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

Bachelor-Thesis Defense Talk by Salih Ates

Date: August 30th , 2023

0

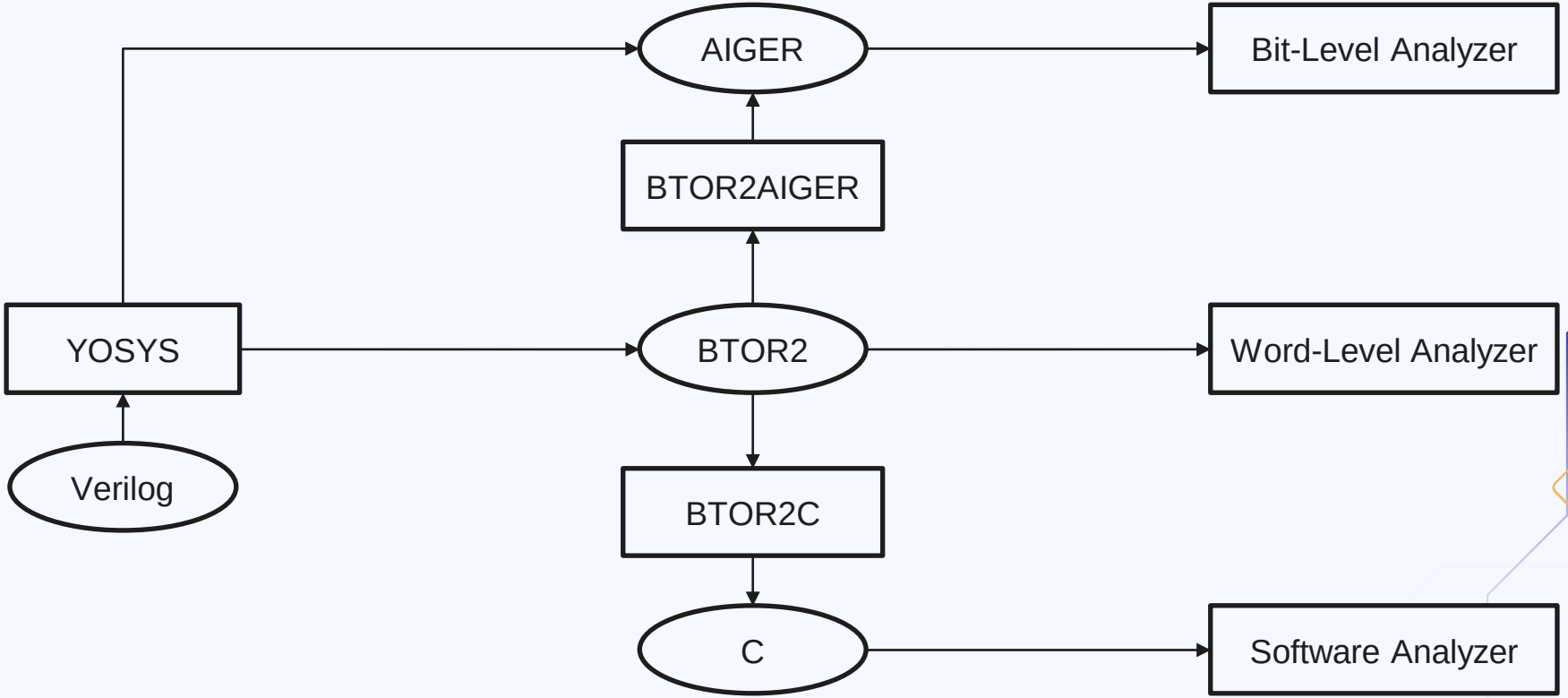
Intro Talk Recap

Motivation

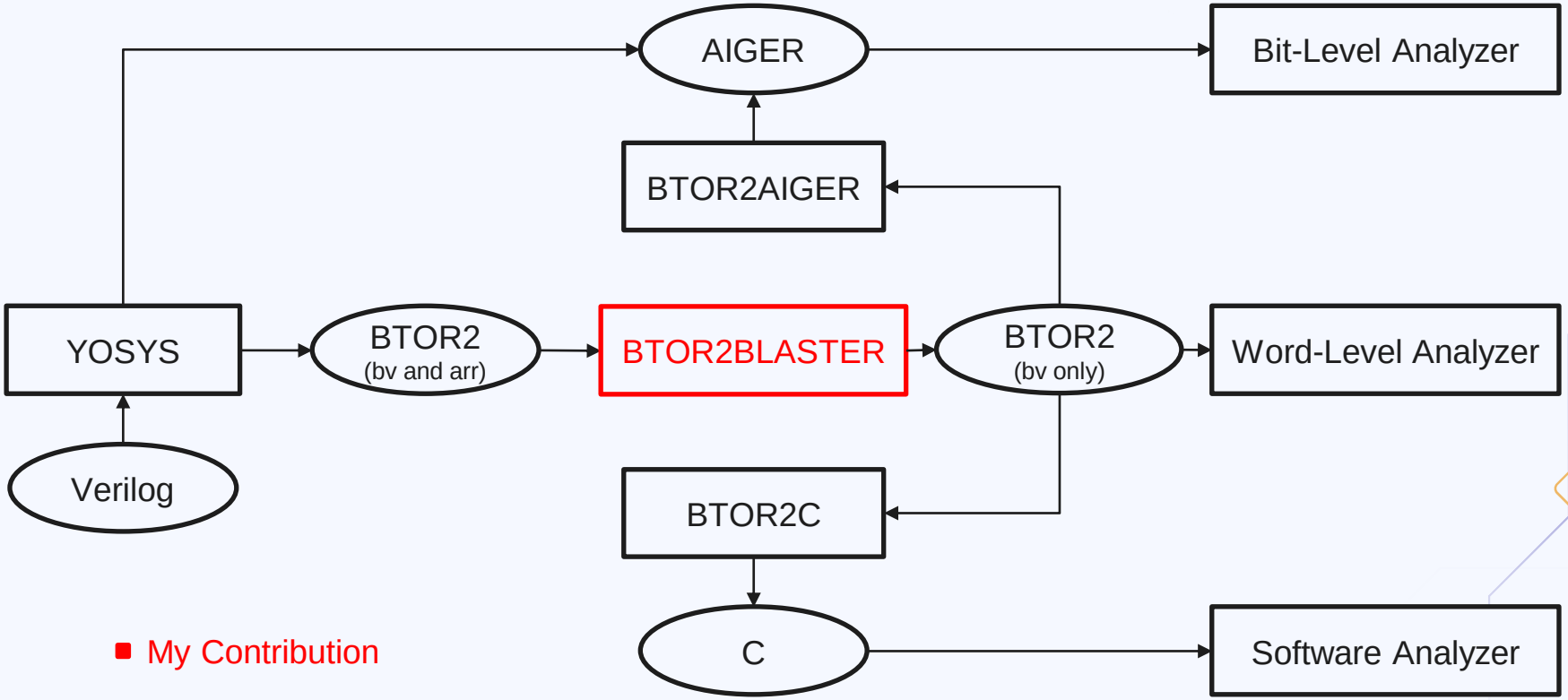
Tool		ABC	AVR	CPAchecker		Esbmc		VeriAbs	
Algorithm		PDR	PDR	Pred. Abs.		k-Induction		Loop Abs.	
Input	Tasks	Aiger	Btor2	C-e	C-l	C-e	C-l	C-e	C-l
Correct results	1498	**862**	736	274	280	401	**410**	392	393
BV proofs	868	**524**	458	188	**189**	88	93	53	49
BV alarms	473	**338**	233	86	91	311	315	337	**342**
Array proofs	140	–	**45**	0	0	0	0	0	0
Array alarms	17	–	0	0	0	**2**	**2**	**2**	**2**
Wrong proofs		0	0	0	0	0	0	2	2
Wrong alarms		0	0	0	0	0	0	1	1
Timeouts		479	559	924	922	554	551	1049	1042
Out of memory		0	3	9	7	543	537	3	4
Other inconclusive		0	200	291	289	0	0	51	56

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



(1) Array blasting



- My Contribution

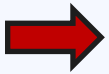
(2) ALAP-Scheduling of write-ops

(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;
```

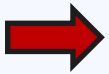
(2) ALAP-Scheduling of write-ops

(as-late-as-possible)

Objective:

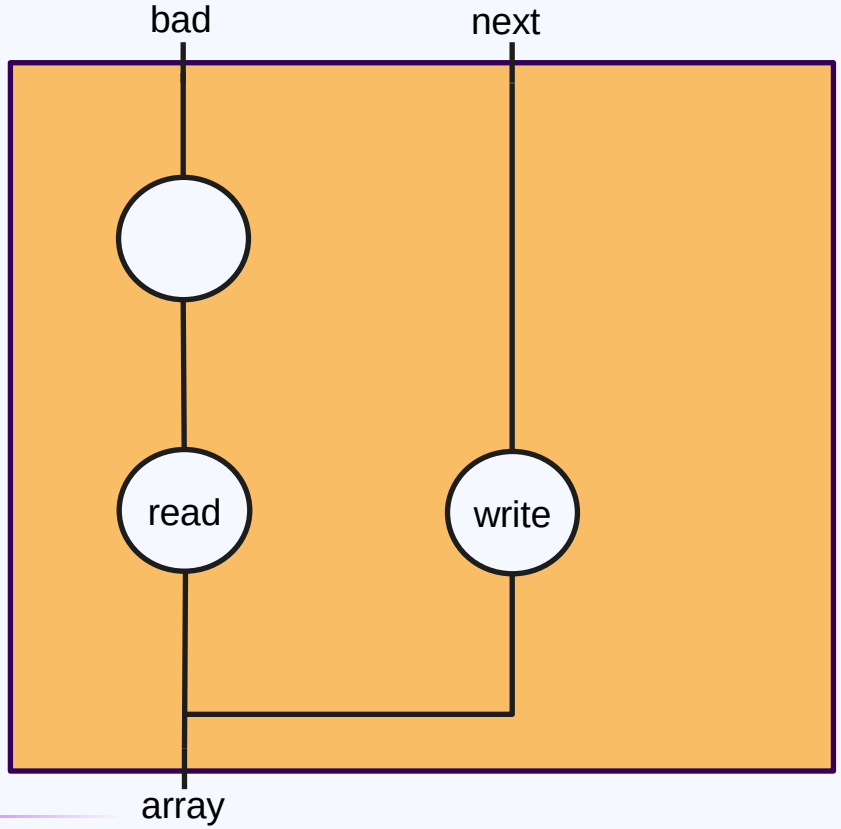
Minimize number of duplications for write operations.

```
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



schedule = [array, write, read]

→ Duplication needed

schedule = [array, read, write]

→ Duplication can be avoided

1

Array blasting

(1) Array blasting

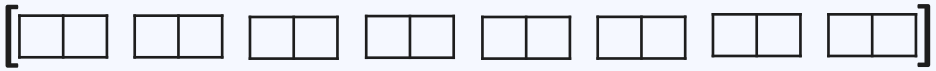
```
1 sort bitvec 1
2 sort bitvec 2
3 sort bitvec 3
4 sort array 3 2
5 state 4
6 input 3
7 constd 2 1
8 write 4 5 6 7
9 read 2 5 6
10 eq 1 5 8
11 neq 1 5 8
12 ite 3 10 5 8
13 init 3 5 8
14 next 3 5 8
```

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

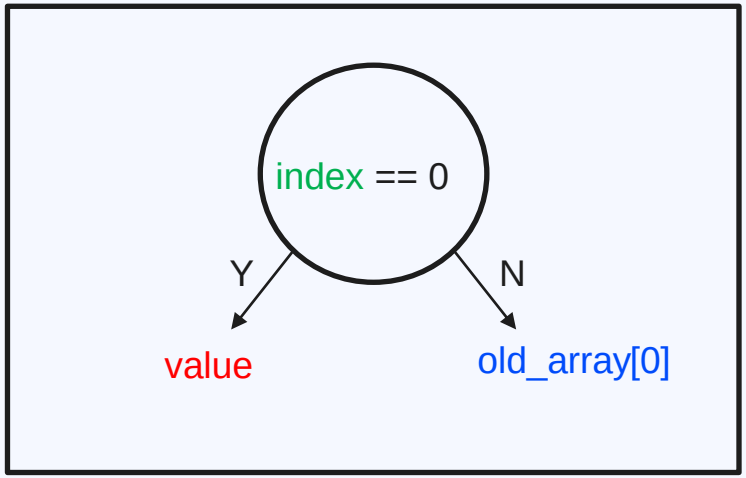
(1) Array blasting – write

```
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
```

```
5 state 2
6 state 2
7 state 2
8 state 2
9 state 2
10 state 2
11 state 2
12 state 2
```

id write sort_n old_array index value

write_array[0] =



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
```

```
5 state 2
6 state 2
7 state 2
8 state 2
9 state 2
10 state 2
11 state 2
12 state 2
```

id write sort_n old_array index value

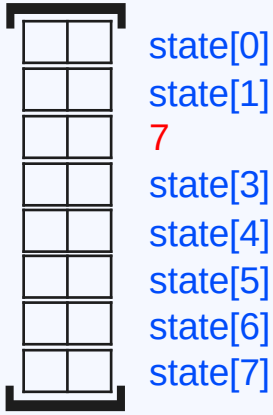
```
 8 ite 2 (input == 0) 7 state[0]
 9 ite 2 (input == 1) 7 state[1]
10 ite 2 (input == 2) 7 state[2]
11 ite 2 (input == 3) 7 state[3]
12 ite 2 (input == 4) 7 state[4]
13 ite 2 (input == 5) 7 state[5]
14 ite 2 (input == 6) 7 state[6]
15 ite 2 (input == 7) 7 state[7]
```

(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
```

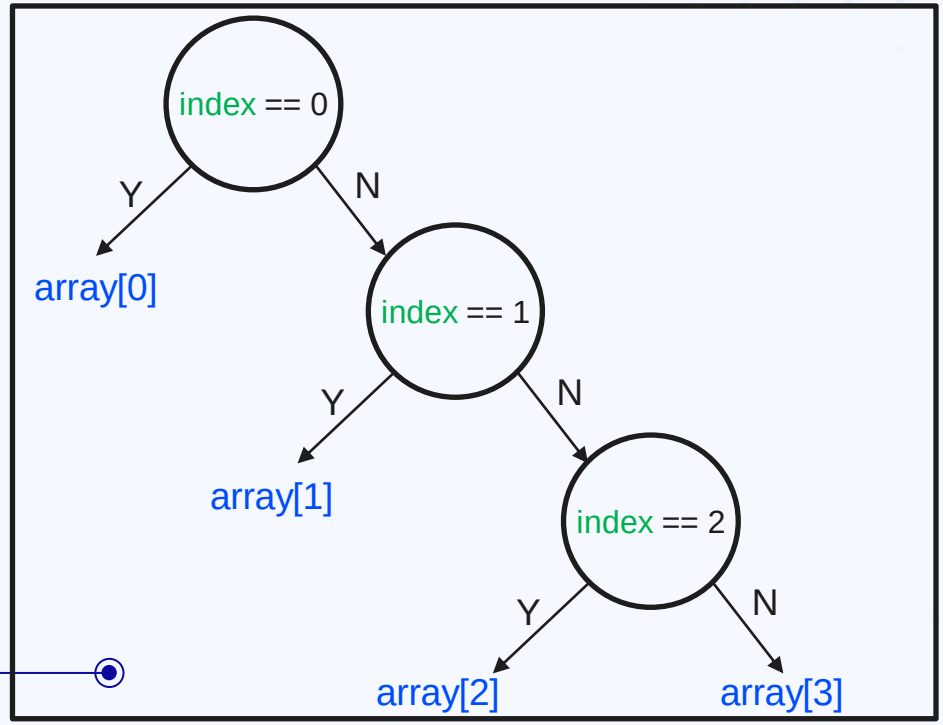
```
5 state 2
6 state 2
7 state 2
8 state 2
9 state 2
10 state 2
11 state 2
12 state 2
```

id write sort_n old_array constd value



(1) Array blasting – read (skewed)

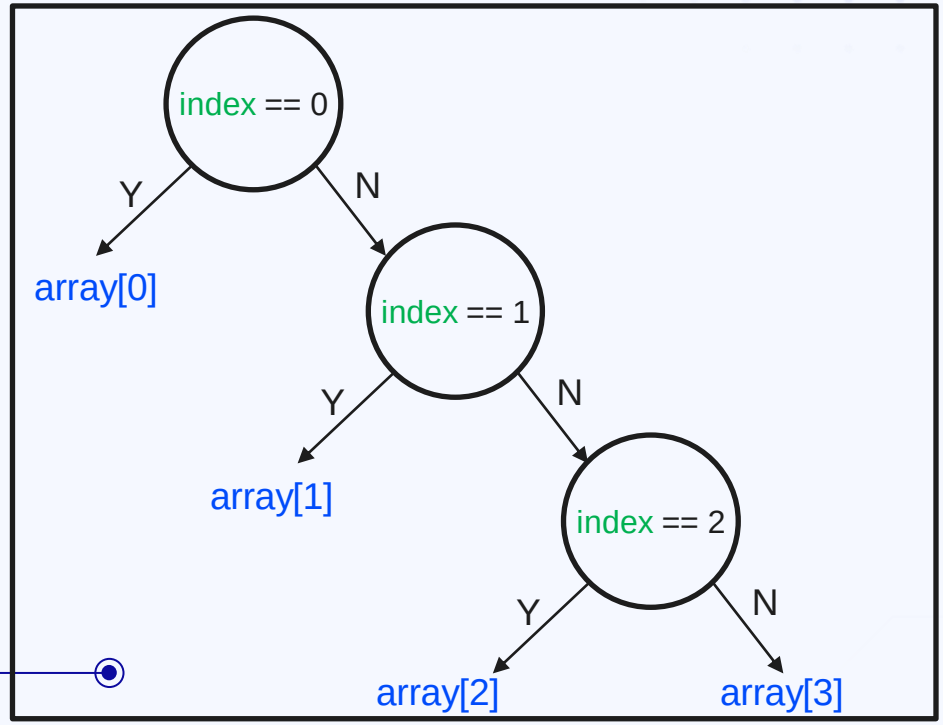
id read sort_n array index



(1) Array blasting – read (skewed)

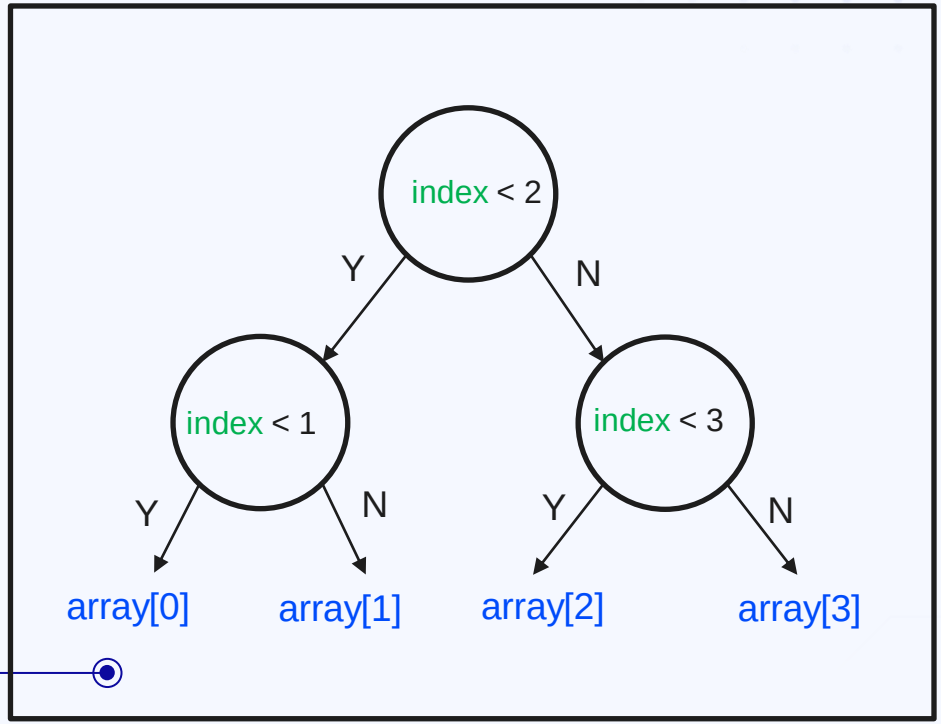
id read sort_n array index

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)

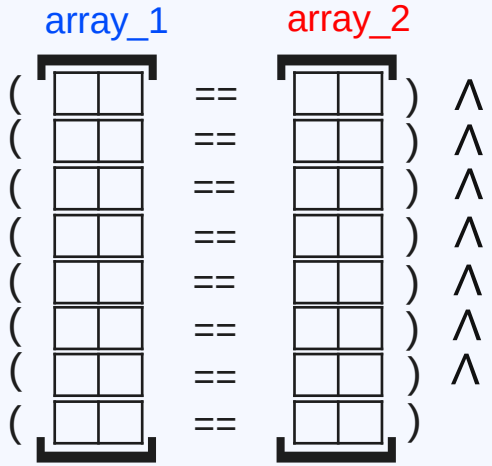
id read sort_n array index



(1) Array blasting – eq/neq

id eq sort_n array_1 array_2
id neq sort_n array_1 array_2

5 state 2		15 state 2
6 state 2		16 state 2
7 state 2		17 state 2
8 state 2	==	18 state 2
9 state 2		19 state 2
10 state 2		20 state 2
11 state 2		21 state 2
12 state 2		22 state 2



(1) Array blasting – ite

id ite sort_n cond array_1 array_2

35 ite 2 cond 5 state 2 15 state 2
36 ite 2 cond 6 state 2 16 state 2
37 ite 2 cond 7 state 2 17 state 2
38 ite 2 cond 8 state 2 18 state 2
39 ite 2 cond 9 state 2 19 state 2
40 ite 2 cond 10 state 2 20 state 2
41 ite 2 cond 11 state 2 21 state 2
42 ite 2 cond 12 state 2 22 state 2

(1) Array blasting – init

id init sort_n array_1 array_2

35 init 2	5 state 2	15 state 2
36 init 2	6 state 2	16 state 2
37 init 2	7 state 2	17 state 2
38 init 2	8 state 2	18 state 2
39 init 2	9 state 2	19 state 2
40 init 2	10 state 2	20 state 2
41 init 2	11 state 2	21 state 2
42 init 2	12 state 2	22 state 2

(1) Array blasting – next

id next sort_n array_1 array_2

35 next 2	5 state 2	15 state 2
36 next 2	6 state 2	16 state 2
37 next 2	7 state 2	17 state 2
38 next 2	8 state 2	18 state 2
39 next 2	9 state 2	19 state 2
40 next 2	10 state 2	20 state 2
41 next 2	11 state 2	21 state 2
42 next 2	12 state 2	22 state 2

(1) Array blasting

All array operations:

- ✓ state
- ✓ write
- ✓ read
- ✓ eq
- ✓ neq
- ✓ ite
- ✓ init
- ✓ next

2 ALAP-Scheduling of write-ops

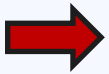
(2) ALAP-Scheduling of write-ops

(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;
```

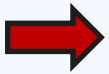
(2) ALAP-Scheduling of write-ops

(as-late-as-possible)

Objective:

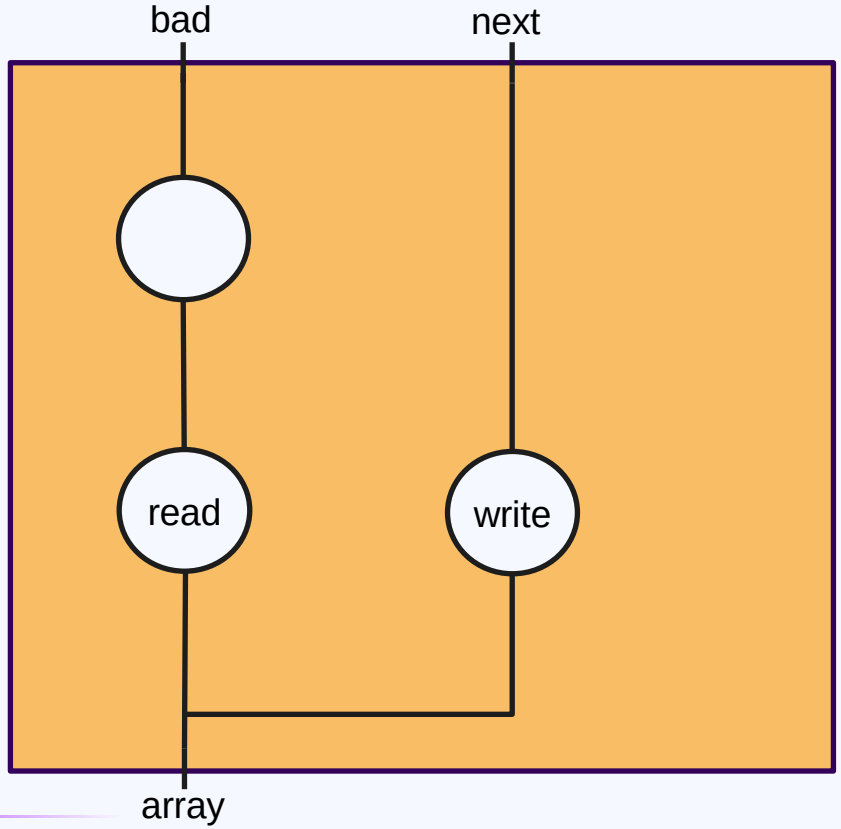
Minimize number of duplications for write operations.

```
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

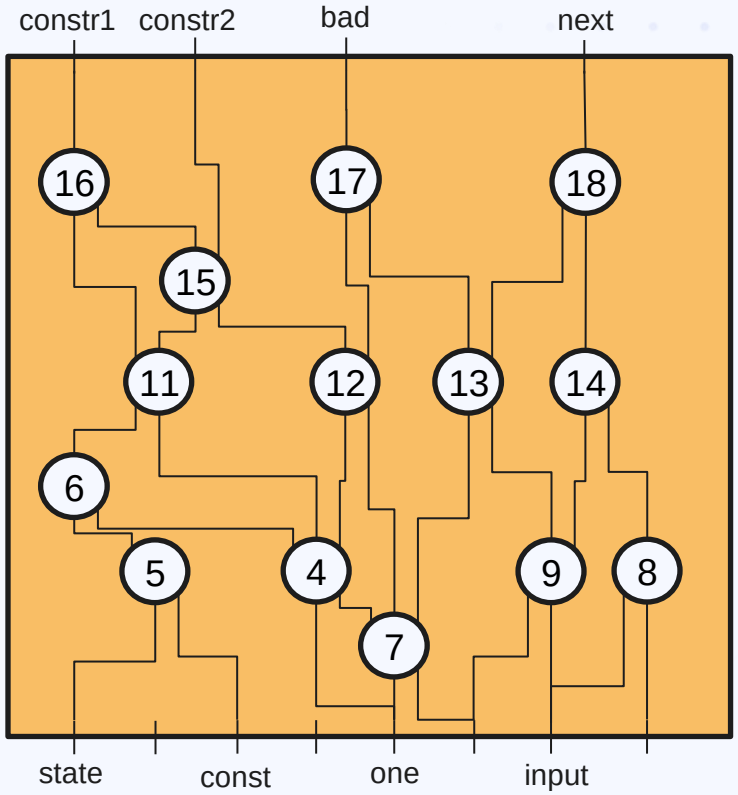


→ write → read → !

→ read → write → ✓

(2) ALAP-Scheduling of write-ops

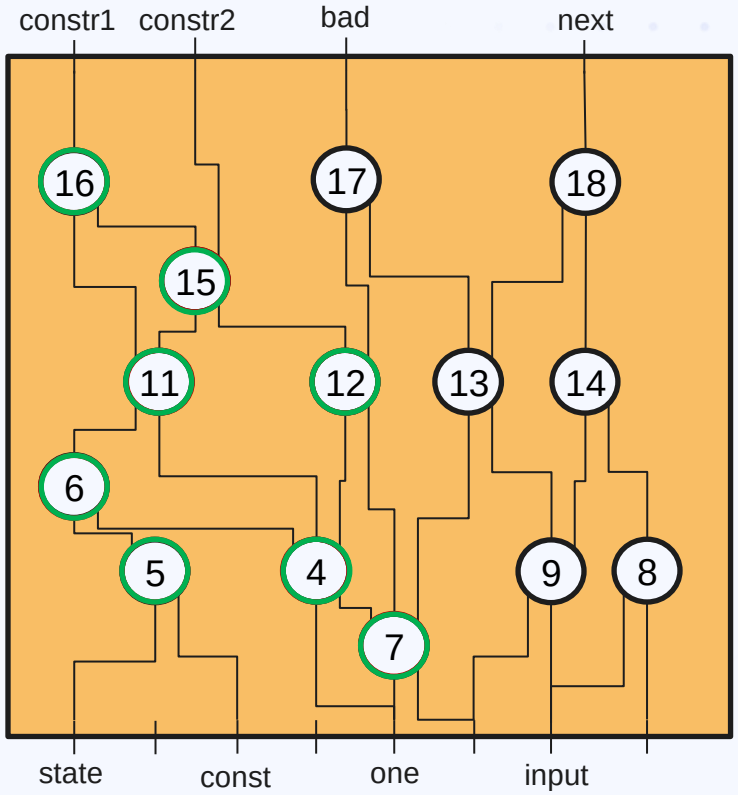
```
for(;;) {
```




```
}
```

(2) ALAP-Scheduling of write-ops

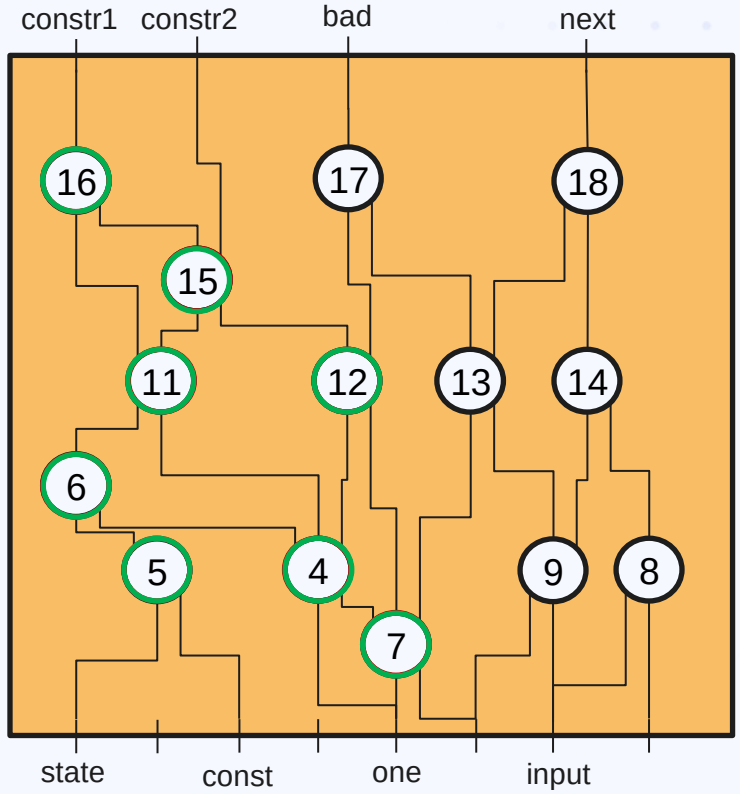
```
for(;;) {
```




```
}
```

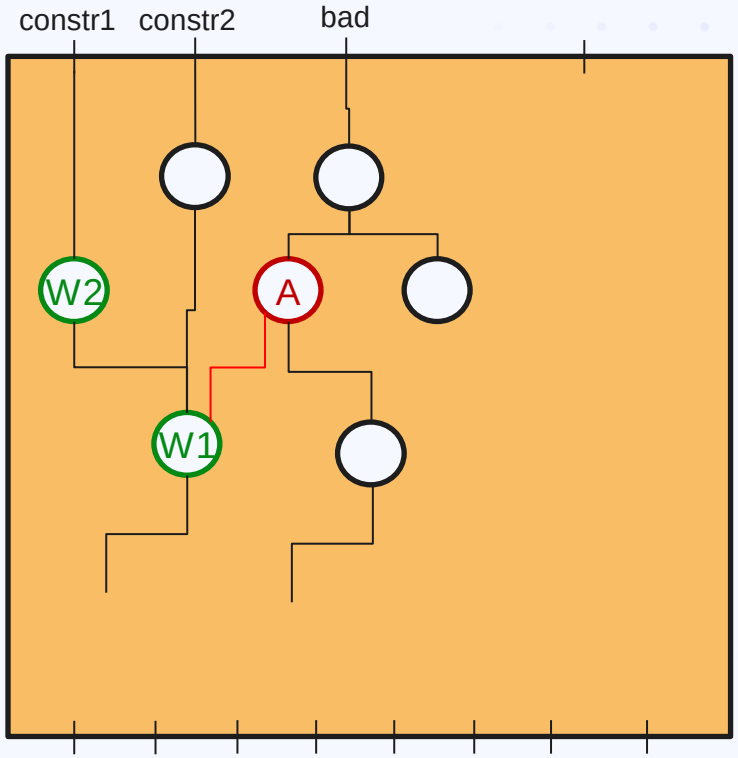
(2) ALAP-Scheduling of write-ops

```
for(;;) {
    var_5 = …
    …
    var_16 = …
    assume(constr1);
    …
    assume(constr2);
    …
    assert(bad);
    …
    state_5 = next_arg_1;
    }
```



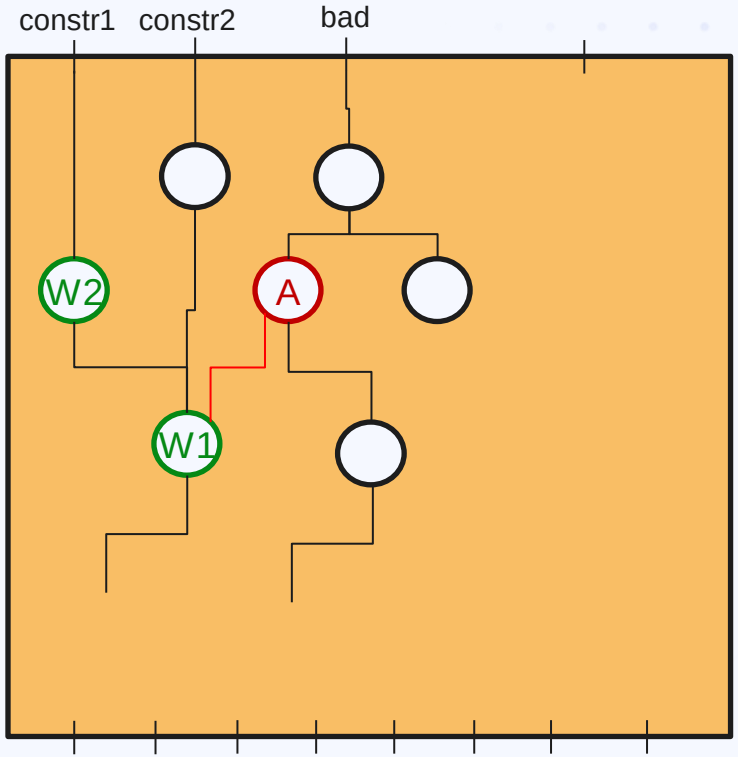
(2) ALAP-Scheduling of write-ops

```
for(;;) {
    W1
    W2
    assume(constr1);
    …
    assume(constr2);
    A
    verifier(bad);
    }
```

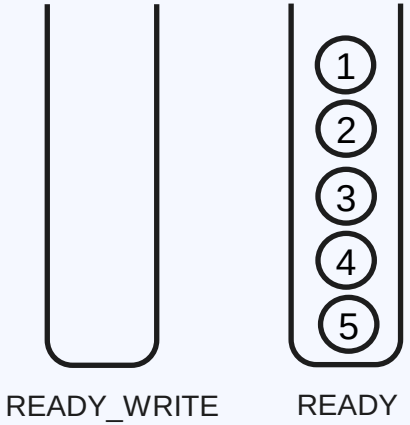


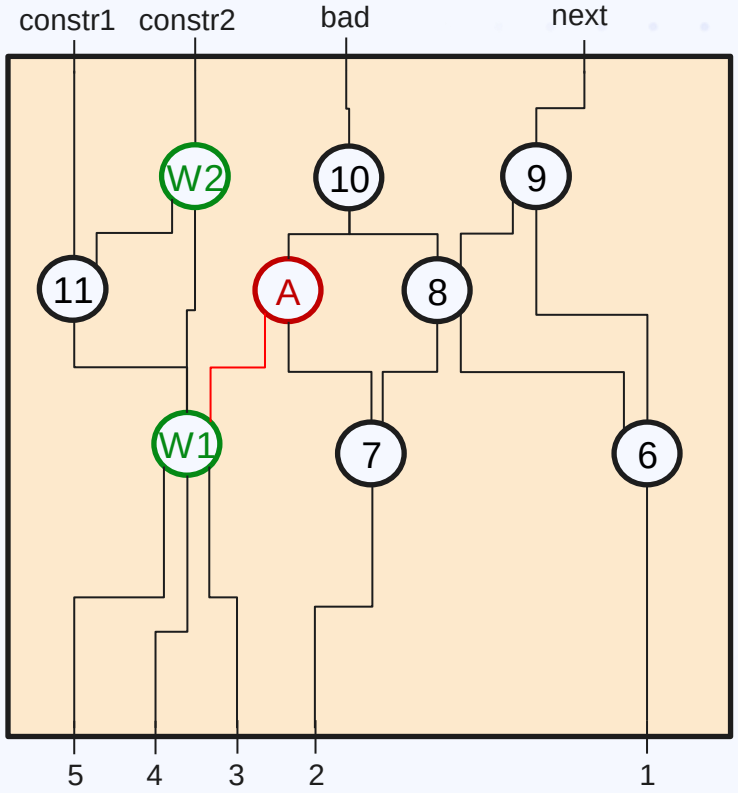
(2) ALAP-Scheduling of write-ops

```
for(;;) {
    W1
    A
    W2
    assume(constr1);
    …
    assume (constr2);
    …
    assert(bad);
    }
```



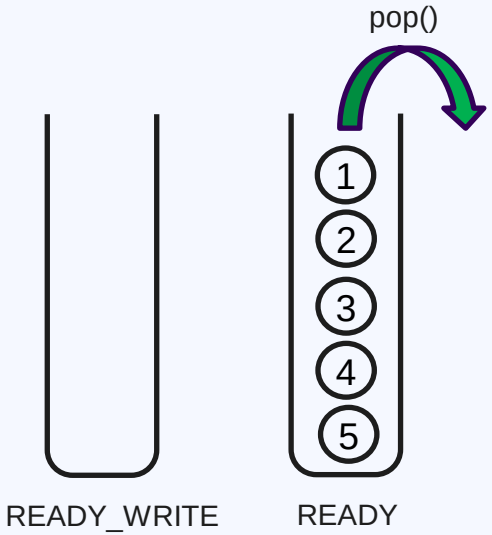
(2) ALAP-Scheduling of write-ops



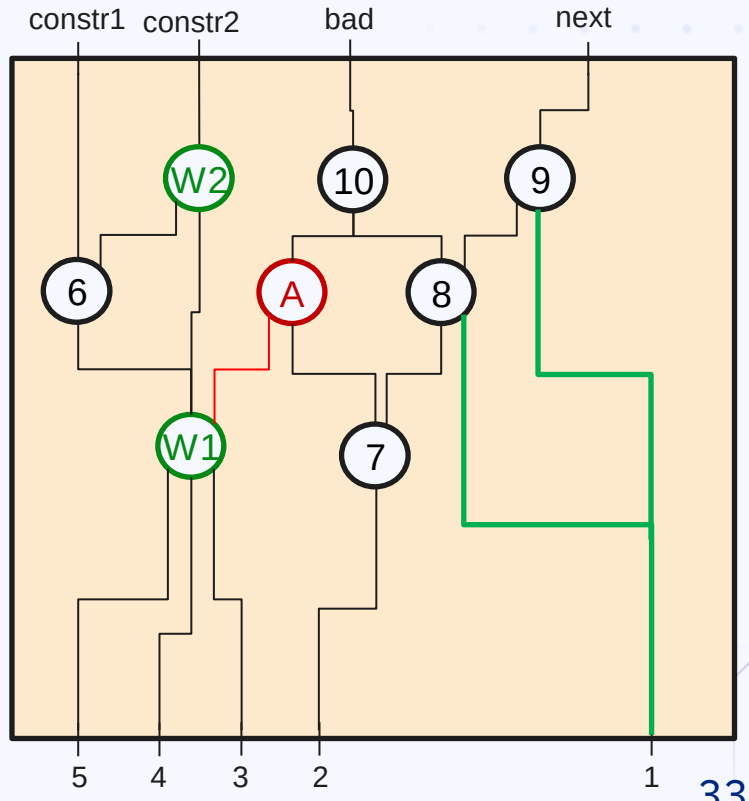


SCHEDULE = [ ]

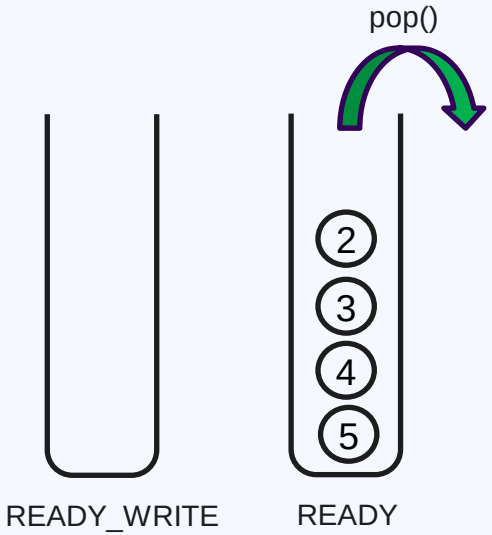
(2) ALAP-Scheduling of write-ops



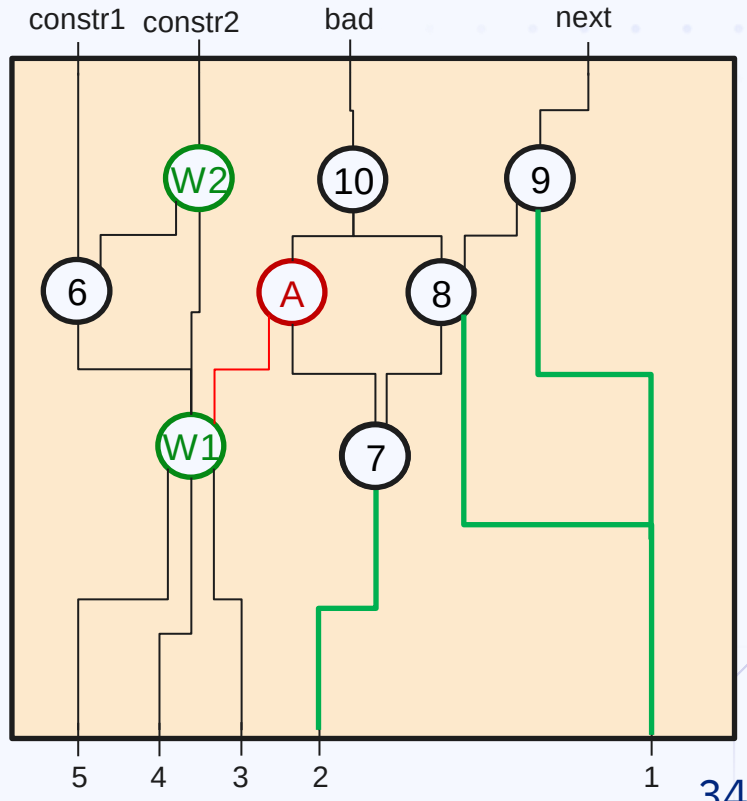
SCHEDULE = [1]



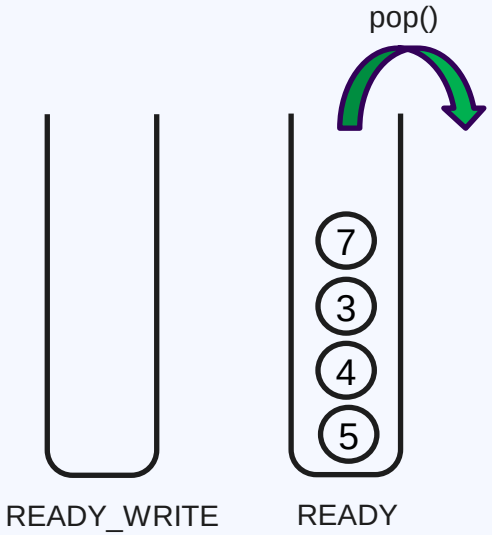
(2) ALAP-Scheduling of write-ops



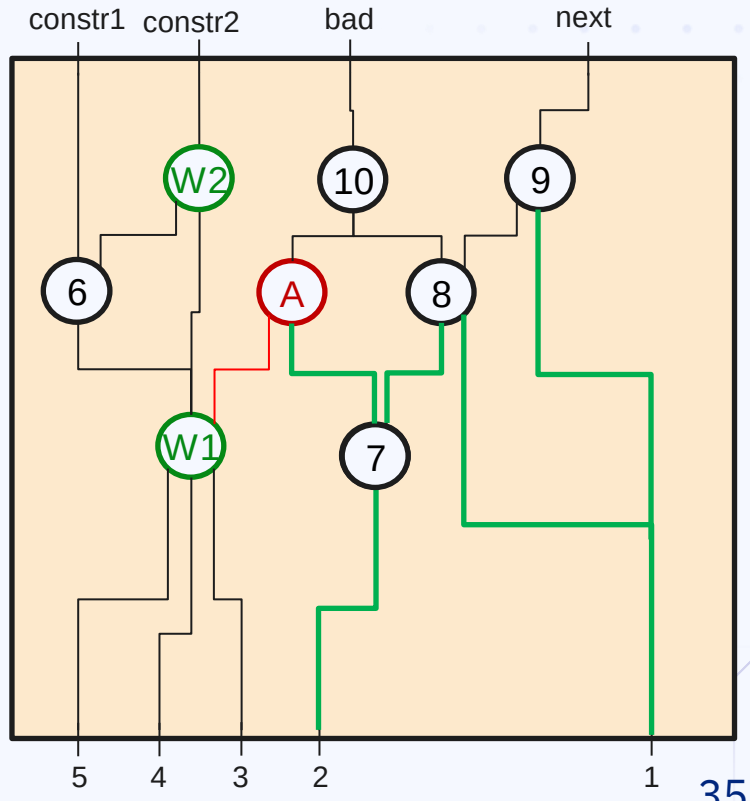
SCHEDULE = [1 2]



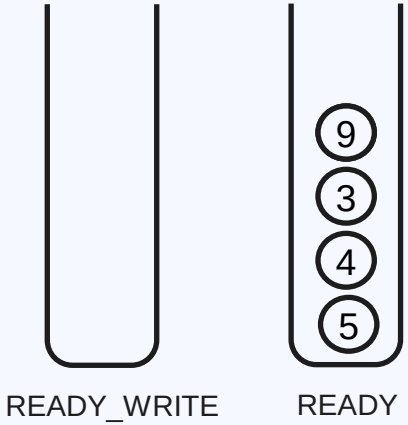
(2) ALAP-Scheduling of write-ops



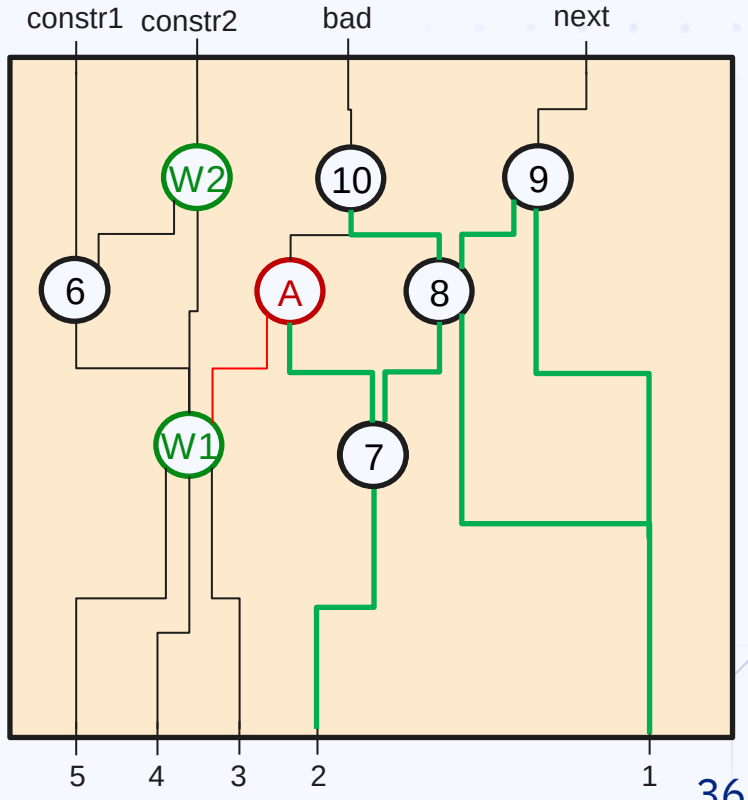
SCHEDULE = [1 2 7]



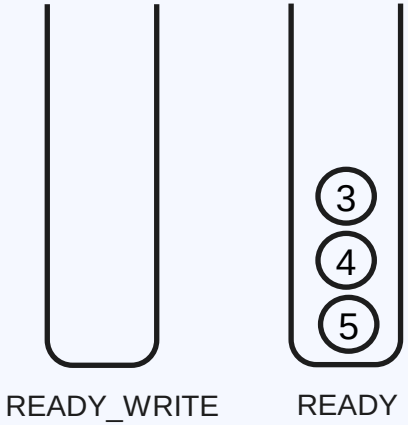
(2) ALAP-Scheduling of write-ops



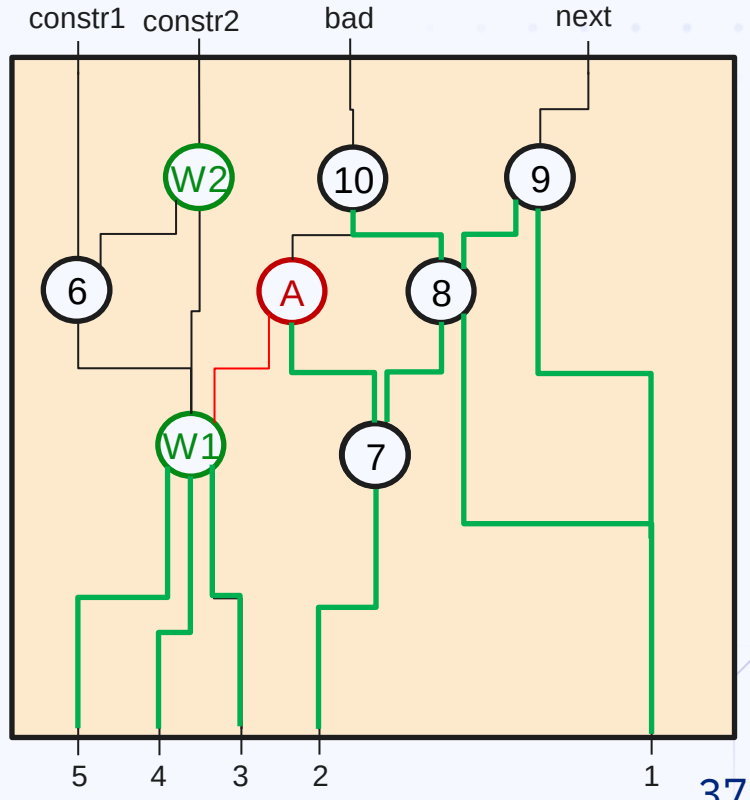
SCHEDULE = [① ② ⑦ ⑧ ⑨]



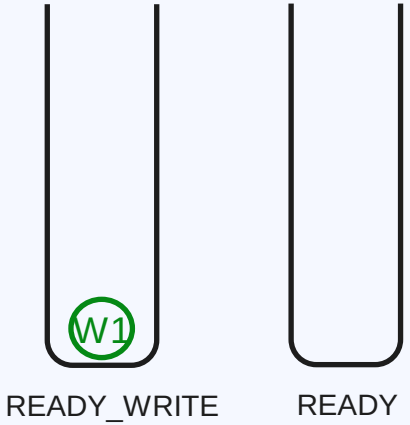
(2) ALAP-Scheduling of write-ops



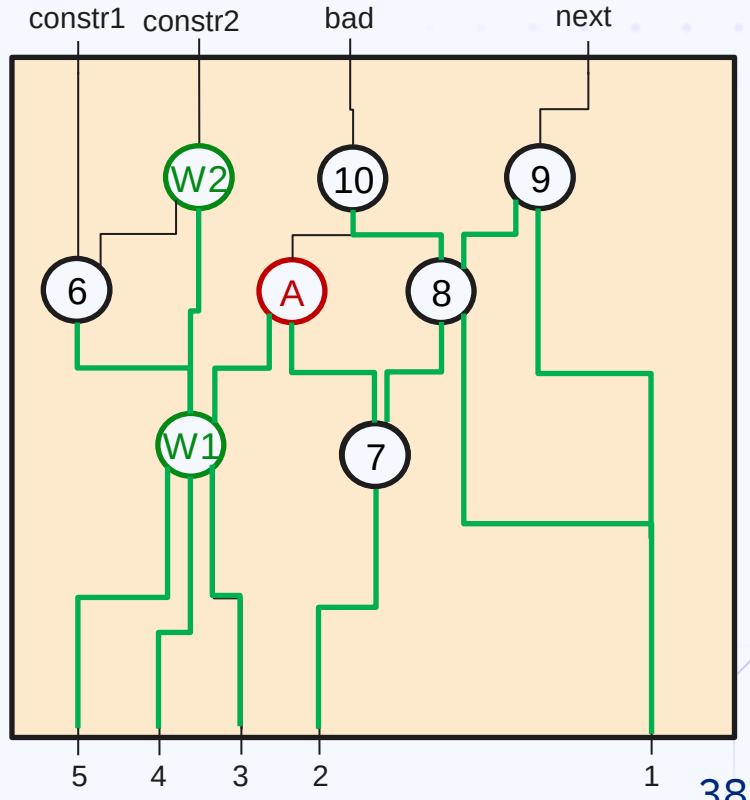
SCHEDULE = [1 2 7 8 9 3 4 5]



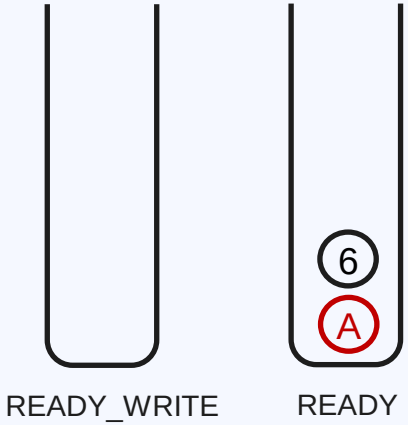
(2) ALAP-Scheduling of write-ops



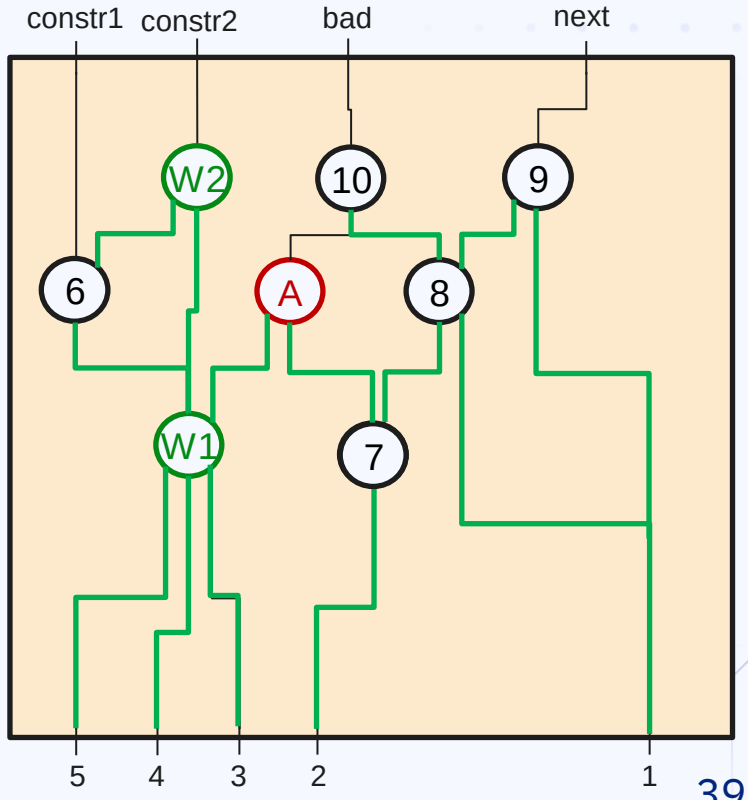
SCHEDULE = [1 2 7 8 9 3 4 5 W1]



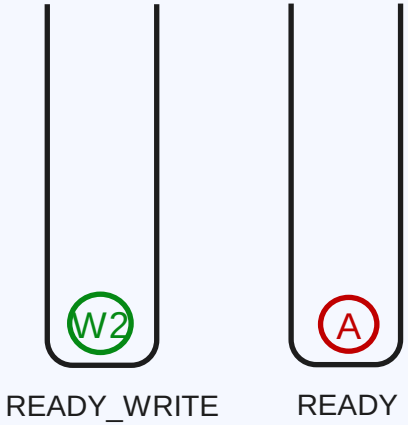
(2) ALAP-Scheduling of write-ops



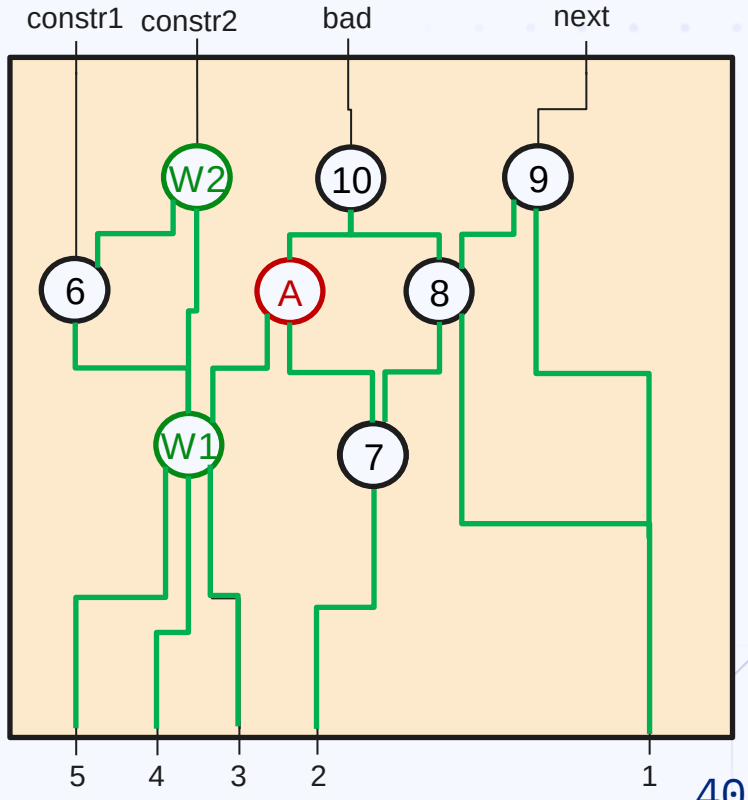
SCHEDULE = [1 2 7 8 9 3 4 5 W1 6]



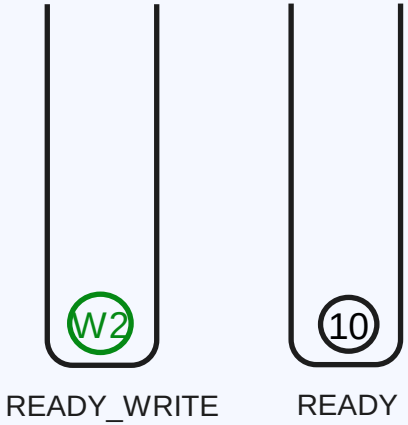
(2) ALAP-Scheduling of write-ops



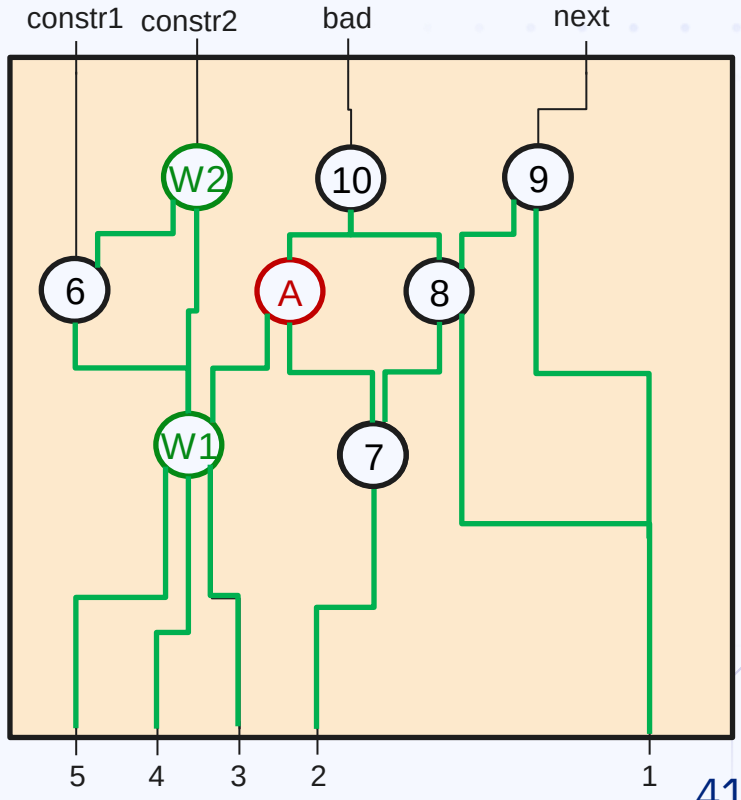
SCHEDULE = [1 2 7 8 9 3 4 5 W1 6 A]



(2) ALAP-Scheduling of write-ops



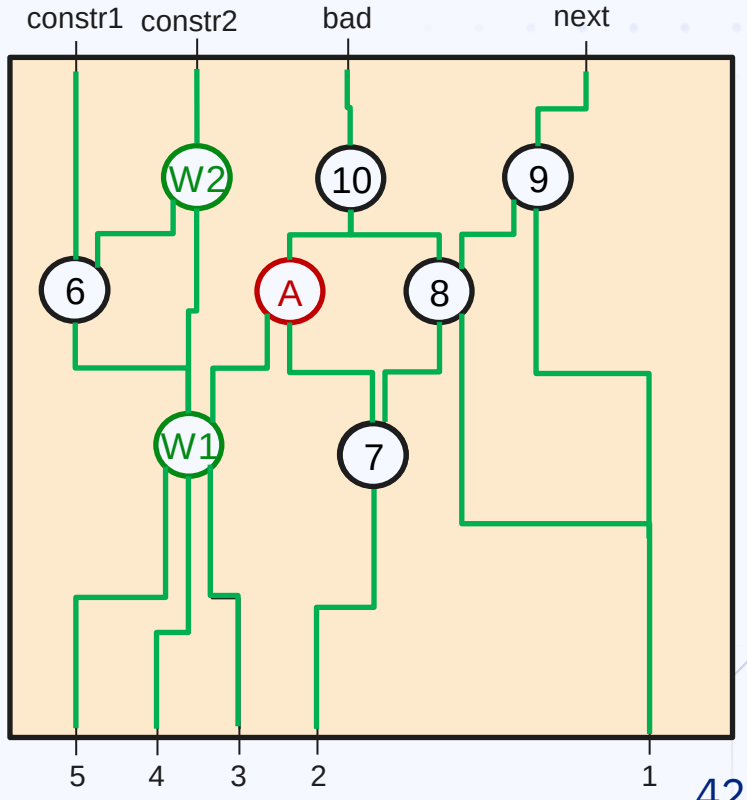
SCHEDULE = [1 2 7 8 9 3 4 5 W1 6 A 10 W2]



(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]

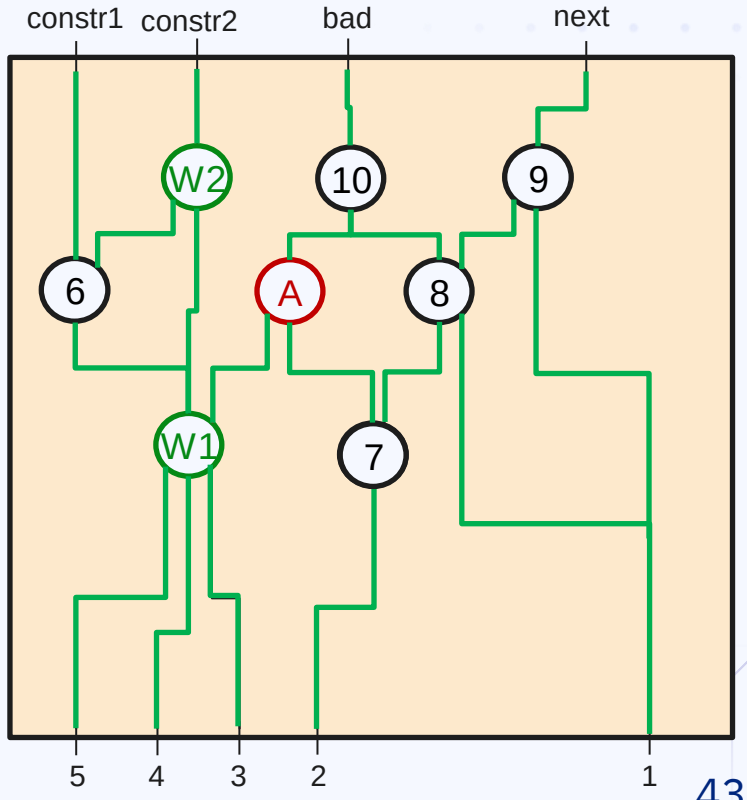


(2) ALAP-Scheduling of write-ops

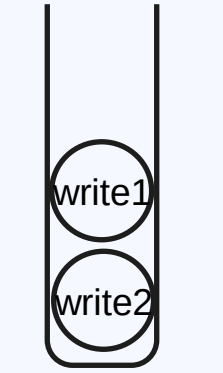
```
for(;;) {
    ...
    W1
    A
    W2
    assume(constr1);
    assume(constr2);
    assert(bad);
    state_x = next_arg
    }
```

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

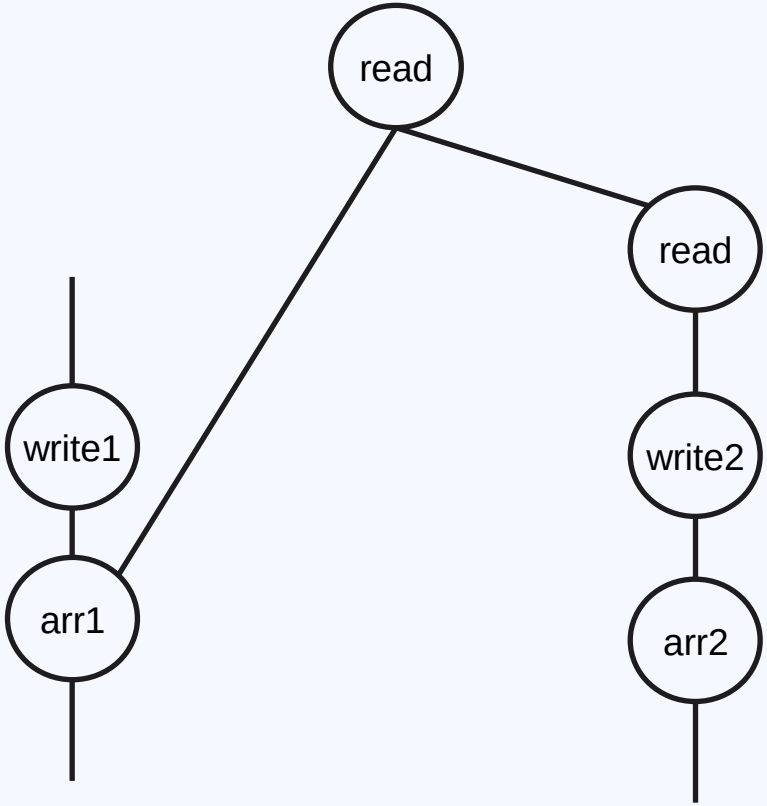
SCHEDULE = [1 2 7 8 9 3 4 5 W1 6 A 10 W2]



(3) Experiments



READY_WRITE



3 Experiments

(3) Experiments

Setup:

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

Verifiers:

- ABC
- AVR
- CPAchecker
- ESBMC
- CBMC

Limits for each task:

- 2 CPU cores
- 15 min of CPU time
- 15 GB RAM

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

Tool	ABC		AVR		
Algorithm	PDR		PDR		
# Tasks	51 (T: 27 F: 24)		318 (T: 294 F: 24)		
Input	AIGER		BTOR2		
Configuration	balanced	skewed	/	balanced	skewed
Proofs	11	11	129	137	136
Alarms	5	5	0	2	2
Wrong proofs	0	0	0	0	0
Wrong alarms	0	0	0	0	0
Timeouts	35	35	84	96	98
Out of memory	0	0	0	0	0
Other inconclusive	0	0	105	83	82

Total correct: 138

Total correct: 143

(3) Experiments

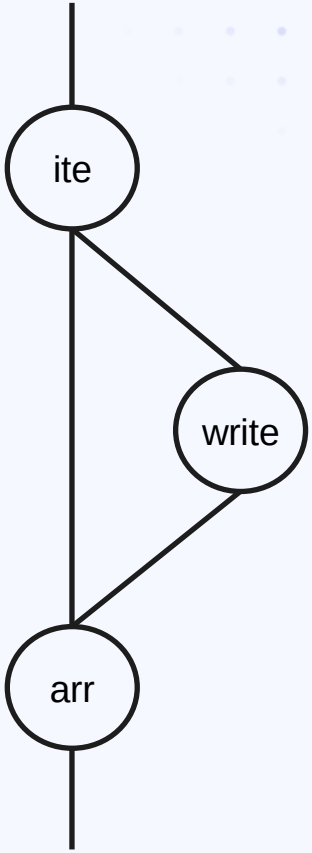
Tool	CPACHECKER				ESBMC				CBMC			
Algorithm	Pred. Abs.				k-Induction				BMC			
# Tasks	175 (T: 154 F: 21)				175 (T: 154 F: 21)				175 (T: 154 F: 21)			
Input	C				C				C			
Configuration	/	balanced	skewed	alap	/	balanced	skewed	alap	/	balanced	skewed	alap
Proofs	0	4	6	0	0	0	0	0	0	0	0	0
Alarms	0	0	0	0	2	2	2	2	19	15	15	20
Wrong proofs	0	0	0	0	0	0	0	0	0	0	0	0
Wrong alarms	0	0	0	0	0	0	0	0	0	0	0	0
Timeouts	173	167	165	173	68	48	48	69	0	0	0	0
Out of memory	0	1	1	0	104	125	125	104	91	84	87	93
Other inconclusive	2	3	3	2	1	0	0	0	65	76	73	62

Total correct: 7

(3) Experiments

1/175 Tasks avoid copies

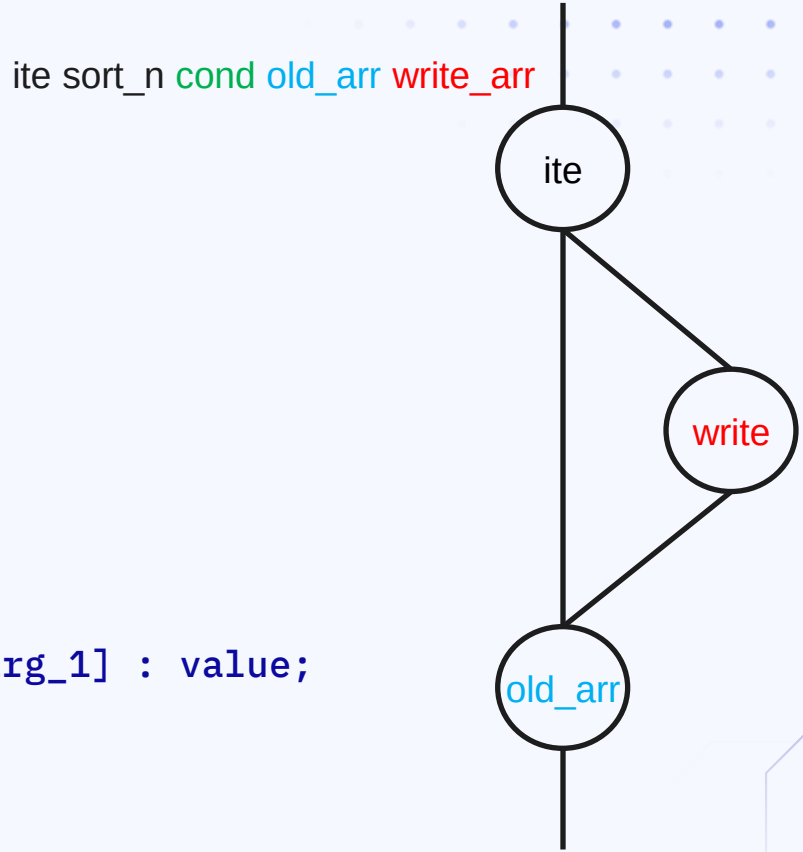
SCHEDULE = [arr, write, ite]



(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 ulte 1 11 15 i_lt_const
17 ite 2 16 14 11 i_ite_res
18 next 2 11 17
19 write 4 10 11 8
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 9
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 6
11 read 1 10 4
12 sort bitvec 1
13 eq 12 11 5
14 bad 13
```

(3) Experiments

Tool	ABC		AVR		
Algorithm	PDR		PDR		
# Tasks	36		36		
Input	AIGER		BTOR2		
Configuration	balanced	skewed	/	balanced	skewed
Proofs	33	33	24	32	29
Alarms	0	0	0	0	0
Wrong proofs	0	0	0	0	0
Wrong alarms	0	0	0	0	0
Timeouts	3	3	0	4	3
Out of memory	0	0	0	0	0
Other inconclusive	0	0	12	0	4

Total correct:33

Total correct: 36

(3) Experiments

Tool	CPACHECKER				ESBMC			
Algorithm	Pred. Abs.				k-Induction			
# Tasks	36				36			
Input	C				C			
Configuration	/	balanced	skewed	alap	/	balanced	skewed	alap
Proofs	4	14	16	25	25	18	18	25
Alarms	0	0	0	0	0	0	0	0
Wrong proofs	0	0	0	0	0	0	0	0
Wrong alarms	0	0	0	0	0	0	0	0
Timeouts	29	12	8	8	8	0	0	10
Out of memory	0	0	2	0	3	18	18	1
Other inconclusive	3	10	10	3	0	0	0	0

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