INFORMATION REUSE FOR MULTI-GOAL REACHABILITY ANALYSES

Dirk Beyer, *Andreas Holzer*, Michael Tautschnig, and Helmut Veith

Motivation

- Test generation with model checkers
- Model checker: Does a program satisfy a temporal specification?
- If not, produce counterexample: program trace

Motivation

- Test generation with model checkers
- Model checker: Does a program satisfy a temporal specification?
- If not, produce counterexample: program trace
- Encode negation of test goal as specification
  - Counterexample: Test case
  - Derive inputs from counterexample
  - No Counterexample: No test case exists!

White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

Decision Coverage

White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

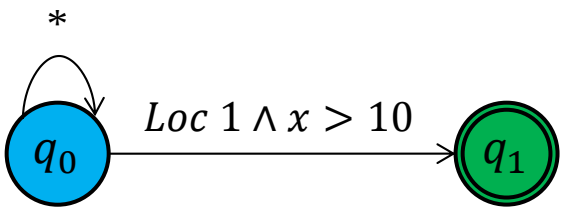
Decision Coverage

White-Box Testing

Program

```
1  if (x > 10)
2      f1 = false;
3  else
4      f1 = true;
5  if (x == 100)
6      f2 = false;
7  if (f1)
8      s = f2;
9  else
10     s = f1;
```

Decision Coverage

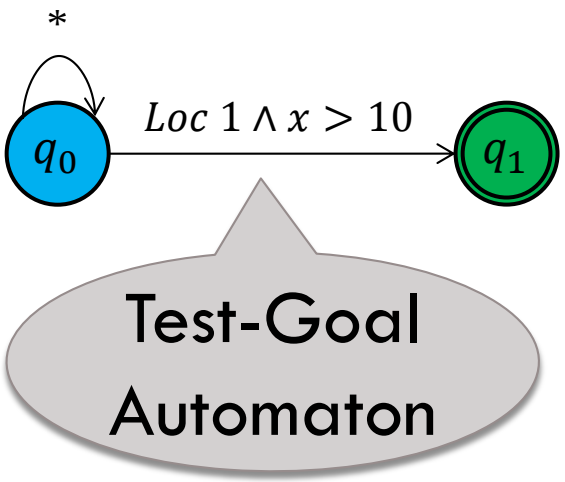


White-Box Testing

Program

```
1  if (x > 10)
2      f1 = false;
3  else
4      f1 = true;
5  if (x == 100)
6      f2 = false;
7  if (f1)
8      s = f2;
9  else
10     s = f1;
```

Decision Coverage

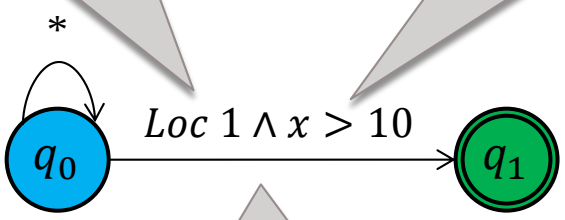


White-Box Testing

Program Location

State Predicate

```
1  if (x > 10)
2      f1 = false;
3  else
4      f1 = true;
5  if (x == 100)
6      f2 = false;
7  if (f1)
8      s = f2;
9  else
10     s = f1;
```



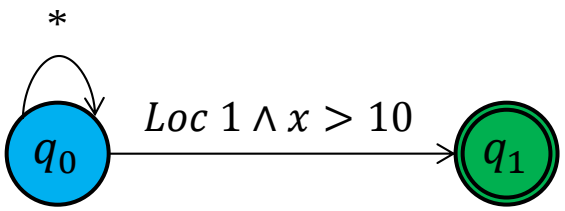
Test-Goal Automaton

White-Box Testing

Program

```
1  if (x > 10)
2      f1 = false;
3  else
4      f1 = true;
5  if (x == 100)
6      f2 = false;
7  if (f1)
8      s = f2;
9  else
10     s = f1;
```

Decision Coverage

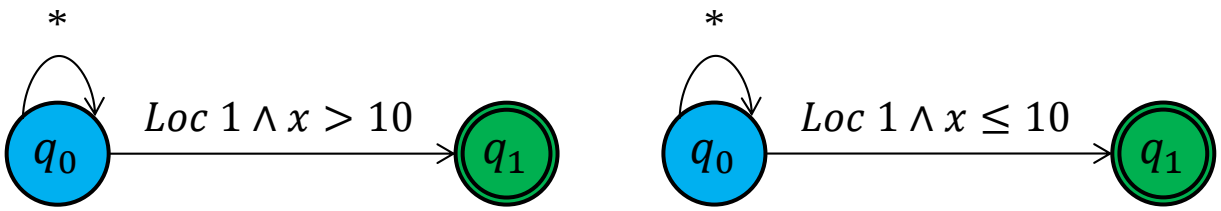


White-Box Testing

Program

```
1  if (x > 10)
2      f1 = false;
3  else
4      f1 = true;
5  if (x == 100)
6      f2 = false;
7  if (f1)
8      s = f2;
9  else
10     s = f1;
```

Decision Coverage

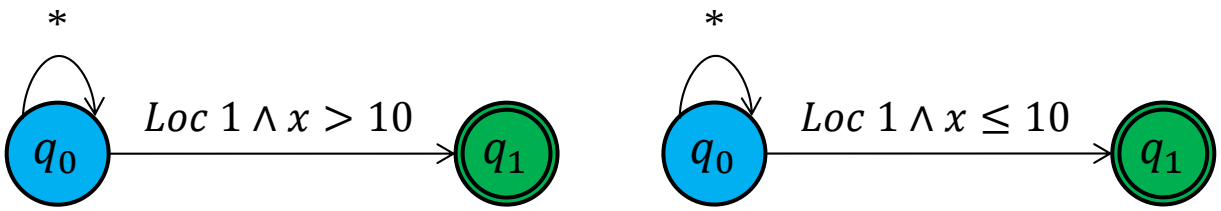


White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

Decision Coverage

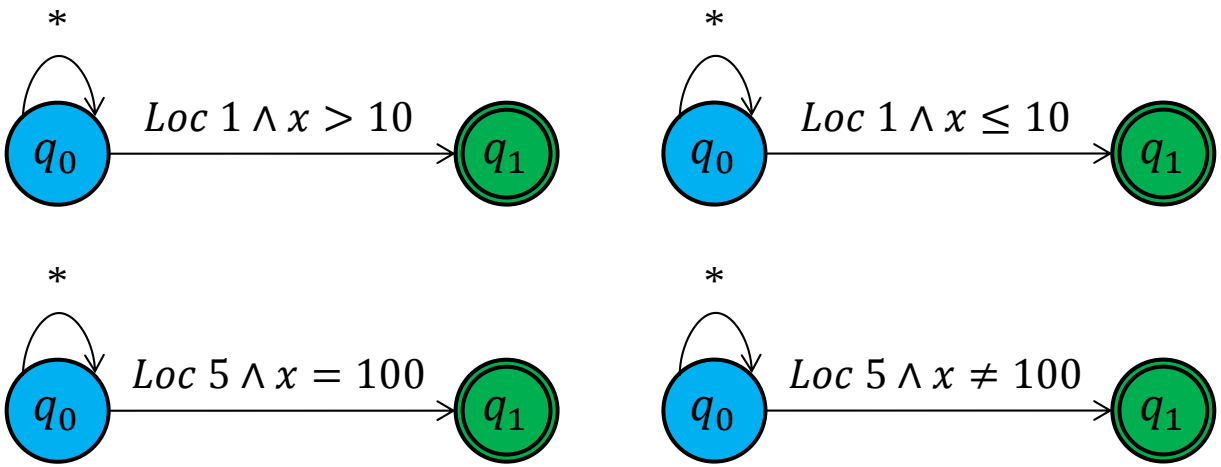


White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

Decision Coverage

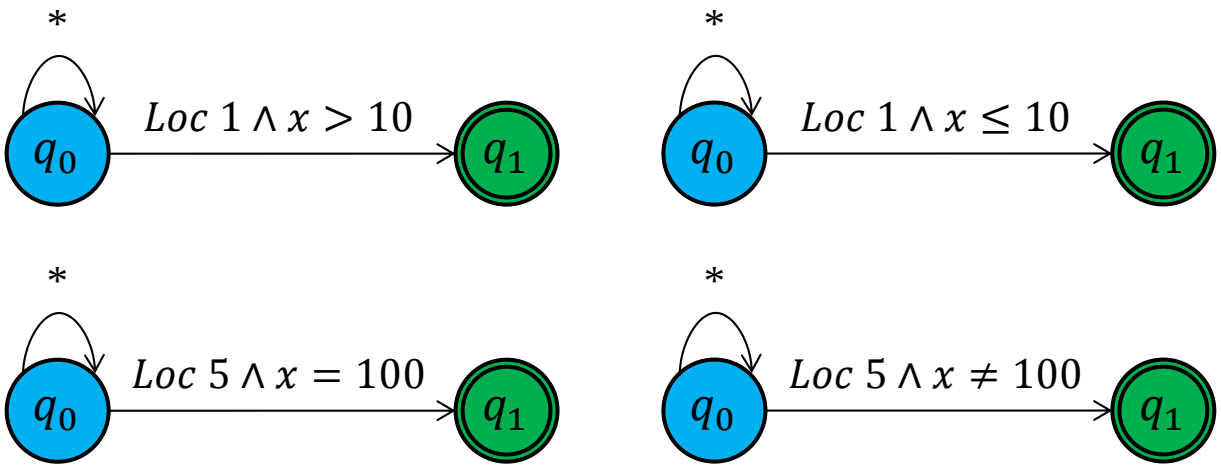


White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

Decision Coverage

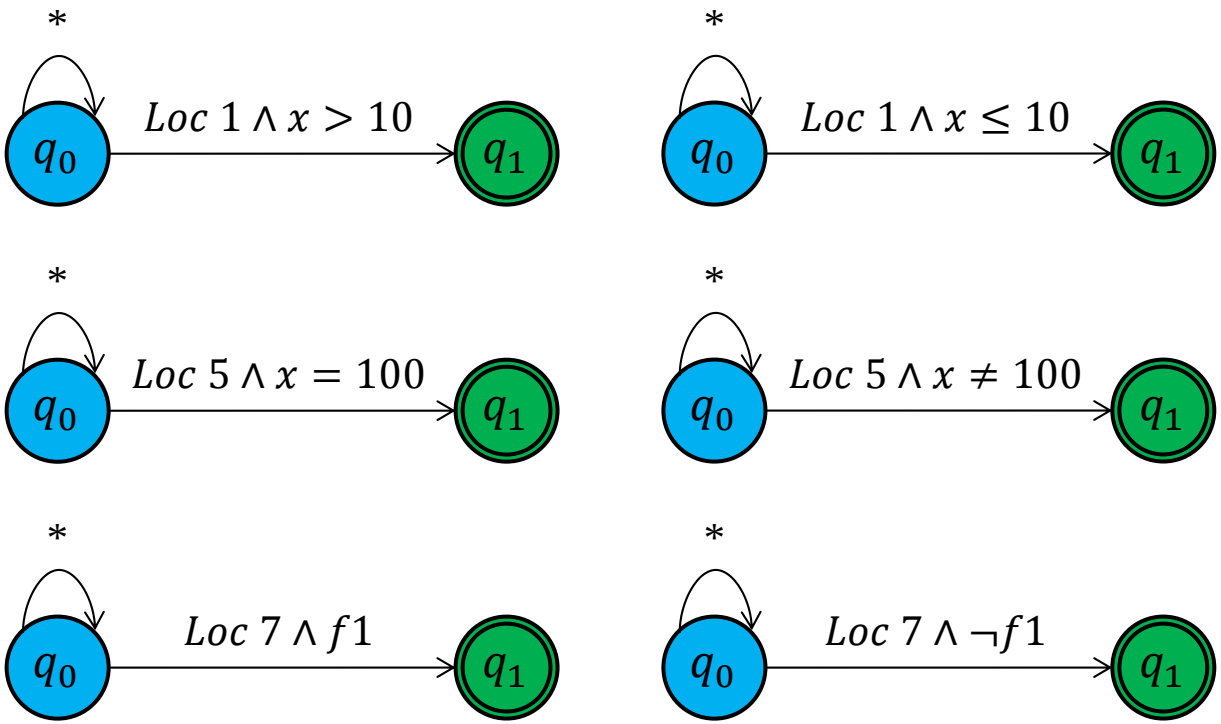


White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

Decision Coverage



White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

White-Box Testing

Program $(Basic\ Block)^2$ **Coverage**

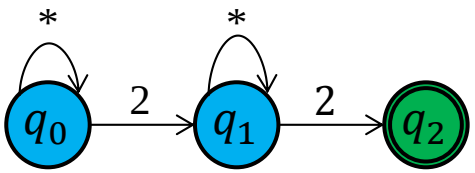
```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

$(Basic\ Block)^2$ Coverage

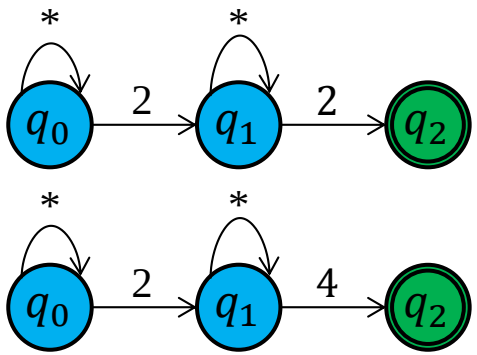


White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

$(Basic\ Block)^2$ Coverage

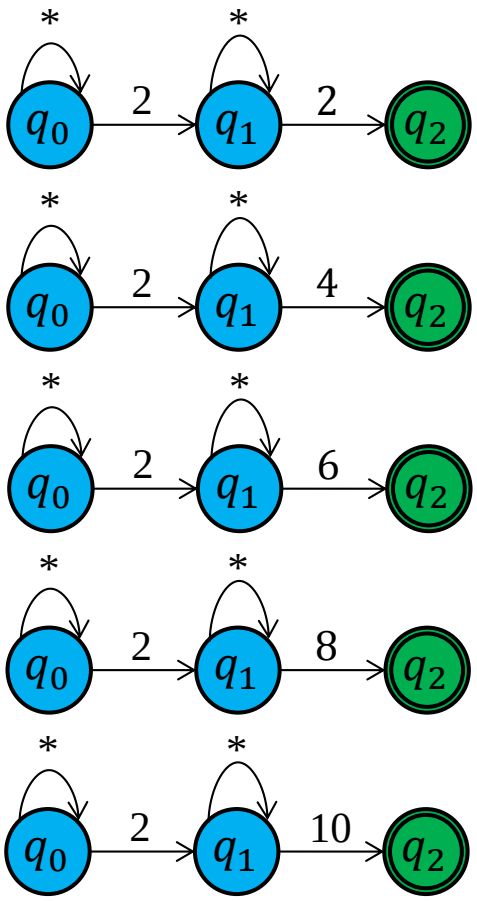


White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

$(Basic\ Block)^2$ Coverage

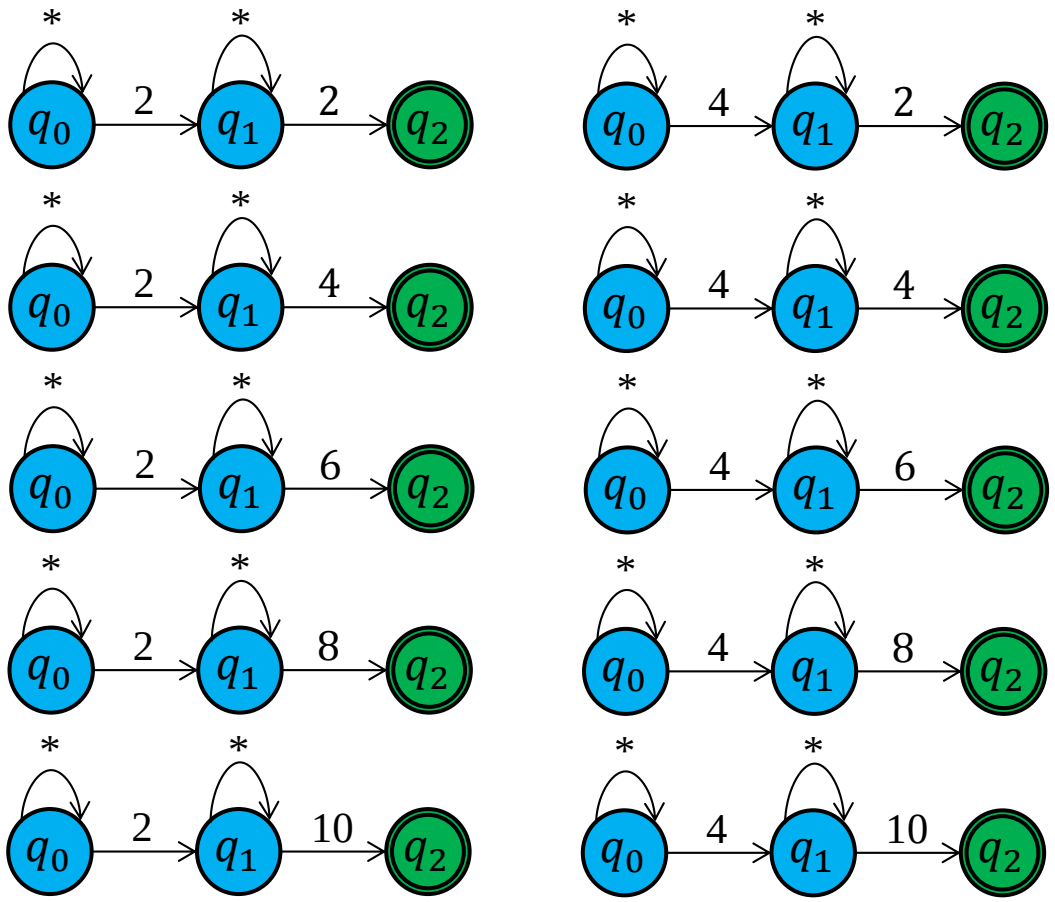


White-Box Testing

Program

```
1  if (x > 10)
2      f1 = false;
3  else
4      f1 = true;
5  if (x == 100)
6      f2 = false;
7  if (f1)
8      s = f2;
9  else
10     s = f1;
```

$(Basic\ Block)^2$ Coverage

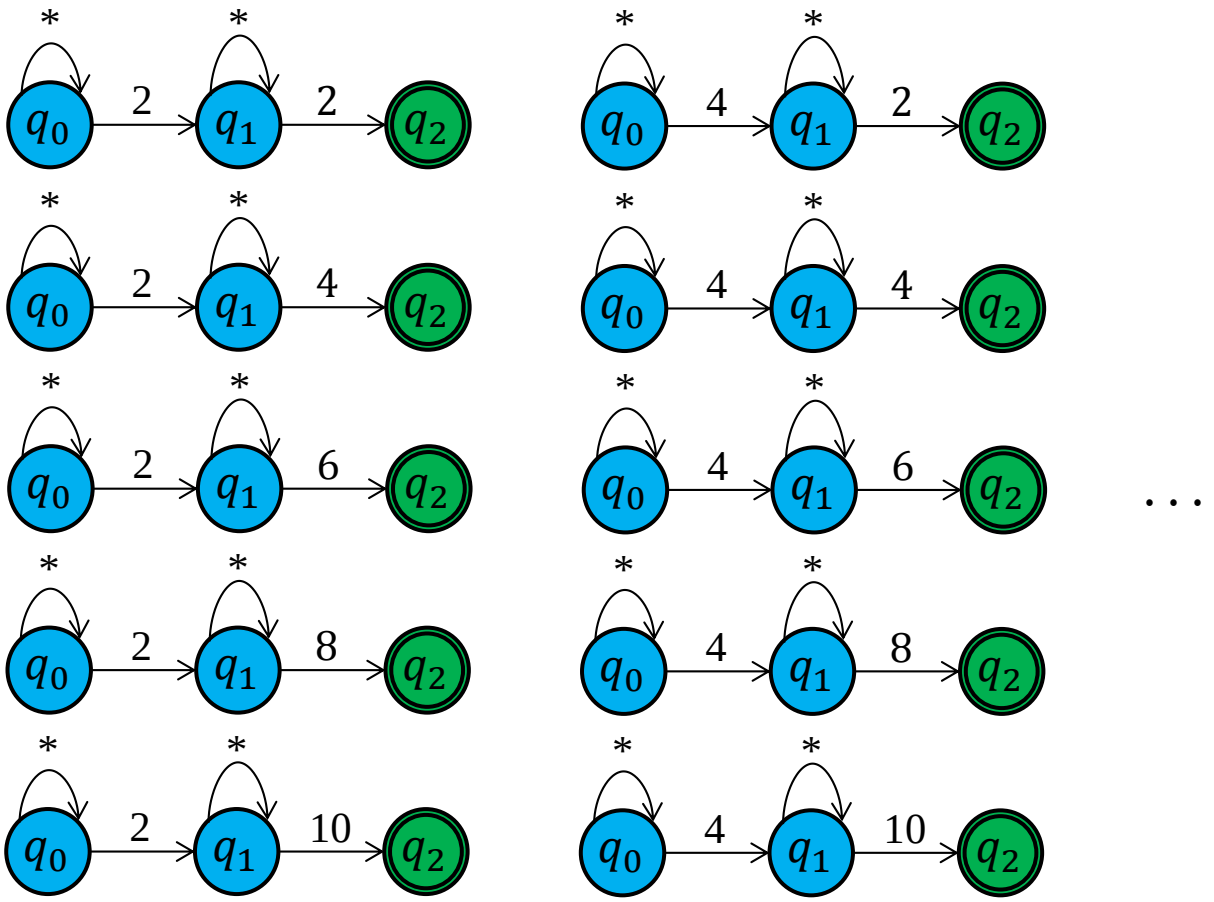


White-Box Testing

Program

```
1  if (x > 10)
2      f1 = false;
3  else
4      f1 = true;
5  if (x == 100)
6      f2 = false;
7  if (f1)
8      s = f2;
9  else
10     s = f1;
```

$(Basic\ Block)^2$ Coverage

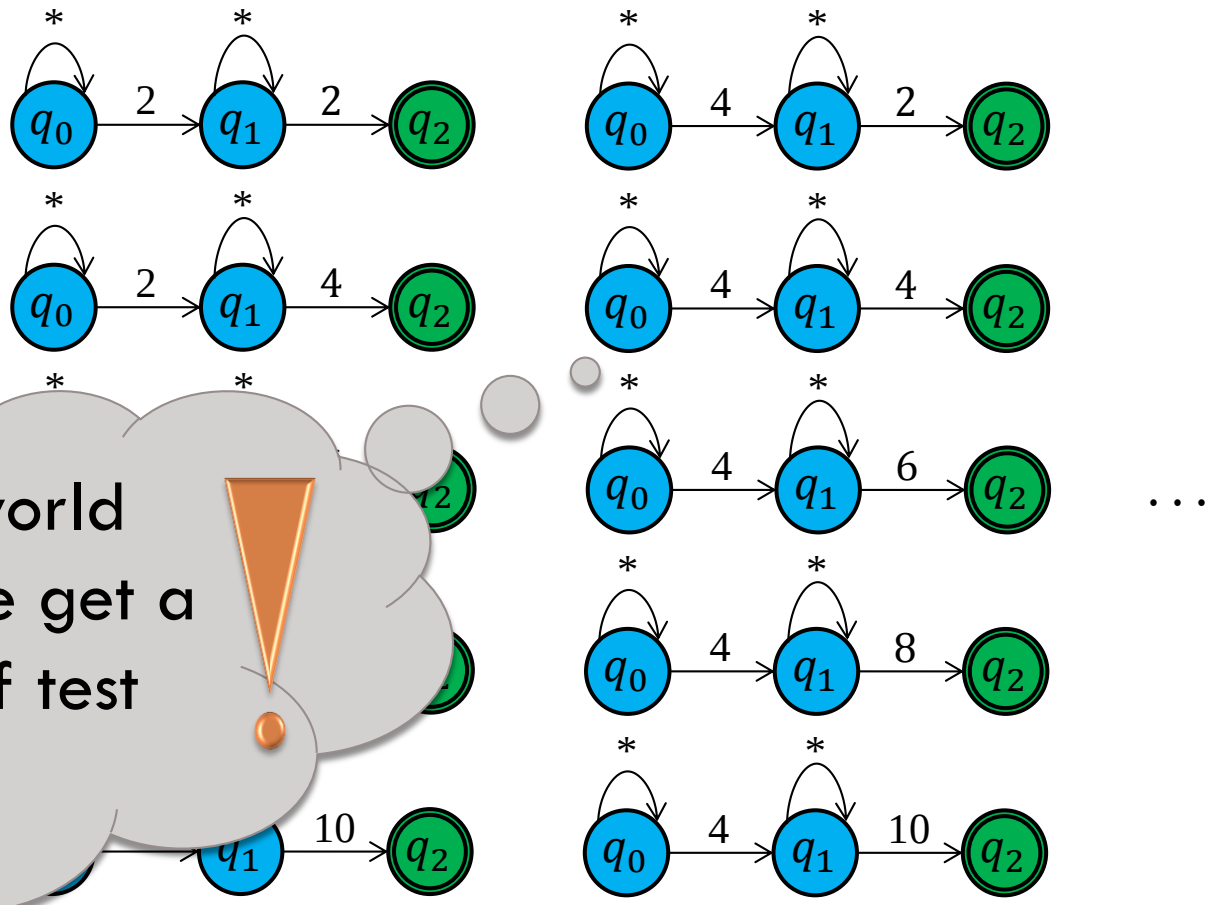


White-Box Testing

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = fal
```

$(Basic\ Block)^2$ Coverage

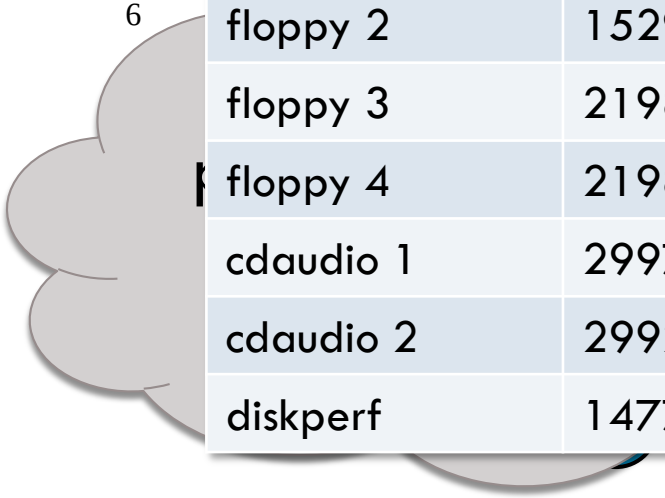


For real-world programs we get a huge set of test goals!

White-Box Testing

Program $(Basic\ Block)^2$ Coverage

Source	Lines of Code	#Goals BB	#Goals BB^2
kbfilter 1	771	118	13924
kbfilter 2	1352	203	41209
kbfilter 3	1349	202	40804
floppy 1	1510	209	43681
floppy 2	1529	209	43681
floppy 3	2198	291	84681
floppy 4	2198	291	84681
cdaudio 1	2997	420	176400
cdaudio 2	2992	417	173889
diskperf	1477	202	40804



. . .

White-Box Testing

- **F**Shell **Q**uery **L**anguage (**FQL**) [VMCAI'09, ASE'10, HVC'10, FASE'11]
  - Specification Language for Coverage Criteria
  - Decision Coverage:

```
cover @DECISIONEDGE
```

  - $Basic\ Block$ Coverage:

```
cover @BASICBLOCKENTRY
```

  - $(Basic\ Block)^2$ Coverage:

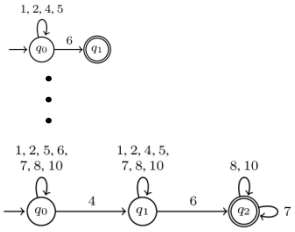
```
cover @BASICBLOCKENTRY -> @BASICBLOCKENTRY
```

  - Coverage Criterion gets instantiated to a set of Test Goals

Multi-Goal Reachability Analysis

Test-Goal Automata

$A_1, A_2, \ldots, A_n$

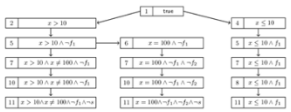


Automaton-Guided Reachability Analysis

Program

```
1  if (x > 10)
2    f1 = false;
3  else
4    f1 = true;
5  if (x == 100)
6    f2 = false;
7  if (f1)
8    s = f2;
9  else
10   s = f1;
```

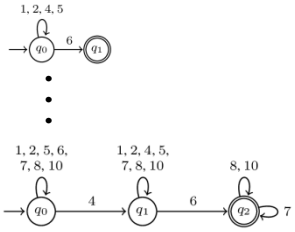
Witnesses of Un-/Satisfiability



Multi-Goal Reachability Analysis

Test-Goal Automata

$A_1, A_2, \ldots, A_n$

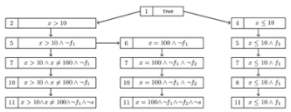


Automaton-Guided Reachability Analysis

Program

```
1  if (x > 10)
2    f1 = false;
3  else
4    f1 = true;
5  if (x == 100)
6    f2 = false;
7  if (f1)
8    s = f2;
9  else
10   s = f1;
```

Witnesses of Un-/Satisfiability



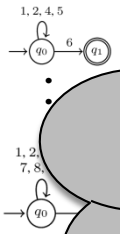
Applications:

- Whitebox/Model-Based Testing
- Reachability-Oracle for Path-Insensitive Program Analyses
- Program Exploration/Debugging

Multi-Goal Reachability Analysis

Test-Goal Automata

$A_1, A_2, \ldots, A_n$



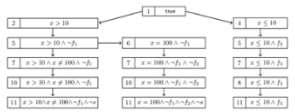
How can we reuse analysis results throughout different automata?

Automaton-Guided Reachability Analysis

Program

```
f1 = true;
if (x == 100)
  f2 = false;
if (f1)
  s = f2;
else
  s = f1;
```

Witnesses of Un-/Satisfiability



Applications:

- Whitebox/Model-Based Testing
- Reachability-Oracle for Path-Insensitive Program Analyses
- Program Exploration/Debugging

Reachability Analysis

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

Reachability Analysis

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

Abstract Reachability Graph

Reachability Analysis

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

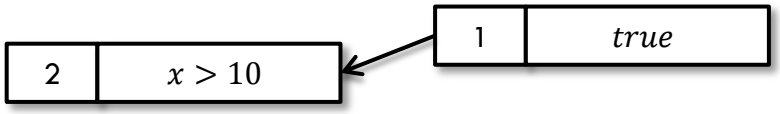
Abstract Reachability Graph

Reachability Analysis

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

Abstract Reachability Graph

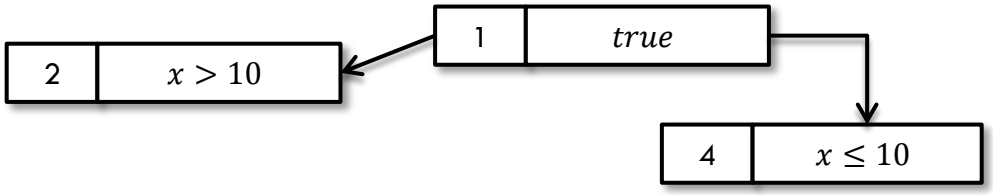


Reachability Analysis

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

Abstract Reachability Graph

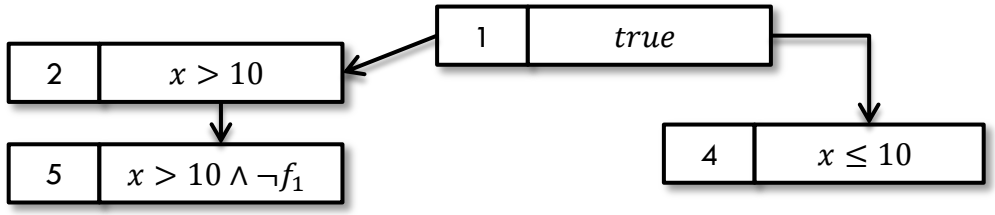


Reachability Analysis

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

Abstract Reachability Graph

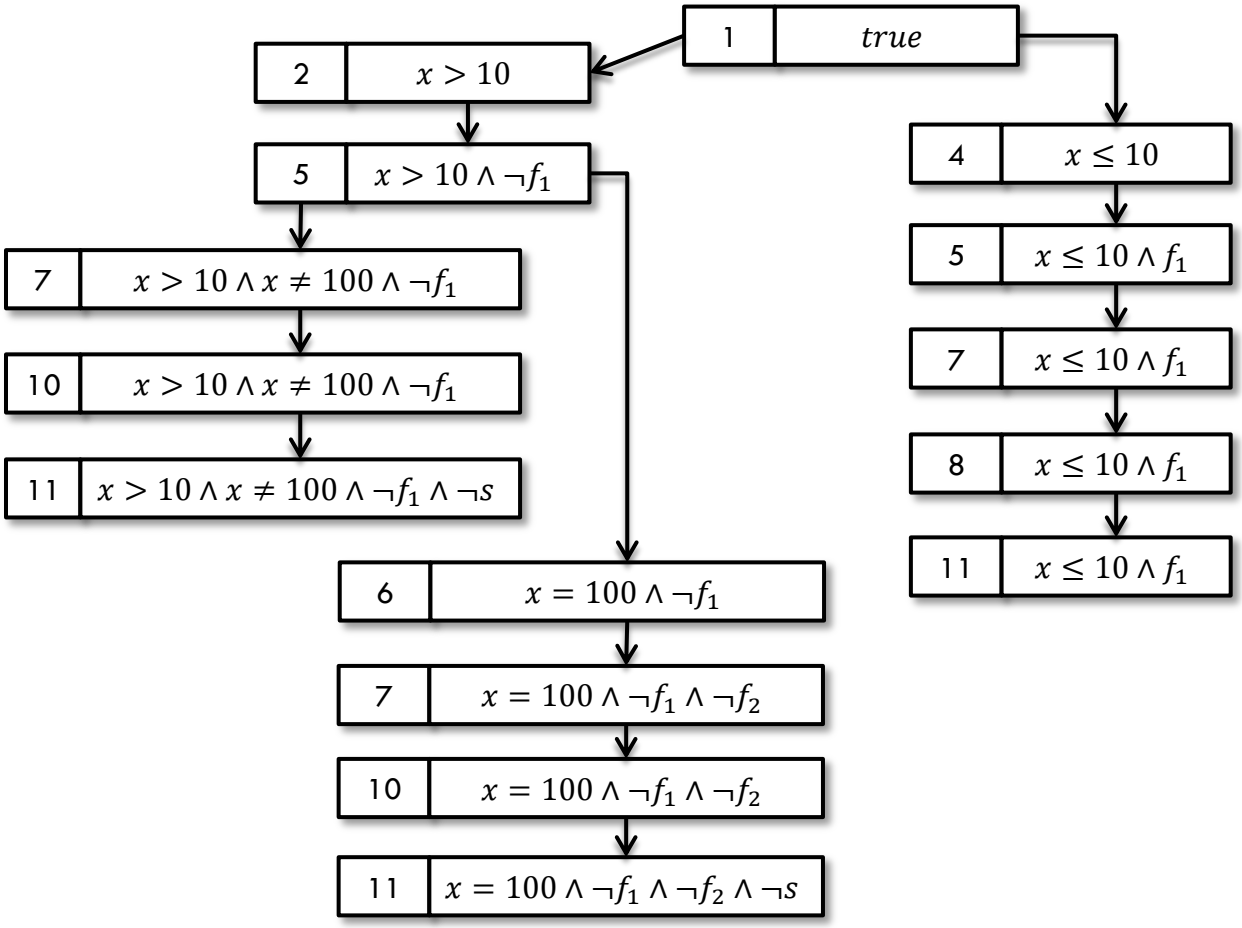


Reachability Analysis

Program

```
1  if  (x > 10)
2       f1 = false;
3  else
4       f1 = true;
5  if  (x == 100)
6       f2 = false;
7  if  (f1)
8       s = f2;
9  else
10      s = f1;
```

Abstract Reachability Graph



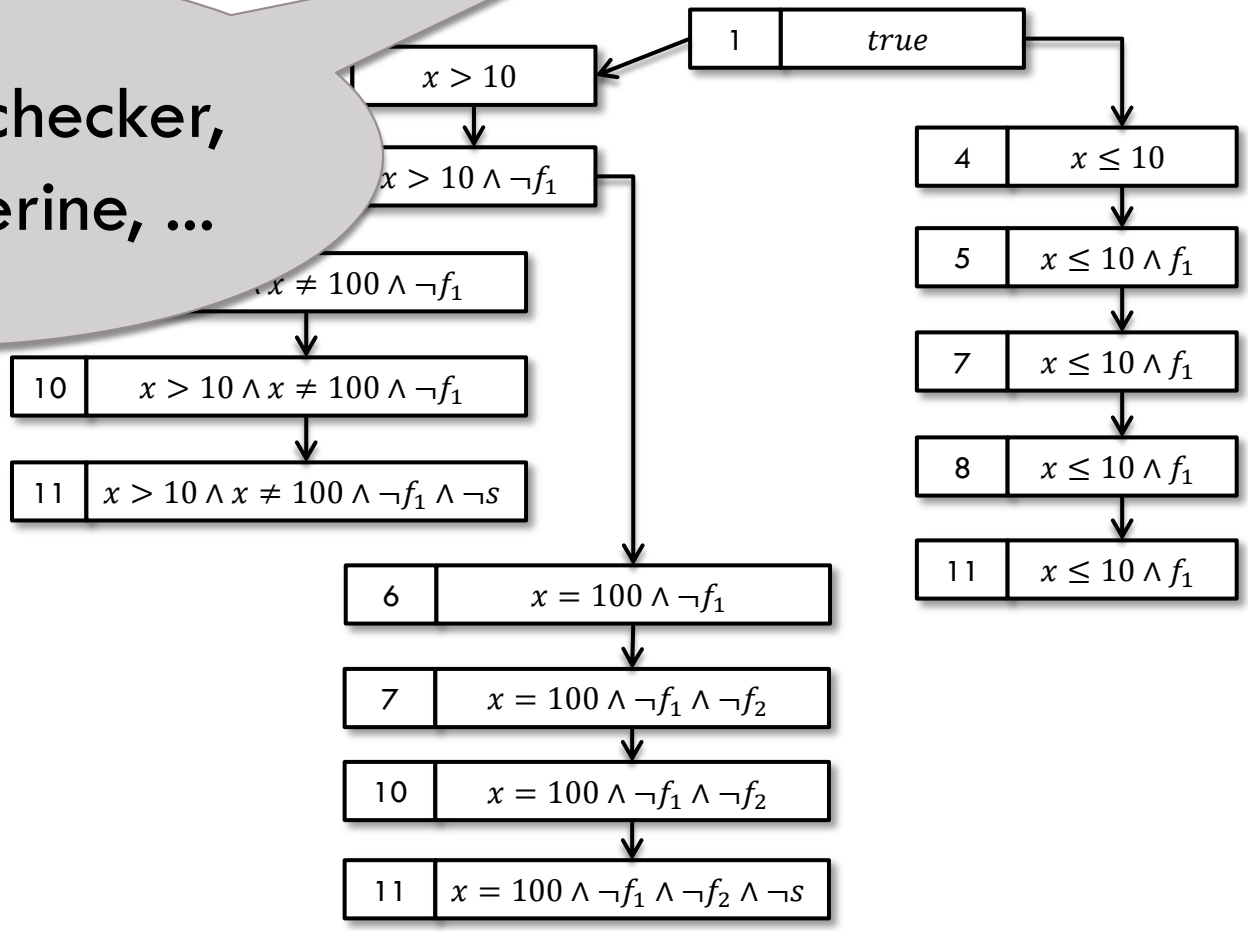
Reachability Analysis

Program

```
1  if (
5
6       f2 = false;
7  if (f1)
8       s = f2;
9  else
10      s = f1;
```

BLAST, CPAchecker,
UFO, Wolverine, ...

Abstract Reachability Graph



Reachability Analysis

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  if  (f1)
8        s = f2;
9  else
10       s = f1;
```

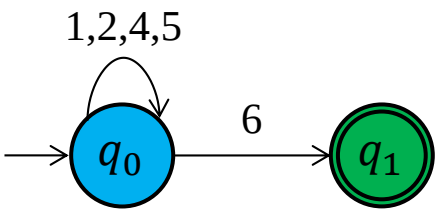
Abstract Reachability Graph

Automaton-Guided Reach. Analysis

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  ...
```

Automaton



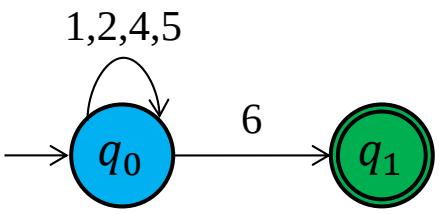
Abstract Reachability Graph

Automaton-Guided Reach. Analysis

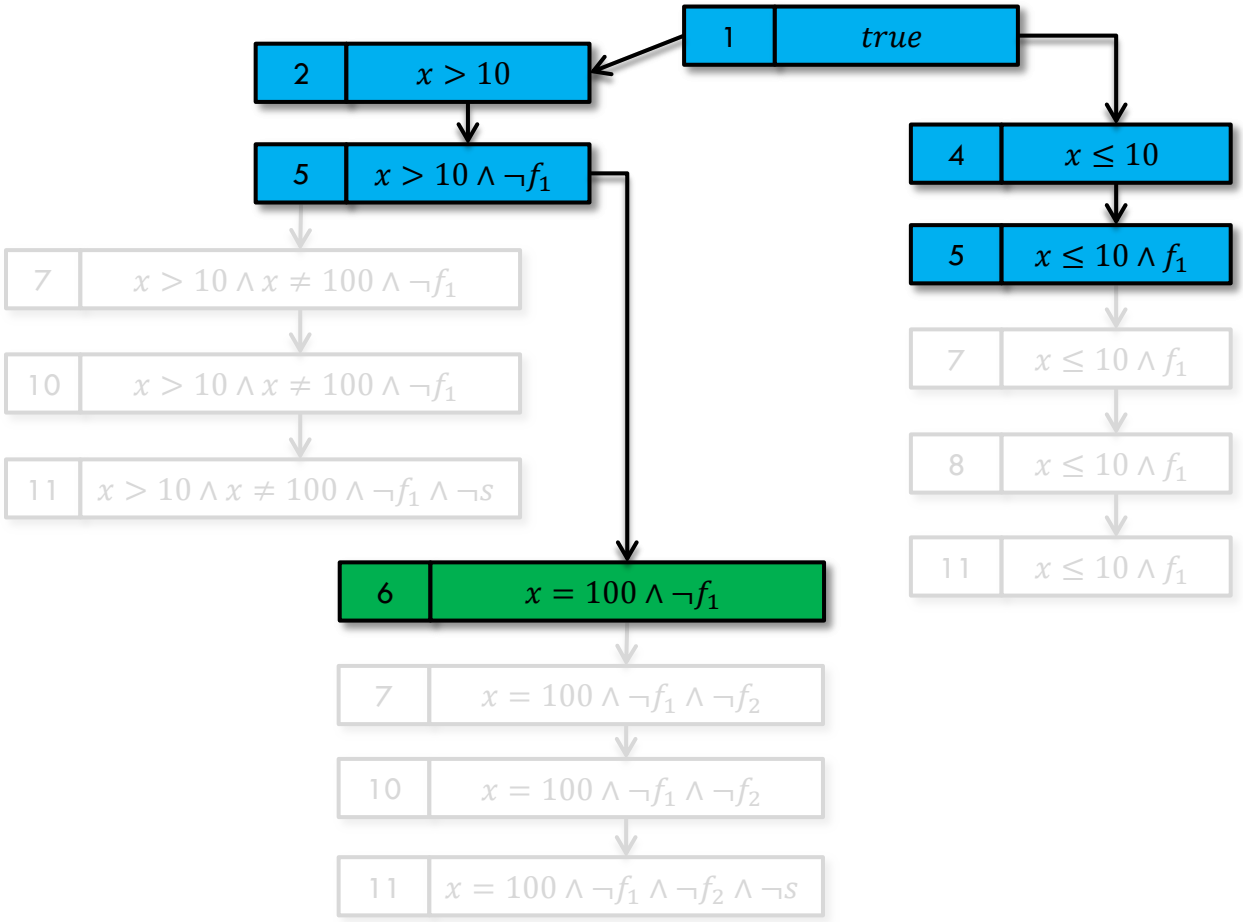
Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  ...
```

Automaton



Abstract Reachability Graph

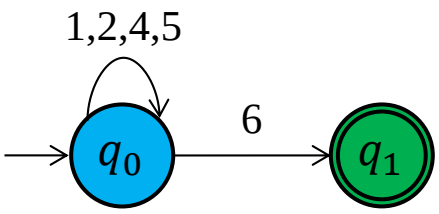


Automaton-Guided Reach. Analysis

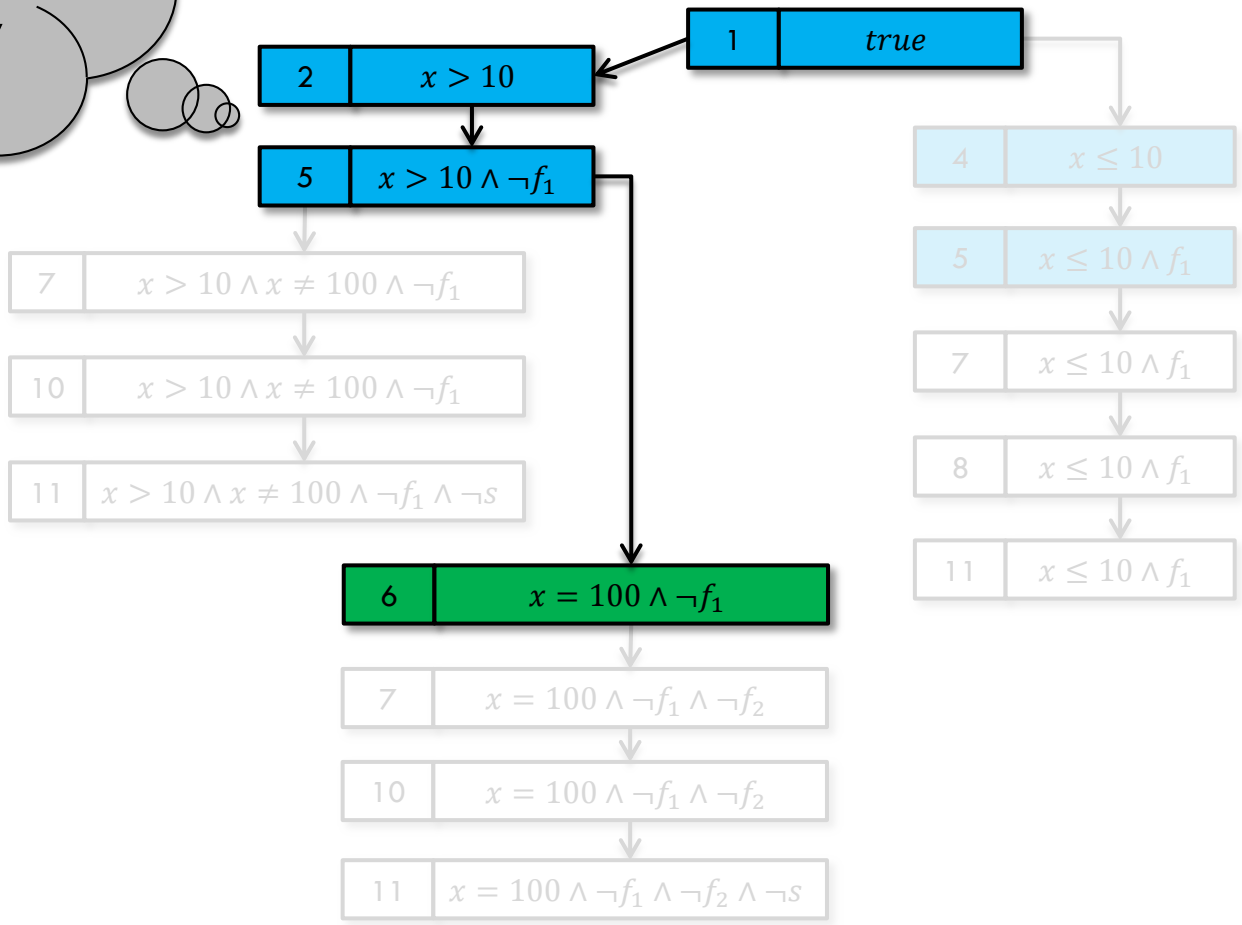
Witness of Satisfiability

```
3
4       f1 = true;
5   if  (x == 100)
6       f2 = false;
7   ...
```

Automaton



Abstract Reachability Graph



Automaton-Guided Reach. Analysis

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  ...
```

Automaton

Abstract Reachability Graph

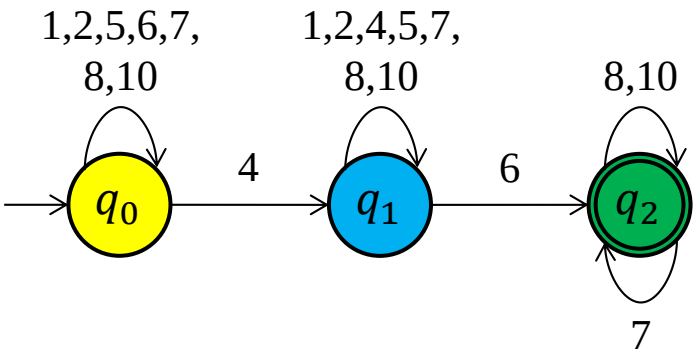
Automaton-Guided Reach. Analysis

Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  ...
```

Abstract Reachability Graph

Automaton

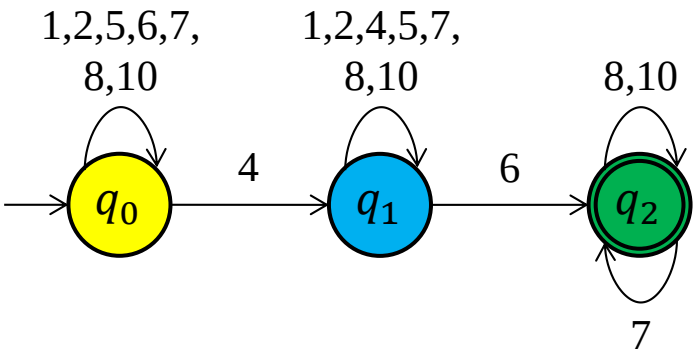


Automaton-Guided Reach. Analysis

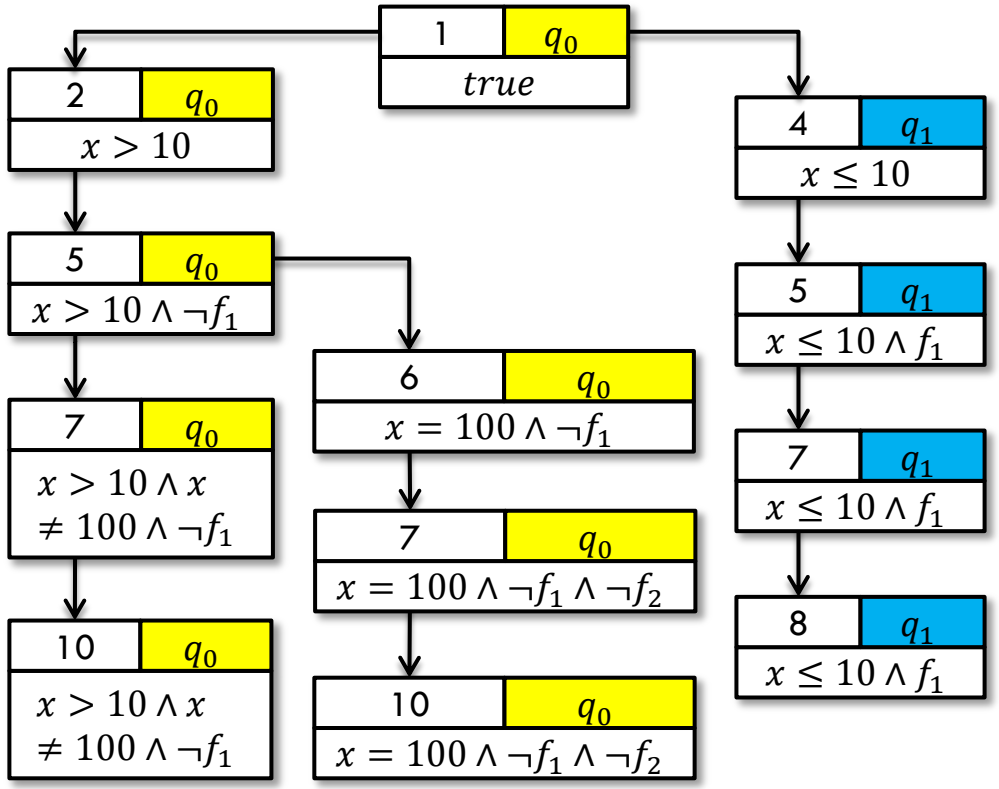
Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  ...
```

Automaton



Abstract Reachability Graph

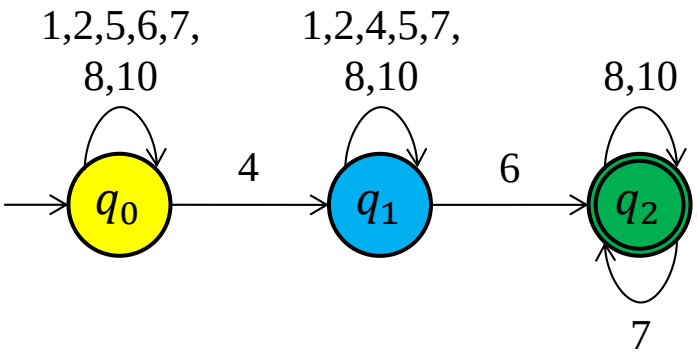


Automaton-Guided Reach. Analysis

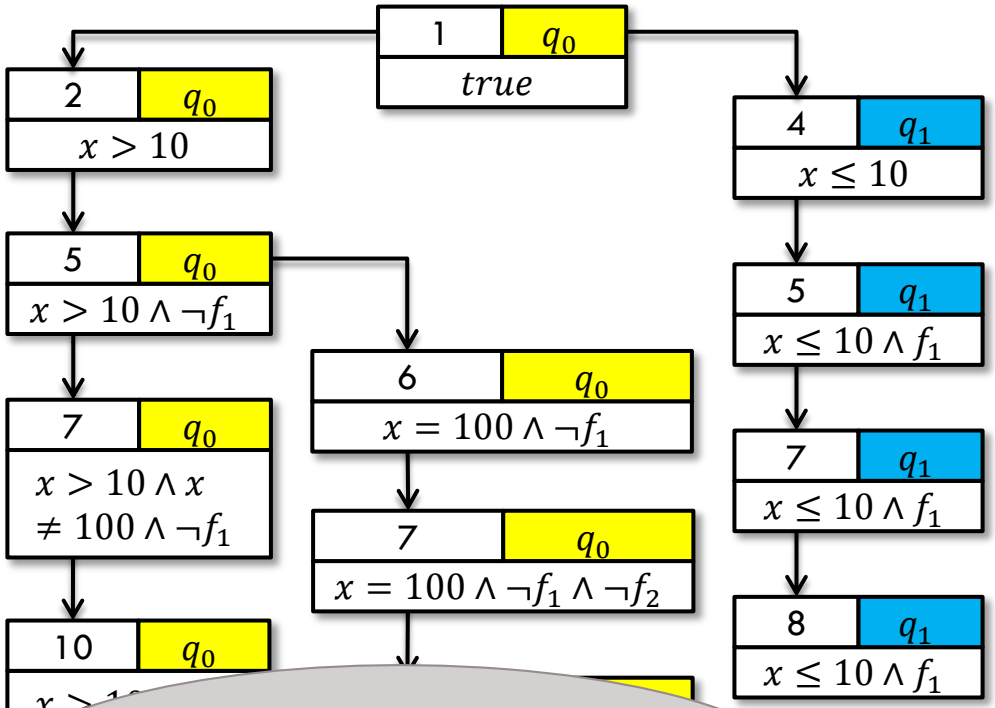
Program

```
1  if  (x > 10)
2        f1 = false;
3  else
4        f1 = true;
5  if  (x == 100)
6        f2 = false;
7  ...
```

Automaton



Abstract Reachability Graph



No abstract state labeled with q_2!

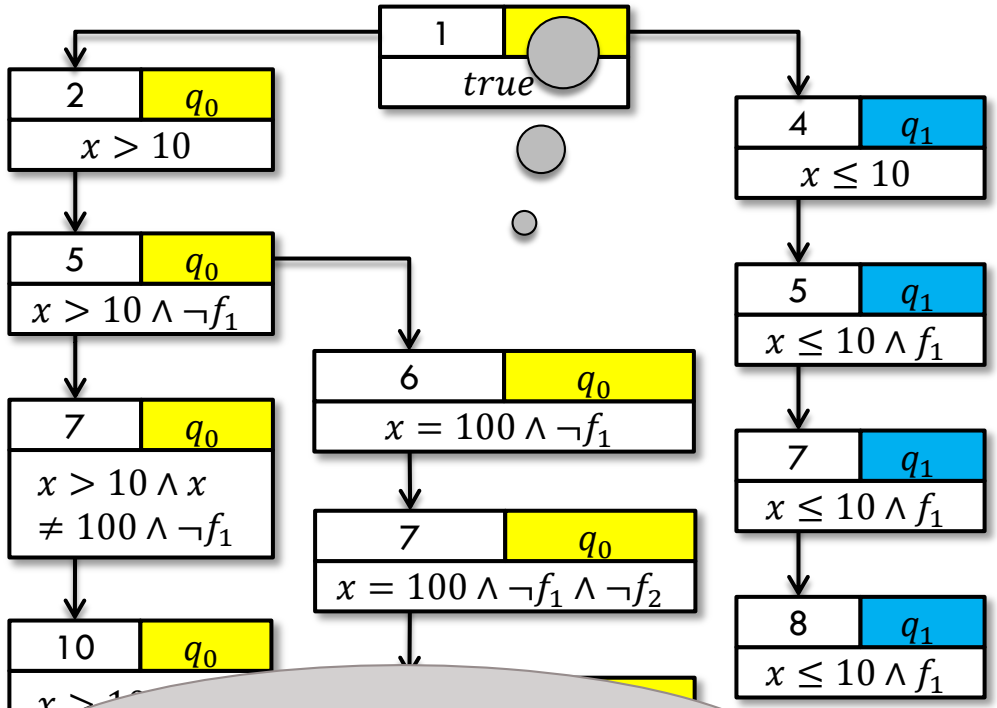
Automaton-Guided Re

Witness of Unsatisfiability

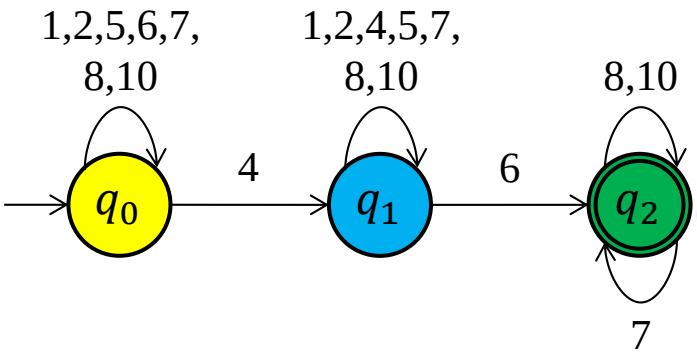
Program

```
1  if (x > 10)
2      f1 = false;
3  else
4      f1 = true;
5  if (x == 100)
6      f2 = false;
7  ...
```

Abstract Reach



Automaton



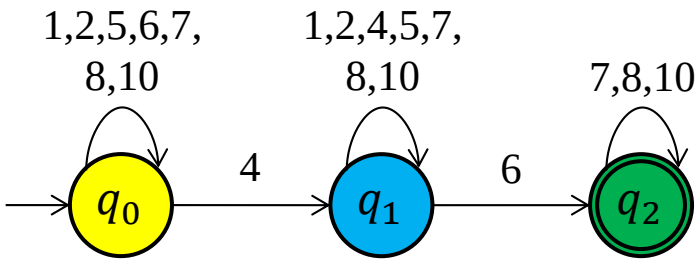
No abstract state labeled with q_2!

Information Reuse

Reuse Information for Shared Parts

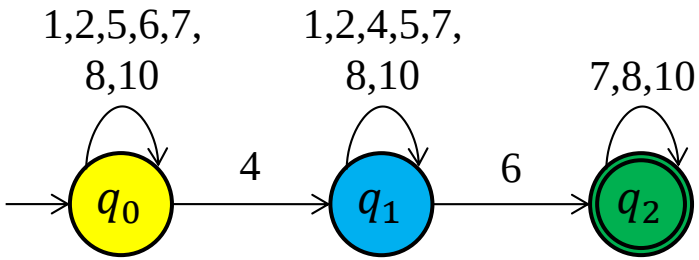
Reuse Information for Shared Parts

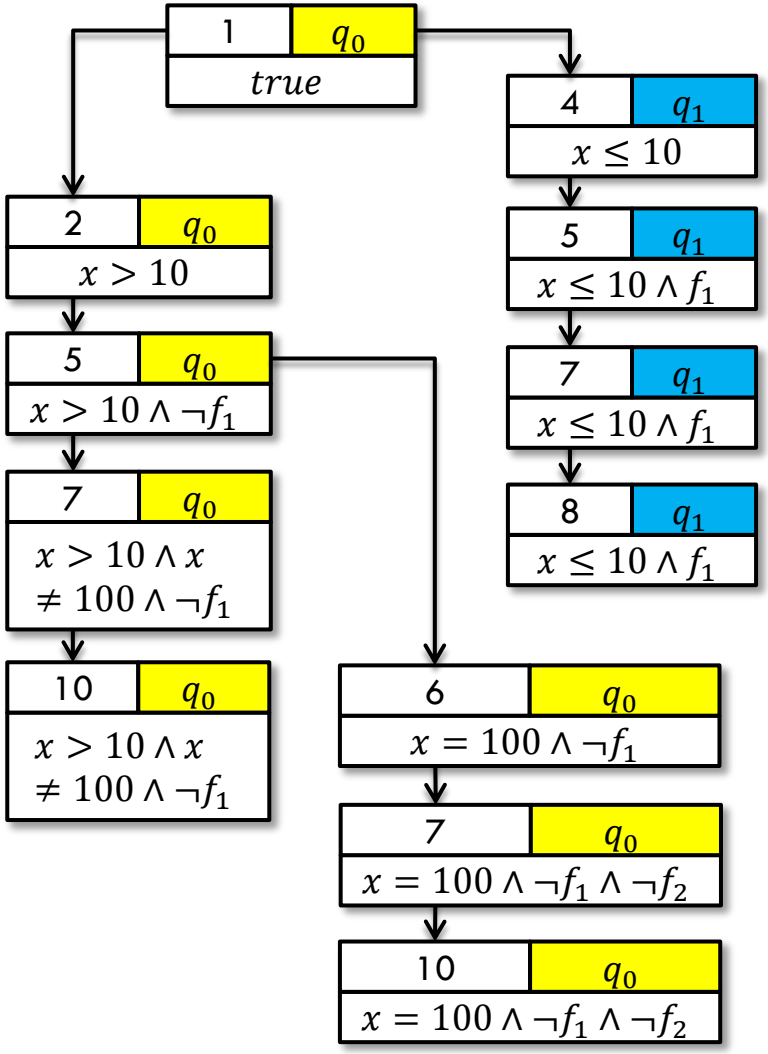
Automaton A



Reuse Information for Shared Parts

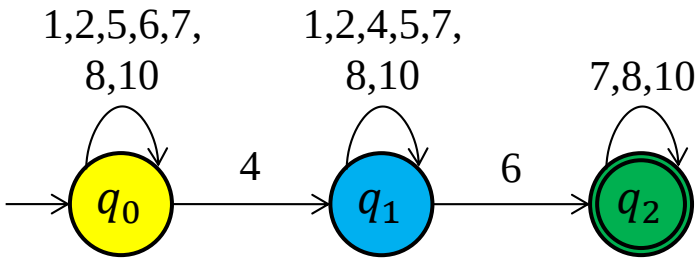
Automaton A

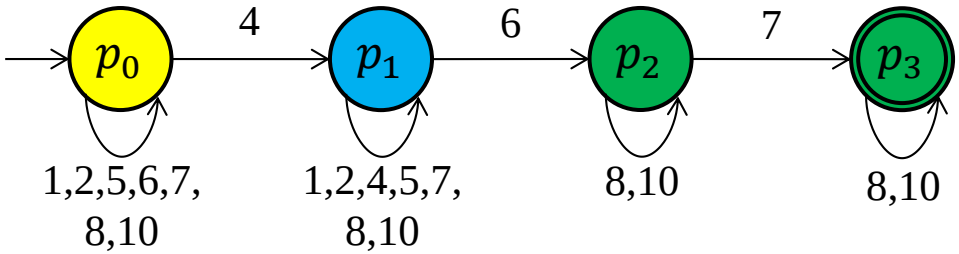




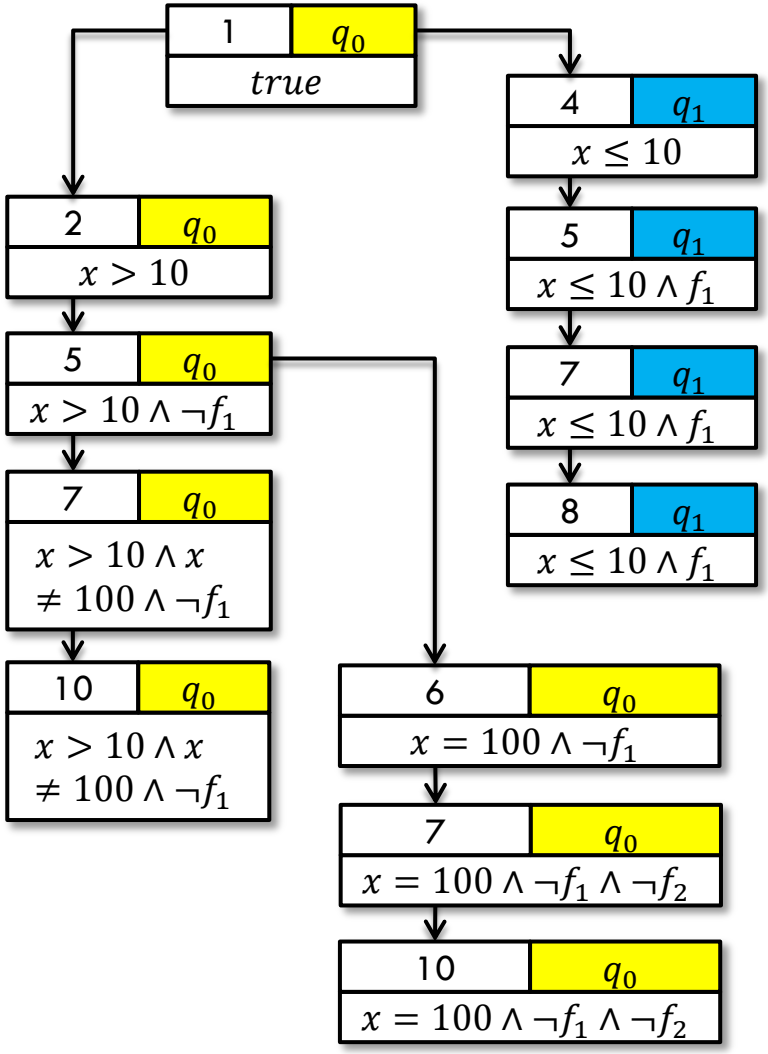
Reuse Information for Shared Parts

Automaton A

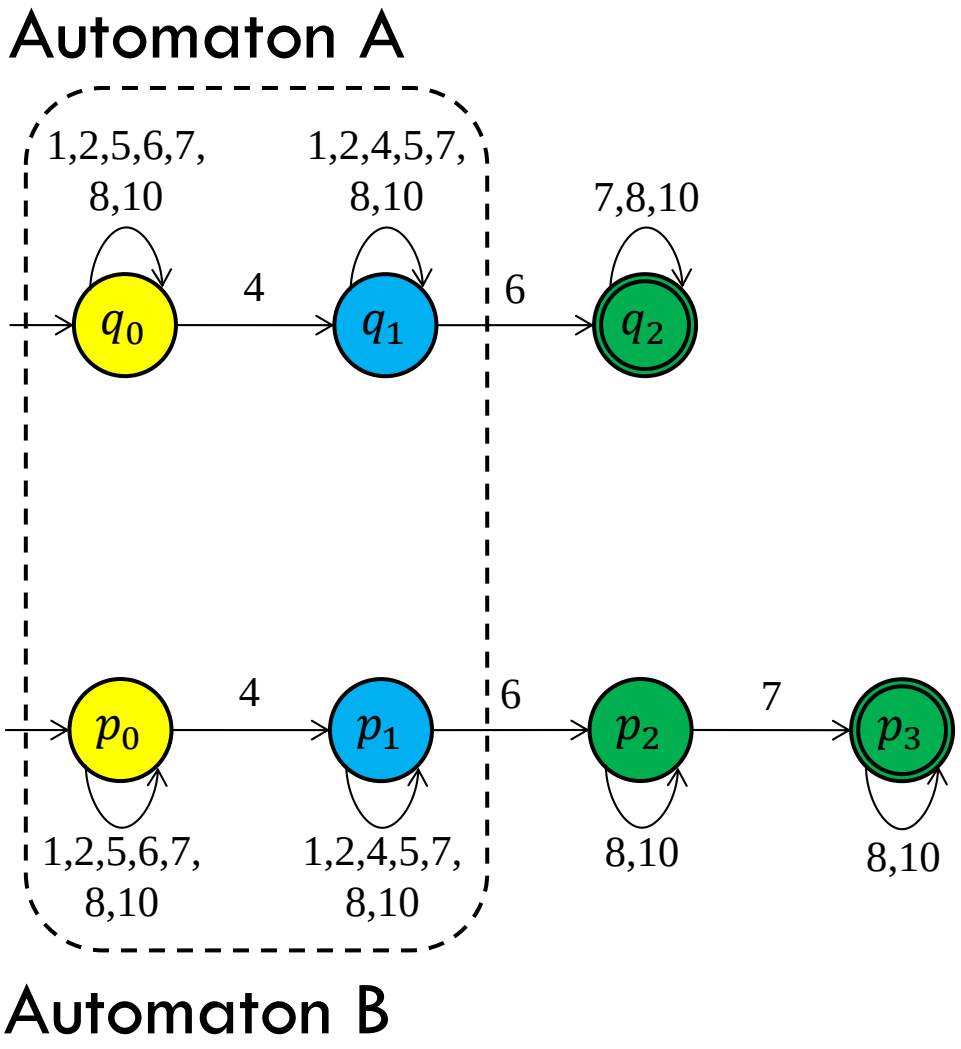


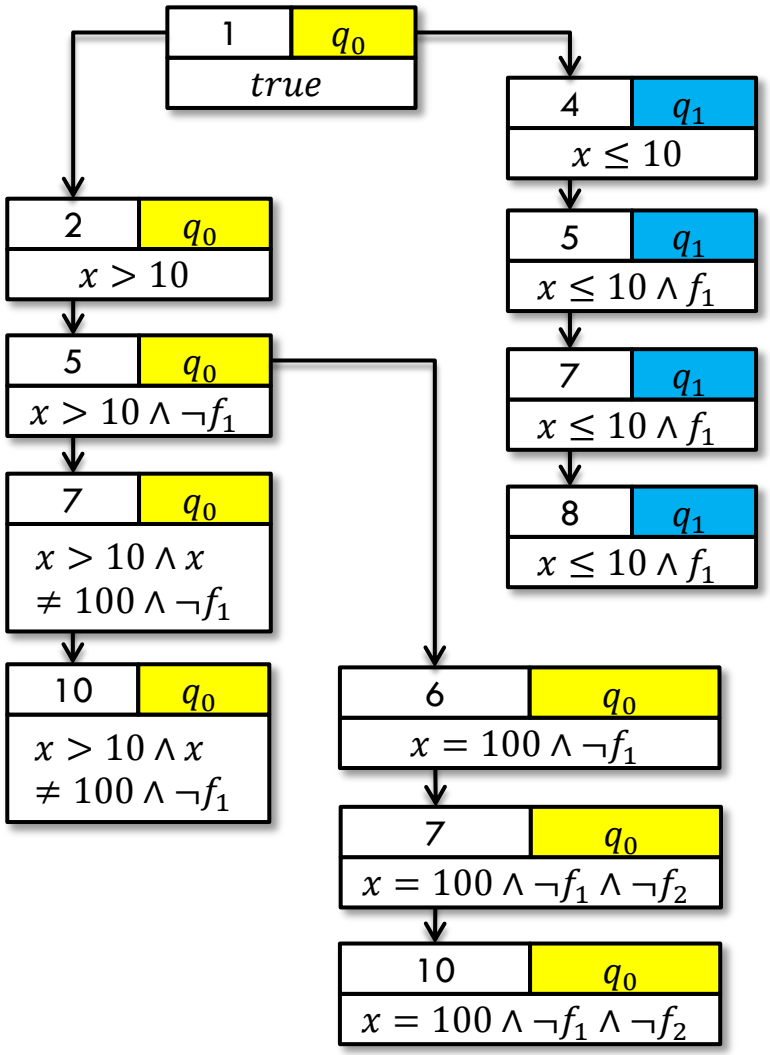


Automaton B

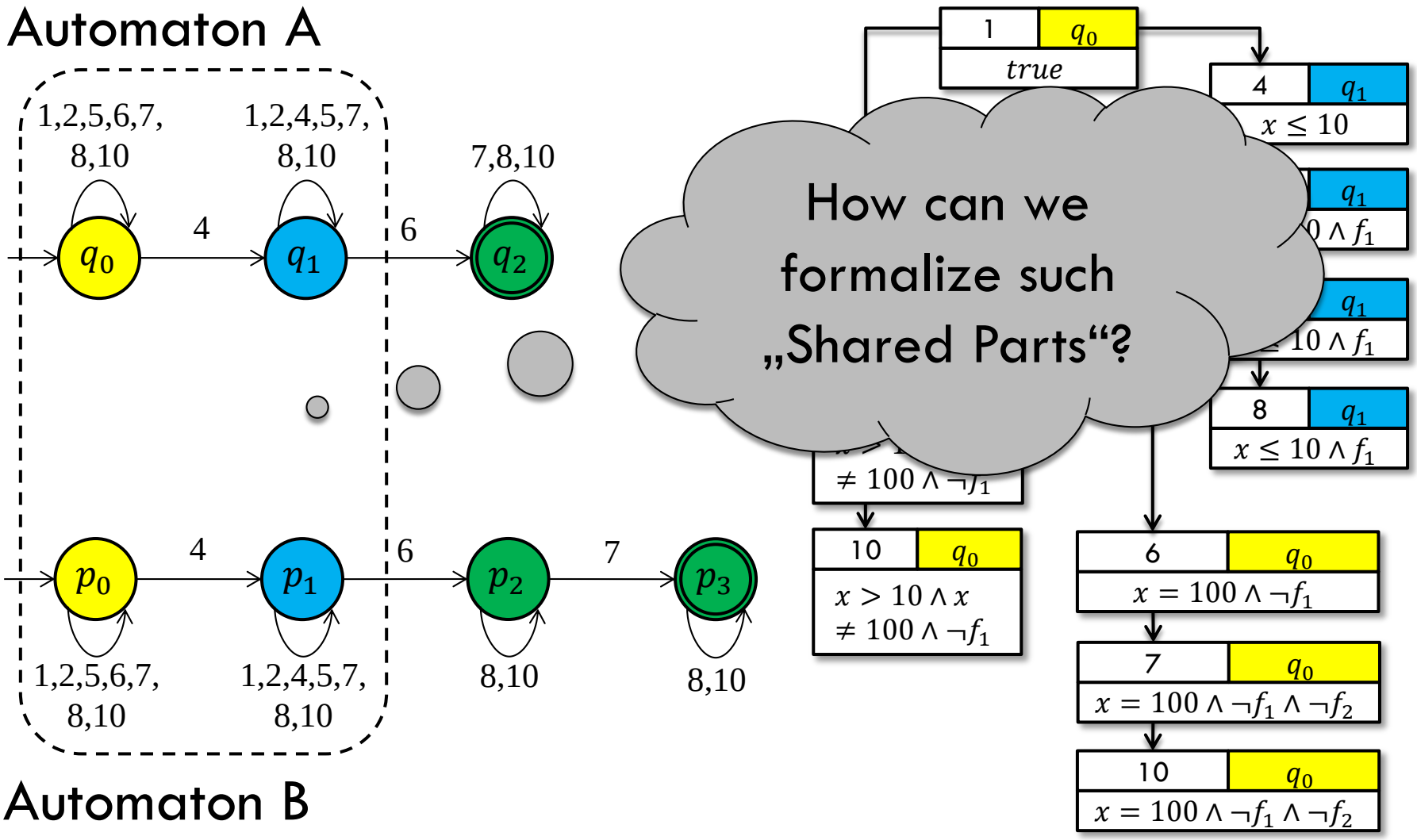


Reuse Information for Shared Parts



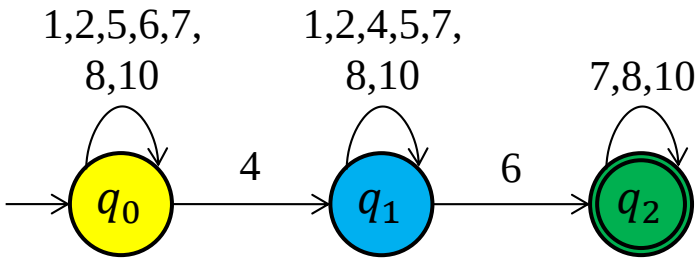


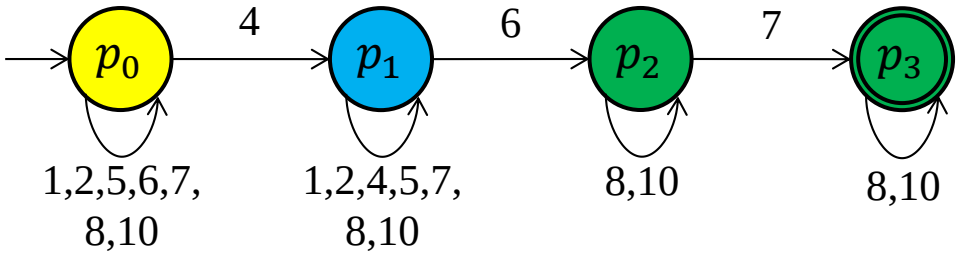
Reuse Information for Shared Parts



Shared Parts: Simulation Relation

Automaton A

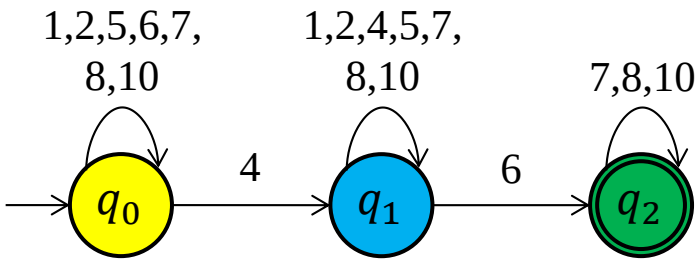




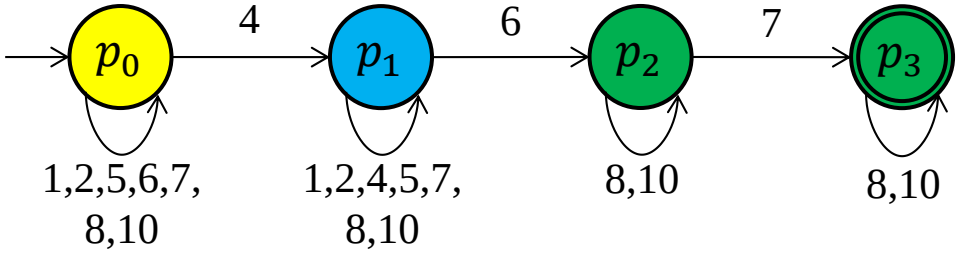
Automaton B

Shared Parts: Simulation Relation

Automaton A



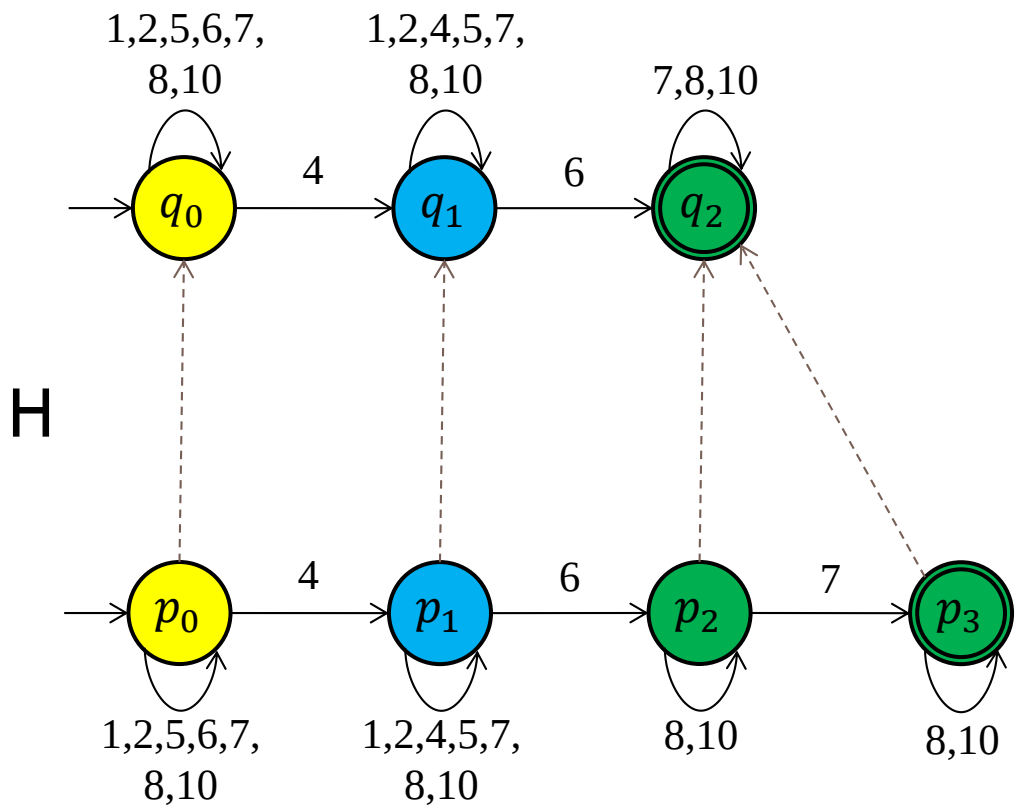
A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:



Automaton B

Shared Parts: Simulation Relation

Automaton A

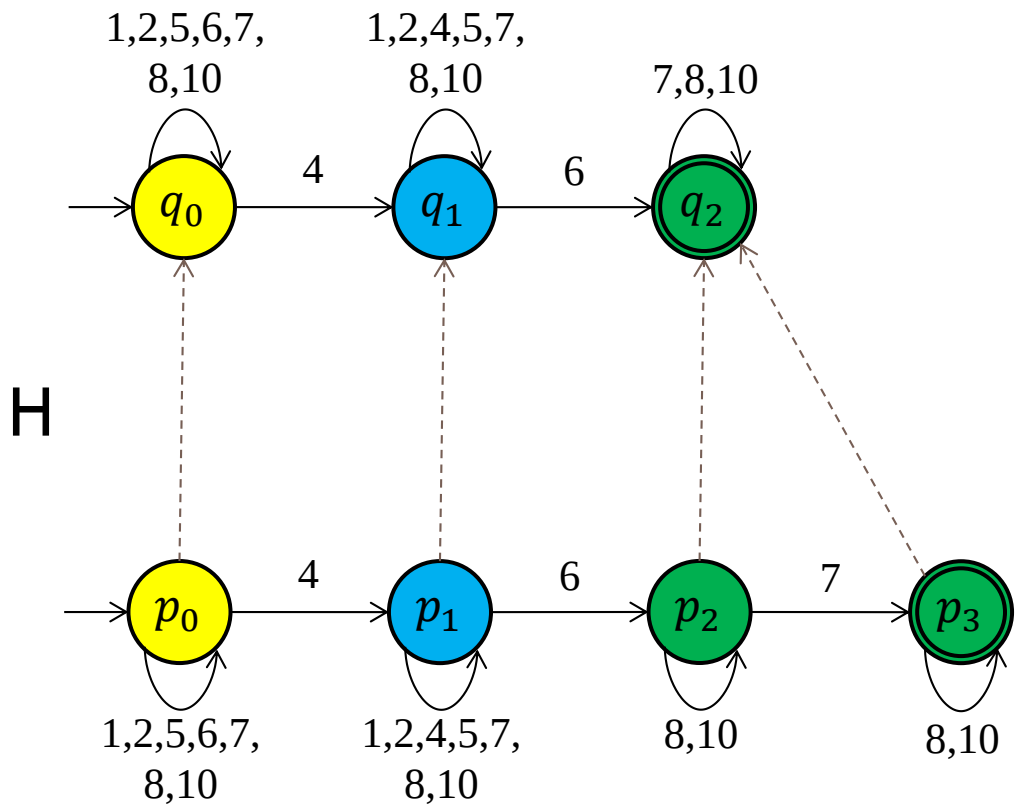


Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

Shared Parts: Simulation Relation

Automaton A



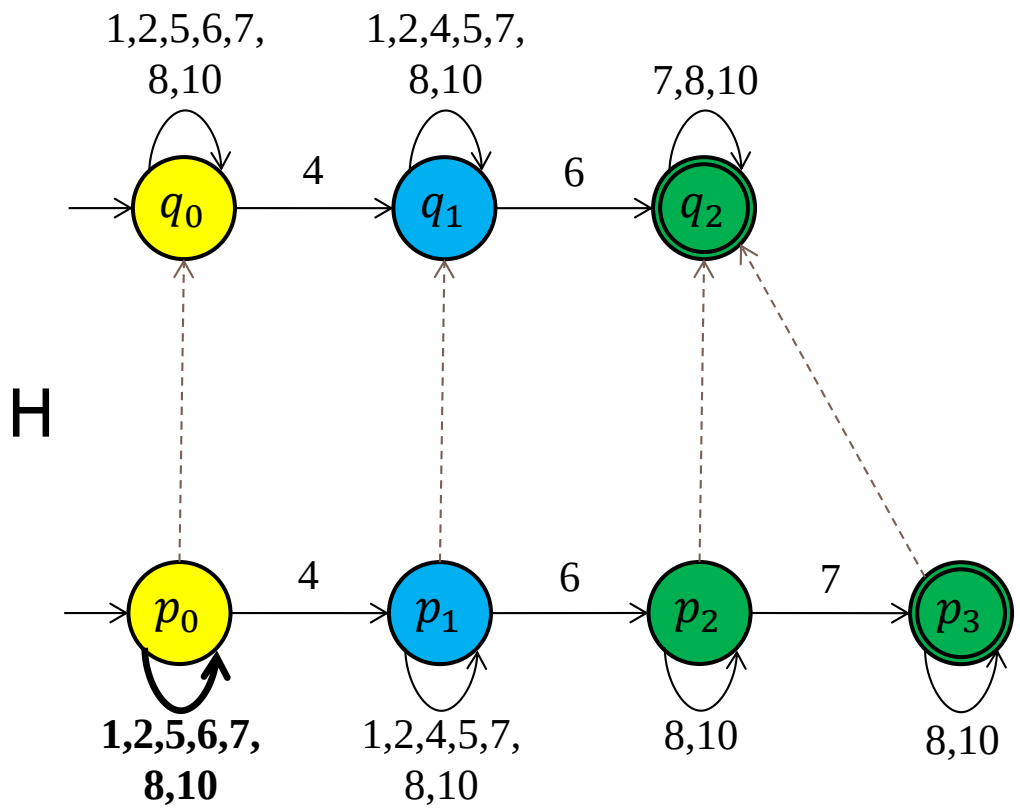
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



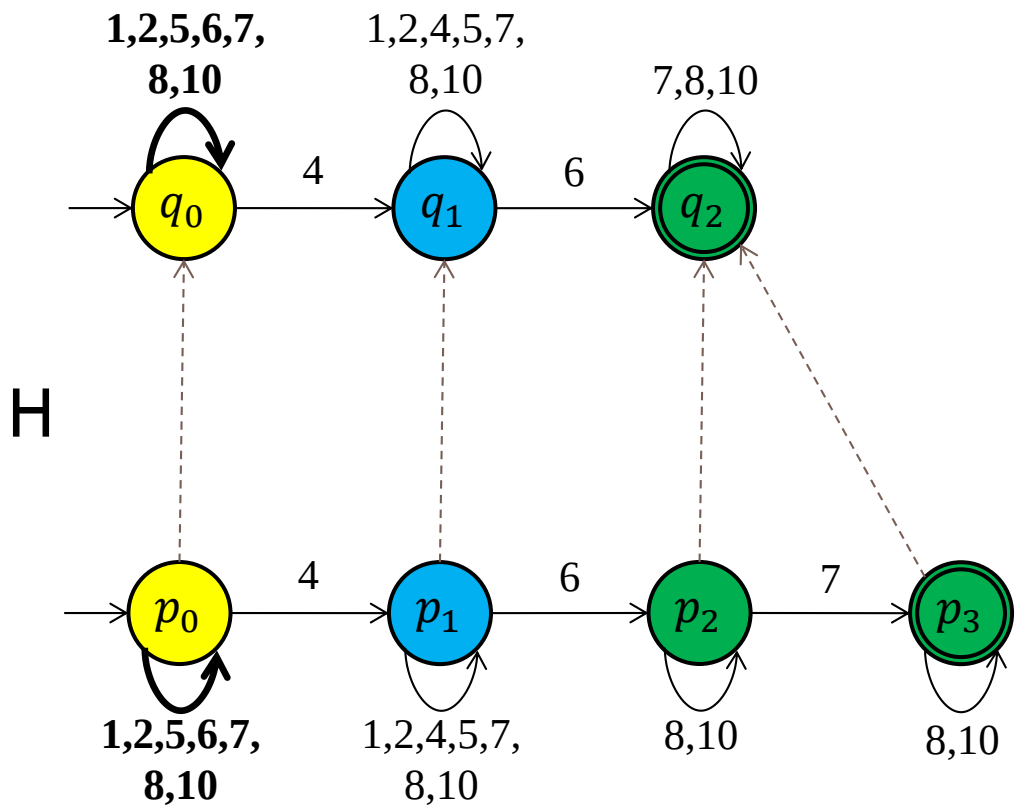
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



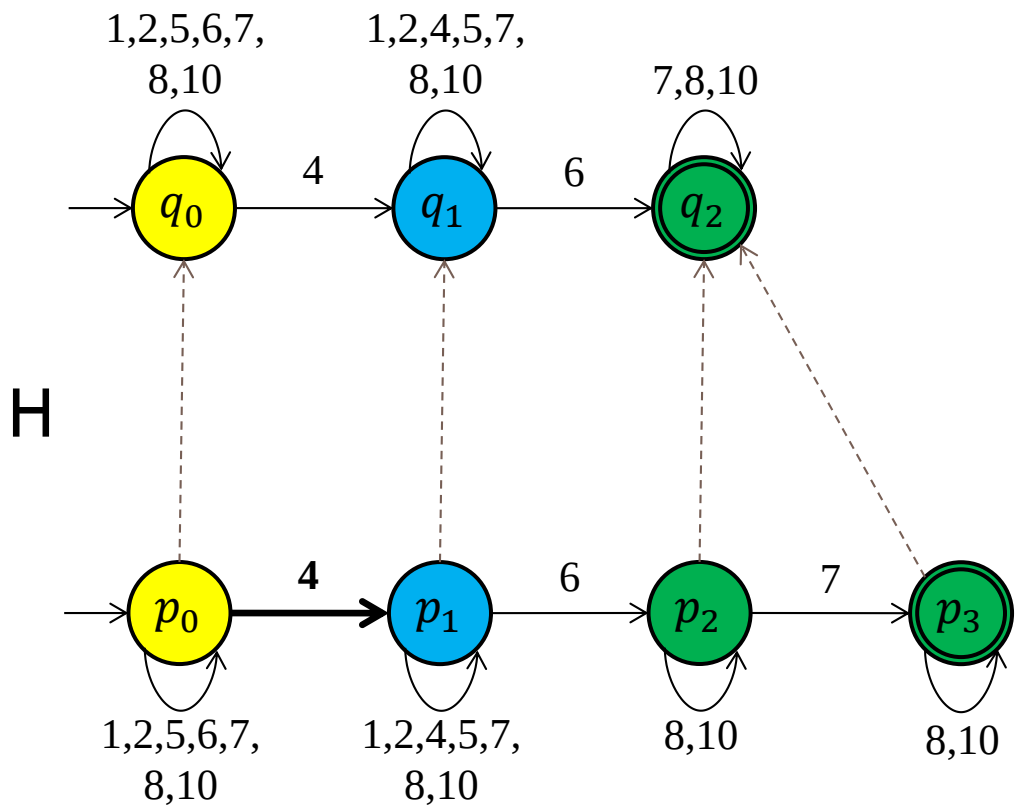
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



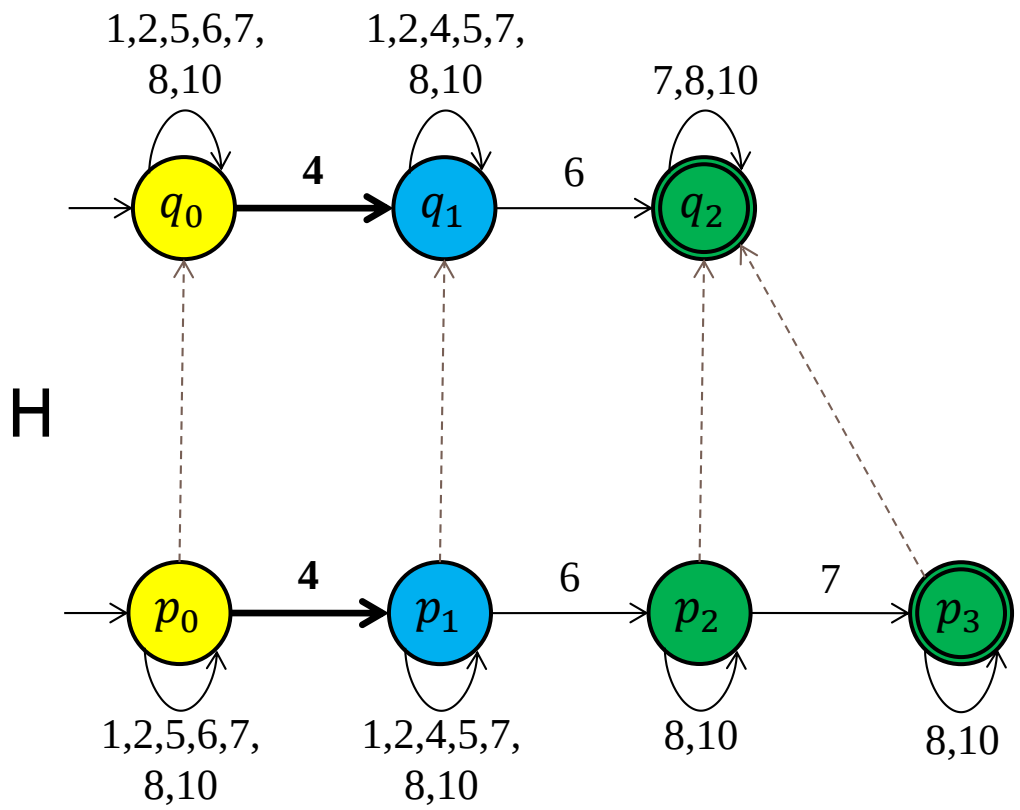
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



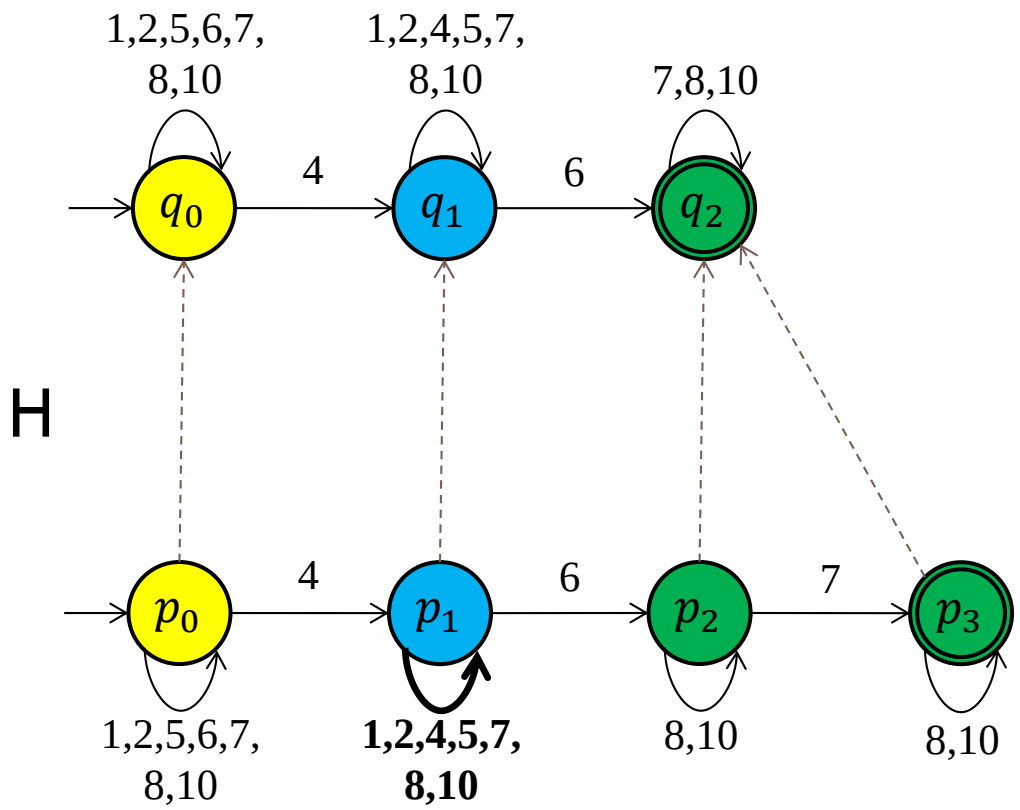
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



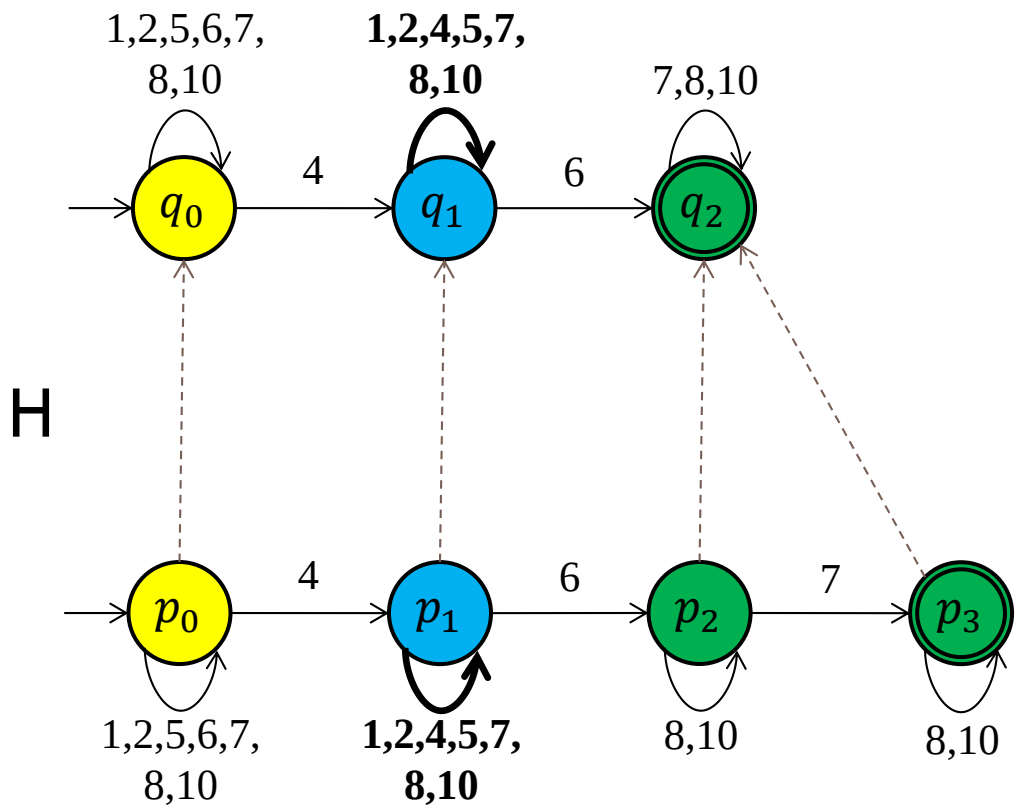
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



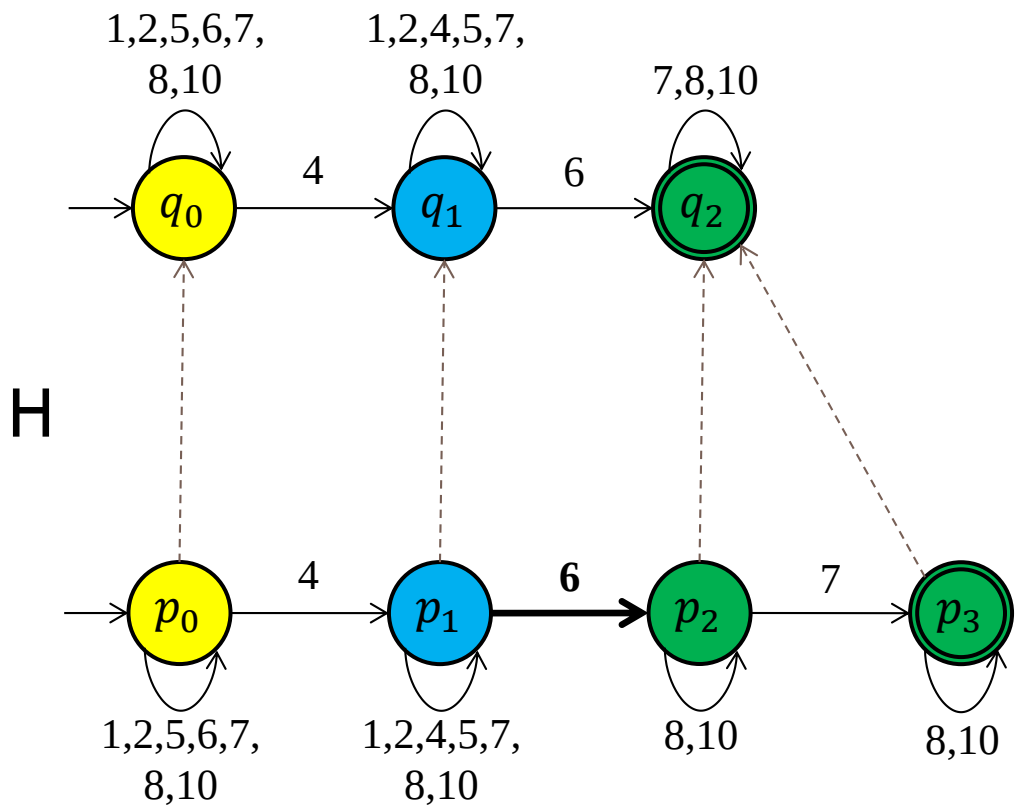
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



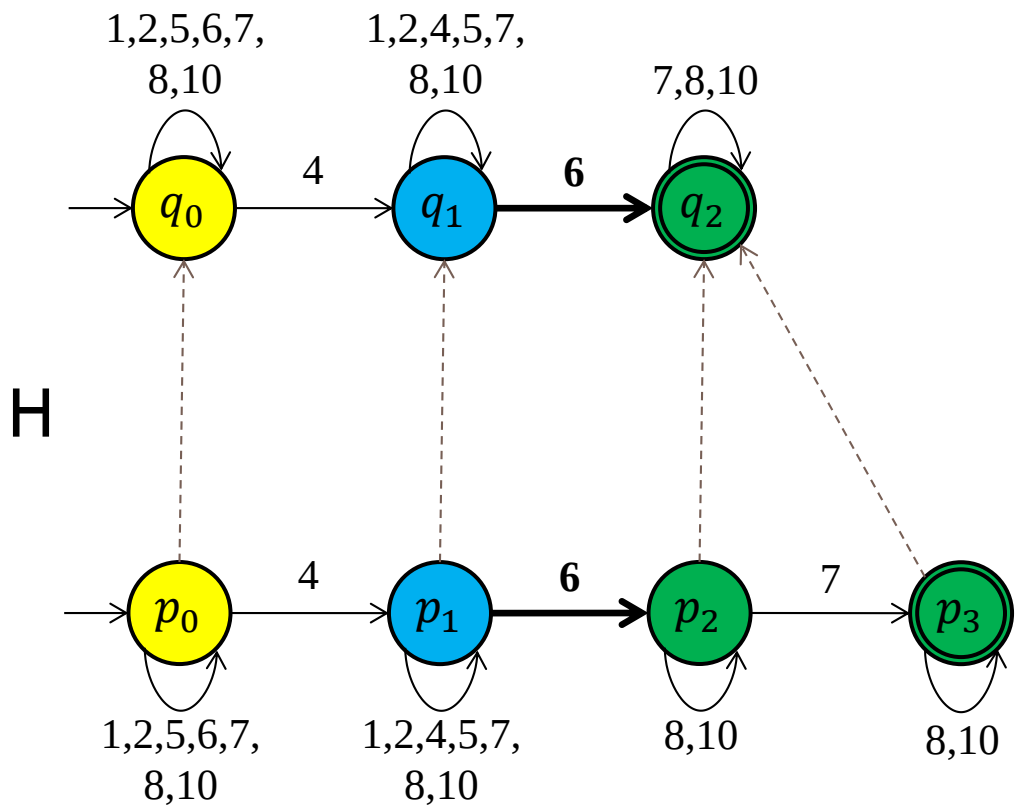
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



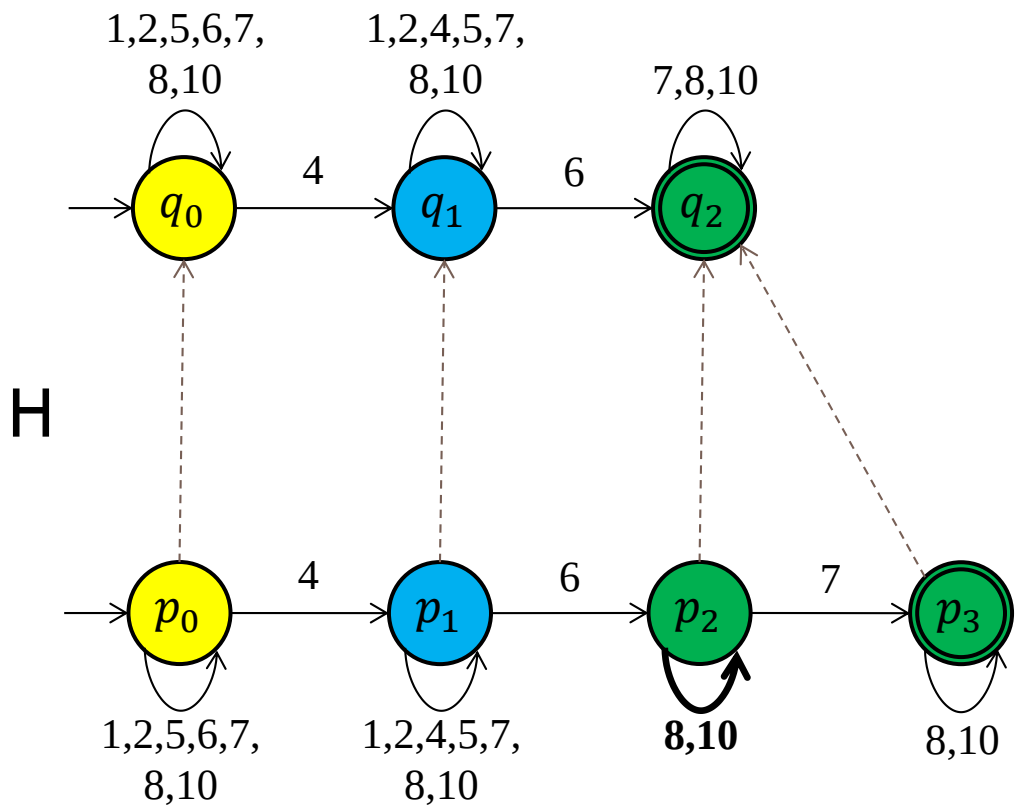
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



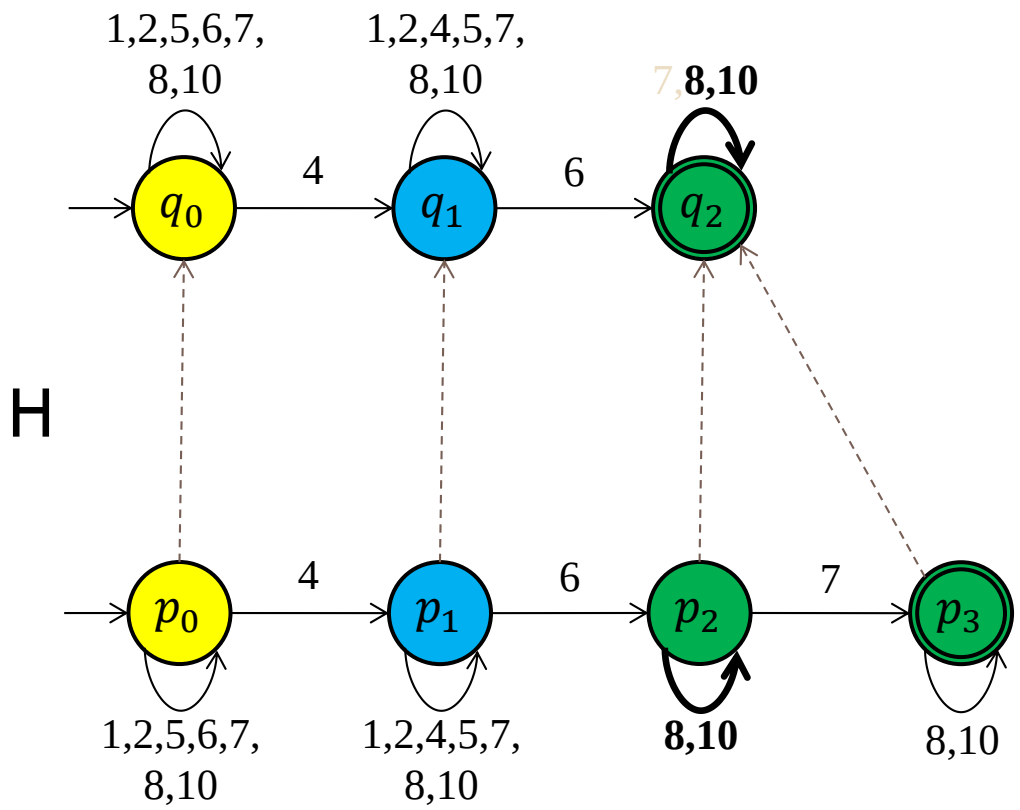
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



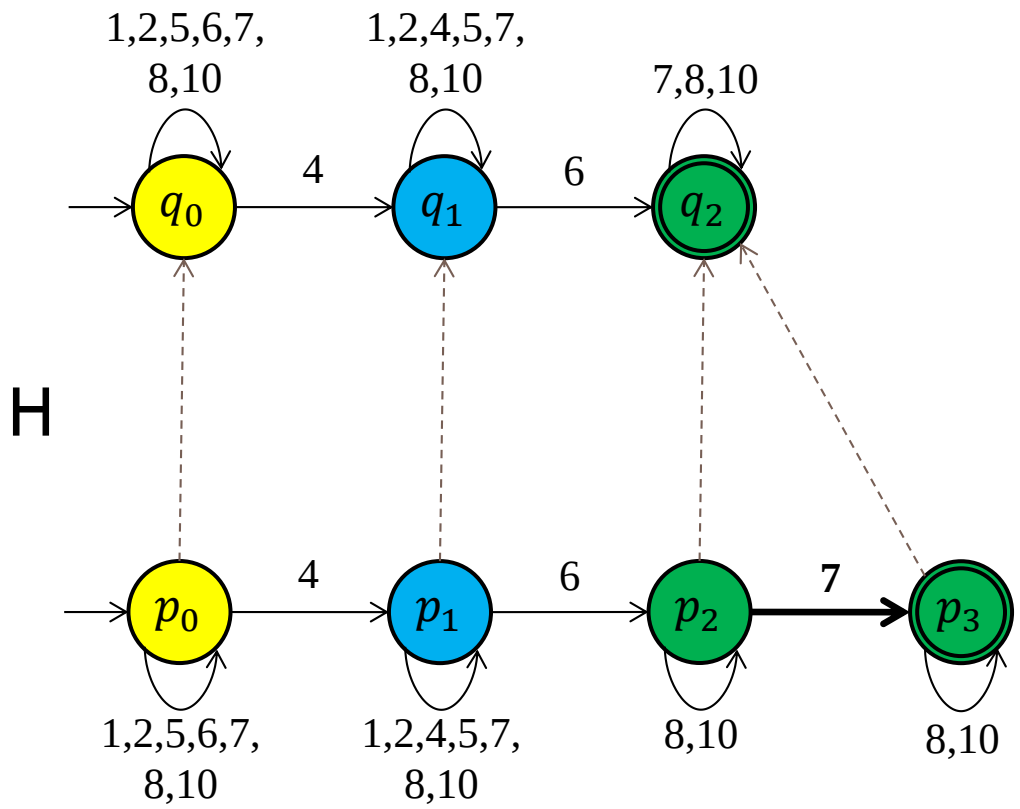
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



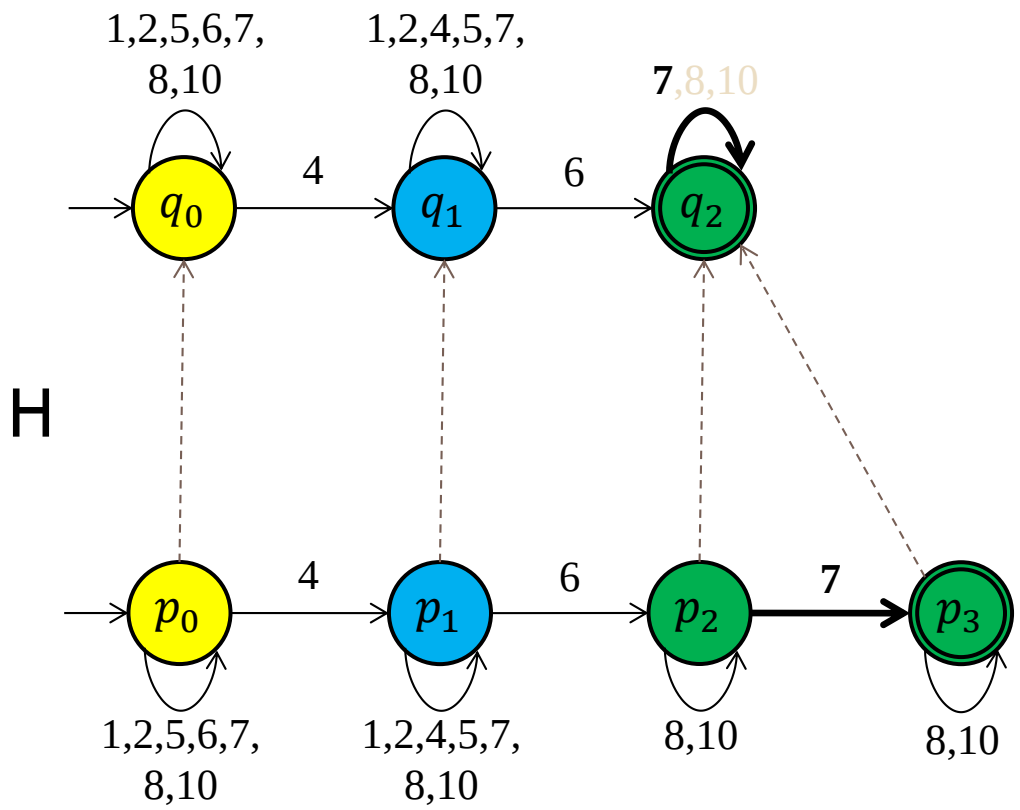
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



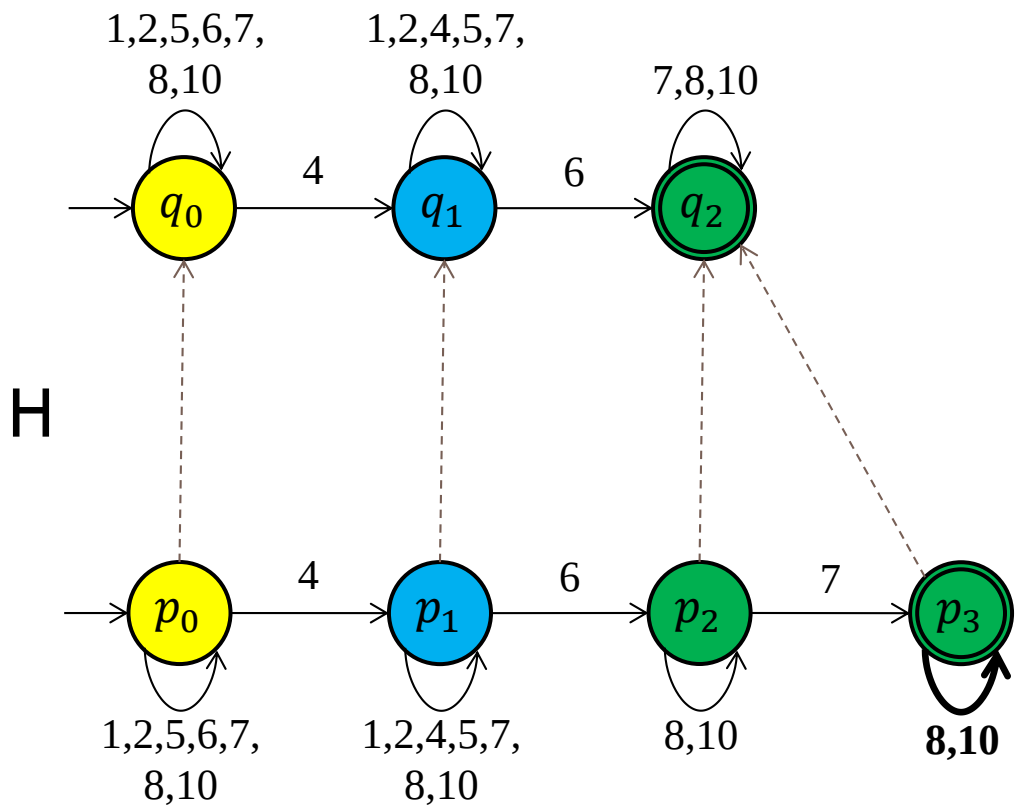
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



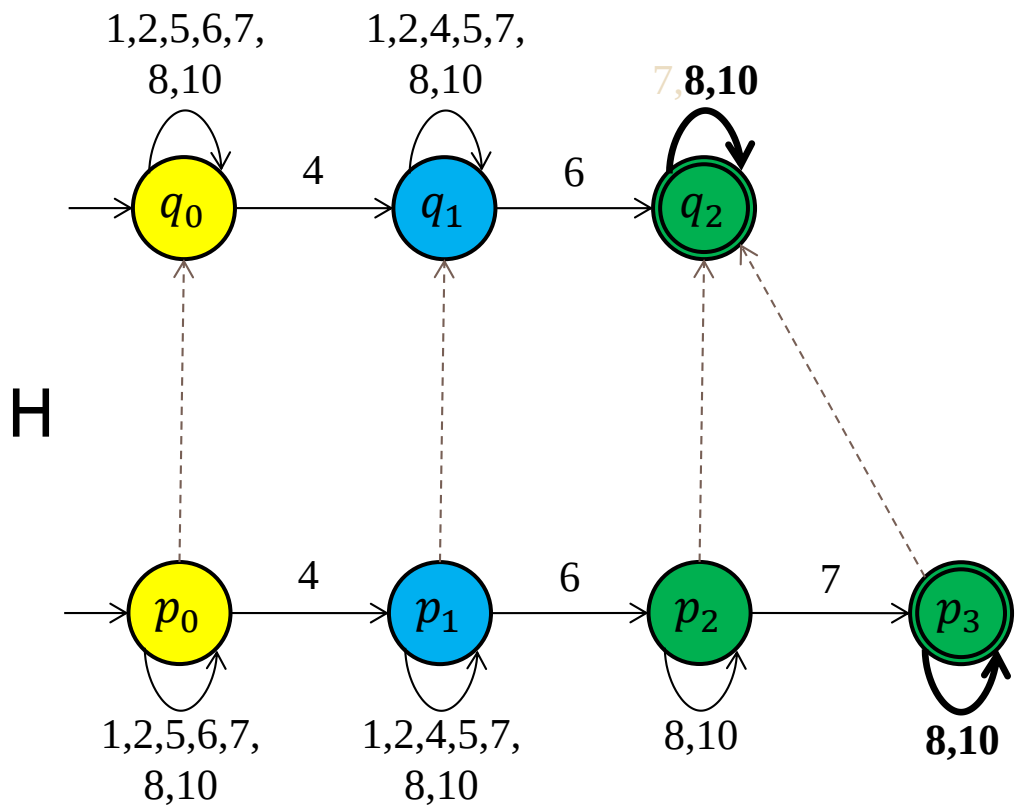
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



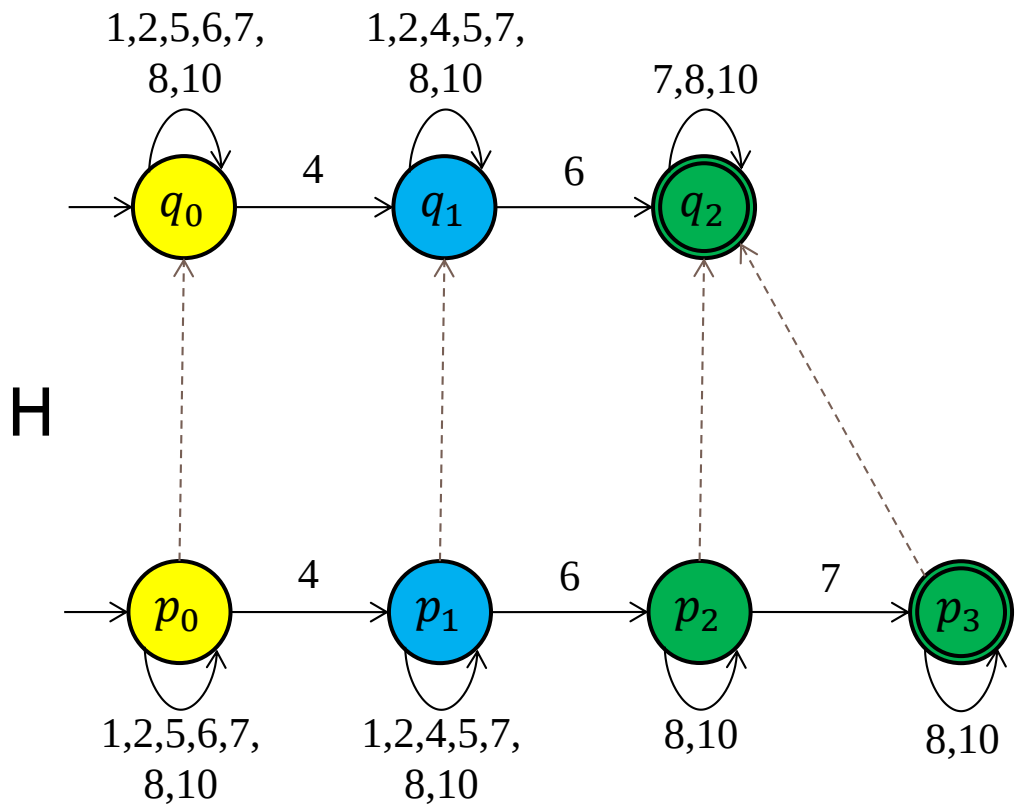
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



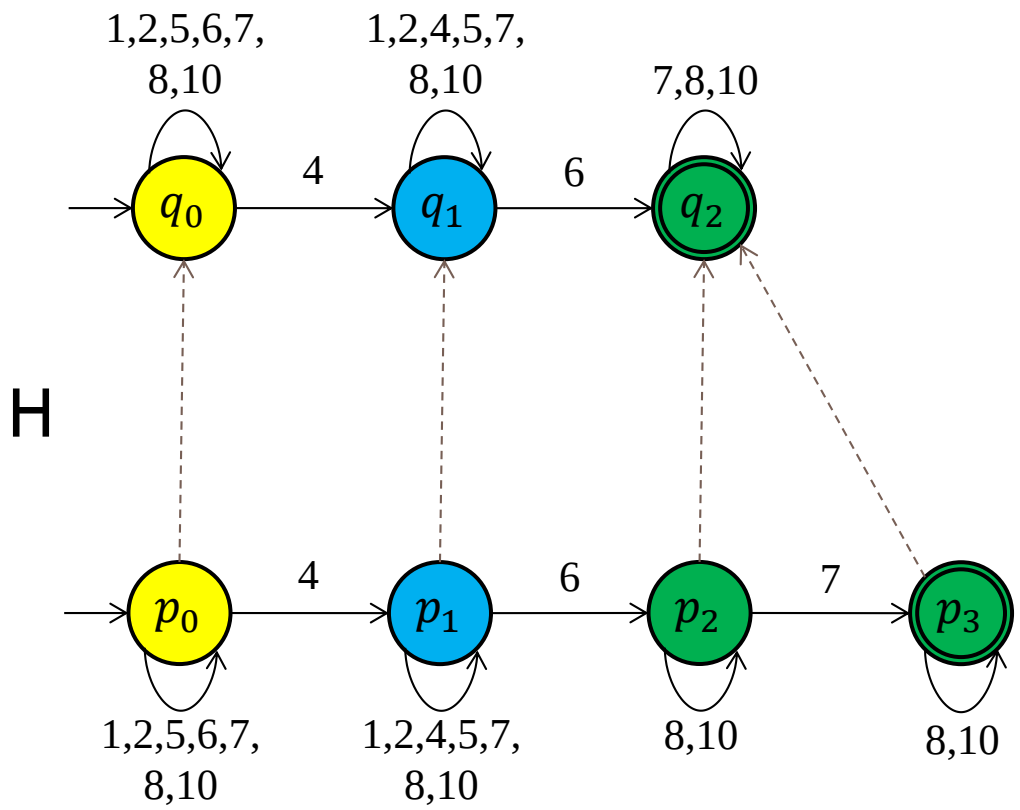
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation

Automaton A



Automaton B

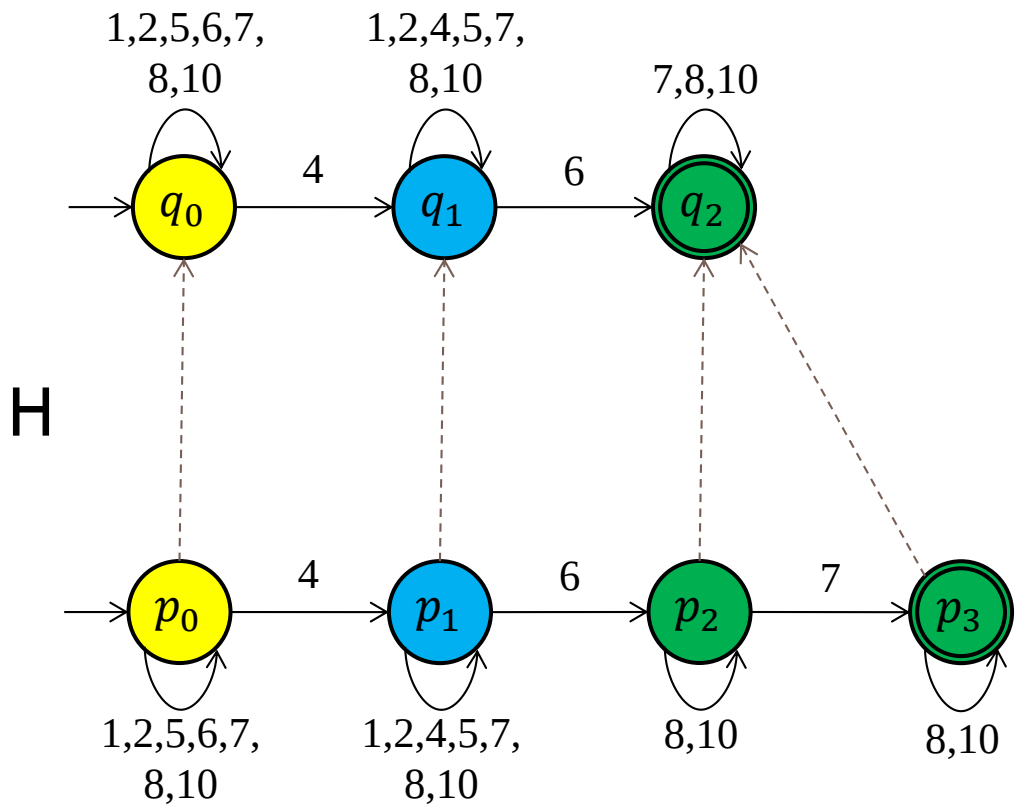
A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$.

We do not distinguish final (initial) states from non-final (non-initial) states!

Shared Parts: Simulation Relation

Automaton A

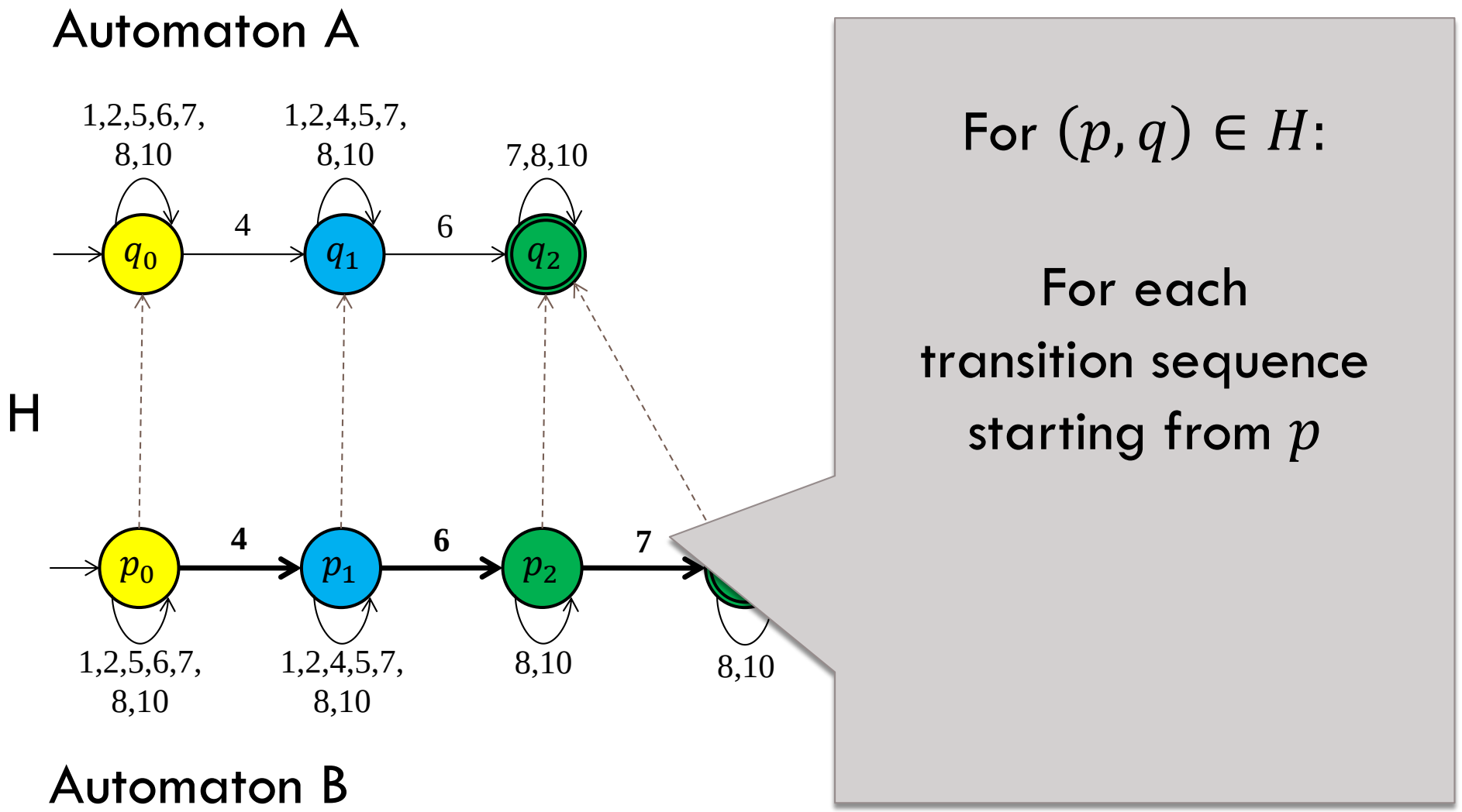


Automaton B

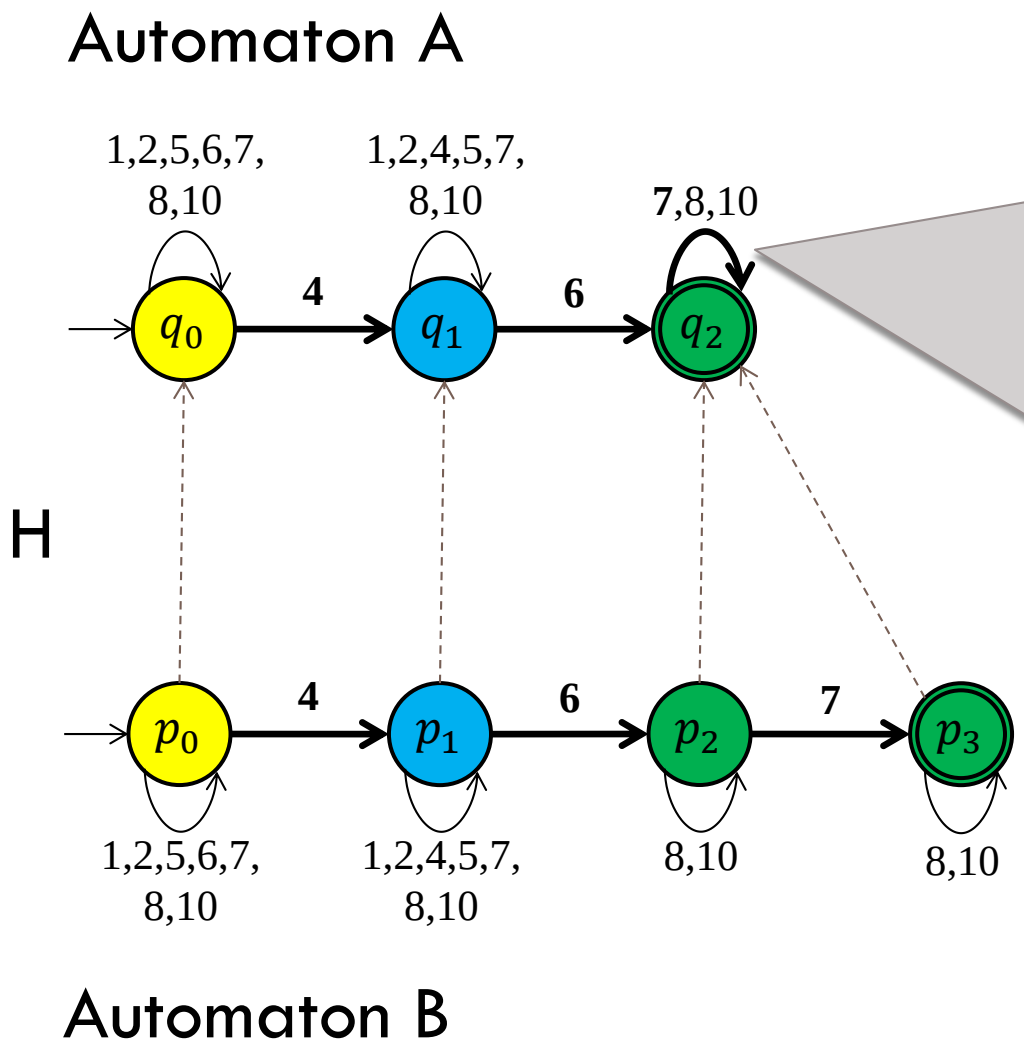
A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

Shared Parts: Simulation Relation



Shared Parts: Simulation Relation



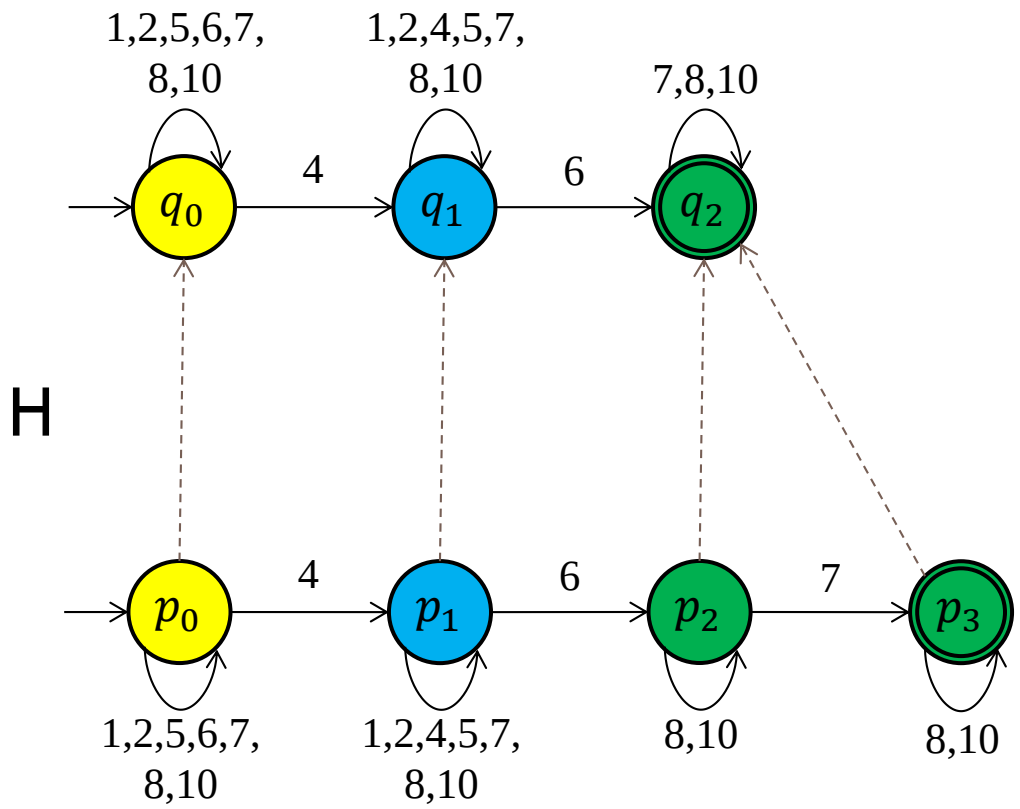
For $(p, q) \in H$:

For each transition sequence starting from p

there is a corresponding transition sequence starting in q

Shared Parts: Simulation Relation

Automaton A

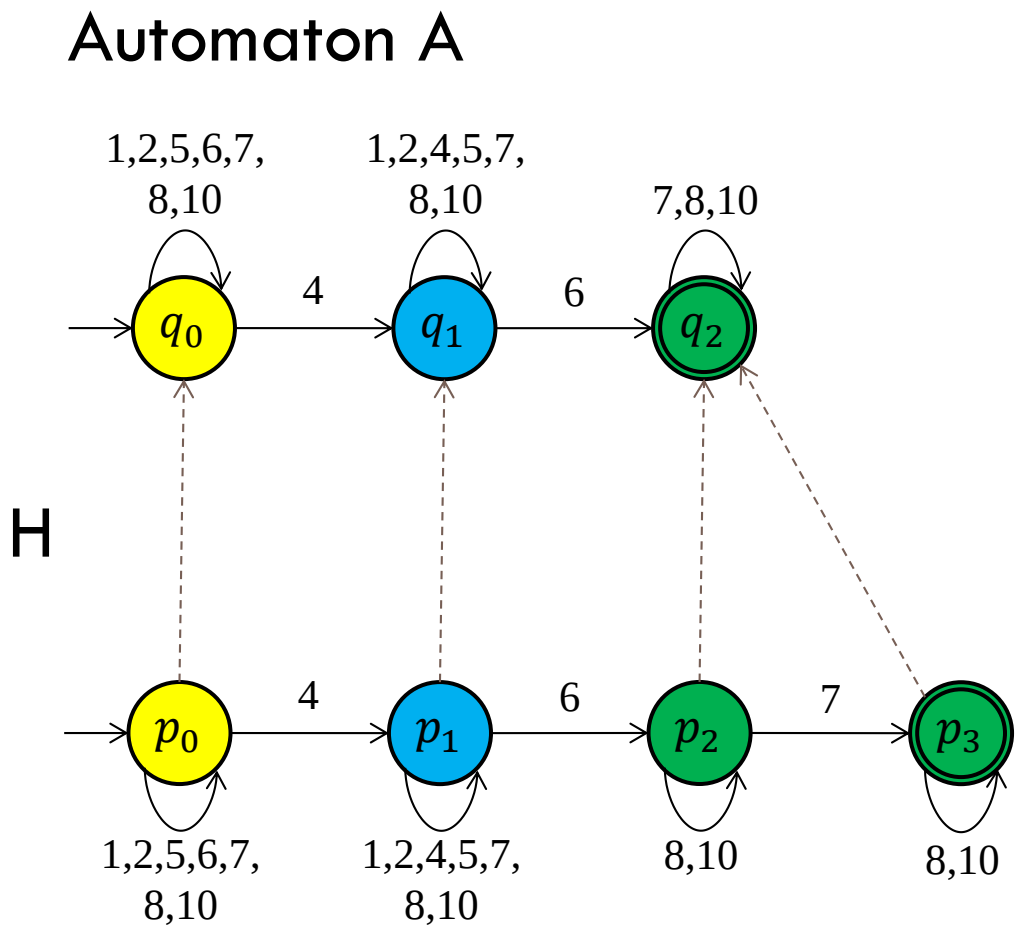


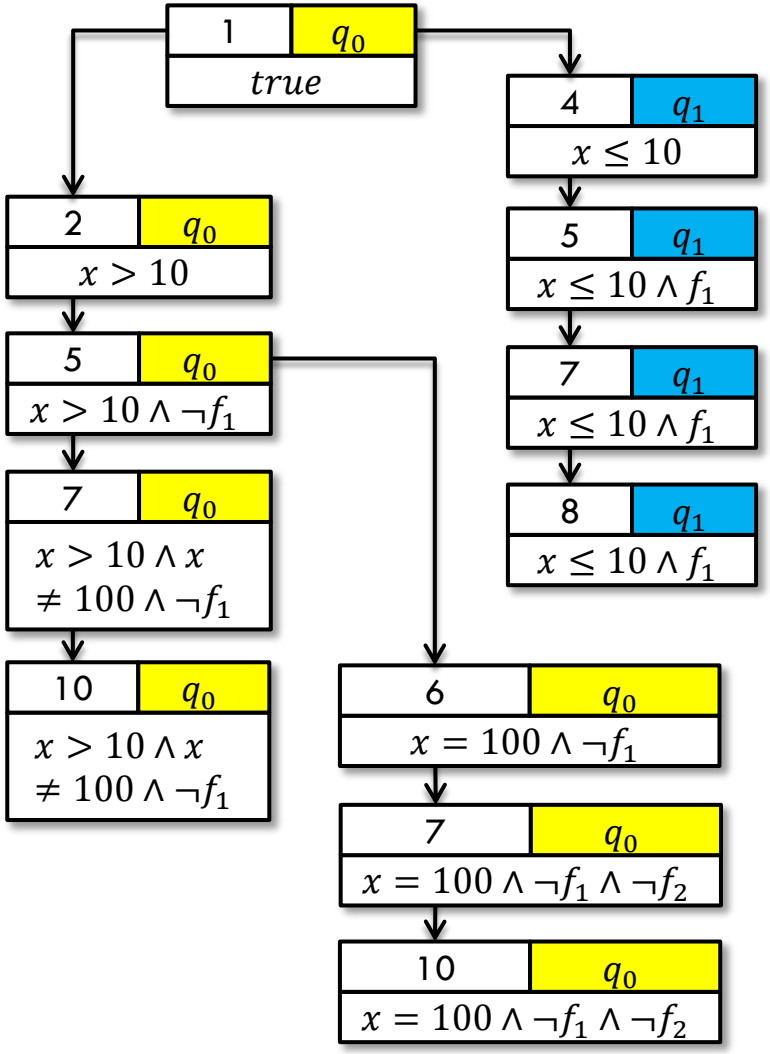
Automaton B

A *simulation relation H* associates states of B with states of A s.t. for $(p, q) \in H$ holds:

For each transition (p, a, p') in B there is a transition (q, a, q') in A such that $(p', q') \in H$

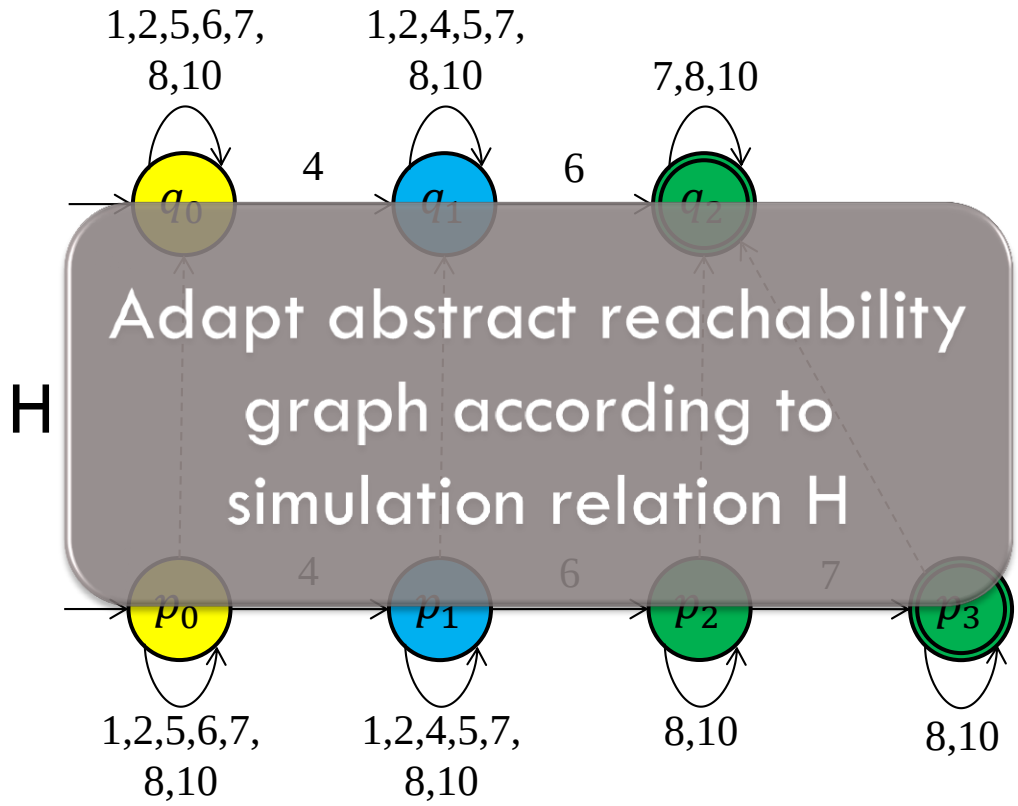
Reuse Information for Shared Parts



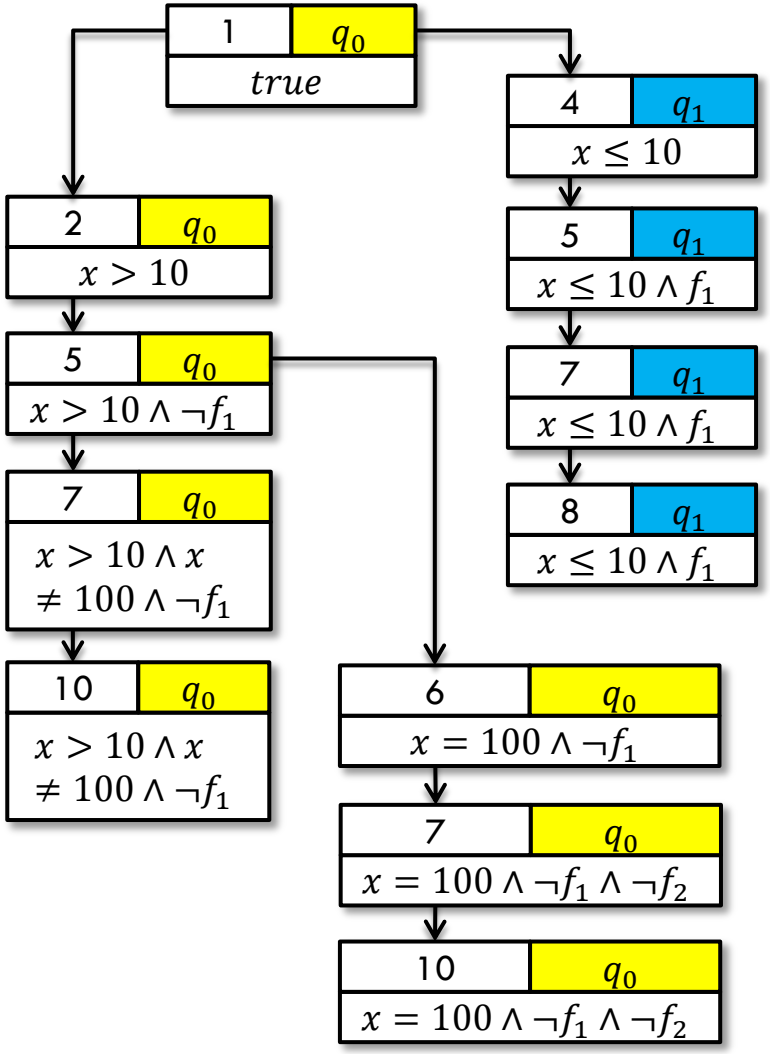


Reuse Information for Shared Parts

Automaton A

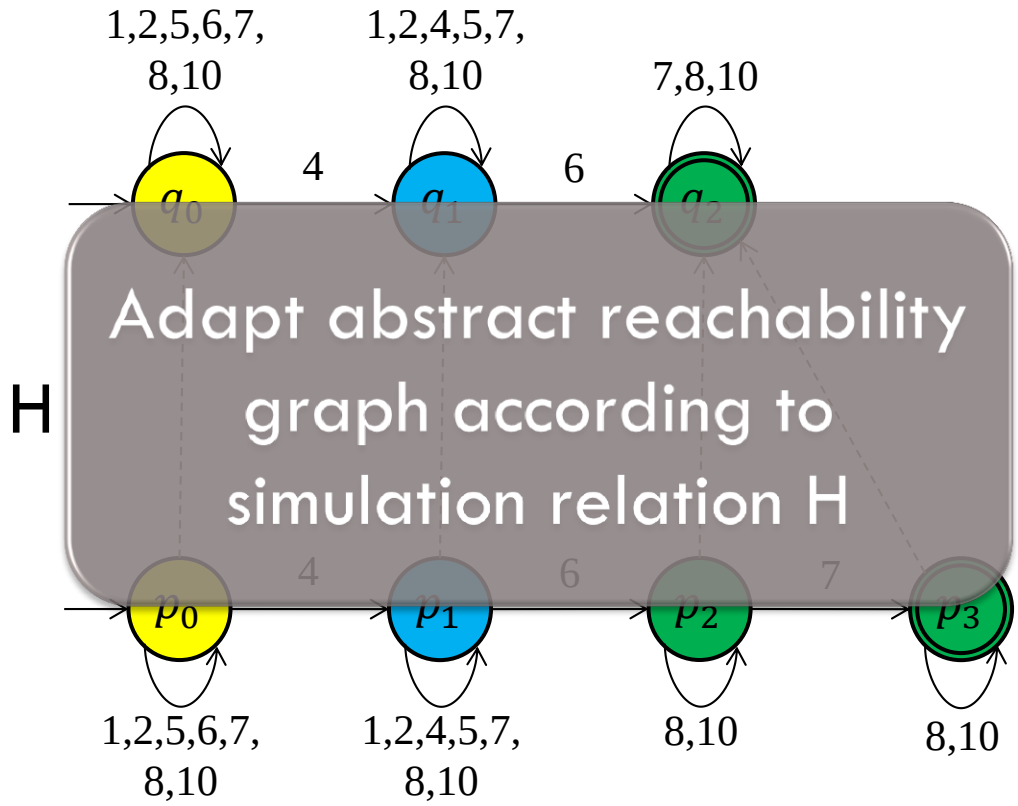


Automaton B

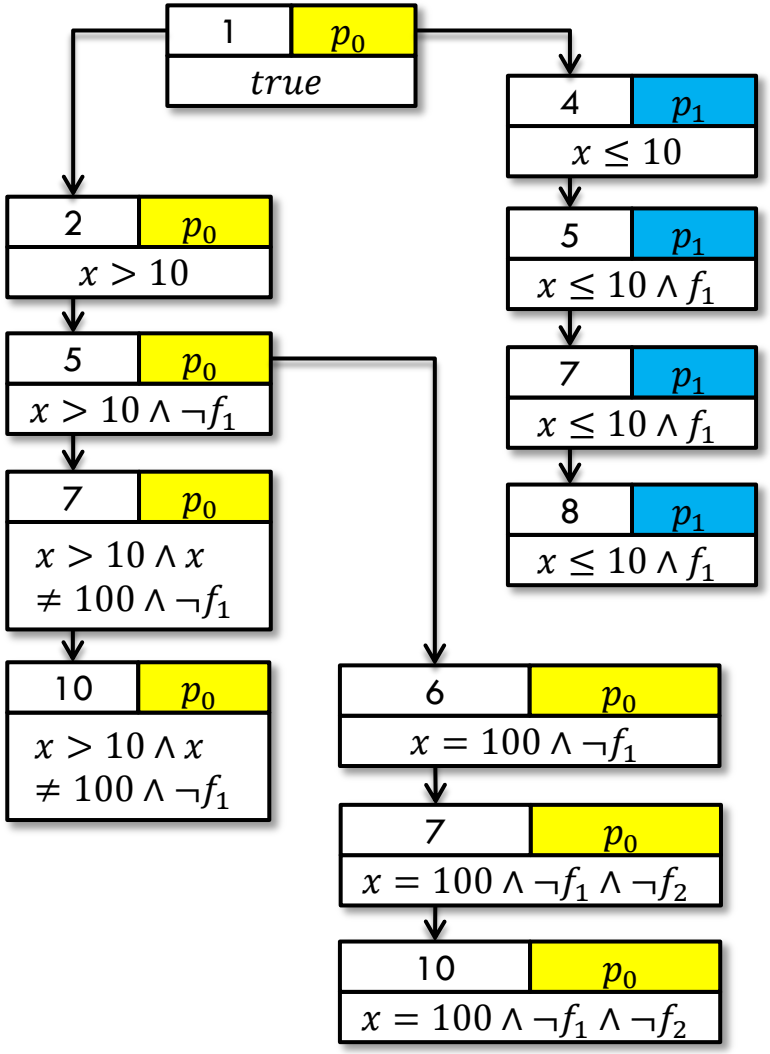


Reuse Information for Shared Parts

Automaton A

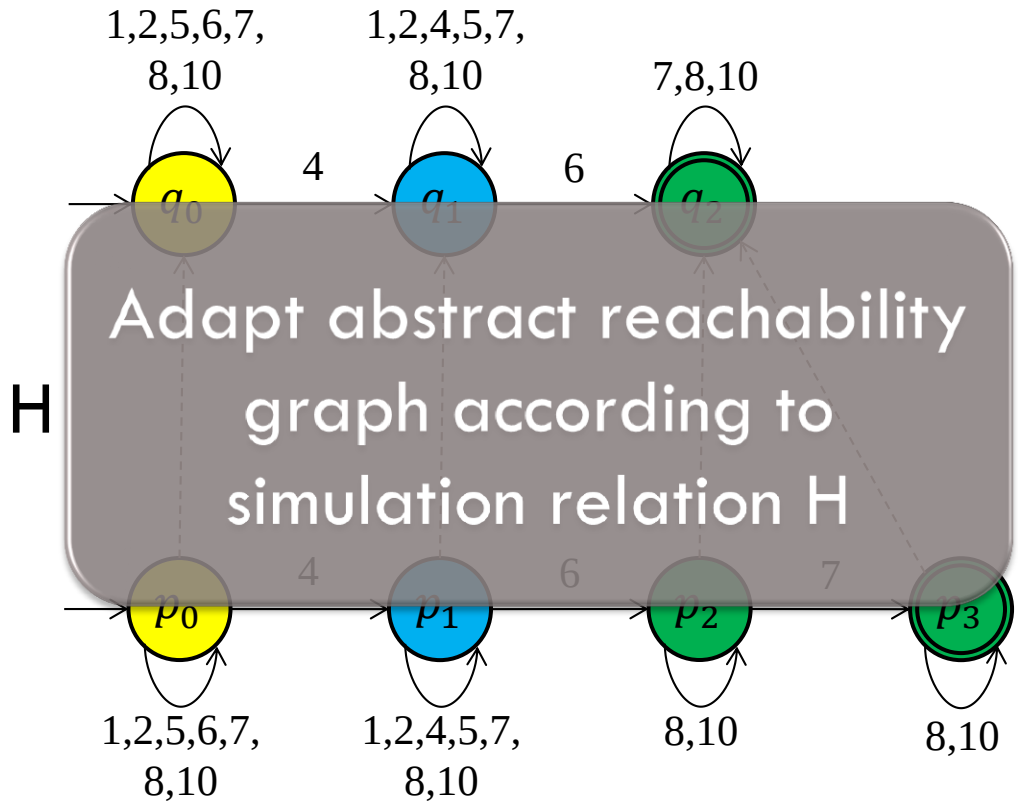


Automaton B

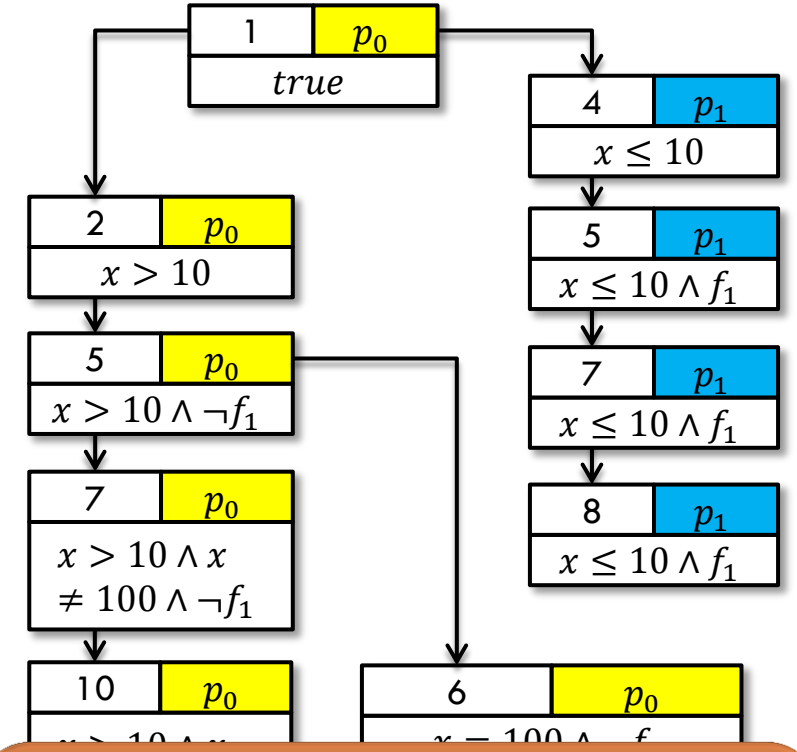


Reuse Information for Shared Parts

Automaton A



Adapt abstract reachability graph according to simulation relation H

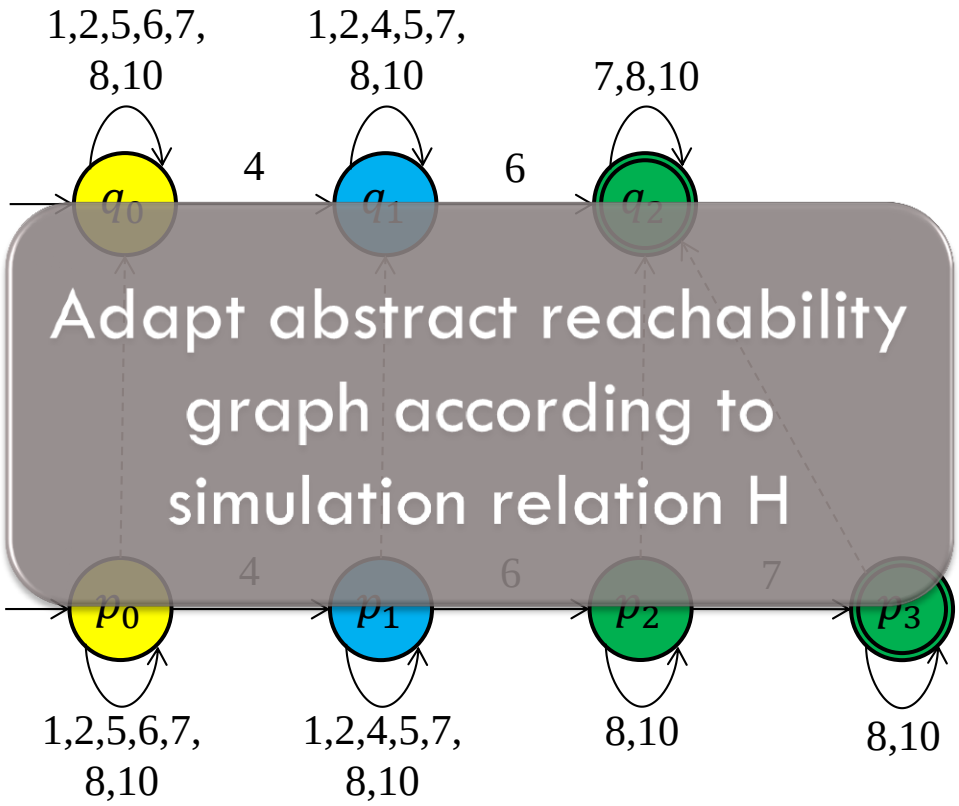


There is no abstract state labeled with p_3!

Automaton B

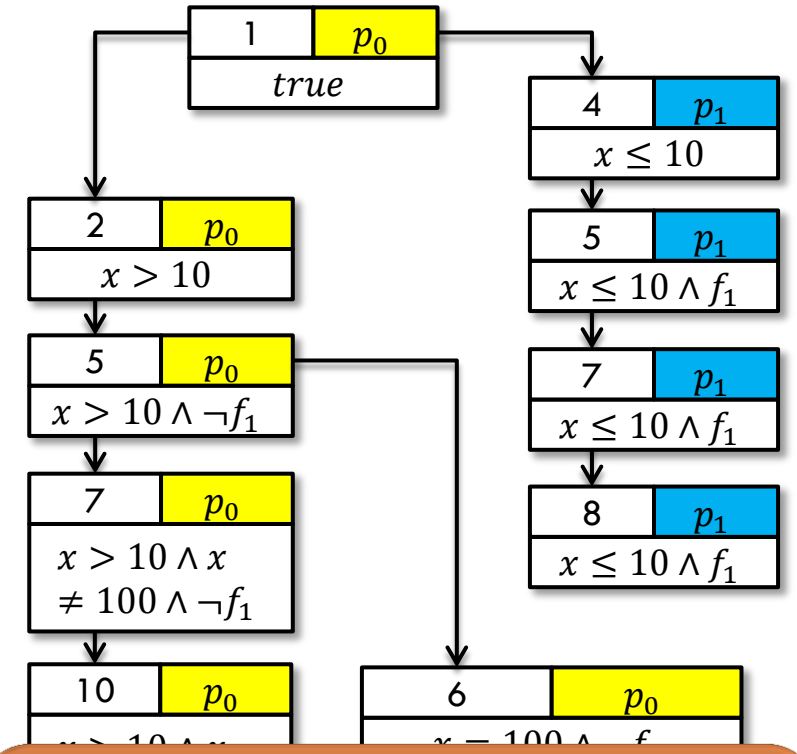
Reuse Information for Shared Parts

Automaton A



Automaton B

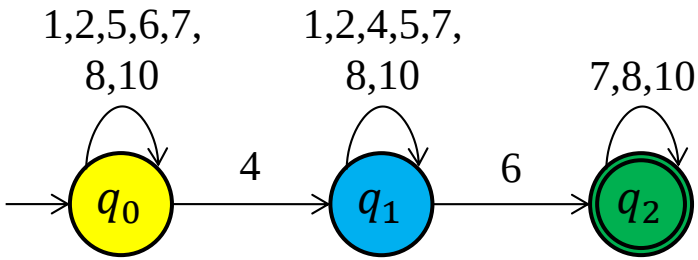
Unsatisfiable!

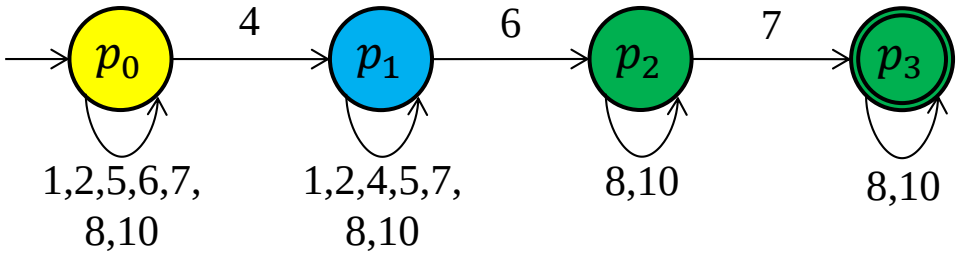


There is no abstract state labeled with p_3!

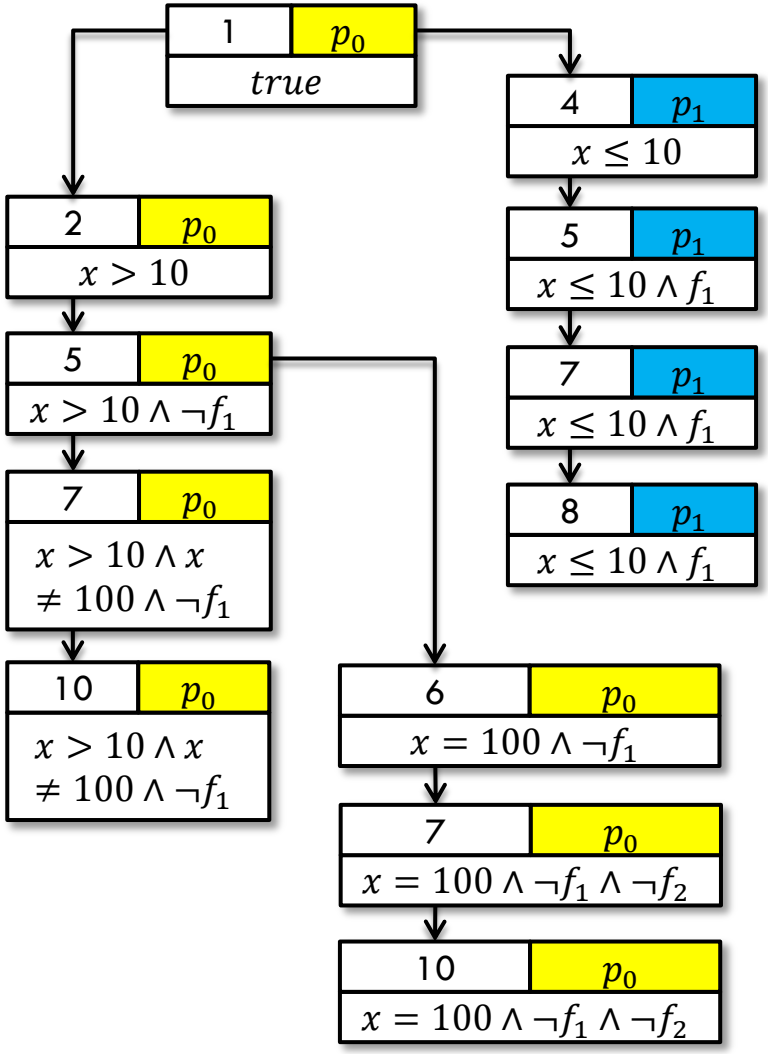
Reuse Information for Shared Parts

Automaton A



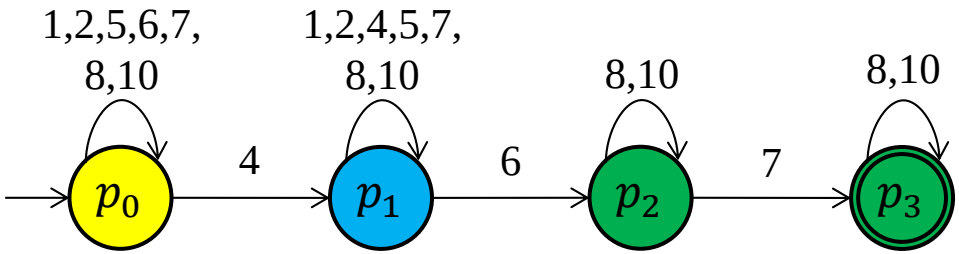


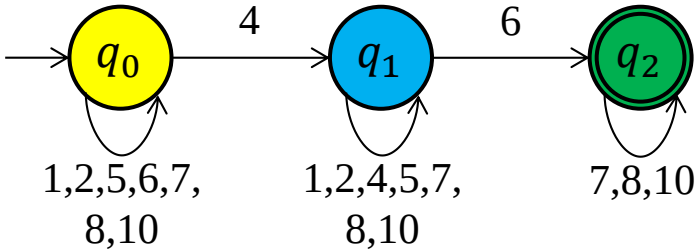
Automaton B



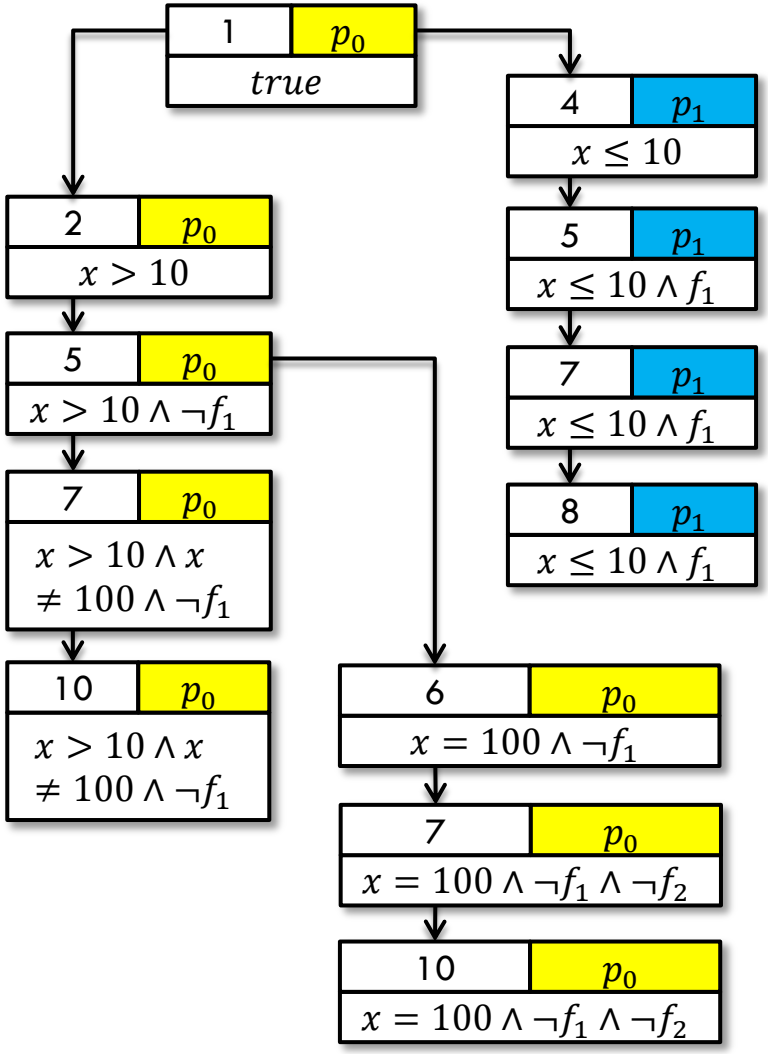
Reuse Information for Shared Parts

Automaton B



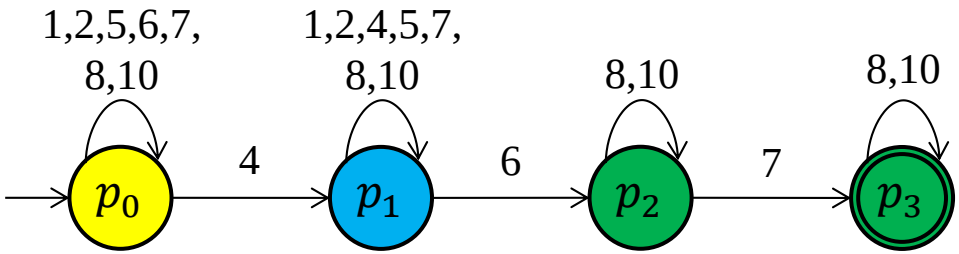


Automaton A

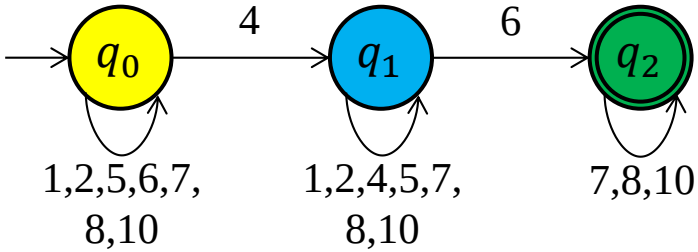


Reuse Information for Shared Parts

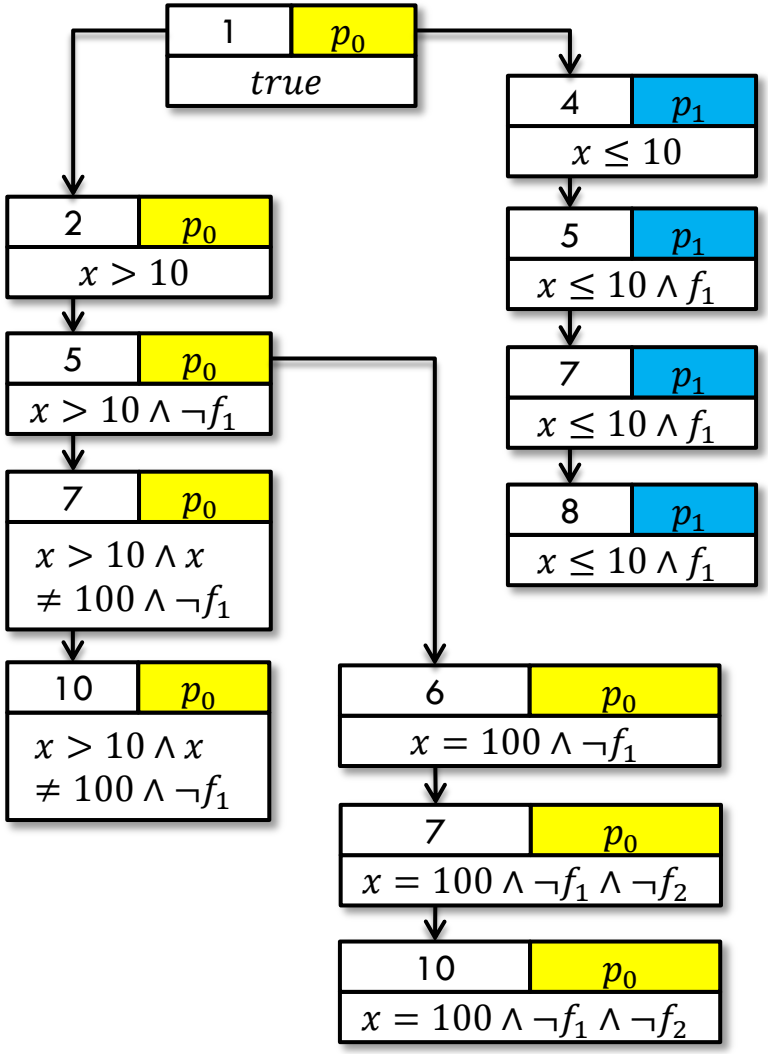
Automaton B



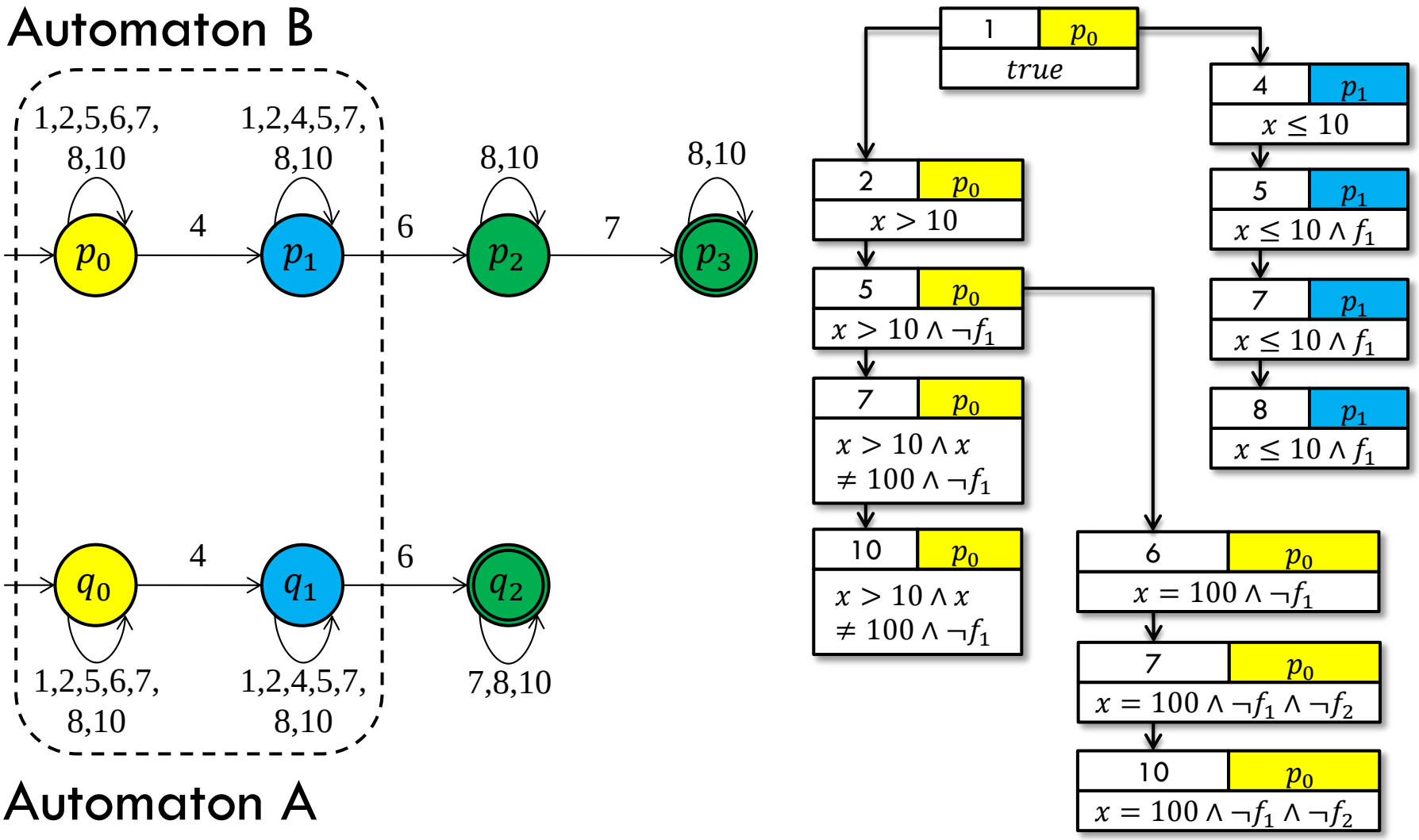
$H = \emptyset$



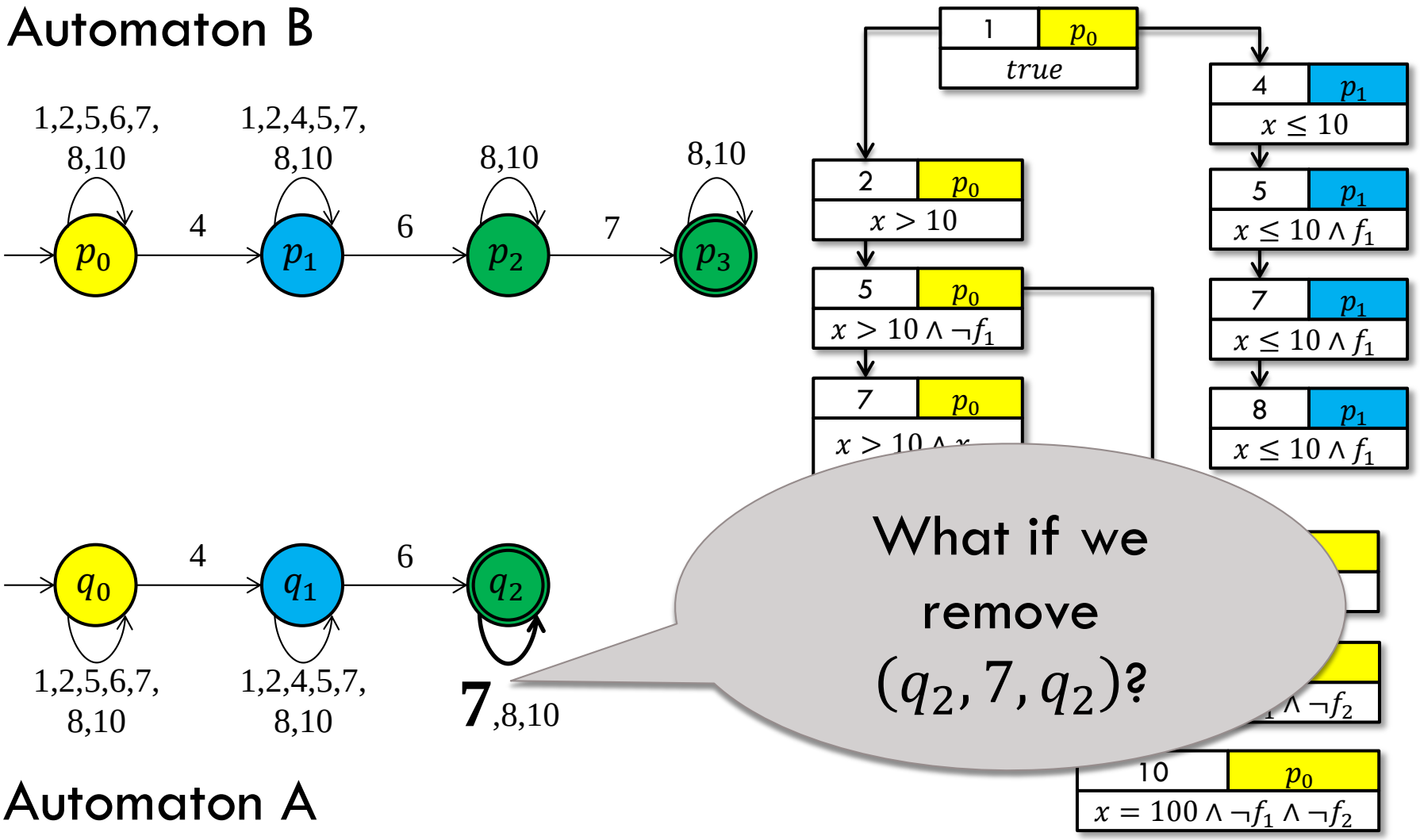
Automaton A



Reuse Information for Shared Parts

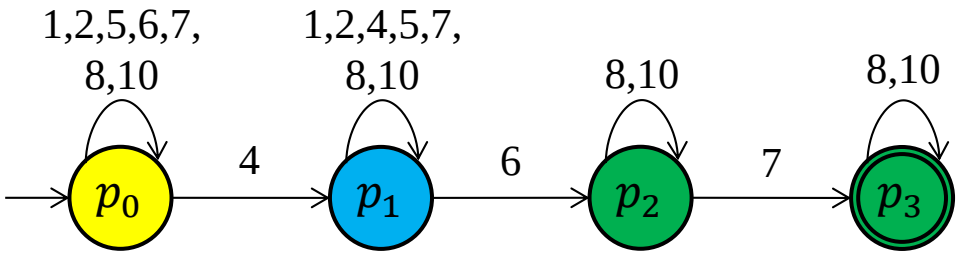


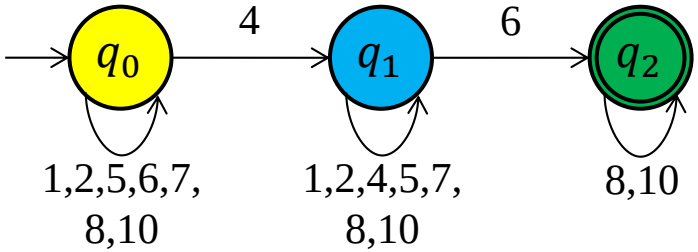
Reuse Information for Shared Parts



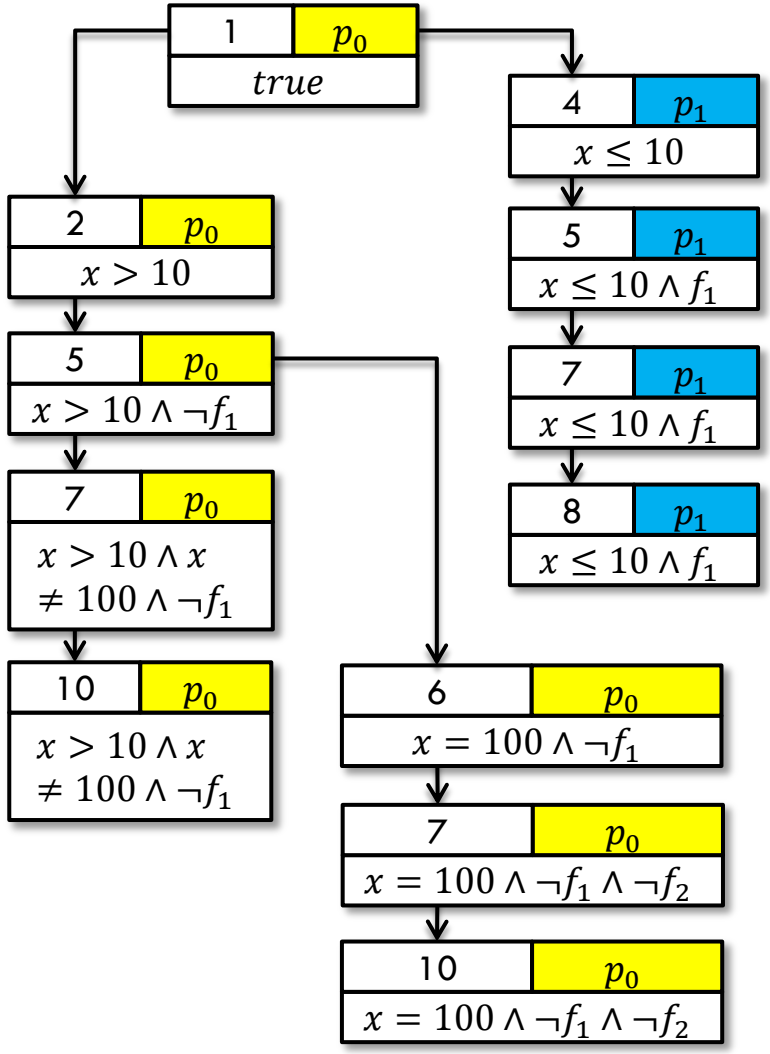
Simulation Modulo $(q_2, 7, q_2)$

Automaton B



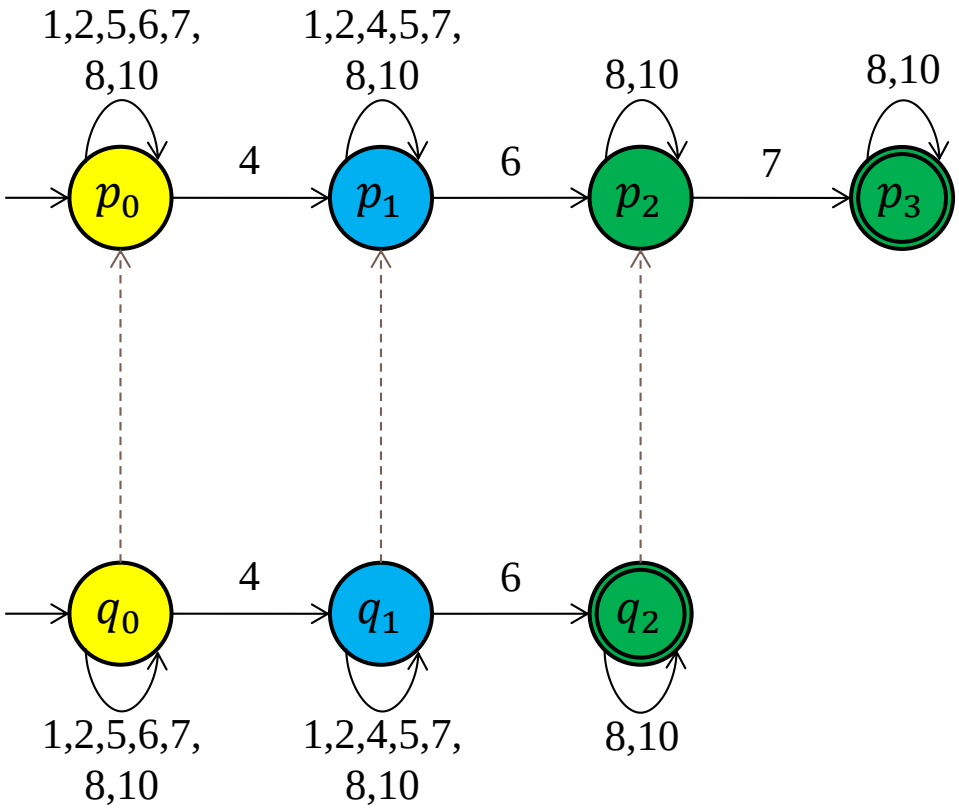


Automaton A‘

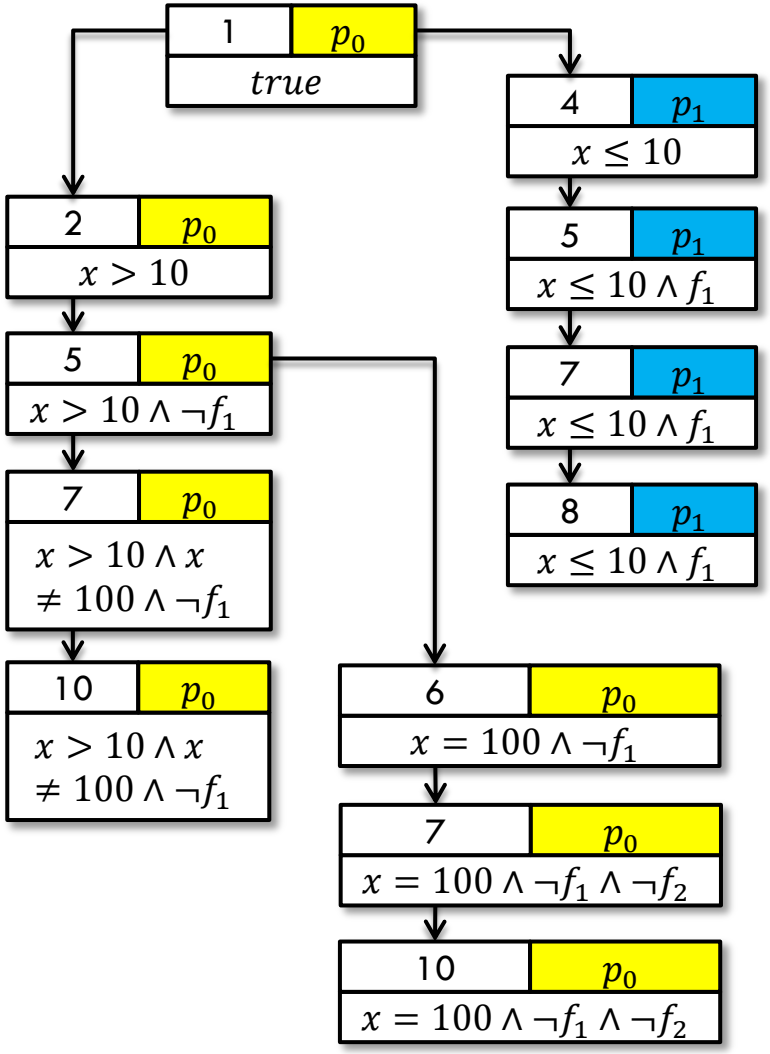


Simulation Modulo $(q_2, 7, q_2)$

Automaton B

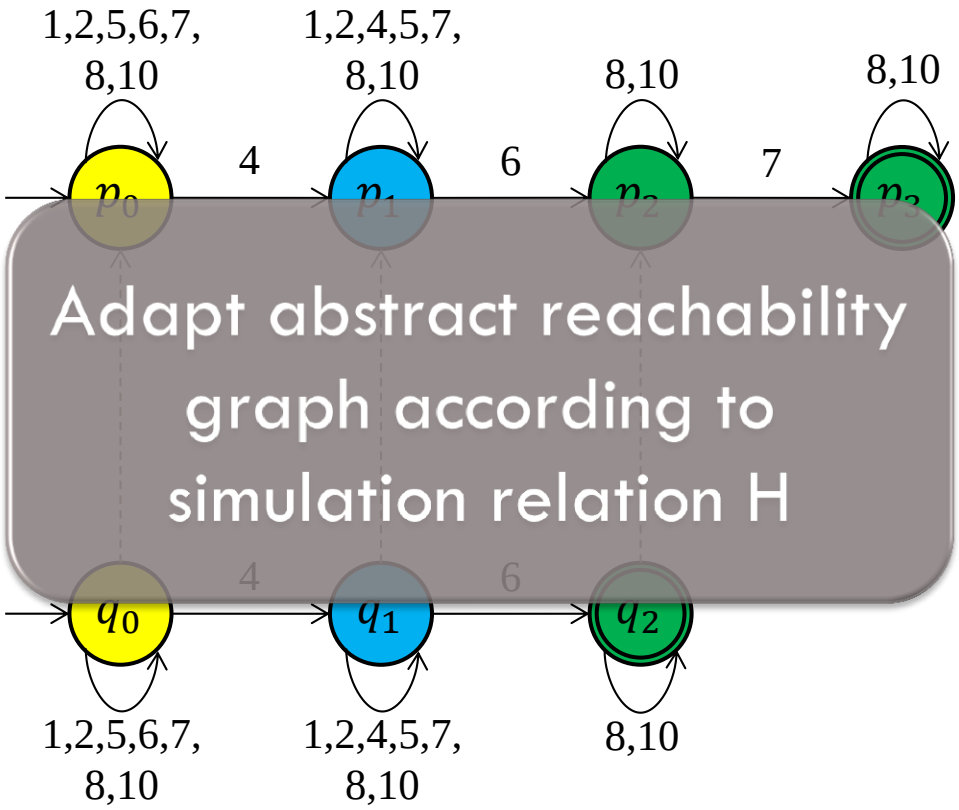


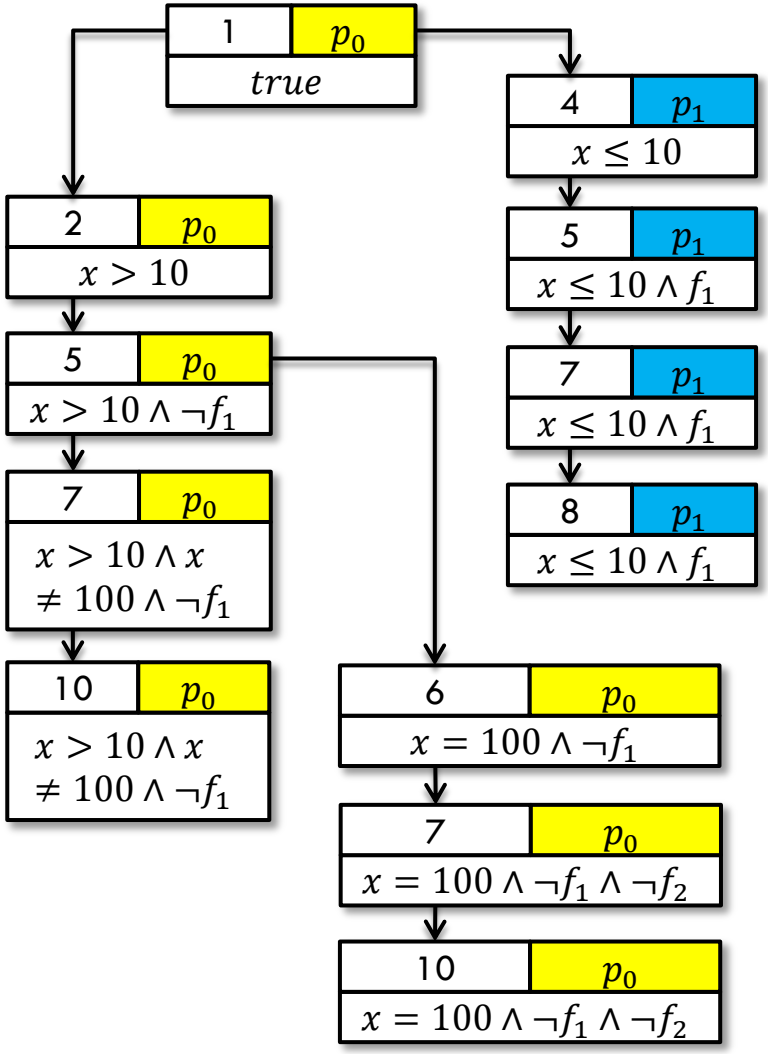
Automaton A‘



Simulation Modulo $(q_2, 7, q_2)$

Automaton B

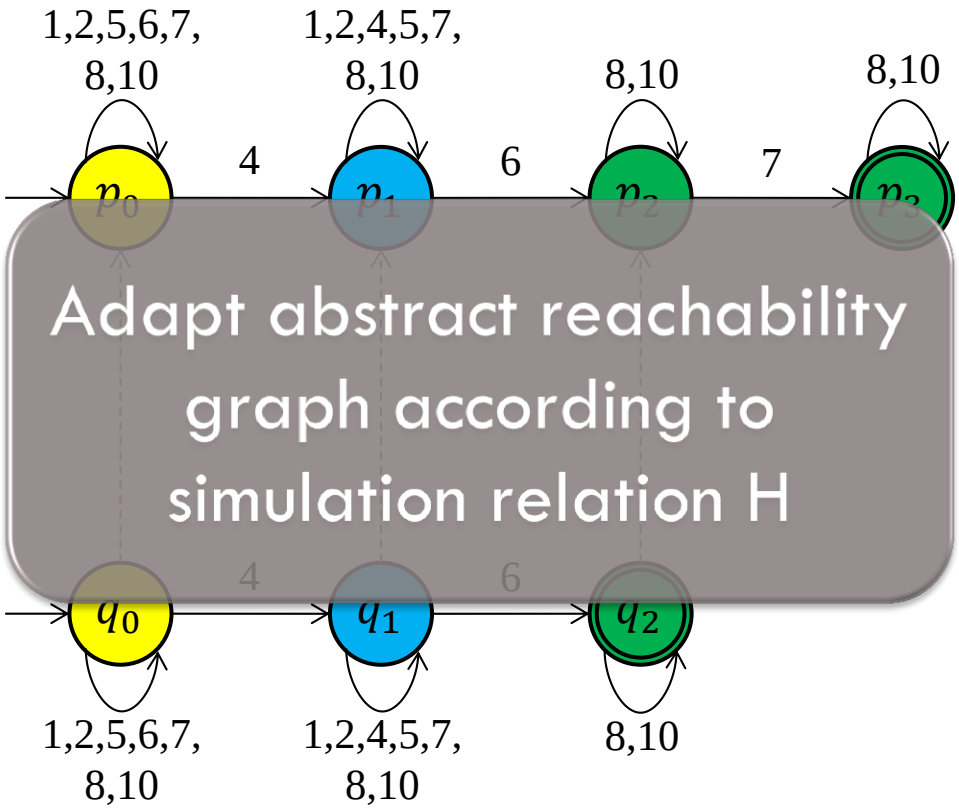


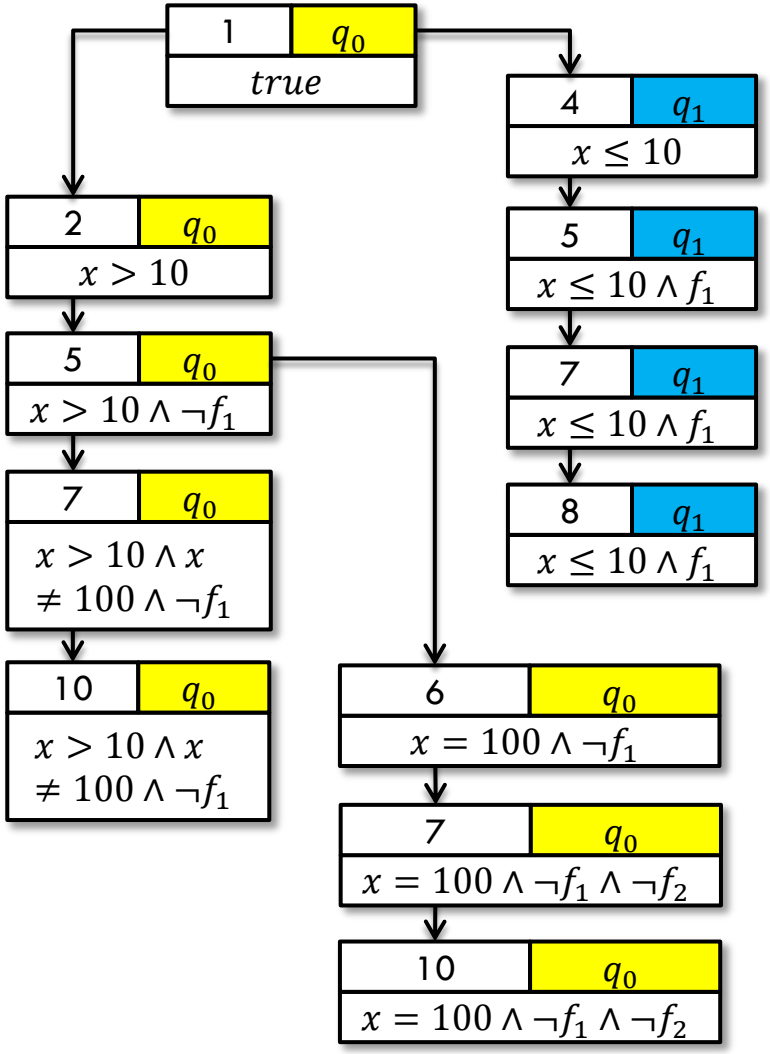


Automaton A‘

Simulation Modulo $(q_2, 7, q_2)$

Automaton B

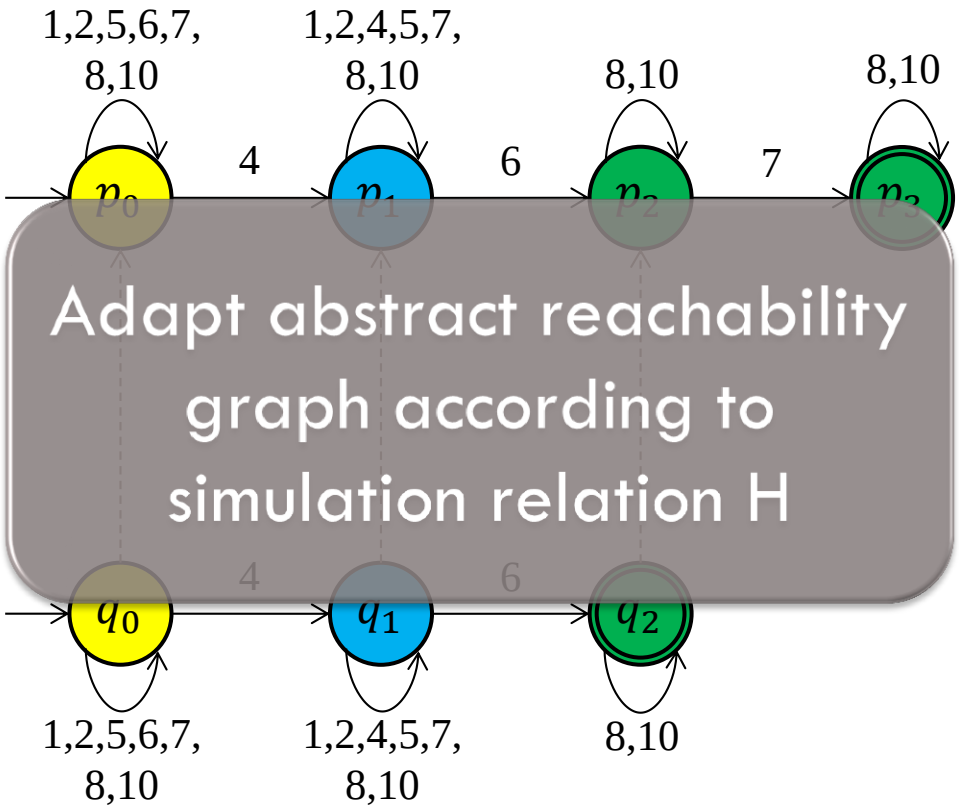




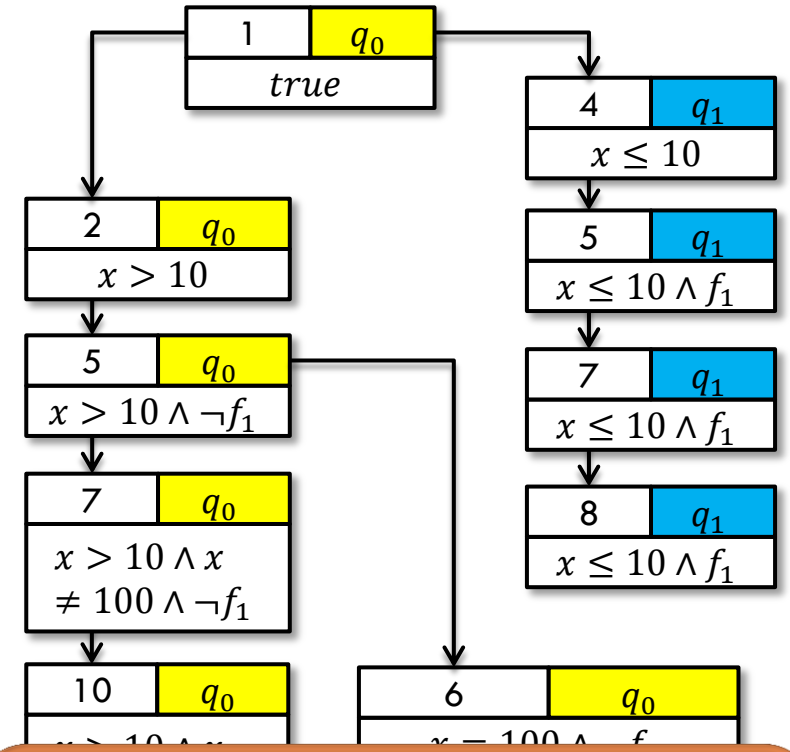
Automaton A‘

Simulation Modulo $(q_2, 7, q_2)$

Automaton B



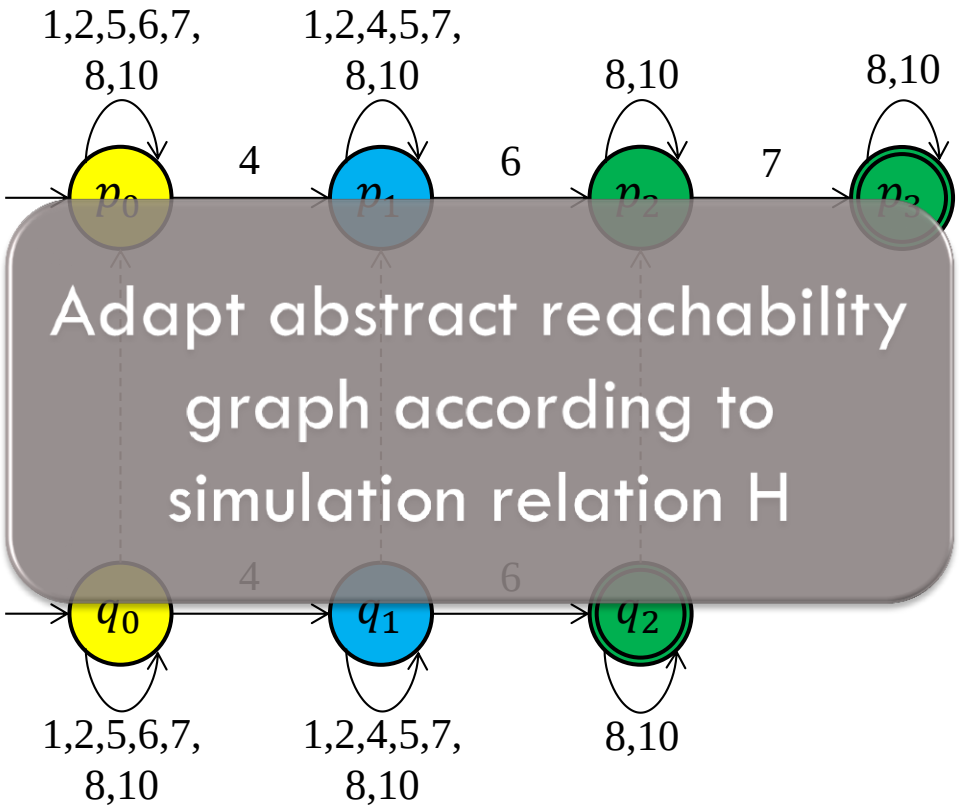
Automaton A‘

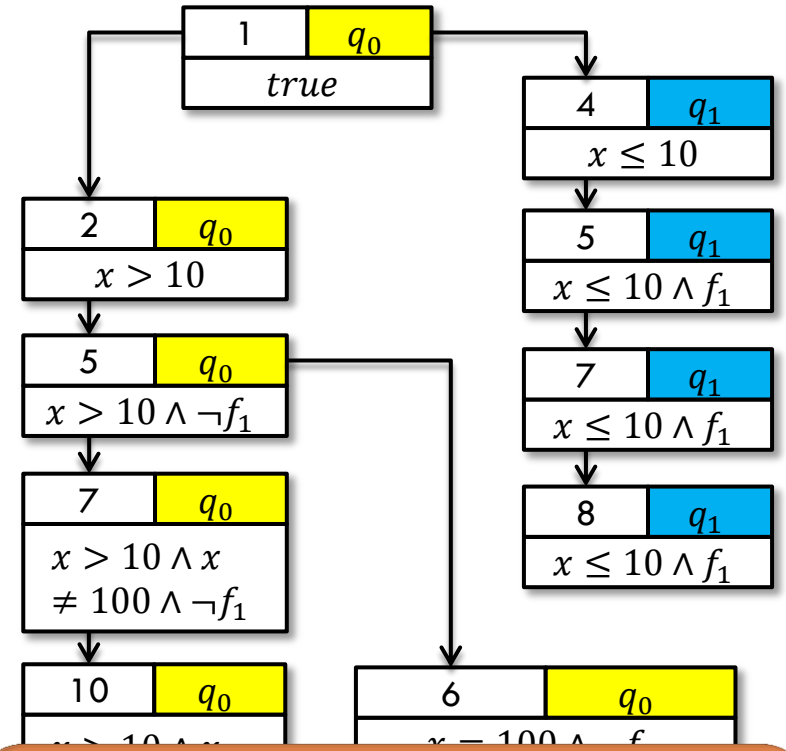


There is no abstract state labeled with q_2!

Simulation Modulo $(q_2, 7, q_2)$

Automaton B





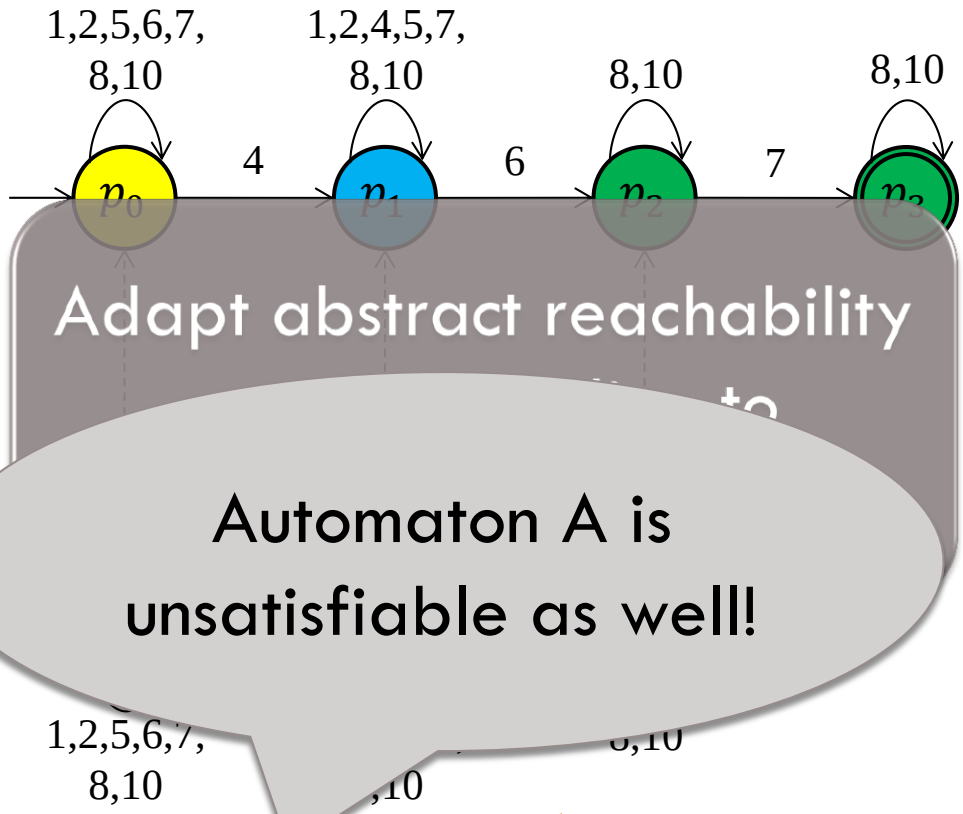
There is no abstract state labeled with q_2!

Automaton A‘

Unsatisfiable!

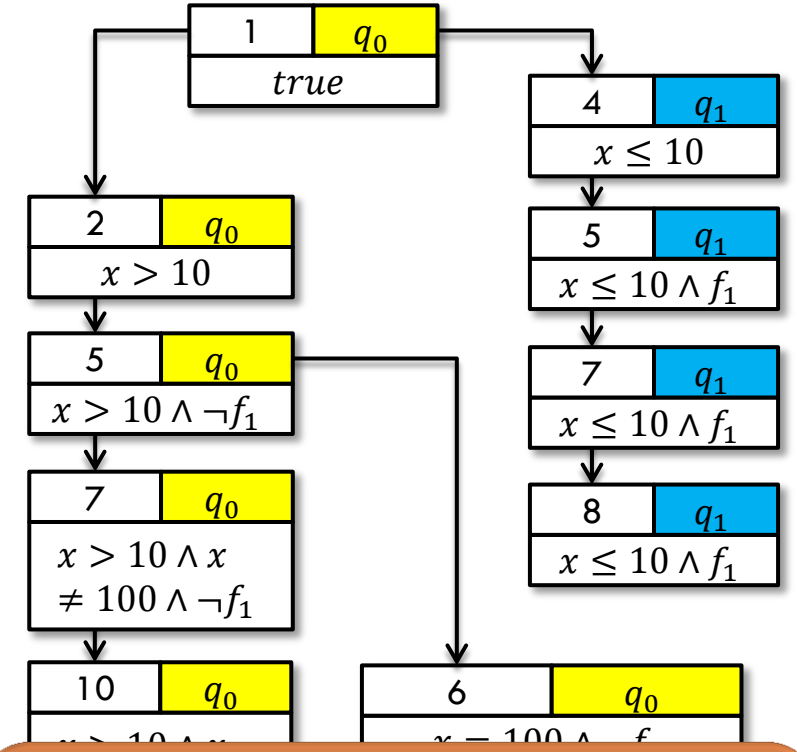
Simulation Modulo $(q_2, 7, q_2)$

Automaton B



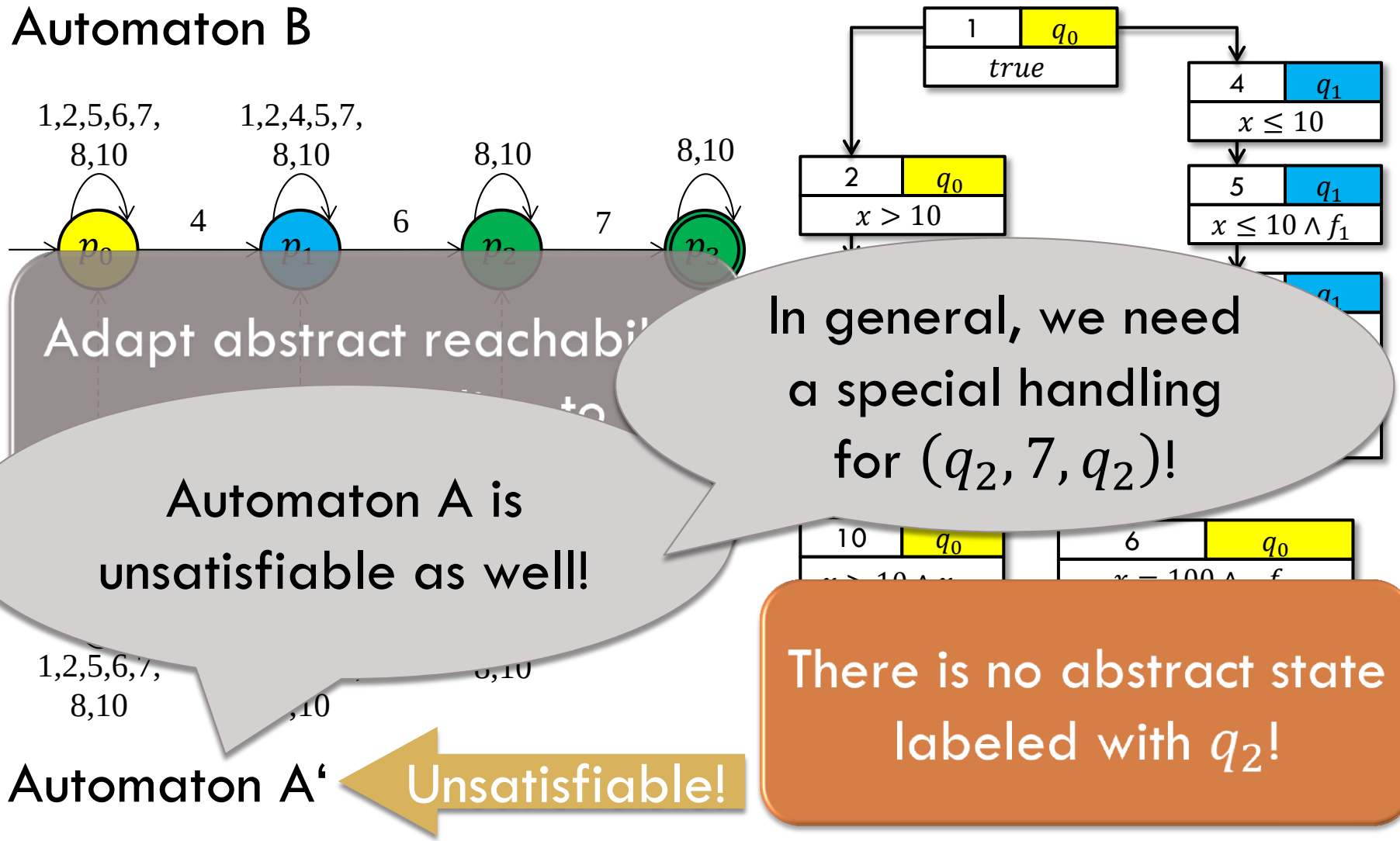
Automaton A'

Unsatisfiable!



There is no abstract state labeled with q_2!

Simulation Modulo $(q_2, 7, q_2)$

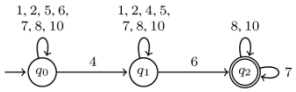


Simulation Modulo X

- Identify a set X of automaton transitions (of the new test goal automaton)
- Exclude X when computing a simulation relation between the old automaton and the new automaton
- Adapt abstract reachability graph
- Reinvoke the reachability analysis for automaton transitions in X
- **The choice of X is not unique!**

State Space Exploration

Automaton



Initial Abstract State

Information Reuse

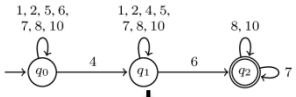
Stored ARGs

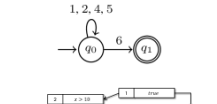
Reachability Analysis

State Space Exploration

Initial Abstract State

Automaton





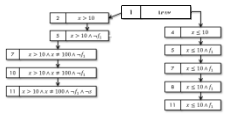
Information Reuse

Stored ARGs

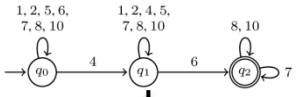
Reachability Analysis

State Space Exploration

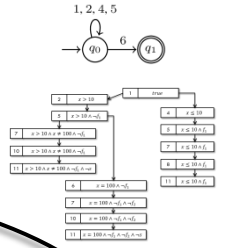
Initial Abstract State



Automaton



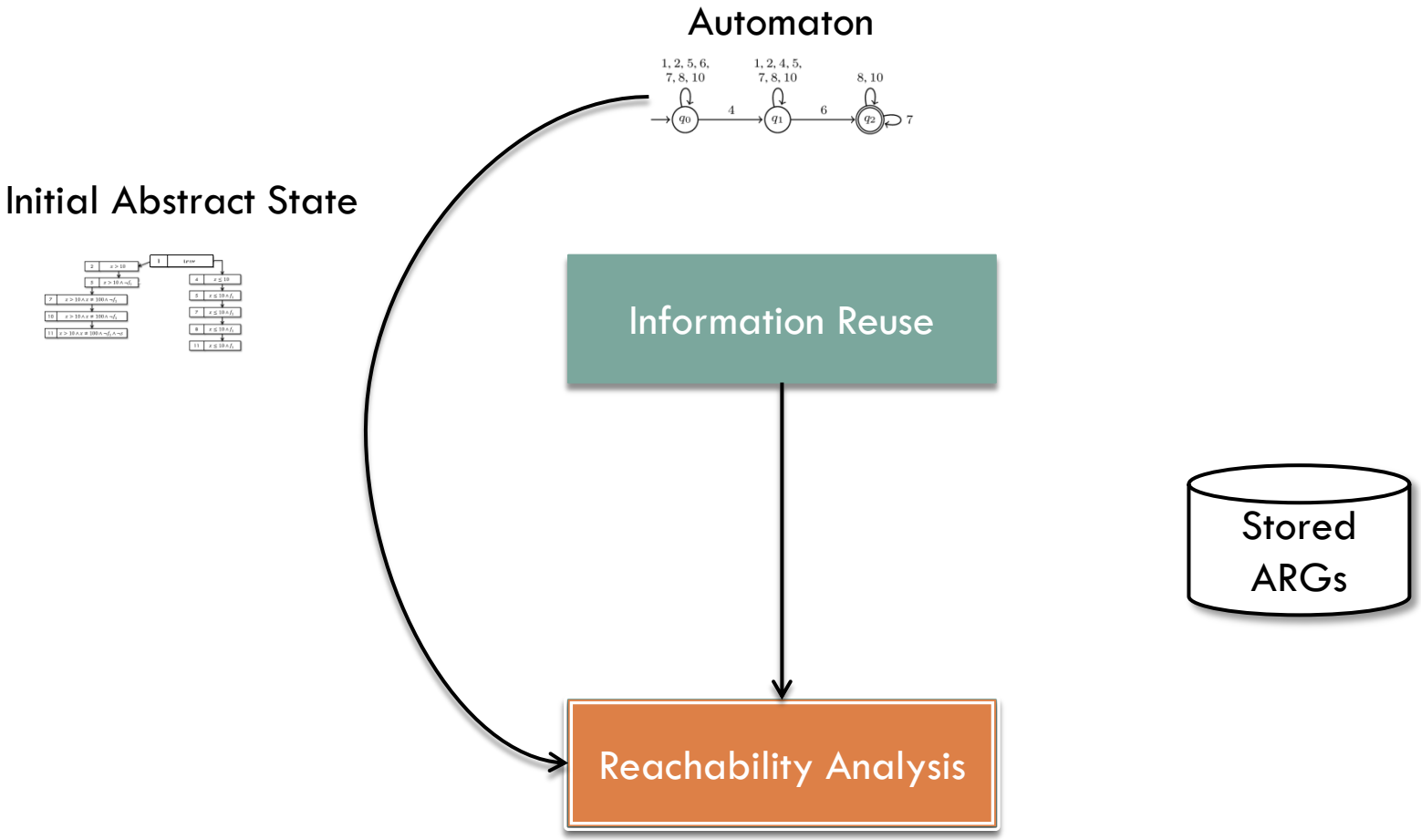
Information Reuse



Stored ARGs

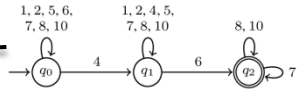
Reachability Analysis

State Space Exploration



State Space Exploration

Automaton



Initial Abstract State

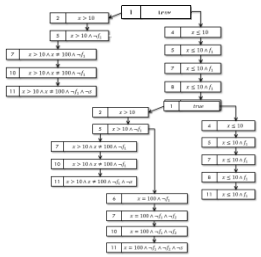
Information Reuse

Reachability Analysis

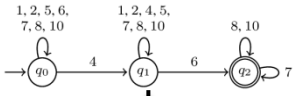
Stored ARGs

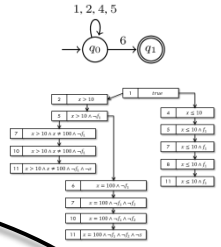
State Space Exploration

Initial Abstract State



Automaton





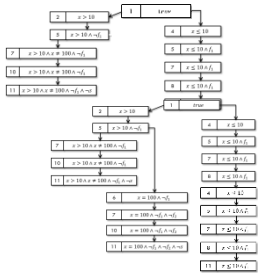
Information Reuse

Stored ARGs

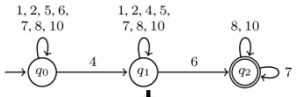
Reachability Analysis

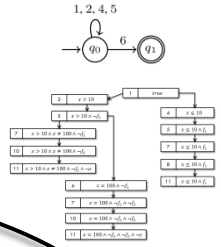
State Space Exploration

Initial Abstract State



Automaton





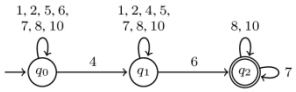
Information Reuse

Stored ARGs

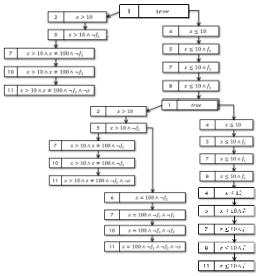
Reachability Analysis

State Space Exploration

Automaton



Initial Abstract State



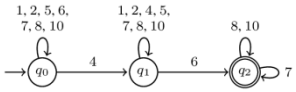
Information Reuse

Stored ARGs

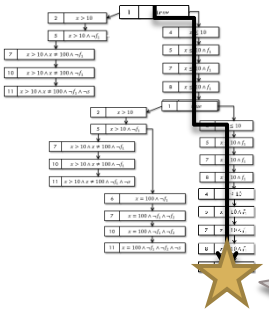
Reachability Analysis

State Space Exploration

Automaton



Initial Abstract State



Information Reuse

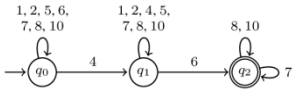
Witness of Satisfiability

Stored ARGs

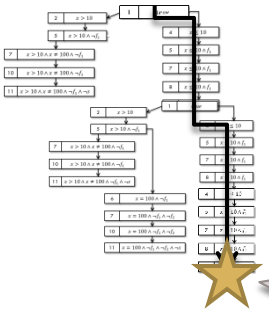
Reachability Analysis

State Space Exploration

Automaton



Initial Abstract State



Derive Inputs

Information Reuse

Witness of Satisfiability

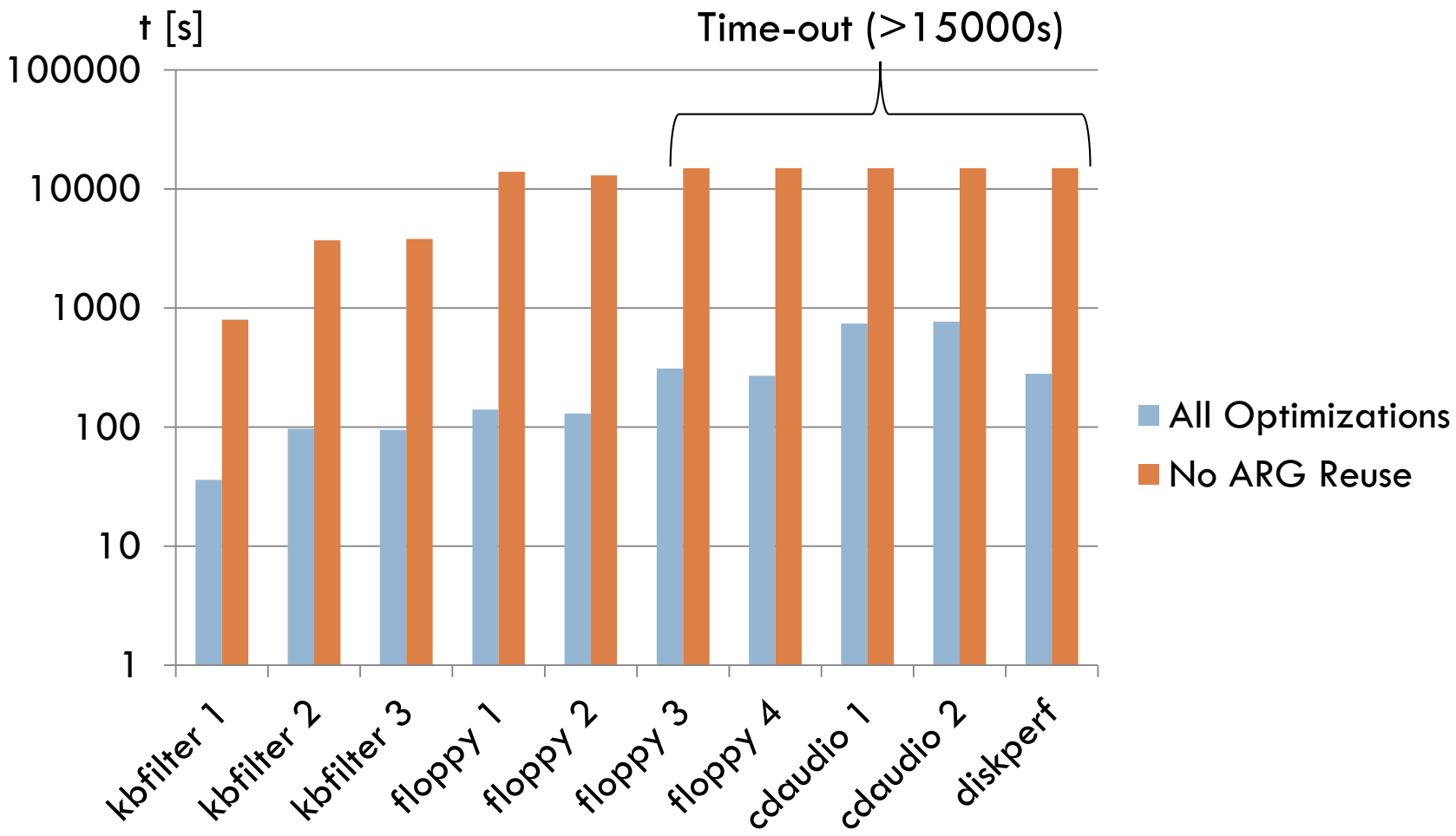
Stored ARGs

Reachability Analysis

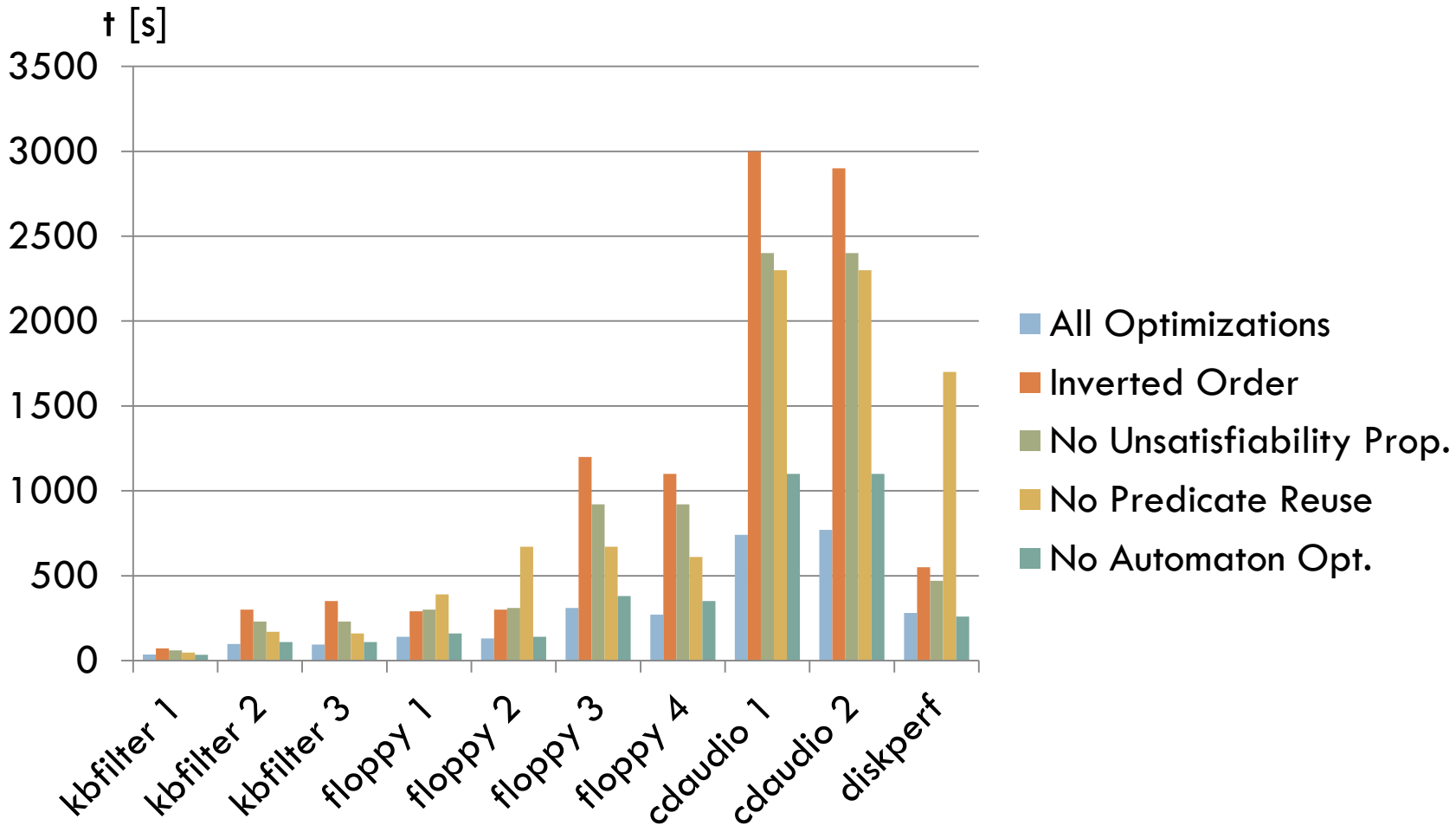
Experiments

- Variants of Basic Block Coverage:
  - BB: Cover each basic block
  - BB^2: Cover each pair of basic blocks
  - BB^3: Cover each triple of basic blocks
- Bounded-Path Coverage

Experiments (BB^2 Coverage)



Experiments (BB^2 Coverage)



Conclusion

- *Simulation Modulo X*: Reuse of reachability information based on relations between test-goal automata.
- *Multi-Goal Reachability Analysis*

Conclusion

- *Simulation Modulo X*: Reuse of reachability information based on relations between test-goal automata.
- *Multi-Goal Reachability Analysis*

Thank you!