BTOR2, BtorMC and Boolector 3.0

Aina Niemetz[1,3], Mathias Preiner[1,3], Clifford Wolf[2] and Armin Biere[3]

1 Stanford University

2 Symbiotic EDA

3 JOHANNES KEPLER UNIVERSITÄT LINZ

CAV'18

30th International Conference on Computer Aided Verification

7th Federated Logic Conference (FLoC'18)

FWF Der Wissenschaftsfonds.

Oxford, UK

July 15, 2018

RiSE

BTOR2 Example

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

```
sat
b0
#0
@0
0 1 turn@0
@1
0 0 turn@1
@2
0 0 turn@2
@3
0 0 turn@3
@4
0 1 turn@4
@5
0 1 turn@5
@6
0 0 turn@6
```

Syntax BTOR2 Model (Part 1)

⟨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]+

Syntax BTOR2 Model (Part 2)

⟨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')+

sequential part in red and underlined

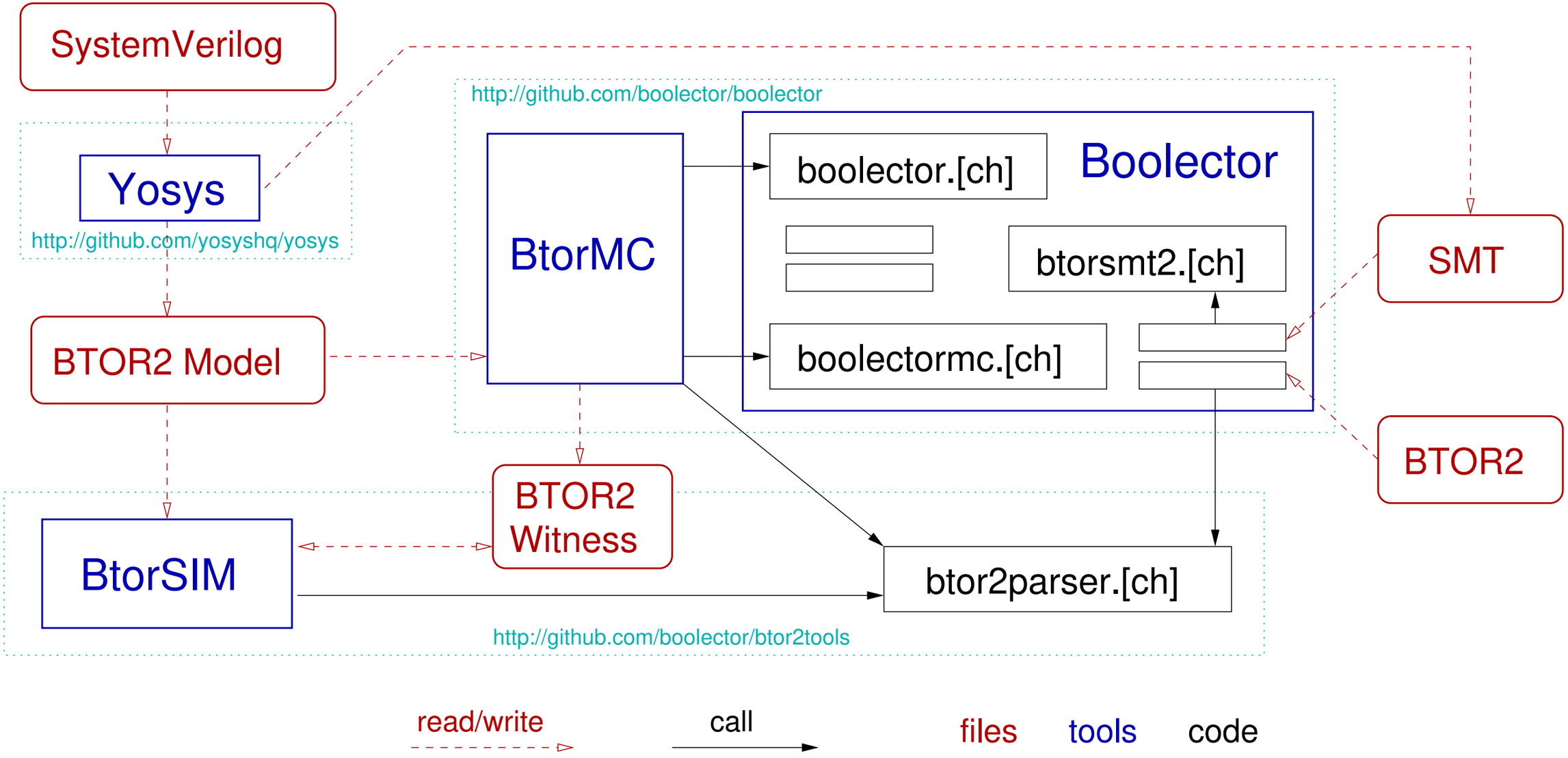
Syntax BTOR2 Witness

⟨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⟩)+ '.'

Operators

indexed		
[su]ext w	(un)signed extension	$\mathcal{B}^n \rightarrow \mathcal{B}^{n+w}$
slice u l	extraction, $n > u \geq l$	$\mathcal{B}^n \rightarrow \mathcal{B}^{u-l+1}$
unary		
not	bit-wise	$\mathcal{B}^n \rightarrow \mathcal{B}^n$
inc, dec, neg	arithmetic	$\mathcal{B}^n \rightarrow \mathcal{B}^n$
redand, redor, redxor	reduction	$\mathcal{B}^n \rightarrow \mathcal{B}^1$
binary		
iff, implies	Boolean	$\mathcal{B}^1 \times \mathcal{B}^1 \rightarrow \mathcal{B}^1$
eq, neq	(dis)equality	$\mathcal{S} \times \mathcal{S} \rightarrow \mathcal{B}^1$
[su]gt, [su]gte, [su]lt, [su]lte	(un)signed inequality	$\mathcal{B}^n \times \mathcal{B}^n \rightarrow \mathcal{B}^1$
and, nand, nor, or, xnor, xor	bit-wise	$\mathcal{B}^n \times \mathcal{B}^n \rightarrow \mathcal{B}^n$
rol, ror, sll, sra, srl	rotate, shift	$\mathcal{B}^n \times \mathcal{B}^n \rightarrow \mathcal{B}^n$
add, mul, [su]div, smod, [su]rem, sub	arithmetic	$\mathcal{B}^n \times \mathcal{B}^n \rightarrow \mathcal{B}^n$
[su]addo, [su]divo, [su]mulo, [su]subo	overflow	$\mathcal{B}^n \times \mathcal{B}^n \rightarrow \mathcal{B}^1$
concat	concatenation	$\mathcal{B}^n \times \mathcal{B}^m \rightarrow \mathcal{B}^{n+m}$
read	array read	$\mathcal{A}^{I \rightarrow \mathcal{E}} \times I \rightarrow \mathcal{E}$
ternary		
ite	conditional	$\mathcal{B}^1 \times \mathcal{B}^n \times \mathcal{B}^n \rightarrow \mathcal{B}^n$
write	array write	$\mathcal{A}^{I \rightarrow \mathcal{E}} \times I \times \mathcal{E} \rightarrow \mathcal{A}^{I \rightarrow \mathcal{E}}$

Flow


SystemVerilog
Yosys
http://github.com/yosyshq/yosys
BTOR2 Model
BtorSIM
http://github.com/boolector/boolector
BtorMC
boolector.[ch]
Boolector
btorsmt2.[ch]
boolectormc.[ch]
SMT
BTOR2
BTOR2
Witness
btor2parser.[ch]
http://github.com/boolector/btor2tools
read/write
call
files
tools
code

Boolector 3.0

Previous

Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0. JSAT'15

Quantifiers

Preiner, M., Niemetz, A., Biere, A.: Counterexample-guided model synthesis. TACAS'17

Local Search

Niemetz, A., Preiner, M., Biere, A., Fröhlich, A.:
Improving local search for bit-vector logics in SMT with path propagation. DIFTS'15

Niemetz, A., Preiner, M., Biere, A.:
Precise and complete propagation based local search for satisfiability modulo theories. CAV'16,

Niemetz, A., Preiner, M., Biere, A.:
Propagation based local search for bit-precise reasoning. FMSD'17

Availability Open source (MIT) http://github.com/boolector

Experiments on Models Synthesized with Yosys

10 real-world System Verilog designs with safety properties from open source projects
(RISC-V Formal, VexRiscv, PicoRV32, PonyLink, ZipCPU)

Benchmark	k	#bad	**BtorMC** Time[s]	**Boolector** Time[s]	**Yices** Time[s]
picorv32-check	30	23	**4.8**	18.9	10.8
picorv32-pcregs	20	3	**63.0**	293.0	TO
ponylink-slaveTXlen-sat	230	1	305.5	406.8	**145.6**
ponylink-slaveTXlen-unsat	231	1	183.8	131.4	**71.4**
VexRiscv-regch0-15	17	2	**9.6**	48.3	12.2
VexRiscv-regch0-20	22	2	528.8	**520.7**	2232.2
VexRiscv-regch0-30	32	2	TO	TO	TO
zipcpu-busdelay	100	50	**157.0**	287.0	181.2
zipcpu-pfcache	100	39	**17.4**	19.9	32.5
zipcpu-zipmmu	30	57	86.0	412.9	**46.5**

BtorMC/BTOR2 vs. unrolled SMT-LIB with a time limit of 3600 seconds
k is the bound and #bad is the number of bad properties

Conclusion

- new word-level model checking and witness format
  - following AIGER format — used in HWMCC for 12 years
  - bit-vector and array sorts — can easily be extended
  - semantics as in SMT-LIB (QF_ABV) + sequential extension
  - properties: safety and invariant constraints, but also liveness and fairness
- model checker BtorMC (with API) based on new Boolector 3.0 — `https://github.com/boolector`
  - experiments on benchmarks synthesized with Yosys — `http://fmv.jku.at/cav18-btor2`
  - simple generic stand-alone parser
  - random simulator and witness checker
  - all open source (MIT license) — incl. Boolector and Lingeling
- word-level track in future HWMCC

We need your support!

Hiring @ JKU: PostDocs (LIT AI Lab), PhDs (LogiCS Doctoral College)