Tutorial on Word-Level Model Checking

Armin Biere

JʞU

**JOHANNES KEPLER
UNIVERSITÄT LINZ**

FMCAD 2020

September 21, 2020

Online

Tutorial on World-Level Model Checking

Armin Biere
Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria
armin.biere@jku.at

Abstract—In SMT bit-vectors and thus word-level reasoning is common and widely used in industry. However, it took until 2019 that the hardware model checking competition started to use word-level benchmarks. Reasoning on the word-level opens up many possibilities for simplification and more powerful reasoning. In SMT we do see advantages due to operating on the word-level, even though, ultimately, bit-blasting and thus transforming the word-level problem into SAT is still the dominant and most important technique. For word-level model checking the situation is different. As the hardware model checking competition in 2019 has shown bit-level solvers are far superior (after bit-blasting the model through an SMT solver though). On the other hand word-level model checking shines for problems with memory modeled with arrays. In this tutorial we revisit the problem of word level model checking, also from a theoretical perspective, give an overview on classical and more recent approaches for word-level model checking and then discuss challenges and future work. The tutorial covered material from the following papers.

REFERENCES

[1] Z. S. Andraus, M. H. Liffiton, and K. A. Sakallah, "Refinement strategies for verification methods based on datapath abstraction," in *Proc. ASP-DAC'06*. IEEE, 2006, pp. 19–24.

[2] ——, "Reveal: A formal verification tool for Verilog designs," in *Proc. LPAR'08*, ser. LNCS, vol. 5330. Springer, 2008, pp. 343–352.

[3] C. Barrett, P. Fontaine, and C. Tinelli, "The Satisfiability Modulo Theories Library (SMT-LIB)," www.SMT-LIB.org, 2016.

[4] A. Biere, "The AIGER And-Inverter Graph (AIG) format version 20071012," FMV Reports Series, JKU Linz, Tech. Rep., 2007.

[5] A. Biere, K. Heljanko, and S. Wieringa, "AIGER 1.9 and beyond," FMV Reports Series, JKU Linz, Tech. Rep., 2011.

[6] A. Biere and M. Preiner, "Hardware model checking competition 2019," http://fmv.jku.at/hwmcc19.

[7] A. Biere, T. van Dijk, and K. Heljanko, "Hardware model checking competition 2017," in *Proc. FMCAD'17*. IEEE, 2017, p. 9.

[8] P. Bjesse, "A practical approach to word level model checking of industrial netlists," in *Proc. CAV'08*, ser. LNCS, vol. 5123. Springer, 2008, pp. 446–458.

[9] ——, "Word-level sequential memory abstraction for model checking," in *Proc. FMCAD'08*. IEEE, 2008, pp. 1–9.

[10] ——, "Word level bitwidth reduction for unbounded hardware model checking," *Formal Methods Syst. Des.*, vol. 35, no. 1, pp. 56–72, 2009.

[11] R. Brummayer, A. Biere, and F. Lonsing, "BTOR: Bit-precise modelling of word-level problems for model checking," in *Proc. SMT'08*. ACM, 2008, pp. 33–38.

[12] G. Cabodi, C. Loiacono, M. Palena, P. Pasini, D. Patti, S. Quer, D. Vendraminetto, A. Biere, and K. Heljanko, "Hardware model checking competition 2014: An analysis and comparison of solvers and benchmarks," *JSAT*, vol. 9, pp. 135–172, 2014 (published 2016).

[13] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuXmv symbolic model checker," in *Proc. CAV'14*, ser. LNCS, vol. 8559. Springer, 2014, pp. 334–342.

[14] L. De Moura, S. Owre, and N. Shankar, "The SAL language manual," *Computer Science Laboratory, SRI Intl., Tech. Rep. CSL-01-01*, 2003.

[15] S. M. German, "A theory of abstraction for arrays," in *Proc. FMCAD'11*. FMCAD Inc., 2011, pp. 176–185.

[16] A. Goel and K. A. Sakallah, "Empirical evaluation of IC3-based model checking techniques on verilog RTL designs," in *Proc. DATE'19*. IEEE, 2019, pp. 618–621.

[17] ——, "Model checking of Verilog RTL using IC3 with syntax-guided abstraction," in *Proc. NFM'19*, ser. LNCS, vol. 11460. Springer, 2019, pp. 166–185.

[18] ——, "AVR: abstractly verifying reachability," in *Proc. TACAS'20*, ser. LNCS, vol. 12078. Springer, 2020, pp. 413–422.

[19] Y. Ho, A. Mishchenko, and R. K. Brayton, "Property directed reachability with word-level abstraction," in *Proc. FMCAD'17*. IEEE, 2017, pp. 132–139.

[20] K. Hoder, N. Bjørner, and L. M. de Moura, "μZ- an efficient engine for fixed points with constraints," in *Proc. CAV'11*, ser. LNCS, vol. 6806. Springer, 2011, pp. 457–462.

[21] A. Irfan, A. Cimatti, A. Griggio, M. Roveri, and R. Sebastiani, "Verilog2SMV: A tool for word-level verification," in *Proc. DATE'16*. IEEE, 2016, pp. 1156–1159.

[22] H. Jain, D. Kroening, N. Sharygina, and E. M. Clarke, "Word-level predicate-abstraction and refinement techniques for verifying RTL Verilog," *IEEE TCAD*, vol. 27, no. 2, pp. 366–379, 2008.

[23] T. Jussila and A. Biere, "Compressing BMC encodings with QBF," *ENTCS*, vol. 174, no. 3, pp. 45–56, 2007.

[24] A. Kölbl, R. Jacoby, H. Jain, and C. Pixley, "Solver technology for system-level to RTL equivalence checking," in *Proc. DATE'09*. IEEE, 2009, pp. 196–201.

[25] G. Kovásznai, A. Fröhlich, and A. Biere, "Complexity of fixed-size bit-vector logics," *Theory Comp. Sys.*, vol. 59, no. 2, pp. 323–376, 2016.

[26] G. Kovásznai, H. Veith, A. Fröhlich, and A. Biere, "On the complexity of symbolic verification and decision problems in bit-vector logic," in *MFCS'14*, ser. LNCS, vol. 8635. Springer, 2014, pp. 481–492.

[27] D. Kroening, "Computing over-approximations with bounded model checking," *ENTCS*, vol. 144, no. 1, pp. 79–92, 2006.

[28] D. Kroening and S. A. Seshia, "Formal verification at higher levels of abstraction," in *Proc. ICCAD'07*. IEEE Comp. Soc., 2007, pp. 572–578.

[29] S. Lee and K. A. Sakallah, "Unbounded scalable verification based on approximate property-directed reachability and datapath abstraction," in *Proc. CAV'14*, ser. LNCS, vol. 8559. Springer, 2014, pp. 849–865.

[30] J. Long, S. Ray, B. Sterin, A. Mishchenko, and R. K. Brayton, "Enhancing ABC for stabilization verification of SystemVerilog/VHDL models," in *Proc. DIFTS'11*, ser. CEUR Work. Proc., vol. 832, 2011.

[31] P. Manolios, S. K. Srinivasan, and D. Vroon, "Automatic memory reductions for RTL model verification," in *Proc. ICCAD'06*. ACM, 2006, pp. 786–793.

[32] R. Mukherjee, P. Schrammel, D. Kroening, and T. Melham, "Unbounded safety verification for hardware using software analyzers," in *Proc. DATE'16*. IEEE, 2016, pp. 1152–1155.

[33] R. Mukherjee, M. Tautschnig, and D. Kroening, "v2c - A Verilog to C translator," in *Proc. TACAS'16*, ser. LNCS, vol. 9636. Springer, 2016, pp. 580–586.

[34] A. Niemetz, M. Preiner, C. Wolf, and A. Biere, "Btor2 , BtorMC and Boolector 3.0," in *Proc. CAV'18*, ser. LNCS, vol. 10981. Springer, 2018, pp. 587–595.

[35] M. Sagiv, "Harnessing SMT solvers for verifying low level programs," 2020, invited talk, *SMT'20*.

[36] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, 1997.

[37] T. Welp and A. Kuehlmann, "QF BV model checking with property directed reachability," in *Proc. DATE'13*, 2013, pp. 791–796.

[38] ——, "Property directed invariant refinement for program verification," in *Proc. DATE'14*. Europ. Design and Automation Ass., 2014, pp. 1–6.

[39] ——, "Property directed reachability for QF_BV with mixed type atomic reasoning units," in *Proc. ASP-DAC'14*. IEEE, 2014, pp. 738–743.

[40] C. Wolf, "Yosys," https://github.com/YosysHQ/yosys.

World-Level Modelling

- bit-precise reasoning: <u>bit-vector</u> as basic modelling element

- thus in essence SMT theory QF_BV of <u>bit-vectors</u> [SMTLIB]

  - sorts: bit $\mathbb{B} = \{0, 1\}$ bit-vector $\mathbb{B}[w] = \mathbb{B}^w$

  - constants: 65_{10} decimal 00100011_2 binary $\overbrace{111\cdots 111}^{35}$ (unary)

  - variables: declared as $b[1]$ and $x[32]$ `bool b, x[32];`

  - comparison: $=, \neq, <, \leq$ (signed and unsigned), …

  - bit-wise operators: $\sim, -, \wedge, \vee, \oplus, \dots$ shifting operators: shift, rotate …

  - arithmetic operators: $+, -, *, /, \dots$ string operators: slicing, append, extend, …

- plus <u>array</u> theory QF_ABV to model memory   main memory, caches, etc.

  - sorts: array $\mathbb{B}[r][2^d] = (\mathbb{B}^d \to \mathbb{B}^r) = \mathbb{B}^{r 2^d} = \mathbb{B}[r \cdot 2^d]$

  - constants: **?**   zero, range initializers, lambdas, quantifiers, …

  - variables: declared as $c[64][1024]$ 8KB cache $m[8][2^{64}]$ main memory
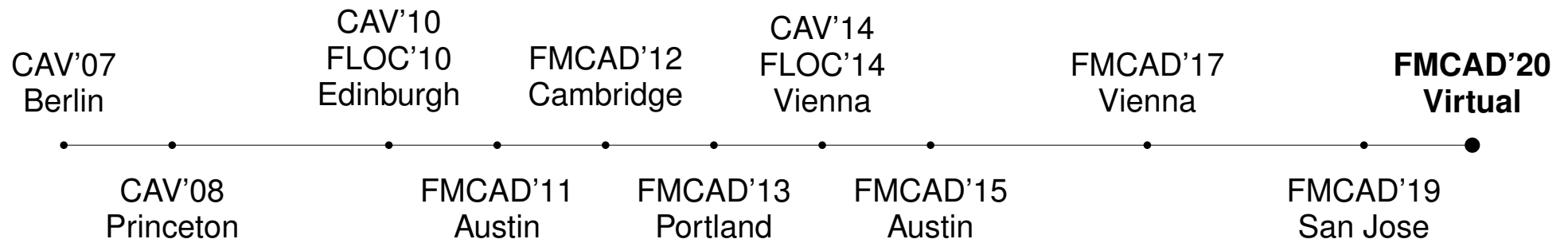
    ```
    (declare-fun c () (Array (_ BitVec 10) (_ BitVec 64)))
    (declare-fun m () (Array (_ BitVec 64) (_ BitVec 8)))
    ```

  - operators: read, write (update)   select, store

Sequential Modelling = State Machines / Kripke Structures / Automata

- use "logic" (e.g., bit-vector formulas) to describe sequential semantics symbolically

- Kripke structure flavor   think "SMV"
  - initialization and (total) transition <u>relation</u>
  - non-deterministic modelling thus inputs are part of the state
  - still usually variable based: state space = possible variable assignments
  - constraints (invariants / fairness) and properties (temporal logic)

- automata or circuit flavor   think "Verilog" or AIGER on the bit-level
  - initialization and transition <u>function</u>   partial initialization important in AIGER
  - separate variables for inputs and states
  - non-determinism modelled with inputs   "$\cdots = *;$" in SLAM,   oracle / Choueka construction
  - constraints, properties and explicit outputs   for simple compositional semantics
  - <mark>clear semantics</mark> close to actual HW / SW

- thus in summary we prefer the second "functional" view   as in AIGER and BTOR
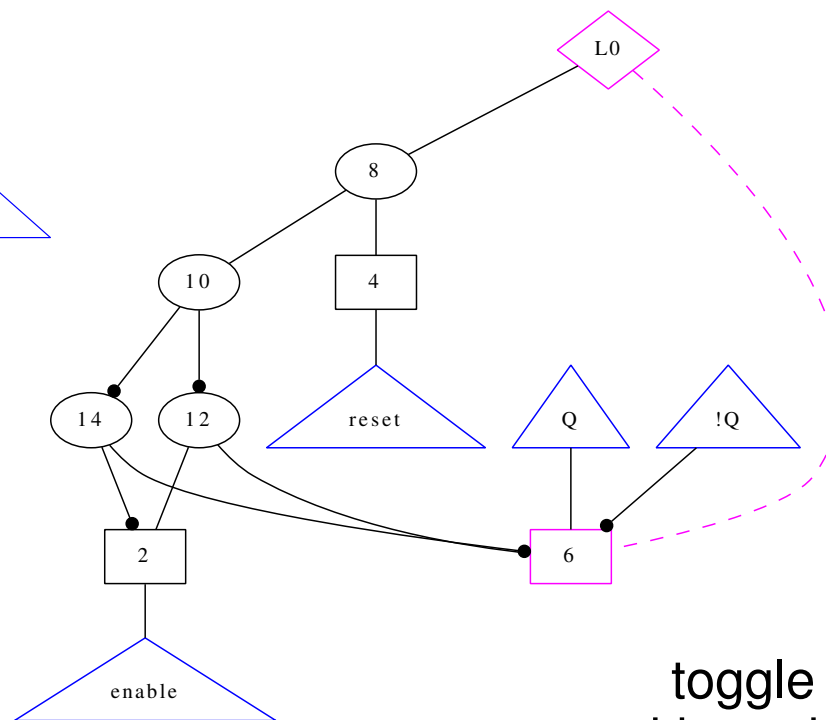 - also gives a faster and simpler to implement model checker   [JussilaBiere'07]
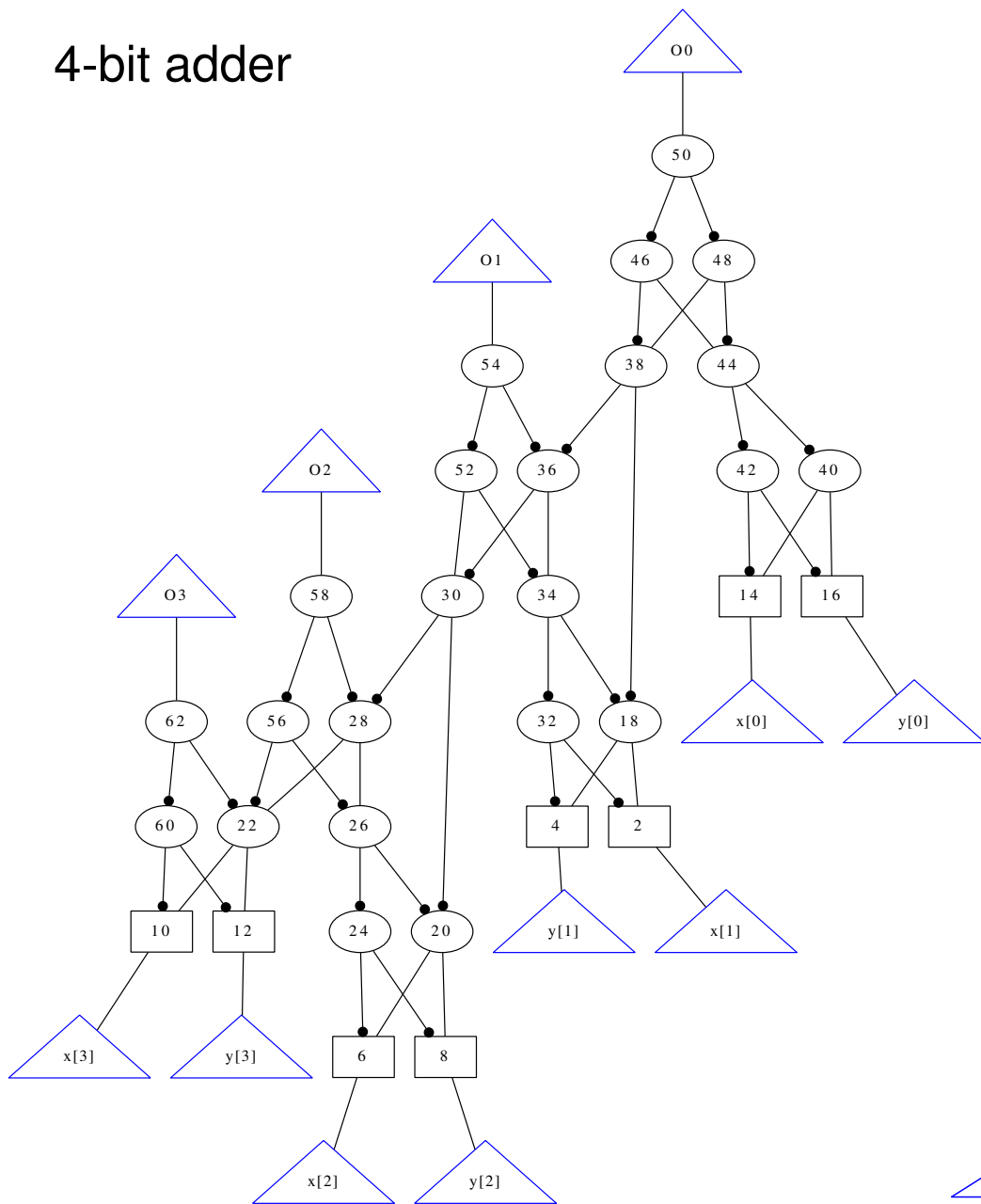
AIGER

- bit-level (propositional) functional model checking format

- bootstrapped first hardware model checking competition (HWMCC'07)

- witness / trace format, tool set for simulation / witness checking , splitting, unrolling …

- simple and clean semantics, common denominator of model checkers   [Biere'07]

- constraints, more general properties and synthesis support   [BiereHeljankoWieringa'11]

- now supported by many HW tools as (binary) exchange format (such as ABC)

- AIG means And-Inverter Graph (formulas with AND and NOT only)

- used since 2007 in the hardware model checking competition (HWMCC)
  [Cabodi et.al. : HWMCC'14]   [BiereVanDijkHeljanko'17]

- collected and selected benchmark sets used in many papers

CAV'07 Berlin		CAV'10 FLOC'10 Edinburgh		FMCAD'12 Cambridge		CAV'14 FLOC'14 Vienna		FMCAD'17 Vienna		**FMCAD'20 Virtual**
	CAV'08 Princeton		FMCAD'11 Austin		FMCAD'13 Portland		FMCAD'15 Austin		FMCAD'19 San Jose	

AIGER

4-bit adder

http://fmv.jku.at/aiger

toggle flip-flop
with enable & reset

- BTOR 1.0   [BrummayerBiereLonsing'09]

  - word-level generalization of the initial AIGER format from 2007 (ASCII version)

  - supports bit-vectors and arrays   (again quantifier-free formulas only)

  - sequential functional extensions as in AIGER

- BTOR 2.0   [NiemetzPreinerWolfBiere'18]

  - resumed word-level motivated by open flows (Yosys) and open cores (RISC-V)

  - incorporated new AIGER 1.9 features from 2011

  - witness format

  - new tools:

    - witness checker / simulator

    - bounded model checker

    - new bit-blaster on top of Boolector's bit-blaster   [Preiner'2019]

  - still lacking: fuzzer, delta debugger, bit-blasting of arrays

  - initialization of arrays still tricky

  - used in HWMCC'19 and HWMCC'20

BTOR Model Example

```
1 sort bitvec 1
2 sort bitvec 3
3 zero 2
4 state 2 cnt          } cnt = 0
5 init 2 4 3
6 input 2 in
7 add 2 4 6            } cnt' = cnt + in
8 next 2 4 7
9 ones 2
10 eq 1 4 9            } bad : (cnt == 7)
11 bad 10
12 constd 2 3
13 ulte 1 6 12         } in ≤ 3
14 constraint 13
```

$$cnt = 0$$

$$cnt' = cnt + in$$

$$bad : (cnt == 7)$$

$$in \leq 3$$

Witness Example

```
sat
b0
#0
@0
0 011 in@0
@1
0 010 in@1
@2
0 010 in@2
@3
0 000 in@3
.
```

BTOR2 Model Format

[NiemetzPreinerWolfBiere'18]

⟨num⟩	::=	positive unsigned integer (greater than zero)
⟨uint⟩	::=	unsigned integer (including zero)
⟨string⟩	::=	sequence of whitespace and printable characters without '\n'
⟨symbol⟩	::=	sequence of printable characters without '\n'
⟨comment⟩	::=	';' ⟨string⟩
⟨nid⟩	::=	⟨num⟩
⟨sid⟩	::=	⟨num⟩
⟨const⟩	::=	'const' ⟨sid⟩ [0-1]+
⟨constd⟩	::=	'constd' ⟨sid⟩ ['-']⟨uint⟩
⟨consth⟩	::=	'consth' ⟨sid⟩ [0-9a-fA-F]+
⟨input⟩	::=	('input' | 'one' | 'ones' | 'zero') ⟨sid⟩	⟨const⟩	⟨constd⟩	⟨consth⟩
⟨**state**⟩	::=	'**state**' ⟨sid⟩
⟨bitvec⟩	::=	'bitvec' ⟨num⟩
⟨array⟩	::=	'array' ⟨sid⟩ ⟨sid⟩
⟨node⟩	::=	⟨sid⟩ 'sort' (⟨array⟩ | ⟨bitvec⟩)
		| ⟨nid⟩ (⟨input⟩ | ⟨state⟩)
		| ⟨nid⟩ ⟨opidx⟩ ⟨sid⟩ ⟨nid⟩ ⟨uint⟩ [⟨uint⟩]
		| ⟨nid⟩ ⟨op⟩ ⟨sid⟩ ⟨nid⟩ [⟨nid⟩ [⟨nid⟩]]
		| ⟨nid⟩ ('**init**' | '**next**') ⟨sid⟩ ⟨nid⟩ ⟨nid⟩
		| ⟨nid⟩ ('**bad**' | '**constraint**' | '**fair**' | 'output') ⟨nid⟩
		| ⟨nid⟩ '**justice**' ⟨num⟩ (⟨nid⟩)+
⟨line⟩	::=	⟨comment⟩	⟨node⟩ [⟨symbol⟩] [⟨comment⟩]
⟨btor⟩	::=	(⟨line⟩'\n')+

`https://github.com/Boolector/btor2tools`

BTOR2 Witness Format

[NiemetzPreinerWolfBiere'18]

⟨binary-string⟩	::=	[0-1]+
⟨bv-assignment⟩	::=	⟨binary-string⟩
⟨array-assignment⟩	::=	'[' ⟨binary-string⟩ ']' ⟨binary-string⟩
⟨assignment⟩	::=	⟨uint⟩ (⟨bv-assignment⟩ | ⟨array-assignment⟩) [⟨symbol⟩]
⟨model⟩	::=	(⟨comment⟩'\n' | ⟨assignment⟩'\n')+
⟨state part⟩	::=	'#' ⟨uint⟩ '\n' ⟨model⟩
⟨input part⟩	::=	'@' ⟨uint⟩ '\n' ⟨model⟩
⟨frame⟩	::=	[**⟨state part⟩**] **⟨input part⟩**
⟨prop⟩	::=	('b' | 'j')⟨uint⟩
⟨header⟩	::=	'sat\n' (**⟨prop⟩**)+ '\n'
⟨witness⟩	::=	(⟨comment⟩'\n')+	**⟨header⟩** (**⟨frame⟩**)+ '.'

https://github.com/Boolector/btor2tools

Another Example Modelling a C program

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
static bool read_bool () {
  int ch = getc (stdin);
  if (ch == '0') return false;
  if (ch == '1') return true;
  exit (0);
}
int main () {
  bool turn;                    // input
  unsigned a = 0, b = 0; // states
  for (;;) {
    turn = read_bool ();
    assert (!(a == 3 && b == 3));
    if (turn) a = a + 1;
    else      b = b + 1;
  }
}
```

```
1 sort bitvec 1          sat
2 sort bitvec 32         b0
3 input 1 turn           #0
4 state 2 a              @0
5 state 2 b              0 1 turn@0
6 zero 2                 @1
7 init 2 4 6             0 0 turn@1
8 init 2 5 6             @2
9 one 2                  0 0 turn@2
10 add 2 4 9             @3
11 add 2 5 9             0 0 turn@3
12 ite 2 3 4 10          @4
13 ite 2 -3 5 11         0 1 turn@4
14 next 2 4 12           @5
15 next 2 5 13           0 1 turn@5
16 constd 2 3            @6
17 eq 1 4 16             0 0 turn@6
18 eq 1 5 16
19 and 1 17 18
20 bad 19
```

Application Specific Sequential Word-Level Formats

- Hardware description languages (HDL):　　(System)-Verilog, System-C, VHDL, …
  - "what you check is what you get"
  - usually have (very) complex semantics and undefined behaviour
  - Yosys, Reveal, Enhanced ABC, commercial model checkers

- Software languages:　　C, Java, JVM, GraalVM, LLVM, assembler, …
  - "what you check is what you get"
  - usually have complex semantics and undefined behaviour
  - "Competition on Software Verification" SV-Comp

- application specific languages problematic
  - hard to reuse solver / checker technology
  - QF_BV is pretty successful in both HW and SW applications
  - encode "undefinedness" precisely is better
  - same should apply to model checking
  - but:　　"v2c – A Verilog to C translator "

[MukherjeeTautschnigKroening'16] [MukherjeeSchrammelKroeningMelham'16]

Other Generic Word-Level Model Checking Formats

- UCLID   [BryantLahiriSeshia]
  - early SMT solving (UF, lambdas, memory) targeting processor verification
  - bounded model checking in essence (manual inductive verification)
- SAL from SRI   [DeMouraOwreShankar'03]   Yices [Duherte'14]
  - focus was orignally on infinite systems
  - sofar not-much interest in bit-precise reasoning
- constrained horn clauses μZ [HoderBjornerDeMoura'11]
  - basically extends an SMT solver (Z3) with (second order) least fix-points
  - active community: workshops, competition, …
  - sofar not-much interest in bit-precise reasoning
- VMT   nuXmv [CAV'14]   Verilog2SMV [DATE'16]   from FBK IRST in Trento
  - SMTLIB with annotations to mark initialization and transition predicates
  - built around (nu)SMV using MathSAT as word-level engine
  - actively supports bit-vectors
- related "Model Checking Competition" (MCC) has Petri nets models (in PNML)
- "classical" protocol modelling languages: Promela (SPIN), Murphi, …

Bit-Blasting Explodes

- show *commutativity of bit-vector addition* for bit-width 1 million:

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 1000000))
(declare-fun y () (_ BitVec 1000000))
(assert (distinct (bvadd x y) (bvadd y x)))
```

- size of SMT2 file: 138 bytes

- bit-blasting with our SMT solver Boolector
  - rewriting turned off
  - except structural hashing
  - produces AIGER circuits of file size   103 MB

- Tseitin transformation leads to CNF in DIMACS format of size   1 GB

Complexity Classification Results for Bit-Vector Logics

our results from [KovásznaiFröhlichBiere-SMT'12] paper extended version in our TOCS'16 article

		quantifiers			
		no		yes	
		uninterpreted functions		uninterpreted functions	
		no	yes	no	yes
encoding	unary	NP QF_BV1 obvious	NP QF_UFBV1 Ackermann	PSPACE BV1 [TACAS'10]	NEXPTIME UFBV1 [FMCAD'10]
	binary	**NEXPTIME** **QF_BV2** **[SMT'12]**	NEXPTIME QF_UFBV2 [SMT'12]	AEXP(poly) BV2 [JonášStrejček-IPL'18]	2NEXPTIME UFBV2 [SMT'12]

QF = "quantifier free"   UF = "uninterpreted functions"   BV = "bit-vector logic"

BV1 = "unary encoded bit-vectors"   BV2 = "binary encoded bit-vectors"

Complexity Classification Results for Arrays and Word-Level Model Checking

- AIGER problems are PSPACE complete

  - since "symbolicl reachability" is PSPACE complete   [Savitch'70]

- now assume (for instance) sequential BTOR 2.0 as input

  - <u>without arrays</u> but sequential problems (model checking)

    - unary encoding (or bit-width as fixed parameter):   PSPACE complete

    - binary encoding: EXSPACE complete   [KovasznaiVeithFröhlichBiere'MFCS14]

  - <u>with arrays</u> and sequential problems (model checking)

    - unary encoding: **?**   EXPSPACE complete?

    - binary encoding: **?**   2EXPSPACE complete?

- benefits of complexity characterizations

  - gives hints what solvers (SAT,SMT, AIGER) can be used as oracles

  - and how many times they have to be called

  - sometimes gives restricted classes   PSPACE sub-class of QF_BV2

Why do we want to do word-level model checking?

- use word-level "structure" for rewriting / simplification

  - allows (shallow) arithmetic reasoning   as in the complexity example

  - word-level local search   [NiemetzPreinerBiere'16/17] [NiemetzPreiner'20]

  - make full use of functional representation

    - global substitution pass instead of congruence closure

    - CNF preprocessing lacks some benefits of circuit representations

    - bit-level circuit intermediate formats (thus bit-level rewriting)

    - BDD / SAT / SMT / cut sweeping to eliminate equivalent expressions

- data and memory abstraction

  - bit-blasting of arithmetic expensive $*_{32}$ has 8000 AIG nodes, $*_{64}$ has 32 000

  - protocols only "move data around":   bit-precise reasoning redundant

  - properties often argue about some "reads" and "writes" only

  - bit-blasting memory is often impossible $m_{32}[8][2^{32}]$ $m_{64}[8][2^{64}]$

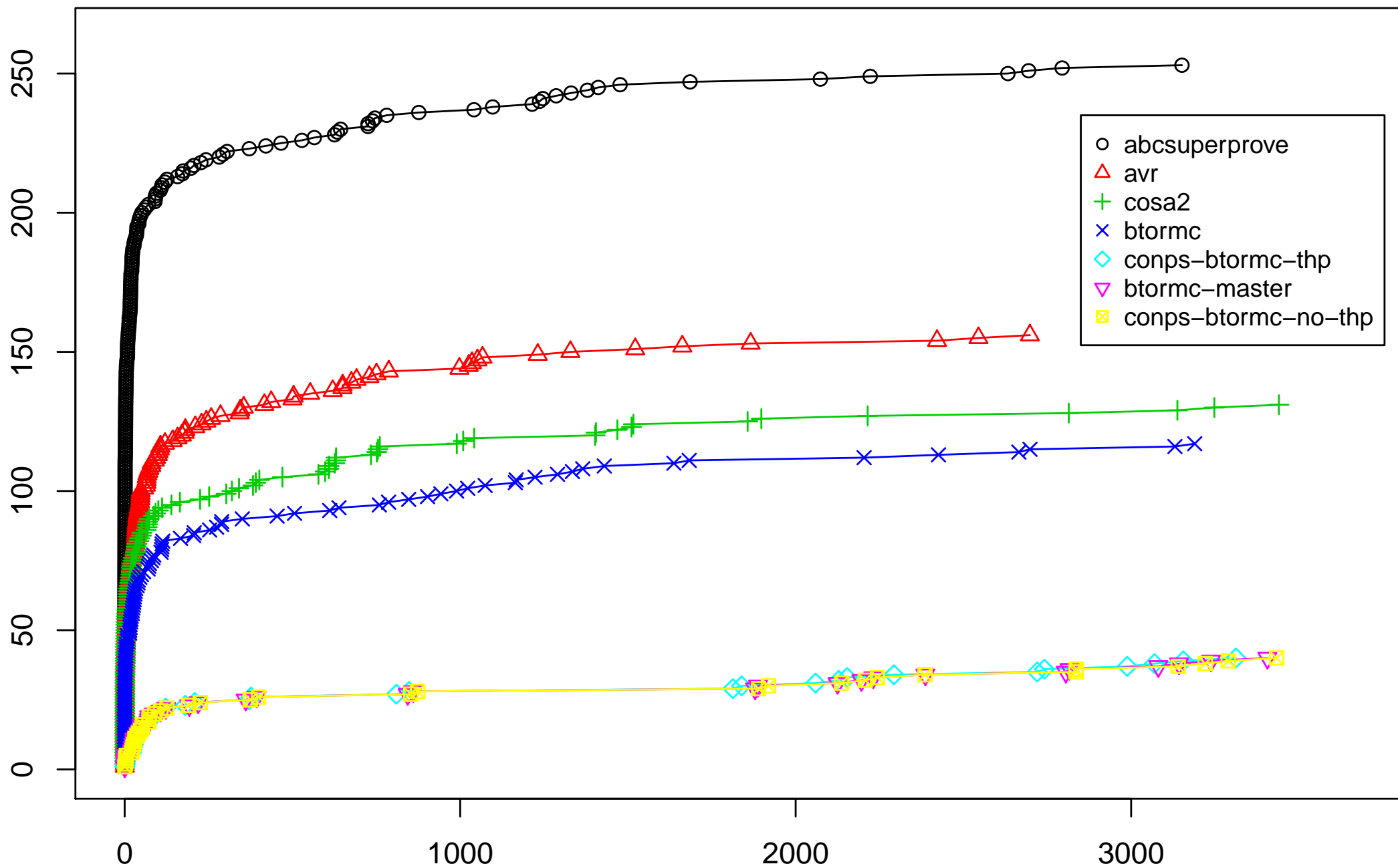- sequential and non-sequential rewriting and abstraction techniques

Eager Data Abstraction

- 1-bit abstractions

  - verify sorting using only "compare & swap" on 0/1 input   zero-one principle [Knuth'73]

  - data independence of protocols [Wolper'86]

- small domain encoding   part of Ackermann's reduction

  - if you only compare n variables then interpret them on the domain $0, \ldots, n-1$

  - reduce those variables to bit-width $\lceil \log n \rceil$

  - eager translation to SAT possible   [PnueliRodehShtrichmanSiegel'99]

  - plain bit-vectors [Johannsen'01/02], model checking [HojatiBrayton'95] [Bjesse'08]

  - need to "slice" bit-vectors in HW to have compatible widths   next state functions too

  - can use different domain size for each "cluster" of compared variables

- abstract uninterpreted functions (UF) through Ackermann eagerly   transformation

  - extends to memories / arrays   (exponentially) eliminate read & write as in UCLID

  - works for plain bit-vectors (thus BMC) but then lazy SMT (QF_AUFBV) is better
    [BurchDill'96] [VelevBryantJain'97] [ManoliosSrinivasanVroon'06] [GanaiGuptaAshar'04/05]

  - model checking requires to change properties [Bjesse'08/09] [German'11]

Lazy Data Abstraction

- akin to "lazy SMT" or CEGAR / Localization

- for instance replace expensive operations (multiplication) with UF
  - abstraction refinement loop using SMT   [AndrausLiffitonSakkalah'06/08]
  - conservative: if abstracted model passes property then original passes it too
  - spurious counter example: refine   "$mult(x,y)$" to "$(x = 0\,?\,0\,:\,mult(x,y))$"
  - refinement can make use of cores or MUS

- combine with IC3 / PDR   [LeeSakallah'14] [GoelSakallah'19/20]
  - predicate abstraction   existing predicates, new predicates?
  - syntax guided abstraction   equality between existing expressions, new expressions?

- how to interpolation into the mix is still unclear
  bit-vectors [Griggio'16] [BackemanRümmerZeljic'18] [OkudonoKing'20]   arrays **?**

- also still needs to be combined with successful bit-level techniques
  - sweeping / temporal decomposision / retiming
  - local search / simulation

HWMCC'19 Results on Bit-Vectors (BV)   without arrays

Legend:
- ○ abcsuperprove
- △ avr
- + cosa2
- × btormc
- ◇ conps-btormc-thp
- ▽ btormc-master
- ⊠ conps-btormc-no-thp

Challenges

- benchmarks: Yosys, open cores, RISC-V already helped a lot, **but need more!**

- apply HW word-level model checkers to SW (from SV-COMP) or vice versa

- symbolic execution of both SW and HW
  - modelling (slices of) programs linearly in a word-level model
  - "Selfie" by Christoph Kirsch has a BTOR2 model of RISC-U

- smart contracts
  - bit-precise semantics lends itself to word-level models
  - as discussed in invited SMT'20 talk by Mooly Sagiv

- certificates:
  - UNSAT proofs in SAT very useful   "biggest math proof ever" by Marijn Heule
  - certificates for (passing properties) in AIGER   with Zhengqi Yu and Keijo Heljanko
  - certificates for UNSAT proofs in QF_BV   [CVC4 team]
  - combine to provide word-level certificates

- make word-level model checkers faster than bit-level checkers $\Rightarrow$ HWMCC'20?