Improving the Encoding of Arrays in Btor2-to-C Translation

Bachelor-Thesis Defense Talk by Salih Ates
Date: August 30th, 2023
Intro Talk Recap
## Motivation

<table>
<thead>
<tr>
<th>Tool Algorithm Input</th>
<th>Tasks</th>
<th>ABC PDR AIGER</th>
<th>AVR PDR BTOR2</th>
<th>CPAchecker Pred. Abs.</th>
<th>ESBMC k-Induction C-e C-l</th>
<th>VERIVABS Loop Abs. C-e C-l</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct results</td>
<td>1498</td>
<td>862 736</td>
<td>274 280 401</td>
<td>410 392 393</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BV proofs</td>
<td>868</td>
<td>524 458</td>
<td>188 189 88</td>
<td>93 53 49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BV alarms</td>
<td>473</td>
<td>338 233</td>
<td>86 91 311</td>
<td>315 337 342</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array proofs</td>
<td>140</td>
<td>– 45</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>Array alarms</td>
<td>17</td>
<td>– 0</td>
<td>0 0 2</td>
<td>2 2 2</td>
<td>2 2</td>
<td></td>
</tr>
<tr>
<td>Wrong proofs</td>
<td>0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>2 2</td>
<td></td>
</tr>
<tr>
<td>Wrong alarms</td>
<td>0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>Timeouts</td>
<td>479</td>
<td>559</td>
<td>924 922 554</td>
<td>551 1049 1042</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Out of memory</td>
<td>0 3</td>
<td>9 7 543 537</td>
<td>3 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other inconclusive</td>
<td>0 200</td>
<td>291 289 51 56</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Bridging Hardware and Software Analysis with Btor2C: A Word-Level-Circuit-to-C Translator, https://doi.org/10.1007/978-3-031-30820-8_12 (Table 1)
(1) Array blasting

- YOSYS
- Verilog
- AIGER
- BTOR2AIGER
- BTOR2
- BTOR2C
- C
- Bit-Level Analyzer
- Word-Level Analyzer
- Software Analyzer
(1) Array blasting

- YOSYS
  - Verilog
  - BTOR2 (bv and arr)
  - BTORBLASTER
  - BTOR2C
  - C
  - My Contribution

- AIGER
- BTOR2AIGER
- BTOR2 (bv only)
- Bit-Level Analyzer
- Word-Level Analyzer
- Software Analyzer
(2) ALAP-Scheduling of write-ops
(as-late-as-possible)

Objective:
Minimize number of duplications for write operations.

1 sort bitvec 3
2 sort array 1 1
3 state 2
4 input 1
5 constd 1 0
6 constd 1 1
7 constd 1 2
8 write 2 3 4 5
9 write 2 8 6 7
10 write 2 9 4 6
11 read 1 10 4
12 sort bitvec 1
13 eq 12 11 5
14 bad 13

SORT_2 write_8;
for(i = 0; i < len(old_arr); ++i){
    write_8[i] = old_arr[i];
}
write_8[arg_1] = value;
(2) ALAP-Scheduling of write-ops
(as-late-as-possible)

Objective:
Minimize number of duplications for write operations.

1 sort bitvec 3
2 sort array 1 1
3 state 2
4 input 1
5 constd 1 0
6 constd 1 1
7 constd 1 2
8 write 2 3 4 5
9 write 2 8 6 7
10 write 2 9 4 6
11 read 1 10 4
12 sort bitvec 1
13 eq 12 11 5
14 bad 13

SORT_2* write_8 = old_arr;
write_8[arg_1] = value;
(2) ALAP-Scheduling of write-ops

\[
\text{array} \rightarrow \text{read} \rightarrow \text{write} \rightarrow \text{next} \rightarrow \text{bad}
\]

\[
\text{schedule} = [\text{array, write, read}]
\rightarrow \text{Duplication needed}
\]

\[
\text{schedule} = [\text{array, read, write}]
\rightarrow \text{Duplication can be avoided}
\]
1 Array blasting
1 sort bitvec 1
2 sort bitvec 2
3 sort bitvec 3
4 sort array 3 2
5 state 4
6 input 3
7 constd 2 1
8 write 4 5 6 7
9 read 2 5 6
10 eq 1 5 8
11 neq 1 5 8
12 ite 3 10 5 8
13 init 3 5 8
14 next 3 5 8

All array operations:
- state
- write
- read
- eq
- neq
- ite
- init
- next
(1) Array blasting – state

2 sort bitvec 2
3 sort bitvec 3
4 sort array 3 2
5 state 4

[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

2 sort bitvec 2
3 sort bitvec 3
4 sort array 3 2
5 state 2
6 state 2
7 state 2
8 state 2
9 state 2
10 state 2
11 state 2
12 state 2


(1) **Array blasting – write**

2 sort bitvec 2
3 sort bitvec 3
4 sort array 3 2
5 state 4
6 input 3
7 constd 2 1
8 write 4 5 6 7

\[
\text{id write sort}_n \ \text{old_array} \ \text{index} \ \text{value}
\]

\[
\text{write_array}[0] = \begin{cases} 
\text{Y} & \text{index} == 0 \\
\text{N} & \text{value} \ \text{old_array}[0]
\end{cases}
\]

id ite sort\_n (index == 0) value old\_array[0]
(1) Array blasting – write

2 sort bitvec 2
3 sort bitvec 3
4 sort array 3 2
5 state 4
6 input 3
7 constd 2 1
8 write 4 5 6 7

id write sort_n old_array index value

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ite</td>
<td>input == 0</td>
<td>state[0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ite</td>
<td>input == 1</td>
<td>state[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ite</td>
<td>input == 2</td>
<td>state[2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ite</td>
<td>input == 3</td>
<td>state[3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ite</td>
<td>input == 4</td>
<td>state[4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ite</td>
<td>input == 5</td>
<td>state[5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ite</td>
<td>input == 6</td>
<td>state[6]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ite</td>
<td>input == 7</td>
<td>state[7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(1) **Array blasting - write**

2 sort bitvec 2
3 sort bitvec 3
4 sort array 3 2
5 state 4
6 constd 2 2
7 constd 2 1
8 write 4 5 6 7

id write sort_n old_array constd value

state[0]
state[1]
7
state[3]
state[4]
state[5]
state[6]
state[7]
(1) Array blasting – read (skewed)

id read sort_n array index

index == 0

Y

N

array[0]

index == 1

Y

N

array[1]

index == 2

Y

N

(1) Array blasting – read (skewed)

id read sort_n array index
2 input 1
3 state 1
4 state 1
5 state 1
6 state 1
7 ite 1 (index == 2) 5 6
8 ite 1 (index == 1) 4 7
9 ite 1 (index == 0) 3 8
(1) Array blasting – read (balanced)

id read sort_n array index

- index < 2
  - Y: array[0]
  - N: array[1]
- index < 1
  - Y: array[2]
  - N: array[3]
(1) Array blasting – eq/neq

id eq sort_n array_1 array_2
id neq sort_n array_1 array_2

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>state 2</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>state 2</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>state 2</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>state 2</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>state 2</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>state 2</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>state 2</td>
<td>21</td>
</tr>
<tr>
<td>12</td>
<td>state 2</td>
<td>22</td>
</tr>
</tbody>
</table>
### (1) Array blasting – ite

<table>
<thead>
<tr>
<th>id ite sort_n cond</th>
<th>array_1</th>
<th>array_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>35 ite 2 cond</td>
<td>5 state 2</td>
<td>15 state 2</td>
</tr>
<tr>
<td>36 ite 2 cond</td>
<td>6 state 2</td>
<td>16 state 2</td>
</tr>
<tr>
<td>37 ite 2 cond</td>
<td>7 state 2</td>
<td>17 state 2</td>
</tr>
<tr>
<td>38 ite 2 cond</td>
<td>8 state 2</td>
<td>18 state 2</td>
</tr>
<tr>
<td>39 ite 2 cond</td>
<td>9 state 2</td>
<td>19 state 2</td>
</tr>
<tr>
<td>40 ite 2 cond</td>
<td>10 state 2</td>
<td>20 state 2</td>
</tr>
<tr>
<td>41 ite 2 cond</td>
<td>11 state 2</td>
<td>21 state 2</td>
</tr>
<tr>
<td>42 ite 2 cond</td>
<td>12 state 2</td>
<td>22 state 2</td>
</tr>
</tbody>
</table>
## (1) Array blasting – init

<table>
<thead>
<tr>
<th>id</th>
<th>init</th>
<th>sort</th>
<th>array_1</th>
<th>array_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>init 2</td>
<td>state 2</td>
<td>15 state 2</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>init 2</td>
<td>6 state 2</td>
<td>16 state 2</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>init 2</td>
<td>7 state 2</td>
<td>17 state 2</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>init 2</td>
<td>8 state 2</td>
<td>18 state 2</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>init 2</td>
<td>9 state 2</td>
<td>19 state 2</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>init 2</td>
<td>10 state 2</td>
<td>20 state 2</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>init 2</td>
<td>11 state 2</td>
<td>21 state 2</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>init 2</td>
<td>12 state 2</td>
<td>22 state 2</td>
<td></td>
</tr>
</tbody>
</table>
### Array blasting – next

<table>
<thead>
<tr>
<th>id</th>
<th>next</th>
<th>sort_n</th>
<th>array_1</th>
<th>array_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>next2</td>
<td>5</td>
<td>state2</td>
<td>15</td>
</tr>
<tr>
<td>36</td>
<td>next2</td>
<td>6</td>
<td>state2</td>
<td>16</td>
</tr>
<tr>
<td>37</td>
<td>next2</td>
<td>7</td>
<td>state2</td>
<td>17</td>
</tr>
<tr>
<td>38</td>
<td>next2</td>
<td>8</td>
<td>state2</td>
<td>18</td>
</tr>
<tr>
<td>39</td>
<td>next2</td>
<td>9</td>
<td>state2</td>
<td>19</td>
</tr>
<tr>
<td>40</td>
<td>next2</td>
<td>10</td>
<td>state2</td>
<td>20</td>
</tr>
<tr>
<td>41</td>
<td>next2</td>
<td>11</td>
<td>state2</td>
<td>21</td>
</tr>
<tr>
<td>42</td>
<td>next2</td>
<td>12</td>
<td>state2</td>
<td>22</td>
</tr>
</tbody>
</table>
(1) Array blasting

All array operations:
- state
- write
- read
- eq
- neq
- ite
- init
- next
2

ALAP-Scheduling of write-ops
Objective:
Minimize number of duplications for write operations.

1 sort bitvec 3
2 sort array 1 1
3 state 2
4 input 1
5 constd 1 0
6 constd 1 1
7 constd 1 2
8 write 2 3 4 5
9 write 2 8 6 7
10 write 2 9 4 6
11 read 1 10 4
12 sort bitvec 1
13 eq 12 11 5
14 bad 13

SORT_2 write_8;
for(i = 0; i < len(old_arr); ++i){
    write_8[i] = old_arr[i];
}
write_8[arg_1] = value;
(2) ALAP-Scheduling of write-ops
(as-late-as-possible)

Objective:
Minimize number of duplications for write operations.

1 sort bitvec 3
2 sort array 1 1
3 state 2
4 input 1
5 constd 1 0
6 constd 1 1
7 constd 1 2
8 write 2 3 4 5
9 write 2 8 6 7
10 write 2 9 4 6
11 read 1 10 4
12 sort bitvec 1
13 eq 12 11 5
14 bad 13

SORT_2* write_8 = old_arr;
write_8[arg_1] = value;
(2) ALAP-Scheduling of write-ops

\[ \text{array} \xrightarrow{\text{write}} \xrightarrow{\text{read}} \xrightarrow{\text{write}} \]

\[ \text{next} \quad \text{bad} \]

The correct sequence is: \( \text{read} \rightarrow \text{write} \rightarrow \).
(2) ALAP-Scheduling of write-ops

```python
for(;;) {
```

```python
}
```
(2) ALAP-Scheduling of write-ops

for(;;) {

}
(2) ALAP-Scheduling of write-ops

for(;;) {
    var_5 = ...
    ...
    var_16 = ...
    assume(constr1);
    ...
    assume(constr2);
    ...
    assert(bad);
    ...
    state_5 = next_arg_1;
}
for(;;) {
  assume(constr1);
  ...
  assume(constr2);
  verifier(bad);
}
(2) ALAP-Scheduling of write-ops

for(;;) {
    assume(constr1);
    ...
    assume (constr2);
    ...
    assert(bad);
}
(2) ALAP-Scheduling of write-ops

![Diagram of ALAP scheduling]

```
SCHEDULE = [5, 4, 3, 2, 1]
```
(2) ALAP-Scheduling of write-ops

SCHEDULE = [1]
(2) ALAP-Scheduling of write-ops

SCHEDULE = [1 2]
(2) ALAP-Scheduling of write-ops

\[ \text{SCHEDULE} = [1, 2, 7] \]
(2) ALAP-Scheduling of write-ops

SCHEDULE = [1 2 7 8 9]
(2) ALAP-Scheduling of write-ops

READY_WRITE

READY

SCHEDULE = [1 2 7 8 9 3 4 5]

37
(2) ALAP-Scheduling of write-ops

\[
\text{SCHEDULE} = [1 \ 2 \ 7 \ 8 \ 9 \ 3 \ 4 \ 5 \ W1]
\]
(2) ALAP-Scheduling of write-ops

SCHEDULE = [1, 2, 7, 8, 9, 3, 4, 5, W1, 6]
(2) ALAP-Scheduling of write-ops

SCHEDULE = [1 2 7 8 9 3 4 5 W1 6 A]

READY_WRITE

READY

constr1 constr2 bad next
(2) ALAP-Scheduling of write-ops

SCHEDULE = [1 2 7 8 9 3 4 5 W1 6 A 10 W2]
(2) ALAP-Scheduling of write-ops

```
for(;;) {
    ..
    W1
    A
    W2
    assume(constr1);
    assume(constr2);
    assert(bad);
    state_x = next_arg
}
```

SCHEDULE = [1 2 7 8 9 3 4 5 W1 6 A 10 W2]
(2) ALAP-Scheduling of write-ops

for(;;) {
  ...
  W1
  A
  W2
  assume(constr1);
  assume(constr2);
  assert(bad);
  state_x = next_arg
}

1. Create intermediate nodes
2. Check constraints
3. Check properties
4. Assign next values

SCHEDULE = [1, 2, 7, 8, 9, 3, 4, 5, W1, 6, A, 10, W2]
(3) Experiments
3

Experiments
(3) Experiments

Setup:
- Ubuntu 22.04 (64 bit)
- Intel Xeon E3-1230 v5 (8 cores)
- 33 GB RAM

Limits for each task:
- 2 CPU cores
- 15 min of CPU time
- 15 GB RAM

Verifiers:
- ABC
- AVR
- CPAchecker
- ESBMC
- CBMC

Tasks:
- Collection of Btor2 tasks used in Btor2C paper
- 318 tasks for AVR
- 175 tasks for Software Verifiers
- 51 tasks for ABC
- 276 safe, 24 unsafe, 18 unknown
- Max array index bit-vector width: 12
(3) Experiments

<table>
<thead>
<tr>
<th>Tool Algorithm</th>
<th>ABC PDR</th>
<th>AVR PDR</th>
</tr>
</thead>
<tbody>
<tr>
<td># Tasks</td>
<td>51 (T: 27 F: 24)</td>
<td>318 (T: 294 F: 24)</td>
</tr>
<tr>
<td>Input Configuration</td>
<td>AIGER balanced skewed</td>
<td>/ balanced skewed</td>
</tr>
<tr>
<td>Proofs</td>
<td>11 11</td>
<td>129 137 136</td>
</tr>
<tr>
<td>Alarms</td>
<td>5 5</td>
<td>0 2 2</td>
</tr>
<tr>
<td>Wrong proofs</td>
<td>0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Wrong alarms</td>
<td>0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Timeouts</td>
<td>35 35</td>
<td>84 96 98</td>
</tr>
<tr>
<td>Out of memory</td>
<td>0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Other inconclusive</td>
<td>0 0</td>
<td>105 83 82</td>
</tr>
</tbody>
</table>

Total correct: 138

Total correct: 143
# Experiments

<table>
<thead>
<tr>
<th>Tool Algorithm</th>
<th>CPACHECKER</th>
<th>ESBMC</th>
<th>CBMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Configuration</td>
<td></td>
<td>balanced</td>
<td>skewed</td>
</tr>
<tr>
<td>Proofs</td>
<td>0</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wrong proofs</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wrong alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Timeouts</td>
<td>173</td>
<td>167</td>
<td>165</td>
</tr>
<tr>
<td>Out of memory</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Other inconclusive</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Total correct: 7
(3) Experiments

1/175 Tasks avoid copies

SCHEDULE = [arr, write, ite]
(4) Experiments

SORT_2 write_arr;
for(i = 0; i < len(old_arr); ++i){
    write_arr[i] = old_arr[i];
}
write_arr[arg_1] = value;

SORT_2* write_arr = old_arr;
write_arr[arg_1] = (cond) ? old_arr[arg_1] : value;
(3) Experiments

1 sort bitvec 1
2 sort bitvec (3-14)
3 sort bitvec 32
4 sort array 2 3
5 constd 1 0
6 constd 1 1
7 constd 2 0
8 constd 3 0
9 state 2 idx
10 state 4 mem
11 state 2 i
12 constd 2 0
13 constd 2 1
14 add 2 11 13
15 constd 2 7
16 ute 1 11 15 i.lt_const
17 ite 2 16 14 11 i.ite_res
18 next 2 11 17
19 write 3 10 11 12
20 init 2 11 7
21 ult 1 9 11
22 read 3 10 9
23 neq 1 22 8
24 and 1 21 23
25 next 4 10 19
26 bad 24

1 sort bitvec 8
2 sort bitvec (3-14)
3 sort array 2 1
4 constd 2 3
5 constd 1 8
6 state 3
7 write 3 6 4 5
8 read 1 7 4
9 constd 1 5
10 write 3 7 4 5
11 read 1 10 4
12 sort bitvec 1
13 eq 12 8 11
14 bad 13

1 sort bitvec (3-14)
2 sort array 1 1
3 state 2
4 input 1
5 constd 1 0
6 constd 1 1
7 constd 1 2
8 write 2 3 4 5
9 write 2 8 6 7
10 write 2 9 4 9
11 read 1 10 4
12 sort bitvec 1
13 eq 12 11 5
14 bad 13
(3) Experiments

<table>
<thead>
<tr>
<th>Tool Algorithm</th>
<th>ABC PDR</th>
<th></th>
<th>AVR PDR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Tasks</td>
<td>36</td>
<td></td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>Input Configuration</td>
<td>AIGER balanced skewed</td>
<td>/ balanced skewed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proofs</td>
<td>33 33</td>
<td></td>
<td>24 32 29</td>
<td></td>
</tr>
<tr>
<td>Alarms</td>
<td>0 0</td>
<td></td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>Wrong proofs</td>
<td>0 0</td>
<td></td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>Wrong alarms</td>
<td>0 0</td>
<td></td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>Timeouts</td>
<td>3 3</td>
<td></td>
<td>0 4    3</td>
<td></td>
</tr>
<tr>
<td>Out of memory</td>
<td>0 0</td>
<td></td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>Other inconclusive</td>
<td>0 0</td>
<td></td>
<td>12 0 4</td>
<td></td>
</tr>
</tbody>
</table>

Total correct: 33

Total correct: 36
## (3) Experiments

<table>
<thead>
<tr>
<th>Tool Algorithm</th>
<th>CPACHECKER Pred. Abs.</th>
<th>ESBMC k-Induction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Balanced</td>
<td>Skewed</td>
</tr>
<tr>
<td>Proofs</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>Alarms</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wrong proofs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wrong alarms</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Timeouts</td>
<td>29</td>
<td>12</td>
</tr>
<tr>
<td>Out of memory</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Other inconclusive</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

- Total correct: 16
- Total correct: 17
- Total correct: 26
- Total correct: 18
- Total correct: 30
- Total correct: 30
4

Conclusion
(4) Conclusion

- Btor2blaster increased number of proofs overall
- 51 out of 318 array containing Btor2 tasks can now be used with ABC
- Usage of both options (skewed and balanced) can increase total number of proofs and alarms
- ITE pattern occurs in all but one task of the collection which makes copies necessary (future work: identify pattern reliably and avoid copy)
- ALAP-Scheduling works as intended on manually created tasks but doesn’t guarantee optimum (future work: integer linear programming for optimal scheduling)
- Array problem likely caused by SMT solver