Automated Benchmarking of Incremental SAT and QBF Solvers

Uwe Egly Florian Lonsing Johannes Oetsch

Knowledge-Based Systems Group, Vienna University of Technology, Austria

20th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, 24 - 28 November 2015, Suva, Fiji

This work is supported by the Austrian Science Fund (FWF) under grant S11409-N23.

Introduction

Propositional Logic (SAT):

- Modelling NP-complete problems in formal verification, AI, ...

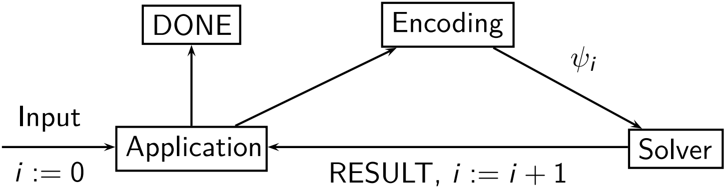
Quantified Boolean Formulas (QBF):

- Existential and universal quantification of propositional variables.
- $Q_1 x_1, \ldots, Q_n x_n.\ \phi$, where $Q_i \in \{\forall, \exists\}$ and ϕ a CNF.
- PSPACE-complete: potentially more succinct encodings than SAT.

Practice:

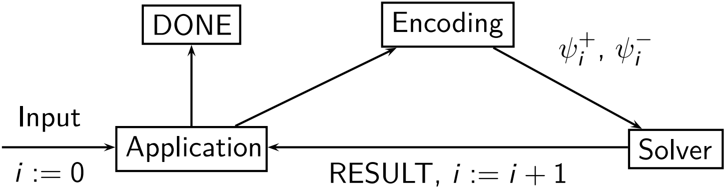
- Despite intractability, solvers often work well on structured problems.
- Applications to problems of higher complexity, e.g. NEXPTIME.
- SAT/QBF solvers are tightly integrated in application workflows.

Abstract Non-Incremental Workflow



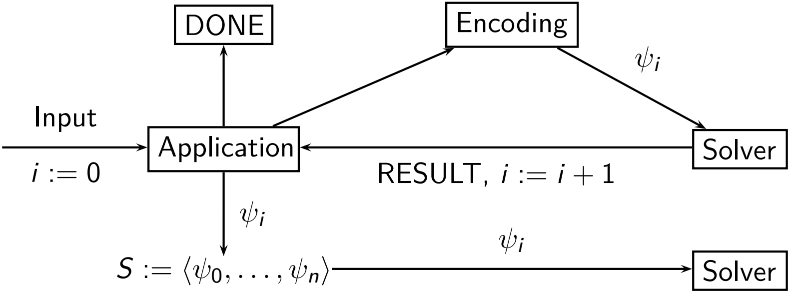
- Application program: bounded model checker, synthesis tool,...
- Input problem solved in stepwise fashion.
- Step i: formula ψ_i written to hard disk or imported by solver via API.
- Solver starts from scratch in each step i: potential redundant work.
- Sequence $\langle \psi_0, \ldots, \psi_n \rangle$ of syntactically related formulas.

Abstract Incremental Workflow



- Step $i = 0$: solver receives initial formula ψ_0.
- Step $i > 0$: solver receives and solves current ψ_i incrementally.
- $\psi_i := (\psi_{i-1} \setminus \psi_i^-) \cup \psi_i^+$ obtained by adding ψ_i^+ and deleting ψ_i^-.
- Solver called incrementally: keep information learned in previous calls.
- Sequence $\langle \psi_0, \ldots, \psi_n \rangle$ compactly represented by ψ_0 and ψ_i^+, ψ_i^-.

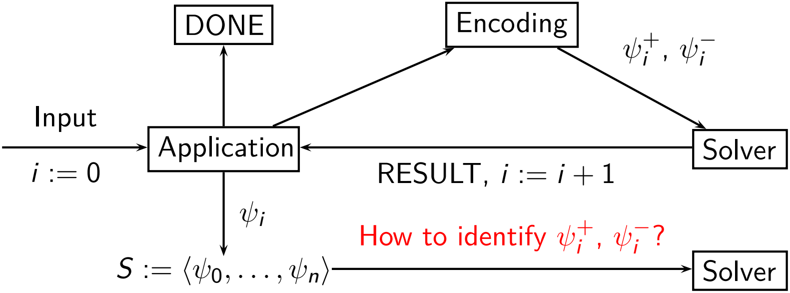
SAT/QBF Solving in Practice



Benchmarking:

- Solver performance is crucial for practical applications.
- Solver development relies on publicly available benchmarks.
- Benchmarks generated by application programs.
- So far: focus on *non-incremental* solving.

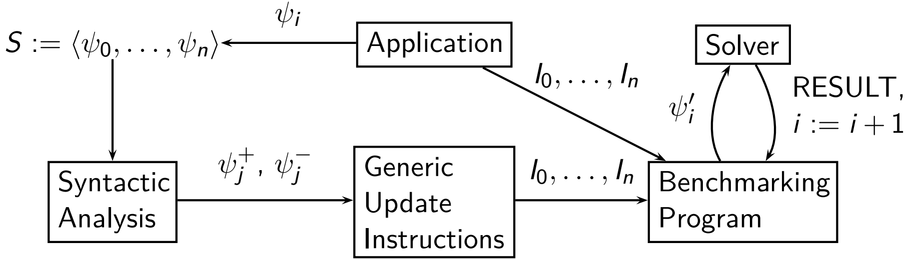
SAT/QBF Solving in Practice



Problem:

- Lack of benchmarks for *incremental* solvers.
- Lack of application programs used to generate formula sequences.
- How to solve available formula sequence $\langle \psi_0, \ldots, \psi_n \rangle$ incrementally?
- So far: incremental solvers tightly coupled with application programs.

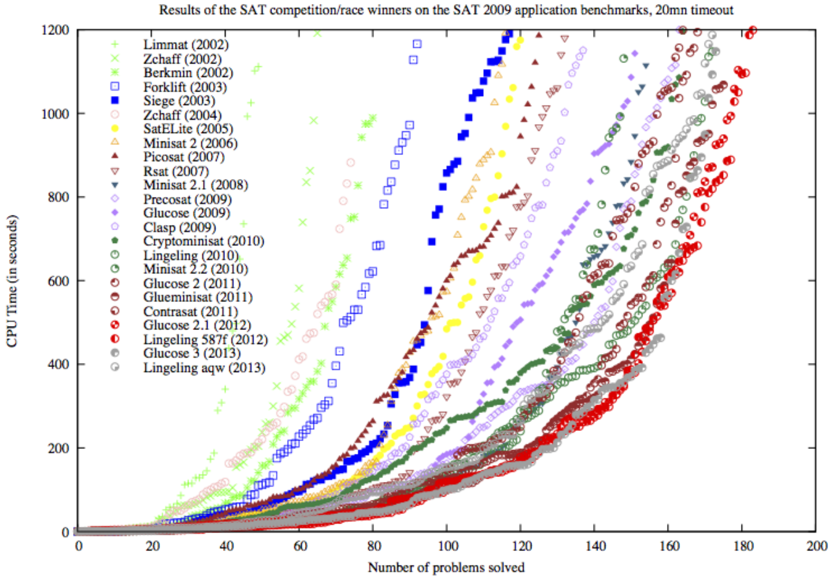
Contributions



Automated Benchmarking:

- Translate sequence S of related formulas into incremental solver calls.
- Identify incremental formula updates: $\psi_i := (\psi_{i-1} \setminus \psi_i^-) \cup \psi_i^+$.
- Compact representation of S by generic update instructions.
- Benchmarking program calls incremental solvers via standardized API.
- Tools used in the *Incremental Library Track* of the *SAT Race 2015*.

Progress in Non-Incremental SAT Solving



Data and plot produced by Daniel Le Berre.

SAT/QBF Competitions

Competition Drives Innovation:

- Annual SAT-related events since 2002: SAT Competitions / Races.
- QBFEVALs (2004-2008, 2010, 2012), QBF Galleries (2013, 2014).
- Solver developers invent new technology: enables new applications.

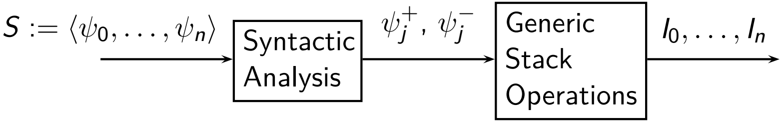
Problem:

- So far, competitions have focused on *non*-incremental solving.
- No benchmarks (i.e. formula sequences) to test incremental solvers.

Our Approach:

- Conversion of available formula sequences into a standardized format.
- Comparison of incremental solvers on the standardized sequence.

Analyzing Formula Sequences



Incremental Updates in $S := \langle \psi_0, \ldots, \psi_n \rangle$:

- Cumulative clauses: appear in ψ_i first and in *all* ψ_j with $i < j$.
- Volatile clauses: appear in ψ_i and are removed to obtain ψ_j with $i < j$.

Stack-Based Representation of $\psi_j := (\psi_{j-1} \setminus \psi_j^-) \cup \psi_j^+$:

- Deletion of volatile clauses ψ_j^- by $\texttt{pop} \in I_j$.
- Temporary addition of volatile clauses ψ_j^+ by $\texttt{push} \in I_j$.
- Permanent addition of cumulative clauses ψ_j^+ by $\texttt{add} \in I_j$.

Stack-Based Formula Representation: Example

Given sequence $S := (\psi_0, \ldots, \psi_3)$ of formulas.

Formula ψ_i:	Cumulative in ψ_i:	Volatile in ψ_i:
$\psi_0 = \{c_1, c_2, vc_1\}$	$C_0 = \{c_1, c_2\}$	$VC_0 = \{vc_1\}$
$\psi_1 = \{c_1, c_2, c_3, vc_1, vc_2\}$	$C_1 = \{c_3\}$	$VC_1 = \{vc_1, vc_2\}$
$\psi_2 = \{c_1, c_2, c_3, c_4, vc_1, vc_3\}$	$C_2 = \{c_4\}$	$VC_2 = \{vc_1, vc_3\}$
$\psi_3 = \{c_1, c_2, c_3, c_4, c_5\}$	$C_3 = \{c_5\}$	$VC_3 = \emptyset$

Stack operations: $\emptyset$
Clauses on stack: $\emptyset$

- S represented by sequence $I = (I_0, \ldots, I_3)$ of stack operations.
- Sequence I is not unique (e.g. reorderings).
- Benchmarking program translates I to incremental solver calls.
- Sequence I may be extracted from S or directly written by application.

Stack-Based Formula Representation: Example

Given sequence $S := (\psi_0, \ldots, \psi_3)$ of formulas.

Formula ψ_i:	Cumulative in ψ_i:	Volatile in ψ_i:
$\psi_0 = \{c_1, c_2, vc_1\}$	$C_0 = \{c_1, c_2\}$	$VC_0 = \{vc_1\}$
$\psi_1 = \{c_1, c_2, c_3, vc_1, vc_2\}$	$C_1 = \{c_3\}$	$VC_1 = \{vc_1, vc_2\}$
$\psi_2 = \{c_1, c_2, c_3, c_4, vc_1, vc_3\}$	$C_2 = \{c_4\}$	$VC_2 = \{vc_1, vc_3\}$
$\psi_3 = \{c_1, c_2, c_3, c_4, c_5\}$	$C_3 = \{c_5\}$	$VC_3 = \emptyset$

Stack operations: $I_0 = \texttt{add}(C_0), \texttt{push}(VC_0)$
Clauses on stack: $\psi_0 = \{\{c_1, c_2\}, \{vc_1\}\}$

- S represented by sequence $I = (I_0, \ldots, I_3)$ of stack operations.
- Sequence I is not unique (e.g. reorderings).
- Benchmarking program translates I to incremental solver calls.
- Sequence I may be extracted from S or directly written by application.

Stack-Based Formula Representation: Example

Given sequence $S := (\psi_0, \ldots, \psi_3)$ of formulas.

Formula ψ_i:	Cumulative in ψ_i:	Volatile in ψ_i:
$\psi_0 = \{c_1, c_2, vc_1\}$	$C_0 = \{c_1, c_2\}$	$VC_0 = \{vc_1\}$
$\psi_1 = \{c_1, c_2, c_3, vc_1, vc_2\}$	$C_1 = \{c_3\}$	$VC_1 = \{vc_1, vc_2\}$
$\psi_2 = \{c_1, c_2, c_3, c_4, vc_1, vc_3\}$	$C_2 = \{c_4\}$	$VC_2 = \{vc_1, vc_3\}$
$\psi_3 = \{c_1, c_2, c_3, c_4, c_5\}$	$C_3 = \{c_5\}$	$VC_3 = \emptyset$

Stack operations: $I_1 = \texttt{pop}(), \texttt{add}(C_1), \texttt{push}(VC_1)$
Clauses on stack: $\psi_1 = \{\{c_1, c_2\}, \{c_3\}, \{vc_1, vc_2\}\}$

- S represented by sequence $I = (I_0, \ldots, I_3)$ of stack operations.
- Sequence I is not unique (e.g. reorderings).
- Benchmarking program translates I to incremental solver calls.
- Sequence I may be extracted from S or directly written by application.

Stack-Based Formula Representation: Example

Given sequence $S := (\psi_0, \ldots, \psi_3)$ of formulas.

Formula ψ_i:	Cumulative in ψ_i:	Volatile in ψ_i:
$\psi_0 = \{c_1, c_2, vc_1\}$	$C_0 = \{c_1, c_2\}$	$VC_0 = \{vc_1\}$
$\psi_1 = \{c_1, c_2, c_3, vc_1, vc_2\}$	$C_1 = \{c_3\}$	$VC_1 = \{vc_1, vc_2\}$
$\psi_2 = \{c_1, c_2, c_3, c_4, vc_1, vc_3\}$	$C_2 = \{c_4\}$	$VC_2 = \{vc_1, vc_3\}$
$\psi_3 = \{c_1, c_2, c_3, c_4, c_5\}$	$C_3 = \{c_5\}$	$VC_3 = \emptyset$

Stack operations: $I_2 = \texttt{pop}(), \texttt{add}(C_2), \texttt{push}(VC_2)$
Clauses on stack: $\psi_2 = \{\{c_1, c_2\}, \{c_3\}, \{c_4\}, \{vc_1, vc_3\}\}$

- S represented by sequence $I = (I_0, \ldots, I_3)$ of stack operations.
- Sequence I is not unique (e.g. reorderings).
- Benchmarking program translates I to incremental solver calls.
- Sequence I may be extracted from S or directly written by application.

Stack-Based Formula Representation: Example

Given sequence $S := (\psi_0, \ldots, \psi_3)$ of formulas.

Formula ψ_i:	Cumulative in ψ_i:	Volatile in ψ_i:
$\psi_0 = \{c_1, c_2, vc_1\}$	$C_0 = \{c_1, c_2\}$	$VC_0 = \{vc_1\}$
$\psi_1 = \{c_1, c_2, c_3, vc_1, vc_2\}$	$C_1 = \{c_3\}$	$VC_1 = \{vc_1, vc_2\}$
$\psi_2 = \{c_1, c_2, c_3, c_4, vc_1, vc_3\}$	$C_2 = \{c_4\}$	$VC_2 = \{vc_1, vc_3\}$
$\psi_3 = \{c_1, c_2, c_3, c_4, c_5\}$	$C_3 = \{c_5\}$	$VC_3 = \emptyset$

Stack operations: $I_3 = \texttt{pop}(), \texttt{add}(C_3), \texttt{push}(VC_3)$
Clauses on stack: $\psi_3 = \{\{c_1, c_2\}, \{c_3\}, \{c_4\}, \{c_5\}, \emptyset\}$

- S represented by sequence $I = (I_0, \ldots, I_3)$ of stack operations.
- Sequence I is not unique (e.g. reorderings).
- Benchmarking program translates I to incremental solver calls.
- Sequence I may be extracted from S or directly written by application.

Stack-Based Formula Representation: Example

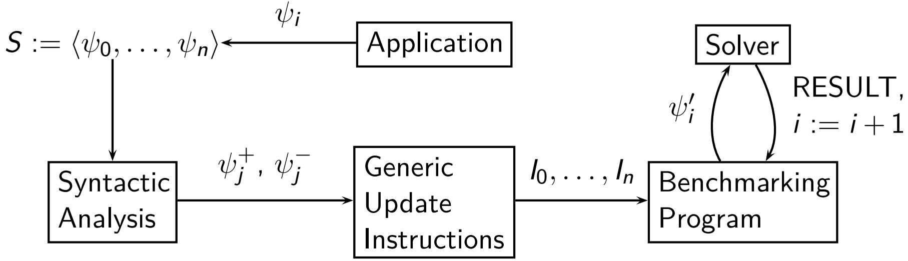
Given sequence $S := (\psi_0, \ldots, \psi_3)$ of formulas.

Formula ψ_i:	Cumulative in ψ_i:	Volatile in ψ_i:
$\psi_0 = \{c_1, c_2, vc_1\}$	$C_0 = \{c_1, c_2\}$	$VC_0 = \{vc_1\}$
$\psi_1 = \{c_1, c_2, c_3, vc_1, vc_2\}$	$C_1 = \{c_3\}$	$VC_1 = \{vc_1, vc_2\}$
$\psi_2 = \{c_1, c_2, c_3, c_4, vc_1, vc_3\}$	$C_2 = \{c_4\}$	$VC_2 = \{vc_1, vc_3\}$
$\psi_3 = \{c_1, c_2, c_3, c_4, c_5\}$	$C_3 = \{c_5\}$	$VC_3 = \emptyset$

Stack operations: $I_3 = \texttt{pop}(), \texttt{add}(C_3), \texttt{push}(VC_3)$
Clauses on stack: $\psi_3 = \{\{c_1, c_2\}, \{c_3\}, \{c_4\}, \{c_5\}, \emptyset\}$

- S represented by sequence $I = (I_0, \ldots, I_3)$ of stack operations.
- Sequence I is not unique (e.g. reorderings).
- Benchmarking program translates I to incremental solver calls.
- Sequence I may be extracted from S or directly written by application.

Generating Sequences of Formulas



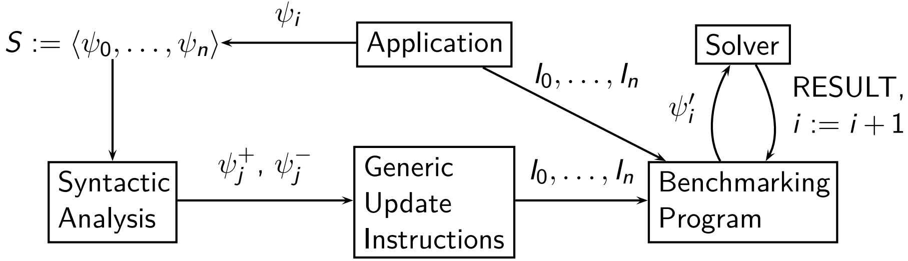
Application Program is not available:

- Convert S into a standardized representation by update instructions.

Application Program is available:

- Integrate solvers directly.
- Represent S directly as sequence of update instructions.
- Set of update instructions is extensible.

Generating Sequences of Formulas



Application Program is not available:

- Convert S into a standardized representation by update instructions.

Application Program is available:

- Integrate solvers directly.
- Represent S directly as sequence of update instructions.
- Set of update instructions is extensible.

Incremental Library Track in the SAT Race 2015

Incremental Library Track – IPASIR

KIT
Karlsruhe Institute of Technology

- IPASIR = Re-entrant Incremental Satisfiability Application Program Interface (acronym reversed)
- IPASIR has 6 methods for SAT solving:
  - add clauses and assumptions (2 methods)
  - set callback for abort
  - solve
  - get model and failed assumptions (2 methods)

Tomáš Balyo, Markus Iser, Carsten Sinz – Sat Race 2015 September 22, 2015 4/20

SAT Race 2015 slides by T. Balyo, M. Iser, C. Sinz.
http://baldur.iti.kit.edu/sat-race-2015/index.php
http://baldur.iti.kit.edu/sat-race-2015/sr15.pdf

Our Contribution:

- Benchmarking program and formula sequences generated from hardware bounded model checking problems.

Incremental Library Track in the SAT Race 2015

Incremental Library Track – Benchmarks

KIT Karlsruhe Institute of Technology

- Partial MaxSat Solving (linear strengthening of a cardinality constraint on soft clauses), 568 pMaxSat problems (industrial track, MaxSat 2014)
- Trivial parallel portfolio SAT solver (clause order diversification), the 100 problems of the parallel track
- Finding all essential (has to be assigned in each satisfying assignment) variables, 50 easiest instances of the main track
- Incremental SAT file interpreter, 50 files generated from HWMCC 2014 instances, 3979 SAT calls in total
  - submitted by Florian Lonsing, Johannes Oetsch, and Uwe Egly

Tomáš Balyo, Markus Iser, Carsten Sinz – Sat Race 2015 September 22, 2015 5/20

SAT Race 2015 slides by T. Balyo, M. Iser, C. Sinz.
http://baldur.iti.kit.edu/sat-race-2015/index.php
http://baldur.iti.kit.edu/sat-race-2015/sr15.pdf

Our Contribution:

- Benchmarking program and formula sequences generated from hardware bounded model checking problems.

Incremental Library Track in the SAT Race 2015

Incremental Library Track – Benchmarks

- Partial MaxSat Solving (linear strengthening of a cardinality constraint on soft clauses), 568 pMaxSat problems (industrial track, MaxSat 2014)
- Trivial parallel portfolio SAT solver (clause order diversification), the 100 problems of the parallel track
- Finding all essential (has to be assigned in each satisfying assignment) variables, 50 easiest instances of the main track
- Incremental SAT file interpreter, 50 files generated from HWMCC 2014 instances, 3979 SAT calls in total
  - submitted by Florian Lonsing, Johannes Oetsch, and Uwe Egly

Tomáš Balyo, Markus Iser, Carsten Sinz – Sat Race 2015 September 22,2015 5/20

SAT Race 2015 slides by T. Balyo, M. Iser, C. Sinz.
http://baldur.iti.kit.edu/sat-race-2015/index.php
http://baldur.iti.kit.edu/sat-race-2015/sr15.pdf

Our Contribution:

- Benchmarking program and formula sequences generated from hardware bounded model checking problems.

Incremental Library Track in the SAT Race 2015

Incremental Library Track – Results

solver name	essent.	pmax	is-file	pfolio	total
#instances	50	568	3979	100	4697
CryptoMiniSat4	**48**	266	**1454**	0	1768
CryptoMiniSat4autotune	47	**271**	1452	0	**1770**
CoMiniSatPs1Earth	45	244	1406	**12**	1707
CoMiniSatPs1Sun	45	250	1434	5	1734
Glucose4	**48**	259	1407	1	1715
Riss505	44	234	1372	4	1654
Riss504	44	244	1370	2	1660
PicoSat961	44	165	1285	5	1499
SatUZK	43	204	842	5	1094

Tomáš Balyo, Markus Iser, Carsten Sinz – Sat Race 2015 September 22, 2015 6/20

SAT Race 2015 slides by T. Balyo, M. Iser, C. Sinz.
http://baldur.iti.kit.edu/sat-race-2015/index.php
http://baldur.iti.kit.edu/sat-race-2015/sr15.pdf

Our Contribution:

- Benchmarking program and formula sequences generated from hardware bounded model checking problems.

Incremental Library Track in the SAT Race 2015

Incremental Library Track – Results

KIT Karlsruhe Institute of Technology

solver name	essent.	pmax	is-file	pfolio	total
#instances	50	568	3979	100	4697
CryptoMiniSat4	**48**	266	**1454**	0	1768
CryptoMiniSat4autotune	47	**271**	1452	0	**1770**
CoMiniSatPs1Earth	45	244	1406	**12**	1707
CoMiniSatPs1Sun	45	250	1434	5	1734
Glucose4	**48**	259	1407	1	1715
Riss505	44	234	1372	4	1654
Riss504	44	244	1370	2	1660
PicoSat961	44	165	1285	5	1499
SatUZK	43	204	842	5	1094

Tomáš Balyo, Markus Iser, Carsten Sinz – Sat Race 2015 September 22,2015 6/20

SAT Race 2015 slides by T. Balyo, M. Iser, C. Sinz.
http://baldur.iti.kit.edu/sat-race-2015/index.php
http://baldur.iti.kit.edu/sat-race-2015/sr15.pdf

Our Contribution:

- Benchmarking program and formula sequences generated from hardware bounded model checking problems.

Conclusion

Automated Benchmarking:

- Decoupled from application program used to generate formulas.
- Compact standardized representation of formula sequences.
- Useful for development of incremental solvers.

Support for Sequences of QBFs:

- Additional instructions to update quantifier prefix of a prenex CNF.

Future Work:

- Application program may depend on a particular solver.
- Do different solvers result in different sequences of formulas?
- How do solvers perform on different sequences?