Traces, Interpolants and Automata

Arijit Shaw Kush Grover

Chennai Mathematical Institute

- Trace Abstraction
- We implemented the paper[1] and tried to understand the approach.
- We'll describe the implementation along with the verification algorithm.

[1] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Software model checking for people who love automata. In International Conference on Computer Aided Verification, pages 36–52. Springer, 2013.

Flow of Algorithm

Program CFG as Automaton

Take an accepting trace

Generate interpolants from the trace

Check for Inclusion

Reconstruct the Program Automaton

Program CFG as Automaton

Take an accepting trace

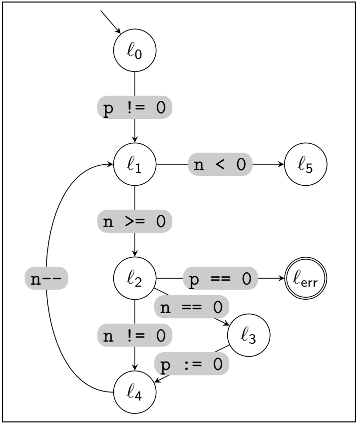
Generate interpolants from the trace

Check for Inclusion

Reconstruct the Program Automaton

ℓ_0:	`assume p != 0;`
ℓ_1:	`while(n >= 0)`
	`{`
ℓ_2:	`    assert p != 0;`
	`    if(n == 0)`
	`    {`
ℓ_3:	`        p := 0;`
	`    }`
ℓ_4:	`    n--;`
	`}`

pseudocode



control flow graph

Program CFG as Automaton

Take an accepting trace

Generate interpolants from the trace

Check for Inclusion

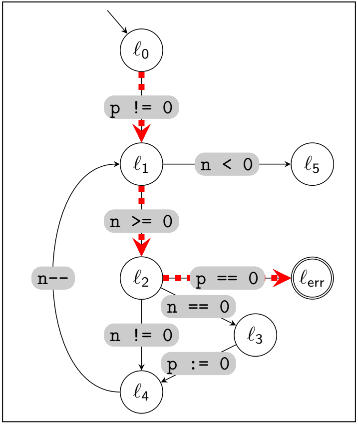
Reconstruct the Program Automaton

```
ℓ0:  assume p != 0;

ℓ1:  while(n >= 0)
     {
ℓ2:      assert p != 0;

         if(n == 0)
         {
ℓ3:          p := 0;
         }
ℓ4:      n--;
     }
```

pseudocode



control flow graph

Program CFG as Automaton

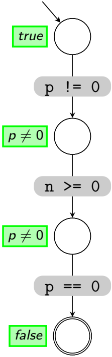
Take an accepting trace

Generate interpolants from the trace

Check for Inclusion

Reconstruct the Program Automaton

- Step 1: Analyze correctness
- Step 2: Construct proof
    - Naive approach: symbolic execution
    - Alternatives:
        - symbolic execution + unsat cores
        - Craig interpolation



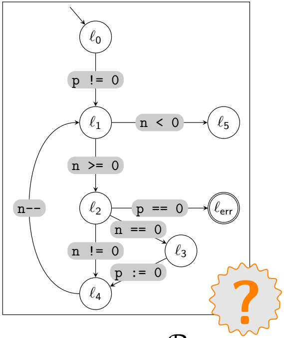
Program CFG as Automaton

Take an accepting trace

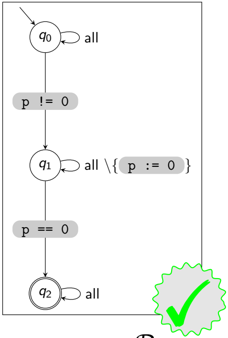
Generate interpolants from the trace

Check for Inclusion

Reconstruct the Program Automaton

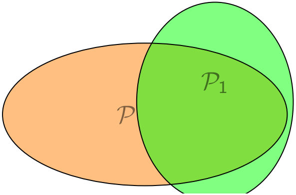


program $\mathcal{P}$



program $\mathcal{P}_1$

Consider only traces in set theoretic difference $\mathcal{P} \setminus \mathcal{P}_1$.



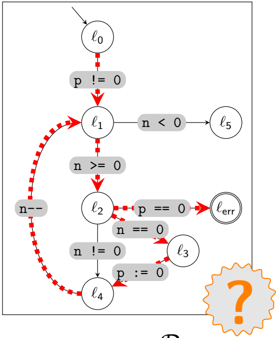
Program CFG as Automaton

Take an accepting trace

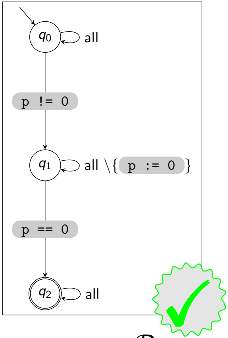
Generate interpolants from the trace

Check for Inclusion

Reconstruct the Program Automaton

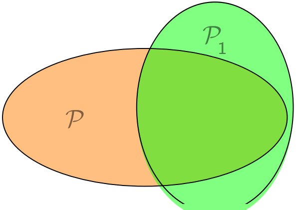


program $\mathcal{P}$

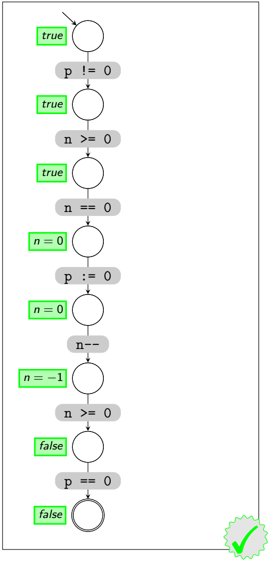


program $\mathcal{P}_1$

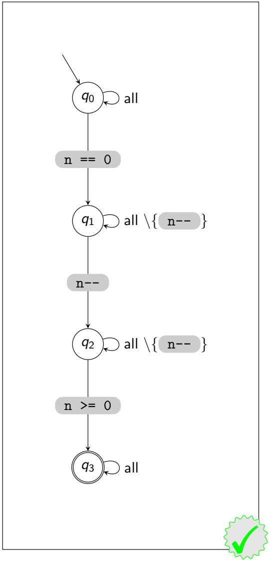
Consider only traces in set theoretic difference $\mathcal{P} \setminus \mathcal{P}_1$.

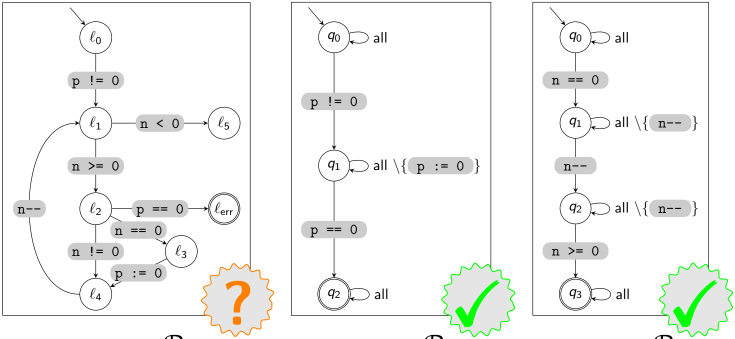


1. take trace π_2
2. consider trace as program $\mathcal{P}_2$
3. analyze correctness of $\mathcal{P}_2$



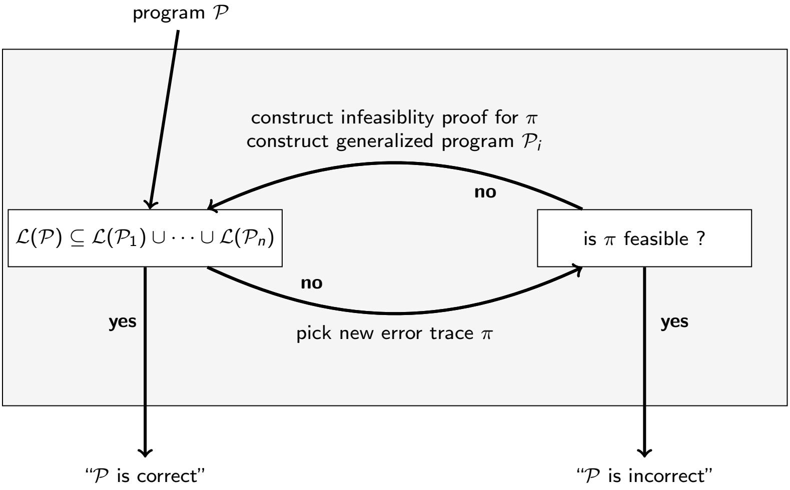
1. take trace π_2
2. consider trace as program $\mathcal{P}_2$
3. analyze correctness or $\mathcal{P}_2$
4. generalize program $\mathcal{P}_2$
    - add transitions
    - merge locations





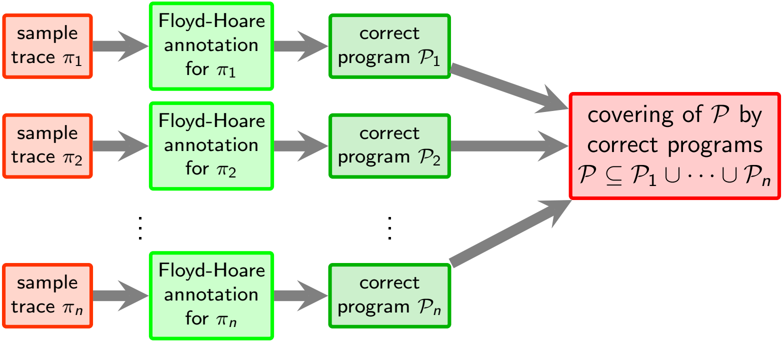
program $\mathcal{P}$ program $\mathcal{P}_1$ program $\mathcal{P}_2$

$$\mathcal{P} \quad \subseteq \quad \mathcal{P}_1 \cup \mathcal{P}_2$$


program $\mathcal{P}$
construct infeasiblity proof for π
construct generalized program $\mathcal{P}_i$
no
$\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\mathcal{P}_1) \cup \cdots \cup \mathcal{L}(\mathcal{P}_n)$
is π feasible ?
no
yes
pick new error trace π
yes
"$\mathcal{P}$ is correct"
"$\mathcal{P}$ is incorrect"

A classical approach to software model checking:

Our approach to software model checking:



Implementation

ULTIMATE Automata Library for trace generation.

- File interface
- `ats` file format.

ULTIMATE > Automata Library > Automata_script   </>   execute

```
1  FiniteAutomaton nfa1 = (
2      alphabet = {A B},
3      states = {q0 q1 q2},
4      initialStates = {q0},
5      finalStates = {q2},
6      transitions = {
7          (q0 A q0)
8          (q0 A q1)
9          (q1 A q2)
10         (q1 B q1)
11         (q2 A q1)
12     }
13 );
14 print(getAcceptedWord(nfa1));
15 FiniteAutomaton nfa2 = (
16     alphabet = {A B},
17     states = {q0 q1 q2},
18     initialStates = {q0},
19     finalStates = {q0},
20     transitions = {
21         (q0 A q1)
22         (q0 B q1)
23         (q0 B q2)
24         (q1 A q1)
25         (q1 A q2)
26         (q2 B q0)
27     }
28 );
29 FiniteAutomaton intersecNfa = intersect(nfa1, nfa2);
30
```

ULTIMATE Results

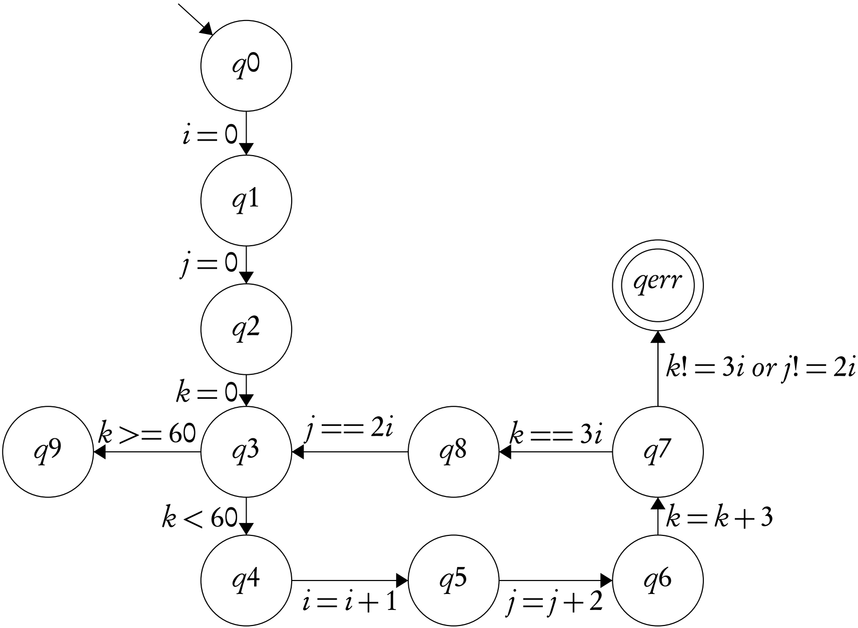
Line	Column	Description
14	1 - 28	print(getAcceptedWord(nfa1)) "A" "A"
31	1 - 27	print(isEmpty(intersecNfa)) true
-	-	Finished interpretation of automata script. You have not used any assert statement in your automata script. Assert statements can be used to check Boolean results.

```c
int main()
{
  unsigned int i = 0;
  unsigned int j = 0;
  unsigned int k = 0;

  while (k < 60) {
    i = i + 1;
    j = j + 2;
    k = k + 3;
    //@ assert((k == 3*i) && (j ==
    2*i));
  }

}
```

```
FiniteAutomaton nfa_0 = (
    alphabet = {"i = 0" "j = 0" "k = 0" "k < 16" "i
= i + 1" "j = j + 2" "k = k + 3" "k != 3 * i" "j !=
2 * i"},
    states = {"q0" "q1" "q2" "q3" "q4" "q5" "q6"
"q7" "qerr"},
    initialStates = {"q0"},
    finalStates = {"qerr"},
    transitions = {
        ("q0" "i = 0" "q1")
        ("q1" "j = 0" "q2")
        ("q2" "k = 0" "q3")
        ("q3" "k < 16" "q4")
        ("q4" "i = i + 1" "q5")
        ("q5" "j = j + 2" "q6")
        ("q6" "k = k + 3" "q3")
        ("q3" "k != 3 * i" "qerr")
        ("q3" "j != 2 * i" "qerr")
    }
);
```


q0
i = 0
q1
j = 0
q2
k = 0
q3
k >= 60
q9
k < 60
q4
i = i + 1
q5
j = j + 2
q6
k = k + 3
q7
k == 3i
q8
j == 2i
k! = 3i or j! = 2i
qerr

Implementation

ULTIMATE Automata Library for trace generation.

- File interface
- `ats` file format.

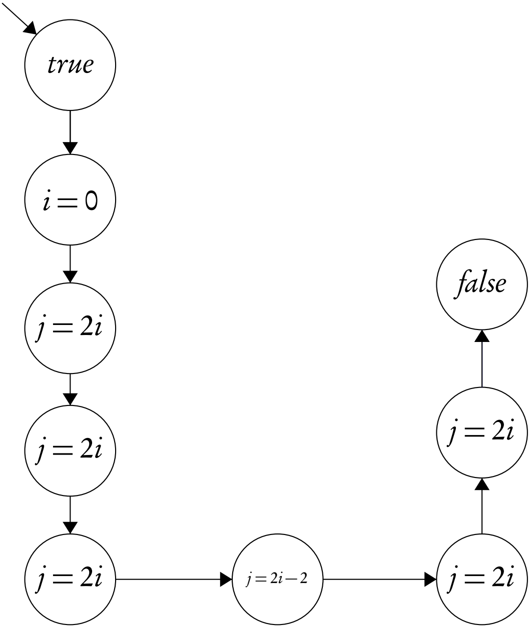
MathSAT for getting interpolants.

- Python API.

```
arijit@waterfall:~/verification/ApnaAutomizer$ time python runner.py -i simple_vardep.ats


[DEBUG] iteration count = 0
[DEBUG] Trace generated : ['i = 0', 'j = 0', 'k = 0', 'k < 300', 'i = i + 1', 'j = j + 2', 'k = k + 3', 'k != 3 * i']
[DEBUG] Found interpolant 0 : true
[DEBUG] Found interpolant 1 : (and (<= i_1 0) (<= 0 i_1))
[DEBUG] Found interpolant 2 : (and (<= i_1 0) (<= 0 i_1))
[DEBUG] Found interpolant 3 : (and (<= (+ (* 3 i_1) (* (- 1) k_1)) 0) (<= 0 (+ (* 3 i_1) (* (- 1) k_1))))
[DEBUG] Found interpolant 4 : (and (<= (+ (* 3 i_1) (* (- 1) k_1)) 0) (<= 0 (+ (* 3 i_1) (* (- 1) k_1))))
[DEBUG] Found interpolant 5 : (and (<= (- 3) (+ k_1 (* (- 3) i_2))) (<= (+ k_1 (* (- 3) i_2)) (- 3)))
[DEBUG] Found interpolant 6 : (and (<= (- 3) (+ k_1 (* (- 3) i_2))) (<= (+ k_1 (* (- 3) i_2)) (- 3)))
[DEBUG] Found interpolant 7 : (and (<= (+ (* 3 i_2) (* (- 1) k_2)) 0) (<= 0 (+ (* 3 i_2) (* (- 1) k_2))))
[DEBUG] Found interpolant 8 : false


[DEBUG] iteration count = 1
[DEBUG] Trace generated : ['i = 0', 'j = 0', 'k = 0', 'k < 300', 'i = i + 1', 'j = j + 2', 'k = k + 3', 'k == 3 * i', 'k < 300',
 'i = i + 1', 'j = j + 2', 'k = k + 3', 'k != 3 * i']
[DEBUG] Found interpolant 0 : true
[DEBUG] Found interpolant 1 : (and (<= i_1 0) (<= 0 i_1))
[DEBUG] Found interpolant 2 : (and (<= i_1 0) (<= 0 i_1))
[DEBUG] Found interpolant 3 : (and (<= (+ (* 3 i_1) (* (- 1) k_1)) 0) (<= 0 (+ (* 3 i_1) (* (- 1) k_1))))
[DEBUG] Found interpolant 4 : (and (<= (+ (* 3 i_1) (* (- 1) k_1)) 0) (<= 0 (+ (* 3 i_1) (* (- 1) k_1))))
[DEBUG] Found interpolant 5 : (and (<= (- 3) (+ k_1 (* (- 3) i_2))) (<= (+ k_1 (* (- 3) i_2)) (- 3)))
[DEBUG] Found interpolant 6 : (and (<= (- 3) (+ k_1 (* (- 3) i_2))) (<= (+ k_1 (* (- 3) i_2)) (- 3)))
[DEBUG] Found interpolant 7 : (and (<= (+ (* 3 i_2) (* (- 1) k_2)) 0) (<= 0 (+ (* 3 i_2) (* (- 1) k_2))))
[DEBUG] Found interpolant 8 : (and (<= (+ (* 3 i_2) (* (- 1) k_2)) 0) (<= 0 (+ (* 3 i_2) (* (- 1) k_2))))
[DEBUG] Found interpolant 9 : (and (<= (+ (* 3 i_2) (* (- 1) k_2)) 0) (<= 0 (+ (* 3 i_2) (* (- 1) k_2))))
[DEBUG] Found interpolant 10 : (and (<= (- 3) (+ k_2 (* (- 3) i_3))) (<= (+ k_2 (* (- 3) i_3)) (- 3)))
[DEBUG] Found interpolant 11 : (and (<= (- 3) (+ k_2 (* (- 3) i_3))) (<= (+ k_2 (* (- 3) i_3)) (- 3)))
[DEBUG] Found interpolant 12 : (and (<= (+ (* 3 i_3) (* (- 1) k_3)) 0) (<= 0 (+ (* 3 i_3) (* (- 1) k_3))))
[DEBUG] Found interpolant 13 : false
```


true
i = 0
j = 2i
j = 2i
j = 2i
j = 2i − 2
j = 2i
j = 2i
false

Implementation

ULTIMATE Automata Library for trace generation.

- File interface
- `ats` file format.

MathSAT for getting interpolants.

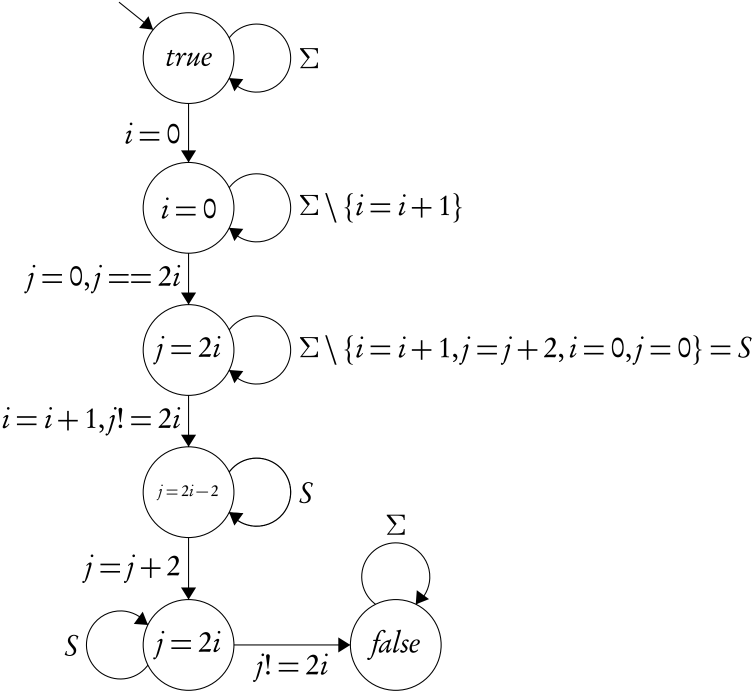
- Python API.

Generating Floyd-Hoare Automaton.

- MathSAT API.
- Gets generated as `ats` file.

Algorithm 1 Floyd Hoare Automaton Generation

```
I ← List all distinct interpolants
Q ← add states q_i corresponding to I_i in I
for two consecutive interpolants I_i and I_{i+1} do
    for each statement st in alphabet do
        if {I_i}st{I_{i+1}} is valid hoare triple then
            add transition (q_i, st, q_{i+1}) in automaton
        end if
    end for
end for
```


true
Σ
i = 0
i = 0
Σ \ {i = i + 1}
j = 0, j == 2i
j = 2i
Σ \ {i = i + 1, j = j + 2, i = 0, j = 0} = S
i = i + 1, j! = 2i
j = 2i − 2
S
j = j + 2
Σ
S
j = 2i
j! = 2i
false

```
FiniteAutomaton fha_1 = (
    alphabet = {"k < 300" "j != 2 * i" "j = 0" "k = k + 3" "i = i + 1" "k == 3 * i" "k != 3 * i" "i = 0" "k = 0" "j = j + 2" },
    states = {q0 q1 q2 q3 q4 q5 },
    initialStates = {q0},
    finalStates = {q5},
    transitions = {
        (q0 "i = 0" q0)
        (q0 "j = 0" q0)
        (q0 "k = 0" q0)
        (q0 "k < 300" q0)
        (q0 "i = i + 1" q0)
        (q0 "j = j + 2" q0)
        (q0 "k = k + 3" q0)
        (q0 "k != 3 * i" q0)
        (q0 "j != 2 * i" q0)
        (q0 "k == 3 * i" q0)
        (q1 "i = 0" q1)
        (q0 "i = 0" q1)
        (q1 "j = 0" q1)
        (q1 "k = 0" q1)
        (q1 "k < 300" q1)
        (q1 "j = j + 2" q1)
        (q1 "k = k + 3" q1)
        (q1 "k != 3 * i" q1)
        (q1 "j != 2 * i" q1)
        (q1 "k == 3 * i" q1)
        (q2 "j = 0" q2)
        (q1 "k = 0" q2)
        (q2 "k < 300" q2)
        (q2 "j = j + 2" q2)
        (q2 "k != 3 * i" q2)
        (q2 "j != 2 * i" q2)
        (q2 "k == 3 * i" q2)
        (q1 "k == 3 * i" q2)
        (q3 "j = 0" q3)
        (q3 "k < 300" q3)
        (q2 "i = i + 1" q3)
        (q3 "j = j + 2" q3)
        (q3 "k != 3 * i" q3)
        (q2 "k != 3 * i" q3)
        (q3 "j != 2 * i" q3)
```

Implementation

ULTIMATE Automata Library for trace generation.

- File interface
- `ats` file format.

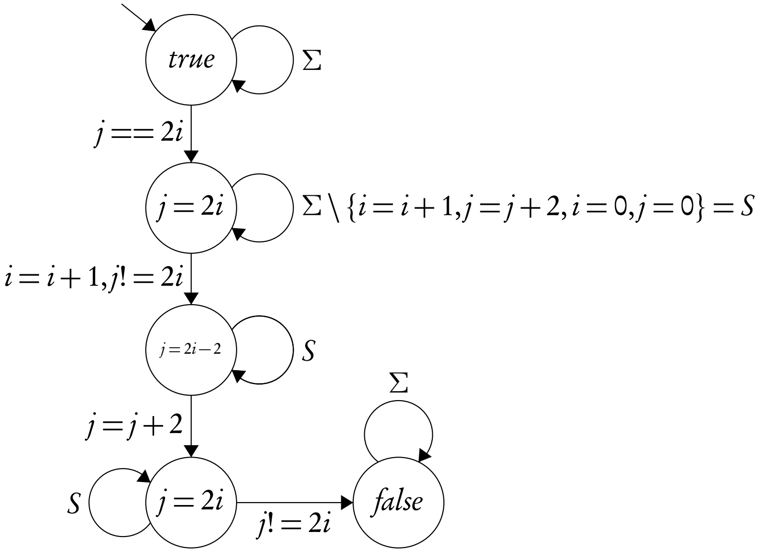
MathSAT for getting interpolants.

- Python API.

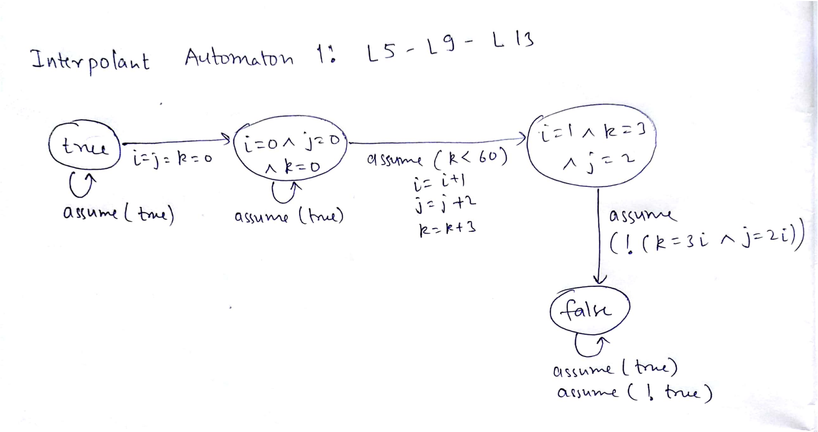
Generating Floyd-Hoare Automaton.

- MathSAT API.
- Gets generated as `ats` file.

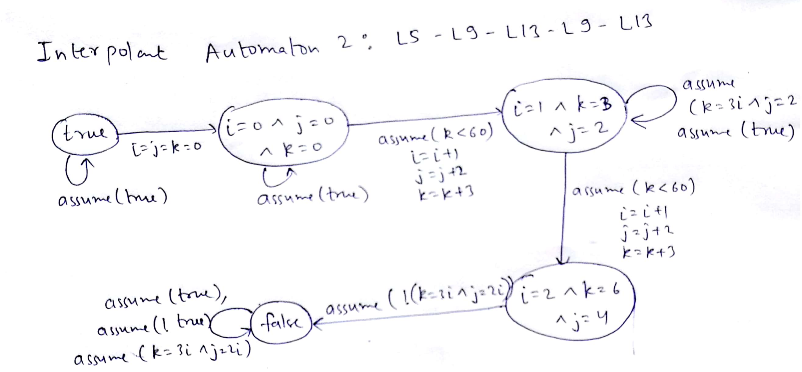
Automata Library for reconstructing Program Automaton.


true
Σ
j == 2i
j = 2i
Σ \ {i = i + 1, j = j + 2, i = 0, j = 0} = S
i = i + 1, j! = 2i
j = 2i − 2
S
j = j + 2
Σ
S
j = 2i
false
j! = 2i

Interpolant Automaton by Ultimate



Interpolant Automaton by Ultimate



Results and Comparison with Ultimate

Testcode	loop	Iterations		Runtime	
		AA	UA	AA	UA
bhrm.c	5	7	37	111s	37s
nested_loop.c	3x3	8	10	124s	15s
simple_vardep.c	20	10	20	69s	40s
simple_vardep.c	100	10	100	69s	1678s

Limitations

- Works only for Linear Integer Arithmetic.
- Input must be given in specified format.
- Program must not contain function calls.

Limitations

- Works only for Linear Integer Arithmetic.
- Input must be given in specified format.
- Program must not contain function calls.

Bottlenecks

- Time taken by Automata Library is too much.
- Time taken to check validity of Hoare triples. For smaller programs too, we are checking a large number of Hoare triples.

Limitations

- Works only for Linear Integer Arithmetic.
- Input must be given in specified format.
- Program must not contain function calls.

Bottlenecks

- Time taken by Automata Library is too much.
- Time taken to check validity of Hoare triples. For smaller programs too, we are checking a large number of Hoare triples.

Optimization

- Choose hoare triples to check smartly.
- Manually inserting invariants.

- Thank you.
- Some of the slides are taken from Matthias' Presentation at EPIT 2018. Thanks to him too.