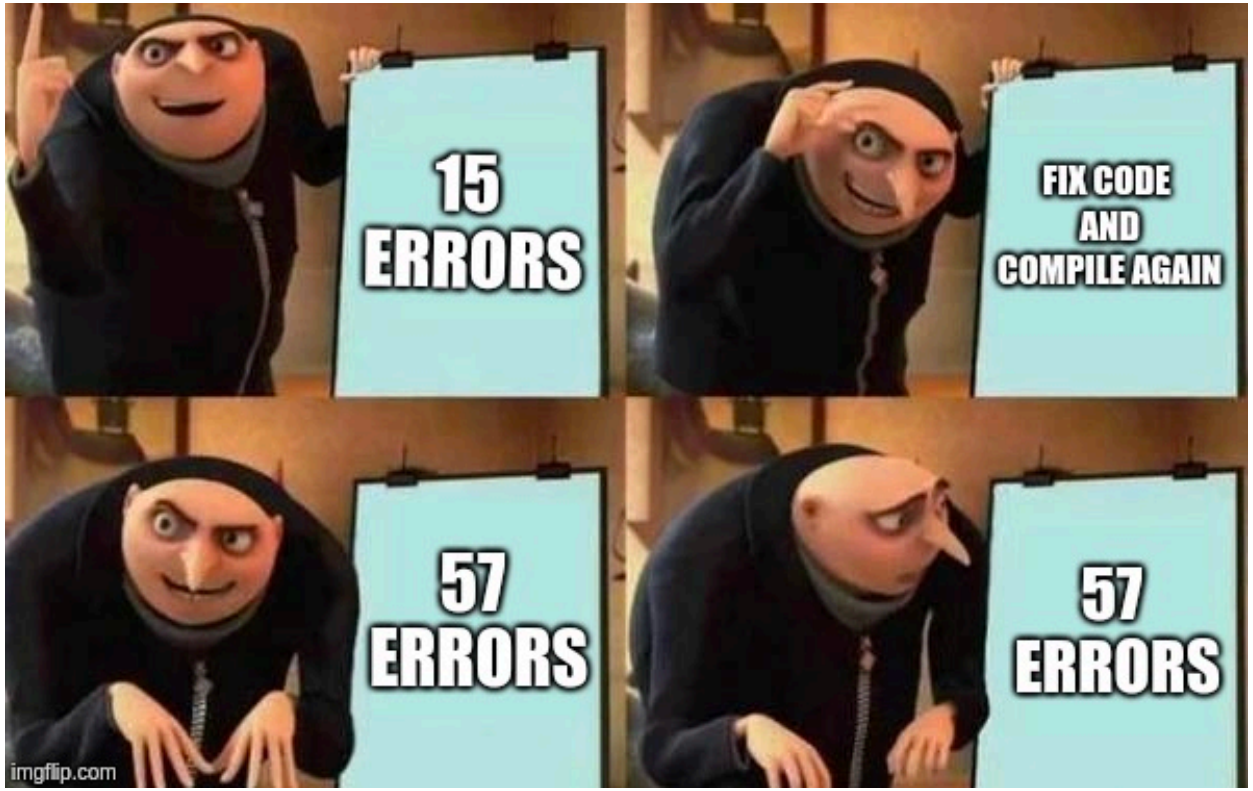
Cause Clue Clauses:

Error Localization using Maximum Satisfiability

20203331 신명근 MyeongGeun Shin

2020.10.14 Wed @ IS893-2020-fall

Debugging is hard



https://www.reddit.com/r/ProgrammerHumor/comments/9g4vml/debugging_be_like/
https://www.facebook.com/caddycode/posts/devlopment-developer-debug-debugging-meme-funnyfriday-fridayfun-caddycode/

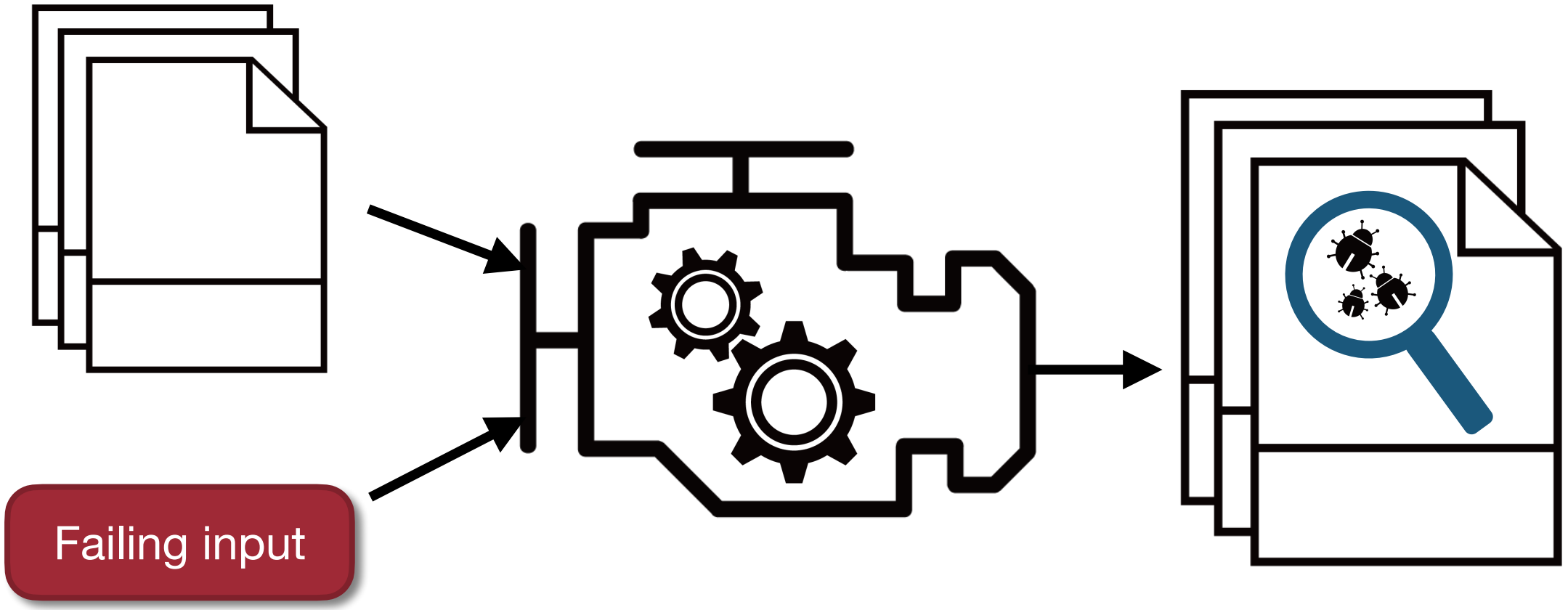
Debugging is HARD!!!

```
1. int* a = 0;
2.
3. ~~~
4.
5. var = &a;
```

Root cause

Null dereference at line 5.

What CCC is about



Some background knowledges

Max SAT

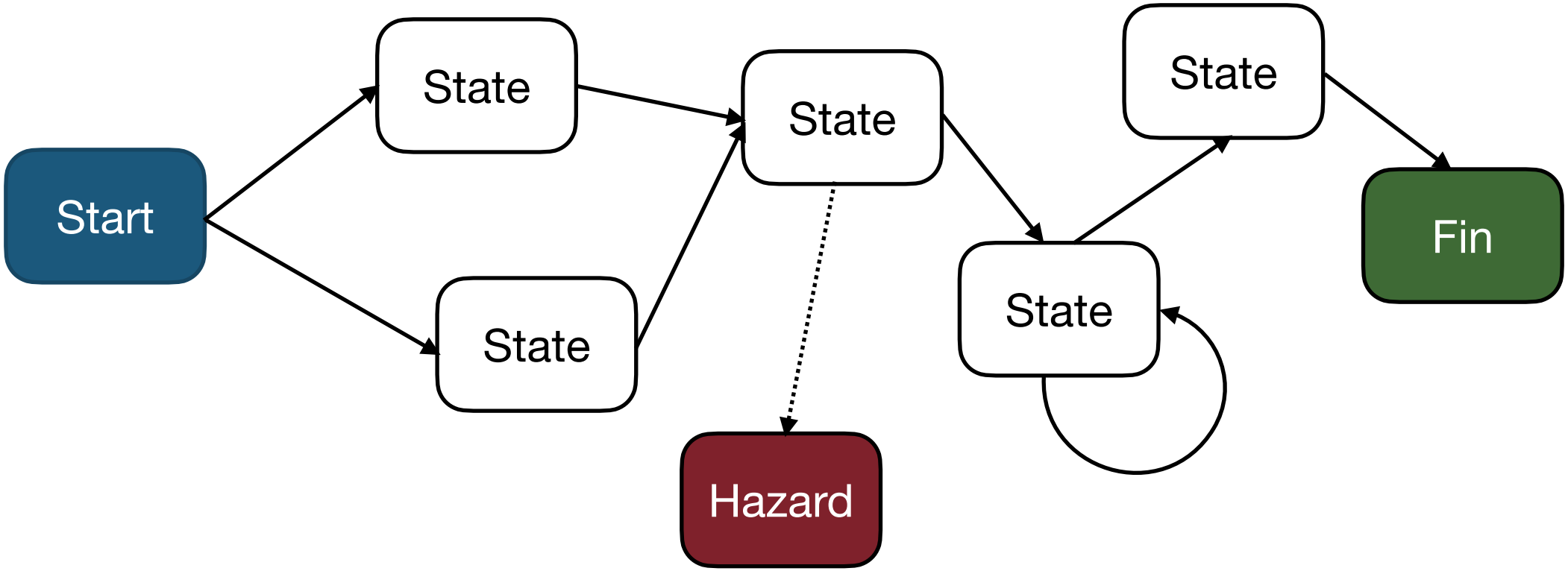
partial Max SAT

$A \wedge B \wedge C \wedge D \wedge E$

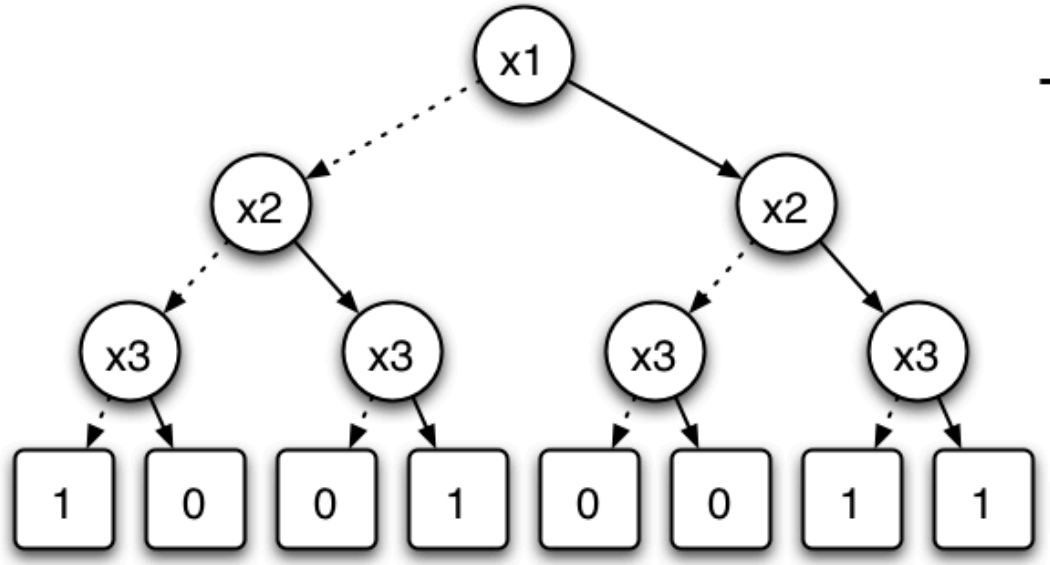
$\mathbf{A} \wedge B \wedge C \wedge D \wedge \mathbf{E}$

Model checking

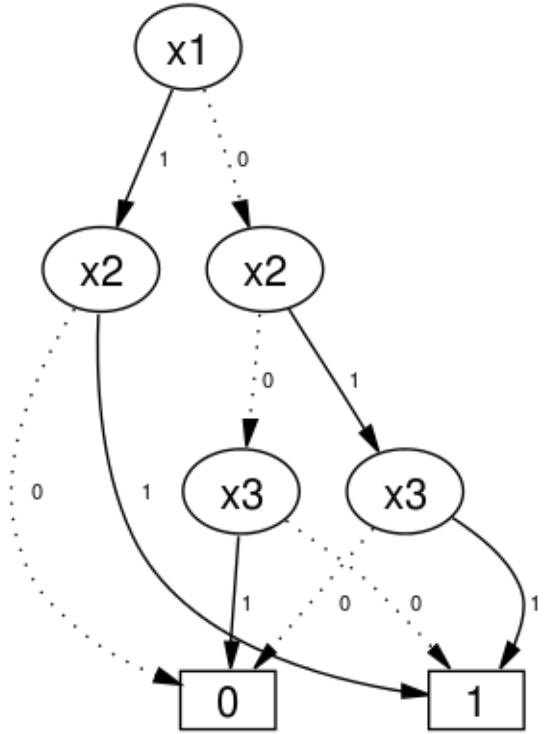
- Checking finite-state model for some spec.



Binary decision diagram

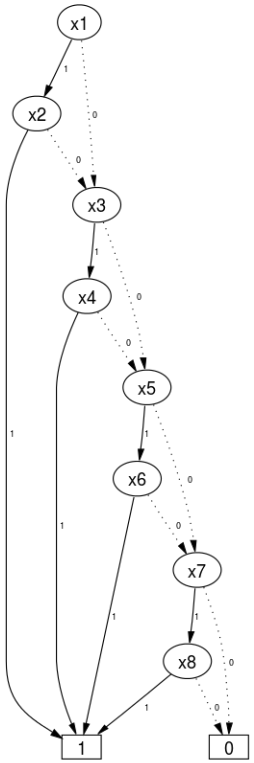


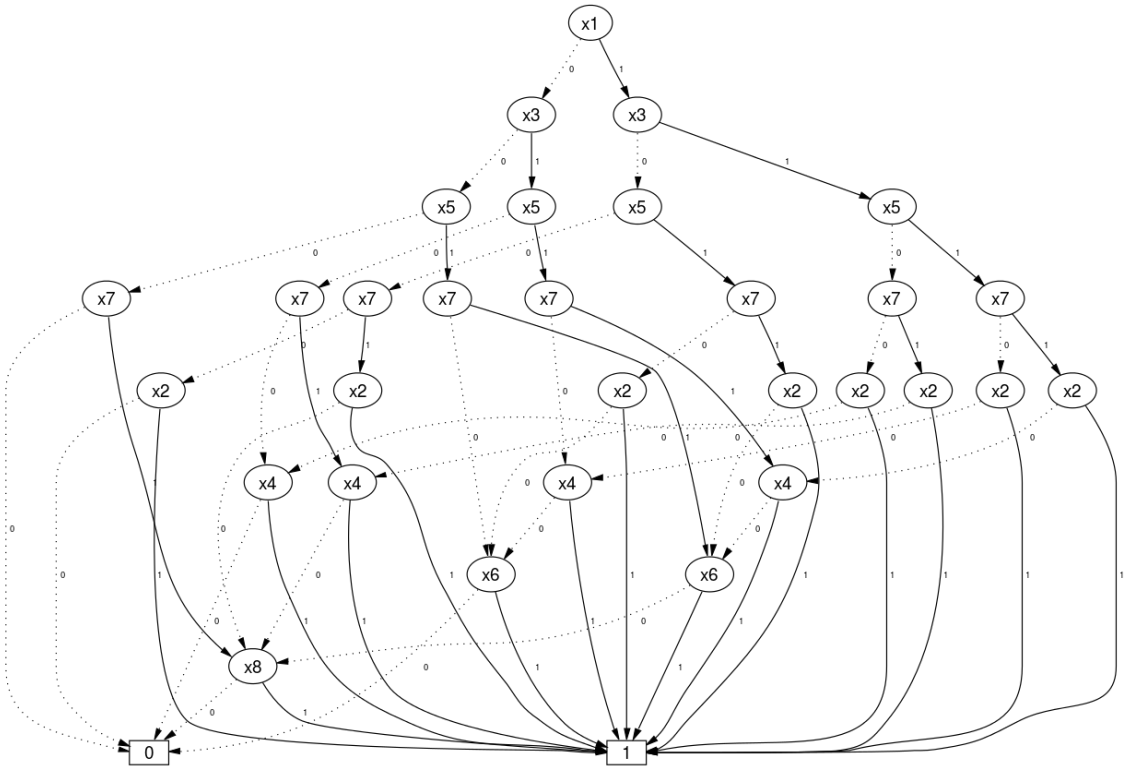
x1	x2	x3	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



BDD (cont')

$$f(x_1, \ldots, x_8) = x_1 x_2 + x_3 x_4 + x_5 x_6 + x_7 x_8$$



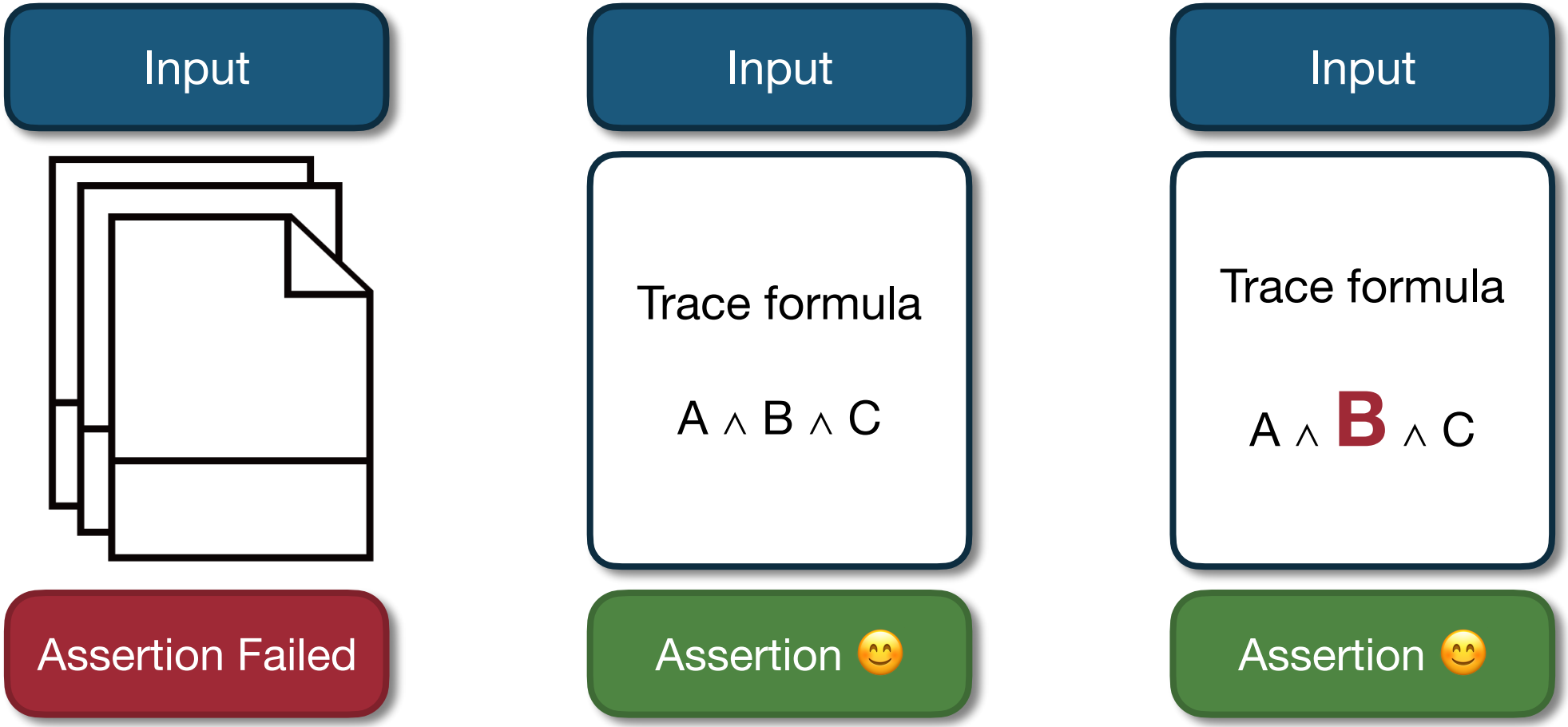


https://en.wikipedia.org/wiki/Binary_decision_diagram

Bounded model checking

- Checks k-depth paths with SAT solver
- Unroll loops
- Gives up some extent of completeness
- Special thanks to SAT researchers!😆
- Each one has its own pros and cons (BDD)

Informal overview



Example

```
 1. int array[3];
 2.
 3. int test_me(int index){
 4.     if ( index ≠ 1 ) // Potential Bug 2
 5.         index = 2;
 6.     else
 7.         index = index + 2; // Potential Bug 1
 8.     ...
 9.     i = index;
10.     return array[i]; // assert(i ⩾ 0 && i<3)
11. }
```

Making trace formula

$$\mathsf{TF} \equiv \mathtt{guard}_1 = (\mathtt{index}_1 \neq 1) \wedge$$
$$\mathtt{index}_2 = 2 \wedge \mathtt{index}_3 = \mathtt{index}_1 + 2 \wedge$$
$$\mathtt{i} = \mathtt{guard}_1 ? \mathtt{index}_2 : \mathtt{index}_3$$

$$\Phi \equiv \underbrace{\mathtt{index}_1 = 1}_{\text{test input}} \wedge \underbrace{\mathsf{TF}}_{\text{trace formula}} \wedge \underbrace{\mathtt{i} < 3}_{\text{assertion}}$$

Making trace formula

$$\mathsf{TF} \equiv \mathtt{guard}_1 = (\mathtt{index}_1 \neq 1) \wedge$$
$$\mathtt{index}_2 = 2 \wedge \mathtt{index}_3 = \mathtt{index}_1 + 2 \wedge$$
$$\mathtt{i} = \mathtt{guard}_1 ? \mathtt{index}_2 : \mathtt{index}_3$$

$$\Phi \equiv \underbrace{\mathtt{index}_1 = 1}_{\text{test input}} \wedge \underbrace{\mathsf{TF}}_{\text{trace formula}} \wedge \underbrace{\mathtt{i} < 3}_{\text{assertion}}$$

Even accepts feedback!

$$\mathsf{TF} \equiv \mathtt{guard}_1 = (\mathtt{index}_1 \neq 1) \wedge$$
$$\mathtt{index}_2 = 2 \wedge \mathtt{index}_3 = \mathtt{index}_1 + 2 \wedge$$
$$\mathtt{i} = \mathtt{guard}_1 ? \mathtt{index}_2 : \mathtt{index}_3$$

$$\Phi \equiv \underbrace{\mathtt{index}_1 = 1}_{\text{test input}} \wedge \underbrace{\mathsf{TF}}_{\text{trace formula}} \wedge \underbrace{\mathtt{i} < 3}_{\text{assertion}}$$

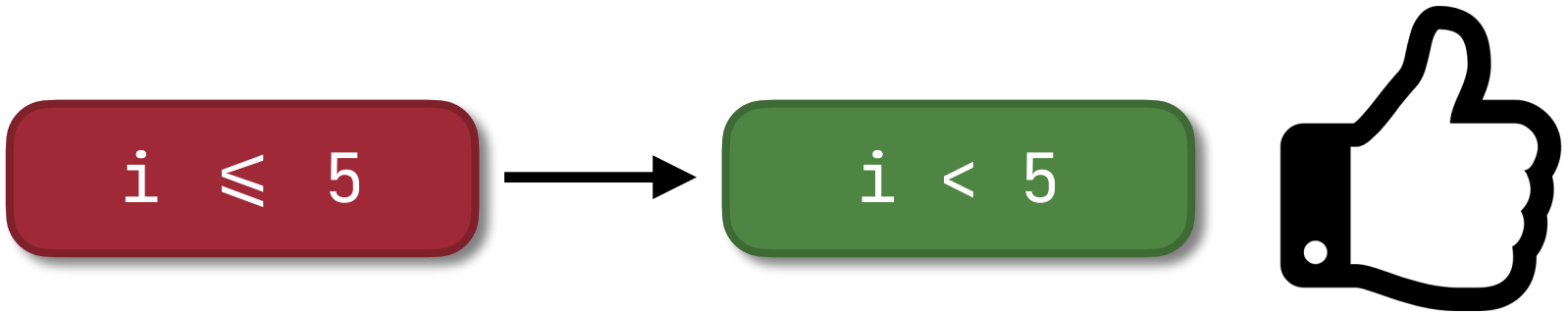
Loop unrolling

```
1   while(cond){
2       stmt;
3   }
4
5   if(cond){
6       stmt;
7       if(cond){
8           stmt;
9           if(cond){
10              ...
11          }
12      }
13  }
```

But limited steps!

One more step!

```
1. int array[5];
2.
3. for (int i=0; i⩽5; i++) { array[i]; }
```



Running BugAssist

	Program	LOC#	Proc#	Reduc	assign#		var#		clause#		Fault#	time
					Before	After	Before	After	Before	After		
1	totinfo	565	7	S	734	21	0.797m	400	1.822m	1225	2	0.19s
2	print_tokens	726	18	C	65698	239	5.507m	7439	53.483m	22634	13	25s
3	schedule	564	21	DS	5914	391	5.173m	0.053m	15.379m	0.142m	13	28s
4	schedule	564	21	DS	41942	5412	78.982m	4.517m	239.385m	13.788m	25	11h
5	totinfo	565	7	CS	865	454	0.862m	0.734m	4.156m	3.728m	3	225s
6	schedule2	374	16	S	398	275	0.021m	0.015m	0.062m	0.048m	9	20s

Discussions

- Performance is much better than traditional way
- Limited scalability
- Can't detect omission fault
- Can be applied in different granularity

Questions?