, *log_slicing_bv**, *... iffyInterleavedModMult-8*, and *multiplyOverflow*. Benchmark 1.1 and 1.3 have a relative small maximal search time. For these benchmarks the performance might can be increased using more processor cores. The rest of the benchmarks with success state ++ have a high maximal search time. For these formulas no great performance gain is expected using more than 4 processor cores. For the successful *bv** benchmarks from

par-inc \ par-inc-flr	++	+	−	−−	o
++	5	0	0	0	4
+	0	13	14	3	0
−	0	0	25	13	0
−−	0	0	**3**	90	0
o	0	0	0	0	201

Table 5.3: Success state change between Parallel-Incremental and Parallel-Incremental-FLR.

category *log_slicing* the number of produced subproblems is rather low and the number of finished subproblems is relatively high. There is only one satisfiable instance (2.7) where performance is clearly improved through splitting.

On the other hand most of the benchmarks exhibit a bad performance compared to the reference times. It seems that for most benchmarks the splitting process is not able to produce simpler subproblems. There are even benchmarks where the splitting process seems to produce harder subproblems than the original problem. An example is benchmark with rank 4.55. It has a maximal search time of $833.28s$ for one SAT call but the reference implementation solves the formulas within $328.30s$. On some benchmarks the produced subproblems can not be closed. For instance on benchmarks 4.51, the number of total subproblems (184) is equal to the number of maximal subproblems (184). On most of the bad performing benchmarks the memory limit is reached (mem $>$ 2.6GB) such that splitting has to be stopped. This hints towards that the high memory consumption is a clear bottleneck in PBoolector.

Parallel PBoolector with FLR

In the next experiment the effect of FLR is evaluated. The results of PBoolector with option "`-flr`" enabled (**Parallel-Incremental-FLR**/**par-inc-flr**) are summarized in Tab. 5.1 row 3. The results with FLR enabled, get slightly worse to the previous experiment. From the the 371 hard benchmarks now only 5 have success state ++ and only 13 have success state +. The number of files with negative success state have been increased. Tab. 5.3 shows the success state change between the Parallel-Incremental run and the Parallel-Incremental-FLR run. The table is read as follows: 5 files with state ++ in par-inc have state ++ in par-inc-flr, 4 files with state ++ in par-inc have state o in par-inc-flr and so forth. Almost all benchmarks have changed to an equal or a worse success state. The only improvement was achieved on 3 files which have moved from −− to −. The detailed results are shown in Tab. C.2 in the appendix. The column cmp contains a cross reference to the Parallel-Incremental run. All remaining successful benchmarks contain 0 failed literals. Benchmakrs which contain failed literals, the results where similar or worse. The only clear success

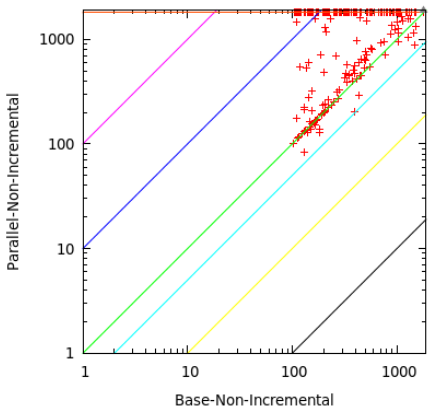


Figure 5.3: Runtime comparison of PBoolector in non-incremental mode (Parallel-Non-Incremental) vs. the reference implementation (Base-Non-Incremental).

compare		filter	benchmarks					
ref	to		total	++	+	−	−−	o
base-ninc	par-ninc	sequ-btor> 100.0	371	0	36	63	122	150
base-ninc	par-ninc-flr	sequ-btor> 100.0	371	0	42	53	126	150

Table 5.4: Runtime comparison for parallel PBoolector runs in non-incremental mode.

state improvement (from −− to −) where on benchmarks 3.38, 3.39 and 3.41, which only contain one failed literal. There is no benchmark where a result through FLR can be found. FLR is not able to improve the results without FLR (Parallel-Incremental).

5.2.3 Results for PBoolector in Non-Incremental Mode

In this section an evaluation is done for the parallel PBoolector implementation in non-incremental mode. Here again 2 different runs are done. In the first run FLR is disabled and in the second run FLR is enabled. The evaluation is done on the 371 hard benchmarks selected in the previous section.

Parallel PBoolector

In the first run the PBoolector (**Parallel-Non-Incremental/par-ninc**) is compared to the reference implementation Base-Non-Incremental. The results, shown in the scatter-plot in Fig. 5.3 respectively in Tab. 5.4 row 1, look very similar to the incremental results. Few of the benchmarks (36) achieve a performance gain. Tab. C.3 in the appendix shows the detailed results. Actually

par-ninc \ par-ninc-flr	++	+	−	−−	o
++	0	0	0	0	0
+	0	33	3	0	0
−	0	**9**	50	4	0
−−	0	0	0	122	0
o	0	0	0	0	150

Table 5.5: Success state change between Parallel-Non-Incremental and Parallel-Non-Incremental-FLR.

the benchmarks from rank 2.1 to 2.6 are interesting because they have a time difference > 100 seconds. This 6 files are mostly from the same benchmark family (category *log-slicing*) as the successful benchmarks in the incremental mode. For most of the benchmarks the parallel implementation is slower. Here it is interesting that all benchmarks which where solved by the incremental PBoolector with success state ++ can not be solved by the non-incremental PBoolector (4.18 *smulov1bw32*, 5.86 *bvurem_16*, 5.25 *... iffyInterleavedModMult-8*, 5.67 *bvudiv_18* and 5.44 *bvsmod_15*). For these files it seems to be crucial that already done work in the SAT solver is not lost (only in incremental mode).

Parallel PBoolector with FLR

In the next experiment the FLR is turned on (**Parallel-Non-Incremental-FLR/par-ninc-flr**). The experiment is summarized in Tab. 5.4 row 2 and detailed results are shown in Tab. C.4 in the appendix. Tab. 5.5 shows the state change compared to the run without FLR.

The success state of 9 *core_ext_con** benchmarks could be improved (files with rank 2.2, 2.5, 2.9 ...). These benchmarks are solved by the FLR with result *unsat*. For the rest of the benchmarks FLR leads to similar or worth runtime behaviour like it was observed in the incremental mode.

5.2.4 Results for Different Configurations

This section will show the results of PBoolector in incremental mode with different configurations. The detailed tables in this section show only the results for benchmarks with success state ++ and +.

PBoolector Look-Ahead Type for Satisfiable Instances

First an evaluation is done to run PBoolector with look-ahead type 15 (**Parallel-Incremental-Sat/par-inc-sat**). This type tries to reduce hard multiplication or adder circuits by setting their parameters to be equal 0 or 1. This look-ahead type is assumed to be able to improve the results only for satisfiable instances (see Sect. 4.3.5). The results are shown in Tab. 5.6 row 2 respectively the detailed results are listed in Tab. C.5 in the appendix. The results are

quite similar to the default Parallel-Incremental run (row 1). Actually there are two satisfiable files (2.11, 2.20) with success state improvement compared to Parallel-Incremental. Several unsatisfiable instances exhibit a more or less big runtime slowdown. As it was expected, the runtimes could only be improved on the satisfiable instances containing multiplication or adder circuits.

PBoolector with different conflict limits

In the following experiments an evaluation is done by running PBoolector with different limit configurations. Many benchmarks produce a high number of subproblems. To reduce the number of produced problems it is tried to increase PBoolector's limit configuration. Two experiments are done with different limit configurations.

- **Parallel-Incremental-HighLimit1** (**par-inc-hl1**):
  `-lmin 10000 -l 30000 -lmax 100000`
- **Parallel-Incremental-HighLimit2** (**par-inc-hl2**):
  `-lmin 30000 -l 50000 -lmax 200000`

The results are shown in Tab. 5.6 row 3 and 4 with the details in Tab. C.6 and Tab. C.7 in the appendix. The success state change are shown in Tab. 5.7 and 5.8. The high limit configurations are able to improve the results of the default PBoolector configuration. It is remarkable that the number of benchmarks with state −− can be decreased by increasing the conflict limits. The number of benchmarks with success state + could be increased with higher limits. Only the number of files with success state ++ could not be improved. Configuration par-inc-hl1 just solves 5 benchmarks with state ++ whereas default PBoolector had 9 benchmarks with ++. On the other hand 4 benchamrks moved from o to ++ with configuration par-inc-hl2. By taking a closer look at the detailed results mostly files could be improved where sequential Boolector is quite efficient anyway. Runtime improvements on files with slow sequential Boolector runtimes would be more interesting. But finding an optimal limit configuration seems to have improvement potential for the algorithm.

compare		filter	number of benchmarks					
ref	to		total	++	+	−	−−	o
base-inc	pbt	sequ-btor> 100.0	371	9	30	38	93	201
base-inc	par-inc-sat	sequ-btor> 100.0	371	8	30	37	94	202
base-inc	par-inc-hl1	sequ-btor> 100.0	371	5	39	65	57	205
base-inc	par-inc-hl2	sequ-btor> 100.0	371	9	55	70	36	201

Table 5.6: Runtime comparison for different parallel PBoolector configurations in incremental mode.

par-inc \ par-inc-hl1	++	+	−	−−	o
++	5	0	0	0	4
+	0	22	7	1	0
−	0	**10**	22	6	0
−−	0	**7**	**36**	50	0
o	0	0	0	0	201

Table 5.7: Success state change between Parallel-Incremental and Parallel-Incremental-HighLimit1.

par-inc \ par-inc-hl2	++	+	−	−−	o
++	5	0	0	0	4
+	0	24	6	0	0
−	0	**17**	21	0	0
−−	0	**14**	**43**	36	0
o	4	0	0	0	197

Table 5.8: Success state change between Parallel-Incremental and Parallel-Incremental-HighLimit2.

PBoolector-sp - Start Parallel Instance

The final run shows the results for the competitive version of PBoolector with option "`-sp`" (see Sect. 4.4.1). For a comparison, the results of all 2488 files are shown in the scatter-plot in Fig 5.4. The Sequential-Boolector implementation is compared to the **Parallel-Incremental-Competitive** (**par-inc-comp**) run. For almost all benchmarks par-inc-comp has similar or better runtimes. The cluster at the bottom is the same cluster which was already observed by comparing incremental and non-incremental base runtimes. Below the diagonal to the right are the few hard benchmarks where PBoolector can significantly outperform Boolector. On benchmarks which are far above the diagonal, the mentioned cache problems of the parallel instance can be observed. Heavy core utilisation of PBoolector negatively influences the parallel sequential solver instance. Actually results are influenced more by the caching issue than by PBoolector improvements. Tab. 5.9 shows the accumulated runtimes for both implementations. The competitive version of PBoolector can not improve the sequential Boolector on the *sc14* competition benchmarks.

5.2.5 Summary

No clear behaviour could be observed with different configurations. Each configuration can solve some benchmarks efficiently whereas on other configurations the runtime for the same benchamrk get worse. For instance there are some

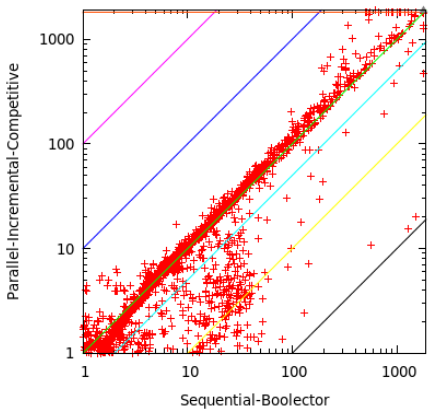


Figure 5.4: Runtime comparison of the competitive PBoolector (Parallel-Incremental-Competitive) vs. the Sequential-Boolector, for all 2488 benchmarks.

Sequential-Boolector		Parallel-Incremental-Competitive	
solved	sum[s]	solved	sum[s]
2336	390587	2316	412802

Table 5.9: Number of solved instances and accumulated runtimes for Sequential-Boolector and Parallel-Incremental-Competitive (oot=oom=1800s).

benchmarks (e.g., *smulov1bw32*) which have a nice speedup in the incremental mode whereas non-incremental mode leads to a clear slowdown. There are some benchmarks from the *ext_con** family which can be improved only in non-incremental mode with FLR. The right configuration can achieve some good results for benchmarks with the appropriate structure. A set of benchmark files from the category *log-slicing* are identified, which exhibit good performance with all configurations. The picture is mostly the same for different configurations: there are many files with a slowdown and some few files which can improve the reference times. The increase of the conflict limit configuration have shown the best improvements.

5.3 Crafted Benchmarks

In this section more detailed experiments are done on a set of crafted formulas. Parametrized formulas are generated with well known structure which should be appropriate for PBoolector. The evaluations (especially in Sect. 5.3.1) will show that the PBoolector implementation can be very powerful on hard benchmarks if they contain the right structure.

5.3.1 Lazy-If-Then-Else Example

The first experiment leads us back to the introductory example (Sect. 1.2), the Lazy-If-Then-Else example (*lazyitex*). The formula is taken from a publication comparing lazy and eager solving approaches for bit-vectors [36]. It was shown that the *lazyitex* structure is applicable for the proposed lazy bit-vector solving approach. This benchmark is relatively hard because it contains several nested 32 bit multiplication operations with conditional parameters. As shown in the example the formula can be efficiently solved if the search space is split by asserting the conditionals predicate. This example emphasizes the power of term rewriting rules in combination with search space splitting. After asserting a predicate the hard multiplication operations can be reduced by term rewriting rules. The subproblems get trivial and can be solved by the rewrite engine. This formula can even be solved without bit-blasting, a pure bit-vector rewriting solver would lead to a solution.

The original problem F was formulated as follows:

$$z \cdot (x_0 = y_0 \; ? \; y_0 \cdot (x_1 = y_1 \; ? \; y_1 : 1) : 1) \\ \neq (x_0 = y_0 \; ? \; x_0 \cdot (x_1 = y_1 \; ? \; x_1 \cdot z : z) : z) \quad (5.1)$$

To recap, the search space is split into three subproblems, which are trivially *unsat* after term-level rewriting:

(1) $F \wedge x_0 \neq y_0 \Rightarrow z \cdot 1 \neq z \Rightarrow z \neq z \Rightarrow \bot$
(2) $F \wedge x_0 = y_0 \wedge x_1 \neq y_1 \Rightarrow z \cdot y_0 \cdot 1 \neq z \cdot x_0 \Rightarrow z \cdot y_0 \neq z \cdot y_0 \Rightarrow \bot$
(3) $F \wedge x_0 = y_0 \wedge x_1 = y_1 \Rightarrow z \cdot y_0 \cdot y_1 \neq z \cdot x_0 \cdot x_1 \Rightarrow z \cdot y_0 \cdot y_1 \neq z \cdot y_0 \cdot y_1 \Rightarrow \bot$

The formula can be generalized using n nested conditions (= number of variable pairs x and y).

$$z \cdot (x_0 = y_0 \; ? \; y_0 \cdot (x_1 = y_1 \; ? \; y_1 \cdot (\ldots (x_{n-1} = y_{n-1} \; ? \; y_{n-1} : 1) \ldots) : 1) : 1) \\ \neq (x_0 = y_0 \; ? \; x_0 \cdot (x_1 = y_1 \; ? \; x_1 \cdot (\ldots (x_{n-1} = y_{n-1} \; ? \; x_{n-1} \cdot z : z) \ldots) : z) : z) \quad (5.2)$$

Crafted Lazy-If-Then-Else Benchmarks

A set of formulas was crafted using the general Equ. 5.2 and 64 benchmarks where generated by varying the number of conditionals n from 2 to 256 and the variable bit-width bw from 2 to 256. The runtimes of the sequential Boolector implementation is compared to the PBoolector implementation. The parallel implementation is run to split only on conditional nodes (BCONDN) using option "`-lt 4`". In the tables listed at the end of this section for each n (rows) and each bw (columns) the wall-clock runtime is printed in seconds.

Run 1: Boolector - Reference Times

The sequential reference implementation (Tab. 5.10) solves all instances with low bit-width quite fast. With increasing bit-width a clear exponential increase of runtime is observable. This behaviour comes from the fact that bit-blasted multiplication operations have an exponential increase of hardness with increasing bit-width (horizontal direction). Nearly each instance runs out of time with bit-width ≥ 8. There are 17/64 benchmarks are solved.

Run 2: PBoolector - Splitting on Conditionals

The second run (Tab. 5.11) shows solving times of PBoolector implementation with splitting on conditionals (BCONDN). Here the parallel implementation achieves a clear improvement to the sequential run 1. The PBoolector implementation solves 12 instances more within the time limit. The multiplications can be reduced by splitting and rewriting. Therefore the exponential behaviour with increasing bit-width can be reduced to a kind of linear behaviour: All benchmarks with 2 nested conditions n are now solved quite efficient, also with very high bit-width of 256. Splitting and term-rewriting leads to a super linear speedup. For example on the benchmark with $n = 2$ and $bw = 8$ a speedup of $745.18/1.69 = 440.93$ on the 4 core machine is achieved.

In the experiment a slowdown compared to Boolector can be observed on low bit-width benchmarks and increasing number of conditions (vertical direction). The slowdown can be explained by taking a look at the used heuristic: The order of which conditional should be split first, is calculated by the simple heuristic $pos + neg$ occurrences. Since each conditional occurs in the formula exactly one time in positive phase all conditional nodes get the same score value of 1. In this case the ordering depends on Boolector's internal DAG structure. If first the bottom most conditionals are used for splitting, subformulas can be reduced by rewriting but does not get trivial immediately. In the worst case for n nested conditionals, 2^n leaves in the search tree have to be generated until all these formulas are trivially solved by the rewrite engine. The vertical direction in the table shows this kind of exponential behaviour in the runtimes. The limitation of solving the instances by PBoolector is rather bounded by number of nested conditionals instead of the bit-width.

Run 3: PBoolector - Splitting on Conditions with Improved Heuristic

For the next run an improved heuristic was implemented which chooses the conditional in a more sophisticated way from top to bottom. Reconsider Equ. 5.2: If the formula is split by selecting the top most conditional first, $x_0 \neq y_0$ leads to $z \neq z$ (trivial *unsat*) and the branch can be closed. On the other branch with $x_0 = y_0$ the top most conditional can be removed. If the splitting is continued on the top most conditional of the remaining formula, one branch can immediately closed again and on the other branch the formula can be reduced. On the *lazyitex* benchmarks this leads to a maximal number of 2 subproblems each

round and produces maximal $n + 1$ leaves in the search tree. In comparison to run 1 where in the worst case 2^n leaves are produced. To select the top most conditional first the heuristic for BCONDN is improved (BCONDN-H1). The the number of not shared subnodes ($nssn$) is added: $h1 = pos + neg + nssn$. Not shared subnodes are all subnodes of the conditional which are referenced (direct or indirect) only by the conditional. If the conditional's predicate is assumed to be *true* all $nssn$ of the else branch can be removed too, because they are not referenced elsewhere in the DAG. The new heuristic $h1$ refines the approximation of the real formula reduction.

Using heuristic $h1$ clearly improves the results of run 2 by solving 22 instances more (Tab. 5.12). Benchmarks with small bit-width are able to outperform sequential Boolector with increasing number of variables. Also performance on higher bit-width benchmarks can significantly be improved. All remaining experiments in this work (also in Sect. 5.2) are done with the improved heuristic for conditionals.

A disadvantage that maximal 2 subproblems can be produced each round is the bad utilization of processor cores. However, the overall solving times by occupying just 2 cores is much more better than in the previous experiment which had extensive use of all 4 available cores.

Tab. 5.13 contains the percentage of time, spent in the SAT solver during all the search calls. As remainder, one search call consists of simplifying the formula and then after bit-blasting the SAT solver is called. On average 93.54% of time is spent in the SAT solver during the search phase. As we know that these instances can be solved on word-level only by rewriting, this ratio is quite high. The remaining three experiments try to improve the results by reducing the SAT solver's effort.

Run 4: PBoolector - Splitting on Conditionals and Non-Incremental Mode

One way to reduce the SAT solvers effort is to run the back end solver in non-incremental mode. The expectations where that less time is spent in the SAT solver (through possible optimisation in non-incremental mode) and the problems are solved more efficient. Tab. 5.14 shows the results. Contrary to the expectations the results get worse. Compared to run 3 where the SAT solver is used in incremental mode, 4 instances less can be solved within the timeout. Also the average SAT ratio of the search calls has been increased to 96.83% (see Tab. 5.15). It is noteable that in non-incremental mode two instance with high bit-width 256 can be solved now which were not solved in incremental mode. The non-incremental mode performs worse on medium bit-width formulas with a high number of nested conditionals.

Run 5: PBoolector - Splitting on Conditionals without Bit-Blasting

A more drastic way to reduce the SAT solver's work is to turn off bit-blasting. This leads to a pure rewriting solver. Layzitex benchmarks are solvable by splitting and rewriting. To turn off bit-blasting the implementation of the boolector_limited_sat() method was modified. If the method is called with limit 0 only rewriting is done. If rewriting does not lead to a result, *unknown* is returned else the result is returned. The PBoolector implementation also leads to a result if the formula is not solvable only by splitting and pure rewriting. If no look-ahead is found boolector_sat() is called without the limit, where the subproblem is bit-blasted and eagerly solved (subproblem is finished). Option "`-nf`" (not finish subproblems) is not possible in combination with limit 0 (minimal limit 0 and maximal limit 0), as the algorithm possibly does not terminate.

In Tab. 5.16 now all 64 benchmark files are solved within the time limit. Remarkably solving times with a maximum of 2.41 seconds are achieved also on the hardest instances. The pure rewriting setting is especially successful for the *lazyitex* benchmarks. On general benchmarks a clear loss of performance can be observed using this setting.

Run 6: PBoolector - Failed Literal Reduction

Finally in run 6, to reduce to the SAT solver's work, the failed literal reduction option "`-flr`" is enabled. The initial failed literal reduction can immediately solve the problem without starting any search call (and therefore the SAT solver) in the main loop. The runtimes shown in Tab. 5.17. For all benchmark $\sim 100\%$ of the runtime is spent in the failed literal reduction. All 64 instances are solved with similar performance as in run 5.

n	$bw2$	$bw4$	$bw8$	$bw16$	$bw32$	$bw64$	$bw128$	$bw256$
2	3.57	1.55	745.18	oot	oot	oot	oot	oot
4	3.60	3.84	oot	oot	oot	oot	oot	oot
8	4.05	5.35	oot	oot	oot	oot	oot	oot
16	4.05	19.11	oot	oot	oot	oot	oot	oot
32	0.49	41.03	oot	oot	oot	oot	oot	oot
64	0.56	105.26	oot	oot	oot	oot	oot	oom
128	0.81	405.29	oot	oot	oot	oot	oot	oom
256	1.60	1637.27	oot	oot	oot	oot	oom	oom

Table 5.10: Run 1 - Boolector - Reference times in seconds. 17/64 solved.

n	$bw2$	$bw4$	$bw8$	$bw16$	$bw32$	$bw64$	$bw128$	$bw256$
2	0.99	1.08	1.69	1.77	2.44	3.74	9.26	41.17
4	2.06	1.81	2.23	4.89	15.09	43.75	73.86	oot
8	2.01	3.05	4.85	12.81	46.45	948.12	oot	oot
16	2.06	13.33	841.23	oot	oot	oot	oot	oot
32	0.33	486.17	oot	oot	oot	oot	oot	oom
64	0.47	oot	oot	oot	oot	oot	oot	oom
128	4.44	oot	oot	oot	oot	oot	oom	oom
256	25.76	oot	oot	oot	oot	oot	oom	oom

Table 5.11: Run 2 - PBoolector (`-lt 4`) - Splitting on conditionals BCONDN. Wall-clock times in seconds. 29/64 solved.

n	$bw2$	$bw4$	$bw8$	$bw16$	$bw32$	$bw64$	$bw128$	$bw256$
2	3.44	0.88	3.93	4.81	5.59	7.15	17.02	55.65
4	3.78	4.17	5.28	6.51	11.49	33.85	83.66	160.67
8	3.69	4.59	6.39	9.09	22.72	68.39	150.74	oot
16	3.56	5.46	8.19	17.42	53.37	146.27	329.53	oot
32	0.36	4.67	12.45	36.81	135.66	362.37	oot	oom
64	0.72	11.84	29.49	106.43	388.43	1376.60	oot	oom
128	1.04	32.24	85.61	339.46	1254.98	oot	oom	oom
256	2.97	101.56	289.72	1212.78	oot	oot	oom	oom

Table 5.12: Run 3 - PBoolector (`-lt 4`) - Splitting on conditionals with improved heuristic BCONDN-H1. Wall-clock times in seconds. 51/64 solved.

n	$bw2$	$bw4$	$bw8$	$bw16$	$bw32$	$bw64$	$bw128$	$bw256$
2	50.00	96.70	99.04	97.65	95.68	91.94	89.06	86.40
4	66.66	99.34	99.01	98.35	97.74	97.17	94.66	86.82
8	86.20	99.18	98.63	97.73	97.09	95.78	90.06	-
16	94.44	98.79	98.12	97.43	96.29	93.14	84.41	-
32	95.45	98.54	97.93	96.60	94.63	89.14	-	-
64	98.14	98.27	96.95	94.84	92.53	89.69	-	-
128	98.10	97.84	95.33	92.78	90.36	-	-	-
256	97.95	97.22	93.60	91.20	-	-	-	-

Table 5.13: Run 3 - Time ratio spend in SAT solver during search in % (Average = 93.54%).

2	0.53	1.05	1.10	1.61	3.95	17.44	47.64	93.88
4	0.44	0.75	1.49	3.43	11.20	56.20	146.96	240.29
8	0.62	1.37	3.24	10.33	41.53	158.23	408.06	641.47
16	0.60	2.77	9.00	38.03	182.01	489.28	883.41	1703.49
32	0.84	6.83	32.45	124.98	608.05	1558.29	oot	oot
64	1.12	21.30	131.89	657.28	oot	oot	oot	oom
128	14.71	74.61	450.44	oot	oot	oot	oot	oom
256	61.86	292.50	oot	oot	oot	oot	oom	oom

Table 5.14: Run 4 - PBoolector (`-lt 4 -ni`) - Splitting on conditionals. The SAT solver runs in non-incremental mode. Wall-clock times in seconds. 47/64 solved.

n	$bw2$	$bw4$	$bw8$	$bw16$	$bw32$	$bw64$	$bw128$	$bw256$
2	80.00	97.83	99.36	98.80	98.64	98.89	98.26	96.21
4	91.35	98.88	98.73	98.42	98.43	98.73	97.85	94.31
8	91.11	98.77	98.50	98.46	98.47	98.25	97.04	91.73
16	93.63	98.03	98.43	98.46	98.66	97.73	94.54	87.85
32	96.94	97.65	98.41	98.24	98.32	97.10	-	-
64	98.06	97.62	98.51	98.64	-	-	-	-
128	97.12	97.52	98.19	-	-	-	-	-
256	96.53	97.76	-	-	-	-	-	-

Table 5.15: Run 4 - PBoolector (`-lt 4 -ni`) - Time ratio spend in SAT solver during search in %(Average = 96.83%).

n	$bw2$	$bw4$	$bw8$	$bw16$	$bw32$	$bw64$	$bw128$	$bw256$
2	0.97	0.87	0.83	0.82	0.81	0.89	0.88	0.85
4	1.08	1.03	0.99	1.04	0.96	1.01	1.04	0.99
8	1.02	0.97	1.04	0.99	1.05	1.00	0.97	1.02
16	1.04	0.99	0.94	0.99	1.02	0.97	1.04	0.99
32	1.01	1.06	0.98	1.03	1.07	1.00	1.16	1.12
64	1.07	1.12	1.11	1.11	1.06	1.17	1.12	1.16
128	1.33	1.29	1.33	1.34	1.29	1.34	1.34	1.19
256	2.28	2.30	2.07	2.30	2.39	2.14	2.39	2.41

Table 5.16: Run 5 - PBoolector (`-lt 4 -lmin 0 -lmax 0 -l 0`) - Splitting on conditionals without bit-blasting. Wall-clock times in seconds. 64/64 solved.

n	$bw2$	$bw4$	$bw8$	$bw16$	$bw32$	$bw64$	$bw128$	$bw256$
2	1.07	1.22	1.14	0.99	1.04	1.10	1.05	1.10
4	0.93	0.98	0.98	1.03	0.99	1.03	1.04	0.99
8	1.04	1.09	1.04	1.09	1.06	1.01	1.06	1.10
16	1.04	1.10	1.02	1.07	1.07	1.06	1.00	1.06
32	1.02	0.97	1.02	1.04	0.99	1.04	1.08	0.95
64	1.08	1.08	1.02	1.08	1.18	1.04	1.10	1.10
128	1.21	1.21	1.02	1.21	1.21	1.19	1.15	1.19
256	1.58	1.61	1.55	1.60	1.61	1.59	1.76	1.76

Table 5.17: Run 6 - PBoolector (`-lt 4 -flr`) - Doing an initial failed literal reduction. Wall-clock times in seconds. 64/64 solved.

5.3.2 Square of Sum

To analyze the impact of splitting on the least significant bit of multiplications a set of benchmarks were generated which checks the distribution rule of multiplications for the square of a sum. The square of a sum with 2 variables is defined as follows:

$$(x_0 + x_1)^2 = x_0^2 + 2x_0x_1 + x_1^2 \tag{5.3}$$

The general form of the square of a sum with n variables:

$$\left(\sum_{0\leq i\leq n} x_i\right)^2 = \left(\sum_{0\leq i\leq n} x_i^2\right) + 2\cdot\left(\sum_{0\leq i<j\leq n} x_i\cdot x_j\right) \tag{5.4}$$

Crafted Square of Sum Benchmarks

A set of formulas was constructed using Equ. 5.4 and 64 benchmarks are generated with varying number n of variables from 2 to 16 and variable bit-width bw from 2 to 256. To check if the formula is valid the negation of the formula is checked for *unsat* The runtimes of the sequential Boolector implementation is compared to the parallel PBoolector. The parallel implementation is run with look-ahead type MULNBIT0 ("`-lt 11`") to split on the multiplication's LSB. In the tables listed at the end of this section for each n (rows) and each bw (columns) the wall-clock runtime is printed in seconds.

Run 1: Boolector - Reference Times

Tab. 5.18 shows the reference times of the sequential implementation. The results are similar to the sequential results of the *lazyitex* benchmarks in the previous section. Formulas with small bit-with are solved quite efficient, whereas increasing the multiplication's bit-width exhibits an exponential increase in solving time.

Run 2: PBoolector - Splitting on Multiplication's LSB

Tab. 5.19 shows the results. The PBoolector implementation is able to solve 1 instance more than the sequential implementation. The improvement is not as significant as for the *lazyitex* benchmarks. In contrast to the *lazyitex* benchmarks here "only" linear speedup can be achieved. Bit-splitting can not simplify the formula with term rewriting rules. Therefore the speedup exhibits a linear behaviour with respect to the used cores. For instance the formula with $n = 4$ and $bw = 6$ achieves a speedup of $129.78/35.63 = 3.64$ on the 4 core machine.

Run 3: Treengeling - Comparison to a Parallel SAT solver

As for bit-splitting no rewriting rules can be triggered the runtimes of PBoolector are comparable to the parallel SAT solver implementation Treengeling. The 64 square of sum benchmarks (QF_BV) are converted to a CNF formula which where run on the SAT solver. The expectation where that PBoolector performs better than the SAT solver, as it is able to split with term level information. In the CNF encoding the information is lost and the multiplications LSB can not identified for splitting anymore. Results in Tab. 5.20 show the opposite, that Treengeling running on 4 cores is able to outperform PBoolector slightly. The same benchmarks are solved, but with better runtimes. This might be explained by the fact, that Treengeling is already a highly optimized tool whereas PBoolector is at an initial state of implementation. Another reason could be that the SAT solver is able to split the CNF formula in a much more fine grained way. Treengeling is able to select more promising Boolean variable whereas PBoolector just selects multiplication's LSBs.

Further Experiments

Several further enhancements where tried, but without improvements to run 2. The enhancements are only mentioned in the following without comparing explicitly the results. It was tried to split the multiplications on the most significant bit. Another experiment was done to split a multiplication not only on one bit, rather each multiplication was split on its n LSBs with $n = 2, 3, 4, \ldots$.

n	$bw2$	$bw4$	$bw6$	$bw8$	$bw10$	$bw12$	$bw14$	$bw16$
2	0.97	1.02	1.06	2.41	36.11	1652.99	oot	oot
4	1.30	1.17	129.78	oot	oot	oot	oot	oot
6	0.22	18.32	oot	oot	oot	oot	oot	oot
8	1.03	oot	oot	oot	oot	oot	oot	oot
10	1.33	oot	oot	oot	oot	oot	oot	oot
12	1.46	oot	oot	oot	oot	oot	oot	oot
14	1.51	oot	oot	oot	oot	oot	oot	oot
16	3.01	oot	oot	oot	oot	oot	oot	oot

Table 5.18: Run 1 - Boolector - Reference times in seconds. 16/64 solved.

n	$bw2$	$bw4$	$bw6$	$bw8$	$bw10$	$bw12$	$bw14$	$bw16$
2	1.11	1.34	1.20	2.23	13.77	380.90	oot	oot
4	0.88	1.00	35.63	oot	oot	oot	oot	oot
6	1.16	7.51	oot	oot	oot	oot	oot	oot
8	1.17	205.64	oot	oot	oot	oot	oot	oot
10	0.85	oot	oot	oot	oot	oot	oot	oot
12	1.12	oot	oot	oot	oot	oot	oot	oot
14	1.74	oot	oot	oot	oot	oot	oot	oot
16	4.77	oot	oot	oot	oot	oot	oot	oot

Table 5.19: Run 2 - PBoolector (`-lt 11`) - Splitting on multiplications LSB MULNBIT0. Wall-clock times in seconds. 17/64 solved.

n	$bw2$	$bw4$	$bw6$	$bw8$	$bw10$	$bw12$	$bw14$	$bw16$
2	0.82	0.95	0.95	2.76	11.44	161.70	oot	oot
4	1.07	1.35	19.73	oot	oot	oot	oot	oot
6	0.55	5.01	oot	oot	oot	oot	oot	oot
8	1.18	146.50	oot	oot	oot	oot	oot	oot
10	1.63	oot	oot	oot	oot	oot	oot	oot
12	0.51	oot	oot	oot	oot	oot	oot	oot
14	1.76	oot	oot	oot	oot	oot	oot	oot
16	1.77	oot	oot	oot	oot	oot	oot	oot

Table 5.20: Run 3 - Treengeling - Parallel SAT solver. Wall-clock times in seconds. 17/64 solved.

5.3.3 Carry Safe Adder

To evaluate the impact of the ADDNBIT0 look-ahead type a benchmark set where generated which verifies a Carry Save Adder (CSA). A CSA is a hardware addition circuit to fast compute the sum of 3 or more bit-vectors. The disadvantage of the conventional Ripple Carry Adder (RCA) summing up 3 (or more) numbers is that the carry bit has to be propagated from one adder unit to the other. For instance the sum of 3 bit-vector numbers $a+b+c$ is calculated in a RCA chain by first, calculating $tmp = a + b$ and second, doing the remaining sum $s = tmp+c$ with the carry out bit of the first sum. Each addition is delayed by its previous additions. To give an idea of how the delay is omitted in a CSA, here a benchmark is sketched which verifies the correctness of a CSA summing up 3 numbers:

$$\begin{aligned} ps &= a \oplus b \oplus c \wedge \\ sc &= (a\&b)|(a\&c)|(b\&c) \wedge \\ y &= ps + (sc << 1) \wedge \\ x &= a + b + c \wedge \\ x &\neq y \end{aligned} \tag{5.5}$$

The partial sum ps is the sum of the three numbers for each bit without a carry. For three binary numbers this is done by a simple xor operation. The shift carry sc computes a separated carry for each bit. Again for 3 binary numbers this is a simple logical operation. The CSA sum y finally is done by shifting sc and perform one arithmetic plus operation (with a RCA). The formula is *unsat* and therefore the CSA is verified to be equal the simple RCA chain (x). The circuit looks similar for adding more than three numbers, the CSA contains only fast logical operations except one final arithmetic Ripple Carry Adder. No adder unit needs to wait for the propagated carry bit.

Crafted Benchmarks and Results

For benchmarks generation an already existing tool GenCSA (implemented at the FMV institute at JKU) was used, which generates parametrized formulas similar to Equ. 5.5. There are 64 benchmarks generated with varying number of variables n from 2 to 16 (number of variables which are summed up by the CSA) and varying bit-with from 2 to 16. The experiments are done in an analogous manner as in Sect. 5.3.2. The formulas are split on the addition's LSBs and Boolector (Tab. 5.21) is compared to the PBoolector (Tab. 5.22) and the Treengeling (Tab. 5.23) implementation. The runtimes for PBoolector are worse than for the sequential Boolector. The sequential implementation solves 34/64 benchmarks within the timeout. The PBoolector implementation solves 27/64 benchmarks with a significant slowdown. The Treengeling implementation clearly outperforms PBoolector (31/64 benchmarks solved), but cannot improve the sequential Boolector results. It could not be verified that splitting on addition's LSB give some improvements. Contrary to multiplications, it seems that setting an addition's bit is not worth for splitting the search space. There-

fore the combined look-ahead types (see Sect. 4.3.5) do not select ADDNBIT0 for splitting.

n	$bw2$	$bw4$	$bw6$	$bw8$	$bw10$	$bw12$	$bw14$	$bw16$
2	1.19	1.25	1.25	0.36	0.36	0.48	0.36	0.36
4	1.18	1.14	1.63	2.51	4.99	5.56	15.84	13.00
6	1.14	2.34	19.31	100.92	602.13	1379.47	1628.03	oot
8	1.01	8.20	221.60	oot	oot	oot	oot	oot
10	0.94	39.12	oot	oot	oot	oot	oot	oot
12	1.10	227.14	oot	oot	oot	oot	oot	oot
14	0.79	253.55	oot	oot	oot	oot	oot	oot
16	1.26	1018.89	oot	oot	oot	oot	oot	oot

Table 5.21: Run 1 - Boolector - Reference times in seconds. 34/64 solved.

n	$bw2$	$bw4$	$bw6$	$bw8$	$bw10$	$bw12$	$bw14$	$bw16$
2	1.06	0.89	0.84	0.82	0.85	0.85	0.84	0.84
4	0.84	0.89	1.16	1.74	8.23	25.92	40.28	42.80
6	0.85	3.39	148.80	1360.44	oot	oot	oot	oot
8	1.10	57.98	oot	oot	oot	oot	oot	oot
10	1.19	508.12	oot	oot	oot	oot	oot	oot
12	110.07	oot	oot	oot	oot	oot	oot	oot
14	92.11	oot	oot	oot	oot	oot	oot	oot
16	707.75	oot	oot	oot	oot	oot	oot	oot

Table 5.22: Run 2 - PBoolector (`-lt 13`) - Splitting on addition's LSB ADDNBIT0. Wall-clock times in seconds. 27/64 solved.

n	$bw2$	$bw4$	$bw6$	$bw8$	$bw10$	$bw12$	$bw14$	$bw16$
2	1.64	1.64	0.69	1.51	1.57	0.57	1.57	1.51
4	1.41	1.57	2.11	4.30	12.30	56.17	40.46	68.70
6	1.06	2.80	73.63	328.70	1091.42	oot	oot	oot
8	1.15	12.01	oot	oot	oot	oot	oot	oot
10	1.23	78.48	oot	oot	oot	oot	oot	oot
12	1.22	211.84	oot	oot	oot	oot	oot	oot
14	1.75	527.05	oot	oot	oot	oot	oot	oot
16	1.78	831.22	oot	oot	oot	oot	oot	oot

Table 5.23: Run 3 - Treengeling - Parallel SAT solver. Wall-clock times in seconds. 31/64 solved.

5.4 Selected Benchmarks

In the final section of the evaluation chapter a closer look is taken at the structure of successful SMT-LIB benchmark files. Some more file structures will be identified which are appropriate for PBoolector.

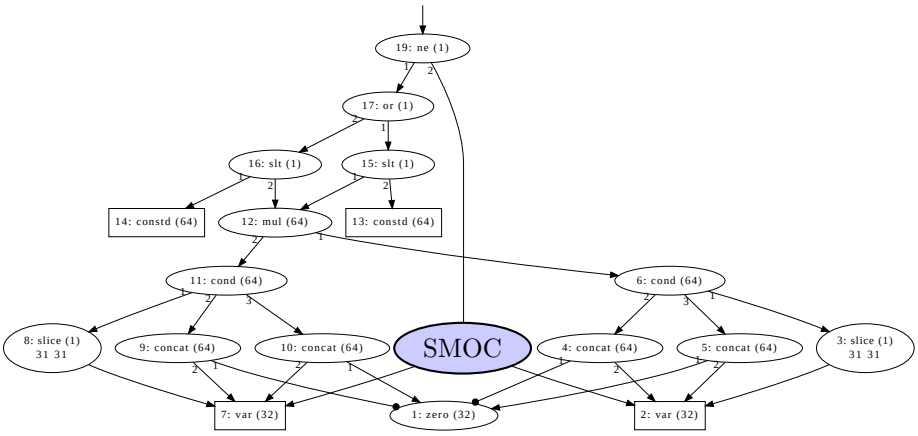


Figure 5.5: The smulov1bw32 benchmarks structure. A signed multiplication overflow circuit (SMOC) is checked against the naive implementation.

5.4.1 Smulov1bw*

Some of the few files which show good performance in Sect. 5.2 are the *smulov1bw** files from the SMT-LIB category *brummmayerbiere2*. This benchmark set is verifying a signed multiplication overflow circuit (SMOC) with different bit-widths'. The circuit is checked against a naive multiplication overflow detector. The following formula checks a SMOC with bit-width 4 against the naive implementation like it is done in the benchmarks:

$$(p_8 = sext(a_4)_8 \cdot sext(b_4)_8 \wedge (p_8 < -8 \vee p_8 > 7)) \neq SMOC(a_4, b_4)$$

The index denotes the bit-vector's bit-width. If the formula is *unsat* the left side (of the inequality) which is the naive implementation is equivalent to the optimized SMOC circuit on the right side. For the naive implementation the 4 bit-vectors a and b are sign extended to a 8 bit vector. The two 8 bit vectors are multiplied and if the product is smaller than $INT\_MIN_4 = -8$ or is greater than $INT\_MAX_4 = 7$, then a 4 bit multiplication overflow have been occurred. For better reading here the variable p which saves the product is introduced. The SMOC which is stated here as an abstract function is a fast overflow detector consisting of one 5 bit multiplication and several logical connectives. The details how the fast SMOC is implemented are found in [17]. A 4 to 8 bit sign-extension is encoded in Boolector with an if conditional and 2 concatenations:

$$sext(a_4)_8 \equiv a_4[3]_1 \; ? \; 1_4 \circ a_4 : 0_4 \circ a_4$$

benchmark	Boolector[s]	PBoolector -lt 14		
		tot (wc)[s]	search tot(wc)[s]	search tot(cpu)[s]
smulov1bw16	9.66	6.67	6.44	21.2
smulov1bw24	133.8	79.00	78.26	224.74
smulov1bw32	1061.94	466.61	463.75	1421.85
smulov1bw48	oot	oot	-	-

(a) Combined look-ahead (`-lt 14`).

benchmark	Boolector[s]	PBoolector -lt 4		
		tot (wc)[s]	search tot(wc)[s]	search tot(cpu)[s]
smulov1bw16	9.66	10.47	10.46	12.79
smulov1bw24	133.8	128.81	128.78	136.72
smulov1bw32	1061.94	1064.22	1064.18	1090.06
smulov1bw48	oot	oot	-	-

(b) Split only on conditions (`-lt 4`).

Table 5.24: Comparison of Boolector vs. PBoolector implementation on *smulov1bw** benchmarks in seconds. For BPoolector the total runtime and the runtime of the search phase is given.

If the MSB of a is one, a is concatenated with a 4 bit one-constant else with a 4 bit zero-constant. The bottleneck of this benchmark set is the high bit-width of the multiplications. To check a 32 bit SMOC leads to a formulas which contains one 64 bit multiplication (in the naive implementation) and one 33 bit multiplication (in the SMOC). Fig. 5.5 shows the formula as DAG. The optimized SMOC is abstracted to one node. PBoolector splits the formula for example by setting conditional 6 to be false. Node 6 is replaced with node 5 which is the sign extension concatenation. A 32 bit zero-constant is concatenated with a 32 bit variable. In the input parameter 1 of the 64 bit multiplication (id 12), now half of the bits are set to the fixed value 0. With the possibility to fix half of the bits in the multiplication's parameter, the SAT solver's performance significantly can be improved. Different to the *lazyitex* benchmarks in Sect. 5.3.1, here multiplications are not eliminated but its solving effort can be reduced. Splitting on the 2 conditionals (node 6 and 11) results into 4 subproblems which are relatively simpler because of the fixed bits in the multiplication parameters.

The results for the combined look-ahead are shown in Tab. 5.24a. PBoolector is able to outperform the sequential implementation very well. Additionally to the total runtime the wall-clock time and cpu time of the search phase is printed. The SMOC contains several one-bit "and" and "slice" operations. With the combined look-ahead the formula is first split on conditionals and then on the one-bit logical connectives of the SMOC. Tab. 5.24b lists the runtimes with

splitting only on conditionals ("`-lt 4`"). The whole benchmark consist of 2 conditionals which produce maximal 4 subproblems to solve in parallel. This experiment should evaluate whether if it is sufficient to split only on the hard parts of the formula (the sign extension conditionals) or not. Actually with the combined look-ahead the wall-clock time is clearly better. But it can be observed that if more splitting is done some overhead is introduced. For instance the 32 bit benchmark file occupies with pure conditional splitting only 1090.06 seconds CPU time whereas the combined look-ahead takes 1421.85 seconds. The disadvantage of producing a low number of subproblems is that they can not be solved with reasonable speedup in parallel. For this benchmarks it is crucial to split on the conditional (to make the hard multiplication simpler). All further splitting introduces some overhead, but the splitting is important to get good parallelization capabilities. For most other SMT-LIB benchmarks which exhibit bad performance probably the introduced overhead is too high.

5.4.2 MultiplyOverflow

The benchmarks *challenge_multiplyOverflow* (Tab. C.1 in the appendix rank 2.14) verifies the overflow detection like it is implemented in the C-API call calloc():

```
1 void *calloc(size_t a, size_t b)
2 {
3   if( ((size_t)-1) / a < b ){ errno = ENOMEM; return NULL; }
4   return memset(malloc(a*b), 0, a*b);
5 }
```

The verification of the 32 bit unsigned multiplication overflow is checked against the naive implementation. The naive implementation is similar to the signed multiplication overflow check which was shown in the previous section, with the difference that no conditional is needed for the unsigned value extension.

$$p_{64} = (0_{32} \circ a_{32}) \cdot (0_{32} \circ a_{32}) \wedge p_{64}[63:32] > 0_{32} \neq ((0_{32} - 1)/a_{32}) < b_{32}$$

The benchmark consists only of one look-ahead node, namely the 64 bit multiplication which can be split on the LSB. Bit splitting on the hard multiplication can improve the result with a speedup of 1.09. With this file it could be shown that bit splitting is also successful on a real world benchmark.

In the category *challenge* the benchmark *integerOverflow* (Tab. C.1 in the appendix rank 5.196) verifies that no signed multiplication overflow can occur in the function baz() if the variable $y \neq 0$:

```
1 int baz(int x, int y)
2 { return (x / y) * y; }
```

benchmark	Boolector[s]	PBoolector[s]	status
problem_3	22.98	oot	*unsat*
problem_7	992.74	oot	*sat*
problem_8	992.88	oot	*sat*
problem_10	oot	1.83	*unsat*
problem_11	oot	1.79	*unsat*
problem_12	oot	oot	*unknown*
problem_13	oot	oot	*unknown*
problem_14	1.09	0.96	*unsat*
problem_16	oot	2.99	*unsat*
problem_17	oot	oot	*unknown*
problem_18	oot	oot	*unknown*
problem_19	oot	22.14	*unsat*
problem_20	145.34	oot	*unsat*
problem_21	oot	1.82	*unsat*
problem_22	191.71	oot	*unsat*
problem_23	1458.02	oot	*unsat*

Table 5.25: Runtime comparison of selected *calypto* benchmark files.

For this benchmark PBoolector should be also capable to outperform Boolector because it is verified against a naive signed multiplication overflow implementation like it is done in the previous section. Actually the 32 bit overflow detection is still to hard for PBoolector that it can be solved within the time limit.

5.4.3 Calypto

The competition benchmarks of 2014 does not contain benchmarks from the category *calypto*. In this family several benchmarks are found which are quite fast for PBoolector. The benchmark set contains 24 small and very hard problems. One reason for its hardness is that the problems contain hard multiplication and/or adder operations with high bit-width. Tab. 5.25 shows the runtime comparison between Boolector and PBoolector. Benchmarks which are solved in less than 10 seconds by both implementations are not shown in the table (except *problem_14*). In the following the successful benchmarks are examined in detailed.

Problem_14

In Tab. 5.25 *problem_14* has fast runtime for both implementations. Actually at the beginning of the experiments Boolector was not able to solve this file within 1800 seconds whereas PBoolector could solve it within few seconds. During the investigation to find out why the benchmark works for PBoolector some new rewriting rules have been found such that Boolector is able to solve the file efficiently too (therefore the fast runtime of 1.09 seconds). In the following, first the reason for fast PBoolector solving is shown and second the new rewriting rule is explained.

This benchmark contains a similar pattern (sign-extension pattern) as the benchmarks in Sect. 5.4.1. The structure of *problem_14* is shown in Fig. B.3 in the appendix. The DAG is relatively small but contains one hard 32 bit and one hard 33 bit multiplication . Setting for example node 8 to be *false* can fix the 16 LSBs of the 32 bit multiplication's (id 12) parameter (id 11). Further the 17 MSBs of the 33 bit multiplication's (id 21) parameter (id 16) are fixed as well. Splitting on node 8 leads to 2 subproblems which are considerably simpler than the original formulas, because half of the multiplication's parameter bits can be fixed to a constant value.

If a closer look is taken to the DAG it can be observed that 32 bits of the two multiplications are structural identical. The multiplication operations can not be shared on term-level as its bit-with is different. But parts of the multiplication can be shared on the AIG level. The problem here is that parameters of the multiplication are exchanged: multiplication 21 has the left part as first parameter and multiplication 12 has the right part as first parameter. As multiplication is commutative, a rewriting rule was introduced in Boolector such that the multiplications parameter ordering on the AIG level is normalized. This is done for example by exchanging the parameters for multiplication 12: node 11 gets parameter 1 and node 6 gets parameter 2. Then, on the AIG level 32 bits of the two multiplications can share its structure and the problems gets very trivial for the SAT solver. Adding this new rewriting the formula can be solved by Boolector in 1.09*s* without any splitting. Actually with this rule a new class of rewriting could be introduced to Boolector. The new rewriting rule operates on the intermediate AIG layer instead on the formula DAG (term layer).

Problem_16

Benchmark *problem_16* is solved by PBoolector in 2.99 seconds whereas Boolector runs out of time. The benchmark file *problem_16* contains 5 hard 32 bit multiplications. The inputs of the multiplications are sign-extended variables. Similar to the *smulov1bw** benchmarks presented in section 5.4.1 parts of the multiplication bit-vectors can be set to a fixed value by splitting on the conditional node (the encoded sign-extension operation). Fig. B.4 in the appendix shows the formula as DAG representation. As the problem is solved without producing a lot of subproblems the real execution tree of a PBoolector can be visualized (Fig. B.5 in the appendix). On the top the original formula is split on a conditional. Two subproblems are produced in round 1, one trivial *unsat* subproblem (left) which can immediately be closed and one reduced subproblem (right). The reduced formula is split until both generated subproblems can be solved within the limit (round 4). On the diagonal the formulas size (hardness) reduction can be observed. Similar to the *lazyitex* benchmarks with the improved heuristic (Sect. 5.3.1), in each round only two problems can be run in parallel because the left branch can always be closed immediately.

Problem_19

Benchmark *problem_19* is a bigger formula than the previous presented ones. It contains several logical operations and three multiplications. One 31 bit multiplication seems to be the bottleneck for this benchmark. The partial DAG of the formula shown in Fig. B.6 in the appendix serves as input parameter for the 31 bit multiplication. If the formula is split by setting the one bit variable (id 287) to *true* or *false* 27 bits of the input parameter can be fixed by the concatenation operators to constant 1 or constant 0. Fixing the multiplication's bits leads to a remarkable speed up in solving times. Actually this benchmark is the reason why the heuristic $pocc + nocc$ (see Sect. 4.3.4) was chosen for the VAR1 look-ahead type. The variable is referenced 27 times positively and 0 times negatively. The multiplication heuristic with score $27 \cdot 0 = 0$ would not select the variable as look-ahead node whereas the addition heuristic with score $27+0 = 27$ is able to select the node for splitting. The pattern shown in Fig. B.6 is just an unconventional way to encode a sign extension. The one-bit variable 287 is the sign-bit of variable 314.

With some new rewriting rules the encoding of the partial DAG could be reduced to a conventional sign extension like it was shown in Sect. 5.4.1 (The index denotes the bit-width):

$$x_1 \circ x_1 \Rightarrow x_1 ? 1_2 : 0_2$$
$$(x_1 ? 1_n : 0_n) \circ x_1 \Rightarrow x_1 ? 1_{n+1} : 0_{n+1}$$

If the rules are applied recursively on the pattern, rewriting would end up in the conventional sign-extension pattern $x_1 ? 1_{27} : 0_{27}$. It is left for future work to find out if such new rewriting rules could speed up Boolector. I assume Boolector will not speed up because, like for other shown benchmarks, the multiplication only gets easy if the bits of its parameters can be fixed directly. The bits of the parameter can only be fixed with help of PBoolector by splitting on the conditional node of the resulting sign-extension pattern.

Other Benchmark Files

There are three more benchmarks which run into timeout with Boolector and which are solved by PBoolector (*problem_10/11/21*). This slow solving time is strange, because all 3 file where solved with a previous Boolector version within 2 seconds. A short investigation identified the non-incremental mode of Boolector as the cause of this slowdown. PBoolector in incremental mode with limit -1 solves the three files within $5s$. If PBoolector runs in non-incremental mode with limit -1 all three files run into timeout.

There are several benchmarks left where PBoolector runs into timeout whereas some of these files are solved by Boolector. It was tried to improve the results of PBoolector by changing the limit and run it with the individual look-ahead types to solve one of the remaining benchmarks. None of these informal experiments was able to improve the results. The benchmarks, not solved by BPoolector,

also contain several sign-extension operations. On some benchmarks less bits are extended and on some benchmarks the extensions serve as input for adders or other "simple" operations. It might be that PBoolector is just able to outperform Boolector if the sign-extensions are inputs of multiplications and a huge number of bits are extended, which was the case for the successful benchmarks. The satisfiable benchmarks (*problem_7/8*) are not applicable for the look-ahead types which are defined to improve satisfiable instances (MULNV0, MULNV1, ADDNV0), because the containing multiplication/additions have no variables as parameters.

5.4.4 Log Slicing

In the category *log-slicing* several benchmarks with reasonable speedup with PBoolector are identified (see Tab. C.1 in the appendix). The benchmarks where developed along complexity evaluation of fixed-size bit-vector formulas [33]. It was shown that a subset of base operations (bit-wise operations, equalities and slicing) suffice to derive NEXP-TIME hardness. A set of benchmarks was generated to check the translation of arithmetic operations to the base operations. Benchmark *bvudiv_16* for instance checks the translation of a 16 bit unsigned division. Because the translation is done by introducing new variables the DAG structure is rather flat for the *bv_** benchmarks and contains various constraint equality nodes. The look-ahead candidates are some few one-bit slice nodes and equalities. Several SLICE1 nodes have only one occurrence, hence it is important for these benchmarks that the heuristic $pocc + nocc$ is used. Actually no clear reason could be found why this kind of benchmarks work well in the parallel approach. One reason might be that the search tree here is bounded by the few number of look-ahead candidates. After some rounds no look-ahead can be found and the problems are finished.

5.4.5 Galois - IffyInterlaevedModMul-8

Also good performance improvement could also achieved for the benchmark *iffyInterleavedModMult-8* (see Tab. C.1 in the appendix, rank 1.3). The benchmark is explained as follows in the formula header: "The benchmarks are checking whether an algorithm for computing modular multiplication by interleaving the multiplication and residue calculations, is equivalent to an algorithm that first multiplies the two inputs, and then performs the residue calculation on the result." It contains 31 BCOND, 16 SLICE1, 1 BEQ1VN and 1 MULNBIT nodes (with bit-width 16) as look-ahead candidates. As in the previous section no clear reason for the successful PBoolector run was found.

There are some properties which this benchmark has in common with the successful *log-slicing_bv** benchmarks: The formula is quite hard with respect to its size (the DAG contains 247 nodes). The structure of the DAG is flat and it contains several equality constraint nodes. The look-ahead candidates are mostly direct children of the constraint nodes. There is one difference which

can be observed that the successful *log-slicing* benchmarks produce few subproblems ($\sim$ 25) whereas the *iffyInterleavedModMul-8* benchmark produces a lot of subproblems (442).

Chapter 6

Conclusion

6.1 Summary

A parallel SMT solving approach for the quantifier-free fragment of fixed-size bit-vectors, based on the Cube and Conquer algorithm, was presented. The approach was implemented in the tool PBoolector based on the state-of-the-art SMT solver Boolector. No single algorithm was able to achieve overall good performance for all kind of benchmarks. On many benchmarks a huge slowdown was observed by the algorithm. The used look-ahead techniques are often not applicable for search space splitting because the solving effort of the produced subproblems can not be reduced. Anyhow, it could be shown that the algorithm is able to outperform the sequential Boolector implementation on formulas containing certain patterns, with up to super-linear speedup. The successful benchmarks are mostly related with hard multiplication circuits, which is one of the bottlenecks in the bit-blasting algorithm. The identified patterns can be classified into 3 types.

- It was shown that nested if-then-else conditionals with interleaving multiplications can be solved very efficient with the PBoolector approach (see Sect. 5.3.1).
- It could be shown that sign-extension encodings, in practice often used with multiplication overflow verification, is able to achieve a well speedup on several real-world benchmarks from the SMT-LIB (see Sect. 5.4).
- It was shown that bit-splitting on the least significant bit for hard multiplication circuits achieves a well linear speedup (see Sect. 5.3.2). On the other hand it was shown that bit-splitting could not achieve a positive speedup for hard adder circuits (see Sect. 5.3.3).

Some additional formula patterns were detected, different to the mentioned multiplication patterns, which work well for PBoolector. A nice side effect was that a new rewriting rule on the AIG Layer was found during closer investigation of the evaluation results of PBoolector (see Sect. 5.4.3 *problem_14*).

6.2 Future work

To conclude the thesis the final section will give an outlook of possible future extension and research proposals to improve the introduced algorithm. First two rather small implementation extensions and next some conceptual improvements are proposed.

As first small extension the update function can be enhanced. The number of solved subproblems by the rewrite engine and the SAT solver could be counted separately. If the number of subproblems solved by the SAT solver is greater than the number of closed subproblems than the limit is increased as it was proposed. But, if a lot of subproblems are solved by the rewrite engine the limit should be decreased. In the case that the SAT solver does not contribute to solve the problems its work effort should be reduced by decreasing the limit. The second extension is to add term elimination in the rewriting engine similar to the variable elimination. This extension would not restrict several look-ahead types to nodes, which contain at least one variables as parameter (BEQ1VN, MULNV0, ...). The rewrite engine would be able to eliminate for example terms like $a \cdot b$ with $a + c$ if it is known that $a \cdot b = a + c$. The evaluation, if these two extensions can bring some performance gain is left for future work.

The assumption that a formula's hardness is proportional to its word-level DAG size might fail for many formula instances. In the future a closer investigation on that topic might be beneficial. Empirical measurements, which compare the assumed hardness and the real solving effort (runtime of the SAT solver) of the generated subproblems are proposed for evaluation. The hardness measure might be improved by estimating the number of clauses and variables, which will be generated by bit-blasting.

Probably, there is also potential in the improvement of the heuristics for scoring the look-ahead candidates. The used heuristics for PBoolector are counting the number of occurrences. A closer investigation can be done by comparing the score of the heuristic with the score of the probing method (which is assumed to select the perfect look-ahead candidate). The Task is to find the best heuristic function for each look-ahead type that maximizes the correlation between the measured reduction and the heuristically estimated reduction. This is obviously a hard task.

A further possible future direction is to detect hard parts in the formula. The evaluation has shown that PBoolector is successful if the solving effort of hard arithmetic formula parts can be reduced. If it is possible to find and reduce these parts from the beginning of the search tree generation, the introduced overhead which was observed for many benchmarks might be reduced. For example if a hard multiplication circuit is split first on an equality node which is not related with the multiplication, two subproblems are produced (still containing the hard multiplication part) which the SAT solver has to solve. A

simple heuristic improvement would be to weight the score with the number of arithmetic operations that are related with the look-ahead node. The more arithmetic operations the higher the weight. The already existing and in-depth evaluated (look-ahead) SAT solver heuristics might be of help to find word-level heuristic improvements. The heuristics might be also improved by introducing and combining more statistical features which describe the formula's structure. For a simple example the reduction of a formula is higher if a parameter of a long logical AND chain is set to 0 than the parameter of a short logical AND chain ($a \wedge b \wedge \ldots \wedge 0 \equiv \bot$).

In the future PBoolector should be extended to other theories supported by Boolector, the theory of arrays, lambda expressions and their combination. An evaluation on the whole set of SMT-LIB benchmarks (for QF_BV $\sim$ 33000 files) can be done, to possibly identify further real world application patterns applicable for PBoolector. On the other hand, additionally hard bit-vector formulas can be crafted to further analyze PBoolectors strengths. The used back end SAT solver Lingeling is optimized for hard standalone CNF formulas. It might be possible to find a SAT solver configuration that is more adequate for the PBoolector application.

Appendices

Appendix A

Table Notations

A.1 Comparison Tables

In the comparison tables the improvements of the parallel implementation is shown compared to its reference implementation. The improvement is categorized into 5 different success states. The table contains the columns:

- compare: The left column "ref" contains the name of the reference tool to which the tool in the right column "to" is compared to.
- filter: Contains information about filtering the benchmark set.
- total: Contains the number of benchmarks which are compared.
- success state "++": Contains the number of benchmarks which are not solved by the left (ref) tool and which is solved by the right (to) tool.
- success state +: Contains the number of benchmarks which are solved faster by the right tool.
- success state −: Contains the number of benchmarks which are solved slower by the right tool.
- success −−: Contains the number of benchmarks which are not solved by right tool although they are solver by the left tool.
- o: The number of benchmarks which are not solved by both implementations.

A.2 Detailed Result Tables

The detailed result table shows statistical values to analyze a PBoolector run. Unnecessary information is not shown. For example in a run which do not use Failed Literal Reduction (FLR) the statistical values for FLR are not printed. The table can consist of following columns:

- rank: The results are ordered by success state. Each benchmark gets a unique ranking number for each run.
- benchmark: The name of the benchmark with preceding category. If the name is to long it is abbreviated with "...". Each benchmark should be clearly identifiable by the name.
- stat: The status of the benchmark. Here to common encoding 0 (*unknown*), 10 (*sat*) and 20 (*unsat*) is used.
- sequ-btor: The runtime of the sequential Boolector implementation in seconds.
- ref: The runtime of the reference implementation to which the parallel run is compared (in seconds). This column contains for example the runtime of the reference implementation Base-Incremental (base-inc).
- runtime statistics for the parallel run (e.g. PBoolector).
  - tot(wc): The total wall-clock runtime in seconds.
  - diff: The difference between the parallel and the reference runtime in seconds. If a timeout has occurred the time limit of 1800 seconds is used for the difference.
  - su: The achieved speedup. If a timeout has occurred the time limit of 1800 seconds is used for the speedup.
  - mem: The memory consumption for the parallel run in GB.
  - rnd: The number of rounds in the main algorithm.
  - subproblems: This section contains detailed information about the produced subproblems.
    - "tot": The total number of produced subproblems.
    - "max": The maximal number of open subproblems.
    - "fin": The number of finished subproblems where no look-ahead was found.
  - flr: This section contains detailed information about the Failed Literal Reduction.
    - tot(wc): The wall-clock time spent in FLR (in seconds).
    - fl: The number of found failed literals.
    - res: The result of FLR (0,10 or 20).
  - lkhd: The wall-clock time spent in the look-ahead phase (in seconds).
  - split: The wall-clock time spent in the split phase (in seconds).
  - search: This section contains detailed information about the search phase.
    - tot(wc): The wall-clock time spent in the search phase (in seconds).

    - tot(cpu): The total CPU time spent in the search phase (in seconds).
    - sat(cpu): The total CPU time spend in the SAT solver during the search (in seconds).
    - min: The minimal time of all `boolector_sat` calls in seconds.
    - max: The maximal time of all `boolector_sat` calls in seconds.
    - mean: The mean time of all `boolector_sat` calls in seconds.

- cmp: A column which lists the ranking of another parallel run for comparison (e.g the rank of par-inc-flr).

Appendix B

Figures

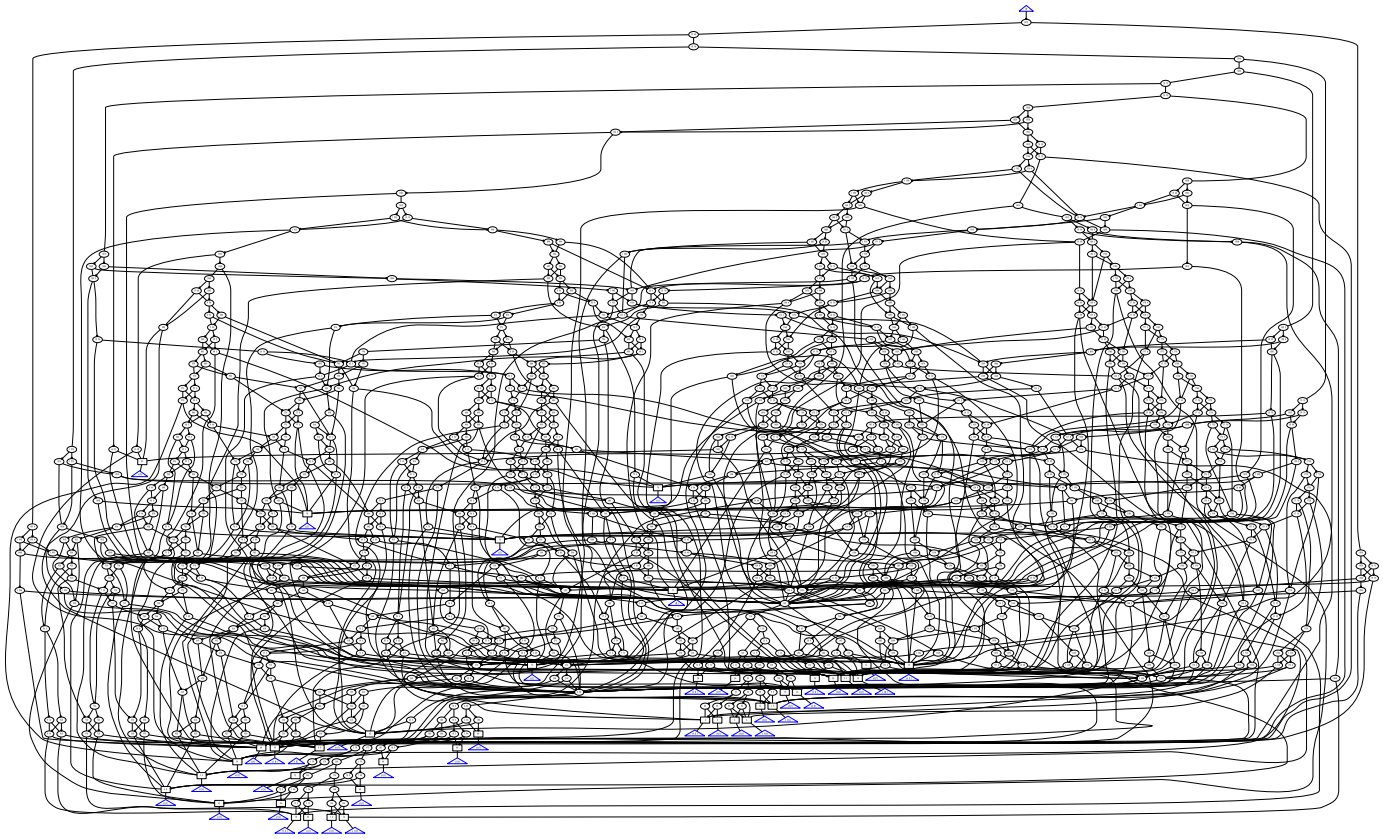


Figure B.1: Formula 1.1 with bit-width 8 encoded as AIG. The triangles are the Boolean input variables. Each node is a AND connective, either negated or not.

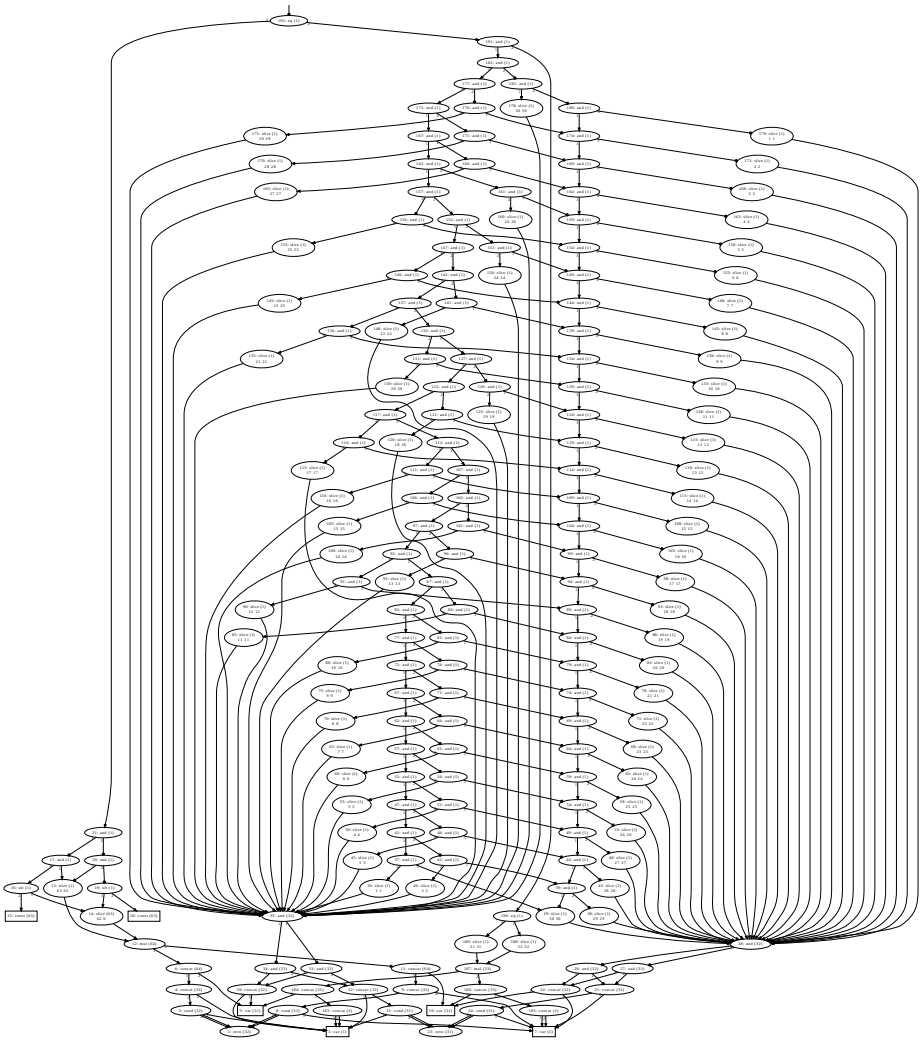


Figure B.2: Benchmark *smulov1bw32*. On the bottom the naive implementation which is checked against the Signed Multiplication Overflow Circuit (SMOC).

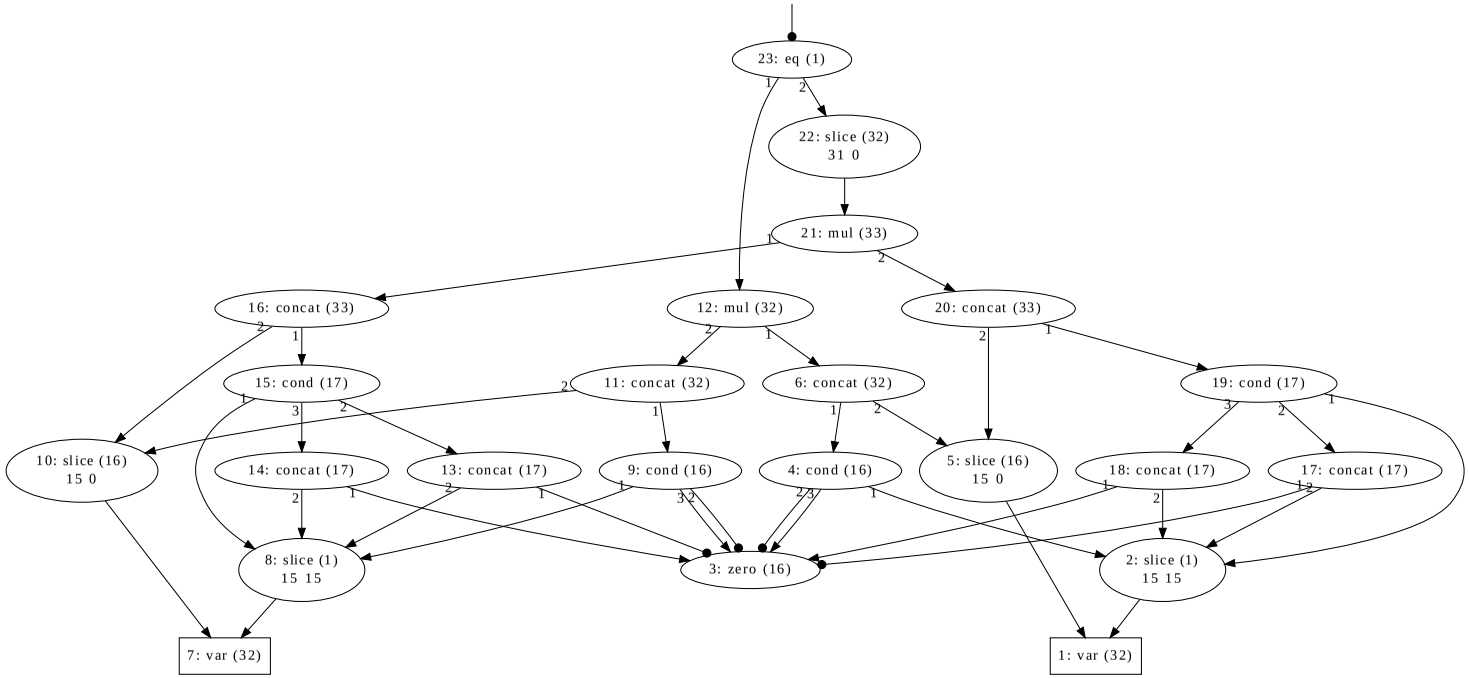


Figure B.3: Benchmark *problem_14* contains 2 hard multiplication operations with sign-extended parameters.

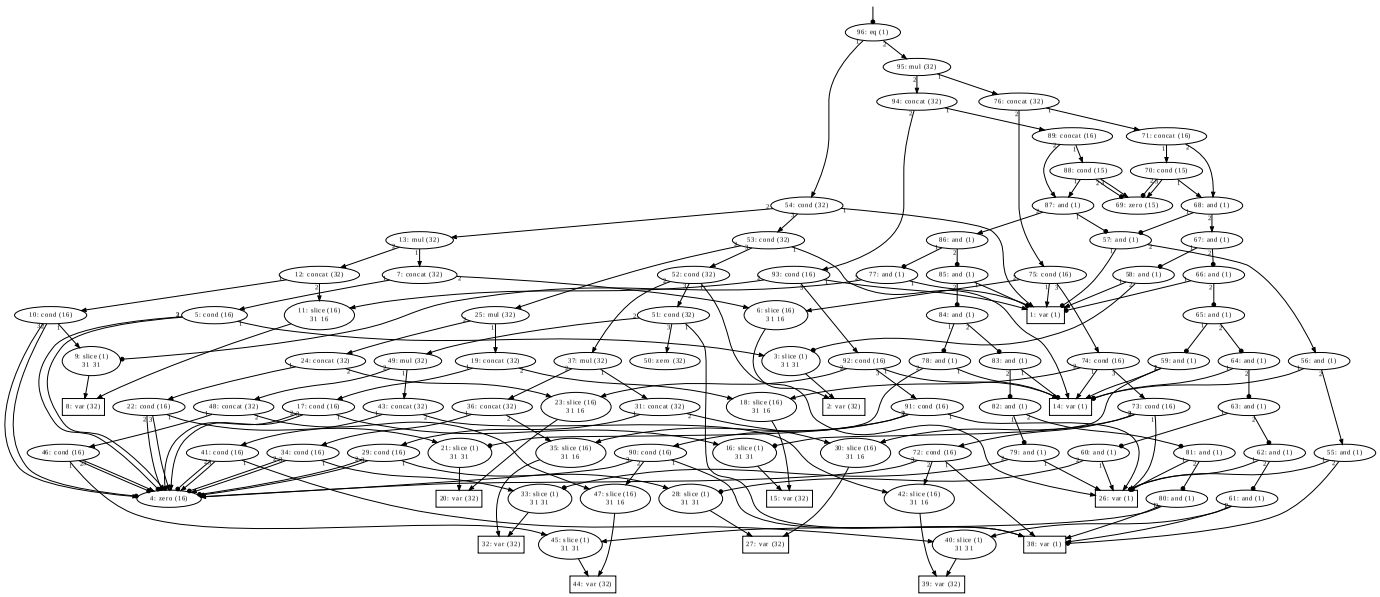


Figure B.4: Benchmark *problem_16* contains several hard multiplication operations with sign-extended parameters.

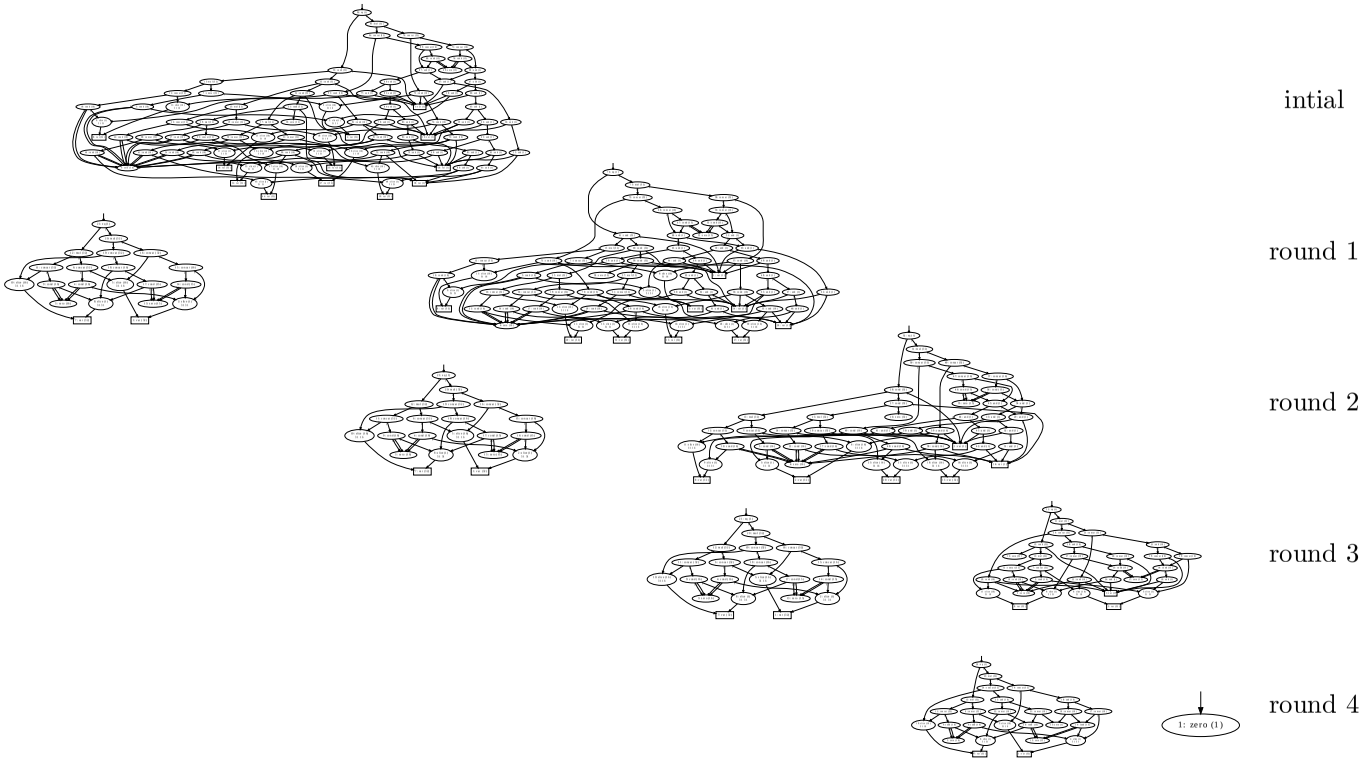


Figure B.5: Visualized execution tree of PBoolector for *problem_16*.

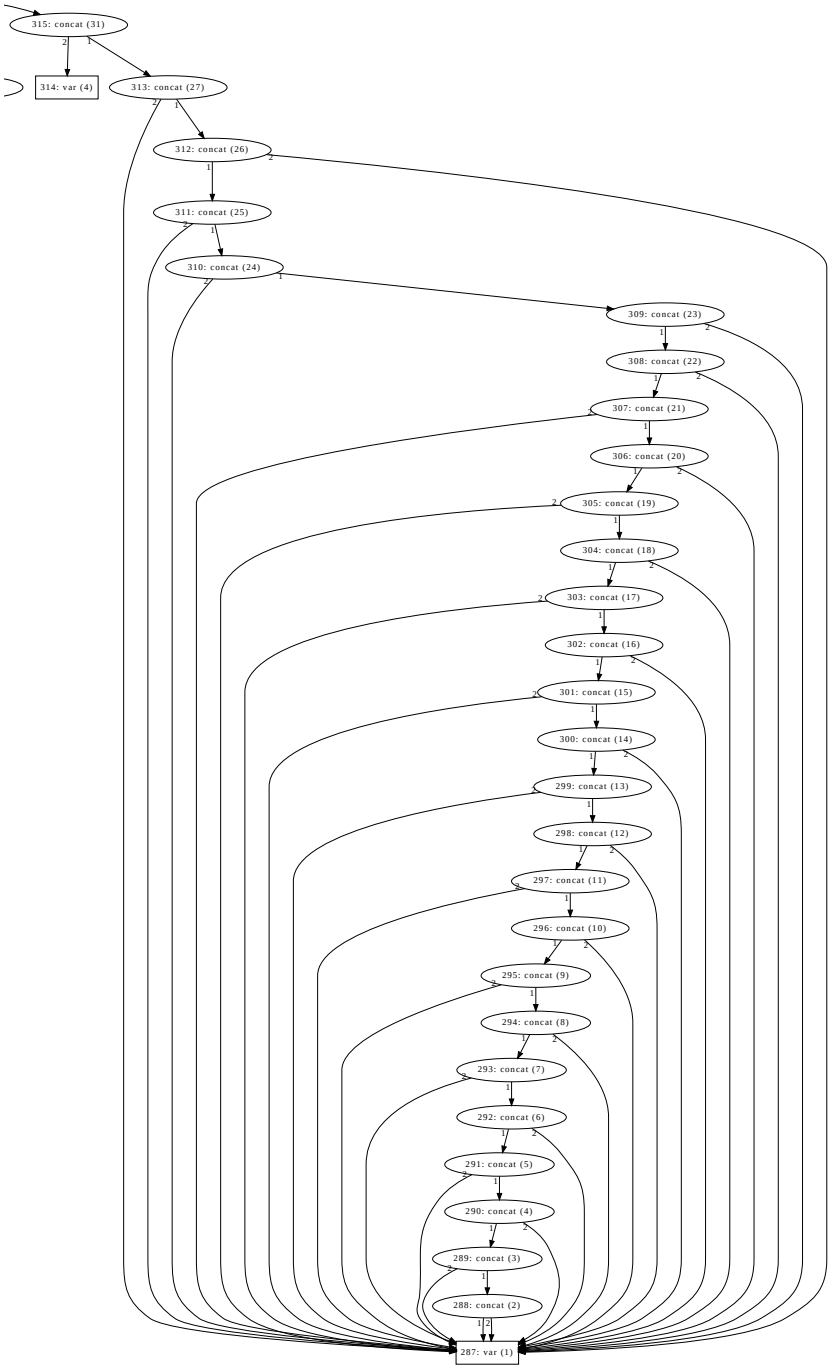


Figure B.6: Partial DAG of benchmark *problem_19*. Node 315 serves as input parameter for a 31 bit multiplication.

Appendix C

Detailed Results

Table C.1: Detailed results of PBoolector (Parallel-Incremental) compared to the reference times Base-Incremental. Last column par-inc-flr contains a cross reference to detailed results of Parallel-Incremental-FLR in Tab.C.2 (Detailed table notations are in Sect. A.2).

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental)																	cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search							par-inc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
1.1	brummayerbiere2_smulov1bw32	20	1056.25	oot	480.29	1319.71	3.75	0.5	54	270	49	41	0.03	2.64	476.48	1321.36	1317.79	0.00	27.57	1.83	1.1
1.2	log-slicing_bvurem_16	20	oot	oot	1002.92	797.08	1.79	0.2	8	24	23	22	0.01	0.34	1001.69	2526.84	2526.63	0.00	340.77	30.44	1.2
1.3	...iffyInterleavedModMult-8	20	oot	oot	1379.26	420.74	1.31	0.4	63	442	142	0	0.07	1.34	1376.50	5110.83	5109.38	0.00	20.32	4.18	1.3
1.4	core_ext_con_056_002_1024	20	279.98	oot	1464.58	335.42	1.23	0.9	1	1	1	1	0.00	0.00	1463.74	1463.74	1463.34	155.61	1308.13	731.87	5.6
1.5	core_ext_con_060_002_1024	20	220.71	oot	1470.71	329.29	1.22	0.6	1	1	1	1	0.00	0.00	1469.97	1469.97	1469.56	166.78	1303.19	734.99	5.8
1.6	core_ext_con_052_002_1024	20	194.60	oot	1495.92	304.08	1.20	0.7	1	1	1	1	0.00	0.00	1495.12	1495.12	1494.75	137.65	1357.48	747.56	5.5
1.7	log-slicing_bvudiv_18	20	oot	oot	1554.09	245.91	1.16	0.8	11	40	31	23	0.03	7.51	1541.40	4286.69	4285.46	0.00	532.36	34.85	1.4
1.8	core_ext_con_064_002_1024	20	344.29	oot	1769.72	30.28	1.02	0.9	1	1	1	1	0.00	0.00	1768.99	1768.99	1768.55	165.43	1603.56	884.50	5.11
1.9	log-slicing_bvsmod_15	20	oot	oot	1779.82	20.18	1.01	0.4	41	184	51	118	0.13	2.87	1775.15	4042.04	4040.50	0.00	138.07	6.07	1.5
2.1	log-slicing_bvudiv_17	20	1270.36	1373.34	621.52	751.82	2.21	0.7	10	33	23	21	0.02	5.94	611.91	1571.46	1570.31	0.00	187.39	15.71	2.1
2.2	log-slicing_bvurem_15	20	oot	1147.71	415.19	732.52	2.76	0.2	8	24	23	20	0.01	0.38	413.94	977.96	977.77	0.00	141.44	12.22	2.2
2.3	log-slicing_bvsmod_14	20	1414.54	1350.59	727.74	622.85	1.86	0.3	39	171	46	107	0.12	2.51	723.54	1912.06	1910.57	0.00	70.94	3.18	2.3
2.4	log-slicing_bvmul_12	20	129.19	962.31	442.54	519.77	2.17	0.1	2	2	2	2	0.00	0.02	441.84	836.40	836.38	0.28	441.05	167.28	2.4
2.5	core_ext_con_048_002_1024	20	487.72	1716.64	1291.35	425.29	1.33	0.7	1	1	1	1	0.00	0.00	1290.61	1290.61	1290.29	124.30	1166.31	645.31	4.6
2.6	core_ext_con_044_002_1024	20	195.82	1571.35	1263.08	308.27	1.24	0.7	1	1	1	1	0.00	0.00	1262.37	1262.37	1262.08	107.34	1155.04	631.19	3.17
2.7	..._test_v7_r7_vr5_c1_s3582	10	326.78	392.56	115.42	277.14	3.40	4.1	21	37	16	0	1.20	47.95	57.14	116.45	95.41	0.18	3.23	1.31	4.56
2.8	log-slicing_bvudiv_16	20	390.99	521.12	253.30	267.82	2.06	0.2	8	24	22	20	0.01	0.51	251.91	738.11	737.87	0.00	86.51	9.23	2.5
2.9	core_ext_con_036_002_1024	20	269.99	1156.19	896.79	259.40	1.29	0.5	1	1	1	1	0.00	0.00	896.10	896.10	895.86	64.61	831.49	448.05	3.18
2.10	..._smtcomp09_lf_bv_formula	20	180.24	207.71	68.58	139.13	3.03	0.9	10	56	32	0	0.07	3.18	64.20	242.25	234.68	0.02	6.61	1.91	4.81
2.11	log-slicing_bvurem_14	20	757.16	270.88	141.23	129.65	1.92	0.1	8	24	22	19	0.01	0.31	140.09	421.75	421.58	0.00	55.84	5.41	2.6
2.12	..._smtcomp09_nt_bv_formula	20	156.72	175.28	82.16	93.12	2.13	1.8	14	80	56	0	0.14	7.92	71.64	271.82	258.47	0.05	20.64	1.31	3.40
2.13	log-slicing_bvsmod_13	20	360.26	438.26	350.88	87.38	1.25	0.3	36	156	50	100	0.11	2.19	346.46	891.52	890.27	0.00	29.15	1.58	2.7
2.14	challenge_multiplyOverflow	20	1035.61	1002.11	917.70	84.41	1.09	0.1	2	2	2	2	0.00	0.02	917.00	1499.06	1499.03	0.56	915.22	299.81	2.8
2.15	log-slicing_bvudiv_15	20	162.70	192.95	116.97	75.98	1.65	0.2	8	23	21	18	0.01	0.49	115.67	339.04	338.83	0.00	40.52	4.52	2.9
2.16	brummayerbiere_nlzbe256	20	117.51	157.97	97.09	60.88	1.63	0.7	3	5	4	0	0.01	4.17	92.02	126.43	122.47	0.15	33.21	14.05	3.20
2.17	log-slicing_bvurem_13	20	147.97	116.67	57.32	59.35	2.04	0.1	7	21	17	17	0.01	0.26	56.29	180.40	180.23	0.00	22.86	2.69	2.10
2.18	brummayerbiere2_smulov1bw24	20	134.33	132.01	73.32	58.69	1.80	0.3	25	132	42	15	0.01	0.64	71.78	220.19	219.12	0.00	15.60	0.67	2.11
2.19	log-slicing_bvsmod_12	20	138.58	174.80	166.82	7.98	1.05	0.3	31	150	52	79	0.08	1.99	163.37	456.70	455.62	0.00	13.65	0.92	2.12
2.20	...yerbiere3_isqrtinvalidvc	10	181.66	15.89	9.50	6.39	1.67	0.1	4	16	16	0	0.00	0.07	8.70	29.17	28.25	0.23	2.28	0.94	2.13
2.21	bmc-bv_magic	10	443.86	453.44	452.99	0.45	1.00	0.1	0	1	1	0	0.00	0.00	0.46	0.46	0.39	0.46	0.46	0.46	3.2
2.22	...ayerbiere2_smulov4bw1024	10	1518.87	10.91	10.63	0.28	1.03	0.5	0	1	1	0	0.00	0.00	9.37	9.37	0.71	9.37	9.37	9.37	3.24
2.23	...mayerbiere2_smulov2bw384	20	122.06	15.64	15.37	0.27	1.02	0.4	0	1	1	0	0.00	0.00	14.33	14.33	8.79	14.33	14.33	14.33	3.9
2.24	...mayerbiere2_smulov2bw448	20	150.63	43.69	43.50	0.19	1.00	0.5	0	1	1	0	0.00	0.00	42.32	42.32	34.56	42.32	42.32	42.32	3.11
2.25	...mayerbiere2_smulov2bw512	20	198.38	36.68	36.53	0.15	1.00	0.6	0	1	1	0	0.00	0.00	35.11	35.11	22.76	35.11	35.11	35.11	3.13
2.26	...ayerbiere2_smulov4bw0512	10	128.78	2.57	2.42	0.15	1.06	0.1	0	1	1	0	0.00	0.00	1.58	1.58	0.17	1.58	1.58	1.58	3.10
2.27	...ayerbiere2_smulov4bw0640	10	141.93	3.96	3.81	0.15	1.04	0.2	0	1	1	0	0.00	0.00	2.89	2.89	0.27	2.89	2.89	2.89	3.14
2.28	...ayerbiere2_smulov4bw0768	10	563.42	5.80	5.66	0.14	1.02	0.4	0	1	1	0	0.00	0.00	4.69	4.69	0.40	4.69	4.69	4.69	3.15
2.29	...ayerbiere2_smulov4bw0896	10	1274.52	8.26	8.16	0.10	1.01	0.5	0	1	1	0	0.00	0.00	6.96	6.96	0.54	6.96	6.96	6.96	3.21
2.30	brummayerbiere_bitrev8192	20	101.97	101.49	101.42	0.07	1.00	0.0	0	1	1	0	0.00	0.00	0.04	0.04	0.00	0.04	0.04	0.04	3.1
3.1	bmc-bv_graycode	10	442.54	449.03	449.15	-0.12	1.00	0.1	0	1	1	0	0.00	0.00	0.04	0.04	0.01	0.04	0.04	0.04	3.3
3.2	...erbiere4_unconstrained02	10	oot	44.23	44.49	-0.26	0.99	4.2	0	1	1	0	0.00	0.00	39.85	39.85	7.77	39.85	39.85	39.85	3.4
3.3	RWS_Example_18.txt	10	160.61	193.35	193.80	-0.45	1.00	0.1	0	1	1	0	0.00	0.00	192.97	192.97	192.81	192.97	192.97	192.97	3.5
3.4	...mayerbiere2_umulov1bw224	20	108.42	90.16	94.61	-4.45	0.95	0.3	1	2	2	0	0.00	2.73	90.91	90.91	89.42	0.00	88.35	30.30	3.6

Continued on next page

Table C.1 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental)																cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-inc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
3.5	...mayerbiere2_umulov1bw256	20	143.94	119.05	134.07	-15.02	0.89	0.4	1	2	2	0	0.00	3.13	129.94	129.95	127.88	0.00	101.94	43.32	3.8
3.6	log-slicing_bvsub_04000	20	545.84	842.77	878.90	-36.13	0.96	0.1	1	1	1	1	0.00	0.00	877.75	877.75	876.83	5.84	871.91	438.88	3.7
3.7	...test_v5_r15_vr1_c1_s8236	20	927.66	285.17	327.32	-42.15	0.87	3.1	34	57	8	0	2.54	96.28	206.86	373.94	329.06	0.25	13.36	3.31	4.70
3.8	...st_v5_r15_vr10_c1_s11127	10	346.03	1009.99	1054.63	-44.64	0.96	3.7	82	95	9	0	5.47	201.04	814.59	1495.21	1434.21	0.02	45.29	6.42	4.19
3.9	...test_v5_r5_vr5_c1_s24018	10	182.81	186.53	321.84	-135.31	0.58	3.4	69	159	29	0	1.67	76.38	226.11	576.00	537.13	0.00	7.16	1.73	4.86
3.10	...est_v7_r7_vr10_c1_s24535	10	237.88	246.30	397.00	-150.70	0.62	3.6	62	116	14	0	3.10	127.33	241.38	445.71	392.64	0.00	6.89	1.90	4.76
3.11	float_mult2.c.50	10	109.39	40.02	198.83	-158.81	0.20	4.5	11	30	12	0	0.41	25.36	167.82	411.60	373.78	0.00	34.21	4.24	4.106
3.12	...biere3_isqrtaddinvalidvc	10	185.60	16.00	190.34	-174.34	0.08	1.9	50	302	219	82	0.05	1.76	187.58	674.71	660.54	0.00	5.82	0.47	3.12
3.13	...smtcomp09_std_bv_formula	20	108.86	169.61	345.90	-176.29	0.49	3.1	30	169	96	0	0.20	9.57	333.92	1296.64	1270.40	0.01	26.02	2.16	3.34
3.14	log-slicing_bvsrem_15	20	791.10	927.62	1138.09	-210.47	0.82	0.3	24	92	46	76	0.06	1.44	1135.44	2967.75	2966.89	0.00	142.67	8.29	3.19
3.15	log-slicing_bvsrem_14	20	256.90	269.58	490.02	-220.44	0.55	0.3	22	92	46	68	0.06	1.34	487.48	1351.29	1350.53	0.00	48.98	4.03	3.16
3.16	...mtcomp09_full_bv_formula	20	172.46	167.31	411.63	-244.32	0.41	3.5	30	186	93	0	0.17	9.60	399.74	1494.83	1473.45	0.00	27.23	2.66	3.35
3.17	pspace_shift1add.22961	20	110.71	110.86	365.95	-255.09	0.30	0.3	1	1	1	1	0.00	0.00	365.16	365.16	364.67	12.64	352.52	182.58	3.22
3.18	pspace_shift1add.23958	20	114.08	115.50	402.43	-286.93	0.29	0.4	1	1	1	1	0.00	0.00	401.58	401.58	401.06	13.85	387.73	200.79	3.23
3.19	pspace_shift1add.24955	20	128.47	121.79	436.59	-314.80	0.28	0.4	1	1	1	1	0.00	0.00	435.77	435.77	435.23	14.84	420.94	217.89	3.25
3.20	pspace_shift1add.25952	20	143.28	129.38	467.51	-338.13	0.28	0.4	1	1	1	1	0.00	0.00	466.75	466.75	466.19	15.50	451.26	233.38	3.26
3.21	pspace_shift1add.26949	20	152.33	135.58	495.29	-359.71	0.27	0.4	1	1	1	1	0.00	0.00	494.54	494.54	493.96	16.37	478.16	247.27	3.27
3.22	brummayerbiere3_isqrt	20	145.81	148.11	527.67	-379.56	0.28	1.4	52	324	164	62	0.05	1.84	524.85	1809.43	1792.98	0.03	35.06	1.49	3.33
3.23	pspace_shift1add.27946	20	167.81	143.34	530.40	-387.06	0.27	0.6	1	1	1	1	0.00	0.00	529.63	529.63	529.04	17.59	512.05	264.82	3.28
3.24	...test_v5_r5_vr1_c1_s14623	10	143.20	147.57	546.76	-399.19	0.27	3.2	55	163	24	0	1.64	70.99	458.89	1350.71	1308.19	0.00	84.94	3.95	4.92
3.25	pspace_shift1add.28943	20	186.84	148.33	563.42	-415.09	0.26	0.6	1	1	1	1	0.00	0.00	562.38	562.38	561.77	18.03	544.35	281.19	3.29
3.26	pspace_shift1add.29940	20	195.38	157.23	595.75	-438.52	0.26	0.6	1	1	1	1	0.00	0.00	595.01	595.01	594.38	19.18	575.83	297.50	3.31
3.27	tacas07_Y86_std	20	129.65	193.97	659.19	-465.22	0.29	3.7	22	108	33	0	0.27	30.60	625.81	2233.46	2187.15	0.01	66.14	7.20	4.84
3.28	brummayerbiere3_isqrtadd	20	144.73	148.88	639.46	-490.58	0.23	1.6	41	289	186	11	0.04	1.64	636.92	2364.62	2349.85	0.01	37.24	2.28	3.32
3.29	log-slicing_bvsdiv_16	20	182.35	223.39	735.07	-511.68	0.30	0.4	23	94	51	73	0.07	1.57	731.06	1932.90	1932.02	0.00	66.81	5.34	3.30
3.30	...test_v7_r12_vr5_c1_s8938	10	615.16	403.94	1013.23	-609.29	0.40	3.7	53	94	10	0	5.39	191.18	778.23	1125.50	1056.25	0.26	110.81	5.71	4.53
3.31	...test_v5_r5_vr10_c1_s7194	10	141.84	299.26	982.39	-683.13	0.30	4.1	70	164	30	0	1.86	81.99	881.94	3005.08	2960.48	0.00	97.68	7.09	4.68
3.32	lfsr_lfsr_004_111_112	20	178.05	181.61	873.19	-691.58	0.21	3.6	18	97	52	0	0.30	31.53	834.76	3049.48	3031.64	0.06	114.57	10.55	3.37
3.33	brummayerbiere3_maxand256	20	102.05	252.03	1015.47	-763.44	0.25	3.8	22	79	31	0	0.35	24.66	986.62	2318.26	2263.92	0.12	199.15	8.95	4.13
3.34	...est_v5_r15_vr1_c1_s32559	20	163.75	217.60	1055.87	-838.27	0.21	4.9	47	80	10	0	3.80	134.12	890.50	1904.85	1830.55	0.24	150.07	7.59	4.79
3.35	lfsr_lfsr_008_031_064	20	oot	42.97	915.97	-873.00	0.05	3.4	36	202	133	0	0.70	19.82	890.36	3443.02	3413.94	0.07	33.41	4.39	3.42
3.36	log-slicing_bvsdiv_17	20	502.64	442.92	1495.58	-1052.66	0.30	1.2	24	92	53	80	0.08	11.44	1481.20	4162.03	4159.87	0.01	128.26	11.13	3.36
3.37	float_newton.5.2.i	20	249.53	196.37	1671.24	-1474.87	0.12	4.1	34	136	27	0	1.35	54.67	1604.12	3147.73	3088.53	0.01	208.46	8.11	4.83
3.38	lfsr_lfsr_008_063_096	20	oot	144.83	1780.43	-1635.60	0.08	3.6	34	145	82	0	0.96	58.22	1709.73	6504.99	6471.56	0.13	83.23	10.73	4.94
4.1	pspace_power2sum.6097	20	388.12	1779.43	oot	-20.57	0.99	2.2	1	1	1	0	0.00	0.00	1621.05	1621.05	1620.63	1621.05	1621.05	1621.05	4.1
4.2	log-slicing_bvslt_15402	20	1048.34	1765.72	oot	-34.28	0.98	2.8	8	36	36	0	0.01	22.83	1353.66	6172.85	6168.74	9.55	258.46	69.36	4.2
4.3	...est_v7_r12_vr1_c1_s10576	20	oot	1756.98	oot	-43.02	0.98	4.8								0.00	0.00	0.00	0.00	0.00	4.3
4.4	pspace_power2sum.5699	20	347.92	1748.12	oot	-51.88	0.97	2.0	1	1	1	0	0.00	0.00	1333.34	1333.34	1332.96	1333.34	1333.34	1333.34	4.4
4.5	log-slicing_bvsub_05994	20	414.31	1726.25	oot	-73.75	0.96	0.1	1	1	1	0	0.00	0.00	6.59	6.59	4.60	6.59	6.59	6.59	4.5
4.6	...test_v5_r15_vr5_c1_s8246	20	1106.64	1711.46	oot	-88.54	0.95	4.5	39	83	10	0	3.41	132.63	143.09	325.00	262.57	0.23	4.32	1.90	4.7
4.7	log-slicing_bvult_14973	20	1019.50	1697.30	oot	-102.70	0.94	3.1	10	38	38	0	0.02	25.44	1649.86	5963.84	5959.40	2.08	261.87	53.73	4.8
4.8	log-slicing_bvsdiv_18	20	1485.52	1599.03	oot	-200.97	0.89	1.2	16	76	52	35	0.05	9.19	1710.01	4589.47	4587.79	0.00	300.84	17.86	4.9
4.9	...test_v7_r7_vr5_c1_s19694	10	1370.11	1539.94	oot	-260.06	0.86	4.1								0.00	0.00	0.00	0.00	0.00	4.10
4.10	...est_v7_r7_vr10_c1_s10625	10	852.76	1379.94	oot	-420.06	0.77	3.9								0.00	0.00	0.00	0.00	0.00	4.12
4.11	log-slicing_bvslt_12430	20	oot	1349.76	oot	-450.24	0.75	2.9	7	39	39	0	0.01	21.75	1548.17	6500.98	6497.32	13.93	235.22	92.87	4.14
4.12	...est_v5_r15_vr5_c1_s26657	20	oot	1282.84	oot	-517.16	0.71	4.5	91	129	10	0	7.00	250.49	725.02	1473.29	1382.13	0.01	14.51	4.59	4.15
4.13	log-slicing_bvslt_10944	20	505.68	1114.25	oot	-685.75	0.62	2.9	8	41	41	0	0.02	21.63	1538.51	6500.08	6496.37	0.74	168.75	73.86	4.16
4.14	...._test_v7_r7_vr1_c1_s4574	10	609.33	1077.24	oot	-722.76	0.60	4.1	57	104	17	0	2.78	115.12	419.96	980.12	919.67	0.16	88.44	2.88	4.17
4.15	...st_v5_r10_vr10_c1_s21502	20	697.82	1056.47	oot	-743.53	0.59	4.2	82	151	16	0	3.97	180.97	873.48	2399.94	2326.46	0.00	139.62	6.00	4.18
4.16	VS3_VS3-benchmark-A9	10	268.45	1009.13	oot	-790.87	0.56	3.3	45	272	171	0	0.43	21.62	1774.31	6839.28	6832.50	0.00	32.68	6.10	4.20

Continued on next page

Table C.1 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental)																	cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search							par-inc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
4.17	...est_v5_r10_vr1_c1_s32538	20	208.16	996.75	oot	-803.25	0.55	3.9	56	126	16	0	3.21	128.83	1471.13	3913.79	3842.61	0.00	114.41	12.12	4.21
4.18	...est_v5_r10_vr5_c1_s13679	10	899.64	937.68	oot	-862.32	0.52	4.4	88	214	15	0	5.59	218.87	956.26	3164.78	3053.28	0.00	19.43	6.13	4.22
4.19	VS3_VS3-benchmark-A7	10	985.79	934.59	oot	-865.41	0.52	3.3	62	390	148	0	0.67	28.97	1730.18	6717.92	6709.49	0.00	22.81	4.49	4.23
4.20	log-slicing_bvslt_7972	20	305.25	931.67	oot	-868.33	0.52	3.1	12	67	67	0	0.02	18.70	1729.79	6696.44	6692.11	3.04	154.25	34.34	4.24
4.21	...servers_slapd_a_vc149923	10	325.64	886.77	oot	-913.23	0.49	3.3	30	67	67	0	0.06	14.26	1775.11	6952.02	6949.13	0.28	50.87	13.19	4.25
4.22	...est_v7_r7_vr10_c1_s32506	10	1754.95	864.50	oot	-935.50	0.48	4.0								0.00	0.00	0.00	0.00	0.00	4.26
4.23	log-slicing_bvslt_9458	20	539.16	833.97	oot	-966.03	0.46	2.9	8	44	44	0	0.02	22.68	1461.75	6272.56	6268.71	15.05	137.59	67.45	4.27
4.24	...test_v7_r7_vr1_c1_s24449	20	679.70	826.28	oot	-973.72	0.46	4.1								0.00	0.00	0.00	0.00	0.00	4.28
4.25	...servers_slapd_a_vc149922	10	325.68	788.47	oot	-1011.53	0.44	3.3	29	67	67	0	0.06	14.47	1711.19	6905.62	6902.76	0.30	50.82	13.20	4.29
4.26	...est_v5_r10_vr1_c1_s19145	10	316.49	780.82	oot	-1019.18	0.43	3.8	67	155	16	0	3.76	161.73	1104.32	3075.57	3002.18	0.01	125.89	8.38	4.30
4.27	...yerbiere_countbitssrl032	20	387.06	713.56	oot	-1086.44	0.40	0.5	27	148	18	23	0.05	2.05	1676.36	3311.13	3299.90	0.00	801.54	9.85	4.31
4.28	log-slicing_bvult_8991	20	542.61	709.90	oot	-1090.10	0.39	2.9	8	46	44	0	0.02	23.27	1714.57	6010.11	6006.37	0.19	171.04	73.29	4.32
4.29	log-slicing_bvult_7994	20	230.10	691.43	oot	-1108.57	0.38	3.2	10	64	64	0	0.02	18.20	1383.43	6682.00	6678.09	2.43	154.87	42.83	4.33
4.30	log-slicing_bvslt_6486	20	191.65	687.42	oot	-1112.58	0.38	3.1	16	95	69	0	0.03	23.58	1682.18	6433.35	6427.70	0.04	130.02	25.03	4.34
4.31	log-slicing_bvult_6997	20	201.19	669.29	oot	-1130.71	0.37	3.7	10	53	53	0	0.02	14.43	1046.40	6668.19	6664.58	8.49	108.66	41.68	4.35
4.32	...est_v5_r10_vr5_c1_s13195	10	337.55	662.60	oot	-1137.40	0.37	4.0								0.00	0.00	0.00	0.00	0.00	4.36
4.33	...1_alt_true.BV.c.cil.c.17	20	oot	621.75	oot	-1178.25	0.35	3.9	22	53	26	0	0.17	10.65	1646.41	6136.21	6104.07	0.02	123.45	19.79	4.37
4.34	log-slicing_bvult_6000	20	406.38	608.33	oot	-1191.67	0.34	3.0	16	98	71	0	0.04	22.92	1763.62	6632.01	6627.43	0.40	123.59	26.01	4.38
4.35	log-slicing_bvslt_8715	20	342.56	598.50	oot	-1201.50	0.33	2.8	8	44	40	0	0.02	20.16	1777.50	6604.26	6600.77	14.68	156.89	83.60	4.39
4.36	...test_v7_r7_vr1_c1_s22845	10	316.63	593.48	oot	-1206.52	0.33	3.8	57	195	18	0	4.38	183.36	266.70	584.23	503.33	0.02	47.35	1.34	4.40
4.37	lfsr_lfsr_008_127_112	20	176.00	591.63	oot	-1208.37	0.33	3.6	18	37	32	0	0.33	38.01	1656.89	6131.38	6109.53	2.03	286.52	25.13	4.41
4.38	log-slicing_bvslt_7229	20	217.01	590.38	oot	-1209.62	0.33	3.0	16	94	70	0	0.03	24.94	1564.80	6495.06	6489.31	0.00	149.97	24.98	4.42
4.39	...est_v5_r15_vr1_c1_s26845	20	619.09	589.84	oot	-1210.16	0.33	4.6	39	95	10	0	3.54	132.61	205.52	590.14	514.10	0.27	5.63	2.60	4.43
4.40	lfsr_lfsr_008_143_112	20	205.39	587.59	oot	-1212.41	0.33	4.1	19	28	28	0	0.45	33.59	1628.64	6390.59	6345.68	1.95	95.59	22.82	4.44
4.41	...est_v5_r10_vr1_c1_s13516	10	231.36	574.53	oot	-1225.47	0.32	3.7								0.00	0.00	0.00	0.00	0.00	4.45
4.42	...tcomp09_stall_bv_formula	20	609.53	493.17	oot	-1306.83	0.27	4.3	12	37	31	0	0.35	17.19	1405.72	6359.95	6321.95	1.03	250.63	39.75	4.46
4.43	float_mul_03_30_4	20	514.80	488.97	oot	-1311.03	0.27	3.3								0.00	0.00	0.00	0.00	0.00	4.47
4.44	float_newton.5.3.i	20	415.28	486.80	oot	-1313.20	0.27	4.1	15	59	23	0	0.93	36.34	260.58	784.74	744.55	0.01	46.47	4.34	4.48
4.45	...test_v5_r10_vr5_c1_s8690	20	353.93	485.91	oot	-1314.09	0.27	4.4	58	133	17	0	3.46	137.34	587.28	1576.56	1495.35	0.02	71.27	3.86	4.49
4.46	float_newton.2.3.i	20	289.38	445.92	oot	-1354.08	0.25	4.2	19	89	23	0	1.38	51.91	230.24	627.64	580.54	0.00	30.08	2.56	4.50
4.47	float_newton.6.3.i	10	564.04	437.06	oot	-1362.94	0.24	4.3	20	106	19	0	1.66	61.54	586.29	1751.06	1694.59	0.00	66.46	6.73	4.51
4.48	lfsr_lfsr_008_143_128	20	261.79	428.53	oot	-1371.47	0.24	4.0	16	31	31	0	0.63	52.26	1554.04	6170.10	6122.53	3.28	251.54	27.55	4.52
4.49	VS3_VS3-benchmark-A6	10	111.93	397.09	oot	-1402.91	0.22	2.1	82	612	131	0	0.99	40.47	1723.27	6617.33	6606.84	0.00	18.94	3.60	4.54
4.50	...test_v7_r7_vr5_c1_s14675	10	364.55	393.97	oot	-1406.03	0.22	4.1								0.00	0.00	0.00	0.00	0.00	4.55
4.51	lfsr_lfsr_008_015_128	20	277.36	380.87	oot	-1419.13	0.21	4.1	59	184	184	0	1.76	21.95	1766.84	6900.93	6808.76	0.14	44.03	4.74	4.57
4.52	log-slicing_bvslt_5743	20	444.58	367.82	oot	-1432.18	0.20	3.0	20	125	73	0	0.04	28.77	911.83	6155.25	6147.77	0.02	293.38	17.95	3.39
4.53	lfsr_lfsr_008_095_128	20	148.89	366.34	oot	-1433.66	0.20	3.6	20	49	37	0	0.51	42.33	1418.36	6305.36	6277.92	1.34	284.13	22.12	4.58
4.54	float_newton.1.3.i	20	385.34	360.03	oot	-1439.97	0.20	4.3	24	124	22	0	1.97	71.92	806.02	2455.95	2387.55	0.00	75.71	7.53	4.59
4.55	brummayerbiere3_minand256	20	289.66	328.30	oot	-1471.70	0.18	1.1	5	6	3	3	0.03	4.62	1661.90	1865.83	1860.89	0.76	833.28	133.27	4.61
4.56	lfsr_lfsr_008_111_112	20	148.60	324.78	oot	-1475.22	0.18	3.6	18	44	37	0	0.41	39.29	1568.73	6017.52	5994.75	3.09	190.65	24.36	4.62
4.57	float_newton.4.3.i	20	334.84	321.37	oot	-1478.63	0.18	3.9	17	88	22	0	1.32	50.45	328.24	877.67	832.53	0.00	48.15	3.87	4.63
4.58	brummayerbiere3_minor256	20	116.13	321.33	oot	-1478.67	0.18	5.2	14	30	18	0	0.15	14.97	743.15	5040.39	4982.56	0.12	473.15	27.10	4.11
4.59	log-slicing_bvslt_5000	20	296.40	315.45	oot	-1484.55	0.18	2.5	20	140	59	0	0.05	28.91	1452.96	6429.94	6422.77	0.02	297.74	20.09	3.38
4.60	float_newton.3.3.i	20	355.49	314.25	oot	-1485.75	0.17	4.5								0.00	0.00	0.00	0.00	0.00	4.64
4.61	lfsr_lfsr_008_159_080	20	194.33	310.35	oot	-1489.65	0.17	3.5	17	50	36	0	0.30	47.04	1657.19	6106.33	6088.18	1.06	116.93	26.55	4.65
4.62	...yerbiere3_minandmaxor064	20	207.79	308.52	oot	-1491.48	0.17	4.0	62	111	14	0	0.24	31.10	1763.82	2525.18	2479.24	0.11	111.33	9.02	4.66
4.63	lfsr_lfsr_008_159_112	20	275.94	300.03	oot	-1499.97	0.17	3.5	9	17	17	0	0.14	25.67	1159.81	6265.18	6252.06	7.67	223.58	56.96	4.67
4.64	lfsr_lfsr_004_159_128	20	112.10	299.74	oot	-1500.26	0.17	3.8	15	54	33	0	0.22	41.74	1451.32	5980.82	5961.99	0.11	297.75	27.82	3.41
4.65	lfsr_lfsr_008_143_096	20	173.94	291.37	oot	-1508.63	0.16	3.5	23	47	39	0	0.58	36.56	1755.22	6673.03	6631.54	1.44	247.68	19.23	4.69
4.66	lfsr_lfsr_008_015_112	20	388.44	270.57	oot	-1529.43	0.15	3.9	60	190	190	0	1.50	18.64	1560.66	6969.76	6885.21	0.13	35.67	4.44	4.71
4.67	..._test_v5_r5_vr5_c1_s2800	10	228.14	268.60	oot	-1531.40	0.15	4.2	69	191	31	0	2.16	97.06	393.03	1288.94	1233.01	0.00	23.35	2.26	4.72

Continued on next page

Table C.1 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental)																cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-inc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
4.68	lfsr_lfsr_004_143_128	20	116.21	259.85	oot	-1540.15	0.14	3.9	24	75	47	0	0.69	57.80	1686.32	6477.89	6438.60	0.36	174.06	16.96	4.73
4.69	lfsr_lfsr_008_079_128	20	107.76	256.88	oot	-1543.12	0.14	3.6	23	79	48	0	0.75	60.69	1705.52	6600.43	6568.82	0.24	136.89	18.33	4.74
4.70	lfsr_lfsr_008_095_096	20	105.52	255.91	oot	-1544.09	0.14	3.7	20	67	45	0	0.47	44.12	1124.29	6661.50	6637.67	0.09	280.31	21.56	4.75
4.71	lfsr_lfsr_008_111_096	20	133.32	238.72	oot	-1561.28	0.13	3.6	19	59	40	0	0.43	51.81	1371.25	6375.97	6352.72	1.47	192.70	23.19	4.77
4.72	brummayerbiere3_maxor064	20	209.14	224.27	oot	-1575.73	0.12	1.9	68	69	3	43	0.13	15.79	1553.26	1701.66	1682.48	0.11	367.10	9.45	4.78
4.73	..._smtcomp09_sw_bv_formula	20	202.42	210.08	oot	-1589.92	0.12	4.7	14	34	34	0	0.20	10.01	1158.33	6215.77	6181.86	0.34	287.17	30.32	4.80
4.74	lfsr_lfsr_008_127_128	20	212.80	198.44	oot	-1601.56	0.11	3.5	17	33	30	0	0.35	42.00	1519.88	5760.96	5737.68	3.42	212.02	25.38	4.82
4.75	lfsr_lfsr_008_143_064	20	103.49	192.94	oot	-1607.06	0.11	4.0	22	60	60	0	0.47	47.52	1735.54	6733.79	6697.75	0.42	217.17	18.86	4.85
4.76	tacas07_Y86_btnft	20	135.77	186.21	oot	-1613.79	0.10	5.0	11	37	24	0	0.15	19.43	1324.71	6496.21	6452.04	0.12	173.44	45.43	4.60
4.77	float_newton.2.2.i	20	143.18	178.62	oot	-1621.38	0.10	4.0								0.00	0.00	0.00	0.00	0.00	4.87
4.78	lfsr_lfsr_008_063_128	20	oot	178.50	oot	-1621.50	0.10	3.7	34	136	65	0	1.37	70.97	1716.10	6544.48	6500.43	0.29	244.84	11.50	4.88
4.79	float_newton.1.2.i	20	139.98	173.63	oot	-1626.37	0.10	4.2								0.00	0.00	0.00	0.00	0.00	4.89
4.80	float_newton.4.2.i	20	143.25	155.94	oot	-1644.06	0.09	4.0	30	150	26	0	1.34	60.15	547.73	1765.02	1703.46	0.00	55.32	4.62	4.90
4.81	float_newton.3.2.i	20	186.56	147.77	oot	-1652.23	0.08	4.4								0.00	0.00	0.00	0.00	0.00	4.91
4.82	lfsr_lfsr_008_159_096	20	275.86	145.85	oot	-1654.15	0.08	3.2	16	46	39	0	0.27	53.90	1710.51	6573.70	6558.65	1.09	134.09	31.16	4.93
4.83	lfsr_lfsr_008_143_080	20	112.13	144.75	oot	-1655.25	0.08	3.9	21	44	44	0	0.45	38.98	1752.20	6775.15	6738.51	0.75	306.01	21.31	4.95
4.84	lfsr_lfsr_008_095_112	20	129.08	142.47	oot	-1657.53	0.08	3.6	19	50	39	0	0.47	41.63	1484.53	6354.45	6330.77	1.58	245.59	23.45	4.96
4.85	lfsr_lfsr_008_127_096	20	113.04	135.23	oot	-1664.77	0.08	3.6	20	49	37	0	0.35	41.36	1582.90	6137.58	6115.45	2.92	157.06	21.76	4.97
4.86	brummayerbiere3_maxxor032	20	143.95	127.04	oot	-1672.96	0.07	1.9	74	543	179	0	0.25	7.13	1784.56	6343.17	6324.49	0.01	50.04	4.19	4.98
4.87	lfsr_lfsr_008_015_096	20	272.75	123.59	oot	-1676.41	0.07	3.9	60	242	242	0	1.51	21.21	1442.89	6973.62	6892.36	0.11	34.07	3.84	4.99
4.88	brummayerbiere_countbits064	20	129.34	121.61	oot	-1678.39	0.07	3.3	40	217	206	0	0.10	4.19	618.66	6995.72	6984.30	0.01	49.00	6.31	4.100
4.89	lfsr_lfsr_008_111_128	20	160.68	112.32	oot	-1687.68	0.06	3.2	19	54	37	0	0.50	50.99	1674.84	6446.70	6424.45	2.91	133.95	24.05	4.101
4.90	lfsr_lfsr_008_047_096	20	160.40	99.36	oot	-1700.64	0.06	3.7	45	235	77	0	1.70	69.44	1704.88	6581.89	6531.37	0.11	136.18	7.73	4.102
4.91	brummayerbiere3_minxor064	20	111.84	93.35	oot	-1706.65	0.05	1.6	80	537	78	0	0.45	9.43	1770.52	5906.32	5857.87	0.01	46.64	4.70	4.103
4.92	lfsr_lfsr_008_159_064	20	149.52	83.48	oot	-1716.52	0.05	3.3	14	51	36	0	0.17	38.26	1582.57	6494.82	6484.77	3.77	138.98	33.65	4.104
4.93	lfsr_lfsr_008_159_048	20	100.56	44.32	oot	-1755.68	0.02	3.5	15	61	41	0	0.13	28.04	1610.17	6647.95	6638.35	2.24	113.62	30.08	4.105
5.1	..._test_v7_r12_vr1_c1_s703	0	oot	oot	sf	1445.78	5.08	4.0	13	20	9	0	1.27	46.55	50.87	106.99	87.11	0.26	6.45	2.33	5.23
5.2	...est_v5_r10_vr10_c1_s7608	20	1199.14	oot	sf	171.17	1.11	4.3	75	165	14	0	4.81	192.56	351.19	899.92	817.54	0.01	9.75	2.27	5.20
5.3	...erbiere4_unconstrained01	0	oom	oom	oom	1.15	1.02	7.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.3
5.4	...erbiere4_unconstrained07	0	oom	oom	oom	0.31	1.00	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.2
5.5	...erbiere4_unconstrained08	0	oom	oom	oom	0.14	1.00	7.1	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.202
5.6	...erbiere4_unconstrained05	0	oom	oom	oom	0.09	1.00	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.204
5.7	core_ext_con_060_001_1024	20	110.60	oot	oot	0.00	1.00	0.1	1	1	1	0	0.00	0.00	141.19	141.19	140.80	141.19	141.19	141.19	5.7
5.8	core_ext_con_060_256_1024	20	101.73	oot	oot	0.00	1.00	3.6	6	17	17	0	21.85	16.40	1373.70	5130.95	4990.91	16.40	273.87	95.02	5.9
5.9	core_ext_con_064_001_1024	20	128.25	oot	oot	0.00	1.00	0.2	1	1	1	0	0.00	0.00	151.54	151.54	151.13	151.54	151.54	151.54	5.10
5.10	fft_Sz1024_34824	0	oot	oot	oot	0.00	1.00	3.3	40	216	216	0	0.80	8.74	483.31	7055.64	7022.57	0.27	38.57	6.30	5.12
5.11	fft_Sz512_15128_0	0	oot	oot	oot	0.00	1.00	3.2	70	413	413	0	0.62	4.39	471.82	7114.80	7078.31	0.15	17.26	2.51	5.13
5.12	fft_Sz512_15128_1	0	oot	oot	oot	0.00	1.00	3.2	70	411	411	0	0.62	4.20	566.51	7110.75	7074.74	0.16	17.26	2.53	5.14
5.13	fft_Sz512_15128_2	0	oot	oot	oot	0.00	1.00	3.2	70	428	428	0	0.54	4.15	560.23	7133.56	7102.59	0.16	16.58	2.51	5.15
5.14	fft_Sz512_15128_4	0	oot	oot	oot	0.00	1.00	3.1	70	416	416	0	0.62	4.30	477.46	7125.12	7088.68	0.14	17.53	2.52	5.16
5.15	fft_Sz512_15128_5	0	oot	oot	oot	0.00	1.00	3.2	70	413	413	0	0.63	4.37	577.73	7116.60	7080.42	0.16	17.55	2.53	5.17
5.16	fft_Sz512_15128_6	0	oot	oot	oot	0.00	1.00	3.2	70	434	434	0	0.55	4.18	454.00	7112.66	7081.80	0.15	16.64	2.48	5.18
5.17	...st_v5_r10_vr10_c1_s15708	0	oot	oot	oot	0.00	1.00	4.3								0.00	0.00	0.00	0.00	0.00	5.19
5.18	...est_v5_r15_vr5_c1_s23844	0	oot	oot	oot	0.00	1.00	4.2								0.00	0.00	0.00	0.00	0.00	5.21
5.19	...st_v7_r12_vr10_c1_s15994	10	1367.71	oot	oot	0.00	1.00	4.4								0.00	0.00	0.00	0.00	0.00	5.22
5.20	...est_v7_r12_vr5_c1_s14336	10	933.17	oot	oot	0.00	1.00	3.8								0.00	0.00	0.00	0.00	0.00	5.24
5.21	...est_v7_r12_vr5_c1_s29826	10	896.50	oot	oot	0.00	1.00	4.4								0.00	0.00	0.00	0.00	0.00	5.25
5.22	...est_v7_r17_vr1_c1_s23882	0	oot	oot	oot	0.00	1.00	4.0	33	59	6	0	4.66	151.18	164.24	341.49	275.35	0.41	8.03	2.87	5.26
5.23	...est_v7_r17_vr1_c1_s30331	0	oot	oot	oot	0.00	1.00	4.3								0.00	0.00	0.00	0.00	0.00	5.27
5.24	...fyInterleavedModMult-128	0	oot	oot	oot	0.00	1.00	4.3	12	8	8	0	0.02	28.44	1677.11	5683.72	5677.60	1.45	302.37	78.94	5.28

Continued on next page

Table C.1 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental)																cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-inc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.25	…ffyInterleavedModMult-32	0	oot	oot	oot	0.00	1.00	3.3	29	116	116	0	0.05	11.16	1556.41	6475.30	6471.64	0.27	110.59	11.24	5.29
5.26	lfsr_lfsr_008_159_128	20	285.74	oot	oot	0.00	1.00	3.6	11	20	20	0	0.19	35.09	1523.00	6241.72	6221.57	4.39	228.79	44.90	5.30
5.27	log-slicing_bvadd_18209	20	101.26	oot	oot	0.00	1.00	1.2	1	1	1	0	0.00	0.00	68.05	68.05	62.39	68.05	68.05	68.05	5.31
5.28	log-slicing_bvadd_19096	20	109.78	oot	oot	0.00	1.00	1.2	1	1	1	0	0.00	0.00	76.13	76.13	70.60	76.13	76.13	76.13	5.32
5.29	log-slicing_bvadd_19983	20	121.59	oot	oot	0.00	1.00	1.2	1	1	1	0	0.00	0.00	82.32	82.32	75.37	82.32	82.32	82.32	5.33
5.30	log-slicing_bvadd_20870	20	132.00	oot	oot	0.00	1.00	1.2	1	1	1	0	0.00	0.00	89.81	89.81	82.02	89.81	89.81	89.81	5.34
5.31	log-slicing_bvadd_21757	20	165.07	oot	oot	0.00	1.00	1.4	1	1	1	0	0.00	0.00	98.60	98.60	88.90	98.60	98.60	98.60	5.35
5.32	log-slicing_bvadd_22644	20	156.03	oot	oot	0.00	1.00	1.4	1	1	1	0	0.00	0.00	102.29	102.29	94.22	102.29	102.29	102.29	5.36
5.33	log-slicing_bvadd_23531	20	167.45	oot	oot	0.00	1.00	1.4	1	1	1	0	0.00	0.00	110.76	110.76	100.30	110.76	110.76	110.76	5.37
5.34	log-slicing_bvadd_24418	20	198.09	oot	oot	0.00	1.00	1.4	1	1	1	0	0.00	0.00	118.53	118.53	106.47	118.53	118.53	118.53	5.38
5.35	log-slicing_bvadd_25305	20	186.34	oot	oot	0.00	1.00	1.4	1	1	1	0	0.00	0.00	125.48	125.48	111.86	125.48	125.48	125.48	5.39
5.36	log-slicing_bvadd_26192	20	202.52	oot	oot	0.00	1.00	1.6	1	1	1	0	0.00	0.00	131.68	131.68	117.33	131.68	131.68	131.68	5.40
5.37	log-slicing_bvadd_27079	20	216.41	oot	oot	0.00	1.00	1.6	1	1	1	0	0.00	0.00	139.52	139.52	121.97	139.52	139.52	139.52	5.41
5.38	log-slicing_bvadd_27966	20	232.01	oot	oot	0.00	1.00	1.7	1	1	1	0	0.00	0.00	147.61	147.61	127.81	147.61	147.61	147.61	5.42
5.39	log-slicing_bvadd_28853	20	245.33	oot	oot	0.00	1.00	1.7	1	1	1	0	0.00	0.00	154.33	154.33	132.01	154.33	154.33	154.33	5.43
5.40	log-slicing_bvadd_29740	20	264.93	oot	oot	0.00	1.00	1.7	1	1	1	0	0.00	0.00	162.23	162.23	136.91	162.23	162.23	162.23	5.44
5.41	log-slicing_bvmul_13	20	428.63	oot	oot	0.00	1.00	0.1	2	2	2	1	0.00	0.02	0.90	1744.92	1744.90	0.27	1743.75	436.23	5.45
5.42	log-slicing_bvmul_14	20	1477.50	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0.02	0.89	1.18	1.16	0.29	0.56	0.39	5.46
5.43	log-slicing_bvmul_15	0	oot	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0.03	0.93	1.25	1.23	0.32	0.58	0.42	5.47
5.44	log-slicing_bvmul_16	0	oot	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0.02	0.85	1.15	1.13	0.30	0.54	0.38	5.48
5.45	log-slicing_bvmul_17	0	oot	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0.22	0.92	1.23	1.12	0.31	0.57	0.41	5.49
5.46	log-slicing_bvmul_18	0	oot	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0.23	0.94	1.21	1.10	0.27	0.57	0.40	5.50
5.47	log-slicing_bvsdiv_19	0	oot	oot	oot	0.00	1.00	1.3	13	66	57	21	0.04	8.60	1383.73	3818.21	3816.78	0.01	718.06	17.76	5.51
5.48	log-slicing_bvsdiv_20	0	oot	oot	oot	0.00	1.00	1.4	11	61	57	10	0.03	8.06	830.78	3206.87	3205.49	0.01	901.53	17.52	5.52
5.49	log-slicing_bvsdiv_21	0	oot	oot	oot	0.00	1.00	1.4	10	59	54	0	0.03	7.61	10.72	48.64	47.25	0.00	1.00	0.32	5.53
5.50	log-slicing_bvsdiv_22	0	oot	oot	oot	0.00	1.00	1.5	10	60	55	1	0.03	8.00	11.23	125.31	123.84	0.00	70.41	0.77	5.54
5.51	log-slicing_bvsdiv_23	0	oot	oot	oot	0.00	1.00	1.5	10	58	56	3	0.03	7.96	11.83	473.86	472.44	0.00	382.98	2.89	5.55
5.52	log-slicing_bvsdiv_24	0	oot	oot	oot	0.00	1.00	1.7	11	64	64	0	0.03	9.08	18.94	68.24	66.66	0.01	1.16	0.39	5.56
5.53	log-slicing_bvsdiv_25	0	oot	oot	oot	0.00	1.00	1.6	10	56	56	3	0.03	8.37	12.47	565.82	564.30	0.06	424.86	3.67	5.57
5.54	log-slicing_bvslt_10201	20	552.92	oot	oot	0.00	1.00	2.9	8	43	43	0	0.02	21.90	1617.93	6398.02	6394.25	8.26	161.93	76.17	5.58
5.55	log-slicing_bvslt_11687	20	665.86	oot	oot	0.00	1.00	3.0	9	42	42	0	0.02	22.62	1749.70	6469.99	6466.19	3.73	229.08	68.11	5.59
5.56	log-slicing_bvslt_13173	20	801.46	oot	oot	0.00	1.00	2.9	8	39	39	0	0.01	23.82	1646.37	6426.06	6422.18	9.27	144.43	75.60	5.60
5.57	log-slicing_bvslt_13916	20	908.99	oot	oot	0.00	1.00	2.9	8	38	38	0	0.02	24.15	1679.83	6424.94	6421.05	2.68	222.83	76.49	5.61
5.58	log-slicing_bvslt_14659	20	838.99	oot	oot	0.00	1.00	2.9	8	37	37	0	0.02	23.13	1612.11	6192.54	6188.59	11.05	193.48	75.52	5.62
5.59	log-slicing_bvslt_16145	20	1188.56	oot	oot	0.00	1.00	2.9	9	36	36	0	0.02	24.45	1674.91	6298.23	6293.91	0.11	212.93	63.62	5.63
5.60	log-slicing_bvslt_16888	20	1502.56	oot	oot	0.00	1.00	2.9	5	21	21	0	0.01	24.21	1529.62	5643.95	5637.55	30.84	307.39	166.00	5.64
5.61	log-slicing_bvslt_17631	20	1750.56	oot	oot	0.00	1.00	2.6	4	16	16	0	0.01	17.32	922.44	5067.39	5061.76	37.05	338.10	194.90	5.65
5.62	log-slicing_bvslt_18374	0	oot	oot	oot	0.00	1.00	2.2	4	16	16	0	0.01	17.46	1763.28	5922.10	5917.78	123.59	664.61	394.81	5.66
5.63	log-slicing_bvslt_19117	0	oot	oot	oot	0.00	1.00	3.1	7	22	22	0	0.01	26.92	1681.60	6164.54	6158.16	15.09	293.92	94.84	5.67
5.64	log-slicing_bvslt_19860	0	oot	oot	oot	0.00	1.00	3.0	7	22	22	0	0.01	27.32	1558.40	5753.29	5746.98	16.14	262.09	88.51	5.68
5.65	log-slicing_bvsmod_16	0	oot	oot	oot	0.00	1.00	0.5	27	120	57	75	0.08	2.19	1706.52	3794.99	3793.83	0.00	244.84	8.27	5.69
5.66	log-slicing_bvsmod_17	0	oot	oot	oot	0.00	1.00	1.3	17	82	55	34	0.06	10.50	1663.84	3763.64	3761.78	0.00	594.57	13.94	5.70
5.67	log-slicing_bvsmod_18	0	oot	oot	oot	0.00	1.00	1.4	13	70	60	9	0.04	8.56	1551.09	2248.69	2247.13	0.00	1049.21	10.97	5.71
5.68	log-slicing_bvsmod_19	0	oot	oot	oot	0.00	1.00	1.4	12	68	62	11	0.03	8.18	143.21	2493.03	2491.46	0.00	870.18	12.22	5.72
5.69	log-slicing_bvsmod_20	0	oot	oot	oot	0.00	1.00	1.4	10	58	56	4	0.03	7.38	853.97	1382.89	1381.37	0.00	844.51	8.38	5.73
5.70	log-slicing_bvsrem_16	0	oot	oot	oot	0.00	1.00	0.4	19	81	48	52	0.05	1.38	1683.88	3729.31	3728.52	0.00	358.46	12.60	5.74
5.71	log-slicing_bvsrem_17	0	oot	oot	oot	0.00	1.00	1.0	16	69	46	39	0.05	8.21	1691.77	4551.82	4550.25	0.00	545.02	19.05	5.75
5.72	log-slicing_bvsrem_18	0	oot	oot	oot	0.00	1.00	1.2	12	59	52	15	0.04	6.82	1463.97	3856.03	3854.70	0.01	843.80	20.96	5.76
5.73	log-slicing_bvsrem_19	0	oot	oot	oot	0.00	1.00	1.3	10	56	51	4	0.03	7.01	739.99	1616.50	1615.23	0.01	731.51	10.71	5.77
5.74	log-slicing_bvsrem_20	0	oot	oot	oot	0.00	1.00	1.1	9	54	50	1	0.02	6.70	8.34	687.91	686.85	0.00	656.11	6.25	5.78
5.75	log-slicing_bvsub_06991	20	307.68	oot	oot	0.00	1.00	0.2	1	1	1	0	0.00	0.00	7.36	7.36	4.69	7.36	7.36	7.36	5.79

Continued on next page

Table C.1 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental)																cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-inc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.76	log-slicing_bvsub_07988	20	321.69	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0.00	8.29	8.29	4.85	8.29	8.29	8.29	5.80
5.77	log-slicing_bvsub_08985	20	378.56	oot	oot	0.00	1.00	0.1	1	1	1	0	0.00	0.00	9.19	9.19	4.92	9.19	9.19	9.19	5.81
5.78	log-slicing_bvsub_09982	20	328.06	oot	oot	0.00	1.00	0.2	1	1	1	0	0.00	0.00	10.19	10.19	5.14	10.19	10.19	10.19	5.82
5.79	log-slicing_bvsub_10979	20	161.21	oot	oot	0.00	1.00	0.1	1	1	1	0	0.00	0.00	11.18	11.18	5.15	11.18	11.18	11.18	5.83
5.80	log-slicing_bvsub_11976	20	664.63	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	12.78	12.78	5.36	12.78	12.78	12.78	5.84
5.81	log-slicing_bvsub_12973	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	13.98	13.98	5.39	13.98	13.98	13.98	5.85
5.82	log-slicing_bvsub_13970	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	15.09	15.09	5.61	15.09	15.09	15.09	5.86
5.83	log-slicing_bvsub_14967	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	16.90	16.90	5.56	16.90	16.90	16.90	5.87
5.84	log-slicing_bvsub_15964	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	18.90	18.90	5.76	18.90	18.90	18.90	5.88
5.85	log-slicing_bvsub_16961	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	20.56	20.56	5.82	20.56	20.56	20.56	5.89
5.86	log-slicing_bvsub_17958	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0.00	15.82	15.82	0.20	15.82	15.82	15.82	5.90
5.87	log-slicing_bvsub_18955	0	oot	oot	oot	0.00	1.00	0.6	1	1	1	0	0.00	0.00	18.04	18.04	0.21	18.04	18.04	18.04	5.91
5.88	log-slicing_bvsub_19952	0	oot	oot	oot	0.00	1.00	0.6	1	1	1	0	0.00	0.00	19.55	19.55	0.21	19.55	19.55	19.55	5.92
5.89	log-slicing_bvsub_20949	0	oot	oot	oot	0.00	1.00	0.6	1	1	1	0	0.00	0.00	22.78	22.78	0.22	22.78	22.78	22.78	5.93
5.90	log-slicing_bvsub_21946	0	oot	oot	oot	0.00	1.00	0.7	1	1	1	0	0.00	0.00	24.95	24.95	0.23	24.95	24.95	24.95	5.94
5.91	log-slicing_bvsub_22943	0	oot	oot	oot	0.00	1.00	0.7	1	1	1	0	0.00	0.00	27.39	27.39	0.24	27.39	27.39	27.39	5.95
5.92	log-slicing_bvsub_23940	0	oot	oot	oot	0.00	1.00	0.8	1	1	1	0	0.00	0.00	32.56	32.56	0.25	32.56	32.56	32.56	5.96
5.93	log-slicing_bvudiv_19	0	oot	oot	oot	0.00	1.00	0.8	8	29	28	11	0.02	5.61	1783.56	4623.64	4622.51	0.00	1349.43	53.15	5.97
5.94	log-slicing_bvudiv_20	0	oot	oot	oot	0.00	1.00	0.9	8	32	30	9	0.02	5.84	1144.58	3234.60	3233.45	0.00	1123.72	36.34	5.98
5.95	log-slicing_bvudiv_21	0	oot	oot	oot	0.00	1.00	1.0	7	32	30	5	0.01	6.40	5.92	501.24	500.17	0.00	413.08	6.60	5.99
5.96	log-slicing_bvudiv_22	0	oot	oot	oot	0.00	1.00	1.0	7	33	32	4	0.02	6.58	6.21	1882.51	1881.36	0.00	1648.69	24.77	5.100
5.97	log-slicing_bvudiv_23	0	oot	oot	oot	0.00	1.00	1.0	7	33	32	4	0.02	6.49	6.37	771.82	770.69	0.01	663.63	10.16	5.101
5.98	log-slicing_bvudiv_24	0	oot	oot	oot	0.00	1.00	1.0	6	28	26	0	0.01	5.45	5.54	17.75	16.62	0.06	1.32	0.33	5.102
5.99	log-slicing_bvudiv_25	0	oot	oot	oot	0.00	1.00	1.2	7	36	32	1	0.01	6.97	6.75	1555.78	1554.45	0.02	1531.66	20.47	5.103
5.100	log-slicing_bvult_10985	20	556.62	oot	oot	0.00	1.00	2.7	7	40	40	0	0.02	21.27	1550.51	6576.17	6572.24	3.53	229.72	90.08	5.104
5.101	log-slicing_bvult_11982	20	518.16	oot	oot	0.00	1.00	2.9	7	34	34	0	0.01	19.19	1516.42	6232.61	6228.83	1.21	173.03	83.10	5.105
5.102	log-slicing_bvult_12979	20	893.49	oot	oot	0.00	1.00	2.9	9	40	40	0	0.01	23.69	1748.24	6385.20	6381.06	2.21	195.32	67.21	5.106
5.103	log-slicing_bvult_13976	20	893.73	oot	oot	0.00	1.00	3.0	8	39	39	0	0.02	25.17	1712.61	6534.54	6530.33	0.15	183.29	77.79	5.107
5.104	log-slicing_bvult_15970	20	1070.63	oot	oot	0.00	1.00	3.1	10	36	36	0	0.01	24.77	1351.23	4874.46	4869.77	0.20	190.95	43.14	5.108
5.105	log-slicing_bvult_16967	0	oot	oot	oot	0.00	1.00	2.9	5	21	21	0	0.01	23.31	1372.34	5776.04	5770.05	51.65	256.20	144.40	5.109
5.106	log-slicing_bvult_17964	0	oot	oot	oot	0.00	1.00	2.9	5	21	21	0	0.01	25.93	1268.47	5503.72	5497.78	39.33	252.85	127.99	5.110
5.107	log-slicing_bvult_18961	0	oot	oot	oot	0.00	1.00	2.3	4	16	16	0	0.01	18.22	1566.89	5066.39	5062.25	73.34	508.61	298.02	5.111
5.108	log-slicing_bvult_19958	0	oot	oot	oot	0.00	1.00	3.2	8	22	22	0	0.01	27.36	1738.21	5681.44	5675.49	0.19	481.65	71.92	5.112
5.109	log-slicing_bvult_20955	0	oot	oot	oot	0.00	1.00	3.0	7	21	21	0	0.01	26.57	1156.98	5151.56	5145.73	10.85	529.51	75.76	5.113
5.110	log-slicing_bvult_21952	0	oot	oot	oot	0.00	1.00	3.0	8	21	21	0	0.01	26.02	1525.47	5716.05	5710.33	3.42	465.07	71.45	5.114
5.111	log-slicing_bvult_22949	0	oot	oot	oot	0.00	1.00	3.0	8	21	21	0	0.01	26.88	1005.14	5915.08	5909.22	12.83	757.18	70.42	5.115
5.112	log-slicing_bvult_23946	0	oot	oot	oot	0.00	1.00	2.9	7	20	20	0	0.01	23.71	655.31	6183.79	6177.98	0.28	674.88	79.28	5.116
5.113	log-slicing_bvult_24943	0	oot	oot	oot	0.00	1.00	3.0	8	19	19	0	0.01	27.26	1112.22	4499.98	4494.42	0.27	414.37	55.56	5.117
5.114	log-slicing_bvult_25940	0	oot	oot	oot	0.00	1.00	3.0	8	19	19	0	0.01	24.35	708.31	3575.96	3570.28	0.21	690.95	43.08	5.118
5.115	log-slicing_bvult_9988	20	555.03	oot	oot	0.00	1.00	2.9	8	44	44	0	0.02	22.95	1618.90	6318.06	6314.09	4.81	194.96	71.80	5.119
5.116	log-slicing_bvurem_17	0	oot	oot	oot	0.00	1.00	0.5	8	30	24	16	0.02	3.43	894.03	3730.51	3729.84	0.00	774.50	40.99	5.120
5.117	log-slicing_bvurem_18	0	oot	oot	oot	0.00	1.00	0.6	7	29	29	7	0.01	3.54	1115.85	2549.92	2549.35	0.00	1111.69	35.91	5.121
5.118	log-slicing_bvurem_19	0	oot	oot	oot	0.00	1.00	0.6	6	29	27	2	0.01	3.47	4.04	443.37	442.76	0.00	393.36	7.15	5.122
5.119	log-slicing_bvurem_20	0	oot	oot	oot	0.00	1.00	0.8	7	32	30	1	0.02	3.88	4.92	428.83	428.21	0.00	413.91	6.60	5.123
5.120	log-slicing_bvurem_21	0	oot	oot	oot	0.00	1.00	0.7	6	30	30	0	0.01	3.82	4.21	15.26	14.62	0.15	0.92	0.25	5.124
5.121	log-slicing_bvurem_22	0	oot	oot	oot	0.00	1.00	0.9	7	32	32	0	0.02	4.43	5.45	16.71	16.06	0.00	0.96	0.26	5.125
5.122	pipe_pipe-noabs.atlas.qf_bv	20	1290.10	oot	oot	0.00	1.00	0.7								0.00	0.00	0.00	0.00	0.00	5.126
5.123	pspace_power2sum.5500	20	496.16	oot	oot	0.00	1.00	2.0	1	1	1	0	0.00	0.00	1208.93	1208.93	1208.57	1208.93	1208.93	1208.93	5.127
5.124	pspace_power2sum.5898	20	343.57	oot	oot	0.00	1.00	2.2	1	1	1	0	0.00	0.00	1472.52	1472.52	1472.12	1472.52	1472.52	1472.52	5.128
5.125	pspace_power2sum.6296	20	434.76	oot	oot	0.00	1.00	2.2	1	1	1	0	0.00	0.00	1780.52	1780.52	1780.08	1780.52	1780.52	1780.52	5.129
5.126	pspace_power2sum.6495	20	504.51	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.130

Continued on next page

Table C.1 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental)																cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-inc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.127	pspace_power2sum.6694	20	1005.81	oot	oot	0.00	1.00	2.0								0.00	0.00	0.00	0.00	0.00	5.131
5.128	pspace_power2sum.6893	20	1102.02	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.132
5.129	pspace_power2sum.7092	20	452.50	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.133
5.130	pspace_power2sum.7291	20	971.68	oot	oot	0.00	1.00	2.0								0.00	0.00	0.00	0.00	0.00	5.134
5.131	pspace_power2sum.7490	20	505.69	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.135
5.132	pspace_power2sum.7689	20	515.93	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.136
5.133	pspace_power2sum.7888	20	1145.49	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.137
5.134	pspace_power2sum.8087	0	oot	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.138
5.135	pspace_power2sum.8286	20	898.41	oot	oot	0.00	1.00	2.0								0.00	0.00	0.00	0.00	0.00	5.139
5.136	pspace_power2sum.8485	20	872.30	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.140
5.137	pspace_power2sum.8684	20	1028.30	oot	oot	0.00	1.00	2.0								0.00	0.00	0.00	0.00	0.00	5.141
5.138	pspace_power2sum.8883	20	1035.88	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.142
5.139	pspace_power2sum.9082	20	1301.91	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.143
5.140	pspace_power2sum.9281	20	1338.53	oot	oot	0.00	1.00	2.0								0.00	0.00	0.00	0.00	0.00	5.144
5.141	pspace_power2sum.9480	20	751.83	oot	oot	0.00	1.00	2.1								0.00	0.00	0.00	0.00	0.00	5.145
5.142	pspace_power2sum.9679	0	oot	oot	oot	0.00	1.00	2.0								0.00	0.00	0.00	0.00	0.00	5.146
5.143	pspace_power2sum.9878	20	573.35	oot	oot	0.00	1.00	2.1								0.00	0.00	0.00	0.00	0.00	5.147
5.144	RWS_Example_15.txt	0	oot	oot	oot	0.00	1.00	0.3								0.00	0.00	0.00	0.00	0.00	5.148
5.145	RWS_Example_19.txt	0	oot	oot	oot	0.00	1.00	0.3								0.00	0.00	0.00	0.00	0.00	5.149
5.146	RWS_Example_20.txt	0	oot	oot	oot	0.00	1.00	0.4								0.00	0.00	0.00	0.00	0.00	5.150
5.147	RWS_Example_7.txt	0	oot	oot	oot	0.00	1.00	0.1								0.00	0.00	0.00	0.00	0.00	5.151
5.148	uum_uum12	0	oot	oot	oot	0.00	1.00	0.6								0.00	0.00	0.00	0.00	0.00	5.152
5.149	uum_uum16	0	oot	oot	oot	0.00	1.00	1.9	60	452	301	0	0.35	3.02	637.62	6943.67	6935.14	0.00	42.01	3.54	5.153
5.150	uum_uum20	0	oot	oot	oot	0.00	1.00	3.1								0.00	0.00	0.00	0.00	0.00	5.154
5.151	uum_uum24	0	oot	oot	oot	0.00	1.00	3.2								0.00	0.00	0.00	0.00	0.00	5.155
5.152	uum_uum28	0	oot	oot	oot	0.00	1.00	3.4								0.00	0.00	0.00	0.00	0.00	5.156
5.153	uum_uum32	0	oot	oot	oot	0.00	1.00	3.2								0.00	0.00	0.00	0.00	0.00	5.157
5.154	VS3_VS3-benchmark-A10	0	oot	oot	oot	0.00	1.00	3.4	45	207	207	0	0.29	17.33	1766.65	7005.49	6999.80	0.17	44.45	5.99	5.158
5.155	VS3_VS3-benchmark-A11	0	oot	oot	oot	0.00	1.00	3.4	47	213	211	0	0.31	16.74	1763.38	6973.01	6967.46	0.15	44.39	5.78	5.159
5.156	VS3_VS3-benchmark-A12	0	oot	oot	oot	0.00	1.00	3.4	32	160	160	0	0.22	12.87	1775.76	7071.22	7067.50	0.37	44.37	9.08	5.160
5.157	VS3_VS3-benchmark-A13	0	oot	oot	oot	0.00	1.00	3.2	30	111	111	0	0.17	11.70	639.74	7067.93	7063.94	0.37	53.63	10.53	5.161
5.158	VS3_VS3-benchmark-A8	0	oot	oot	oot	0.00	1.00	3.2	34	153	153	0	0.23	13.92	1784.07	7033.20	7029.43	0.37	44.83	9.29	5.162
5.159	VS3_VS3-benchmark-S2	0	oot	oot	oot	0.00	1.00	3.5	20	42	42	0	0.07	15.64	1745.30	6779.89	6777.36	1.79	108.51	22.91	5.163
5.160	...1_alt_true.BV.c.cil.c.21	0	oot	oot	oot	0.00	1.00	4.7	11	15	14	0	0.06	5.96	1597.03	5953.73	5920.10	0.53	321.59	58.95	5.164
5.161	brummayerbiere2_smulov1bw48	0	oot	oot	oot	0.00	1.00	1.4								0.00	0.00	0.00	0.00	0.00	5.165
5.162	...erbiere3_isqrtaddeqcheck	0	oot	oot	oot	0.00	1.00	3.3	60	413	248	13	0.11	5.11	723.78	6504.26	6484.73	0.00	93.81	3.72	5.166
5.163	...mayerbiere3_isqrtaddnoif	0	oot	oot	oot	0.00	1.00	1.9	30	217	110	0	0.07	3.86	1358.86	6672.76	6663.85	0.02	165.22	11.12	5.167
5.164	...mayerbiere3_isqrteqcheck	0	oot	oot	oot	0.00	1.00	3.2	50	336	228	0	0.09	4.76	1103.39	6636.53	6615.94	0.00	99.25	4.71	5.168
5.165	brummayerbiere3_isqrtnoif	0	oot	oot	oot	0.00	1.00	2.7	29	200	122	0	0.04	3.98	1689.64	6590.98	6580.93	0.00	186.22	11.96	5.169
5.166	brummayerbiere3_maxor128	0	oot	oot	oot	0.00	1.00	0.9	10	11	3	8	0.05	6.97	1564.51	1697.83	1692.19	1.19	266.06	58.55	5.170
5.167	brummayerbiere3_maxor256	0	oot	oot	oot	0.00	1.00	4.4	13	8	3	1	0.05	26.76	1305.73	2666.01	2643.62	0.71	423.94	80.79	5.171
5.168	brummayerbiere3_maxxor064	0	oot	oot	oot	0.00	1.00	3.9	28	144	95	0	0.13	7.48	1623.16	6507.30	6490.59	0.01	130.82	12.16	5.172
5.169	brummayerbiere3_maxxor128	0	oot	oot	oot	0.00	1.00	4.1	16	38	38	0	0.06	12.01	1779.01	6561.64	6545.42	0.52	147.09	28.65	5.173
5.170	brummayerbiere3_maxxor256	0	oot	oot	oot	0.00	1.00	3.6	9	10	10	0	0.03	16.47	1208.76	5502.20	5486.96	1.26	466.29	76.42	5.174
5.171	...biere3_maxxormaxorand032	0	oot	oot	oot	0.00	1.00	3.4	130	898	304	0	0.97	14.77	1737.71	5979.51	5943.09	0.00	46.30	1.97	5.175
5.172	...biere3_maxxormaxorand064	0	oot	oot	oot	0.00	1.00	3.9	67	424	82	0	1.12	31.47	1721.58	5382.33	5332.68	0.00	109.02	4.70	5.176
5.173	...biere3_maxxormaxorand128	0	oot	oot	oot	0.00	1.00	4.1	16	31	25	0	0.19	17.84	1634.64	5628.71	5594.85	0.08	181.50	26.18	5.177
5.174	...biere3_maxxormaxorand256	0	oot	oot	oot	0.00	1.00	4.0								0.00	0.00	0.00	0.00	0.00	5.178
5.175	...yerbiere3_minandmaxor128	0	oot	oot	oot	0.00	1.00	4.3	17	67	17	1	0.30	28.09	1557.51	5156.81	5105.12	0.13	376.47	28.81	5.179
5.176	...yerbiere3_minandmaxor256	0	oot	oot	oot	0.00	1.00	4.9	9	10	7	1	0.09	21.61	1174.34	4471.89	4445.52	8.47	548.10	86.00	5.180
5.177	brummayerbiere3_minxor128	0	oot	oot	oot	0.00	1.00	4.8	33	179	77	0	0.34	11.18	1782.28	6607.94	6575.14	0.08	103.21	10.81	5.181

Continued on next page

Table C.1 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental)																cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-inc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.178	brummayerbiere3_minxor256	0	oot	oot	oot	0.00	1.00	4.8	14	32	18	0	0.14	13.31	1410.12	5711.40	5672.88	0.14	342.05	30.38	5.182
5.179	...erbiere3_minxorminand064	0	oot	oot	oot	0.00	1.00	4.8	55	316	127	0	0.80	15.54	1756.82	6361.53	6298.96	0.01	58.28	5.47	5.183
5.180	...erbiere3_minxorminand128	0	oot	oot	oot	0.00	1.00	4.8	17	47	43	0	0.27	12.25	1629.40	6366.90	6337.27	0.07	201.31	24.77	5.184
5.181	...erbiere3_minxorminand256	0	oot	oot	oot	0.00	1.00	4.6	9	12	11	0	0.17	17.83	1070.39	4184.51	4155.48	0.64	614.60	56.55	5.185
5.182	brummayerbiere3_mulhs16	0	oot	oot	oot	0.00	1.00	0.3	14	89	53	2	0.00	0.24	12.04	1744.27	1744.08	0.00	1222.12	7.82	5.186
5.183	brummayerbiere3_mulhs32	0	oot	oot	oot	0.00	1.00	1.1	21	142	92	1	0.01	1.54	27.14	98.60	97.93	0.00	13.08	0.25	5.187
5.184	brummayerbiere3_mulhs64	0	oot	oot	oot	0.00	1.00	3.9	35	142	78	0	0.01	10.89	1722.15	6349.79	6347.04	0.00	320.31	9.55	5.188
5.185	...mayerbiere_countbits1024	0	oot	oot	oot	0.00	1.00	1.8	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.189
5.186	brummayerbiere_countbits128	0	oot	oot	oot	0.00	1.00	3.5	25	72	70	0	0.07	9.69	1752.75	6623.52	6608.30	0.05	104.18	15.30	5.190
5.187	brummayerbiere_countbits256	0	oot	oot	oot	0.00	1.00	3.8	11	22	20	0	0.05	14.66	1738.56	6026.47	6007.48	0.30	253.95	50.64	5.191
5.188	brummayerbiere_countbits512	0	oot	oot	oot	0.00	1.00	3.7	5	6	4	0	0.05	23.28	1493.69	2955.00	2937.20	1.14	485.69	184.69	5.192
5.189	...biere_countbitsrotate032	0	oot	oot	oot	0.00	1.00	0.2	5	16	16	0	0.01	0.08	17.88	50.39	50.14	0.69	3.93	1.63	5.193
5.190	...biere_countbitsrotate064	0	oot	oot	oot	0.00	1.00	0.7	6	28	28	0	0.02	0.92	113.64	479.62	477.47	0.08	14.84	7.74	5.194
5.191	...biere_countbitsrotate128	0	oot	oot	oot	0.00	1.00	1.6	5	16	16	0	0.03	2.96	386.34	1037.07	1031.83	0.25	80.75	33.45	5.195
5.192	...biere_countbitsrotate256	0	oot	oot	oot	0.00	1.00	4.3	5	8	8	0	0.03	7.44	1091.98	3909.74	3893.00	7.51	521.47	156.39	5.196
5.193	...yerbiere_countbitssrl064	0	oot	oot	oot	0.00	1.00	1.3	13	68	20	0	0.05	4.26	77.76	218.29	207.09	0.00	18.77	1.58	5.197
5.194	...yerbiere_countbitssrl128	0	oot	oot	oot	0.00	1.00	3.8	15	66	23	0	0.14	20.24	691.17	2940.16	2910.18	0.00	120.66	15.81	5.198
5.195	...yerbiere_countbitssrl256	0	oot	oot	oot	0.00	1.00	4.4	13	10	5	0	0.06	30.39	1564.85	4214.21	4168.68	0.12	489.00	78.04	5.199
5.196	challenge_integerOverflow	0	oot	oot	oot	0.00	1.00	0.4	5	12	8	0	0.00	0.17	13.52	39.98	39.47	0.00	9.72	1.74	5.200
5.197	...erbiere4_unconstrained04	0	oom	oom	oom	-0.21	1.00	7.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.4
5.198	...erbiere4_unconstrained03	0	oom	oom	oom	-0.36	0.99	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.201
5.199	...erbiere4_unconstrained06	0	oom	oom	oom	-0.38	0.99	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.1
5.200	...erbiere4_unconstrained09	0	oom	oom	oom	-0.65	0.99	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.203
5.201	...erbiere4_unconstrained10	0	oom	oom	oom	-1.20	0.99	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.205

Table C.2: Detailed results of PBoolector with FLR (Parallel-Incremental-FLR) compared to the reference time Base-Incremental. Last column par-inc contains cross reference to detailed results of Parallel-Incremental in Tab.C.1.

rank	benchmark	stat	sequ-btor	ref	PBoolector with FLR (Parallel-Incremental-FLR)																		cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-inc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
1.1	brummayerbiere2_smulov1bw32	20	1056.25	oot	492.85	1307.15	3.65	0.6	55	276	52	43	0.40	0	0	488.50	1373.24	1369.56	0.00	28.29	1.87	1.1
1.2	log-slicing_bvurem_16	20	oot	oot	1009.32	790.68	1.78	0.2	8	24	23	22	0.04	0	0	1008.14	2527.61	2527.43	0.00	340.62	30.45	1.2
1.3	...iffyInterleavedModMult-8	20	oot	oot	1406.26	393.74	1.28	0.4	67	484	132	0	0.10	2	0	1403.18	5227.03	5225.57	0.00	21.50	3.92	1.3
1.4	log-slicing_bvudiv_18	20	oot	oot	1544.48	255.52	1.17	0.8	11	40	31	23	0.08	0	0	1532.73	4254.44	4253.22	0.00	535.61	34.59	1.7
1.5	log-slicing_bvsmod_15	20	oot	oot	1752.25	47.75	1.03	0.4	40	182	51	113	0.10	0	0	1747.47	4090.61	4089.06	0.00	137.77	6.22	1.9
2.1	log-slicing_bvudiv_17	20	1270.36	1373.34	623.39	749.95	2.20	0.7	10	33	23	21	0.08	0	0	613.84	1576.84	1575.68	0.00	186.44	15.77	2.1
2.2	log-slicing_bvurem_15	20	oot	1147.71	416.20	731.51	2.76	0.2	8	24	23	20	0.05	0	0	415.02	963.48	963.28	0.00	136.89	12.04	2.2
2.3	log-slicing_bvsmod_14	20	1414.54	1350.59	745.48	605.11	1.81	0.4	41	180	46	118	0.10	0	0	740.96	1779.42	1777.90	0.00	63.29	2.81	2.3
2.4	log-slicing_bvmul_12	20	129.19	962.31	423.77	538.54	2.27	0.1	2	2	2	2	0.00	0	0	423.07	817.83	817.81	0.28	422.28	163.57	2.4
2.5	log-slicing_bvudiv_16	20	390.99	521.12	251.41	269.71	2.07	0.2	8	24	22	20	0.04	0	0	250.03	739.93	739.67	0.00	86.78	9.25	2.8
2.6	log-slicing_bvurem_14	20	757.16	270.88	133.14	137.74	2.03	0.1	8	24	22	19	0.05	0	0	132.04	445.39	445.22	0.00	60.87	5.71	2.11
2.7	log-slicing_bvsmod_13	20	360.26	438.26	359.28	78.98	1.22	0.3	35	158	50	98	0.10	0	0	355.42	925.82	924.56	0.00	28.09	1.64	2.13
2.8	challenge_multiplyOverflow	20	1035.61	1002.11	925.22	76.89	1.08	0.1	2	2	2	2	0.00	0	0	924.46	1514.03	1514.00	0.56	922.68	302.81	2.14
2.9	log-slicing_bvudiv_15	20	162.70	192.95	117.69	75.26	1.64	0.2	8	23	21	18	0.04	0	0	116.38	340.97	340.75	0.00	41.75	4.55	2.15
2.10	log-slicing_bvurem_13	20	147.97	116.67	57.13	59.54	2.04	0.1	7	21	17	17	0.05	0	0	56.02	180.88	180.75	0.00	23.19	2.70	2.17
2.11	brummayerbiere2_smulov1bw24	20	134.33	132.01	73.35	58.66	1.80	0.3	25	133	42	15	0.24	0	0	71.66	220.89	219.71	0.00	15.99	0.67	2.18
2.12	log-slicing_bvsmod_12	20	138.58	174.80	161.49	13.31	1.08	0.3	32	150	51	77	0.10	0	0	157.87	459.84	458.76	0.00	13.93	0.93	2.19
2.13	...yerbiere3_isqrtinvalidvc	10	181.66	15.89	9.49	6.40	1.67	0.1	4	16	16	0	0.17	0	0	8.60	29.17	28.27	0.23	2.26	0.94	2.20
3.1	brummayerbiere_bitrev8192	20	101.97	101.49	101.50	-0.01	1.00	0.0	0	1	1	0	0.00	0	0	0.04	0.04	0.00	0.04	0.04	0.04	2.30
3.2	bmc-bv_magic	10	443.86	453.44	453.47	-0.03	1.00	0.2	0	1	1	0	0.21	1	0	0.45	0.45	0.39	0.45	0.45	0.45	2.21
3.3	bmc-bv_graycode	10	442.54	449.03	449.37	-0.34	1.00	0.2	0	1	1	0	0.21	1	0	0.04	0.04	0.00	0.04	0.04	0.04	3.1
3.4	...erbiere4_unconstrained02	10	oot	44.23	44.75	-0.52	0.99	4.2	0	1	1	0	0.00	0	0	39.91	39.91	7.79	39.91	39.91	39.91	3.2
3.5	RWS_Example_18.txt	10	160.61	193.35	198.63	-5.28	0.97	0.1	0	1	1	0	5.05	0	0	192.76	192.76	192.60	192.76	192.76	192.76	3.3
3.6	...mayerbiere2_umulov1bw224	20	108.42	90.16	112.07	-21.91	0.80	0.3	1	2	2	0	17.48	0	0	90.90	90.90	89.42	0.00	88.35	30.30	3.4
3.7	log-slicing_bvsub_04000	20	545.84	842.77	880.53	-37.76	0.96	0.1	1	1	1	1	0.00	0	0	879.37	879.37	878.45	5.84	873.53	439.68	3.6
3.8	...mayerbiere2_umulov1bw256	20	143.94	119.05	157.25	-38.20	0.76	0.4	1	2	2	0	22.60	0	0	130.51	130.51	128.44	0.00	102.56	43.50	3.5
3.9	...mayerbiere2_smulov2bw384	20	122.06	15.64	89.07	-73.43	0.18	0.4	0	1	1	0	74.04	0	0	14.07	14.07	8.85	14.07	14.07	14.07	2.23
3.10	...ayerbiere2_smulov4bw0512	10	128.78	2.57	87.79	-85.22	0.03	0.1	0	1	1	0	85.38	0	0	1.58	1.58	0.17	1.58	1.58	1.58	2.26
3.11	...mayerbiere2_smulov2bw448	20	150.63	43.69	139.15	-95.46	0.31	0.5	0	1	1	0	95.59	0	0	42.50	42.50	34.74	42.50	42.50	42.50	2.24
3.12	...biere3_isqrtaddinvalidvc	10	185.60	16.00	126.91	-110.91	0.13	1.9	40	270	222	30	0.16	0	0	124.20	472.25	458.51	0.00	5.53	0.42	3.12
3.13	...mayerbiere2_smulov2bw512	20	198.38	36.68	161.17	-124.49	0.23	0.6	0	1	1	0	124.51	0	0	35.27	35.27	22.96	35.27	35.27	35.27	2.25
3.14	...ayerbiere2_smulov4bw0640	10	141.93	3.96	135.31	-131.35	0.03	0.2	0	1	1	0	131.59	0	0	2.87	2.87	0.27	2.87	2.87	2.87	2.27
3.15	...ayerbiere2_smulov4bw0768	10	563.42	5.80	181.98	-176.18	0.03	0.4	0	1	1	0	176.26	0	0	4.72	4.72	0.41	4.72	4.72	4.72	2.28
3.16	log-slicing_bvsrem_14	20	256.90	269.58	477.64	-208.06	0.56	0.3	23	91	46	71	0.08	0	0	475.09	1359.33	1358.57	0.00	55.63	4.08	3.15
3.17	core_ext_con_044_002_1024	20	195.82	1571.35	1786.91	-215.56	0.88	0.3	1	1	1	1	7.52	4	0	1778.69	1778.69	1778.41	88.85	1689.84	889.35	2.6
3.18	core_ext_con_036_002_1024	20	269.99	1156.19	1377.96	-221.77	0.84	0.2	1	1	1	1	5.09	4	0	1372.23	1372.23	1372.00	68.21	1304.02	686.11	2.9
3.19	log-slicing_bvsrem_15	20	791.10	927.62	1154.09	-226.47	0.80	0.3	23	92	46	76	0.08	0	0	1151.34	2926.24	2925.45	0.00	133.06	8.31	3.14
3.20	brummayerbiere_nlzbe256	20	117.51	157.97	389.26	-231.29	0.41	1.3	7	15	6	0	48.85	1	0	332.13	674.79	663.60	0.10	137.65	23.27	2.16
3.21	...ayerbiere2_smulov4bw0896	10	1274.52	8.26	246.91	-238.65	0.03	0.5	0	1	1	0	238.90	0	0	6.94	6.94	0.54	6.94	6.94	6.94	2.29
3.22	pspace_shift1add.22961	20	110.71	110.86	369.94	-259.08	0.30	0.3	1	1	1	1	0.00	0	0	369.22	369.22	368.73	12.68	356.53	184.61	3.17
3.23	pspace_shift1add.23958	20	114.08	115.50	403.71	-288.21	0.29	0.4	1	1	1	1	0.00	0	0	402.94	402.94	402.42	13.87	389.07	201.47	3.18
3.24	...ayerbiere2_smulov4bw1024	10	1518.87	10.91	317.42	-306.51	0.03	0.5	0	1	1	0	306.92	0	0	9.32	9.32	0.71	9.32	9.32	9.32	2.22
3.25	pspace_shift1add.24955	20	128.47	121.79	436.12	-314.33	0.28	0.4	1	1	1	1	0.00	0	0	435.42	435.42	434.87	14.61	420.81	217.71	3.19
3.26	pspace_shift1add.25952	20	143.28	129.38	468.00	-338.62	0.28	0.4	1	1	1	1	0.00	0	0	467.19	467.19	466.64	15.51	451.68	233.60	3.20
3.27	pspace_shift1add.26949	20	152.33	135.58	494.53	-358.95	0.27	0.4	1	1	1	1	0.00	0	0	493.76	493.76	493.18	16.34	477.42	246.88	3.21

Continued on next page

Table C.2 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector with FLR (Parallel-Incrmental-FLR)																	cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-inc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
3.28	pspace_shift1add.27946	20	167.81	143.34	530.91	-387.57	0.27	0.6	1	1	1	1	0.00	0	0	530.17	530.17	529.57	17.65	512.52	265.08	3.23
3.29	pspace_shift1add.28943	20	186.84	148.33	563.87	-415.54	0.26	0.6	1	1	1	1	0.00	0	0	563.11	563.11	562.49	18.42	544.69	281.56	3.25
3.30	log-slicing_bvsdiv_16	20	182.35	223.39	661.97	-438.58	0.34	0.4	24	94	50	75	0.08	0	0	659.13	1901.04	1900.17	0.00	66.88	5.21	3.29
3.31	pspace_shift1add.29940	20	195.38	157.23	596.64	-439.41	0.26	0.6	1	1	1	1	0.00	0	0	595.89	595.89	595.26	19.38	576.51	297.95	3.26
3.32	brummayerbiere3_isqrtadd	20	144.73	148.88	638.11	-489.23	0.23	1.6	41	289	186	11	0.16	0	0	635.68	2364.05	2349.27	0.01	37.96	2.28	3.28
3.33	brummayerbiere3_isqrt	20	145.81	148.11	912.75	-764.64	0.16	1.5	100	538	170	243	0.16	0	0	908.42	3248.27	3229.06	0.00	33.81	1.31	3.22
3.34	...smtcomp09_std_bv_formula	20	108.86	169.61	1024.81	-855.20	0.17	3.5	41	258	86	0	696.45	21	0	311.95	1212.30	1175.79	0.01	20.71	1.52	3.13
3.35	...mtcomp09_full_bv_formula	20	172.46	167.31	1060.76	-893.45	0.16	3.6	30	186	93	0	627.72	18	0	419.75	1649.47	1621.92	0.01	19.56	2.97	3.16
3.36	log-slicing_bvsdiv_17	20	502.64	442.92	1390.96	-948.04	0.32	1.2	24	92	53	80	0.12	0	0	1376.76	4150.79	4148.57	0.01	130.46	11.10	3.36
3.37	lfsr_lfsr_004_111_112	20	178.05	181.61	1141.95	-960.34	0.16	3.6	17	92	53	0	377.46	1	0	727.29	2630.16	2612.84	0.19	96.65	9.53	3.32
3.38	log-slicing_bvslt_5000	20	296.40	315.45	1354.53	-1039.08	0.23	2.6	17	107	60	0	0.49	1	0	1319.55	4768.28	4762.58	0.00	273.06	17.73	4.59
3.39	log-slicing_bvslt_5743	20	444.58	367.82	1435.73	-1067.91	0.26	1.8	14	80	47	0	0.49	1	0	1407.18	5223.87	5219.10	0.06	241.12	26.38	4.52
3.40	...smtcomp09_nt_bv_formula	20	156.72	175.28	1326.84	-1151.56	0.13	3.6	40	248	104	0	684.26	22	0	628.45	2352.33	2323.04	0.00	61.88	2.68	2.12
3.41	lfsr_lfsr_004_159_128	20	112.10	299.74	1502.37	-1202.63	0.20	3.8	16	75	34	0	508.08	1	0	930.74	3256.03	3233.29	0.12	117.89	14.80	4.64
3.42	lfsr_lfsr_008_031_064	20	oot	42.97	1403.15	-1360.18	0.03	3.2	36	200	120	0	489.46	1	0	888.04	3432.92	3406.89	0.15	37.88	4.63	3.35
4.1	pspace_power2sum.6097	20	388.12	1779.43	oot	-20.57	0.99	2.2	1	1	1	0	0.00	0	0	1620.82	1620.82	1620.40	1620.82	1620.82	1620.82	4.1
4.2	log-slicing_bvslt_15402	20	1048.34	1765.72	oot	-34.28	0.98	2.9	9	37	37	0	0.63	1	0	1384.27	6328.19	6324.02	8.34	218.96	61.44	4.2
4.3	...est_v7_r12_vr1_c1_s10576	20	oot	1756.98	oot	-43.02	0.98	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.3
4.4	pspace_power2sum.5699	20	347.92	1748.12	oot	-51.88	0.97	2.0	1	1	1	0	0.00	0	0	1333.58	1333.58	1333.19	1333.58	1333.58	1333.58	4.4
4.5	log-slicing_bvsub_05994	20	414.31	1726.25	oot	-73.75	0.96	0.1	1	1	1	0	0.00	0	0	6.60	6.60	4.61	6.60	6.60	6.60	4.5
4.6	core_ext_con_048_002_1024	20	487.72	1716.64	oot	-83.36	0.95	0.1	1	1	1	0	8.92	4	0	101.91	101.91	101.62	101.91	101.91	101.91	2.5
4.7	...test_v5_r15_vr5_c1_s8246	20	1106.64	1711.46	oot	-88.54	0.95	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.6
4.8	log-slicing_bvult_14973	20	1019.50	1697.30	oot	-102.70	0.94	3.1	10	38	38	0	0.35	0	0	1655.94	5997.06	5992.62	2.01	263.78	54.03	4.7
4.9	log-slicing_bvsdiv_18	20	1485.52	1599.03	oot	-200.97	0.89	1.2	15	70	52	33	0.12	0	0	1539.52	4133.00	4131.40	0.00	283.94	17.15	4.8
4.10	...test_v7_r7_vr5_c1_s19694	10	1370.11	1539.94	oot	-260.06	0.86	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.9
4.11	brummayerbiere3_minor256	20	116.13	321.33	sf	-305.52	0.51	3.7	9	30	18	0	66.72	0	0	235.31	535.99	498.41	0.11	82.50	5.89	4.58
4.12	...est_v7_r7_vr10_c1_s10625	10	852.76	1379.94	oot	-420.06	0.77	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.10
4.13	brummayerbiere3_maxand256	20	102.05	252.03	sf	-443.77	0.36	4.0	12	52	31	0	64.29	0	0	307.23	784.94	754.43	0.11	88.98	5.00	3.33
4.14	log-slicing_bvslt_12430	20	oot	1349.76	oot	-450.24	0.75	2.5	6	32	32	0	0.63	1	0	1495.08	6284.51	6281.13	13.72	226.95	108.35	4.11
4.15	...est_v5_r15_vr5_c1_s26657	20	oot	1282.84	oot	-517.16	0.71	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.12
4.16	log-slicing_bvslt_10944	20	505.68	1114.25	oot	-685.75	0.62	2.9	9	43	43	0	0.63	1	0	1518.50	6438.81	6434.90	0.12	208.53	65.04	4.13
4.17	...test_v7_r7_vr1_c1_s4574	10	609.33	1077.24	oot	-722.76	0.60	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.14
4.18	...st_v5_r10_vr10_c1_s21502	20	697.82	1056.47	oot	-743.53	0.59	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.15
4.19	...st_v5_r15_vr10_c1_s11127	10	346.03	1009.99	oot	-790.01	0.56	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.8
4.20	VS3_VS3-benchmark-A9	10	268.45	1009.13	oot	-790.87	0.56	3.3	45	290	144	0	1.12	1	0	1771.43	6798.16	6791.38	0.00	31.41	6.35	4.16
4.21	...est_v5_r10_vr1_c1_s32538	20	208.16	996.75	oot	-803.25	0.55	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.17
4.22	...est_v5_r10_vr5_c1_s13679	10	899.64	937.68	oot	-862.32	0.52	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.18
4.23	VS3_VS3-benchmark-A7	10	985.79	934.59	oot	-865.41	0.52	3.2	60	348	148	0	0.99	1	0	1631.68	6789.37	6781.33	0.00	23.79	4.68	4.19
4.24	log-slicing_bvslt_7972	20	305.25	931.67	oot	-868.33	0.52	3.2	13	73	64	0	0.49	1	0	1755.21	6705.61	6700.63	2.35	158.47	31.78	4.20
4.25	...servers_slapd_a_vc149923	10	325.64	886.77	oot	-913.23	0.49	3.3	30	67	67	0	0.16	1	0	1764.73	6928.03	6925.15	0.30	49.43	13.05	4.21
4.26	...est_v7_r7_vr10_c1_s32506	10	1754.95	864.50	oot	-935.50	0.48	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.22
4.27	log-slicing_bvslt_9458	20	539.16	833.97	oot	-966.03	0.46	2.9	9	44	44	0	0.63	1	0	1656.34	6188.65	6184.70	14.94	155.49	63.15	4.23
4.28	...test_v7_r7_vr1_c1_s24449	20	679.70	826.28	oot	-973.72	0.46	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.24
4.29	...servers_slapd_a_vc149922	10	325.68	788.47	oot	-1011.53	0.44	3.3	29	67	67	0	0.21	1	0	1706.47	6916.38	6913.54	0.30	51.02	13.17	4.25
4.30	...est_v5_r10_vr1_c1_s19145	10	316.49	780.82	oot	-1019.18	0.43	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.26
4.31	...yerbiere_countbitssrl032	20	387.06	713.56	oot	-1086.44	0.40	0.5						1	0		0.00	0.00	0.00	0.00	0.00	4.27
4.32	log-slicing_bvult_8991	20	542.61	709.90	oot	-1090.10	0.39	2.9	8	46	44	0	0.35	0	0	1711.19	6016.28	6012.53	0.18	171.65	73.37	4.28
4.33	log-slicing_bvult_7994	20	230.10	691.43	oot	-1108.57	0.38	3.3	10	64	64	0	0.27	0	0	1382.77	6679.86	6675.91	2.40	155.66	42.82	4.29
4.34	log-slicing_bvslt_6486	20	191.65	687.42	oot	-1112.58	0.38	2.9	16	112	60	0	0.49	1	0	1609.96	6031.95	6025.79	0.00	398.68	24.03	4.30
4.35	log-slicing_bvult_6997	20	201.19	669.29	oot	-1130.71	0.37	3.7	10	53	53	0	0.27	0	0	1051.28	6652.78	6649.30	7.80	108.65	41.84	4.31

Continued on next page

Table C.2 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector with FLR (Parallel-Incrmental-FLR)																		cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-inc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
4.36	...est_v5_r10_vr5_c1_s13195	10	337.55	662.60	oot	-1137.40	0.37	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.32
4.37	...1_alt_true.BV.c.cil.c.17	20	oot	621.75	oot	-1178.25	0.35	3.8	11	24	23	0	1126.54	26	0	468.51	2354.50	2334.71	0.12	105.97	18.54	4.33
4.38	log-slicing_bvult_6000	20	406.38	608.33	oot	-1191.67	0.34	3.0	16	98	71	0	0.27	0	0	1761.60	6646.68	6642.13	0.39	125.00	26.07	4.34
4.39	log-slicing_bvslt_8715	20	342.56	598.50	oot	-1201.50	0.33	2.9	9	45	45	0	0.63	1	0	1756.77	6395.25	6391.39	12.26	155.44	67.32	4.35
4.40	...test_v7_r7_vr1_c1_s22845	10	316.63	593.48	oot	-1206.52	0.33	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.36
4.41	lfsr_lfsr_008_127_112	20	176.00	591.63	oot	-1208.37	0.33	3.4	8	19	19	0	1435.97	1	0	277.79	1145.79	1133.81	1.50	81.22	12.59	4.37
4.42	log-slicing_bvslt_7229	20	217.01	590.38	oot	-1209.62	0.33	3.3	20	122	65	0	0.49	1	0	1538.19	5682.98	5675.72	0.00	95.94	17.82	4.38
4.43	...est_v5_r15_vr1_c1_s26845	20	619.09	589.84	oot	-1210.16	0.33	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.39
4.44	lfsr_lfsr_008_143_112	20	205.39	587.59	oot	-1212.41	0.33	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.40
4.45	...est_v5_r10_vr1_c1_s13516	10	231.36	574.53	oot	-1225.47	0.32	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.41
4.46	...tcomp09_stall_bv_formula	20	609.53	493.17	oot	-1306.83	0.27	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.42
4.47	float_mul_03_30_4	20	514.80	488.97	oot	-1311.03	0.27	3.6	70	431	146	0	93.99	28	0	1163.51	6328.64	6308.41	0.00	61.79	4.14	4.43
4.48	float_newton.5.3.i	20	415.28	486.80	oot	-1313.20	0.27	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.44
4.49	...test_v5_r10_vr5_c1_s8690	20	353.93	485.91	oot	-1314.09	0.27	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.45
4.50	float_newton.2.3.i	20	289.38	445.92	oot	-1354.08	0.25	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.46
4.51	float_newton.6.3.i	10	564.04	437.06	oot	-1362.94	0.24	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.47
4.52	lfsr_lfsr_008_143_128	20	261.79	428.53	oot	-1371.47	0.24	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.48
4.53	...test_v7_r12_vr5_c1_s8938	10	615.16	403.94	oot	-1396.06	0.22	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.30
4.54	VS3_VS3-benchmark-A6	10	111.93	397.09	oot	-1402.91	0.22	2.0	82	613	131	0	0.90	1	0	1737.46	6516.74	6506.09	0.00	21.93	3.55	4.49
4.55	...test_v7_r7_vr5_c1_s14675	10	364.55	393.97	oot	-1406.03	0.22	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.50
4.56	...._test_v7_r7_vr5_c1_s3582	10	326.78	392.56	oot	-1407.44	0.22	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	2.7
4.57	lfsr_lfsr_008_015_128	20	277.36	380.87	oot	-1419.13	0.21	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.51
4.58	lfsr_lfsr_008_095_128	20	148.89	366.34	oot	-1433.66	0.20	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.53
4.59	float_newton.1.3.i	20	385.34	360.03	oot	-1439.97	0.20	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.54
4.60	tacas07_Y86_btnft	20	135.77	186.21	oom	-1446.66	0.11	6.8									0.00	0.00	0.00	0.00	0.00	4.76
4.61	brummayerbiere3_minand256	20	289.66	328.30	oot	-1471.70	0.18	1.1	5	6	3	3	93.09	0	0	1660.23	1864.29	1859.19	0.83	833.02	133.16	4.55
4.62	lfsr_lfsr_008_111_112	20	148.60	324.78	oot	-1475.22	0.18	2.9	9	24	24	0	1424.61	1	0	320.88	1070.14	1061.57	5.51	55.54	11.26	4.56
4.63	float_newton.4.3.i	20	334.84	321.37	oot	-1478.63	0.18	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.57
4.64	float_newton.3.3.i	20	355.49	314.25	oot	-1485.75	0.17	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.60
4.65	lfsr_lfsr_008_159_080	20	194.33	310.35	oot	-1489.65	0.17	3.3	10	23	23	0	777.79	1	0	675.93	3518.54	3507.20	4.52	157.43	29.57	4.61
4.66	...yerbiere3_minandmaxor064	20	207.79	308.52	oot	-1491.48	0.17	4.0	61	110	14	0	25.39	0	0	1695.85	2457.54	2412.01	0.12	111.76	8.81	4.62
4.67	lfsr_lfsr_008_159_112	20	275.94	300.03	oot	-1499.97	0.17	3.6	7	16	16	0	1437.08	1	0	286.07	1066.86	1054.20	1.45	78.89	15.46	4.63
4.68	...test_v5_r5_vr10_c1_s7194	10	141.84	299.26	oot	-1500.74	0.17	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.31
4.69	lfsr_lfsr_008_143_096	20	173.94	291.37	oot	-1508.63	0.16	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.65
4.70	...test_v5_r15_vr1_c1_s8236	20	927.66	285.17	oot	-1514.83	0.16	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.7
4.71	lfsr_lfsr_008_015_112	20	388.44	270.57	oot	-1529.43	0.15	3.3	41	223	223	0	1406.57	1	0	366.19	1426.89	1364.81	0.11	29.17	1.31	4.66
4.72	...._test_v5_r5_vr5_c1_s2800	10	228.14	268.60	oot	-1531.40	0.15	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.67
4.73	lfsr_lfsr_004_143_128	20	116.21	259.85	oot	-1540.15	0.14	3.8	12	34	34	0	1496.57	1	0	215.80	1013.46	994.83	0.31	42.96	5.96	4.68
4.74	lfsr_lfsr_008_079_128	20	107.76	256.88	oot	-1543.12	0.14	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.69
4.75	lfsr_lfsr_008_095_096	20	105.52	255.91	oot	-1544.09	0.14	3.4	13	44	44	0	1035.30	1	0	702.72	2728.32	2716.47	3.67	66.13	14.28	4.70
4.76	...est_v7_r7_vr10_c1_s24535	10	237.88	246.30	oot	-1553.70	0.14	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.10
4.77	lfsr_lfsr_008_111_096	20	133.32	238.72	oot	-1561.28	0.13	3.3	11	28	28	0	1005.36	1	0	654.10	2568.24	2555.07	4.48	68.24	17.96	4.71
4.78	brummayerbiere3_maxor064	20	209.14	224.27	oot	-1575.73	0.12	1.9	68	69	3	43	5.98	0	0	1556.42	1701.65	1682.68	0.17	370.99	9.45	4.72
4.79	...est_v5_r15_vr1_c1_s32559	20	163.75	217.60	oot	-1582.40	0.12	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.34
4.80	...._smtcomp09_sw_bv_formula	20	202.42	210.08	oot	-1589.92	0.12	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.73
4.81	...._smtcomp09_lf_bv_formula	20	180.24	207.71	oot	-1592.29	0.12	3.6	33	203	108	0	1076.28	53	0	700.61	2723.99	2701.16	0.00	58.71	4.02	2.10
4.82	lfsr_lfsr_008_127_128	20	212.80	198.44	oot	-1601.56	0.11	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.74
4.83	float_newton.5.2.i	20	249.53	196.37	oot	-1603.63	0.11	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.37
4.84	tacas07_Y86_std	20	129.65	193.97	oot	-1606.03	0.11	0.9	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.27
4.85	lfsr_lfsr_008_143_064	20	103.49	192.94	oot	-1607.06	0.11	3.7	11	35	35	0	1474.98	1	0	291.22	1002.28	986.67	1.94	27.06	6.86	4.75
4.86	...test_v5_r5_vr5_c1_s24018	10	182.81	186.53	oot	-1613.47	0.10	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.9

Continued on next page

Table C.2 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector with FLR (Parallel-Incrmental-FLR)																	cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-inc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
4.87	float_newton.2.2.i	20	143.18	178.62	oot	-1621.38	0.10	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.77
4.88	lfsr_lfsr_008_063_128	20	oot	178.50	oot	-1621.50	0.10	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.78
4.89	float_newton.1.2.i	20	139.98	173.63	oot	-1626.37	0.10	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.79
4.90	float_newton.4.2.i	20	143.25	155.94	oot	-1644.06	0.09	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.80
4.91	float_newton.3.2.i	20	186.56	147.77	oot	-1652.23	0.08	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.81
4.92	...test_v5_r5_vr1_c1_s14623	10	143.20	147.57	oot	-1652.43	0.08	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.24
4.93	lfsr_lfsr_008_159_096	20	275.86	145.85	oot	-1654.15	0.08	3.1	9	20	20	0	1014.96	1	0	489.12	2713.37	2705.90	9.72	81.02	25.36	4.82
4.94	lfsr_lfsr_008_063_096	20	oot	144.83	oot	-1655.17	0.08	3.4	20	74	58	0	1041.15	1	0	630.01	2716.16	2697.55	0.38	45.27	7.97	3.38
4.95	lfsr_lfsr_008_143_080	20	112.13	144.75	oot	-1655.25	0.08	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.83
4.96	lfsr_lfsr_008_095_112	20	129.08	142.47	oot	-1657.53	0.08	3.3	10	28	28	0	1429.94	1	0	307.45	1090.82	1078.81	3.55	29.62	9.65	4.84
4.97	lfsr_lfsr_008_127_096	20	113.04	135.23	oot	-1664.77	0.08	3.1	16	48	48	0	1037.72	1	0	675.93	2675.96	2663.14	1.95	47.90	11.89	4.85
4.98	brummayerbiere3_maxxor032	20	143.95	127.04	oot	-1672.96	0.07	1.7	70	481	156	0	0.79	0	0	1782.86	6443.20	6426.32	0.01	39.17	4.87	4.86
4.99	lfsr_lfsr_008_015_096	20	272.75	123.59	oot	-1676.41	0.07	3.4	30	131	131	0	1020.62	1	0	249.68	2996.40	2957.90	0.24	34.16	4.16	4.87
4.100	brummayerbiere_countbits064	20	129.34	121.61	oot	-1678.39	0.07	3.2						1	0		0.00	0.00	0.00	0.00	0.00	4.88
4.101	lfsr_lfsr_008_111_128	20	160.68	112.32	oot	-1687.68	0.06	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.89
4.102	lfsr_lfsr_008_047_096	20	160.40	99.36	oot	-1700.64	0.06	3.6	24	107	85	0	1033.57	1	0	725.14	2818.79	2792.74	0.13	81.89	6.51	4.90
4.103	brummayerbiere3_minxor064	20	111.84	93.35	oot	-1706.65	0.05	1.2	76	438	78	0	4.46	0	0	1769.34	5288.79	5249.05	0.01	45.86	5.25	4.91
4.104	lfsr_lfsr_008_159_064	20	149.52	83.48	oot	-1716.52	0.05	3.3	12	37	29	0	489.06	1	0	1099.63	4780.82	4772.79	2.19	130.23	28.97	4.92
4.105	lfsr_lfsr_008_159_048	20	100.56	44.32	oot	-1755.68	0.02	3.5	14	57	41	0	267.69	1	0	1282.55	5763.90	5754.69	4.31	113.71	27.71	4.93
4.106	float_mult2.c.50	10	109.39	40.02	oot	-1759.98	0.02	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.11
5.1	...erbiere4_unconstrained06	0	oom	oom	oom	0.88	1.02	6.8	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.199
5.2	...erbiere4_unconstrained07	0	oom	oom	oom	0.70	1.01	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.4
5.3	...erbiere4_unconstrained01	0	oom	oom	oom	0.48	1.01	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.3
5.4	...erbiere4_unconstrained04	0	oom	oom	oom	0.18	1.00	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.197
5.5	core_ext_con_052_002_1024	20	194.60	oot	oot	0.00	1.00	0.1	1	1	1	0	10.44	4	0	112.83	112.83	112.48	112.83	112.83	112.83	1.6
5.6	core_ext_con_056_002_1024	20	279.98	oot	oot	0.00	1.00	0.1	1	1	1	0	11.99	4	0	122.15	122.15	121.78	122.15	122.15	122.15	1.4
5.7	core_ext_con_060_001_1024	20	110.60	oot	oot	0.00	1.00	0.1	1	1	1	0	1.02	2	0	140.88	140.88	140.49	140.88	140.88	140.88	5.7
5.8	core_ext_con_060_002_1024	20	220.71	oot	oot	0.00	1.00	0.1	1	1	1	0	13.75	4	0	130.03	130.03	129.64	130.03	130.03	130.03	1.5
5.9	core_ext_con_060_256_1024	20	101.73	oot	oot	0.00	1.00	0.2	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.8
5.10	core_ext_con_064_001_1024	20	128.25	oot	oot	0.00	1.00	0.1	1	1	1	0	1.16	2	0	150.15	150.15	149.73	150.15	150.15	150.15	5.9
5.11	core_ext_con_064_002_1024	20	344.29	oot	oot	0.00	1.00	0.1	1	1	1	0	15.65	4	0	145.73	145.73	145.31	145.73	145.73	145.73	1.8
5.12	fft_Sz1024_34824	0	oot	oot	oot	0.00	1.00	3.3	40	216	216	0	118.21	0	0	483.43	6561.05	6528.19	0.26	38.21	5.93	5.10
5.13	fft_Sz512_15128_0	0	oot	oot	oot	0.00	1.00	3.1	70	413	413	0	22.61	0	0	471.55	7022.92	6986.38	0.14	16.97	2.48	5.11
5.14	fft_Sz512_15128_1	0	oot	oot	oot	0.00	1.00	3.2	70	411	411	0	22.63	0	0	565.83	7025.48	6989.44	0.15	17.17	2.50	5.12
5.15	fft_Sz512_15128_2	0	oot	oot	oot	0.00	1.00	3.3	70	428	428	0	21.31	1	0	561.13	7049.67	7018.92	0.17	16.67	2.48	5.13
5.16	fft_Sz512_15128_4	0	oot	oot	oot	0.00	1.00	3.1	70	416	416	0	22.62	0	0	477.87	7040.11	7003.72	0.15	17.62	2.49	5.14
5.17	fft_Sz512_15128_5	0	oot	oot	oot	0.00	1.00	3.2	70	413	413	0	22.61	0	0	577.64	7017.30	6981.04	0.15	17.19	2.50	5.15
5.18	fft_Sz512_15128_6	0	oot	oot	oot	0.00	1.00	3.1	70	434	434	0	21.43	1	0	454.08	7050.96	7020.23	0.14	16.65	2.46	5.16
5.19	...st_v5_r10_vr10_c1_s15708	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.17
5.20	...est_v5_r10_vr10_c1_s7608	20	1199.14	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.2
5.21	...est_v5_r15_vr5_c1_s23844	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.18
5.22	...st_v7_r12_vr10_c1_s15994	10	1367.71	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.19
5.23	...._test_v7_r12_vr1_c1_s703	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.1
5.24	...est_v7_r12_vr5_c1_s14336	10	933.17	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.20
5.25	...est_v7_r12_vr5_c1_s29826	10	896.50	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.21
5.26	...est_v7_r17_vr1_c1_s23882	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.22
5.27	...est_v7_r17_vr1_c1_s30331	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.23
5.28	...fyInterleavedModMult-128	0	oot	oot	oot	0.00	1.00	4.1	12	8	8	0	18.55	2	0	1592.91	5654.84	5650.22	0.84	388.52	74.41	5.24
5.29	...ffyInterleavedModMult-32	0	oot	oot	oot	0.00	1.00	3.3	29	116	116	0	1.14	2	0	1752.95	6483.81	6480.36	0.22	131.09	11.52	5.25
5.30	lfsr_lfsr_008_159_128	20	285.74	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.26

Continued on next page

Table C.2 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector with FLR (Parallel-Incrmental-FLR)																		cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-inc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.31	log-slicing_bvadd_18209	20	101.26	oot	oot	0.00	1.00	1.2	1	1	1	0	0.00	0	0	68.34	68.34	63.39	68.34	68.34	68.34	5.27
5.32	log-slicing_bvadd_19096	20	109.78	oot	oot	0.00	1.00	1.2	1	1	1	0	0.00	0	0	75.37	75.37	69.13	75.37	75.37	75.37	5.28
5.33	log-slicing_bvadd_19983	20	121.59	oot	oot	0.00	1.00	1.2	1	1	1	0	0.00	0	0	81.32	81.32	75.21	81.32	81.32	81.32	5.29
5.34	log-slicing_bvadd_20870	20	132.00	oot	oot	0.00	1.00	1.2	1	1	1	0	0.00	0	0	89.14	89.14	82.31	89.14	89.14	89.14	5.30
5.35	log-slicing_bvadd_21757	20	165.07	oot	oot	0.00	1.00	1.4	1	1	1	0	0.00	0	0	96.64	96.64	88.58	96.64	96.64	96.64	5.31
5.36	log-slicing_bvadd_22644	20	156.03	oot	oot	0.00	1.00	1.4	1	1	1	0	0.00	0	0	104.19	104.19	94.42	104.19	104.19	104.19	5.32
5.37	log-slicing_bvadd_23531	20	167.45	oot	oot	0.00	1.00	1.4	1	1	1	0	0.00	0	0	109.52	109.52	99.06	109.52	109.52	109.52	5.33
5.38	log-slicing_bvadd_24418	20	198.09	oot	oot	0.00	1.00	1.4	1	1	1	0	0.00	0	0	120.79	120.79	107.53	120.79	120.79	120.79	5.34
5.39	log-slicing_bvadd_25305	20	186.34	oot	oot	0.00	1.00	1.4	1	1	1	0	0.00	0	0	125.12	125.12	111.17	125.12	125.12	125.12	5.35
5.40	log-slicing_bvadd_26192	20	202.52	oot	oot	0.00	1.00	1.6	1	1	1	0	0.00	0	0	131.63	131.63	117.88	131.63	131.63	131.63	5.36
5.41	log-slicing_bvadd_27079	20	216.41	oot	oot	0.00	1.00	1.6	1	1	1	0	0.00	0	0	140.73	140.73	122.60	140.73	140.73	140.73	5.37
5.42	log-slicing_bvadd_27966	20	232.01	oot	oot	0.00	1.00	1.7	1	1	1	0	0.00	0	0	147.99	147.99	128.61	147.99	147.99	147.99	5.38
5.43	log-slicing_bvadd_28853	20	245.33	oot	oot	0.00	1.00	1.7	1	1	1	0	0.00	0	0	154.67	154.67	132.25	154.67	154.67	154.67	5.39
5.44	log-slicing_bvadd_29740	20	264.93	oot	oot	0.00	1.00	1.7	1	1	1	0	0.00	0	0	160.66	160.66	136.68	160.66	160.66	160.66	5.40
5.45	log-slicing_bvmul_13	20	428.63	oot	oot	0.00	1.00	0.1	2	2	2	1	0.00	0	0	0.90	1742.21	1742.18	0.27	1741.04	435.55	5.41
5.46	log-slicing_bvmul_14	20	1477.50	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0	0	0.89	1.19	1.16	0.29	0.56	0.40	5.42
5.47	log-slicing_bvmul_15	0	oot	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0	0	0.93	1.25	1.23	0.32	0.58	0.42	5.43
5.48	log-slicing_bvmul_16	0	oot	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0	0	0.85	1.16	1.13	0.30	0.54	0.39	5.44
5.49	log-slicing_bvmul_17	0	oot	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0	0	0.92	1.23	1.11	0.31	0.57	0.41	5.45
5.50	log-slicing_bvmul_18	0	oot	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0	0	0.94	1.21	1.10	0.27	0.57	0.40	5.46
5.51	log-slicing_bvsdiv_19	0	oot	oot	oot	0.00	1.00	1.3	14	70	53	19	0.12	0	0	1496.56	3763.14	3761.61	0.00	742.88	17.67	5.47
5.52	log-slicing_bvsdiv_20	0	oot	oot	oot	0.00	1.00	1.3	13	68	54	14	0.12	0	0	1428.95	2759.09	2757.47	0.01	776.43	13.73	5.48
5.53	log-slicing_bvsdiv_21	0	oot	oot	oot	0.00	1.00	1.4	10	61	57	1	0.12	0	0	10.90	78.72	77.31	0.00	25.27	0.47	5.49
5.54	log-slicing_bvsdiv_22	0	oot	oot	oot	0.00	1.00	1.4	10	55	53	2	0.12	0	0	18.05	135.49	134.07	0.00	66.16	0.91	5.50
5.55	log-slicing_bvsdiv_23	0	oot	oot	oot	0.00	1.00	1.5	10	58	56	3	0.12	0	0	11.73	473.67	472.25	0.00	382.94	2.89	5.51
5.56	log-slicing_bvsdiv_24	0	oot	oot	oot	0.00	1.00	1.7	11	64	64	0	0.12	0	0	18.93	68.85	67.30	0.01	1.16	0.39	5.52
5.57	log-slicing_bvsdiv_25	0	oot	oot	oot	0.00	1.00	1.5	10	58	57	3	0.12	0	0	12.30	1572.37	1570.78	0.00	553.92	9.53	5.53
5.58	log-slicing_bvslt_10201	20	552.92	oot	oot	0.00	1.00	2.9	9	44	44	0	0.63	1	0	1528.03	6488.65	6484.72	0.14	224.61	65.54	5.54
5.59	log-slicing_bvslt_11687	20	665.86	oot	oot	0.00	1.00	2.9	9	42	42	0	0.63	1	0	1548.50	6025.56	6021.59	10.16	217.20	62.12	5.55
5.60	log-slicing_bvslt_13173	20	801.46	oot	oot	0.00	1.00	2.9	9	40	40	0	0.63	1	0	1500.06	6249.72	6245.81	0.57	206.73	63.13	5.56
5.61	log-slicing_bvslt_13916	20	908.99	oot	oot	0.00	1.00	2.9	9	39	39	0	0.63	1	0	1573.70	6495.99	6492.00	4.39	159.74	63.69	5.57
5.62	log-slicing_bvslt_14659	20	838.99	oot	oot	0.00	1.00	3.0	9	38	38	0	0.63	1	0	1646.88	6673.92	6669.89	11.23	174.62	66.08	5.58
5.63	log-slicing_bvslt_16145	20	1188.56	oot	oot	0.00	1.00	3.1	10	36	36	0	0.63	1	0	1612.58	5341.81	5337.42	0.11	512.02	48.12	5.59
5.64	log-slicing_bvslt_16888	20	1502.56	oot	oot	0.00	1.00	2.9	5	21	21	0	0.82	1	0	1508.04	5333.28	5326.56	30.87	339.71	156.86	5.60
5.65	log-slicing_bvslt_17631	20	1750.56	oot	oot	0.00	1.00	2.6	4	16	16	0	0.81	1	0	927.88	5521.81	5516.00	49.97	382.51	197.21	5.61
5.66	log-slicing_bvslt_18374	0	oot	oot	oot	0.00	1.00	1.4	3	8	8	0	0.81	1	0	1027.77	5530.11	5525.70	123.15	658.79	425.39	5.62
5.67	log-slicing_bvslt_19117	0	oot	oot	oot	0.00	1.00	3.0	6	22	22	0	0.81	1	0	1627.82	6481.35	6474.78	48.06	226.51	127.09	5.63
5.68	log-slicing_bvslt_19860	0	oot	oot	oot	0.00	1.00	3.0	6	21	21	0	0.82	1	0	1740.49	6593.64	6587.31	40.86	293.54	140.29	5.64
5.69	log-slicing_bvsmod_16	0	oot	oot	oot	0.00	1.00	0.5	26	122	57	66	0.10	0	0	1485.74	3634.67	3633.46	0.00	230.19	8.17	5.65
5.70	log-slicing_bvsmod_17	0	oot	oot	oot	0.00	1.00	1.3	17	82	55	34	0.15	0	0	1656.78	3763.70	3761.82	0.00	594.87	13.94	5.66
5.71	log-slicing_bvsmod_18	0	oot	oot	oot	0.00	1.00	1.4	13	70	60	9	0.15	0	0	1533.46	2236.54	2235.01	0.00	1032.95	10.91	5.67
5.72	log-slicing_bvsmod_19	0	oot	oot	oot	0.00	1.00	1.5	12	68	62	11	0.15	0	0	143.32	2494.44	2492.86	0.00	878.40	12.23	5.68
5.73	log-slicing_bvsmod_20	0	oot	oot	oot	0.00	1.00	1.4	10	58	56	4	0.15	0	0	850.38	1380.74	1379.25	0.00	840.99	8.37	5.69
5.74	log-slicing_bvsrem_16	0	oot	oot	oot	0.00	1.00	0.4	20	83	48	55	0.08	0	0	1744.98	4140.75	4139.95	0.00	349.17	13.40	5.70
5.75	log-slicing_bvsrem_17	0	oot	oot	oot	0.00	1.00	1.0	16	69	46	39	0.11	0	0	1694.77	4564.37	4562.78	0.00	547.46	19.10	5.71
5.76	log-slicing_bvsrem_18	0	oot	oot	oot	0.00	1.00	1.2	12	59	52	15	0.11	0	0	1418.83	3814.76	3813.45	0.00	804.19	20.73	5.72
5.77	log-slicing_bvsrem_19	0	oot	oot	oot	0.00	1.00	1.4	10	56	51	7	0.11	0	0	11.97	4839.04	4837.78	0.01	1188.59	30.06	5.73
5.78	log-slicing_bvsrem_20	0	oot	oot	oot	0.00	1.00	1.1	9	54	50	1	0.11	0	0	8.37	689.31	688.24	0.00	657.55	6.27	5.74
5.79	log-slicing_bvsub_06991	20	307.68	oot	oot	0.00	1.00	0.2	1	1	1	0	0.00	0	0	7.35	7.35	4.69	7.35	7.35	7.35	5.75
5.80	log-slicing_bvsub_07988	20	321.69	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0	0	8.31	8.31	4.86	8.31	8.31	8.31	5.76
5.81	log-slicing_bvsub_08985	20	378.56	oot	oot	0.00	1.00	0.1	1	1	1	0	0.00	0	0	9.19	9.19	4.92	9.19	9.19	9.19	5.77

Continued on next page

Table C.2 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector with FLR (Parallel-Incrmental-FLR)																	cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-inc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.82	log-slicing_bvsub_09982	20	328.06	oot	oot	0.00	1.00	0.2	1	1	1	0	0.00	0	0	10.18	10.18	5.13	10.18	10.18	10.18	5.78
5.83	log-slicing_bvsub_10979	20	161.21	oot	oot	0.00	1.00	0.1	1	1	1	0	0.00	0	0	11.22	11.22	5.17	11.22	11.22	11.22	5.79
5.84	log-slicing_bvsub_11976	20	664.63	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	12.77	12.77	5.35	12.77	12.77	12.77	5.80
5.85	log-slicing_bvsub_12973	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	13.94	13.94	5.36	13.94	13.94	13.94	5.81
5.86	log-slicing_bvsub_13970	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	15.06	15.06	5.61	15.06	15.06	15.06	5.82
5.87	log-slicing_bvsub_14967	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	16.76	16.76	5.57	16.76	16.76	16.76	5.83
5.88	log-slicing_bvsub_15964	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	18.77	18.77	5.74	18.77	18.77	18.77	5.84
5.89	log-slicing_bvsub_16961	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	20.43	20.43	5.84	20.43	20.43	20.43	5.85
5.90	log-slicing_bvsub_17958	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0	0	15.94	15.94	0.19	15.94	15.94	15.94	5.86
5.91	log-slicing_bvsub_18955	0	oot	oot	oot	0.00	1.00	0.6	1	1	1	0	0.00	0	0	17.91	17.91	0.20	17.91	17.91	17.91	5.87
5.92	log-slicing_bvsub_19952	0	oot	oot	oot	0.00	1.00	0.6	1	1	1	0	0.00	0	0	19.61	19.61	0.21	19.61	19.61	19.61	5.88
5.93	log-slicing_bvsub_20949	0	oot	oot	oot	0.00	1.00	0.6	1	1	1	0	0.00	0	0	23.06	23.06	0.22	23.06	23.06	23.06	5.89
5.94	log-slicing_bvsub_21946	0	oot	oot	oot	0.00	1.00	0.7	1	1	1	0	0.00	0	0	24.48	24.48	0.23	24.48	24.48	24.48	5.90
5.95	log-slicing_bvsub_22943	0	oot	oot	oot	0.00	1.00	0.7	1	1	1	0	0.00	0	0	27.98	27.98	0.24	27.98	27.98	27.98	5.91
5.96	log-slicing_bvsub_23940	0	oot	oot	oot	0.00	1.00	0.8	1	1	1	0	0.00	0	0	32.77	32.77	0.25	32.77	32.77	32.77	5.92
5.97	log-slicing_bvudiv_19	0	oot	oot	oot	0.00	1.00	0.8	7	29	28	10	0.08	0	0	376.14	3630.26	3629.23	0.00	1346.78	46.54	5.93
5.98	log-slicing_bvudiv_20	0	oot	oot	oot	0.00	1.00	0.9	8	32	30	9	0.08	0	0	1150.81	3231.64	3230.52	0.00	1130.18	36.31	5.94
5.99	log-slicing_bvudiv_21	0	oot	oot	oot	0.00	1.00	1.0	7	32	30	5	0.08	0	0	5.94	497.91	496.81	0.00	410.65	6.55	5.95
5.100	log-slicing_bvudiv_22	0	oot	oot	oot	0.00	1.00	1.0	7	33	32	4	0.08	0	0	6.23	1876.56	1875.40	0.00	1642.64	24.69	5.96
5.101	log-slicing_bvudiv_23	0	oot	oot	oot	0.00	1.00	1.0	7	33	32	4	0.08	0	0	6.35	786.16	785.05	0.01	677.08	10.34	5.97
5.102	log-slicing_bvudiv_24	0	oot	oot	oot	0.00	1.00	1.0	6	28	26	0	0.08	0	0	5.55	17.75	16.63	0.05	1.32	0.33	5.98
5.103	log-slicing_bvudiv_25	0	oot	oot	oot	0.00	1.00	1.2	7	36	32	1	0.08	0	0	6.72	1553.89	1552.57	0.02	1529.82	20.45	5.99
5.104	log-slicing_bvult_10985	20	556.62	oot	oot	0.00	1.00	2.7	7	40	40	0	0.35	0	0	1554.54	6317.68	6313.77	3.67	222.39	88.98	5.100
5.105	log-slicing_bvult_11982	20	518.16	oot	oot	0.00	1.00	2.9	7	34	34	0	0.35	0	0	1521.64	6261.33	6257.56	1.19	172.83	83.48	5.101
5.106	log-slicing_bvult_12979	20	893.49	oot	oot	0.00	1.00	2.9	9	40	40	0	0.35	0	0	1753.03	6397.03	6392.89	2.20	197.44	67.34	5.102
5.107	log-slicing_bvult_13976	20	893.73	oot	oot	0.00	1.00	2.9	8	39	39	0	0.35	0	0	1718.64	6555.12	6550.90	2.91	178.33	78.04	5.103
5.108	log-slicing_bvult_15970	20	1070.63	oot	oot	0.00	1.00	3.1	10	36	36	0	0.35	0	0	1350.62	4872.24	4867.54	0.24	194.40	43.12	5.104
5.109	log-slicing_bvult_16967	0	oot	oot	oot	0.00	1.00	2.9	5	21	21	0	0.46	0	0	1378.12	5835.06	5829.14	52.95	259.36	145.88	5.105
5.110	log-slicing_bvult_17964	0	oot	oot	oot	0.00	1.00	2.9	5	21	21	0	0.45	0	0	1261.34	5638.19	5632.25	39.77	253.37	131.12	5.106
5.111	log-slicing_bvult_18961	0	oot	oot	oot	0.00	1.00	2.4	4	16	16	0	0.45	0	0	1537.17	5454.02	5449.52	73.43	508.28	287.05	5.107
5.112	log-slicing_bvult_19958	0	oot	oot	oot	0.00	1.00	3.0	8	22	22	0	0.44	0	0	1730.88	5664.39	5658.48	0.18	481.28	71.70	5.108
5.113	log-slicing_bvult_20955	0	oot	oot	oot	0.00	1.00	3.0	7	21	21	0	0.45	0	0	1187.21	5162.18	5156.41	10.79	520.68	75.91	5.109
5.114	log-slicing_bvult_21952	0	oot	oot	oot	0.00	1.00	3.0	8	21	21	0	0.45	0	0	1502.00	5927.46	5921.79	3.26	470.02	73.18	5.110
5.115	log-slicing_bvult_22949	0	oot	oot	oot	0.00	1.00	3.0	8	21	21	0	0.45	0	0	1009.12	5921.76	5915.91	12.71	757.10	70.50	5.111
5.116	log-slicing_bvult_23946	0	oot	oot	oot	0.00	1.00	2.9	7	20	20	0	0.44	0	0	662.13	6185.75	6179.98	0.26	666.12	79.30	5.112
5.117	log-slicing_bvult_24943	0	oot	oot	oot	0.00	1.00	3.0	8	19	19	0	0.44	0	0	1105.45	4451.90	4446.27	0.19	413.65	54.96	5.113
5.118	log-slicing_bvult_25940	0	oot	oot	oot	0.00	1.00	3.0	8	19	19	0	0.45	0	0	705.07	3550.83	3545.27	0.21	700.46	42.78	5.114
5.119	log-slicing_bvult_9988	20	555.03	oot	oot	0.00	1.00	2.9	8	44	44	0	0.35	0	0	1623.91	6332.17	6328.20	4.76	205.96	71.15	5.115
5.120	log-slicing_bvurem_17	0	oot	oot	oot	0.00	1.00	0.5	8	30	24	16	0.08	0	0	892.31	3711.96	3711.31	0.00	773.35	40.79	5.116
5.121	log-slicing_bvurem_18	0	oot	oot	oot	0.00	1.00	0.7	7	29	29	7	0.08	0	0	1109.75	2544.06	2543.47	0.00	1105.64	35.83	5.117
5.122	log-slicing_bvurem_19	0	oot	oot	oot	0.00	1.00	0.6	6	29	27	2	0.08	0	0	4.02	443.69	443.11	0.00	393.76	7.16	5.118
5.123	log-slicing_bvurem_20	0	oot	oot	oot	0.00	1.00	0.8	7	32	30	1	0.08	0	0	4.90	426.81	426.21	0.00	411.88	6.57	5.119
5.124	log-slicing_bvurem_21	0	oot	oot	oot	0.00	1.00	0.8	6	30	30	0	0.08	0	0	4.25	15.34	14.70	0.16	0.92	0.25	5.120
5.125	log-slicing_bvurem_22	0	oot	oot	oot	0.00	1.00	0.8	7	32	32	0	0.08	0	0	5.47	17.13	16.47	0.11	0.96	0.27	5.121
5.126	pipe_pipe-noabs.atlas.qf_bv	20	1290.10	oot	oot	0.00	1.00	0.7	56	419	45	0	1.08	1	0	1769.38	6305.12	6292.68	0.00	59.43	6.70	5.122
5.127	pspace_power2sum.5500	20	496.16	oot	oot	0.00	1.00	2.0	1	1	1	0	0.00	0	0	1208.09	1208.09	1207.72	1208.09	1208.09	1208.09	5.123
5.128	pspace_power2sum.5898	20	343.57	oot	oot	0.00	1.00	2.2	1	1	1	0	0.00	0	0	1471.22	1471.22	1470.83	1471.22	1471.22	1471.22	5.124
5.129	pspace_power2sum.6296	20	434.76	oot	oot	0.00	1.00	2.2	1	1	1	0	0.00	0	0	1780.20	1780.20	1779.76	1780.20	1780.20	1780.20	5.125
5.130	pspace_power2sum.6495	20	504.51	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.126
5.131	pspace_power2sum.6694	20	1005.81	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.127
5.132	pspace_power2sum.6893	20	1102.02	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.128

Continued on next page

Table C.2 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector with FLR (Parallel-Incrmental-FLR)																	cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-inc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.133	pspace_power2sum.7092	20	452.50	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.129
5.134	pspace_power2sum.7291	20	971.68	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.130
5.135	pspace_power2sum.7490	20	505.69	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.131
5.136	pspace_power2sum.7689	20	515.93	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.132
5.137	pspace_power2sum.7888	20	1145.49	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.133
5.138	pspace_power2sum.8087	0	oot	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.134
5.139	pspace_power2sum.8286	20	898.41	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.135
5.140	pspace_power2sum.8485	20	872.30	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.136
5.141	pspace_power2sum.8684	20	1028.30	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.137
5.142	pspace_power2sum.8883	20	1035.88	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.138
5.143	pspace_power2sum.9082	20	1301.91	oot	oot	0.00	1.00	2.1	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.139
5.144	pspace_power2sum.9281	20	1338.53	oot	oot	0.00	1.00	2.1	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.140
5.145	pspace_power2sum.9480	20	751.83	oot	oot	0.00	1.00	2.1	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.141
5.146	pspace_power2sum.9679	0	oot	oot	oot	0.00	1.00	2.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.142
5.147	pspace_power2sum.9878	20	573.35	oot	oot	0.00	1.00	2.1	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.143
5.148	RWS_Example_15.txt	0	oot	oot	oot	0.00	1.00	0.3	0	1	1	0	7.77	1	0	0.00	0.00	0.00	0.00	0.00	0.00	5.144
5.149	RWS_Example_19.txt	0	oot	oot	oot	0.00	1.00	0.3	0	1	1	0	35.51	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.145
5.150	RWS_Example_20.txt	0	oot	oot	oot	0.00	1.00	0.4	0	1	1	0	70.48	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.146
5.151	RWS_Example_7.txt	0	oot	oot	oot	0.00	1.00	0.1	0	1	1	0	1.70	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.147
5.152	uum_uum12	0	oot	oot	oot	0.00	1.00	0.6						1	0		0.00	0.00	0.00	0.00	0.00	5.148
5.153	uum_uum16	0	oot	oot	oot	0.00	1.00	1.5	50	368	238	0	2.12	1	0	1321.00	6864.70	6858.39	0.00	34.63	4.98	5.149
5.154	uum_uum20	0	oot	oot	oot	0.00	1.00	3.1	60	418	342	0	4.41	1	0	783.64	6895.97	6883.78	0.00	38.68	3.44	5.150
5.155	uum_uum24	0	oot	oot	oot	0.00	1.00	3.4	50	250	248	0	6.35	1	0	1308.58	6816.27	6804.58	0.00	49.82	4.81	5.151
5.156	uum_uum28	0	oot	oot	oot	0.00	1.00	3.4						1	0		0.00	0.00	0.00	0.00	0.00	5.152
5.157	uum_uum32	0	oot	oot	oot	0.00	1.00	3.1	35	167	167	0	10.23	1	0	1776.31	6589.61	6579.04	0.30	102.93	7.86	5.153
5.158	VS3_VS3-benchmark-A10	0	oot	oot	oot	0.00	1.00	3.5	45	207	207	0	1.17	1	0	1765.98	7004.51	6998.89	0.18	44.64	5.99	5.154
5.159	VS3_VS3-benchmark-A11	0	oot	oot	oot	0.00	1.00	3.5	47	213	211	0	1.11	1	0	1769.25	6964.71	6959.21	0.15	44.55	5.80	5.155
5.160	VS3_VS3-benchmark-A12	0	oot	oot	oot	0.00	1.00	3.3	34	176	176	0	1.19	1	0	1783.44	7073.00	7069.16	0.38	43.64	8.85	5.156
5.161	VS3_VS3-benchmark-A13	0	oot	oot	oot	0.00	1.00	3.2	30	111	111	0	1.57	1	0	638.84	7007.29	7003.31	0.35	53.41	10.46	5.157
5.162	VS3_VS3-benchmark-A8	0	oot	oot	oot	0.00	1.00	3.3	33	145	145	0	1.32	1	0	1782.58	7011.37	7007.78	0.37	45.38	9.46	5.158
5.163	VS3_VS3-benchmark-S2	0	oot	oot	oot	0.00	1.00	3.5	20	42	42	0	1.37	1	0	1745.30	6793.88	6791.36	1.83	110.27	22.95	5.159
5.164	...1_alt_true.BV.c.cil.c.21	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.160
5.165	brummayerbiere2_smulov1bw48	0	oot	oot	oot	0.00	1.00	1.3	48	311	67	40	0.86	0	0	1770.43	5466.74	5459.05	0.00	104.95	6.61	5.161
5.166	...erbiere3_isqrtaddeqcheck	0	oot	oot	oot	0.00	1.00	3.3	60	413	248	13	0.19	0	0	731.44	6722.55	6703.07	0.00	94.68	3.85	5.162
5.167	...mayerbiere3_isqrtaddnoif	0	oot	oot	oot	0.00	1.00	1.9	30	217	110	0	0.03	0	0	1353.78	6678.39	6669.46	0.02	166.00	11.11	5.163
5.168	...mayerbiere3_isqrteqcheck	0	oot	oot	oot	0.00	1.00	3.1	50	337	228	0	0.19	0	0	1101.94	6614.95	6594.47	0.00	99.12	4.69	5.164
5.169	brummayerbiere3_isqrtnoif	0	oot	oot	oot	0.00	1.00	2.8	29	202	122	0	0.03	0	0	1717.05	6493.70	6483.70	0.00	185.55	11.72	5.165
5.170	brummayerbiere3_maxor128	0	oot	oot	oot	0.00	1.00	0.9	10	11	3	8	24.04	0	0	1563.00	1694.92	1689.37	1.17	266.04	58.45	5.166
5.171	brummayerbiere3_maxor256	0	oot	oot	oot	0.00	1.00	4.3	13	8	3	1	94.00	0	0	1296.51	2667.51	2644.78	0.71	416.14	80.83	5.167
5.172	brummayerbiere3_maxxor064	0	oot	oot	oot	0.00	1.00	3.9	27	136	94	0	3.08	0	0	1760.75	6533.04	6517.87	0.02	108.06	12.84	5.168
5.173	brummayerbiere3_maxxor128	0	oot	oot	oot	0.00	1.00	4.0	17	38	38	0	12.30	0	0	1771.26	6359.84	6343.44	0.42	161.53	25.96	5.169
5.174	brummayerbiere3_maxxor256	0	oot	oot	oot	0.00	1.00	3.6	9	10	10	0	50.28	0	0	1203.83	5222.16	5206.84	1.25	472.84	73.55	5.170
5.175	...biere3_maxxormaxorand032	0	oot	oot	oot	0.00	1.00	3.4	130	871	304	0	10.43	0	0	1713.67	6034.44	5997.96	0.00	46.31	1.97	5.171
5.176	...biere3_maxxormaxorand064	0	oot	oot	oot	0.00	1.00	3.8	66	419	82	0	41.67	0	0	1654.03	5214.50	5167.00	0.00	109.49	4.57	5.172
5.177	...biere3_maxxormaxorand128	0	oot	oot	oot	0.00	1.00	3.8	15	31	25	0	156.18	0	0	1434.06	5582.66	5548.82	0.08	181.34	26.09	5.173
5.178	...biere3_maxxormaxorand256	0	oot	oot	oot	0.00	1.00	4.0	7	6	6	0	639.47	0	0	877.70	2718.18	2688.09	21.68	268.95	87.68	5.174
5.179	...yerbiere3_minandmaxor128	0	oot	oot	oot	0.00	1.00	4.5	17	67	17	1	98.05	0	0	1554.30	4846.09	4794.80	0.12	372.88	27.38	5.175
5.180	...yerbiere3_minandmaxor256	0	oot	oot	oot	0.00	1.00	4.7	9	10	7	1	374.84	0	0	1194.29	3474.87	3448.56	8.16	307.62	70.92	5.176
5.181	brummayerbiere3_minxor128	0	oot	oot	oot	0.00	1.00	4.6	32	177	77	0	16.62	0	0	1739.97	6551.58	6517.98	0.08	103.44	10.74	5.177
5.182	brummayerbiere3_minxor256	0	oot	oot	oot	0.00	1.00	4.8	14	32	18	0	66.56	0	0	1406.83	5662.36	5625.16	0.17	343.24	30.28	5.178
5.183	...erbiere3_minxorminand064	0	oot	oot	oot	0.00	1.00	5.1	54	315	127	0	49.37	0	0	1728.19	6218.22	6155.67	0.01	57.79	5.45	5.179

Continued on next page

Table C.2 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector with FLR (Parallel-Incrmental-FLR)																	cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-inc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.184	...erbiere3_minxorminand128	0	oot	oot	oot	0.00	1.00	4.8	16	47	43	0	183.59	0	0	1234.40	5846.41	5818.97	0.07	198.33	23.67	5.180
5.185	...erbiere3_minxorminand256	0	oot	oot	oot	0.00	1.00	4.4	8	12	11	0	720.94	0	0	455.08	2255.08	2228.47	0.72	121.58	32.68	5.181
5.186	brummayerbiere3_mulhs16	0	oot	oot	oot	0.00	1.00	0.4	20	137	79	1	0.00	0	0	17.02	1676.54	1676.25	0.00	1605.04	4.63	5.182
5.187	brummayerbiere3_mulhs32	0	oot	oot	oot	0.00	1.00	1.0	20	141	88	0	0.00	0	0	23.22	101.30	100.61	0.00	4.65	0.26	5.183
5.188	brummayerbiere3_mulhs64	0	oot	oot	oot	0.00	1.00	3.7	38	141	77	0	0.00	0	0	1520.01	5346.42	5343.49	0.00	434.84	7.34	5.184
5.189	...mayerbiere_countbits1024	0	oot	oot	oot	0.00	1.00	1.8						1	0		0.00	0.00	0.00	0.00	0.00	5.185
5.190	brummayerbiere_countbits128	0	oot	oot	oot	0.00	1.00	3.6						1	0		0.00	0.00	0.00	0.00	0.00	5.186
5.191	brummayerbiere_countbits256	0	oot	oot	oot	0.00	1.00	3.7						1	0		0.00	0.00	0.00	0.00	0.00	5.187
5.192	brummayerbiere_countbits512	0	oot	oot	oot	0.00	1.00	3.6						1	0		0.00	0.00	0.00	0.00	0.00	5.188
5.193	...biere_countbitsrotate032	0	oot	oot	oot	0.00	1.00	0.2						0	0		0.00	0.00	0.00	0.00	0.00	5.189
5.194	...biere_countbitsrotate064	0	oot	oot	oot	0.00	1.00	0.7						0	0		0.00	0.00	0.00	0.00	0.00	5.190
5.195	...biere_countbitsrotate128	0	oot	oot	oot	0.00	1.00	1.6						0	0		0.00	0.00	0.00	0.00	0.00	5.191
5.196	...biere_countbitsrotate256	0	oot	oot	oot	0.00	1.00	4.3						0	0		0.00	0.00	0.00	0.00	0.00	5.192
5.197	...yerbiere_countbitssrl064	0	oot	oot	oot	0.00	1.00	1.1						1	0		0.00	0.00	0.00	0.00	0.00	5.193
5.198	...yerbiere_countbitssrl128	0	oot	oot	oot	0.00	1.00	3.8						1	0		0.00	0.00	0.00	0.00	0.00	5.194
5.199	...yerbiere_countbitssrl256	0	oot	oot	oot	0.00	1.00	4.7						1	0		0.00	0.00	0.00	0.00	0.00	5.195
5.200	challenge_integerOverflow	0	oot	oot	oot	0.00	1.00	0.4	5	12	8	0	0.00	0	0	13.50	39.94	39.44	0.00	9.70	1.74	5.196
5.201	...erbiere4_unconstrained03	0	oom	oom	oom	-0.14	1.00	7.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.198
5.202	...erbiere4_unconstrained08	0	oom	oom	oom	-0.18	1.00	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.5
5.203	...erbiere4_unconstrained09	0	oom	oom	oom	-0.41	0.99	6.8	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.200
5.204	...erbiere4_unconstrained05	0	oom	oom	oom	-0.56	0.99	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.6
5.205	...erbiere4_unconstrained10	0	oom	oom	oom	-1.83	0.99	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.201

Table C.3: Detailed results of PBoolector in non-incremental mode (Parallel-Non-Incremental) compared to the reference times Base-Non-Incremental. Last column par-ninc-flr contains a cross reference to detailed results of Parallel-Non-Incremental-FLR in Tab.C.4.

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental)																cmp
				base-ninc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-ninc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
2.1	log-slicing_bvmul_14	20	1477.50	1496.45	885.18	611.27	1.69	0.1	2	2	2	2	0.00	0.00	884.59	1724.80	1724.72	0.27	883.92	344.96	2.1
2.2	log-slicing_bvsmod_14	20	1414.54	1430.98	1147.20	283.78	1.25	0.1	13	42	25	39	0.03	0.05	1146.49	3000.90	2998.50	0.21	231.73	19.87	2.3
2.3	log-slicing_bvudiv_17	20	1270.36	1291.07	1042.83	248.24	1.24	0.2	5	9	6	6	0.01	0.03	1042.10	2912.85	2911.62	0.33	615.74	126.65	2.4
2.4	...yerbiere_countbitssrl032	20	387.06	395.11	201.87	193.24	1.96	0.1	34	123	14	41	0.04	0.02	201.20	480.15	474.29	0.00	52.87	1.68	3.13
2.5	...ayerbiere2_smulov4bw1024	10	1518.87	1499.29	1326.78	172.51	1.13	1.6	3	3	2	2	0.00	0.00	1326.11	1902.83	1841.34	6.66	1124.79	271.83	2.7
2.6	log-slicing_bvmul_13	20	428.63	431.54	290.69	140.85	1.48	0.1	2	2	2	2	0.00	0.00	290.07	537.38	537.30	0.24	289.32	107.48	2.6
2.7	challenge_multiplyOverflow	20	1035.61	1041.44	941.87	99.57	1.11	0.1	2	2	2	2	0.00	0.00	941.15	1453.43	1453.31	4.11	930.75	290.69	2.8
2.8	log-slicing_bvurem_14	20	757.16	767.68	704.68	63.00	1.09	0.1	3	3	3	3	0.00	0.00	704.42	1009.34	1009.17	0.77	528.50	126.17	2.14
2.9	...yerbiere3_isqrtinvalidvc	10	181.66	183.06	129.70	53.36	1.41	0.0	30	224	137	0	0.02	0.02	129.02	495.60	482.91	0.02	9.82	0.75	2.15
2.10	log-slicing_bvmul_12	20	129.19	131.36	82.95	48.41	1.58	0.0	2	2	2	2	0.00	0.00	82.37	136.89	136.81	0.22	81.81	27.38	2.16
2.11	...biere3_isqrtaddinvalidvc	10	185.60	181.99	138.30	43.69	1.32	0.0	30	224	140	0	0.02	0.02	137.65	518.22	503.60	0.02	9.68	0.77	2.10
2.12	log-slicing_bvudiv_15	20	162.70	166.23	150.32	15.91	1.11	0.0	3	3	3	3	0.00	0.00	149.70	199.06	198.85	0.62	98.57	24.88	2.19
2.13	log-slicing_bvsmod_12	20	138.58	141.04	127.26	13.78	1.11	0.1	13	42	25	39	0.03	0.04	126.56	337.95	335.77	0.17	23.11	2.24	2.20
2.14	log-slicing_bvurem_13	20	147.97	149.47	137.85	11.62	1.08	0.0	3	3	3	3	0.00	0.00	137.18	186.88	186.72	0.63	87.40	23.36	2.21
2.15	log-slicing_bvadd_27966	20	232.01	236.99	234.42	2.57	1.01	0.3	0	1	1	0	0.00	0.00	234.23	234.04	215.67	234.04	234.04	234.04	2.27
2.16	log-slicing_bvadd_26192	20	202.52	207.67	205.87	1.80	1.01	0.3	0	1	1	0	0.00	0.00	205.64	205.46	190.50	205.46	205.46	205.46	2.23
2.17	log-slicing_bvadd_23531	20	167.45	173.18	171.46	1.72	1.01	0.2	0	1	1	0	0.00	0.00	170.82	164.24	154.56	164.24	164.24	164.24	2.25
2.18	log-slicing_bvadd_21757	20	165.07	169.85	168.44	1.41	1.01	0.2	0	1	1	0	0.00	0.00	168.30	162.91	155.30	162.91	162.91	162.91	2.28
2.19	log-slicing_bvadd_24418	20	198.09	204.18	203.26	0.92	1.00	0.3	0	1	1	0	0.00	0.00	202.66	202.44	191.48	202.44	202.44	202.44	2.37
2.20	log-slicing_bvadd_29740	20	264.93	273.86	273.05	0.81	1.00	0.3	0	1	1	0	0.00	0.00	272.38	271.24	244.68	271.24	271.24	271.24	2.26
2.21	log-slicing_bvadd_25305	20	186.34	190.48	189.88	0.60	1.00	0.3	0	1	1	0	0.00	0.00	189.23	189.04	177.15	189.04	189.04	189.04	2.39
2.22	log-slicing_bvadd_19983	20	121.59	124.66	124.20	0.46	1.00	0.2	0	1	1	0	0.00	0.00	124.02	119.86	113.01	119.86	119.86	119.86	2.24
2.23	log-slicing_bvadd_18209	20	101.26	102.51	102.13	0.38	1.00	0.2	0	1	1	0	0.00	0.00	101.55	98.67	94.44	98.67	98.67	98.67	3.5
2.24	log-slicing_bvadd_20870	20	132.00	135.08	134.70	0.38	1.00	0.2	0	1	1	0	0.00	0.00	134.07	129.35	122.34	129.35	129.35	129.35	2.29
2.25	pspace_shift1add.22961	20	110.71	113.32	113.05	0.27	1.00	0.1	0	1	1	0	0.00	0.00	112.49	112.41	111.93	112.41	112.41	112.41	2.41
2.26	pspace_shift1add.24955	20	128.47	131.62	131.37	0.25	1.00	0.1	0	1	1	0	0.00	0.00	130.71	130.62	130.07	130.62	130.62	130.62	2.30
2.27	brummayerbiere_bitrev8192	20	101.97	101.56	101.32	0.24	1.00	0.0	0	1	1	0	0.00	0.00	0.05	0.04	0.00	0.04	0.04	0.04	2.31
2.28	pspace_shift1add.25952	20	143.28	146.71	146.53	0.18	1.00	0.1	0	1	1	0	0.00	0.00	145.94	145.85	145.28	145.85	145.85	145.85	2.38
2.29	log-slicing_bvadd_19096	20	109.78	113.21	113.05	0.16	1.00	0.2	0	1	1	0	0.00	0.00	112.45	108.95	103.24	108.95	108.95	108.95	2.40
2.30	pspace_shift1add.29940	20	195.38	200.45	200.31	0.14	1.00	0.1	0	1	1	0	0.00	0.00	199.69	199.59	198.96	199.59	199.59	199.59	2.36
2.31	pspace_shift1add.27946	20	167.81	172.13	172.00	0.13	1.00	0.2	0	1	1	0	0.00	0.00	171.31	171.22	170.63	171.22	171.22	171.22	2.32
2.32	pspace_shift1add.28943	20	186.84	191.46	191.35	0.11	1.00	0.1	0	1	1	0	0.00	0.00	190.72	190.61	190.00	190.61	190.61	190.61	2.34
2.33	log-slicing_bvsub_05994	20	414.31	415.57	415.47	0.10	1.00	0.1	0	1	1	0	0.00	0.00	414.86	413.99	412.00	413.99	413.99	413.99	2.42
2.34	pspace_shift1add.26949	20	152.33	156.05	155.95	0.10	1.00	0.1	0	1	1	0	0.00	0.00	155.39	155.30	154.72	155.30	155.30	155.30	2.35
2.35	RWS_Example_18.txt	10	160.61	160.56	160.47	0.09	1.00	0.1	0	1	1	0	0.00	0.00	159.79	159.69	159.47	159.69	159.69	159.69	3.11
2.36	pspace_shift1add.23958	20	114.08	116.81	116.80	0.01	1.00	0.1	0	1	1	0	0.00	0.00	116.02	115.94	115.43	115.94	115.94	115.94	2.33
3.1	log-slicing_bvsub_04000	20	545.84	545.62	545.62	0.00	1.00	0.1	0	1	1	0	0.00	0.00	545.01	544.58	543.63	544.58	544.58	544.58	3.1
3.2	bmc-bv_graycode	10	442.54	448.89	448.93	-0.04	1.00	0.3	0	1	1	0	0.00	0.00	0.18	0.18	0.04	0.18	0.18	0.18	3.2
3.3	log-slicing_bvadd_22644	20	156.03	165.93	166.46	-0.53	1.00	0.2	0	1	1	0	0.00	0.00	165.69	159.68	151.55	159.68	159.68	159.68	3.3
3.4	bmc-bv_magic	10	443.86	453.08	453.67	-0.59	1.00	0.3	0	1	1	0	0.00	0.00	0.60	0.60	0.45	0.60	0.60	0.60	3.7
3.5	log-slicing_bvadd_27079	20	216.41	220.70	221.90	-1.20	0.99	0.3	0	1	1	0	0.00	0.00	221.33	221.15	203.33	221.15	221.15	221.15	3.4
3.6	log-slicing_bvadd_28853	20	245.33	251.65	252.88	-1.23	1.00	0.3	0	1	1	0	0.00	0.00	252.29	252.10	230.18	252.10	252.10	252.10	3.6
3.7	brummayerbiere3_isqrt	20	145.81	146.46	161.97	-15.51	0.90	0.0	42	308	186	0	0.03	0.03	161.25	600.46	580.36	0.02	9.79	0.57	2.22
3.8	log-slicing_bvsmod_13	20	360.26	371.81	395.16	-23.35	0.94	0.1	13	42	25	39	0.03	0.05	394.47	1173.36	1170.97	0.19	70.80	7.77	3.8

Continued on next page

Table C.3 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental)																cmp
				base-ninc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-ninc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
3.9	core_ext_con_052_002_1024	20	194.60	205.80	239.38	-33.58	0.86	0.4	1	1	1	1	0.00	0.00	238.82	238.73	238.01	42.46	196.27	119.37	2.17
3.10	core_ext_con_044_002_1024	20	195.82	197.33	236.77	-39.44	0.83	0.4	1	1	1	1	0.00	0.00	236.11	236.04	235.47	39.24	196.81	118.02	2.9
3.11	core_ext_con_048_002_1024	20	487.72	492.48	535.40	-42.92	0.92	0.4	1	1	1	1	0.00	0.00	534.72	534.65	534.04	43.70	490.95	267.32	2.2
3.12	core_ext_con_056_002_1024	20	279.98	284.33	327.73	-43.40	0.87	0.4	1	1	1	1	0.00	0.00	327.07	326.98	326.21	45.63	281.34	163.49	2.12
3.13	core_ext_con_060_002_1024	20	220.71	220.86	267.17	-46.31	0.83	0.4	1	1	1	1	0.00	0.00	266.48	266.38	265.58	45.46	220.92	133.19	3.9
3.14	core_ext_con_036_002_1024	20	269.99	268.42	315.38	-46.96	0.85	0.3	1	1	1	1	0.00	0.00	314.74	314.68	314.21	46.11	268.57	157.34	2.5
3.15	brummayerbiere3_isqrtadd	20	144.73	153.58	206.36	-52.78	0.74	0.0	37	255	157	0	0.03	0.02	205.64	738.90	719.64	0.02	9.46	0.91	3.10
3.16	core_ext_con_064_002_1024	20	344.29	330.81	385.43	-54.62	0.86	0.5	1	1	1	1	0.00	0.00	384.83	384.72	383.87	55.69	329.03	192.36	2.18
3.17	core_ext_con_060_001_1024	20	110.60	111.42	173.69	-62.27	0.64	0.4	1	1	1	1	0.00	0.00	173.00	172.90	172.14	60.94	111.97	86.45	2.13
3.18	log-slicing_bvudiv_16	20	390.99	398.23	471.57	-73.34	0.84	0.1	7	15	6	6	0.01	0.02	470.97	1422.29	1421.60	0.02	339.13	40.64	3.12
3.19	core_ext_con_064_001_1024	20	128.25	129.32	212.58	-83.26	0.61	0.5	1	1	1	1	0.00	0.00	211.86	211.76	210.94	83.82	127.94	105.88	2.11
3.20	...ayerbiere2_smulov4bw0512	10	128.78	130.66	224.36	-93.70	0.58	0.5	3	3	2	2	0.00	0.00	223.62	350.78	341.55	0.71	98.56	50.11	3.14
3.21	pspace_power2sum.5898	20	343.57	351.34	451.05	-99.71	0.78	0.3	1	1	1	1	0.00	0.00	450.45	450.21	449.42	104.23	345.98	225.11	3.15
3.22	pspace_power2sum.5699	20	347.92	348.90	456.66	-107.76	0.76	0.3	1	1	1	1	0.00	0.00	456.04	455.81	455.05	107.81	348.00	227.90	3.16
3.23	log-slicing_bvsub_06991	20	307.68	315.94	425.63	-109.69	0.74	0.1	1	1	1	1	0.00	0.00	425.05	423.59	418.33	110.94	312.66	211.80	3.17
3.24	pspace_power2sum.6097	20	388.12	387.37	506.75	-119.38	0.76	0.4	1	1	1	1	0.00	0.00	506.12	505.87	505.03	117.27	388.60	252.93	3.18
3.25	pspace_power2sum.6495	20	504.51	505.78	625.69	-119.91	0.81	0.4	1	1	1	1	0.00	0.00	625.13	624.85	623.94	120.52	504.33	312.42	3.19
3.26	pspace_power2sum.8684	20	1028.30	1025.18	1154.73	-129.55	0.89	0.4	1	1	1	1	0.00	0.00	1154.16	1154.04	1152.58	115.51	1038.53	577.02	3.20
3.27	log-slicing_bvsub_10979	20	161.21	145.19	276.56	-131.37	0.52	0.1	1	1	1	1	0.00	0.00	275.97	270.14	257.67	124.09	146.05	135.07	3.21
3.28	log-slicing_bvsub_07988	20	321.69	327.33	470.39	-143.06	0.70	0.1	1	1	1	1	0.00	0.00	470.10	466.85	460.01	139.29	327.56	233.42	3.22
3.29	pspace_power2sum.5500	20	496.16	493.29	642.79	-149.50	0.77	0.4	1	1	1	1	0.00	0.00	642.13	641.92	641.19	150.94	490.98	320.96	3.23
3.30	log-slicing_bvsub_08985	20	378.56	390.79	555.37	-164.58	0.70	0.1	1	1	1	1	0.00	0.00	554.80	551.15	542.69	162.53	388.62	275.58	3.24
3.31	pspace_power2sum.6296	20	434.76	435.13	600.38	-165.25	0.72	0.4	1	1	1	1	0.00	0.00	599.82	599.55	598.69	156.96	442.59	299.78	3.25
3.32	lfsr_lfsr_004_159_128	20	112.10	152.78	323.39	-170.61	0.47	3.4	2	1	1	0	0.00	0.00	319.32	305.46	98.59	82.61	138.79	101.82	4.99
3.33	pspace_power2sum.7092	20	452.50	454.24	640.07	-185.83	0.71	0.4	1	1	1	1	0.00	0.00	639.50	638.98	637.97	179.57	459.42	319.49	3.26
3.34	pspace_power2sum.7689	20	515.93	517.58	716.92	-199.34	0.72	0.4	1	1	1	1	0.00	0.00	716.32	715.66	714.51	199.69	515.98	357.83	3.27
3.35	pspace_power2sum.9480	20	751.83	748.66	956.80	-208.14	0.78	0.5	1	1	1	1	0.00	0.00	956.18	956.05	954.40	208.59	747.46	478.03	3.28
3.36	pspace_power2sum.9878	20	573.35	578.37	791.88	-213.51	0.73	0.4	1	1	1	1	0.00	0.00	791.31	791.18	789.41	218.62	572.56	395.59	3.29
3.37	log-slicing_bvsub_11976	20	664.63	696.11	926.47	-230.36	0.75	0.4	1	1	1	1	0.00	0.00	925.46	918.69	904.30	247.66	671.03	459.35	3.30
3.38	pspace_power2sum.8485	20	872.30	872.26	1107.48	-235.22	0.79	0.6	1	1	1	1	0.00	0.00	1106.89	1106.78	1105.36	234.82	871.95	553.39	3.31
3.39	pspace_power2sum.9281	20	1338.53	1337.10	1572.51	-235.41	0.85	0.8	1	1	1	1	0.00	0.00	1571.84	1571.72	1570.11	235.16	1336.56	785.86	3.32
3.40	pspace_power2sum.8883	20	1035.88	1037.85	1273.81	-235.96	0.81	0.4	1	1	1	1	0.00	0.00	1273.25	1273.13	1271.60	228.99	1044.14	636.56	3.33
3.41	log-slicing_bvsub_09982	20	328.06	333.74	571.22	-237.48	0.58	0.2	1	1	1	1	0.00	0.00	570.56	565.88	555.78	235.31	330.57	282.94	3.34
3.42	pspace_power2sum.7490	20	505.69	505.54	754.73	-249.19	0.67	0.3	1	1	1	1	0.00	0.00	754.02	753.39	752.28	247.58	505.81	376.69	3.35
3.43	...ayerbiere2_smulov4bw0768	10	563.42	566.25	821.65	-255.40	0.69	1.1	3	3	2	2	0.00	0.00	820.98	1215.31	1185.28	2.93	648.31	173.62	3.36
3.44	pspace_power2sum.7291	20	971.68	970.12	1228.72	-258.60	0.79	0.6	1	1	1	1	0.00	0.00	1228.11	1227.51	1226.43	257.68	969.83	613.76	3.37
3.45	lfsr_lfsr_004_111_112	20	178.05	198.93	477.92	-278.99	0.42	5.1	7	4	3	0	4.80	16.89	448.34	1032.86	433.51	44.66	73.67	51.64	4.86
3.46	...ayerbiere2_smulov4bw0896	10	1274.52	1280.01	1572.66	-292.65	0.81	1.3	3	3	2	2	0.00	0.00	1571.95	1953.93	1909.88	4.66	1368.43	279.13	3.38
3.47	pspace_power2sum.7888	20	1145.49	1133.23	1508.77	-375.54	0.75	0.8	1	1	1	1	0.00	0.00	1508.14	1507.45	1506.23	369.40	1138.05	753.72	3.39
3.48	brummayerbiere2_smulov1bw24	20	134.33	135.54	533.51	-397.97	0.25	0.0	74	496	124	0	0.07	0.06	532.59	1863.25	1851.73	0.01	8.15	1.47	3.48
3.49	brummayerbiere_nlzbe256	20	117.51	119.31	540.34	-421.03	0.22	0.4	15	81	41	0	0.11	0.08	539.44	1641.24	1584.76	0.07	39.33	8.46	3.44
3.50	log-slicing_bvsrem_14	20	256.90	263.76	688.53	-424.77	0.38	0.1	7	21	16	18	0.02	0.03	687.82	1905.52	1904.35	0.19	179.54	29.32	3.40
3.51	pspace_power2sum.6694	20	1005.81	1017.69	1465.41	-447.72	0.69	0.6	1	1	1	1	0.00	0.00	1464.81	1464.52	1463.57	456.91	1007.61	732.26	3.42
3.52	...ayerbiere2_smulov4bw0640	10	141.93	143.77	594.52	-450.75	0.24	0.7	3	3	2	2	0.00	0.00	593.80	1023.00	1005.40	1.67	410.64	146.14	3.41
3.53	pspace_power2sum.8286	20	898.41	893.24	1348.12	-454.88	0.66	0.4	1	1	1	1	0.00	0.00	1347.48	1347.37	1345.99	454.67	892.70	673.68	3.43
3.54	log-slicing_bvsdiv_16	20	182.35	185.31	716.38	-531.07	0.26	0.1	10	38	19	18	0.02	0.04	715.70	1811.37	1809.63	0.02	146.47	17.93	3.45
3.55	pspace_power2sum.6893	20	1102.02	1106.41	1649.32	-542.91	0.67	0.6	1	1	1	1	0.00	0.00	1648.64	1648.33	1647.35	545.72	1102.60	824.16	3.46
3.56	log-slicing_bvult_6000	20	406.38	410.02	967.37	-557.35	0.42	0.7	5	10	9	10	0.96	0.17	965.56	3021.01	3007.54	26.29	372.55	104.17	3.47
3.57	brummayerbiere3_maxor064	20	209.14	209.48	1164.41	-954.93	0.18	0.1	129	129	3	109	0.19	0.13	1163.32	1384.66	1369.70	0.00	36.74	3.78	3.49
3.58	log-slicing_bvsdiv_17	20	502.64	511.08	1511.03	-999.95	0.34	0.2	9	27	16	24	0.03	0.08	1510.27	4988.46	4983.08	0.04	334.91	58.69	3.50
3.59	log-slicing_bvult_6997	20	201.19	204.81	1299.62	-1094.81	0.16	0.7	5	10	9	10	0.99	0.16	1297.76	3678.51	3664.86	27.05	477.88	126.85	3.51

Continued on next page

Table C.3 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental)																cmp
				base-ninc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-ninc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
3.60	log-slicing_bvslt_5743	20	444.58	444.92	1606.19	-1161.27	0.28	0.7	7	18	14	18	1.48	0.28	1603.70	5575.50	5553.28	24.93	323.86	97.82	3.52
3.61	log-slicing_bvult_7994	20	230.10	234.30	1565.61	-1331.31	0.15	0.7	5	10	9	10	0.96	0.18	1564.17	3894.66	3880.89	27.94	613.80	134.30	3.53
3.62	lfsr_lfsr_008_159_048	20	100.56	110.87	1462.91	-1352.04	0.08	4.1	9	12	12	0	3.66	10.54	1442.36	5348.98	3983.46	25.73	105.60	61.48	4.122
3.63	lfsr_lfsr_008_159_064	20	149.52	166.93	1555.33	-1388.40	0.11	5.4	10	8	6	0	5.73	16.93	1524.01	4605.45	3027.27	42.10	156.81	86.90	4.95
4.1	log-slicing_bvslt_17631	20	1750.56	1785.50	oot	-14.50	0.99	6.4	4	14	14	0	40.81	1.03	497.45	2424.10	2302.57	38.99	182.60	89.78	4.1
4.2	...est_v7_r12_vr5_c1_s14336	10	933.17	1778.65	oot	-21.35	0.99	2.5	19	87	80	0	4.54	4.12	1712.38	6029.47	5691.95	0.18	46.31	36.99	4.2
4.3	..._smtcomp09_lf_bv_formula	20	180.24	202.33	sf	-39.18	0.84	0.2								0.00	0.00	0.00	0.00	0.00	4.85
4.4	...smtcomp09_std_bv_formula	20	108.86	128.59	sf	-106.17	0.55	0.2								0.00	0.00	0.00	0.00	0.00	4.114
4.5	...mtcomp09_full_bv_formula	20	172.46	159.04	sf	-204.78	0.44	0.3								0.00	0.00	0.00	0.00	0.00	4.96
4.6	...test_v5_r15_vr5_c1_s8246	20	1106.64	1531.37	oot	-268.63	0.85	2.3	18	93	88	0	4.19	3.83	1786.88	6233.24	5899.56	0.16	70.95	35.22	4.3
4.7	log-slicing_bvslt_16888	20	1502.56	1524.13	oot	-275.87	0.85	6.4	4	14	14	0	37.89	1.01	561.14	1992.90	1871.32	35.99	208.37	73.81	4.4
4.8	log-slicing_bvsdiv_18	20	1485.52	1509.37	oot	-290.63	0.84	0.2	7	34	24	4	0.02	0.10	1550.86	2281.99	2277.59	0.13	896.93	30.43	4.5
4.9	...est_v5_r15_vr5_c1_s26657	20	oot	1480.60	oot	-319.40	0.82	2.4	18	93	88	0	4.17	3.78	1729.70	6252.95	5912.07	0.16	43.92	34.55	4.6
4.10	...est_v7_r7_vr10_c1_s32506	10	1754.95	1394.41	oot	-405.59	0.77	1.7	23	119	112	0	3.12	3.09	1792.38	6400.61	6048.31	0.09	26.72	21.85	4.7
4.11	..._smtcomp09_sw_bv_formula	20	202.42	204.90	sf	-455.58	0.31	0.4								0.00	0.00	0.00	0.00	0.00	4.83
4.12	pspace_power2sum.9082	20	1301.91	1303.66	oot	-496.34	0.72	0.7	1	1	1	0	0.00	0.00	508.18	508.12	507.34	508.12	508.12	508.12	4.8
4.13	pipe_pipe-noabs.atlas.qf_bv	20	1290.10	1298.23	oot	-501.77	0.72	0.1	54	416	62	0	0.12	0.08	1779.46	6478.55	6464.01	0.00	29.32	6.57	4.9
4.14	lfsr_lfsr_008_159_128	20	285.74	363.40	oom	-550.56	0.40	6.8	5	1	1	0	0.00	0.00	850.76	800.98	228.08	125.88	248.06	160.20	4.52
4.15	...st_v5_r10_vr10_c1_s21502	20	697.82	1247.16	oot	-552.84	0.69	1.7	20	110	104	0	2.87	2.84	1337.19	6536.31	6183.34	0.09	28.01	22.70	4.10
4.16	log-slicing_bvslt_16145	20	1188.56	1208.81	oot	-591.19	0.67	2.0	4	12	12	0	4.12	0.40	503.38	1521.73	1489.01	30.30	211.11	66.16	4.11
4.17	log-slicing_bvult_15970	20	1070.63	1091.11	oot	-708.89	0.61	2.1	4	12	12	0	3.66	0.40	436.49	1239.80	1204.38	31.53	159.78	53.90	4.12
4.18	brummayerbiere2_smulov1bw32	20	1056.25	1078.07	oot	-721.93	0.60	0.1								0.00	0.00	0.00	0.00	0.00	4.13
4.19	log-slicing_bvslt_15402	20	1048.34	1064.76	oot	-735.24	0.59	2.0	4	12	12	0	3.99	0.40	510.69	1710.23	1678.80	30.78	154.53	74.36	4.14
4.20	log-slicing_bvult_14973	20	1019.50	1027.31	oot	-772.69	0.57	2.0	4	12	12	0	4.01	0.38	376.48	1151.56	1119.44	28.24	137.04	50.07	4.15
4.21	...test_v7_r7_vr5_c1_s19694	10	1370.11	999.27	oot	-800.73	0.56	1.7	22	111	104	0	2.88	2.89	1722.77	6404.93	6057.97	0.09	27.34	22.16	4.16
4.22	VS3_VS3-benchmark-A7	10	985.79	987.88	oot	-812.12	0.55	0.4	70	532	306	0	0.69	0.66	1625.31	6928.30	6737.79	0.26	16.02	3.16	4.17
4.23	log-slicing_bvslt_13916	20	908.99	933.26	oot	-866.74	0.52	2.0	4	12	12	3	4.00	0.40	426.36	4759.93	4724.84	29.76	1310.66	183.07	4.18
4.24	log-slicing_bvult_12979	20	893.49	915.23	oot	-884.77	0.51	2.0	4	12	12	0	3.76	0.40	276.09	1017.17	986.99	32.70	73.31	44.22	4.19
4.25	log-slicing_bvult_13976	20	893.73	911.14	oot	-888.86	0.51	2.0	4	12	12	0	3.81	0.39	338.85	1108.63	1073.99	29.62	112.16	48.20	4.20
4.26	...est_v5_r15_vr5_c1_s23844	20	oot	875.10	oot	-924.90	0.49	2.4	18	93	88	0	4.22	3.81	1738.20	6281.30	5936.59	0.17	45.59	34.70	4.21
4.27	log-slicing_bvslt_14659	20	838.99	854.35	oot	-945.65	0.47	2.0	4	12	12	3	3.87	0.39	370.22	4394.03	4356.57	29.44	1386.40	169.00	4.22
4.28	log-slicing_bvslt_13173	20	801.46	815.34	oot	-984.66	0.45	2.0	4	12	12	1	4.00	0.40	307.31	2010.41	1977.75	27.72	1000.84	83.77	4.23
4.29	log-slicing_bvsrem_15	20	791.10	801.92	oot	-998.08	0.45	0.1	7	21	16	16	0.01	0.03	1046.52	4738.97	4737.81	0.22	445.16	75.22	4.24
4.30	float_newton.6.3.i	10	564.04	787.53	oot	-1012.47	0.44	0.8	32	227	46	0	2.48	2.66	1756.05	6329.59	5900.73	0.05	36.50	12.10	4.25
4.31	..._test_v7_r7_vr1_c1_s4574	10	609.33	754.52	oot	-1045.48	0.42	1.9	20	144	144	0	3.30	3.39	1386.57	6772.58	6366.36	10.66	31.27	16.24	4.26
4.32	...est_v5_r10_vr10_c1_s7608	20	1199.14	715.00	oot	-1085.00	0.40	1.7	20	109	104	0	2.89	2.84	1334.87	6485.09	6127.84	0.09	29.24	22.68	4.27
4.33	...est_v5_r10_vr5_c1_s13679	10	899.64	696.33	oot	-1103.67	0.39	1.7	20	111	104	0	2.79	2.79	1281.28	6598.90	6236.86	0.10	27.41	21.92	4.28
4.34	log-slicing_bvslt_11687	20	665.86	672.39	oot	-1127.61	0.37	2.0	4	12	12	2	3.95	0.40	282.45	3424.54	3390.90	27.47	1288.00	136.98	4.29
4.35	...st_v5_r15_vr10_c1_s11127	10	346.03	669.69	oot	-1130.31	0.37	2.4	18	93	88	0	4.20	3.82	1716.35	6356.93	6009.26	0.16	41.27	34.36	4.30
4.36	...tcomp09_stall_bv_formula	20	609.53	615.59	oot	-1184.41	0.34	0.7	20	142	49	0	2.54	2.13	1526.58	6483.44	6221.10	0.03	73.79	19.89	4.31
4.37	...test_v7_r7_vr1_c1_s22845	10	316.63	605.49	oot	-1194.51	0.34	1.9	20	144	144	0	3.29	3.38	1295.96	6762.78	6345.49	6.76	28.45	14.86	4.32
4.38	log-slicing_bvult_9988	20	555.03	572.93	oot	-1227.07	0.32	2.2	5	12	12	4	5.39	0.41	1553.63	5690.25	5654.24	27.64	1297.99	210.75	4.33
4.39	log-slicing_bvslt_10201	20	552.92	571.43	oot	-1228.57	0.32	1.9	4	12	12	3	3.94	0.40	229.00	4100.15	4063.01	30.36	1392.06	157.70	4.34
4.40	log-slicing_bvult_10985	20	556.62	561.14	oot	-1238.86	0.31	2.0	4	12	12	0	3.96	0.41	283.00	1005.89	972.24	30.53	73.28	43.73	4.35
4.41	log-slicing_bvult_8991	20	542.61	554.93	oot	-1245.07	0.31	1.9	4	12	12	0	3.86	0.41	240.27	907.97	876.96	29.61	62.36	39.48	4.36
4.42	log-slicing_bvslt_9458	20	539.16	553.43	oot	-1246.57	0.31	1.9	4	12	12	3	3.94	0.40	238.63	3290.04	3254.37	28.13	925.11	126.54	4.37
4.43	log-slicing_bvult_11982	20	518.16	528.44	oot	-1271.56	0.29	2.0	4	12	12	0	3.77	0.40	251.88	960.96	928.00	28.78	82.07	41.78	4.38
4.44	float_mul_03_30_4	20	514.80	520.18	oot	-1279.82	0.29	0.3	103	808	264	0	0.94	0.74	1795.68	7067.22	6838.32	0.00	12.63	2.85	4.39
4.45	log-slicing_bvslt_10944	20	505.68	510.44	oot	-1289.56	0.28	2.0	4	12	12	3	3.92	0.41	288.39	3349.22	3315.38	29.34	993.71	128.82	4.40
4.46	...test_v5_r10_vr5_c1_s8690	20	353.93	459.79	oot	-1340.21	0.26	1.7	20	109	104	0	2.85	2.83	1328.53	6519.32	6165.68	0.10	28.20	22.48	4.41

Continued on next page

Table C.3 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental)																	cmp
				base-ninc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search							par-ninc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
4.47	float_newton.3.3.i	20	355.49	450.85	oot	-1349.15	0.25	0.8	32	232	60	0	2.52	2.77	1791.81	6225.22	5815.16	0.04	34.14	12.45	4.42
4.48	float_newton.1.3.i	20	385.34	445.62	oot	-1354.38	0.25	0.8	37	263	66	0	2.87	3.10	1773.38	6498.07	6056.41	0.03	36.95	11.79	4.43
4.49	float_newton.4.3.i	20	334.84	418.44	oot	-1381.56	0.23	0.8	31	224	48	0	2.40	2.63	1782.71	6088.28	5693.25	0.04	36.96	12.71	4.44
4.50	...est_v5_r15_vr1_c1_s26845	20	619.09	415.20	oot	-1384.80	0.23	2.3	19	136	136	0	5.10	4.99	1780.65	6614.46	6199.55	10.07	51.83	21.83	4.45
4.51	float_newton.5.3.i	20	415.28	404.47	oot	-1395.53	0.22	0.8	33	238	44	0	2.68	2.88	1715.33	6346.62	5915.44	0.05	38.15	11.69	4.46
4.52	lfsr_lfsr_008_015_112	20	388.44	399.49	oot	-1400.51	0.22	3.1	30	81	81	0	6.58	11.82	1707.13	6880.26	4513.26	5.45	17.68	12.29	4.47
4.53	...est_v7_r7_vr10_c1_s10625	10	852.76	391.84	oot	-1408.16	0.22	1.6	22	111	104	0	2.90	2.90	1734.13	6329.59	5984.75	0.09	28.76	22.21	4.48
4.54	float_newton.2.3.i	20	289.38	385.61	oot	-1414.39	0.21	0.8	33	243	40	0	2.59	2.77	1768.04	6122.85	5708.17	0.02	36.96	11.98	4.49
4.55	..._test_v7_r7_vr5_c1_s3582	10	326.78	384.03	oot	-1415.97	0.21	1.7	22	111	104	0	2.90	2.91	1734.96	6353.72	6012.59	0.09	27.33	22.29	4.50
4.56	..._smtcomp09_nt_bv_formula	20	156.72	137.27	oom	-1418.12	0.09	6.8								0.00	0.00	0.00	0.00	0.00	4.109
4.57	...test_v7_r12_vr5_c1_s8938	10	615.16	367.14	oot	-1432.86	0.20	2.5	19	87	80	0	4.65	4.20	1730.58	6043.76	5705.72	0.30	46.21	37.54	4.51
4.58	log-slicing_bvslt_8715	20	342.56	350.68	oot	-1449.32	0.19	2.0	5	18	14	5	5.53	0.63	1281.48	4729.69	4683.41	27.65	1003.32	118.24	4.53
4.59	lfsr_lfsr_008_159_112	20	275.94	338.98	oot	-1461.02	0.19	4.9	9	1	1	0	0.00	0.00	1626.84	1567.23	816.60	115.61	198.12	174.14	4.54
4.60	lfsr_lfsr_008_143_128	20	261.79	336.96	oot	-1463.04	0.19	6.2	13	1	1	0	0.00	0.00	1796.22	1686.67	455.33	119.25	225.10	129.74	4.55
4.61	...test_v7_r7_vr1_c1_s24449	20	679.70	330.95	oot	-1469.05	0.18	2.0	20	144	144	0	3.33	3.37	1460.46	6790.56	6387.91	13.06	31.19	17.15	4.56
4.62	...servers_slapd_a_vc149923	10	325.64	330.06	oot	-1469.94	0.18	1.4	40	304	304	0	0.23	2.25	1569.52	7015.22	6743.36	5.50	13.44	6.26	4.57
4.63	...servers_slapd_a_vc149922	10	325.68	329.37	oot	-1470.63	0.18	1.4	40	304	304	0	0.24	2.22	1603.77	6939.67	6671.75	5.33	13.41	6.33	4.58
4.64	lfsr_lfsr_008_159_096	20	275.86	320.47	oot	-1479.53	0.18	6.5	11	2	2	0	3.80	10.73	1760.62	3263.16	1941.70	94.47	183.35	155.39	4.59
4.65	...est_v5_r10_vr5_c1_s13195	10	337.55	314.00	oot	-1486.00	0.17	1.7	20	109	104	0	2.83	2.83	1329.06	6505.58	6149.09	0.05	29.05	22.51	4.61
4.66	log-slicing_bvslt_7972	20	305.25	307.98	oot	-1492.02	0.17	0.7	6	18	14	9	1.49	0.29	1421.53	5691.90	5671.96	27.48	677.21	129.36	4.62
4.67	log-slicing_bvslt_5000	20	296.40	297.74	oot	-1502.26	0.17	0.7	6	18	14	13	1.36	0.29	1319.67	5047.76	5028.74	23.58	451.86	103.02	4.63
4.68	brummayerbiere3_minand256	20	289.66	295.21	oot	-1504.79	0.16	0.3	18	19	3	16	0.10	0.06	1724.32	2649.99	2631.78	0.12	280.15	50.00	4.64
4.69	lfsr_lfsr_008_015_128	20	277.36	286.74	oot	-1513.26	0.16	3.1	28	65	65	0	5.93	10.57	1733.88	6835.84	4381.61	6.15	19.20	13.56	4.65
4.70	...est_v5_r10_vr1_c1_s32538	20	208.16	285.14	oot	-1514.86	0.16	1.9	22	160	160	0	3.37	3.50	1792.37	6805.84	6397.89	7.07	28.27	14.21	4.66
4.71	lfsr_lfsr_008_127_128	20	212.80	282.86	oot	-1517.14	0.16	6.2	13	1	1	0	0.00	0.00	1742.35	1628.76	489.09	109.50	214.97	125.29	4.67
4.72	lfsr_lfsr_008_015_096	20	272.75	280.34	oot	-1519.66	0.16	3.1	35	94	94	0	6.31	11.29	1754.49	6886.57	4430.12	4.40	15.19	9.52	4.68
4.73	VS3_VS3-benchmark-A9	10	268.45	271.77	oot	-1528.23	0.15	0.5	70	543	388	0	0.78	0.71	1742.66	7030.71	6815.32	0.29	6.42	3.08	4.69
4.74	lfsr_lfsr_008_143_112	20	205.39	260.60	oot	-1539.40	0.14	4.6	14	1	1	0	0.00	0.00	1694.16	1604.50	575.22	102.63	182.32	114.61	4.70
4.75	...est_v5_r10_vr1_c1_s13516	10	231.36	254.96	oot	-1545.04	0.14	1.9	20	144	144	0	3.20	3.27	1314.58	6744.38	6342.18	10.26	27.88	15.22	4.71
4.76	...est_v5_r15_vr1_c1_s32559	20	163.75	250.60	oot	-1549.40	0.14	2.3	17	113	111	0	4.44	4.21	1709.01	6529.22	6141.84	0.10	41.83	25.02	4.72
4.77	...test_v5_r5_vr10_c1_s7194	10	141.84	249.36	oot	-1550.64	0.14	0.9	32	200	118	0	2.06	2.07	1729.33	6529.60	6213.34	0.02	50.03	11.14	4.73
4.78	lfsr_lfsr_008_127_112	20	176.00	226.47	oot	-1573.53	0.13	4.5	13	1	1	0	0.00	0.00	1775.97	1697.69	830.84	95.75	167.16	130.59	4.74
4.79	lfsr_lfsr_008_159_080	20	194.33	224.52	oot	-1575.48	0.12	5.8	13	3	3	0	4.35	12.49	1671.56	4559.48	2828.13	76.25	158.59	126.65	4.75
4.80	..._test_v5_r5_vr5_c1_s2800	10	228.14	223.67	oot	-1576.33	0.12	0.9	39	261	108	0	2.70	2.74	1768.60	6472.71	6094.52	0.02	16.55	9.23	4.76
4.81	log-slicing_bvslt_7229	20	217.01	221.19	oot	-1578.81	0.12	0.7	6	18	14	10	1.38	0.29	1447.44	5043.38	5023.82	27.55	709.75	112.08	4.77
4.82	lfsr_lfsr_008_143_096	20	173.94	217.04	oot	-1582.96	0.12	6.1	13	2	2	0	3.54	10.22	1652.51	3313.83	1701.60	89.04	136.68	122.73	4.78
4.83	...test_v7_r7_vr5_c1_s14675	10	364.55	212.19	oot	-1587.81	0.12	1.6	22	111	104	0	2.90	2.90	1718.65	6392.29	6041.66	0.09	28.11	22.12	4.79
4.84	lfsr_lfsr_008_111_128	20	160.68	210.28	oot	-1589.72	0.12	4.2	16	1	1	0	0.00	0.00	1732.29	1645.75	593.78	88.88	153.00	102.86	4.80
4.85	...yerbiere3_minandmaxor064	20	207.79	206.66	oot	-1593.34	0.11	0.1	138	649	23	0	0.62	0.50	1785.44	3488.62	3420.46	0.01	21.39	2.64	4.81
4.86	...est_v7_r7_vr10_c1_s24535	10	237.88	205.90	oot	-1594.10	0.11	1.7	23	119	112	0	3.17	3.13	1749.38	6372.74	6016.60	0.09	26.67	21.31	4.82
4.87	lfsr_lfsr_008_095_128	20	148.89	203.35	oot	-1596.65	0.11	5.6	13	2	2	0	3.21	8.87	1748.53	3219.41	1109.53	118.21	158.10	128.78	4.84
4.88	...mayerbiere2_smulov2bw512	20	198.38	198.78	oot	-1601.22	0.11	2.7	6	8	4	0	0.09	0.02	1500.39	2048.24	1928.17	11.62	402.59	186.20	4.87
4.89	lfsr_lfsr_008_111_112	20	148.60	194.64	oot	-1605.36	0.11	3.6	17	1	1	0	0.00	0.00	1738.70	1655.78	615.60	83.03	140.61	97.40	4.88
4.90	log-slicing_bvslt_6486	20	191.65	194.40	oot	-1605.60	0.11	0.7	7	18	14	14	1.47	0.30	1611.91	5727.80	5706.33	26.98	445.80	108.07	4.89
4.91	lfsr_lfsr_008_095_112	20	129.08	179.91	oot	-1620.09	0.10	5.0	13	2	2	0	2.79	8.38	1778.82	3298.18	1346.24	118.39	157.06	131.93	4.90
4.92	float_newton.5.2.i	20	249.53	175.92	oot	-1624.08	0.10	0.7	49	366	86	0	2.61	2.47	1790.12	6622.77	6212.40	0.01	27.27	7.69	4.91
4.93	lfsr_lfsr_008_047_096	20	160.40	173.71	oot	-1626.29	0.10	4.0	15	12	12	0	4.01	9.77	1721.41	6681.71	3429.18	29.61	50.19	44.25	4.92
4.94	lfsr_lfsr_008_111_096	20	133.32	172.22	oot	-1627.78	0.10	5.9	18	2	2	0	3.23	9.67	1773.80	3267.84	1315.61	78.81	120.91	93.37	4.93
4.95	float_newton.4.2.i	20	143.25	170.46	oot	-1629.54	0.09	0.6	44	273	58	0	1.87	1.82	1779.73	6479.38	6163.67	0.01	31.11	10.37	4.94
4.96	...test_v5_r5_vr1_c1_s14623	10	143.20	158.20	oot	-1641.80	0.09	1.1	40	304	208	0	2.21	2.40	1563.36	6735.09	6390.62	0.02	58.22	6.91	4.97
4.97	lfsr_lfsr_004_143_128	20	116.21	156.03	oot	-1643.97	0.09	6.0	18	2	2	0	3.26	10.24	1777.82	3260.84	1353.61	80.98	130.66	93.17	4.98

Continued on next page

Table C.3 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental)																cmp
				base-ninc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-ninc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
4.98	...mayerbiere2_smulov2bw448	20	150.63	150.34	oot	-1649.66	0.08	2.4								0.00	0.00	0.00	0.00	0.00	4.100
4.99	lfsr_lfsr_008_127_096	20	113.04	149.61	oot	-1650.39	0.08	5.8	16	2	2	0	3.17	9.86	1749.69	3235.57	1650.19	86.99	130.05	104.37	4.101
4.100	lfsr_lfsr_008_079_128	20	107.76	149.60	oot	-1650.40	0.08	5.8	14	3	3	0	4.35	11.45	1694.45	4548.76	1834.88	102.22	133.97	116.63	4.102
4.101	brummayerbiere3_maxxor032	20	143.95	148.04	oot	-1651.96	0.08	0.1	150	1171	205	0	0.32	0.23	1788.05	6722.35	6675.35	0.05	12.67	1.89	4.103
4.102	lfsr_lfsr_008_095_096	20	105.52	147.68	oot	-1652.32	0.08	6.4	15	4	3	0	6.98	21.06	1740.42	4647.48	1939.64	95.85	129.78	110.65	4.104
4.103	tacas07_Y86_btnft	20	135.77	146.59	oot	-1653.41	0.08	0.7	29	208	168	0	0.54	0.53	1785.75	6852.60	6494.58	7.88	38.77	11.59	4.60
4.104	...mayerbiere2_umulov1bw256	20	143.94	145.22	oot	-1654.78	0.08	1.3								0.00	0.00	0.00	0.00	0.00	4.105
4.105	...est_v5_r10_vr1_c1_s19145	10	316.49	144.16	oot	-1655.84	0.08	1.9	22	160	157	0	3.40	3.56	1768.37	6784.87	6374.31	7.75	27.70	13.93	4.106
4.106	float_newton.3.2.i	20	186.56	140.68	oot	-1659.32	0.08	0.6	41	265	80	0	1.84	1.70	1655.85	6148.51	5838.61	0.01	61.62	10.06	4.107
4.107	lfsr_lfsr_008_143_080	20	112.13	139.64	oot	-1660.36	0.08	5.4	16	3	3	0	4.17	11.91	1711.19	4639.51	2571.08	64.25	118.69	103.10	4.108
4.108	float_newton.1.2.i	20	139.98	134.22	oot	-1665.78	0.07	0.6	41	302	72	0	2.00	1.92	1702.43	6485.13	6137.98	0.01	49.29	9.20	4.110
4.109	...test_v5_r5_vr5_c1_s24018	10	182.81	131.06	oot	-1668.94	0.07	0.9	40	268	114	0	2.74	2.74	1790.90	6468.42	6083.11	0.02	15.85	9.07	4.111
4.110	brummayerbiere_countbits064	20	129.34	130.20	oot	-1669.80	0.07	0.2	92	713	631	0	0.21	0.16	1782.12	7098.38	7004.29	0.00	18.40	1.55	4.112
4.111	tacas07_Y86_std	20	129.65	129.50	oot	-1670.50	0.07	0.7	26	184	144	0	0.45	0.44	1779.91	6820.29	6509.41	0.00	37.40	12.56	4.113
4.112	...mayerbiere2_smulov2bw384	20	122.06	122.27	oot	-1677.73	0.07	1.9								0.00	0.00	0.00	0.00	0.00	4.115
4.113	...test_v5_r15_vr1_c1_s8236	20	927.66	120.26	oot	-1679.74	0.07	2.3	17	120	120	0	4.72	4.48	1729.48	6607.16	6205.01	13.07	41.88	23.68	4.116
4.114	lfsr_lfsr_008_143_064	20	103.49	119.65	oot	-1680.35	0.07	4.8	13	6	6	0	3.86	11.33	1621.51	5328.92	3477.13	35.93	111.50	75.06	4.117
4.115	brummayerbiere3_minor256	20	116.13	118.55	oot	-1681.45	0.07	0.7	29	216	216	0	0.48	0.40	1784.58	6182.27	5943.39	2.63	55.35	10.32	4.118
4.116	float_newton.2.2.i	20	143.18	116.29	oot	-1683.71	0.06	0.7	47	293	98	0	2.02	1.94	1743.10	6480.44	6107.75	0.02	27.92	8.96	4.119
4.117	brummayerbiere3_minxor064	20	111.84	115.98	oot	-1684.02	0.06	0.2								0.00	0.00	0.00	0.00	0.00	4.120
4.118	VS3_VS3-benchmark-A6	10	111.93	112.68	oot	-1687.32	0.06	0.2	50	371	176	0	0.47	0.41	1470.60	6861.67	6759.81	0.27	16.86	5.52	4.121
4.119	core_ext_con_060_256_1024	20	101.73	110.06	oot	-1689.94	0.06	3.6	9	40	40	0	53.70	12.25	1566.30	6094.00	4498.13	22.66	134.33	61.56	4.123
4.120	...mayerbiere2_umulov1bw224	20	108.42	109.18	oot	-1690.82	0.06	1.0	26	162	22	0	0.23	0.12	1743.23	6169.86	5802.68	1.12	86.84	18.42	4.124
4.121	float_mult2.c.50	10	109.39	108.05	oot	-1691.95	0.06	1.2	20	137	136	0	1.22	1.27	1508.14	6716.69	6386.44	6.31	41.84	18.81	4.125
4.122	brummayerbiere3_maxand256	20	102.05	103.83	oot	-1696.17	0.06	0.8	37	280	280	0	0.62	0.53	1793.22	6364.25	6035.67	0.95	39.24	6.87	4.126
5.1	...mayerbiere_countbits1024	0	oot	oot	oom	253.69	1.16	6.9	3	7	6	0	0.16	0.03	1531.01	2291.10	2217.37	0.03	625.04	327.30	5.1
5.2	...erbiere4_unconstrained10	0	oom	oom	oom	1.33	1.01	7.0	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.2
5.3	...erbiere4_unconstrained03	0	oom	oom	oom	0.82	1.02	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.3
5.4	...erbiere4_unconstrained07	0	oom	oom	oom	0.62	1.01	6.8	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.7
5.5	...erbiere4_unconstrained06	0	oom	oom	oom	0.51	1.01	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.4
5.6	...erbiere4_unconstrained04	0	oom	oom	oom	0.36	1.00	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.150
5.7	...erbiere4_unconstrained05	0	oom	oom	oom	0.27	1.00	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.149
5.8	...erbiere4_unconstrained01	0	oom	oom	oom	0.11	1.00	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.6
5.9	fft_Sz1024_34824	0	oot	oot	oot	0.00	1.00	1.1	90	704	704	0	2.07	1.73	1754.12	7065.50	6613.40	1.55	4.96	1.72	5.8
5.10	fft_Sz512_15128_0	0	oot	oot	oot	0.00	1.00	0.7								0.00	0.00	0.00	0.00	0.00	5.9
5.11	fft_Sz512_15128_1	0	oot	oot	oot	0.00	1.00	0.5								0.00	0.00	0.00	0.00	0.00	5.10
5.12	fft_Sz512_15128_2	0	oot	oot	oot	0.00	1.00	0.7	170	1336	1256	0	1.33	0.97	1594.09	7096.82	6612.45	0.07	2.32	0.52	5.11
5.13	fft_Sz512_15128_4	0	oot	oot	oot	0.00	1.00	0.8								0.00	0.00	0.00	0.00	0.00	5.12
5.14	fft_Sz512_15128_5	0	oot	oot	oot	0.00	1.00	0.7								0.00	0.00	0.00	0.00	0.00	5.13
5.15	fft_Sz512_15128_6	0	oot	oot	oot	0.00	1.00	0.6	160	1232	1016	0	1.25	0.89	1762.64	7104.16	6696.49	0.07	2.29	0.62	5.14
5.16	...st_v5_r10_vr10_c1_s15708	0	oot	oot	oot	0.00	1.00	1.7	20	110	104	0	2.84	2.84	1310.62	6544.90	6187.18	0.10	27.52	22.04	5.15
5.17	...st_v7_r12_vr10_c1_s15994	10	1367.71	oot	oot	0.00	1.00	2.4	19	87	80	0	4.52	4.10	1721.42	5967.63	5637.20	0.17	47.19	37.07	5.16
5.18	...est_v7_r12_vr1_c1_s10576	0	oot	oot	oot	0.00	1.00	2.3	14	96	96	0	4.54	4.22	1701.34	6663.12	6233.07	21.14	43.57	28.35	5.17
5.19	...._test_v7_r12_vr1_c1_s703	0	oot	oot	oot	0.00	1.00	2.2	14	96	96	0	4.52	4.15	1720.32	6678.42	6242.91	21.16	46.58	28.91	5.18
5.20	...est_v7_r12_vr5_c1_s29826	10	896.50	oot	oot	0.00	1.00	2.6	19	87	80	0	4.56	4.14	1673.47	6087.47	5743.79	0.18	44.41	36.23	5.19
5.21	...est_v7_r17_vr1_c1_s23882	0	oot	oot	oot	0.00	1.00	3.1	10	64	64	0	4.38	4.19	1267.66	6651.62	6198.45	33.13	57.17	41.57	5.20
5.22	...est_v7_r17_vr1_c1_s30331	0	oot	oot	oot	0.00	1.00	3.0	10	64	64	0	4.29	4.17	1248.26	6663.36	6206.71	30.16	57.64	40.88	5.21
5.23	...fyInterleavedModMult-128	0	oot	oot	oot	0.00	1.00	2.6	10	62	62	0	0.14	1.04	1232.19	6712.69	6176.87	34.62	49.66	40.44	5.22
5.24	...ffyInterleavedModMult-32	0	oot	oot	oot	0.00	1.00	1.1	60	442	442	0	0.25	1.77	1686.85	6768.16	6507.57	2.55	13.04	3.33	5.23
5.25	...iffyInterleavedModMult-8	0	oot	oot	oot	0.00	1.00	0.1	114	869	170	0	0.14	0.52	1780.19	6792.16	6777.03	0.00	8.88	2.70	5.24

Continued on next page

Table C.3 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental)																cmp
				base-ninc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-ninc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.26	lfsr_lfsr_008_031_064	0	oot	oot	oot	0.00	1.00	3.2	20	41	41	0	4.70	11.48	1627.84	6812.77	4725.66	8.61	31.70	21.56	5.25
5.27	lfsr_lfsr_008_063_096	0	oot	oot	oot	0.00	1.00	5.3	14	6	6	0	4.41	12.41	1741.30	5031.53	2271.44	52.98	79.25	68.93	5.26
5.28	lfsr_lfsr_008_063_128	0	oot	oot	oot	0.00	1.00	5.8	18	4	4	0	3.92	10.21	1738.61	6288.33	2639.30	76.87	99.20	93.86	5.27
5.29	log-slicing_bvmul_15	0	oot	oot	oot	0.00	1.00	0.1	2	2	2	0	0.00	0.00	0.86	1.15	1.09	0.29	0.53	0.38	5.28
5.30	log-slicing_bvmul_16	0	oot	oot	oot	0.00	1.00	0.2	4	8	8	0	0.00	0.01	1.65	4.29	4.05	0.20	0.48	0.29	5.29
5.31	log-slicing_bvmul_17	0	oot	oot	oot	0.00	1.00	0.1	2	2	2	0	0.00	0.00	1.22	1.66	1.34	0.47	0.67	0.55	5.30
5.32	log-slicing_bvmul_18	0	oot	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0.00	1.30	1.75	1.43	0.49	0.76	0.58	5.31
5.33	log-slicing_bvsdiv_19	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	0.01	0.04	6.49	19.52	17.42	0.43	2.22	0.72	5.32
5.34	log-slicing_bvsdiv_20	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	0.01	0.04	7.08	21.62	19.44	0.46	2.17	0.80	5.33
5.35	log-slicing_bvsdiv_21	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	0.01	0.04	6.62	22.19	20.38	0.42	2.44	0.82	5.34
5.36	log-slicing_bvsdiv_22	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	0.01	0.04	7.41	24.13	21.93	0.54	2.69	0.89	5.35
5.37	log-slicing_bvsdiv_23	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	0.01	0.05	9.38	27.18	24.92	0.58	3.38	1.01	5.36
5.38	log-slicing_bvsdiv_24	0	oot	oot	oot	0.00	1.00	0.3	4	14	14	0	0.01	0.04	9.67	29.41	27.12	0.61	3.42	1.09	5.37
5.39	log-slicing_bvsdiv_25	0	oot	oot	oot	0.00	1.00	0.3	4	14	14	0	0.01	0.04	10.33	31.09	29.06	0.57	3.59	1.15	5.38
5.40	log-slicing_bvslt_12430	0	oot	oot	oot	0.00	1.00	2.2	4	12	12	0	4.26	0.40	261.98	1037.10	1003.91	29.50	76.52	45.09	5.39
5.41	log-slicing_bvslt_18374	0	oot	oot	oot	0.00	1.00	6.4	4	14	14	0	38.68	1.03	785.43	3061.27	2947.61	39.65	236.65	113.38	5.40
5.42	log-slicing_bvslt_19117	0	oot	oot	oot	0.00	1.00	6.4	4	14	14	0	38.22	1.02	584.57	2395.10	2273.97	37.38	236.63	88.71	5.41
5.43	log-slicing_bvslt_19860	0	oot	oot	oot	0.00	1.00	6.4	4	14	14	0	38.88	1.01	609.61	2217.54	2093.11	34.05	257.06	82.13	5.42
5.44	log-slicing_bvsmod_15	0	oot	oot	oot	0.00	1.00	0.2	11	42	25	21	0.03	0.05	1787.81	4840.09	4837.93	0.23	613.82	39.35	5.43
5.45	log-slicing_bvsmod_16	0	oot	oot	oot	0.00	1.00	0.1	9	44	30	2	0.02	0.05	1744.87	2249.19	2247.21	0.02	1735.08	22.95	5.44
5.46	log-slicing_bvsmod_17	0	oot	oot	oot	0.00	1.00	0.3	9	45	31	7	0.03	0.14	101.89	3933.98	3926.09	0.04	1129.78	36.43	5.45
5.47	log-slicing_bvsmod_18	0	oot	oot	oot	0.00	1.00	0.1	4	15	15	0	0.01	0.04	5.93	19.56	17.57	0.35	1.96	0.67	5.46
5.48	log-slicing_bvsmod_19	0	oot	oot	oot	0.00	1.00	0.1	4	15	15	0	0.01	0.05	6.76	22.49	20.22	0.46	2.43	0.78	5.47
5.49	log-slicing_bvsmod_20	0	oot	oot	oot	0.00	1.00	0.2	4	15	15	0	0.01	0.05	8.11	26.50	24.22	0.47	2.72	0.91	5.48
5.50	log-slicing_bvsrem_16	0	oot	oot	oot	0.00	1.00	0.2	7	28	18	5	0.01	0.03	543.20	3605.90	3604.60	0.02	1161.35	56.34	5.49
5.51	log-slicing_bvsrem_17	0	oot	oot	oot	0.00	1.00	0.3	6	27	21	4	0.02	0.08	8.86	3855.18	3851.01	0.04	957.54	62.18	5.50
5.52	log-slicing_bvsrem_18	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	0.01	0.05	6.54	21.65	19.71	0.20	1.88	0.80	5.51
5.53	log-slicing_bvsrem_19	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	0.01	0.04	7.44	23.05	20.86	0.43	2.13	0.85	5.52
5.54	log-slicing_bvsrem_20	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	0.01	0.04	8.28	25.27	23.03	0.46	2.55	0.94	5.53
5.55	log-slicing_bvsub_12973	0	oot	oot	oot	0.00	1.00	0.5	1	1	1	0	0.00	0.00	320.98	318.33	309.77	318.33	318.33	318.33	5.54
5.56	log-slicing_bvsub_13970	0	oot	oot	oot	0.00	1.00	0.2	1	1	1	0	0.00	0.00	394.29	391.48	381.59	391.48	391.48	391.48	5.55
5.57	log-slicing_bvsub_14967	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0.00	313.96	310.98	299.81	310.98	310.98	310.98	5.56
5.58	log-slicing_bvsub_15964	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	373.46	366.93	353.83	366.93	366.93	366.93	5.57
5.59	log-slicing_bvsub_16961	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	574.48	567.70	552.85	567.70	567.70	567.70	5.58
5.60	log-slicing_bvsub_17958	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	392.68	385.63	369.24	385.63	385.63	385.63	5.59
5.61	log-slicing_bvsub_18955	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	499.98	491.91	474.14	491.91	491.91	491.91	5.60
5.62	log-slicing_bvsub_19952	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0.00	532.12	522.83	501.88	522.83	522.83	522.83	5.61
5.63	log-slicing_bvsub_20949	0	oot	oot	oot	0.00	1.00	0.5	1	1	1	0	0.00	0.00	675.51	665.25	643.23	665.25	665.25	665.25	5.62
5.64	log-slicing_bvsub_21946	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0.00	894.40	882.86	858.74	882.86	882.86	882.86	5.63
5.65	log-slicing_bvsub_22943	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0.00	828.12	815.55	786.23	815.55	815.55	815.55	5.64
5.66	log-slicing_bvsub_23940	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0.00	722.11	708.54	676.60	708.54	708.54	708.54	5.65
5.67	log-slicing_bvudiv_18	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	2.90	5.80	5.48	0.94	1.93	1.16	5.66
5.68	log-slicing_bvudiv_19	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	3.64	7.22	6.89	1.20	2.27	1.44	5.67
5.69	log-slicing_bvudiv_20	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	4.17	8.44	8.10	1.41	2.34	1.69	5.68
5.70	log-slicing_bvudiv_21	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	4.30	8.80	8.45	1.48	2.67	1.76	5.69
5.71	log-slicing_bvudiv_22	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	4.57	8.65	8.31	1.35	2.68	1.73	5.70
5.72	log-slicing_bvudiv_23	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	4.85	9.77	9.41	1.58	3.14	1.95	5.71
5.73	log-slicing_bvudiv_24	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	5.21	10.50	10.15	1.75	3.36	2.10	5.72
5.74	log-slicing_bvudiv_25	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	5.60	11.60	11.24	1.97	3.43	2.32	5.73
5.75	log-slicing_bvult_16967	0	oot	oot	oot	0.00	1.00	6.1	3	6	6	0	19.05	0.48	399.52	938.41	874.55	39.68	190.66	85.31	5.74
5.76	log-slicing_bvult_17964	0	oot	oot	oot	0.00	1.00	6.1	3	6	6	0	18.15	0.48	456.09	928.81	864.60	45.83	218.25	84.44	5.75

Continued on next page

Table C.3 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental)																cmp
				base-ninc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-ninc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.77	log-slicing_bvult_18961	0	oot	oot	oot	0.00	1.00	6.1	3	6	6	0	18.62	0.47	559.09	1043.35	978.65	38.46	287.99	94.85	5.76
5.78	log-slicing_bvult_19958	0	oot	oot	oot	0.00	1.00	6.1	3	6	6	0	21.17	0.48	449.94	873.85	808.30	42.56	268.20	79.44	5.77
5.79	log-slicing_bvult_20955	0	oot	oot	oot	0.00	1.00	6.1	3	6	6	0	18.78	0.49	556.07	1148.90	1084.27	54.40	268.79	104.45	5.78
5.80	log-slicing_bvult_21952	0	oot	oot	oot	0.00	1.00	6.1	3	6	6	0	18.82	0.47	555.47	991.54	926.11	40.56	276.67	90.14	5.79
5.81	log-slicing_bvult_22949	0	oot	oot	oot	0.00	1.00	6.1	3	6	6	0	18.77	0.49	698.00	1359.63	1293.56	40.50	317.99	123.60	5.80
5.82	log-slicing_bvult_23946	0	oot	oot	oot	0.00	1.00	6.2	3	6	6	0	18.82	0.49	522.26	1207.61	1144.19	43.67	317.98	109.78	5.81
5.83	log-slicing_bvult_24943	0	oot	oot	oot	0.00	1.00	6.2	3	6	6	0	19.20	0.49	589.70	1221.96	1154.48	48.78	300.43	111.09	5.82
5.84	log-slicing_bvult_25940	0	oot	oot	oot	0.00	1.00	6.2	3	6	6	0	18.80	0.48	632.20	1302.69	1240.60	52.34	273.86	118.43	5.83
5.85	log-slicing_bvurem_15	0	oot	oot	oot	0.00	1.00	0.1	3	3	3	2	0.00	0.00	587.80	1767.65	1767.42	0.88	1177.13	252.52	5.84
5.86	log-slicing_bvurem_16	0	oot	oot	oot	0.00	1.00	0.2	7	25	16	0	0.01	0.03	14.58	43.13	41.88	0.10	3.03	0.88	5.85
5.87	log-slicing_bvurem_17	0	oot	oot	oot	0.00	1.00	0.2	6	24	14	0	0.02	0.07	11.13	34.09	31.34	0.31	2.89	0.73	5.86
5.88	log-slicing_bvurem_18	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	4.75	9.58	9.22	1.59	2.99	1.92	5.87
5.89	log-slicing_bvurem_19	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	5.40	10.96	10.59	1.86	3.39	2.19	5.88
5.90	log-slicing_bvurem_20	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	5.87	11.89	11.51	1.95	3.72	2.38	5.89
5.91	log-slicing_bvurem_21	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	6.95	13.80	13.42	2.26	4.51	2.76	5.90
5.92	log-slicing_bvurem_22	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	0.00	0.01	7.19	14.38	14.00	2.35	4.56	2.88	5.91
5.93	pspace_power2sum.8087	0	oot	oot	oot	0.00	1.00	0.8	1	1	1	0	0.00	0.00	707.88	707.52	706.89	707.52	707.52	707.52	5.92
5.94	pspace_power2sum.9679	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0.00	187.66	187.59	186.73	187.59	187.59	187.59	5.93
5.95	RWS_Example_15.txt	0	oot	oot	oot	0.00	1.00	0.4	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.94
5.96	RWS_Example_19.txt	0	oot	oot	oot	0.00	1.00	0.6	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.95
5.97	RWS_Example_20.txt	0	oot	oot	oot	0.00	1.00	0.8	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.96
5.98	RWS_Example_7.txt	0	oot	oot	oot	0.00	1.00	0.1	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.97
5.99	uum_uum12	0	oot	oot	oot	0.00	1.00	0.5	241	1865	309	0	3.08	4.50	1788.38	6911.90	6505.36	0.00	6.73	0.95	5.98
5.100	uum_uum16	0	oot	oot	oot	0.00	1.00	1.2	132	1035	453	0	2.96	6.22	1781.07	6950.47	6496.81	0.01	9.48	1.33	5.99
5.101	uum_uum20	0	oot	oot	oot	0.00	1.00	1.9	79	603	280	0	6.16	12.16	1776.80	6860.89	6261.75	0.23	11.99	2.63	5.100
5.102	uum_uum24	0	oot	oot	oot	0.00	1.00	2.9	57	283	283	0	3.93	9.21	1771.86	6942.09	6375.98	0.62	16.47	3.89	5.101
5.103	uum_uum28	0	oot	oot	oot	0.00	1.00	2.9	44	159	159	0	4.39	9.79	1783.65	6954.40	6329.91	1.02	15.90	6.23	5.102
5.104	uum_uum32	0	oot	oot	oot	0.00	1.00	2.9	41	153	153	0	3.91	9.48	1773.66	6887.25	6333.17	1.17	27.93	6.50	5.103
5.105	VS3_VS3-benchmark-A10	0	oot	oot	oot	0.00	1.00	0.6	70	544	497	0	0.70	0.70	1764.14	7064.09	6823.83	0.46	7.30	2.84	5.104
5.106	VS3_VS3-benchmark-A11	0	oot	oot	oot	0.00	1.00	0.5	60	463	393	0	0.55	0.56	1631.73	7015.80	6813.47	0.37	9.24	3.26	5.105
5.107	VS3_VS3-benchmark-A12	0	oot	oot	oot	0.00	1.00	0.6	70	544	452	0	0.75	0.70	1657.70	7035.71	6789.37	0.46	8.05	2.75	5.106
5.108	VS3_VS3-benchmark-A13	0	oot	oot	oot	0.00	1.00	0.6	53	408	384	0	0.58	0.61	1784.52	7033.17	6802.33	2.55	11.65	3.82	5.107
5.109	VS3_VS3-benchmark-A8	0	oot	oot	oot	0.00	1.00	0.6	60	464	410	0	0.65	0.63	1630.52	7055.29	6834.16	2.61	7.81	3.41	5.108
5.110	VS3_VS3-benchmark-S2	0	oot	oot	oot	0.00	1.00	0.6	20	136	136	0	0.19	0.15	1570.38	6716.02	6544.96	15.78	37.56	18.10	5.109
5.111	...1_alt_true.BV.c.cil.c.17	0	oot	oot	oot	0.00	1.00	2.4	31	171	140	0	0.69	2.22	1724.11	6806.75	6517.76	0.03	34.85	9.53	5.110
5.112	...1_alt_true.BV.c.cil.c.21	0	oot	oot	oot	0.00	1.00	3.1	26	134	133	0	0.70	2.50	1701.52	6714.53	6403.87	7.41	70.88	13.06	5.111
5.113	brummayerbiere2_smulov1bw48	0	oot	oot	oot	0.00	1.00	0.2								0.00	0.00	0.00	0.00	0.00	5.112
5.114	...erbiere3_isqrtaddeqcheck	0	oot	oot	oot	0.00	1.00	0.1	50	375	241	0	0.09	0.06	1261.79	6852.99	6810.59	0.04	28.43	4.83	5.113
5.115	...mayerbiere3_isqrtaddnoif	0	oot	oot	oot	0.00	1.00	0.1	60	458	184	0	0.08	0.05	1665.91	6836.62	6817.89	0.00	27.82	4.86	5.114
5.116	...mayerbiere3_isqrteqcheck	0	oot	oot	oot	0.00	1.00	0.1	48	368	198	0	0.09	0.06	1779.24	6862.90	6825.97	0.00	31.57	5.70	5.115
5.117	brummayerbiere3_isqrtnoif	0	oot	oot	oot	0.00	1.00	0.0	142	1076	131	0	0.14	0.11	1799.01	6893.83	6870.86	0.00	16.30	2.31	5.116
5.118	brummayerbiere3_maxor128	0	oot	oot	oot	0.00	1.00	0.2	20	21	3	18	0.06	0.03	1736.29	1956.63	1943.57	2.58	131.18	33.16	5.117
5.119	brummayerbiere3_maxor256	0	oot	oot	oot	0.00	1.00	0.6	19	20	3	17	0.10	0.07	1750.52	2311.10	2276.16	2.69	142.83	41.27	5.118
5.120	brummayerbiere3_maxxor064	0	oot	oot	oot	0.00	1.00	0.3	94	736	593	0	0.33	0.25	1759.31	6988.37	6814.40	0.18	29.82	1.53	5.119
5.121	brummayerbiere3_maxxor128	0	oot	oot	oot	0.00	1.00	0.5	50	384	384	0	0.37	0.24	1793.57	6953.24	6631.81	3.32	8.34	4.89	5.120
5.122	brummayerbiere3_maxxor256	0	oot	oot	oot	0.00	1.00	1.2	17	120	120	0	0.22	0.17	1740.35	6760.72	6428.44	14.98	34.03	24.23	5.121
5.123	...biere3_maxxormaxorand032	0	oot	oot	oot	0.00	1.00	0.3								0.00	0.00	0.00	0.00	0.00	5.122
5.124	...biere3_maxxormaxorand064	0	oot	oot	oot	0.00	1.00	0.7	91	712	712	0	1.02	0.71	1792.47	6830.70	6281.78	1.01	16.21	1.44	5.123
5.125	...biere3_maxxormaxorand128	0	oot	oot	oot	0.00	1.00	0.8	33	248	248	0	0.72	0.64	1788.47	6649.18	6200.16	4.96	35.64	7.68	5.124
5.126	...biere3_maxxormaxorand256	0	oot	oot	oot	0.00	1.00	2.0	11	72	72	0	0.60	0.55	1693.57	6558.75	6139.36	23.58	73.49	35.45	5.125
5.127	...yerbiere3_minandmaxor128	0	oot	oot	oot	0.00	1.00	0.4	62	469	26	0	0.77	0.52	1751.11	6099.52	5914.26	0.06	29.84	6.30	5.126

Continued on next page

Table C.3 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental)																cmp
				base-ninc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-ninc-flr
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.128	...yerbiere3_minandmaxor256	0	oot	oot	oot	0.00	1.00	1.8	18	117	19	0	0.47	0.39	1785.74	5835.12	5500.65	1.32	82.37	26.28	5.127
5.129	brummayerbiere3_minxor128	0	oot	oot	oot	0.00	1.00	0.7	90	704	704	0	0.71	0.57	1584.47	6888.29	6360.19	1.03	18.61	1.49	5.128
5.130	brummayerbiere3_minxor256	0	oot	oot	oot	0.00	1.00	0.7	29	216	216	0	0.47	0.40	1769.95	6467.91	6215.45	7.10	42.24	10.73	5.129
5.131	...erbiere3_minxorminand064	0	oot	oot	oot	0.00	1.00	0.5	70	544	419	0	0.82	0.56	1197.48	6887.99	6556.75	0.06	18.16	2.57	5.130
5.132	...erbiere3_minxorminand128	0	oot	oot	oot	0.00	1.00	0.8	30	224	224	0	0.66	0.60	1369.90	6897.73	6552.70	7.09	29.31	8.49	5.131
5.133	...erbiere3_minxorminand256	0	oot	oot	oot	0.00	1.00	1.7	10	64	64	0	0.54	0.54	1464.50	6647.01	6394.35	43.67	80.36	48.88	5.132
5.134	brummayerbiere3_mulhs16	0	oot	oot	oot	0.00	1.00	0.2	20	137	92	1	0.01	0.01	23.24	87.59	85.25	0.00	1.33	0.23	5.133
5.135	brummayerbiere3_mulhs32	0	oot	oot	oot	0.00	1.00	0.2	18	121	86	2	0.01	0.01	66.37	239.44	231.92	0.02	6.70	0.85	5.134
5.136	brummayerbiere3_mulhs64	0	oot	oot	oot	0.00	1.00	0.2	18	125	101	0	0.01	0.01	292.29	1035.61	1001.74	0.09	8.86	3.77	5.135
5.137	...erbiere4_unconstrained02	0	oot	oot	oot	0.00	1.00	5.2	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.136
5.138	brummayerbiere_countbits128	0	oot	oot	oot	0.00	1.00	0.3	50	381	380	0	0.29	0.15	1510.24	6971.34	6738.73	0.00	20.70	4.14	5.137
5.139	brummayerbiere_countbits256	0	oot	oot	oot	0.00	1.00	0.7	20	141	140	0	0.27	0.11	1527.18	6825.42	6539.01	0.01	38.78	18.70	5.138
5.140	brummayerbiere_countbits512	0	oot	oot	oot	0.00	1.00	2.3	10	61	60	0	0.27	0.12	1530.49	6470.14	6051.34	0.02	81.79	51.76	5.139
5.141	...biere_countbitsrotate032	0	oot	oot	oot	0.00	1.00	0.1	51	391	285	0	0.25	0.16	1766.26	6947.97	6930.12	0.00	29.44	4.30	5.140
5.142	...biere_countbitsrotate064	0	oot	oot	oot	0.00	1.00	0.9	57	434	432	0	1.16	0.84	1795.59	6754.01	6650.05	0.02	15.73	3.65	5.141
5.143	...biere_countbitsrotate128	0	oot	oot	oot	0.00	1.00	1.3	18	128	128	0	3.90	1.82	1716.40	6507.39	6401.10	7.33	52.98	21.76	5.142
5.144	...biere_countbitsrotate256	0	oot	oot	oot	0.00	1.00	3.9	10	58	58	0	35.43	4.44	1236.87	6307.59	5953.80	34.24	100.14	37.77	5.143
5.145	...yerbiere_countbitssrl064	0	oot	oot	oot	0.00	1.00	0.1	47	294	24	21	0.11	0.06	1743.05	5340.22	5318.59	0.00	445.67	8.32	5.144
5.146	...yerbiere_countbitssrl128	0	oot	oot	oot	0.00	1.00	0.4	51	331	23	39	0.26	0.12	1751.69	4219.96	4157.64	0.00	228.35	5.67	5.145
5.147	...yerbiere_countbitssrl256	0	oot	oot	oot	0.00	1.00	1.4	21	141	43	0	0.24	0.10	1701.42	5947.15	5726.95	0.00	69.56	18.64	5.146
5.148	challenge_integerOverflow	0	oot	oot	oot	0.00	1.00	0.2	5	12	8	2	0.00	0.00	17.88	2023.50	2022.71	0.03	1066.96	80.94	5.147
5.149	...erbiere4_unconstrained08	0	oom	oom	oom	-0.05	1.00	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.148
5.150	...erbiere4_unconstrained09	0	oom	oom	oom	-0.09	1.00	6.9	0	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.5

Table C.4: Detailed results of PBoolector in non-incremental modes with FLR (Parallel-Non-Incremental-FLR) compared to the reference times Base-Non-Incremental. Last column par-ninc contains cross reference to detailed results of Parallel-Non-Incremental in Tab.C.3.

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental-FLR)																	cmp
				par-ninc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-ninc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
2.1	log-slicing_bvmul_14	20	1477.50	1496.45	884.50	611.95	1.69	0.1	2	2	2	2	0.00	0	0	883.99	1727.43	1727.34	0.27	883.32	345.49	2.1
2.2	core_ext_con_048_002_1024	20	487.72	492.48	127.69	364.79	3.86	0.0	0	1	1	0	127.22	47	20	0.00	0.00	0.00	0.00	0.00	0.00	3.11
2.3	log-slicing_bvsmod_14	20	1414.54	1430.98	1148.52	282.46	1.25	0.1	13	42	25	39	1.25	0	0	1146.62	2999.07	2996.55	0.21	233.27	19.86	2.2
2.4	log-slicing_bvudiv_17	20	1270.36	1291.07	1051.94	239.13	1.23	0.2	5	9	6	6	4.07	0	0	1047.29	2923.98	2922.73	0.33	623.21	127.13	2.3
2.5	core_ext_con_036_002_1024	20	269.99	268.42	52.29	216.13	5.13	0.0	0	1	1	0	51.82	35	20	0.00	0.00	0.00	0.00	0.00	0.00	3.14
2.6	log-slicing_bvmul_13	20	428.63	431.54	291.21	140.33	1.48	0.1	2	2	2	2	0.00	0	0	290.69	538.92	538.84	0.24	289.94	107.78	2.6
2.7	...ayerbiere2_smulov4bw1024	10	1518.87	1499.29	1364.74	134.55	1.10	1.6	3	3	2	2	0.04	1	0	1364.13	1945.92	1883.51	6.84	1161.62	277.99	2.5
2.8	challenge_multiplyOverflow	20	1035.61	1041.44	928.19	113.25	1.12	0.1	2	2	2	2	0.00	0	0	927.67	1427.64	1427.52	4.08	917.33	285.53	2.7
2.9	core_ext_con_044_002_1024	20	195.82	197.33	98.67	98.66	2.00	0.0	0	1	1	0	98.17	43	20	0.00	0.00	0.00	0.00	0.00	0.00	3.10
2.10	...biere3_isqrtaddinvalidvc	10	185.60	181.99	86.93	95.06	2.09	0.0	30	209	179	0	0.16	0	0	86.20	330.49	316.78	0.02	2.62	0.44	2.11
2.11	core_ext_con_064_001_1024	20	128.25	129.32	43.94	85.38	2.94	0.0	0	1	1	0	43.43	63	20	0.00	0.00	0.00	0.00	0.00	0.00	3.19
2.12	core_ext_con_056_002_1024	20	279.98	284.33	205.26	79.07	1.39	0.0	0	1	1	0	204.78	55	20	0.00	0.00	0.00	0.00	0.00	0.00	3.12
2.13	core_ext_con_060_001_1024	20	110.60	111.42	36.08	75.34	3.09	0.0	0	1	1	0	35.62	59	20	0.00	0.00	0.00	0.00	0.00	0.00	3.17
2.14	log-slicing_bvurem_14	20	757.16	767.68	700.57	67.11	1.10	0.1	3	3	3	3	0.70	0	0	699.35	1003.64	1003.47	0.77	523.79	125.45	2.8
2.15	...yerbiere3_isqrtinvalidvc	10	181.66	183.06	124.53	58.53	1.47	0.0	30	222	141	0	0.16	0	0	123.78	476.18	463.62	0.02	9.71	0.70	2.9
2.16	log-slicing_bvmul_12	20	129.19	131.36	83.08	48.28	1.58	0.0	2	2	2	2	0.00	0	0	82.48	137.48	137.40	0.22	81.92	27.50	2.10
2.17	core_ext_con_052_002_1024	20	194.60	205.80	166.08	39.72	1.24	0.0	0	1	1	0	165.52	51	20	0.00	0.00	0.00	0.00	0.00	0.00	3.9
2.18	core_ext_con_064_002_1024	20	344.29	330.81	313.33	17.48	1.06	0.0	0	1	1	0	312.84	63	20	0.00	0.00	0.00	0.00	0.00	0.00	3.16
2.19	log-slicing_bvudiv_15	20	162.70	166.23	150.61	15.62	1.10	0.0	3	3	3	3	0.68	0	0	149.40	198.71	198.55	0.59	98.35	24.84	2.12
2.20	log-slicing_bvsmod_12	20	138.58	141.04	129.38	11.66	1.09	0.1	13	42	25	39	1.25	0	0	127.46	338.60	336.35	0.17	23.12	2.24	2.13
2.21	log-slicing_bvurem_13	20	147.97	149.47	138.78	10.69	1.08	0.0	3	3	3	3	0.69	0	0	137.43	187.29	187.13	0.64	87.44	23.41	2.14
2.22	brummayerbiere3_isqrt	20	145.81	146.46	143.80	2.66	1.02	0.0	32	219	162	0	0.47	0	0	142.76	531.17	516.94	0.01	9.78	0.72	3.7
2.23	log-slicing_bvadd_26192	20	202.52	207.67	206.17	1.50	1.01	0.3	0	1	1	0	0.00	0	0	205.66	205.48	190.49	205.48	205.48	205.48	2.16
2.24	log-slicing_bvadd_19983	20	121.59	124.66	123.62	1.04	1.01	0.2	0	1	1	0	0.00	0	0	123.11	118.95	112.99	118.95	118.95	118.95	2.22
2.25	log-slicing_bvadd_23531	20	167.45	173.18	172.18	1.00	1.01	0.2	0	1	1	0	0.00	0	0	171.61	165.03	154.56	165.03	165.03	165.03	2.17
2.26	log-slicing_bvadd_29740	20	264.93	273.86	273.00	0.86	1.00	0.3	0	1	1	0	0.00	0	0	272.53	271.40	244.64	271.40	271.40	271.40	2.20
2.27	log-slicing_bvadd_27966	20	232.01	236.99	236.17	0.82	1.00	0.3	0	1	1	0	0.00	0	0	235.71	235.52	215.76	235.52	235.52	235.52	2.15
2.28	log-slicing_bvadd_21757	20	165.07	169.85	169.19	0.66	1.00	0.2	0	1	1	0	0.00	0	0	168.66	163.27	156.00	163.27	163.27	163.27	2.18
2.29	log-slicing_bvadd_20870	20	132.00	135.08	134.44	0.64	1.00	0.2	0	1	1	0	0.00	0	0	133.88	129.16	122.48	129.16	129.16	129.16	2.24
2.30	pspace_shift1add_24955	20	128.47	131.62	131.20	0.42	1.00	0.1	0	1	1	0	0.00	0	0	130.67	130.59	130.04	130.59	130.59	130.59	2.26
2.31	brummayerbiere_bitrev8192	20	101.97	101.56	101.19	0.37	1.00	0.0	0	1	1	0	0.00	0	0	0.05	0.04	0.00	0.04	0.04	0.04	2.27
2.32	pspace_shift1add_27946	20	167.81	172.13	171.81	0.32	1.00	0.2	0	1	1	0	0.00	0	0	171.28	171.19	170.60	171.19	171.19	171.19	2.31
2.33	pspace_shift1add_23958	20	114.08	116.81	116.52	0.29	1.00	0.1	0	1	1	0	0.00	0	0	115.98	115.90	115.39	115.90	115.90	115.90	2.36
2.34	pspace_shift1add_28943	20	186.84	191.46	191.21	0.25	1.00	0.1	0	1	1	0	0.00	0	0	190.73	190.63	190.02	190.63	190.63	190.63	2.32
2.35	pspace_shift1add_26949	20	152.33	156.05	155.82	0.23	1.00	0.1	0	1	1	0	0.00	0	0	155.34	155.25	154.67	155.25	155.25	155.25	2.34
2.36	pspace_shift1add_29940	20	195.38	200.45	200.22	0.23	1.00	0.1	0	1	1	0	0.00	0	0	199.74	199.63	199.00	199.63	199.63	199.63	2.30
2.37	log-slicing_bvadd_24418	20	198.09	204.18	203.96	0.22	1.00	0.3	0	1	1	0	0.00	0	0	203.44	203.22	191.50	203.22	203.22	203.22	2.19
2.38	pspace_shift1add_25952	20	143.28	146.71	146.49	0.22	1.00	0.1	0	1	1	0	0.00	0	0	145.94	145.85	145.29	145.85	145.85	145.85	2.28
2.39	log-slicing_bvadd_25305	20	186.34	190.48	190.29	0.19	1.00	0.3	0	1	1	0	0.00	0	0	189.80	189.62	177.22	189.62	189.62	189.62	2.21
2.40	log-slicing_bvadd_19096	20	109.78	113.21	113.04	0.17	1.00	0.2	0	1	1	0	0.00	0	0	112.56	109.06	103.23	109.06	109.06	109.06	2.29
2.41	pspace_shift1add_22961	20	110.71	113.32	113.15	0.17	1.00	0.1	0	1	1	0	0.00	0	0	112.50	112.42	111.94	112.42	112.42	112.42	2.25
2.42	log-slicing_bvsub_05994	20	414.31	415.57	415.52	0.05	1.00	0.1	0	1	1	0	0.00	0	0	414.99	414.12	412.13	414.12	414.12	414.12	2.33
3.1	log-slicing_bvsub_04000	20	545.84	545.62	545.68	-0.06	1.00	0.1	0	1	1	0	0.00	0	0	545.11	544.68	543.73	544.68	544.68	544.68	3.1
3.2	bmc-bv_graycode	10	442.54	448.89	449.14	-0.25	1.00	0.3	0	1	1	0	0.21	1	0	0.17	0.17	0.04	0.17	0.17	0.17	3.2

Continued on next page

Table C.4 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental-FLR)																		cmp
				par-ninc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-ninc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
3.3	log-slicing_bvadd_22644	20	156.03	165.93	166.20	-0.27	1.00	0.2	0	1	1	0	0.00	0	0	165.67	159.67	151.47	159.67	159.67	159.67	3.3
3.4	log-slicing_bvadd_27079	20	216.41	220.70	221.10	-0.40	1.00	0.3	0	1	1	0	0.00	0	0	220.58	220.40	203.29	220.40	220.40	220.40	3.5
3.5	log-slicing_bvadd_18209	20	101.26	102.51	103.32	-0.81	0.99	0.2	0	1	1	0	0.00	0	0	102.82	99.94	94.44	99.94	99.94	99.94	2.23
3.6	log-slicing_bvadd_28853	20	245.33	251.65	253.28	-1.63	0.99	0.3	0	1	1	0	0.00	0	0	252.68	252.48	230.17	252.48	252.48	252.48	3.6
3.7	bmc-bv_magic	10	443.86	453.08	477.93	-24.85	0.95	0.3	0	1	1	0	25.00	1	0	0.61	0.60	0.46	0.60	0.60	0.60	3.4
3.8	log-slicing_bvsmod_13	20	360.26	371.81	396.69	-24.88	0.94	0.1	13	42	25	39	1.23	0	0	394.84	1172.38	1170.09	0.19	71.04	7.76	3.8
3.9	core_ext_con_060_002_1024	20	220.71	220.86	254.81	-33.95	0.87	0.0	0	1	1	0	254.32	59	20	0.00	0.00	0.00	0.00	0.00	0.00	3.13
3.10	brummayerbiere3_isqrtadd	20	144.73	153.58	207.18	-53.60	0.74	0.0	37	255	158	0	0.47	0	0	206.14	742.94	723.79	0.02	9.34	0.91	3.15
3.11	RWS_Example_18.txt	10	160.61	160.56	222.87	-62.31	0.72	0.1	0	1	1	0	62.04	0	0	160.19	160.10	159.88	160.10	160.10	160.10	2.35
3.12	log-slicing_bvudiv_16	20	390.99	398.23	470.25	-72.02	0.85	0.1	7	15	6	6	0.68	0	0	468.99	1420.41	1419.69	0.01	337.03	40.58	3.18
3.13	...yerbiere_countbitssrl032	20	387.06	395.11	485.02	-89.91	0.81	0.1	40	164	19	42	1.28	2	0	483.07	1042.74	1036.40	0.00	150.72	2.63	2.4
3.14	...ayerbiere2_smulov4bw0512	10	128.78	130.66	224.74	-94.08	0.58	0.4	3	3	2	2	0.02	1	0	224.10	350.27	340.71	0.80	98.44	50.04	3.20
3.15	pspace_power2sum.5898	20	343.57	351.34	451.33	-99.99	0.78	0.3	1	1	1	1	0.00	0	0	450.80	450.56	449.76	104.25	346.30	225.28	3.21
3.16	pspace_power2sum.5699	20	347.92	348.90	456.52	-107.62	0.76	0.3	1	1	1	1	0.00	0	0	456.01	455.78	455.02	107.76	348.02	227.89	3.22
3.17	log-slicing_bvsub_06991	20	307.68	315.94	425.52	-109.58	0.74	0.1	1	1	1	1	0.00	0	0	425.01	423.55	418.29	110.83	312.72	211.77	3.23
3.18	pspace_power2sum.6097	20	388.12	387.37	506.73	-119.36	0.76	0.4	1	1	1	1	0.00	0	0	506.18	505.93	505.09	117.22	388.71	252.96	3.24
3.19	pspace_power2sum.6495	20	504.51	505.78	626.21	-120.43	0.81	0.4	1	1	1	1	0.00	0	0	625.62	625.34	624.43	120.54	504.80	312.67	3.25
3.20	pspace_power2sum.8684	20	1028.30	1025.18	1155.19	-130.01	0.89	0.4	1	1	1	1	0.00	0	0	1154.70	1154.58	1153.11	115.58	1039.00	577.29	3.26
3.21	log-slicing_bvsub_10979	20	161.21	145.19	275.94	-130.75	0.53	0.2	1	1	1	1	0.00	0	0	275.46	269.62	257.14	123.37	146.25	134.81	3.27
3.22	log-slicing_bvsub_07988	20	321.69	327.33	471.25	-143.92	0.69	0.1	1	1	1	1	0.00	0	0	470.79	467.54	460.69	139.50	328.04	233.77	3.28
3.23	pspace_power2sum.5500	20	496.16	493.29	642.91	-149.62	0.77	0.4	1	1	1	1	0.00	0	0	642.33	642.12	641.40	150.89	491.22	321.06	3.29
3.24	log-slicing_bvsub_08985	20	378.56	390.79	554.88	-164.09	0.70	0.1	1	1	1	1	0.00	0	0	554.42	550.76	542.28	162.20	388.57	275.38	3.30
3.25	pspace_power2sum.6296	20	434.76	435.13	600.55	-165.42	0.72	0.4	1	1	1	1	0.00	0	0	599.99	599.72	598.85	156.97	442.75	299.86	3.31
3.26	pspace_power2sum.7092	20	452.50	454.24	640.66	-186.42	0.71	0.4	1	1	1	1	0.00	0	0	640.12	639.61	638.59	179.52	460.10	319.81	3.33
3.27	pspace_power2sum.7689	20	515.93	517.58	716.25	-198.67	0.72	0.4	1	1	1	1	0.00	0	0	715.75	715.10	713.94	199.67	515.42	357.55	3.34
3.28	pspace_power2sum.9480	20	751.83	748.66	956.73	-208.07	0.78	0.5	1	1	1	1	0.00	0	0	956.23	956.10	954.44	208.71	747.39	478.05	3.35
3.29	pspace_power2sum.9878	20	573.35	578.37	792.22	-213.85	0.73	0.4	1	1	1	1	0.00	0	0	791.67	791.54	789.78	218.62	572.92	395.77	3.36
3.30	log-slicing_bvsub_11976	20	664.63	696.11	927.12	-231.01	0.75	0.4	1	1	1	1	0.00	0	0	926.61	919.84	905.47	247.80	672.03	459.92	3.37
3.31	pspace_power2sum.8485	20	872.30	872.26	1106.80	-234.54	0.79	0.6	1	1	1	1	0.00	0	0	1106.27	1106.15	1104.73	234.34	871.81	553.08	3.38
3.32	pspace_power2sum.9281	20	1338.53	1337.10	1572.09	-234.99	0.85	0.8	1	1	1	1	0.00	0	0	1571.56	1571.43	1569.82	234.76	1336.67	785.71	3.39
3.33	pspace_power2sum.8883	20	1035.88	1037.85	1273.71	-235.86	0.81	0.4	1	1	1	1	0.00	0	0	1273.19	1273.07	1271.55	228.69	1044.38	636.54	3.40
3.34	log-slicing_bvsub_09982	20	328.06	333.74	570.86	-237.12	0.58	0.1	1	1	1	1	0.00	0	0	570.24	565.55	555.45	235.20	330.35	282.78	3.41
3.35	pspace_power2sum.7490	20	505.69	505.54	754.07	-248.53	0.67	0.3	1	1	1	1	0.00	0	0	753.59	752.96	751.85	247.59	505.37	376.48	3.42
3.36	...ayerbiere2_smulov4bw0768	10	563.42	566.25	821.71	-255.46	0.69	1.0	3	3	2	2	0.03	1	0	821.16	1200.92	1170.89	3.10	648.70	171.56	3.43
3.37	pspace_power2sum.7291	20	971.68	970.12	1227.62	-257.50	0.79	0.5	1	1	1	1	0.00	0	0	1227.04	1226.44	1225.36	257.57	968.86	613.22	3.44
3.38	...ayerbiere2_smulov4bw0896	10	1274.52	1280.01	1566.26	-286.25	0.82	1.3	3	3	2	2	0.03	1	0	1565.67	1943.69	1899.65	4.81	1363.02	277.67	3.46
3.39	pspace_power2sum.7888	20	1145.49	1133.23	1508.42	-375.19	0.75	0.8	1	1	1	1	0.00	0	0	1507.89	1507.20	1505.97	369.43	1137.77	753.60	3.47
3.40	log-slicing_bvsrem_14	20	256.90	263.76	692.07	-428.31	0.38	0.1	7	21	16	18	1.02	0	0	690.47	1908.94	1907.65	0.20	179.90	29.37	3.50
3.41	...ayerbiere2_smulov4bw0640	10	141.93	143.77	588.97	-445.20	0.24	0.7	3	3	2	2	0.02	1	0	588.37	1018.73	1001.22	1.68	405.16	145.53	3.52
3.42	pspace_power2sum.6694	20	1005.81	1017.69	1464.70	-447.01	0.69	0.6	1	1	1	1	0.00	0	0	1464.16	1463.87	1462.92	456.76	1007.11	731.93	3.51
3.43	pspace_power2sum.8286	20	898.41	893.24	1347.46	-454.22	0.66	0.5	1	1	1	1	0.00	0	0	1346.83	1346.72	1345.33	454.63	892.08	673.36	3.53
3.44	brummayerbiere_nlzbe256	20	117.51	119.31	632.74	-513.43	0.19	1.3	15	75	31	0	66.86	2	0	565.06	1487.30	1439.84	0.07	54.39	8.65	3.49
3.45	log-slicing_bvsdiv_16	20	182.35	185.31	716.60	-531.29	0.26	0.1	10	38	19	18	1.05	0	0	714.94	1809.01	1807.28	0.02	148.07	17.91	3.54
3.46	pspace_power2sum.6893	20	1102.02	1106.41	1648.66	-542.25	0.67	0.6	1	1	1	1	0.00	0	0	1648.14	1647.83	1646.84	545.80	1102.03	823.91	3.55
3.47	log-slicing_bvult_6000	20	406.38	410.02	996.59	-586.57	0.41	0.6	5	10	9	10	65.75	0	0	929.12	2941.27	2927.74	26.00	361.67	101.42	3.56
3.48	brummayerbiere2_smulov1bw24	20	134.33	135.54	774.69	-639.15	0.17	0.0	113	836	174	0	0.26	0	0	773.60	2774.72	2753.10	0.01	8.04	1.10	3.48
3.49	brummayerbiere3_maxor064	20	209.14	209.48	1175.60	-966.12	0.18	0.1	129	129	3	109	5.99	0	0	1168.70	1391.29	1376.13	0.00	36.97	3.80	3.57
3.50	log-slicing_bvsdiv_17	20	502.64	511.08	1516.81	-1005.73	0.34	0.2	9	27	16	24	5.85	0	0	1510.21	4993.11	4987.67	0.04	334.98	58.74	3.58
3.51	log-slicing_bvult_6997	20	201.19	204.81	1374.03	-1169.22	0.15	0.7	5	10	9	10	65.81	0	0	1306.52	3696.73	3683.58	27.39	481.28	127.47	3.59
3.52	log-slicing_bvslt_5743	20	444.58	444.92	1687.85	-1242.93	0.26	0.7	7	18	14	18	66.94	1	0	1618.46	5575.37	5552.34	25.07	355.45	97.81	3.60
3.53	log-slicing_bvult_7994	20	230.10	234.30	1635.72	-1401.42	0.14	0.7	5	10	9	10	65.73	0	0	1568.30	3914.62	3900.83	27.98	618.73	134.99	3.61

Continued on next page

Table C.4 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental-FLR)																	cmp
				par-ninc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-ninc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
4.1	log-slicing_bvslt_17631	20	1750.56	1785.50	oot	-14.50	0.99	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.1
4.2	...est_v7_r12_vr5_c1_s14336	10	933.17	1778.65	oot	-21.35	0.99	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.2
4.3	...test_v5_r15_vr5_c1_s8246	20	1106.64	1531.37	oot	-268.63	0.85	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.6
4.4	log-slicing_bvslt_16888	20	1502.56	1524.13	oot	-275.87	0.85	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.7
4.5	log-slicing_bvsdiv_18	20	1485.52	1509.37	oot	-290.63	0.84	0.2	7	34	24	4	5.82	0	0	1527.25	2266.25	2261.79	0.14	877.04	30.22	4.8
4.6	...est_v5_r15_vr5_c1_s26657	20	oot	1480.60	oot	-319.40	0.82	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.9
4.7	...est_v7_r7_vr10_c1_s32506	10	1754.95	1394.41	oot	-405.59	0.77	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.10
4.8	pspace_power2sum.9082	20	1301.91	1303.66	oot	-496.34	0.72	0.7	1	1	1	0	0.00	0	0	508.82	508.76	507.98	508.76	508.76	508.76	4.12
4.9	pipe_pipe-noabs.atlas.qf_bv	20	1290.10	1298.23	oot	-501.77	0.72	0.1	98	745	44	0	1.18	1	0	1796.32	6523.42	6508.97	0.00	22.50	3.98	4.13
4.10	...st_v5_r10_vr10_c1_s21502	20	697.82	1247.16	oot	-552.84	0.69	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.15
4.11	log-slicing_bvslt_16145	20	1188.56	1208.81	oot	-591.19	0.67	2.0	4	12	12	0	265.63	1	0	500.61	1513.04	1482.26	30.32	210.79	65.78	4.16
4.12	log-slicing_bvult_15970	20	1070.63	1091.11	oot	-708.89	0.61	2.0	4	12	12	0	263.35	0	0	432.13	1230.73	1195.20	32.30	156.79	53.51	4.17
4.13	brummayerbiere2_smulov1bw32	20	1056.25	1078.07	oot	-721.93	0.60	0.1	137	1052	317	0	0.44	0	0	1786.17	6738.69	6682.10	0.02	15.68	1.70	4.18
4.14	log-slicing_bvslt_15402	20	1048.34	1064.76	oot	-735.24	0.59	2.1	4	12	12	0	267.89	1	0	505.28	1695.26	1662.58	32.42	153.77	73.71	4.19
4.15	log-slicing_bvult_14973	20	1019.50	1027.31	oot	-772.69	0.57	2.0	4	12	12	0	263.07	0	0	377.14	1132.95	1099.67	28.32	137.17	49.26	4.20
4.16	...test_v7_r7_vr5_c1_s19694	10	1370.11	999.27	oot	-800.73	0.56	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.21
4.17	VS3_VS3-benchmark-A7	10	985.79	987.88	oot	-812.12	0.55	0.4	65	481	290	0	1.14	1	0	1774.83	6968.87	6794.00	0.27	14.60	3.46	4.22
4.18	log-slicing_bvslt_13916	20	908.99	933.26	oot	-866.74	0.52	2.0	4	12	12	1	267.30	1	0	422.32	2496.40	2462.60	30.30	824.25	104.02	4.23
4.19	log-slicing_bvult_12979	20	893.49	915.23	oot	-884.77	0.51	2.0	4	12	12	0	261.81	0	0	280.63	1038.32	1002.62	34.12	73.28	45.14	4.24
4.20	log-slicing_bvult_13976	20	893.73	911.14	oot	-888.86	0.51	2.0	4	12	12	0	263.19	0	0	334.66	1093.95	1063.33	29.30	112.62	47.56	4.25
4.21	...est_v5_r15_vr5_c1_s23844	20	oot	875.10	oot	-924.90	0.49	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.26
4.22	log-slicing_bvslt_14659	20	838.99	854.35	oot	-945.65	0.47	2.0	4	12	12	2	269.93	1	0	367.75	2933.01	2898.21	30.05	1054.35	117.32	4.27
4.23	log-slicing_bvslt_13173	20	801.46	815.34	oot	-984.66	0.45	2.0	4	12	12	1	266.53	1	0	306.88	1959.85	1930.88	27.48	957.15	81.66	4.28
4.24	log-slicing_bvsrem_15	20	791.10	801.92	oot	-998.08	0.45	0.1	7	21	16	16	1.02	0	0	1046.61	4740.17	4738.88	0.22	446.07	75.24	4.29
4.25	float_newton.6.3.i	10	564.04	787.53	oot	-1012.47	0.44	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.30
4.26	...._test_v7_r7_vr1_c1_s4574	10	609.33	754.52	oot	-1045.48	0.42	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.31
4.27	...est_v5_r10_vr10_c1_s7608	20	1199.14	715.00	oot	-1085.00	0.40	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.32
4.28	...est_v5_r10_vr5_c1_s13679	10	899.64	696.33	oot	-1103.67	0.39	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.33
4.29	log-slicing_bvslt_11687	20	665.86	672.39	oot	-1127.61	0.37	2.0	4	12	12	1	266.10	1	0	275.99	2131.75	2101.20	27.78	1042.55	88.82	4.34
4.30	...st_v5_r15_vr10_c1_s11127	10	346.03	669.69	oot	-1130.31	0.37	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.35
4.31	...tcomp09_stall_bv_formula	20	609.53	615.59	oot	-1184.41	0.34	1.2	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.36
4.32	...test_v7_r7_vr1_c1_s22845	10	316.63	605.49	oot	-1194.51	0.34	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.37
4.33	log-slicing_bvult_9988	20	555.03	572.93	oot	-1227.07	0.32	2.0	4	12	12	2	263.02	0	0	194.34	3041.76	3006.71	27.81	1132.43	121.67	4.38
4.34	log-slicing_bvslt_10201	20	552.92	571.43	oot	-1228.57	0.32	2.0	4	12	12	2	265.88	1	0	228.88	2771.59	2739.70	29.06	1039.57	110.86	4.39
4.35	log-slicing_bvult_10985	20	556.62	561.14	oot	-1238.86	0.31	2.0	4	12	12	0	265.75	0	0	288.68	1015.58	981.70	29.31	73.91	44.16	4.40
4.36	log-slicing_bvult_8991	20	542.61	554.93	oot	-1245.07	0.31	1.9	4	12	12	0	263.35	0	0	246.31	951.20	918.34	30.28	61.51	41.36	4.41
4.37	log-slicing_bvslt_9458	20	539.16	553.43	oot	-1246.57	0.31	2.0	4	12	12	3	266.20	1	0	234.67	3238.54	3205.42	28.65	893.53	124.56	4.42
4.38	log-slicing_bvult_11982	20	518.16	528.44	oot	-1271.56	0.29	2.0	4	12	12	0	262.97	0	0	250.98	962.36	928.87	29.67	82.09	41.84	4.43
4.39	float_mul_03_30_4	20	514.80	520.18	oot	-1279.82	0.29	0.2	110	856	192	0	139.92	20	0	1634.11	6515.38	6307.88	0.01	18.87	2.80	4.44
4.40	log-slicing_bvslt_10944	20	505.68	510.44	oot	-1289.56	0.28	2.0	4	12	12	3	269.85	1	0	292.18	3356.42	3325.58	29.22	990.27	129.09	4.45
4.41	...test_v5_r10_vr5_c1_s8690	20	353.93	459.79	oot	-1340.21	0.26	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.46
4.42	float_newton.3.3.i	20	355.49	450.85	oot	-1349.15	0.25	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.47
4.43	float_newton.1.3.i	20	385.34	445.62	oot	-1354.38	0.25	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.48
4.44	float_newton.4.3.i	20	334.84	418.44	oot	-1381.56	0.23	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.49
4.45	...est_v5_r15_vr1_c1_s26845	20	619.09	415.20	oot	-1384.80	0.23	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.50
4.46	float_newton.5.3.i	20	415.28	404.47	oot	-1395.53	0.22	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.51
4.47	lfsr_lfsr_008_015_112	20	388.44	399.49	oot	-1400.51	0.22	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.52
4.48	...est_v7_r7_vr10_c1_s10625	10	852.76	391.84	oot	-1408.16	0.22	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.53
4.49	float_newton.2.3.i	20	289.38	385.61	oot	-1414.39	0.21	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.54
4.50	...._test_v7_r7_vr5_c1_s3582	10	326.78	384.03	oot	-1415.97	0.21	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.55

Continued on next page

Table C.4 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental-FLR)																		cmp
				par-ninc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-ninc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
4.51	...test_v7_r12_vr5_c1_s8938	10	615.16	367.14	oot	-1432.86	0.20	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.57
4.52	lfsr_lfsr_008_159_128	20	285.74	363.40	oot	-1436.60	0.20	6.5	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.14
4.53	log-slicing_bvslt_8715	20	342.56	350.68	oot	-1449.32	0.19	2.0	5	18	14	4	266.29	1	0	1279.09	4341.47	4296.70	27.48	998.90	111.32	4.58
4.54	lfsr_lfsr_008_159_112	20	275.94	338.98	oot	-1461.02	0.19	4.5	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.59
4.55	lfsr_lfsr_008_143_128	20	261.79	336.96	oot	-1463.04	0.19	5.8	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.60
4.56	...test_v7_r7_vr1_c1_s24449	20	679.70	330.95	oot	-1469.05	0.18	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.61
4.57	...servers_slapd_a_vc149923	10	325.64	330.06	oot	-1469.94	0.18	1.4	40	304	304	0	0.58	1	0	1579.13	6997.19	6722.93	5.28	13.53	6.32	4.62
4.58	...servers_slapd_a_vc149922	10	325.68	329.37	oot	-1470.63	0.18	1.3	40	304	304	0	0.71	1	0	1587.61	6989.49	6717.64	5.29	13.37	6.33	4.63
4.59	lfsr_lfsr_008_159_096	20	275.86	320.47	oot	-1479.53	0.18	3.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.64
4.60	tacas07_Y86_btnft	20	135.77	146.59	oom	-1483.13	0.09	6.8	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.103
4.61	...est_v5_r10_vr5_c1_s13195	10	337.55	314.00	oot	-1486.00	0.17	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.65
4.62	log-slicing_bvslt_7972	20	305.25	307.98	oot	-1492.02	0.17	0.7	6	18	14	6	66.91	1	0	1429.29	4676.06	4658.81	27.76	688.76	114.05	4.66
4.63	log-slicing_bvslt_5000	20	296.40	297.74	oot	-1502.26	0.17	0.7	6	18	14	10	67.06	1	0	1311.62	4137.70	4120.80	22.99	451.75	91.95	4.67
4.64	brummayerbiere3_minand256	20	289.66	295.21	oot	-1504.79	0.16	0.3	17	18	3	15	92.68	0	0	1672.82	2484.56	2467.79	0.12	275.82	50.71	4.68
4.65	lfsr_lfsr_008_015_128	20	277.36	286.74	oot	-1513.26	0.16	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.69
4.66	...est_v5_r10_vr1_c1_s32538	20	208.16	285.14	oot	-1514.86	0.16	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.70
4.67	lfsr_lfsr_008_127_128	20	212.80	282.86	oot	-1517.14	0.16	6.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.71
4.68	lfsr_lfsr_008_015_096	20	272.75	280.34	oot	-1519.66	0.16	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.72
4.69	VS3_VS3-benchmark-A9	10	268.45	271.77	oot	-1528.23	0.15	0.5	70	543	369	0	1.27	1	0	1730.29	6992.87	6778.58	0.28	6.25	3.07	4.73
4.70	lfsr_lfsr_008_143_112	20	205.39	260.60	oot	-1539.40	0.14	4.3	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.74
4.71	...est_v5_r10_vr1_c1_s13516	10	231.36	254.96	oot	-1545.04	0.14	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.75
4.72	...est_v5_r15_vr1_c1_s32559	20	163.75	250.60	oot	-1549.40	0.14	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.76
4.73	...test_v5_r5_vr10_c1_s7194	10	141.84	249.36	oot	-1550.64	0.14	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.77
4.74	lfsr_lfsr_008_127_112	20	176.00	226.47	oot	-1573.53	0.13	4.2	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.78
4.75	lfsr_lfsr_008_159_080	20	194.33	224.52	oot	-1575.48	0.12	1.9	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.79
4.76	..._test_v5_r5_vr5_c1_s2800	10	228.14	223.67	oot	-1576.33	0.12	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.80
4.77	log-slicing_bvslt_7229	20	217.01	221.19	oot	-1578.81	0.12	0.7	6	18	14	6	67.27	1	0	1444.82	3738.99	3722.19	27.67	711.09	91.19	4.81
4.78	lfsr_lfsr_008_143_096	20	173.94	217.04	oot	-1582.96	0.12	2.9	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.82
4.79	...test_v7_r7_vr5_c1_s14675	10	364.55	212.19	oot	-1587.81	0.12	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.83
4.80	lfsr_lfsr_008_111_128	20	160.68	210.28	oot	-1589.72	0.12	3.8									0.00	0.00	0.00	0.00	0.00	4.84
4.81	...yerbiere3_minandmaxor064	20	207.79	206.66	oot	-1593.34	0.11	0.1	137	641	23	0	25.44	0	0	1764.62	3426.04	3358.98	0.01	21.39	2.63	4.85
4.82	...est_v7_r7_vr10_c1_s24535	10	237.88	205.90	oot	-1594.10	0.11	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.86
4.83	..._smtcomp09_sw_bv_formula	20	202.42	204.90	oot	-1595.10	0.11	1.5	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.11
4.84	lfsr_lfsr_008_095_128	20	148.89	203.35	oot	-1596.65	0.11	2.7	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.87
4.85	..._smtcomp09_lf_bv_formula	20	180.24	202.33	oot	-1597.67	0.11	0.9	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.3
4.86	lfsr_lfsr_004_111_112	20	178.05	198.93	oot	-1601.07	0.11	1.7	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.45
4.87	...mayerbiere2_smulov2bw512	20	198.38	198.78	oot	-1601.22	0.11	2.5	6	8	4	0	132.48	0	0	1497.44	2047.55	1927.34	12.18	403.84	186.14	4.88
4.88	lfsr_lfsr_008_111_112	20	148.60	194.64	oot	-1605.36	0.11	3.4	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.89
4.89	log-slicing_bvslt_6486	20	191.65	194.40	oot	-1605.60	0.11	0.7	7	18	14	14	67.12	1	0	1597.23	5703.31	5682.70	27.27	445.21	107.61	4.90
4.90	lfsr_lfsr_008_095_112	20	129.08	179.91	oot	-1620.09	0.10	2.3	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.91
4.91	float_newton.5.2.i	20	249.53	175.92	oot	-1624.08	0.10	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.92
4.92	lfsr_lfsr_008_047_096	20	160.40	173.71	oot	-1626.29	0.10	0.6	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.93
4.93	lfsr_lfsr_008_111_096	20	133.32	172.22	oot	-1627.78	0.10	2.9	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.94
4.94	float_newton.4.2.i	20	143.25	170.46	oot	-1629.54	0.09	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.95
4.95	lfsr_lfsr_008_159_064	20	149.52	166.93	oot	-1633.07	0.09	1.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.63
4.96	...mtcomp09_full_bv_formula	20	172.46	159.04	oot	-1640.96	0.09	3.4	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.5
4.97	...test_v5_r5_vr1_c1_s14623	10	143.20	158.20	oot	-1641.80	0.09	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.96
4.98	lfsr_lfsr_004_143_128	20	116.21	156.03	oot	-1643.97	0.09	2.9	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.97
4.99	lfsr_lfsr_004_159_128	20	112.10	152.78	oot	-1647.22	0.08	3.3	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.32
4.100	...mayerbiere2_smulov2bw448	20	150.63	150.34	oot	-1649.66	0.08	2.2	7	10	4	0	102.91	0	0	1591.58	3123.76	3001.99	8.12	458.12	208.25	4.98
4.101	lfsr_lfsr_008_127_096	20	113.04	149.61	oot	-1650.39	0.08	2.8	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.99

Continued on next page

Table C.4 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental-FLR)																	cmp
				par-ninc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-ninc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
4.102	lfsr_lfsr_008_079_128	20	107.76	149.60	oot	-1650.40	0.08	1.8	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.100
4.103	brummayerbiere3_maxxor032	20	143.95	148.04	oot	-1651.96	0.08	0.1	160	1254	284	0	0.79	0	0	1764.47	6747.90	6694.29	0.06	12.55	1.63	4.101
4.104	lfsr_lfsr_008_095_096	20	105.52	147.68	oot	-1652.32	0.08	2.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.102
4.105	...mayerbiere2_umulov1bw256	20	143.94	145.22	oot	-1654.78	0.08	1.2						0	0		0.00	0.00	0.00	0.00	0.00	4.104
4.106	...est_v5_r10_vr1_c1_s19145	10	316.49	144.16	oot	-1655.84	0.08	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.105
4.107	float_newton.3.2.i	20	186.56	140.68	oot	-1659.32	0.08	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.106
4.108	lfsr_lfsr_008_143_080	20	112.13	139.64	oot	-1660.36	0.08	1.8	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.107
4.109	..._smtcomp09_nt_bv_formula	20	156.72	137.27	oot	-1662.73	0.08	0.7	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.56
4.110	float_newton.1.2.i	20	139.98	134.22	oot	-1665.78	0.07	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.108
4.111	...test_v5_r5_vr5_c1_s24018	10	182.81	131.06	oot	-1668.94	0.07	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.109
4.112	brummayerbiere_countbits064	20	129.34	130.20	oot	-1669.80	0.07	0.2						2	0		0.00	0.00	0.00	0.00	0.00	4.110
4.113	tacas07_Y86_std	20	129.65	129.50	oot	-1670.50	0.07	0.8	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.111
4.114	...smtcomp09_std_bv_formula	20	108.86	128.59	oot	-1671.41	0.07	0.7	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.4
4.115	...mayerbiere2_smulov2bw384	20	122.06	122.27	oot	-1677.73	0.07	1.9	10	16	4	0	79.37	0	0	1670.57	4180.30	4023.91	5.22	299.07	154.83	4.112
4.116	...test_v5_r15_vr1_c1_s8236	20	927.66	120.26	oot	-1679.74	0.07	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.113
4.117	lfsr_lfsr_008_143_064	20	103.49	119.65	oot	-1680.35	0.07	1.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.114
4.118	brummayerbiere3_minor256	20	116.13	118.55	oot	-1681.45	0.07	0.7	29	216	216	0	67.81	0	0	1725.63	6015.83	5785.38	2.59	55.87	10.18	4.115
4.119	float_newton.2.2.i	20	143.18	116.29	oot	-1683.71	0.06	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.116
4.120	brummayerbiere3_minxor064	20	111.84	115.98	oot	-1684.02	0.06	0.2	108	847	364	0	4.39	0	0	1790.39	6738.62	6550.49	0.02	21.44	1.78	4.117
4.121	VS3_VS3-benchmark-A6	10	111.93	112.68	oot	-1687.32	0.06	0.3	60	453	237	0	1.03	1	0	1711.08	6919.62	6791.22	0.28	16.32	4.32	4.118
4.122	lfsr_lfsr_008_159_048	20	100.56	110.87	oot	-1689.13	0.06	0.5	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	3.62
4.123	core_ext_con_060_256_1024	20	101.73	110.06	oot	-1689.94	0.06	0.3	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.119
4.124	...mayerbiere2_umulov1bw224	20	108.42	109.18	oot	-1690.82	0.06	1.0	26	162	22	0	23.40	0	0	1743.38	6085.57	5726.96	1.05	87.49	18.27	4.120
4.125	float_mult2.c.50	10	109.39	108.05	oot	-1691.95	0.06	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	4.121
4.126	brummayerbiere3_maxand256	20	102.05	103.83	oot	-1696.17	0.06	0.7	34	256	256	0	65.12	0	0	1705.07	6131.79	5815.19	0.96	39.28	6.87	4.122
5.1	...mayerbiere_countbits1024	0	oot	oot	oom	723.66	1.67	6.8	2	4	4	0	376.83	2	0	684.58	908.71	876.10	227.22	436.33	302.90	5.1
5.2	...erbiere4_unconstrained10	0	oom	oom	oom	1.33	1.01	7.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.2
5.3	...erbiere4_unconstrained03	0	oom	oom	oom	1.10	1.02	7.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.3
5.4	...erbiere4_unconstrained06	0	oom	oom	oom	0.81	1.02	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.5
5.5	...erbiere4_unconstrained09	0	oom	oom	oom	0.67	1.01	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.150
5.6	...erbiere4_unconstrained01	0	oom	oom	oom	0.38	1.01	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.8
5.7	...erbiere4_unconstrained07	0	oom	oom	oom	0.20	1.00	7.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.4
5.8	fft_Sz1024_34824	0	oot	oot	oot	0.00	1.00	1.0	80	624	624	0	122.64	0	0	1422.15	6586.48	6166.23	1.55	4.95	1.71	5.9
5.9	fft_Sz512_15128_0	0	oot	oot	oot	0.00	1.00	0.7	150	1168	1080	0	23.48	0	0	1655.52	7004.72	6579.48	0.05	2.70	0.67	5.10
5.10	fft_Sz512_15128_1	0	oot	oot	oot	0.00	1.00	0.5						0	0		0.00	0.00	0.00	0.00	0.00	5.11
5.11	fft_Sz512_15128_2	0	oot	oot	oot	0.00	1.00	0.7	170	1336	1256	0	21.08	1	0	1591.63	7014.34	6531.02	0.07	2.31	0.52	5.12
5.12	fft_Sz512_15128_4	0	oot	oot	oot	0.00	1.00	0.8						0	0		0.00	0.00	0.00	0.00	0.00	5.13
5.13	fft_Sz512_15128_5	0	oot	oot	oot	0.00	1.00	0.7						0	0		0.00	0.00	0.00	0.00	0.00	5.14
5.14	fft_Sz512_15128_6	0	oot	oot	oot	0.00	1.00	0.6						1	0		0.00	0.00	0.00	0.00	0.00	5.15
5.15	...st_v5_r10_vr10_c1_s15708	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.16
5.16	...st_v7_r12_vr10_c1_s15994	10	1367.71	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.17
5.17	...est_v7_r12_vr1_c1_s10576	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.18
5.18	..._test_v7_r12_vr1_c1_s703	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.19
5.19	...est_v7_r12_vr5_c1_s29826	10	896.50	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.20
5.20	...est_v7_r17_vr1_c1_s23882	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.21
5.21	...est_v7_r17_vr1_c1_s30331	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.22
5.22	...fyInterleavedModMult-128	0	oot	oot	oot	0.00	1.00	2.7	10	61	61	0	39.25	1	0	1251.45	6486.84	5977.97	34.35	50.56	40.80	5.23
5.23	...ffyInterleavedModMult-32	0	oot	oot	oot	0.00	1.00	1.2	60	446	446	0	1.52	1	0	1563.69	6997.75	6712.33	2.57	12.98	3.14	5.24
5.24	...iffyInterleavedModMult-8	0	oot	oot	oot	0.00	1.00	0.1	150	1141	205	0	0.11	1	0	1746.75	6813.41	6791.27	0.00	8.97	1.83	5.25
5.25	lfsr_lfsr_008_031_064	0	oot	oot	oot	0.00	1.00	0.2	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.26

Continued on next page

Table C.4 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental-FLR)																	cmp
				par-ninc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-ninc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.26	lfsr_lfsr_008_063_096	0	oot	oot	oot	0.00	1.00	1.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.27
5.27	lfsr_lfsr_008_063_128	0	oot	oot	oot	0.00	1.00	1.4	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.28
5.28	log-slicing_bvmul_15	0	oot	oot	oot	0.00	1.00	0.1	2	2	2	0	0.00	0	0	0.87	1.16	1.09	0.30	0.54	0.39	5.29
5.29	log-slicing_bvmul_16	0	oot	oot	oot	0.00	1.00	0.2	4	8	8	0	0.00	0	0	1.65	4.31	4.08	0.21	0.47	0.29	5.30
5.30	log-slicing_bvmul_17	0	oot	oot	oot	0.00	1.00	0.1	2	2	2	0	0.00	0	0	1.21	1.64	1.32	0.46	0.68	0.55	5.31
5.31	log-slicing_bvmul_18	0	oot	oot	oot	0.00	1.00	0.2	2	2	2	0	0.00	0	0	1.30	1.75	1.43	0.49	0.76	0.58	5.32
5.32	log-slicing_bvsdiv_19	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	5.81	0	0	6.50	19.62	17.51	0.44	2.22	0.73	5.33
5.33	log-slicing_bvsdiv_20	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	5.82	0	0	7.07	21.60	19.43	0.47	2.16	0.80	5.34
5.34	log-slicing_bvsdiv_21	0	oot	oot	oot	0.00	1.00	0.3	4	14	14	0	5.82	0	0	6.62	22.27	20.45	0.42	2.43	0.82	5.35
5.35	log-slicing_bvsdiv_22	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	5.79	0	0	7.39	24.06	21.82	0.54	2.69	0.89	5.36
5.36	log-slicing_bvsdiv_23	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	5.81	0	0	9.38	27.20	24.92	0.58	3.38	1.01	5.37
5.37	log-slicing_bvsdiv_24	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	5.78	0	0	9.63	29.39	27.16	0.61	3.42	1.09	5.38
5.38	log-slicing_bvsdiv_25	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	5.84	0	0	10.35	31.24	29.17	0.58	3.59	1.16	5.39
5.39	log-slicing_bvslt_12430	0	oot	oot	oot	0.00	1.00	2.0	4	12	12	0	264.40	1	0	260.63	1040.10	1007.08	30.15	74.26	45.22	5.40
5.40	log-slicing_bvslt_18374	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.41
5.41	log-slicing_bvslt_19117	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.42
5.42	log-slicing_bvslt_19860	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.43
5.43	log-slicing_bvsmod_15	0	oot	oot	oot	0.00	1.00	0.2	11	42	25	21	1.25	0	0	1784.26	4839.79	4837.63	0.23	613.02	39.35	5.44
5.44	log-slicing_bvsmod_16	0	oot	oot	oot	0.00	1.00	0.1	9	44	30	2	1.23	0	0	1757.79	2264.25	2262.32	0.02	1748.07	23.10	5.45
5.45	log-slicing_bvsmod_17	0	oot	oot	oot	0.00	1.00	0.3	9	45	31	7	6.66	0	0	101.95	3935.18	3927.21	0.04	1122.22	36.44	5.46
5.46	log-slicing_bvsmod_18	0	oot	oot	oot	0.00	1.00	0.1	4	15	15	0	6.57	0	0	5.93	19.63	17.57	0.37	1.94	0.68	5.47
5.47	log-slicing_bvsmod_19	0	oot	oot	oot	0.00	1.00	0.1	4	15	15	0	6.62	0	0	6.73	22.55	20.23	0.45	2.43	0.78	5.48
5.48	log-slicing_bvsmod_20	0	oot	oot	oot	0.00	1.00	0.2	4	15	15	0	6.57	0	0	8.06	26.57	24.30	0.48	2.73	0.92	5.49
5.49	log-slicing_bvsrem_16	0	oot	oot	oot	0.00	1.00	0.2	7	28	18	5	1.02	0	0	582.86	3620.11	3618.78	0.02	1149.90	56.56	5.50
5.50	log-slicing_bvsrem_17	0	oot	oot	oot	0.00	1.00	0.3	6	27	21	4	5.63	0	0	8.88	3859.03	3854.73	0.04	968.52	62.24	5.51
5.51	log-slicing_bvsrem_18	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	5.62	0	0	6.53	21.59	19.72	0.20	1.87	0.80	5.52
5.52	log-slicing_bvsrem_19	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	5.68	0	0	7.41	23.20	20.95	0.45	2.12	0.86	5.53
5.53	log-slicing_bvsrem_20	0	oot	oot	oot	0.00	1.00	0.2	4	14	14	0	5.61	0	0	8.28	25.28	23.06	0.46	2.54	0.94	5.54
5.54	log-slicing_bvsub_12973	0	oot	oot	oot	0.00	1.00	0.5	1	1	1	0	0.00	0	0	321.69	319.04	310.47	319.04	319.04	319.04	5.55
5.55	log-slicing_bvsub_13970	0	oot	oot	oot	0.00	1.00	0.2	1	1	1	0	0.00	0	0	394.91	392.10	382.21	392.10	392.10	392.10	5.56
5.56	log-slicing_bvsub_14967	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0	0	314.11	311.12	299.95	311.12	311.12	311.12	5.57
5.57	log-slicing_bvsub_15964	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	373.42	366.89	353.92	366.89	366.89	366.89	5.58
5.58	log-slicing_bvsub_16961	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	573.33	566.56	551.85	566.56	566.56	566.56	5.59
5.59	log-slicing_bvsub_17958	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	392.43	385.39	369.22	385.39	385.39	385.39	5.60
5.60	log-slicing_bvsub_18955	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	498.71	490.63	472.02	490.63	490.63	490.63	5.61
5.61	log-slicing_bvsub_19952	0	oot	oot	oot	0.00	1.00	0.3	1	1	1	0	0.00	0	0	531.59	522.30	501.97	522.30	522.30	522.30	5.62
5.62	log-slicing_bvsub_20949	0	oot	oot	oot	0.00	1.00	0.5	1	1	1	0	0.00	0	0	675.54	665.29	643.19	665.29	665.29	665.29	5.63
5.63	log-slicing_bvsub_21946	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0	0	896.85	885.31	859.77	885.31	885.31	885.31	5.64
5.64	log-slicing_bvsub_22943	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0	0	843.84	831.28	803.10	831.28	831.28	831.28	5.65
5.65	log-slicing_bvsub_23940	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0	0	718.51	704.94	675.59	704.94	704.94	704.94	5.66
5.66	log-slicing_bvudiv_18	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.07	0	0	2.89	5.78	5.46	0.93	1.92	1.16	5.67
5.67	log-slicing_bvudiv_19	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.07	0	0	3.63	7.21	6.89	1.20	2.27	1.44	5.68
5.68	log-slicing_bvudiv_20	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.07	0	0	4.18	8.43	8.10	1.42	2.34	1.69	5.69
5.69	log-slicing_bvudiv_21	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.09	0	0	4.28	8.79	8.46	1.47	2.65	1.76	5.70
5.70	log-slicing_bvudiv_22	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.08	0	0	4.55	8.65	8.31	1.36	2.66	1.73	5.71
5.71	log-slicing_bvudiv_23	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.09	0	0	4.85	9.77	9.42	1.57	3.14	1.95	5.72
5.72	log-slicing_bvudiv_24	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.09	0	0	5.20	10.49	10.15	1.75	3.34	2.10	5.73
5.73	log-slicing_bvudiv_25	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.08	0	0	5.60	11.56	11.20	1.95	3.42	2.31	5.74
5.74	log-slicing_bvult_16967	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.75
5.75	log-slicing_bvult_17964	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.76
5.76	log-slicing_bvult_18961	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.77

Continued on next page

Table C.4 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental-FLR)																	cmp
				par-ninc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-ninc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.77	log-slicing_bvult_19958	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.78
5.78	log-slicing_bvult_20955	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.79
5.79	log-slicing_bvult_21952	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.80
5.80	log-slicing_bvult_22949	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.81
5.81	log-slicing_bvult_23946	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.82
5.82	log-slicing_bvult_24943	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.83
5.83	log-slicing_bvult_25940	0	oot	oot	oot	0.00	1.00	0.1	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.84
5.84	log-slicing_bvurem_15	0	oot	oot	oot	0.00	1.00	0.1	3	3	3	2	0.68	0	0	584.12	1767.54	1767.36	0.87	1180.72	252.51	5.85
5.85	log-slicing_bvurem_16	0	oot	oot	oot	0.00	1.00	0.2	7	25	16	0	0.69	0	0	14.58	43.24	41.92	0.11	3.03	0.88	5.86
5.86	log-slicing_bvurem_17	0	oot	oot	oot	0.00	1.00	0.2	6	24	14	0	4.08	0	0	11.21	34.16	31.37	0.32	2.89	0.73	5.87
5.87	log-slicing_bvurem_18	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.06	0	0	4.76	9.59	9.24	1.59	3.00	1.92	5.88
5.88	log-slicing_bvurem_19	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.06	0	0	5.42	10.98	10.61	1.85	3.41	2.20	5.89
5.89	log-slicing_bvurem_20	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.08	0	0	5.88	11.90	11.53	1.97	3.72	2.38	5.90
5.90	log-slicing_bvurem_21	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.08	0	0	6.94	13.82	13.43	2.27	4.49	2.76	5.91
5.91	log-slicing_bvurem_22	0	oot	oot	oot	0.00	1.00	0.1	2	3	3	0	4.07	0	0	7.21	14.42	14.04	2.33	4.58	2.88	5.92
5.92	pspace_power2sum.8087	0	oot	oot	oot	0.00	1.00	0.8	1	1	1	0	0.00	0	0	707.99	707.62	706.99	707.62	707.62	707.62	5.93
5.93	pspace_power2sum.9679	0	oot	oot	oot	0.00	1.00	0.4	1	1	1	0	0.00	0	0	187.64	187.57	186.71	187.57	187.57	187.57	5.94
5.94	RWS_Example_15.txt	0	oot	oot	oot	0.00	1.00	0.4	0	1	1	0	66.40	1	0	0.00	0.00	0.00	0.00	0.00	0.00	5.95
5.95	RWS_Example_19.txt	0	oot	oot	oot	0.00	1.00	0.5	0	1	1	0	475.57	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.96
5.96	RWS_Example_20.txt	0	oot	oot	oot	0.00	1.00	0.7	0	1	1	0	1029.92	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.97
5.97	RWS_Example_7.txt	0	oot	oot	oot	0.00	1.00	0.1	0	1	1	0	15.43	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.98
5.98	uum_uum12	0	oot	oot	oot	0.00	1.00	0.4	177	1367	213	0	208.01	2	0	1574.08	6033.85	5786.01	0.00	7.78	1.35	5.99
5.99	uum_uum16	0	oot	oot	oot	0.00	1.00	1.3	93	720	469	0	496.48	2	0	1292.37	5047.87	4700.70	0.06	10.60	1.30	5.100
5.100	uum_uum20	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.101
5.101	uum_uum24	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.102
5.102	uum_uum28	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.103
5.103	uum_uum32	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.104
5.104	VS3_VS3-benchmark-A10	0	oot	oot	oot	0.00	1.00	0.6	70	544	499	0	1.36	1	0	1756.61	7066.60	6827.51	0.47	7.22	2.84	5.105
5.105	VS3_VS3-benchmark-A11	0	oot	oot	oot	0.00	1.00	0.5	60	463	391	0	1.27	1	0	1632.76	7006.64	6804.97	0.36	9.25	3.26	5.106
5.106	VS3_VS3-benchmark-A12	0	oot	oot	oot	0.00	1.00	0.5	70	536	397	0	1.39	1	0	1686.61	6956.46	6727.71	0.45	20.71	2.90	5.107
5.107	VS3_VS3-benchmark-A13	0	oot	oot	oot	0.00	1.00	0.6	53	408	384	0	1.80	1	0	1783.80	7012.11	6783.43	2.52	9.41	3.80	5.108
5.108	VS3_VS3-benchmark-A8	0	oot	oot	oot	0.00	1.00	0.5	60	464	410	0	1.53	1	0	1648.18	7046.09	6820.55	2.63	7.74	3.45	5.109
5.109	VS3_VS3-benchmark-S2	0	oot	oot	oot	0.00	1.00	0.6	20	136	136	0	1.57	1	0	1567.26	6734.20	6561.58	15.87	37.54	18.15	5.110
5.110	...1_alt_true.BV.c.cil.c.17	0	oot	oot	oot	0.00	1.00	2.2	20	137	124	0	1178.11	24	0	473.38	2226.01	2056.30	0.04	23.58	5.68	5.111
5.111	...1_alt_true.BV.c.cil.c.21	0	oot	oot	oot	0.00	1.00	0.0	0	1	0	0	0.00			0.00	0.00	0.00	0.00	0.00	0.00	5.112
5.112	brummayerbiere2_smulov1bw48	0	oot	oot	oot	0.00	1.00	0.1						0	0		0.00	0.00	0.00	0.00	0.00	5.113
5.113	...erbiere3_isqrtaddeqcheck	0	oot	oot	oot	0.00	1.00	0.1	50	376	240	0	0.37	0	0	1266.05	6852.85	6810.92	0.03	28.06	4.83	5.114
5.114	...mayerbiere3_isqrtaddnoif	0	oot	oot	oot	0.00	1.00	0.1	51	378	159	0	0.05	0	0	1780.52	6778.89	6762.69	0.00	27.72	6.07	5.115
5.115	...mayerbiere3_isqrteqcheck	0	oot	oot	oot	0.00	1.00	0.1	48	368	198	0	0.37	0	0	1781.41	6865.88	6828.89	0.00	32.02	5.70	5.116
5.116	brummayerbiere3_isqrtnoif	0	oot	oot	oot	0.00	1.00	0.0	145	1100	131	0	0.05	0	0	1777.63	6846.14	6822.82	0.01	16.28	2.26	5.117
5.117	brummayerbiere3_maxor128	0	oot	oot	oot	0.00	1.00	0.2	20	21	3	18	24.06	0	0	1741.68	1963.39	1950.24	2.87	130.36	33.28	5.118
5.118	brummayerbiere3_maxor256	0	oot	oot	oot	0.00	1.00	0.6	18	19	3	16	93.62	0	0	1690.16	2195.94	2164.12	3.12	140.68	42.23	5.119
5.119	brummayerbiere3_maxxor064	0	oot	oot	oot	0.00	1.00	0.3	93	728	588	0	3.10	0	0	1743.55	6985.81	6812.46	0.17	29.61	1.54	5.120
5.120	brummayerbiere3_maxxor128	0	oot	oot	oot	0.00	1.00	0.5	50	384	384	0	12.30	0	0	1746.01	6949.90	6628.96	3.09	8.36	4.80	5.121
5.121	brummayerbiere3_maxxor256	0	oot	oot	oot	0.00	1.00	1.3	17	120	120	0	50.36	0	0	1742.48	6574.18	6251.22	18.88	32.77	24.26	5.122
5.122	...biere3_maxxormaxorand032	0	oot	oot	oot	0.00	1.00	0.2						0	0		0.00	0.00	0.00	0.00	0.00	5.123
5.123	...biere3_maxxormaxorand064	0	oot	oot	oot	0.00	1.00	0.7	90	704	704	0	41.54	0	0	1676.54	6695.15	6177.38	1.03	16.14	1.59	5.124
5.124	...biere3_maxxormaxorand128	0	oot	oot	oot	0.00	1.00	0.8	30	224	224	0	155.48	0	0	1297.55	6043.09	5636.13	4.87	35.58	7.72	5.125
5.125	...biere3_maxxormaxorand256	0	oot	oot	oot	0.00	1.00	2.0	10	64	64	0	639.73	0	0	1122.88	4075.31	3807.46	23.57	74.37	36.06	5.126
5.126	...yerbiere3_minandmaxor128	0	oot	oot	oot	0.00	1.00	0.4	61	461	26	0	98.02	0	0	1668.82	5766.35	5585.68	0.06	29.44	6.08	5.127
5.127	...yerbiere3_minandmaxor256	0	oot	oot	oot	0.00	1.00	1.7	14	85	19	0	375.09	0	0	1377.80	4277.99	4037.65	1.32	82.44	27.08	5.128

Continued on next page

Table C.4 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Non-Incremental-FLR)																		cmp
				par-ninc	tot(wc)	diff	su	mem	rnds	subproblems			flr			search						par-ninc
										tot	max	fin	tot(wc)	fl	res	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[]
5.128	brummayerbiere3_minxor128	0	oot	oot	oot	0.00	1.00	0.6	90	704	704	0	16.64	0	0	1612.92	6767.00	6257.13	1.05	18.58	1.52	5.129
5.129	brummayerbiere3_minxor256	0	oot	oot	oot	0.00	1.00	0.7	28	208	208	0	67.40	0	0	1730.13	6222.73	5983.18	7.19	42.65	10.82	5.130
5.130	...erbiere3_minxorminand064	0	oot	oot	oot	0.00	1.00	0.5	80	624	434	0	49.41	0	0	1572.53	6530.75	6148.68	0.00	18.18	2.08	5.131
5.131	...erbiere3_minxorminand128	0	oot	oot	oot	0.00	1.00	0.8	30	224	224	0	183.51	0	0	1371.64	6194.58	5882.87	7.49	29.31	8.58	5.132
5.132	...erbiere3_minxorminand256	0	oot	oot	oot	0.00	1.00	1.6	7	40	40	0	721.94	0	0	893.93	3701.36	3562.47	43.69	74.86	49.35	5.133
5.133	brummayerbiere3_mulhs16	0	oot	oot	oot	0.00	1.00	0.1	19	129	89	0	0.00	0	0	21.49	75.57	73.69	0.01	0.92	0.25	5.134
5.134	brummayerbiere3_mulhs32	0	oot	oot	oot	0.00	1.00	0.2	18	123	87	0	0.00	0	0	65.55	229.48	222.00	0.02	2.38	0.81	5.135
5.135	brummayerbiere3_mulhs64	0	oot	oot	oot	0.00	1.00	0.2	18	125	101	0	0.00	0	0	293.69	1034.35	1000.83	0.10	8.84	3.76	5.136
5.136	...erbiere4_unconstrained02	0	oot	oot	oot	0.00	1.00	5.2	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.137
5.137	brummayerbiere_countbits128	0	oot	oot	oot	0.00	1.00	0.3						2	0		0.00	0.00	0.00	0.00	0.00	5.138
5.138	brummayerbiere_countbits256	0	oot	oot	oot	0.00	1.00	0.8						2	0		0.00	0.00	0.00	0.00	0.00	5.139
5.139	brummayerbiere_countbits512	0	oot	oot	oot	0.00	1.00	2.3						2	0		0.00	0.00	0.00	0.00	0.00	5.140
5.140	...biere_countbitsrotate032	0	oot	oot	oot	0.00	1.00	0.1	51	391	285	0	0.32	0	0	1760.99	6936.92	6919.10	0.00	29.72	4.29	5.141
5.141	...biere_countbitsrotate064	0	oot	oot	oot	0.00	1.00	0.9	56	426	424	0	3.92	0	0	1785.28	6743.24	6639.33	0.02	15.95	3.65	5.142
5.142	...biere_countbitsrotate128	0	oot	oot	oot	0.00	1.00	1.3	18	128	128	0	52.01	0	0	1734.57	6236.87	6135.14	7.28	52.44	21.73	5.143
5.143	...biere_countbitsrotate256	0	oot	oot	oot	0.00	1.00	2.6						0	0		0.00	0.00	0.00	0.00	0.00	5.144
5.144	...yerbiere_countbitssrl064	0	oot	oot	oot	0.00	1.00	0.1	39	185	14	48	2.80	2	0	1496.02	2936.71	2920.42	0.00	394.32	7.04	5.145
5.145	...yerbiere_countbitssrl128	0	oot	oot	oot	0.00	1.00	0.5	57	405	46	27	7.07	2	0	1572.31	5508.79	5408.21	0.00	55.36	5.64	5.146
5.146	...yerbiere_countbitssrl256	0	oot	oot	oot	0.00	1.00	1.1	40	282	38	16	20.23	2	0	1631.05	5961.12	5781.34	0.00	97.45	9.30	5.147
5.147	challenge_integerOverflow	0	oot	oot	oot	0.00	1.00	0.2	5	12	8	2	0.01	0	0	17.92	2021.30	2020.49	0.03	1064.35	80.85	5.148
5.148	...erbiere4_unconstrained08	0	oom	oom	oom	-0.06	1.00	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.149
5.149	...erbiere4_unconstrained05	0	oom	oom	oom	-0.10	1.00	7.0	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.7
5.150	...erbiere4_unconstrained04	0	oom	oom	oom	-0.58	0.99	6.9	0	1	1	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00	0.00	5.6

Table C.5: Detailed results of PBoolector with look-ahead for satisfiable instances (Parallel-Incremental-Sat) compared to the reference times Base-Incremental. Last column par-inc contains cross reference to detailed results of Parallel-Incremental in Tab.C.1.

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental-Sat)																	cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search							par-inc
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
1.1	log-slicing_bvurem_16	20	oot	oot	1004.85	795.15	1.79	0.2	8	24	23	22	0.01	0.34	1003.87	2534.62	2534.43	0.00	340.98	30.54	1.2
1.2	brummayerbiere2_smulov1bw32	20	1056.25	oot	1019.39	780.61	1.77	0.9	107	599	83	106	0.08	5.66	1012.15	3066.87	3056.04	0.00	28.26	1.80	1.1
1.3	...iffyInterleavedModMult-8	20	oot	oot	1283.79	516.21	1.40	0.2	78	506	78	0	0.09	1.43	1281.02	4664.22	4662.67	0.00	21.96	3.80	1.3
1.4	core_ext_con_056_002_1024	20	279.98	oot	1461.86	338.14	1.23	0.9	1	1	1	1	0.00	0.00	1461.28	1461.28	1460.89	155.06	1306.23	730.64	1.4
1.5	core_ext_con_060_002_1024	20	220.71	oot	1465.95	334.05	1.23	0.6	1	1	1	1	0.00	0.00	1465.46	1465.46	1465.05	165.93	1299.53	732.73	1.5
1.6	core_ext_con_052_002_1024	20	194.60	oot	1496.51	303.49	1.20	0.7	1	1	1	1	0.00	0.00	1496.00	1496.00	1495.63	138.21	1357.79	748.00	1.6
1.7	log-slicing_bvudiv_18	20	oot	oot	1542.54	257.46	1.17	0.9	11	40	31	23	0.03	7.55	1530.56	4253.01	4251.80	0.00	530.99	34.58	1.7
1.8	core_ext_con_064_002_1024	20	344.29	oot	1769.00	31.00	1.02	0.9	1	1	1	1	0.00	0.00	1768.46	1768.46	1768.02	164.74	1603.72	884.23	1.8
2.1	log-slicing_bvudiv_17	20	1270.36	1373.34	621.99	751.35	2.21	0.7	10	33	23	21	0.02	5.97	612.46	1573.29	1572.17	0.00	186.02	15.73	2.1
2.2	log-slicing_bvurem_15	20	oot	1147.71	412.38	735.33	2.78	0.2	8	24	23	20	0.01	0.41	411.37	984.93	984.75	0.00	136.54	12.31	2.2
2.3	log-slicing_bvsmod_14	20	1414.54	1350.59	802.28	548.31	1.68	0.3	42	174	46	111	0.12	2.71	797.95	1926.19	1924.65	0.00	62.34	3.11	2.3
2.4	log-slicing_bvmul_12	20	129.19	962.31	421.28	541.03	2.28	0.1	2	2	2	2	0.00	0.02	420.55	814.16	814.14	0.28	419.75	162.83	2.4
2.5	core_ext_con_048_002_1024	20	487.72	1716.64	1295.00	421.64	1.33	0.7	1	1	1	1	0.00	0.00	1294.50	1294.50	1294.19	124.73	1169.77	647.25	2.5
2.6	core_ext_con_044_002_1024	20	195.82	1571.35	1267.20	304.15	1.24	0.7	1	1	1	1	0.00	0.00	1266.69	1266.69	1266.40	107.60	1159.09	633.35	2.6
2.7	log-slicing_bvudiv_16	20	390.99	521.12	250.44	270.68	2.08	0.2	8	24	22	20	0.01	0.51	249.22	739.08	738.85	0.00	86.27	9.24	2.8
2.8	core_ext_con_036_002_1024	20	269.99	1156.19	896.21	259.98	1.29	0.5	1	1	1	1	0.00	0.00	895.72	895.72	895.48	64.54	831.19	447.86	2.9
2.9	..._smtcomp09_lf_bv_formula	20	180.24	207.71	20.50	187.21	10.13	0.8	8	40	32	0	0.05	2.41	17.28	55.66	50.22	0.02	2.89	0.59	2.10
2.10	log-slicing_bvurem_14	20	757.16	270.88	128.60	142.28	2.11	0.1	8	24	22	19	0.01	0.30	127.65	443.73	443.57	0.00	60.16	5.69	2.11
2.11	float_newton.6.3.i	10	564.04	437.06	303.75	133.31	1.44	4.0	10	30	21	0	0.41	18.56	279.78	881.37	846.10	0.38	54.66	7.60	4.47
2.12	..._smtcomp09_nt_bv_formula	20	156.72	175.28	82.19	93.09	2.13	1.8	14	80	56	0	0.15	7.88	71.92	274.24	260.74	0.06	20.73	1.32	2.12
2.13	challenge_multiplyOverflow	20	1035.61	1002.11	923.88	78.23	1.08	0.1	2	2	2	2	0.00	0.02	923.33	1507.28	1507.25	0.56	921.55	301.46	2.14
2.14	log-slicing_bvudiv_15	20	162.70	192.95	117.35	75.60	1.64	0.2	8	23	21	18	0.01	0.50	116.21	339.61	339.37	0.00	40.85	4.53	2.15
2.15	brummayerbiere_nlzbe256	20	117.51	157.97	97.08	60.89	1.63	0.7	3	5	4	0	0.01	4.19	92.14	127.54	123.57	0.11	34.31	14.17	2.16
2.16	log-slicing_bvurem_13	20	147.97	116.67	57.57	59.10	2.03	0.1	7	21	17	17	0.01	0.26	56.72	191.82	191.66	0.00	25.87	2.86	2.17
2.17	log-slicing_bvsmod_13	20	360.26	438.26	387.37	50.89	1.13	0.3	40	174	46	105	0.11	2.48	383.25	860.04	858.66	0.00	28.23	1.44	2.13
2.18	brummayerbiere2_smulov1bw24	20	134.33	132.01	91.00	41.01	1.45	0.2	28	136	48	10	0.02	0.60	89.76	246.11	244.70	0.00	15.93	0.77	2.18
2.19	...yerbiere3_isqrtinvalidvc	10	181.66	15.89	4.11	11.78	3.87	0.0	5	11	8	0	0.00	0.04	3.57	6.44	6.32	0.00	1.02	0.31	2.20
2.20	...biere3_isqrtaddinvalidvc	10	185.60	16.00	4.83	11.17	3.31	0.0	5	11	8	0	0.00	0.04	4.31	7.16	7.04	0.00	1.36	0.34	3.12
2.21	bmc-bv_magic	10	443.86	453.44	452.57	0.87	1.00	0.1	0	1	1	0	0.00	0.00	0.46	0.46	0.39	0.46	0.46	0.46	2.21
2.22	...ayerbiere2_smulov4bw1024	10	1518.87	10.91	10.40	0.51	1.05	0.5	0	1	1	0	0.00	0.00	9.31	9.31	0.71	9.31	9.31	9.31	2.22
2.23	...mayerbiere2_smulov2bw384	20	122.06	15.64	15.17	0.47	1.03	0.4	0	1	1	0	0.00	0.00	14.33	14.33	8.80	14.33	14.33	14.33	2.23
2.24	...ayerbiere2_smulov4bw0512	10	128.78	2.57	2.17	0.40	1.18	0.1	0	1	1	0	0.00	0.00	1.58	1.58	0.17	1.58	1.58	1.58	2.26
2.25	...ayerbiere2_smulov4bw0640	10	141.93	3.96	3.56	0.40	1.11	0.2	0	1	1	0	0.00	0.00	2.89	2.89	0.27	2.89	2.89	2.89	2.27
2.26	...ayerbiere2_smulov4bw0896	10	1274.52	8.26	7.88	0.38	1.05	0.5	0	1	1	0	0.00	0.00	6.97	6.97	0.54	6.97	6.97	6.97	2.29
2.27	...ayerbiere2_smulov4bw0768	10	563.42	5.80	5.54	0.26	1.05	0.4	0	1	1	0	0.00	0.00	4.70	4.70	0.40	4.70	4.70	4.70	2.28
2.28	brummayerbiere_bitrev8192	20	101.97	101.49	101.24	0.25	1.00	0.0	0	1	1	0	0.00	0.00	0.04	0.04	0.00	0.04	0.04	0.04	2.30
2.29	...mayerbiere2_smulov2bw448	20	150.63	43.69	43.54	0.15	1.00	0.5	0	1	1	0	0.00	0.00	42.59	42.59	34.85	42.59	42.59	42.59	2.24
2.30	...mayerbiere2_smulov2bw512	20	198.38	36.68	36.59	0.09	1.00	0.7	0	1	1	0	0.00	0.00	35.29	35.29	22.98	35.29	35.29	35.29	2.25

Table C.6: Detailed results of PBoolector with high limit configuration 1 (Parallel-Incremental-HighLimit1) compared to the reference times Base-Incremental. Last column par-inc contains cross reference to detailed results of Parallel-Incremental in Tab.C.1.

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental-HighLimit1)																cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-inc
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
1.1	brummayerbiere2_smulov1bw32	20	1056.25	oot	342.10	1457.90	5.26	0.2	27	81	16	0	0.01	1.05	340.46	828.19	826.55	0.00	23.23	5.14	1.1
1.2	log-slicing_bvurem_16	20	oot	oot	958.45	841.55	1.88	0.2	8	24	23	21	0.02	0.33	957.62	2579.59	2579.43	0.01	389.92	31.46	1.2
1.3	log-slicing_bvudiv_18	20	oot	oot	1341.04	458.96	1.34	0.5	9	28	22	21	0.02	5.22	1332.98	4178.89	4178.17	0.00	477.85	46.95	1.7
1.4	...iffyInterleavedModMult-8	20	oot	oot	1461.86	338.14	1.23	0.3	32	204	103	0	0.03	0.62	1460.54	5515.66	5515.05	0.00	39.70	9.71	1.3
1.5	log-slicing_bvsmod_15	20	oot	oot	1629.76	170.24	1.10	0.3	36	156	51	106	0.10	2.36	1626.10	4011.06	4010.13	0.00	143.21	6.99	1.9
2.1	log-slicing_bvudiv_17	20	1270.36	1373.34	552.38	820.96	2.49	0.5	9	26	22	20	0.02	4.80	544.97	1592.23	1591.56	0.00	172.22	19.18	2.1
2.2	log-slicing_bvurem_15	20	oot	1147.71	357.44	790.27	3.21	0.1	8	24	22	19	0.01	0.38	356.61	1044.64	1044.51	0.00	134.70	13.22	2.2
2.3	log-slicing_bvsmod_14	20	1414.54	1350.59	708.56	642.03	1.91	0.3	30	139	46	60	0.08	1.95	705.45	2047.61	2046.80	0.00	39.68	4.69	2.3
2.4	...est_v5_r10_vr1_c1_s32538	20	208.16	996.75	415.43	581.32	2.40	1.8	13	20	8	0	0.55	24.52	385.74	442.54	433.71	0.04	66.91	11.35	4.17
2.5	log-slicing_bvmul_12	20	129.19	962.31	400.83	561.48	2.40	0.1	2	2	2	2	0.00	0.02	400.43	792.35	792.33	1.26	396.55	158.47	2.4
2.6	log-slicing_bvudiv_16	20	390.99	521.12	253.56	267.56	2.06	0.2	8	23	21	17	0.01	0.47	252.53	705.05	704.89	0.01	83.41	9.66	2.8
2.7	log-slicing_bvsmod_13	20	360.26	438.26	259.43	178.83	1.69	0.3	25	121	45	53	0.07	1.60	256.78	813.85	813.19	0.00	18.79	2.19	2.13
2.8	log-slicing_bvurem_14	20	757.16	270.88	148.88	122.00	1.82	0.1	8	24	22	17	0.01	0.30	147.64	453.30	453.17	0.00	60.10	6.04	2.11
2.9	...est_v5_r10_vr5_c1_s13195	10	337.55	662.60	559.79	102.81	1.18	2.1	16	29	8	0	0.76	34.29	518.13	693.54	680.36	0.05	142.65	12.17	4.32
2.10	..._smtcomp09_lf_bv_formula	20	180.24	207.71	113.38	94.33	1.83	1.0	8	40	32	0	0.06	2.74	109.85	342.07	334.84	0.49	23.96	3.60	2.10
2.11	...smtcomp09_std_bv_formula	20	108.86	169.61	77.62	91.99	2.19	0.9	6	24	24	0	0.05	2.44	74.52	219.63	212.72	0.12	23.63	3.99	3.13
2.12	pspace_power2sum.5699	20	347.92	1748.12	1659.51	88.61	1.05	2.0	1	1	1	1	0.00	0.00	1658.97	1658.97	1658.59	111.32	1547.64	829.48	4.4
2.13	float_newton.6.3.i	10	564.04	437.06	351.21	85.85	1.24	1.6	9	17	6	0	0.24	11.97	336.56	562.36	552.16	0.00	74.18	17.04	4.47
2.14	brummayerbiere2_smulov1bw24	20	134.33	132.01	54.07	77.94	2.44	0.1	25	74	12	0	0.01	0.47	53.12	153.58	152.64	0.00	4.14	1.04	2.18
2.15	log-slicing_bvudiv_15	20	162.70	192.95	116.59	76.36	1.65	0.1	8	23	20	15	0.01	0.46	115.62	317.13	316.96	0.00	30.67	4.60	2.15
2.16	log-slicing_bvurem_13	20	147.97	116.67	57.76	58.91	2.02	0.1	7	21	17	16	0.01	0.26	57.01	181.24	181.12	0.00	22.82	2.79	2.17
2.17	log-slicing_bvsmod_12	20	138.58	174.80	117.06	57.74	1.49	0.2	14	80	36	2	0.03	0.97	115.36	408.74	408.33	0.00	13.44	2.14	2.19
2.18	brummayerbiere3_isqrtadd	20	144.73	148.88	105.56	43.32	1.41	0.2	9	44	32	0	0.01	0.24	104.91	347.33	344.21	0.03	15.81	3.37	3.28
2.19	log-slicing_bvslt_7972	20	305.25	931.67	891.51	40.16	1.05	1.0	7	34	23	0	0.01	10.05	877.46	2646.06	2643.46	0.23	168.60	35.76	4.20
2.20	brummayerbiere3_isqrt	20	145.81	148.11	122.46	25.65	1.21	0.2	10	51	34	0	0.01	0.29	121.66	423.76	420.09	0.18	16.20	3.56	3.22
2.21	log-slicing_bvsub_04000	20	545.84	842.77	817.57	25.20	1.03	0.1	1	1	1	1	0.00	0.00	816.65	816.65	815.73	179.19	637.46	408.33	3.6
2.22	lfsr_lfsr_004_111_112	20	178.05	181.61	176.96	4.65	1.03	0.2	2	3	2	0	0.01	1.62	174.81	210.02	209.34	15.86	107.30	42.00	3.32
2.23	...yerbiere3_isqrtinvalidvc	10	181.66	15.89	12.21	3.68	1.30	0.0	2	4	4	0	0.00	0.02	11.86	20.01	19.88	0.75	6.13	2.86	2.20
2.24	bmc-bv_graycode	10	442.54	449.03	447.98	1.05	1.00	0.1	0	1	1	0	0.00	0.00	0.04	0.04	0.01	0.04	0.04	0.04	3.1
2.25	log-slicing_bvsub_05994	20	414.31	1726.25	1725.24	1.01	1.00	0.2	1	1	1	1	0.00	0.00	1723.86	1723.86	1721.87	435.48	1288.38	861.93	4.5
2.26	...mayerbiere2_smulov2bw512	20	198.38	36.68	35.90	0.78	1.02	0.7	0	1	1	0	0.00	0.00	34.84	34.84	22.53	34.84	34.84	34.84	2.25
2.27	bmc-bv_magic	10	443.86	453.44	452.75	0.69	1.00	0.1	0	1	1	0	0.00	0.00	0.46	0.46	0.39	0.46	0.46	0.46	2.21
2.28	...ayerbiere2_smulov4bw1024	10	1518.87	10.91	10.23	0.68	1.07	0.5	0	1	1	0	0.00	0.00	9.34	9.34	0.71	9.34	9.34	9.34	2.22
2.29	...mayerbiere2_smulov2bw448	20	150.63	43.69	43.04	0.65	1.02	0.5	0	1	1	0	0.00	0.00	42.26	42.26	34.49	42.26	42.26	42.26	2.24
2.30	...mayerbiere2_smulov2bw384	20	122.06	15.64	15.02	0.62	1.04	0.4	0	1	1	0	0.00	0.00	14.28	14.28	8.77	14.28	14.28	14.28	2.23
2.31	...mayerbiere2_umulov1bw256	20	143.94	119.05	118.45	0.60	1.01	0.1	0	1	1	0	0.00	0.00	118.06	118.06	116.42	118.06	118.06	118.06	3.5
2.32	RWS_Example_18.txt	10	160.61	193.35	192.76	0.59	1.00	0.1	0	1	1	0	0.00	0.00	192.28	192.28	192.13	192.28	192.28	192.28	3.3
2.33	...ayerbiere2_smulov4bw0512	10	128.78	2.57	2.04	0.53	1.26	0.1	0	1	1	0	0.00	0.00	1.58	1.58	0.17	1.58	1.58	1.58	2.26
2.34	...ayerbiere2_smulov4bw0896	10	1274.52	8.26	7.79	0.47	1.06	0.5	0	1	1	0	0.00	0.00	6.96	6.96	0.54	6.96	6.96	6.96	2.29
2.35	float_mult2.c.50	10	109.39	40.02	39.57	0.45	1.01	0.2	0	1	1	0	0.00	0.00	39.08	39.08	38.42	39.08	39.08	39.08	3.11
2.36	...ayerbiere2_smulov4bw0640	10	141.93	3.96	3.53	0.43	1.12	0.2	0	1	1	0	0.00	0.00	2.90	2.90	0.27	2.90	2.90	2.90	2.27
2.37	...ayerbiere2_smulov4bw0768	10	563.42	5.80	5.37	0.43	1.08	0.4	0	1	1	0	0.00	0.00	4.71	4.71	0.40	4.71	4.71	4.71	2.28
2.38	...mayerbiere2_umulov1bw224	20	108.42	90.16	89.96	0.20	1.00	0.1	0	1	1	0	0.00	0.00	89.53	89.53	88.60	89.53	89.53	89.53	3.4
2.39	lfsr_lfsr_008_159_048	20	100.56	44.32	44.13	0.19	1.00	0.1	0	1	1	0	0.00	0.00	43.63	43.63	43.20	43.63	43.63	43.63	4.93

Table C.7: Detailed results of PBoolector with high limit configuration 2 (Parallel-Incremental-HighLimit2) compared to the reference times Base-Incremental. Last column par-inc contains cross reference to detailed results of Parallel-Incremental in Tab.C.1.

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental-HighLimit2)																cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-inc
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
1.1	brummayerbiere2_smulov1bw32	20	1056.25	oot	311.82	1488.18	5.77	0.3	28	77	14	0	0.01	1.05	310.27	930.65	928.94	0.02	51.14	6.08	1.1
1.2	lfsr_lfsr_008_159_128	20	285.74	oot	1004.41	795.59	1.79	0.8	3	6	4	0	0.07	12.59	989.61	1549.11	1545.61	15.46	621.20	140.83	5.26
1.3	log-slicing_bvurem_16	20	oot	oot	1092.57	707.43	1.65	0.2	8	24	22	20	0.01	0.36	1091.77	2897.49	2897.35	0.00	375.09	36.22	1.2
1.4	...st_v7_r12_vr10_c1_s15994	10	1367.71	oot	1151.79	648.21	1.56	1.4	26	27	2	0	2.16	76.11	1062.83	1091.12	1074.47	0.26	149.69	20.59	5.19
1.5	log-slicing_bvudiv_18	20	oot	oot	1393.56	406.44	1.29	0.6	9	28	22	21	0.02	5.15	1385.67	4238.81	4238.09	0.00	510.35	47.63	1.7
1.6	...iffyInterleavedModMult-8	20	oot	oot	1526.93	273.07	1.18	0.3	22	142	77	0	0.02	0.44	1525.93	5633.49	5633.01	0.00	69.73	15.14	1.3
1.7	log-slicing_bvult_9988	20	555.03	oot	1621.09	178.91	1.11	0.6	5	11	6	0	0.01	7.21	1609.63	3588.81	3586.87	1.15	680.62	170.90	5.115
1.8	log-slicing_bvsmod_15	20	oot	oot	1702.72	97.28	1.06	0.4	23	126	50	24	0.07	1.94	1699.69	5410.26	5409.52	0.00	110.60	16.15	1.9
1.9	...est_v7_r12_vr5_c1_s29826	10	896.50	oot	1737.65	62.35	1.04	1.4	41	42	2	0	3.22	114.48	1603.78	1696.79	1673.38	0.04	170.71	20.44	5.21
2.1	log-slicing_bvudiv_17	20	1270.36	1373.34	609.86	763.48	2.25	0.5	9	26	21	18	0.02	4.90	602.38	1672.71	1672.01	0.01	162.15	20.91	2.1
2.2	log-slicing_bvsmod_14	20	1414.54	1350.59	631.96	718.63	2.14	0.3	19	108	39	14	0.05	1.54	629.56	2218.77	2218.18	0.00	54.88	8.16	2.3
2.3	log-slicing_bvurem_15	20	oot	1147.71	489.62	658.09	2.34	0.1	8	24	22	17	0.01	0.36	488.72	1087.90	1087.75	0.00	123.65	14.51	2.2
2.4	...st_v5_r15_vr10_c1_s11127	10	346.03	1009.99	563.42	446.57	1.79	1.2	10	11	2	0	0.64	24.38	534.20	583.15	576.77	0.06	160.46	27.77	3.8
2.5	log-slicing_bvmul_12	20	129.19	962.31	545.63	416.68	1.76	0.1	2	2	2	2	0.00	0.02	545.33	944.01	943.98	2.66	534.46	188.80	2.4
2.6	...est_v5_r10_vr1_c1_s13516	10	231.36	574.53	266.12	308.41	2.16	0.8	5	6	2	0	0.18	8.94	255.24	261.16	257.33	0.17	127.42	23.74	4.41
2.7	log-slicing_bvslt_7972	20	305.25	931.67	626.16	305.51	1.49	0.3	4	8	4	0	0.00	2.86	622.07	1039.39	1038.60	2.41	385.62	69.29	4.20
2.8	log-slicing_bvudiv_16	20	390.99	521.12	264.08	257.04	1.97	0.2	8	23	20	15	0.01	0.50	263.10	756.01	755.84	0.00	65.55	10.80	2.8
2.9	..._test_v7_r7_vr1_c1_s4574	10	609.33	1077.24	914.71	162.53	1.18	1.2	29	33	4	0	1.18	50.23	856.18	1015.39	1003.26	0.03	121.79	15.62	4.14
2.10	log-slicing_bvsmod_13	20	360.26	438.26	290.63	147.63	1.51	0.2	15	74	29	4	0.03	1.02	288.89	952.52	952.13	0.00	32.47	5.70	2.13
2.11	log-slicing_bvult_6000	20	406.38	608.33	460.85	147.48	1.32	0.2	3	5	4	0	0.00	1.75	458.21	778.99	778.44	0.63	284.64	86.55	4.34
2.12	log-slicing_bvurem_14	20	757.16	270.88	158.17	112.71	1.71	0.1	7	22	19	17	0.01	0.30	157.47	514.09	513.97	0.00	57.50	7.56	2.11
2.13	...est_v7_r7_vr10_c1_s24535	10	237.88	246.30	142.13	104.17	1.73	0.5	1	2	2	0	0.03	1.85	139.45	141.78	139.40	2.32	120.41	47.26	3.10
2.14	log-slicing_bvult_6997	20	201.19	669.29	576.12	93.17	1.16	0.2	2	4	4	0	0.00	1.32	574.11	813.99	813.49	25.75	391.94	116.28	4.31
2.15	log-slicing_bvudiv_15	20	162.70	192.95	114.53	78.42	1.68	0.1	7	20	18	13	0.01	0.41	113.71	365.25	365.10	0.01	26.52	6.19	2.15
2.16	log-slicing_bvsmod_12	20	138.58	174.80	118.90	55.90	1.47	0.1	9	45	22	0	0.02	0.56	117.81	390.28	390.04	0.00	22.53	4.11	2.19
2.17	brummayerbiere3_isqrtadd	20	144.73	148.88	95.57	53.31	1.56	0.1	5	22	16	0	0.00	0.11	95.09	285.40	283.92	0.03	18.79	6.64	3.28
2.18	brummayerbiere3_isqrt	20	145.81	148.11	103.77	44.34	1.43	0.1	6	23	16	0	0.00	0.13	103.30	308.38	306.86	0.04	22.92	6.85	3.22
2.19	brummayerbiere2_smulov1bw24	20	134.33	132.01	88.49	43.52	1.49	0.1	17	32	4	0	0.01	0.25	87.79	155.20	154.74	0.08	11.40	2.46	2.18
2.20	log-slicing_bvurem_13	20	147.97	116.67	75.57	41.10	1.54	0.1	7	21	15	11	0.01	0.25	74.94	208.69	208.58	0.00	23.83	3.79	2.17
2.21	log-slicing_bvult_7994	20	230.10	691.43	656.61	34.82	1.05	0.3	2	4	4	0	0.00	1.50	654.40	857.55	857.00	28.43	513.51	122.51	4.29
2.22	pspace_shift1add.22961	20	110.71	110.86	107.97	2.89	1.03	1.1	0	1	1	0	0.00	0.00	107.60	107.60	107.11	107.60	107.60	107.60	3.17
2.23	pspace_shift1add.25952	20	143.28	129.38	127.10	2.28	1.02	1.3	0	1	1	0	0.00	0.00	126.66	126.66	126.11	126.66	126.66	126.66	3.20
2.24	lfsr_lfsr_008_159_080	20	194.33	310.35	308.12	2.23	1.01	0.3	0	1	1	0	0.00	0.00	307.67	307.67	306.94	307.67	307.67	307.67	4.61
2.25	core_ext_con_048_002_1024	20	487.72	1716.64	1714.75	1.89	1.00	0.3	0	1	1	0	0.00	0.00	1714.40	1714.40	1714.10	1714.40	1714.40	1714.40	2.5
2.26	...mayerbiere2_umulov1bw256	20	143.94	119.05	117.29	1.76	1.02	0.1	0	1	1	0	0.00	0.00	116.96	116.96	115.33	116.96	116.96	116.96	3.5
2.27	lfsr_lfsr_008_159_048	20	100.56	44.32	42.58	1.74	1.04	0.1	0	1	1	0	0.00	0.00	42.17	42.17	41.73	42.17	42.17	42.17	4.93
2.28	pspace_shift1add.29940	20	195.38	157.23	155.52	1.71	1.01	1.4	0	1	1	0	0.00	0.00	155.16	155.16	154.53	155.16	155.16	155.16	3.26
2.29	pspace_shift1add.28943	20	186.84	148.33	146.71	1.62	1.01	1.4	0	1	1	0	0.00	0.00	146.30	146.30	145.68	146.30	146.30	146.30	3.25
2.30	lfsr_lfsr_008_159_096	20	275.86	145.85	144.26	1.59	1.01	0.2	0	1	1	0	0.00	0.00	143.80	143.80	142.96	143.80	143.80	143.80	4.82

Continued on next page

Table C.7 Continued from previous page

rank	benchmark	stat	sequ-btor	ref	PBoolector (Parallel-Incremental-HighLimit2)																cmp
				base-inc	tot(wc)	diff	su	mem	rnds	subproblems			lkhd	split	search						par-inc
										tot	max	fin	tot(wc)	tot(wc)	tot(wc)	tot(cpu)	sat(cpu)	min	max	mean	
[]	[]	[]	[s]	[s]	[s]	[s]	[]	[GB]	[]	[]	[]	[]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[s]	[]
2.31	bmc-bv_magic	10	443.86	453.44	452.44	1.00	1.00	0.1	0	1	1	0	0.00	0.00	0.46	0.46	0.39	0.46	0.46	0.46	2.21
2.32	lfsr_lfsr_008_111_096	20	133.32	238.72	237.74	0.98	1.00	0.2	0	1	1	0	0.00	0.00	237.29	237.29	236.70	237.29	237.29	237.29	4.71
2.33	pspace_shift1add.23958	20	114.08	115.50	114.65	0.85	1.01	1.2	0	1	1	0	0.00	0.00	114.29	114.29	113.77	114.29	114.29	114.29	3.18
2.34	...ayerbiere2_smulov4bw1024	10	1518.87	10.91	10.16	0.75	1.07	0.5	0	1	1	0	0.00	0.00	9.31	9.31	0.71	9.31	9.31	9.31	2.22
2.35	...biere3_isqrtaddinvalidvc	10	185.60	16.00	15.30	0.70	1.05	0.0	0	1	1	0	0.00	0.00	15.02	15.02	15.00	15.02	15.02	15.02	3.12
2.36	pspace_shift1add.27946	20	167.81	143.34	142.66	0.68	1.00	1.3	0	1	1	0	0.00	0.00	141.82	141.82	141.23	141.82	141.82	141.82	3.23
2.37	pspace_power2sum.5699	20	347.92	1748.12	1747.51	0.61	1.00	2.0	0	1	1	0	0.00	0.00	1747.07	1747.07	1746.68	1747.07	1747.07	1747.07	4.4
2.38	...ayerbiere2_smulov4bw0640	10	141.93	3.96	3.37	0.59	1.18	0.2	0	1	1	0	0.00	0.00	2.89	2.89	0.27	2.89	2.89	2.89	2.27
2.39	...ayerbiere2_smulov4bw0896	10	1274.52	8.26	7.68	0.58	1.08	0.5	0	1	1	0	0.00	0.00	6.95	6.95	0.54	6.95	6.95	6.95	2.29
2.40	...est_v5_r10_vr1_c1_s32538	20	208.16	996.75	996.18	0.57	1.00	0.9	26	27	2	0	1.11	44.17	944.87	980.78	969.81	0.04	125.34	18.51	4.17
2.41	...mayerbiere2_smulov2bw384	20	122.06	15.64	15.08	0.56	1.04	0.4	0	1	1	0	0.00	0.00	14.36	14.36	8.82	14.36	14.36	14.36	2.23
2.42	brummayerbiere_bitrev8192	20	101.97	101.49	100.95	0.54	1.01	0.0	0	1	1	0	0.00	0.00	0.04	0.04	0.00	0.04	0.04	0.04	2.30
2.43	...ayerbiere2_smulov4bw0512	10	128.78	2.57	2.05	0.52	1.25	0.1	0	1	1	0	0.00	0.00	1.57	1.57	0.17	1.57	1.57	1.57	2.26
2.44	...ayerbiere2_smulov4bw0768	10	563.42	5.80	5.32	0.48	1.09	0.4	0	1	1	0	0.00	0.00	4.71	4.71	0.40	4.71	4.71	4.71	2.28
2.45	float_mult2.c.50	10	109.39	40.02	39.58	0.44	1.01	0.2	0	1	1	0	0.00	0.00	39.13	39.13	38.47	39.13	39.13	39.13	3.11
2.46	lfsr_lfsr_008_159_064	20	149.52	83.48	83.05	0.43	1.01	0.1	0	1	1	0	0.00	0.00	82.62	82.62	82.07	82.62	82.62	82.62	4.92
2.47	...mayerbiere2_smulov2bw448	20	150.63	43.69	43.27	0.42	1.01	0.5	0	1	1	0	0.00	0.00	42.51	42.51	34.76	42.51	42.51	42.51	2.24
2.48	...yerbiere3_isqrtinvalidvc	10	181.66	15.89	15.48	0.41	1.03	0.0	0	1	1	0	0.00	0.00	15.11	15.11	15.10	15.11	15.11	15.11	2.20
2.49	...mayerbiere2_umulov1bw224	20	108.42	90.16	89.77	0.39	1.00	0.1	0	1	1	0	0.00	0.00	89.36	89.36	88.43	89.36	89.36	89.36	3.4
2.50	pspace_shift1add.24955	20	128.47	121.79	121.45	0.34	1.00	1.2	0	1	1	0	0.00	0.00	121.02	121.02	120.48	121.02	121.02	121.02	3.19
2.51	...mayerbiere2_smulov2bw512	20	198.38	36.68	36.36	0.32	1.01	0.6	0	1	1	0	0.00	0.00	35.28	35.28	22.96	35.28	35.28	35.28	2.25
2.52	core_ext_con_044_002_1024	20	195.82	1571.35	1571.10	0.25	1.00	0.3	0	1	1	0	0.00	0.00	1570.72	1570.72	1570.44	1570.72	1570.72	1570.72	2.6
2.53	bmc-bv_graycode	10	442.54	449.03	448.80	0.23	1.00	0.1	0	1	1	0	0.00	0.00	0.04	0.04	0.01	0.04	0.04	0.04	3.1
2.54	lfsr_lfsr_004_111_112	20	178.05	181.61	181.44	0.17	1.00	0.1	0	1	1	0	0.00	0.00	181.04	181.04	180.72	181.04	181.04	181.04	3.32
2.55	pspace_power2sum.6097	20	388.12	1779.43	1779.26	0.17	1.00	2.2	0	1	1	0	0.00	0.00	1778.63	1778.63	1778.21	1778.63	1778.63	1778.63	4.1

Bibliography

[1] M. Aigner, A. Biere, C.M. Kirsch, A. Niemetz, and M. Preiner. Analysis of Portfolio-Style Parallel SAT Solving on Current Multi-Core Architectures. In *Proc. Workshop on Pragmatics of SAT (PoS)*, EPiC. EasyChair, 2013.

[2] Hamid R. Arabnia, editor. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2009, Las Vegas, Nevada, USA, July 13-17, 2009, 2 Volumes*. CSREA Press, 2009.

[3] Adrian Balint, Anaton Belov, Marjin Heule, and Matti Järvisalo, editors. *Proceedings of SAT Competition 2013*, volume B-2013-1 of *Department of Computer Science Series of Publications B*. University of Helsinki, 2013.

[4] Adrian Balint, Anaton Belov, Marjin Heule, and Matti Järvisalo, editors. *Proceedings of SAT Competition 2013*, volume B-2014-2 of *Department of Computer Science Series of Publications B*. University of Helsinki, 2014.

[5] Clark Barrett, Morgan Deters, Leonardo Mendonça de Moura, Albert Oliveras, and Aaron Stump. 6 Years of SMT-COMP. *J. Autom. Reasoning*, 50(3):243–277, 2013.

[6] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. Technical report, Department of Computer Science, The University of Iowa, 2010. Available at `www.SMT-LIB.org`.

[7] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability Modulo Theories. In Biere et al. [14], pages 825–885.

[8] Armin Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. Technical report, 2010.

[9] Armin Biere. Lingeling and Friends at the SAT Competition 2011. Technical report, 2011.

[10] Armin Biere. Lingeling and Friends Entering the SAT Challenge 2012. In A. Balint, A. Belov, A. Diepold, S. Gerber, M. Järvisalo, and C. Sinz, editors, *Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions*, volume B-2012-2 of *Department of Computer Science Series of Publications B, University of Helsinki*, pages 33–34, 2012.

[11] Armin Biere. Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013. In Balint et al. [3], pages 51–52.

[12] Armin Biere. Yet another Local Search Solver and Lingeling and Friends Entering the SAT Competition 2014. In Balint et al. [4], pages 39–40.

[13] Armin Biere and Roderick Bloem, editors. *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*. Springer, 2014.

[14] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[15] Ahmed Bouajjani and Oded Maler, editors. *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*. Springer, 2009.

[16] Aaron R. Bradley and Zohar Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[17] Robert Brummayer. *Efficient SMT Solving for Bit-Vectors and the Extensional Theory of Arrays*. PhD Thesis, Johannes Kepler University Linz, 2009.

[18] Robert Brummayer and Armin Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In Kowalewski and Philippou [46], pages 174–177.

[19] Robert Brummayer and Armin Biere. Lemmas on demand for the extensional theory of arrays. *JSAT*, 6(1-3):165–201, 2009.

[20] Robert Brummayer, Armin Biere, and Florian Lonsing. BTOR: bit-precise modelling of word-level problems for model checking. In *Proc. 1st International Workshop on Bit-Precise Reasoning*, pages 33–38, New York, NY, USA, 2008. ACM.

[21] Andrei A. Bulatov and Arseny M. Shur, editors. *Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings*, volume 7913 of *Lecture Notes in Computer Science*. Springer, 2013.

[22] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT Solver. In Piterman and Smolka [53], pages 93–107.

[23] Alessandro Cimatti and Roberto Sebastiani, editors. *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*. Springer, 2012.

[24] David R. Cok, Alberto Griggio, Roberto Bruttomesso, and Morgan Deters. The 2012 SMT Competition. In Fontaine and Goel [31], pages 131–142.

[25] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971.

[26] Werner Damm and Holger Hermanns, editors. *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*. Springer, 2007.

[27] Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *J. ACM*, 7(3):201–215, 1960.

[28] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In Ramakrishnan and Rehof [55], pages 337–340.

[29] Manfred Dietrich, editor. *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV), Dresden, Germany, February 22-24, 2010*. Fraunhofer Verlag, 2010.

[30] Kerstin Eder, João Lourenço, and Onn Shehory, editors. *Hardware and Software: Verification and Testing - 7th International Haifa Verification Conference, HVC 2011, Haifa, Israel, December 6-8, 2011, Revised Selected Papers*, volume 7261 of *Lecture Notes in Computer Science*. Springer, 2012.

[31] Pascal Fontaine and Amit Goel, editors. *10th International Workshop on Satisfiability Modulo Theories, SMT 2012, Manchester, UK, June 30 - July 1, 2012*, volume 20 of *EPiC Series*. EasyChair, 2013.

[32] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure. *JSAT*, 1(3-4):209–236, 2007.

[33] Andreas Fröhlich, Gergely Kovásznai, and Armin Biere. More on the Complexity of Quantifier-Free Fixed-Size Bit-Vector Logics with Binary Encoding. In Bulatov and Shur [21], pages 378–390.

[34] Malay K. Ganai and Alper Sen, editors. *Proceedings of the Second International Workshop on Design and Implementation of Formal Tools and Systems, Portland, OR, USA, October 19, 2013*, volume 1130 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[35] Vijay Ganesh and David L. Dill. A Decision Procedure for Bit-Vectors and Arrays. In Damm and Hermanns [26], pages 519–531.

[36] Liana Hadarean, Kshitij Bansal, Dejan Jovanovic, Clark Barrett, and Cesare Tinelli. A Tale of Two Solvers: Eager and Lazy Approaches to Bit-Vectors. In Biere and Bloem [13], pages 680–695.

[37] Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. ManySAT: a Parallel SAT Solver. *JSAT*, 6(4):245–262, 2009.

[38] Marijn Heule, Mark Dufour, Joris van Zwieten, and Hans van Maaren. March_eq: Implementing Additional Reasoning into an Efficient Look-Ahead SAT Solver. In Hoos and Mitchell [42], pages 345–359.

[39] Marijn Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads. In Eder et al. [30], pages 50–65.

[40] Marijn Heule and Hans van Maaren. Look-Ahead Based SAT Solvers. In Biere et al. [14], pages 155–184.

[41] M.J.H. Heule. Smart solving, 2008.

[42] Holger H. Hoos and David G. Mitchell, editors. *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*. Springer, 2005.

[43] S. Hölldobler, V. H. Nguyen, J. Stocklina, and P. Steinke. A short overview on modern parallel SAT-solvers. *Proceedings of the International Conference on Advanced Computer Science and Information System*, pages 201–206, 2011.

[44] Natalia Kalinnik, Erika Ábrahám, Tobias Schubert, Ralf Wimmer, and Bernd Becker. Exploiting Different Strategies for the Parallelization of an SMT Solver. In Dietrich [29], pages 97–106.

[45] Natalia Kalinnik, Tobias Schubert, Erika Ábrahám, Ralf Wimmer, and Bernd Becker. Picoso - A Parallel Interval Constraint Solver. In Arabnia [2], pages 473–479.

[46] Stefan Kowalewski and Anna Philippou, editors. *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*. Springer, 2009.

[47] Daniel Kroening and Ofer Strichman. *Decision Procedures – an Algorithmic Point of View*. EATCS. Springer, 2008.

[48] Oliver Kullmann. Fundaments of Branching Heuristics. In Biere et al. [14], pages 205–244.

[49] Jiang Long, Robert K. Brayton, and Michael L. Case. LEC: Learning Driven Data-path Equivalence Checking. In Ganai and Sen [34].

[50] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. An overview of parallel SAT solving. *Constraints*, 17(3):304–347, 2012.

[51] Aina Niemetz, Mathias Preiner, and Armin Biere. Turbo-Charging Lemmas on Demand with Don't Care Reasoning. In *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design FMCAD 2014*, 2014.

[52] Kullmann Oliver. Investigating the behaviour of a SAT solver on random formulas. Technical report, University of Wales Swansea, October 2002.

[53] Nir Piterman and Scott A. Smolka, editors. *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7795 of *Lecture Notes in Computer Science*. Springer, 2013.

[54] Mathias Preiner, Aina Niemetz, and Armin Biere. Lemmas on Demand for Lambdas. In Ganai and Sen [34].

[55] C. R. Ramakrishnan and Jakob Rehof, editors. *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*. Springer, 2008.

[56] João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-Driven Clause Learning SAT Solvers. In Biere et al. [14], pages 131–153.

[57] G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, Part II, pages 115–125, 1970.

[58] Peter van der Tak, Marijn Heule, and Armin Biere. Concurrent Cube-and-Conquer - (Poster Presentation). In Cimatti and Sebastiani [23], pages 475–476.

[59] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Mendonça de Moura. A Concurrent Portfolio Approach to SMT Solving. In Bouajjani and Maler [15], pages 715–720.

europass Lebenslauf

ANGABEN ZUR PERSON

Christian Reisenberger

Obere Donaulände 11/6, 4020 Linz (Österreich)

+43 69981452872

chris.reisenberger@gmx.at

Geschlecht Männlich | Geburtsdatum 26/10/1980 | Staatsangehörigkeit österreichisch

SCHUL- UND BERUFSBILDUNG

10/2011–Heute

Johannes Kepler Universität, Linz

Masterstudium Informatik

10/2008–08/2011

Bachelor of Science

Johannes Kepler Universität, Linz

Bachelorstudium Informatik

10/2006–02/2008

Fachhochschule, Hagenberg

Bachelorstudium Hard- und Software System Engineering

09/1995–06/2000

Matura

Höhere Technische Lehranstalt, Wels

Fachbereich Elektrotechnik

BERUFSERFAHRUNG

05/2013–05/2014

Softwareentwickler

Selbstständig

04/2011–04/2013

Industriepraktikum Software Tester

Fujitsu Semiconductor Embedded Solutions Austria GMBH, Linz

05/2008–12/2008

Softwareentwickler

Ars Electronica FutureLab, Linz

05/2005–06/2006

Auslandsaufenthalt in Zentralamerika

- Sechsmonatiges Volontariat bei einem Sozialprojekt in Nicaragua (Granada).
- 1 Monat internationaler Friedensbeobachter in Mexiko (Chiapas)

12/2001–03/2005

Lichttechniker

Theater Phönix, Linz

10/2000–09/2001

Zivildienst

Das Dorf, Altenhof

PERSÖNLICHE FÄHIGKEITEN

Muttersprache(n) Deutsch

Weitere Sprache(n)

	VERSTEHEN		SPRECHEN		SCHREIBEN
	Hören	Lesen	An Gesprächen teilnehmen	Zusammenhängendes Sprechen	
Englisch	C1	C1	B2	B2	C1
Spanisch	B1	B1	B1	B1	B1

A1/2: elementare Sprachverwendung - B1/2: selbstständige Sprachverwendung - C1/2: kompetente Sprachverwendung
Gemeinsamer Europäischer Referenzrahmen für Sprachen

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, am 20. November 2014

Christian Reisenberger, BSc