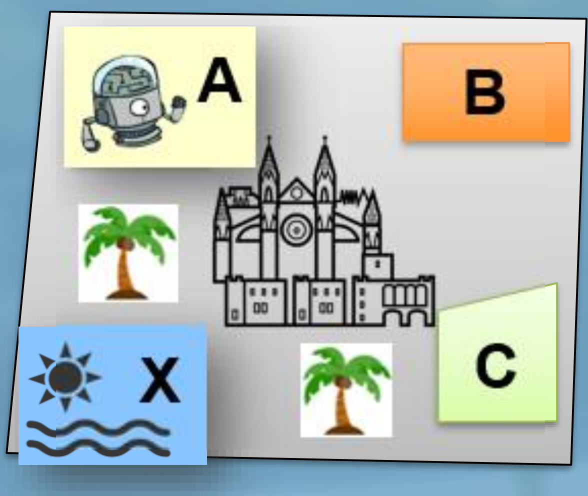
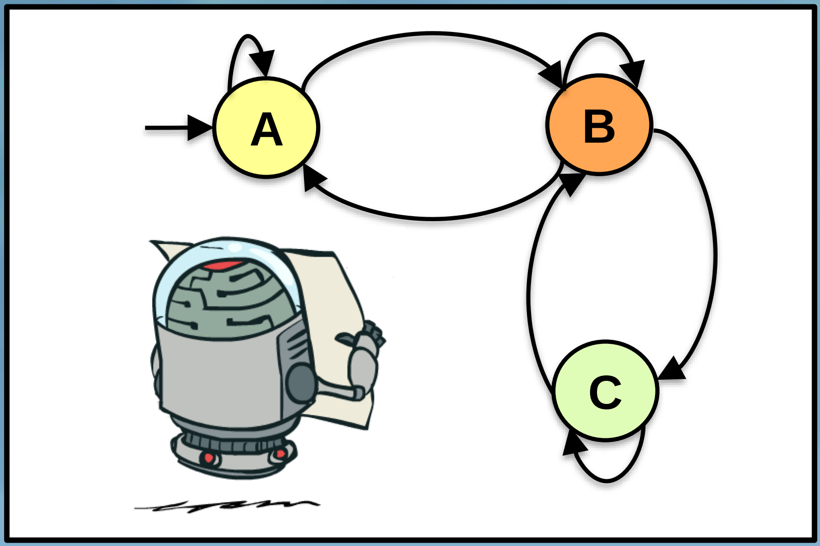
Graz University of Technology
Institute for Applied Information Processing and Communications

Automata and LTL Model Checking
Part-2

Bettina Könighofer





Model Checking SS21

May 27th 2021

Homework 8
Intersection of Büchi Automata

- Question
  - In every interval we first wait for $\mathbf{F}_1$ and then wait for $\mathbf{F}_2$.
  - We ignore accepting states that don't appear in this order.
  - Might we miss accepting paths in $\mathcal{B}$?

$\mathcal{B} = (\Sigma, Q, \Delta, Q^0, F)$ s.t. $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$ is defined as follows:

- $Q = Q_1 \times Q_2 \times \{0, 1, 2\}$
- $Q^0 = Q_1^0 \times Q_2^0 \times \{0\}$
- $F = Q_1 \times Q_2 \times \{2\}$

$((q_1,q_2,x), a, (q'_1,q'_2,x')) \in \Delta \Leftrightarrow$

(1) $(q_1,a,q'_1) \in \Delta_1$ and $(q_2,a,q'_2) \in \Delta_2$ and

(2) If $x=0$ and $q'_1 \in F_1$ then $x'=1$
If $x=1$ and $q'_2 \in F_2$ then $x'=2$
If $x=2$ then $x'=0$
Else, $x'=x$

SCOS
Secure & Correct Systems

Homework 8
Intersection of Büchi Automata

- Question
  - In every interval we first wait for $\mathbf{F}_1$ and then wait for $\mathbf{F}_2$.
  - We ignore accepting states that don’t appear in this order.
  - Might we miss accepting paths in $\mathcal{B}$?

- Answer
  - No. Since on an accepting path there are infinitely many of those, ignoring finite number of them in each interval will still lead us to the conclusion that the run is accepting

Homework 8 - Intersection of Büchi Automata

- Question
  - How do we define the transition relation for $\mathcal{B}$, if x is over {0,1} only?

With x over {0,1,2} we had:

$\mathcal{B} = (\mathbf{\Sigma},\mathbf{Q},\mathbf{\Delta},\mathbf{Q}^0,\mathbf{F})$ s.t. $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$ is defined as follows:

- $\mathbf{Q} = \mathbf{Q}_1 \times \mathbf{Q}_2 \times \{0, 1, 2\}$
- $\mathbf{Q}^0 = \mathbf{Q}_1^0 \times \mathbf{Q}_2^0 \times \{0\}$
- $\mathbf{F} = \mathbf{Q}_1 \times \mathbf{Q}_2 \times \{2\}$

$((q_1,q_2,x), a, (q'_1,q'_2,x')) \in \mathbf{\Delta} \Leftrightarrow$

(1) $(q_1,a,q'_1) \in \mathbf{\Delta}_1$ and $(q_2,a,q'_2) \in \mathbf{\Delta}_2$ and

(2) If $x=0$ and $q'_1 \in \mathbf{F}_1$ then $x'=1$
If $x=1$ and $q'_2 \in \mathbf{F}_2$ then $x'=2$
If $x=2$ then $x'=0$
Else, $x'=x$

Homework 8 - Intersection of Büchi Automata

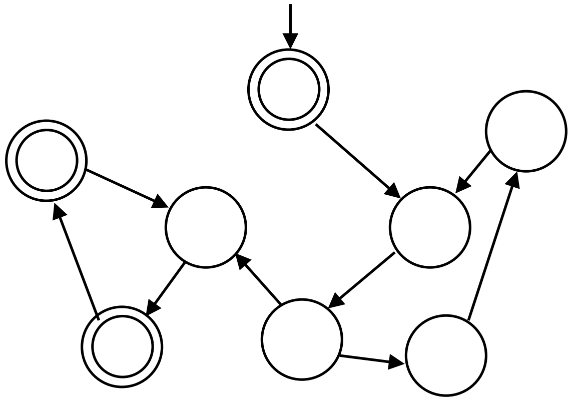
- Question
  - How do we define the transition relation for $\mathcal{B}$, if x is over {0,1} only?
- Answer
  - For $\mathbf{\Delta}$
    - (2) If $x=0$ and $q_1 \in \mathbf{F}_1$ then $x'=1$
      If $x=1$ and $q_2 \in \mathbf{F}_2$ then $x'=0$
      Else, $x'=x$
  - For $\mathbf{F}$
    - $\mathbf{F} = \mathbf{F}_1 \times \mathbf{Q}_2 \times \{0\}$

Outline

- Finite automata on finite words
- Automata on infinite words (Büchi automata)
- Deterministic vs non-deterministic Büchi automata
- Intersection of Büchi automata
- Checking emptiness of Büchi automata
- Generalized Büchi automata
- Automata and Kripke Structures
- Model checking using automata
- Translation of LTL to Büchi automata

Checking for emptiness of $\mathcal{L}(\mathcal{B})$

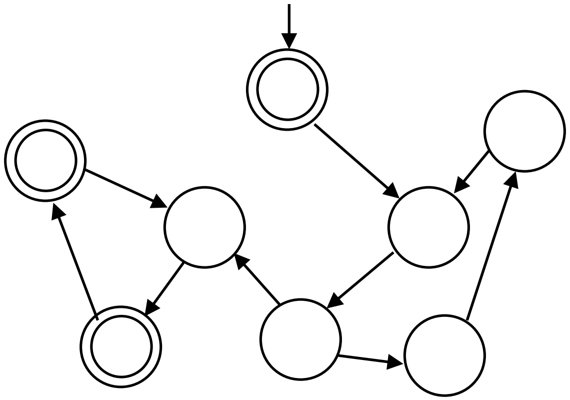
- An infinite run $\boldsymbol{\rho}$ is accepting $\Leftrightarrow$ it visits an accepting state an infinite number of times.
  - $\mathbf{inf}(\boldsymbol{\rho}) \cap \mathbf{F} \neq \emptyset$



Checking for emptiness of $\mathcal{L}(\mathcal{B})$

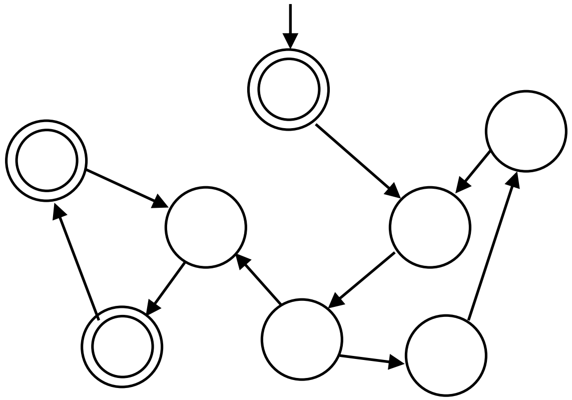
- An infinite run $\boldsymbol{\rho}$ is accepting $\Leftrightarrow$ it visits an accepting state an infinite number of times.
  - $\mathbf{inf(\rho) \cap F \neq \emptyset}$

ToDo How to check for **L(A) = ∅?**



Checking for emptiness of $\mathcal{L}(\mathcal{B})$

- An infinite run $\boldsymbol{\rho}$ is accepting $\Leftrightarrow$ it visits an accepting state an infinite number of times.
  - $\mathbf{inf(\rho) \cap F \neq \emptyset}$
- How to check for **$L(A) = \emptyset$?**
- Find a reachable accepting state on a cycle.



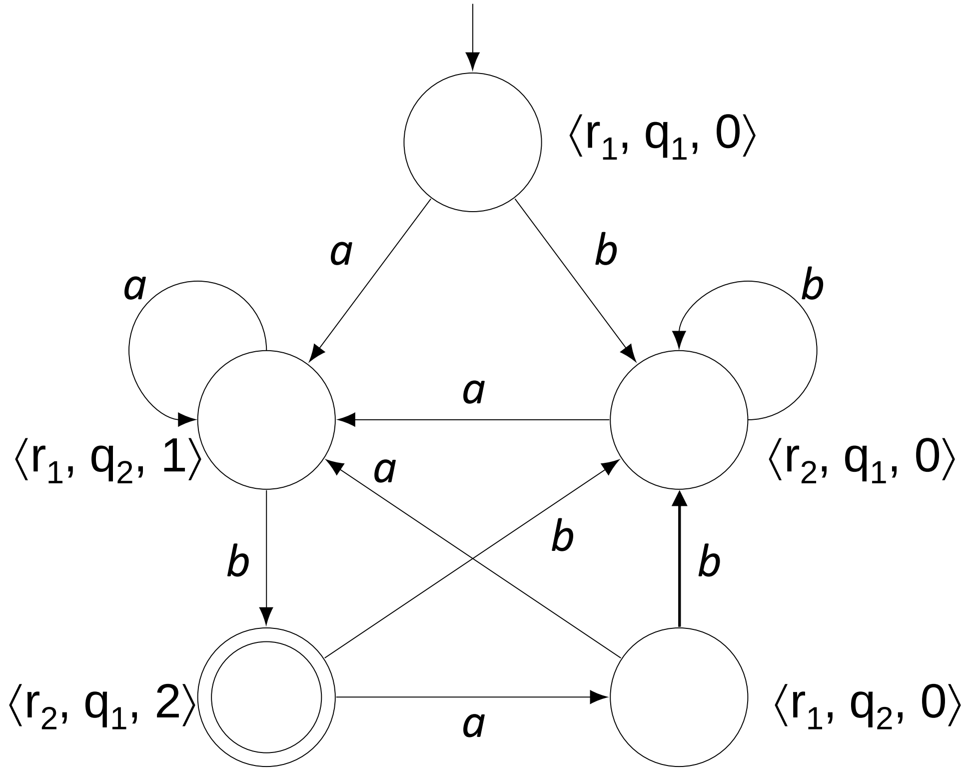
Non-emptiness ⇔ Existence of reachable accepting cycles

***Lemma:* Let $\mathcal{B} = (\Sigma, Q, \Delta, Q^0, F)$ be a Büchi automaton. The following conditions are equivalent:**

- *$\mathcal{L}(\mathcal{B})$ is nonempty.*
- *$\mathcal{B}$ contains a strongly connected component **C**, which includes an accepting state. Moreover, **C** is reachable from an initial state of $\mathcal{B}$.*
- *The graph induced by $\mathcal{B}$ contains a path from an initial state of $\mathcal{B}$ to a state $t \in F$ and a path from t back to itself.*

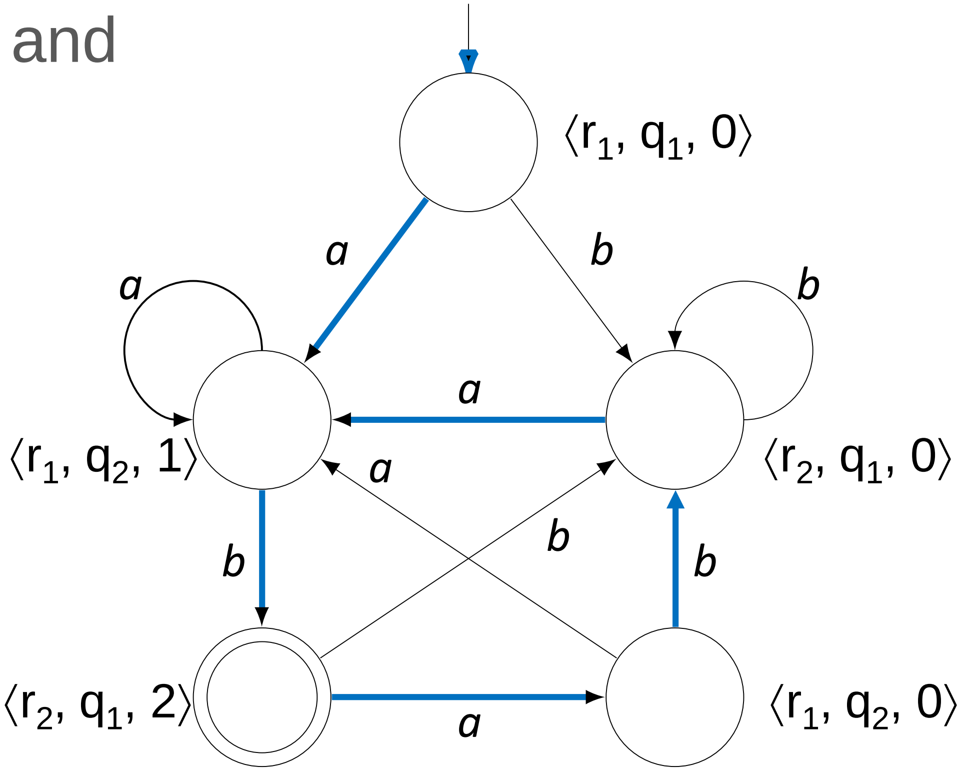
Example

- Is the language $\mathcal{L}(\mathcal{B})$ empty?



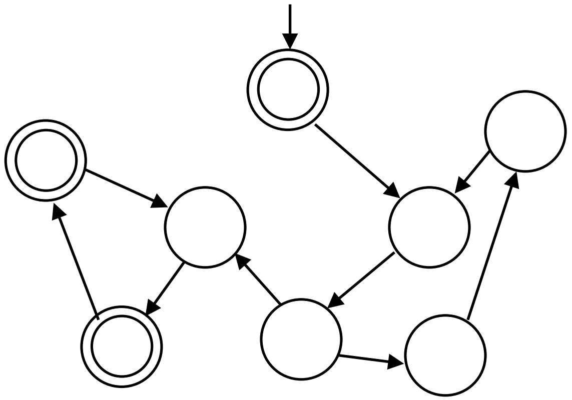
Example

- The language $\mathcal{L}(\mathcal{B})$ is nonempty.
  - $\mathcal{L}(\mathcal{B})$ = {inf number of a's and inf number of b's}
- $<r_2,q_1,2>$ is accepting and reachable from $<r_1,q_1,0>$ and reachable from itself



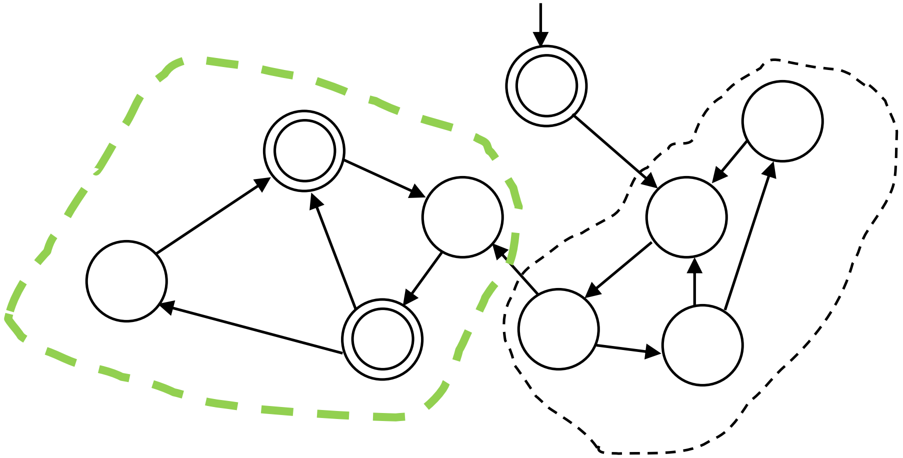
Emptiness and Accepting runs

- There can be an exponential number of cycles in a graph!
- But we want stay polynomial in the size of the graph…



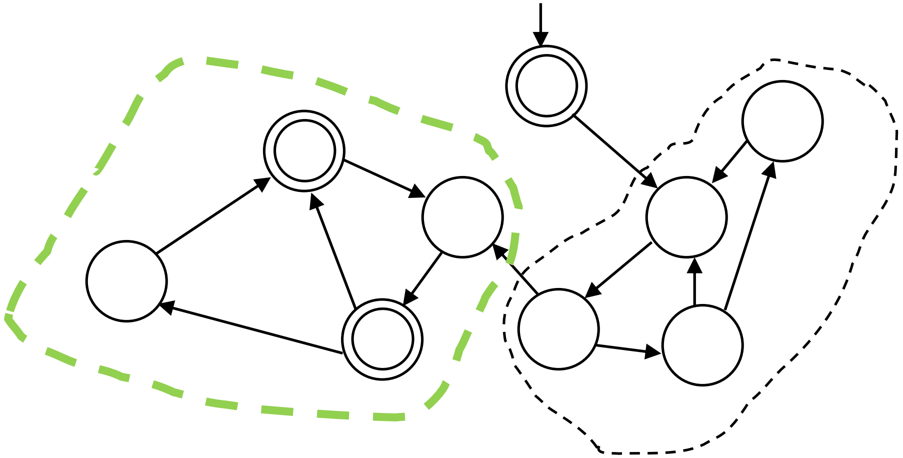
Finding Accepting Runs

- Rather than looking for cycles, look for SCCs:
  - A maximal Strongly Connected Component (SCC):
    - maximal set of nodes where each node is reachable from all others.
  - Find a **reachable SCC** with an **accepting** node.



Finding Accepting Runs

- Finding SCC's is linear in the size of the graph.
- Relies on a modified DFS
  - Record 'finishing time'
  - Let us recall DFS…

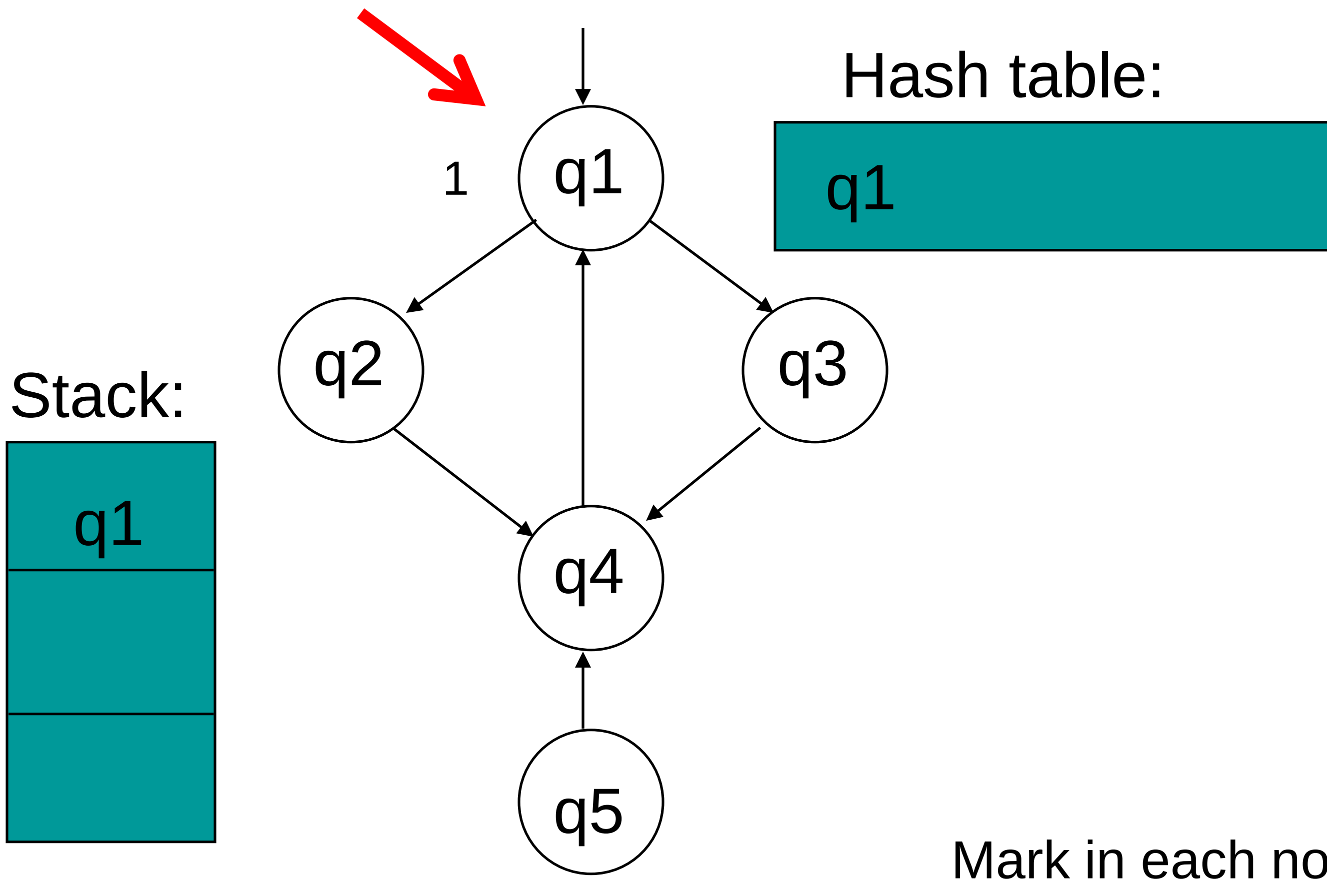


Depth First Search

Program DFS
for each initial state s_0:
    dfs(s_0)

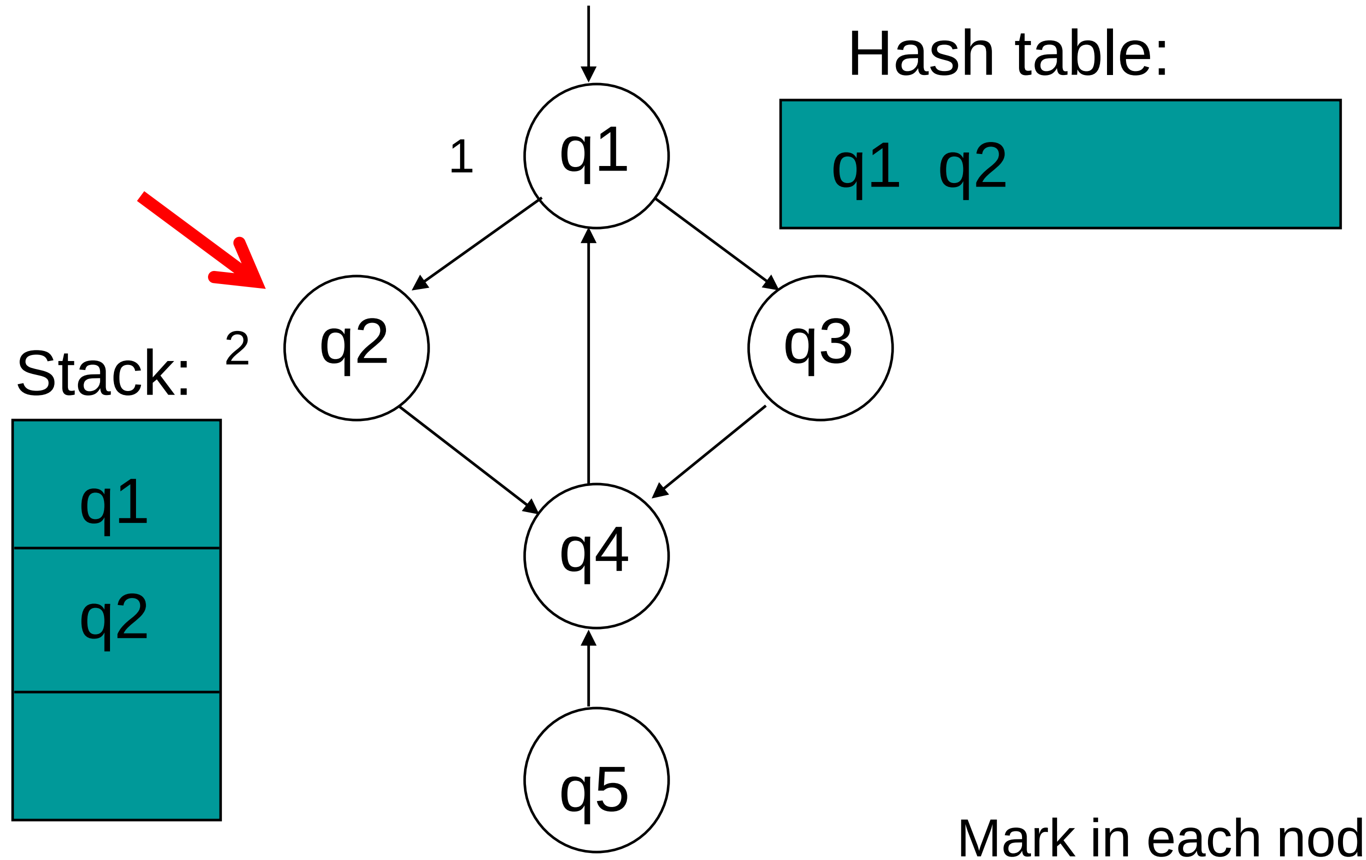
dfs(s)
for each s' such that $R(s, s')$:
    if new(s'):
        dfs(s')

Example DFS



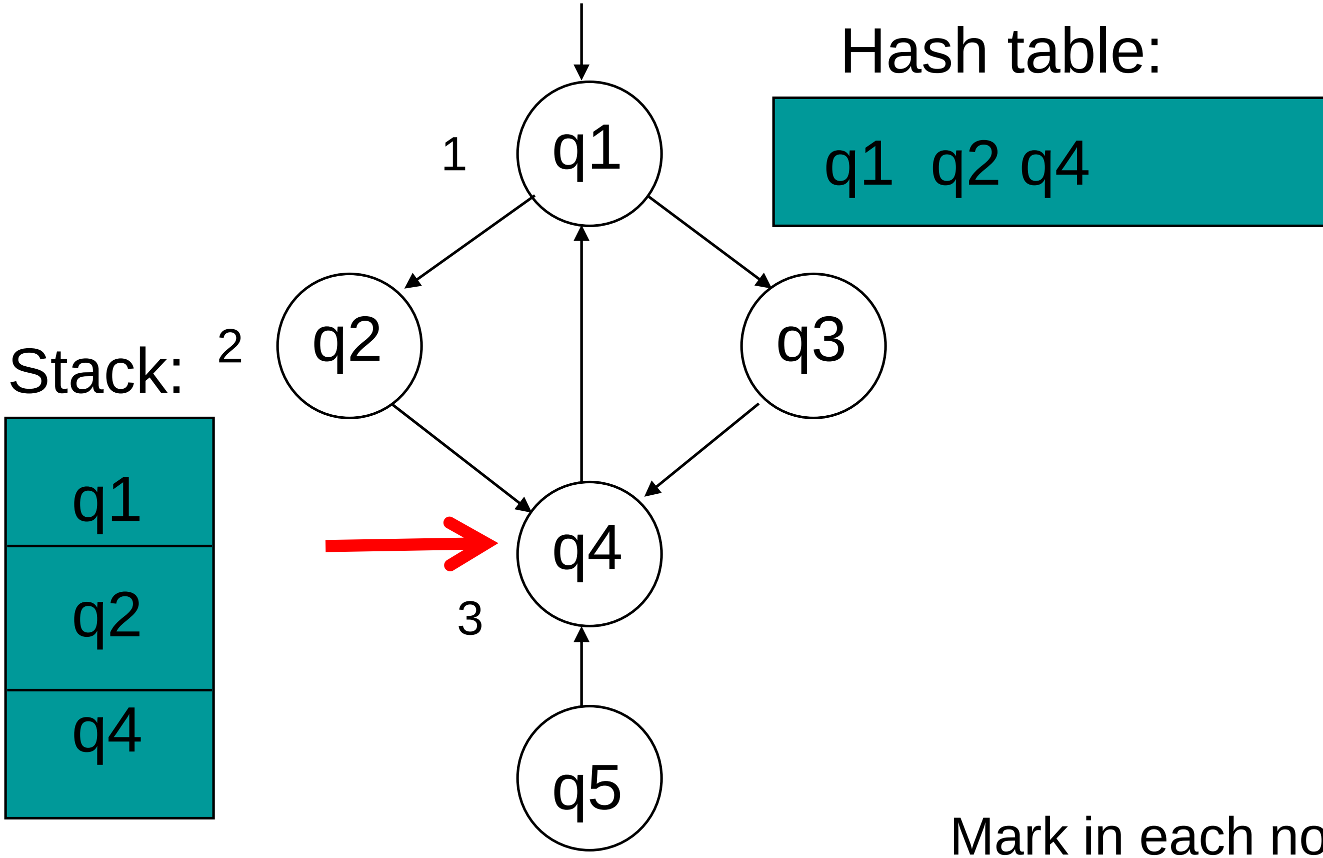
Mark in each node:
- Start time
- Finish time

Example DFS



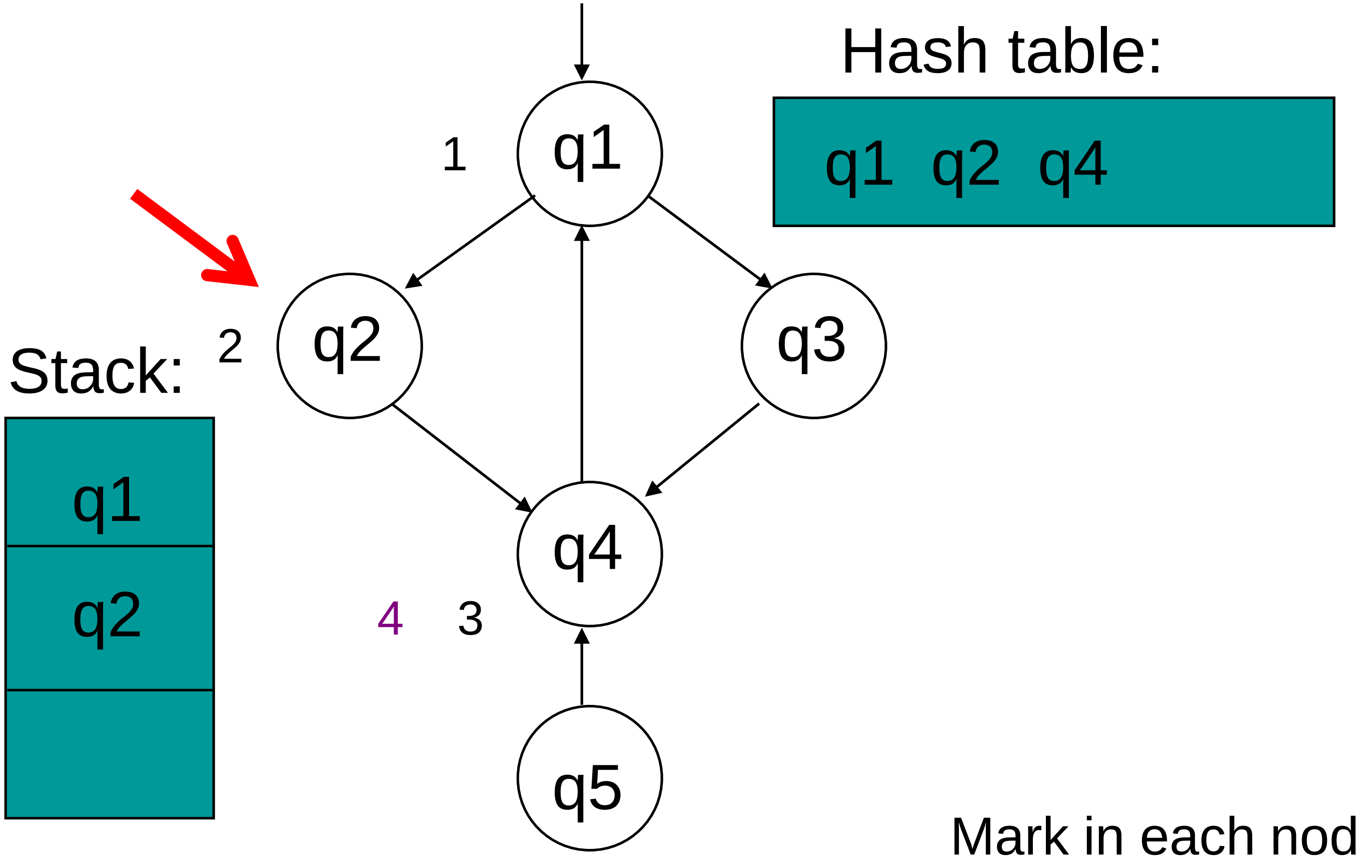
Mark in each node:
- Start time
- Finish time

Example DFS



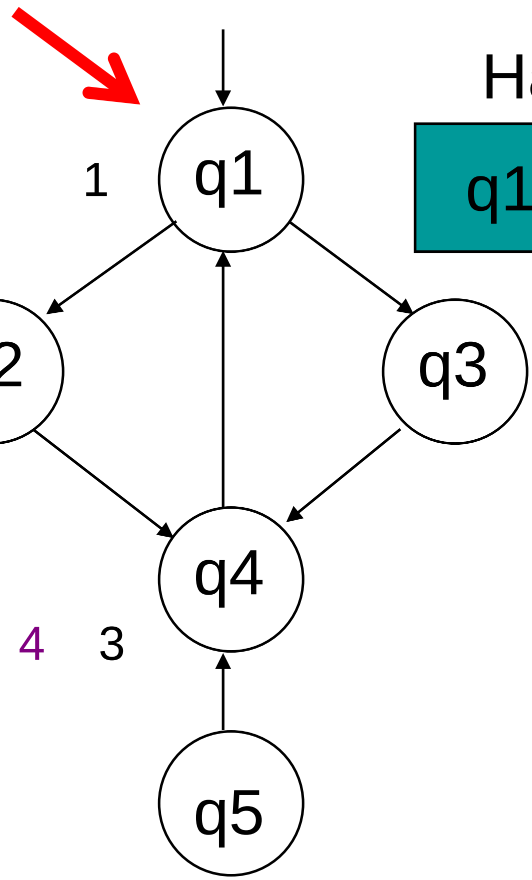
Mark in each node:
- Start time
- Finish time

Example DFS



Mark in each node:
- Start time
- Finish time

Example DFS



Hash table:

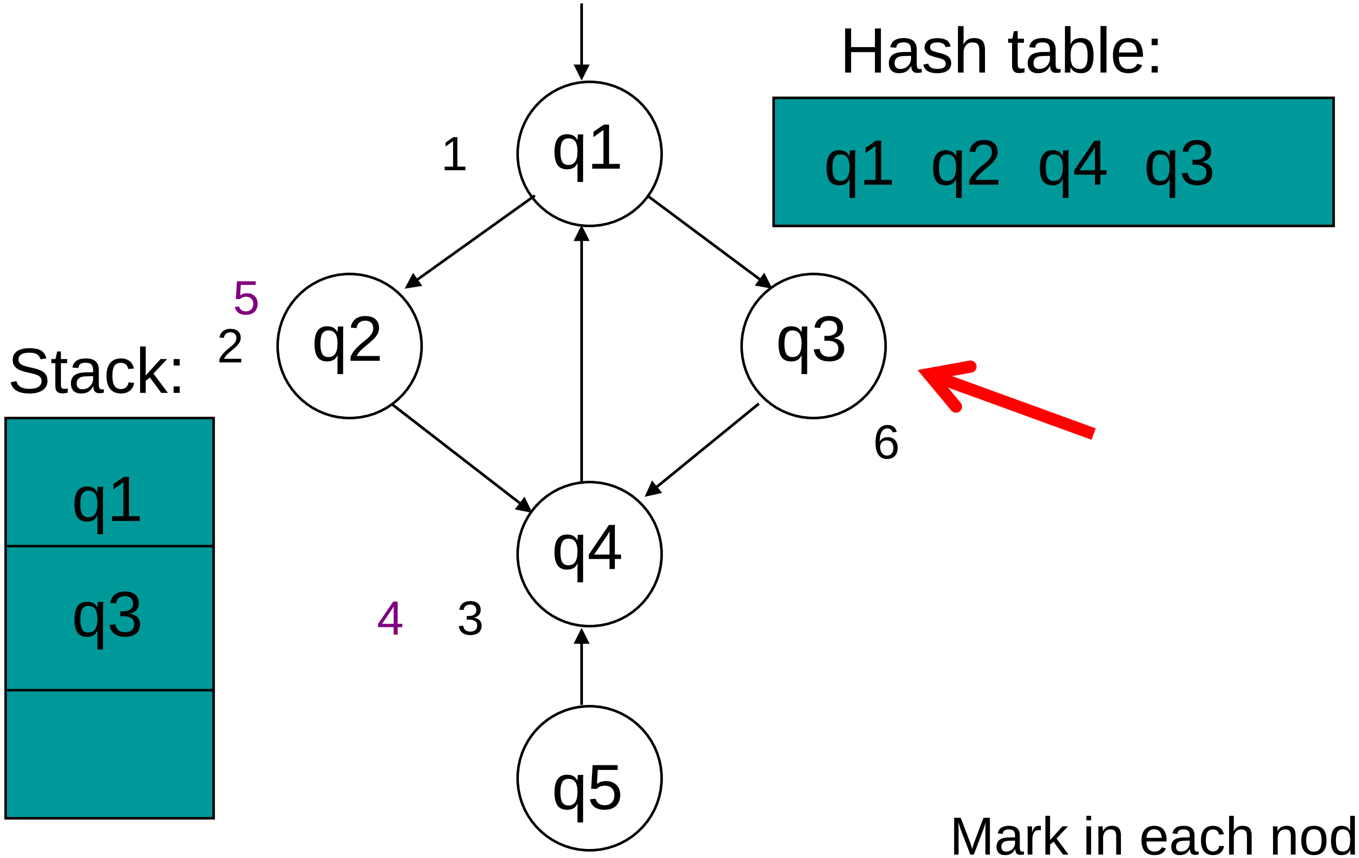
q1 q2 q4

Stack:

q1

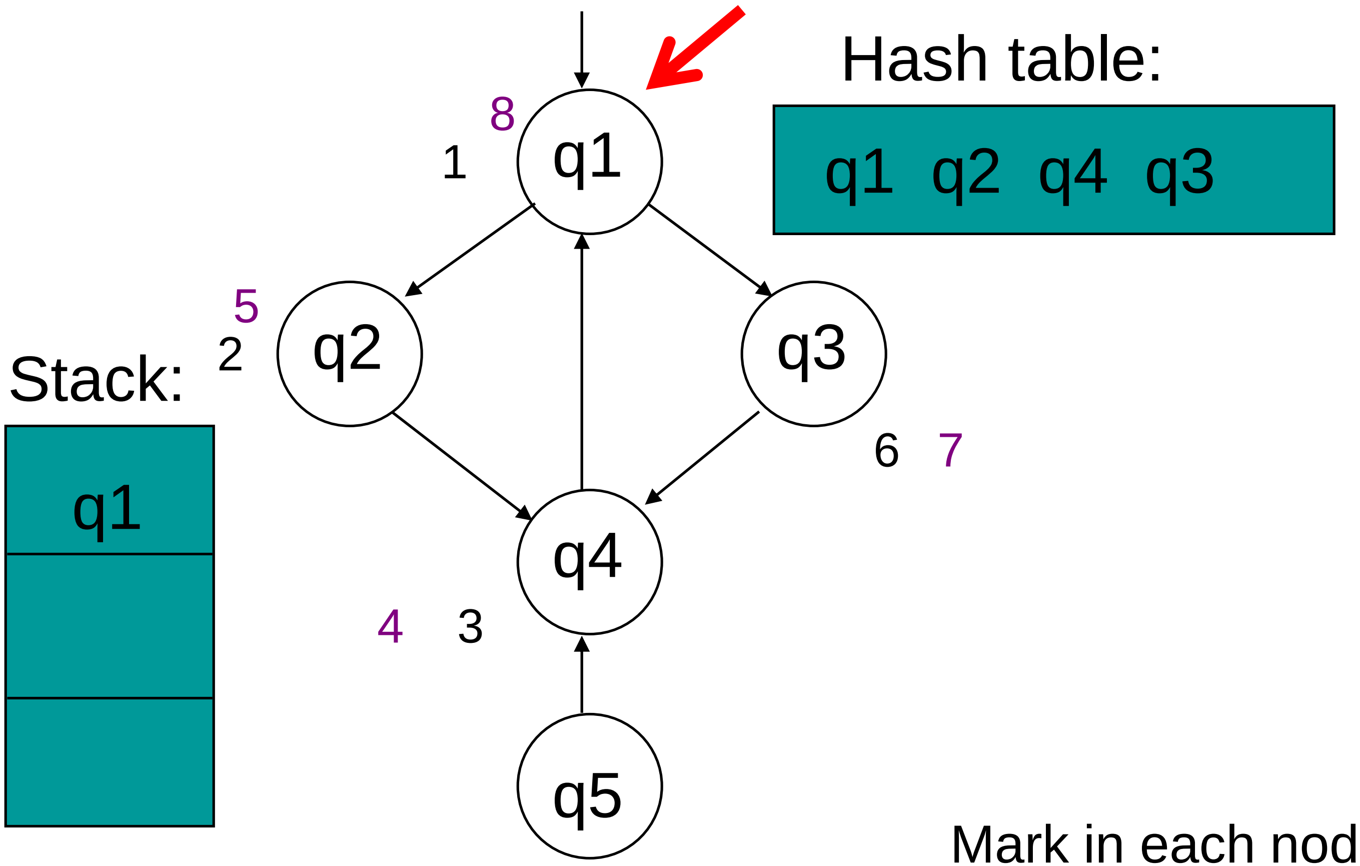
Mark in each node:
- Start time
- Finish time

Example DFS



Mark in each node:
- Start time
- Finish time

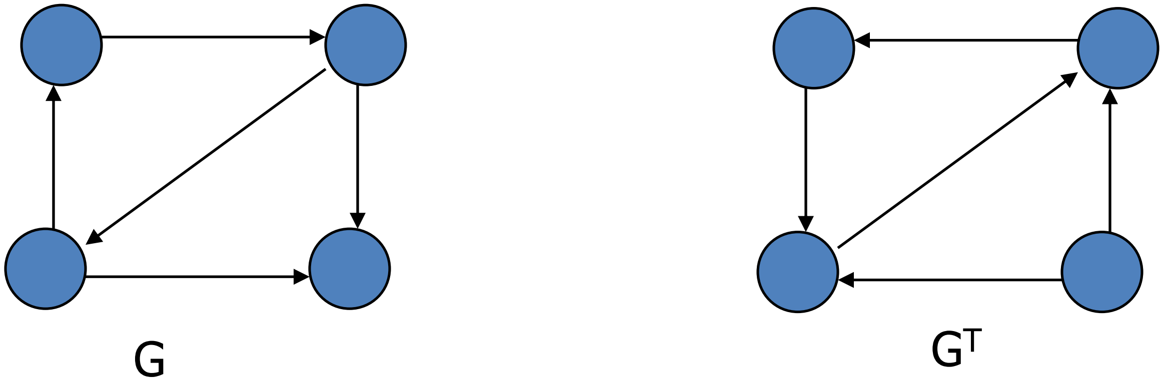
Example DFS



Mark in each node:
- Start time
- Finish time

An algorithm for finding SCCs

- The **transpose of G**, written **G^T**, is derived from G by reversing its edges.



An algorithm for finding SCCs

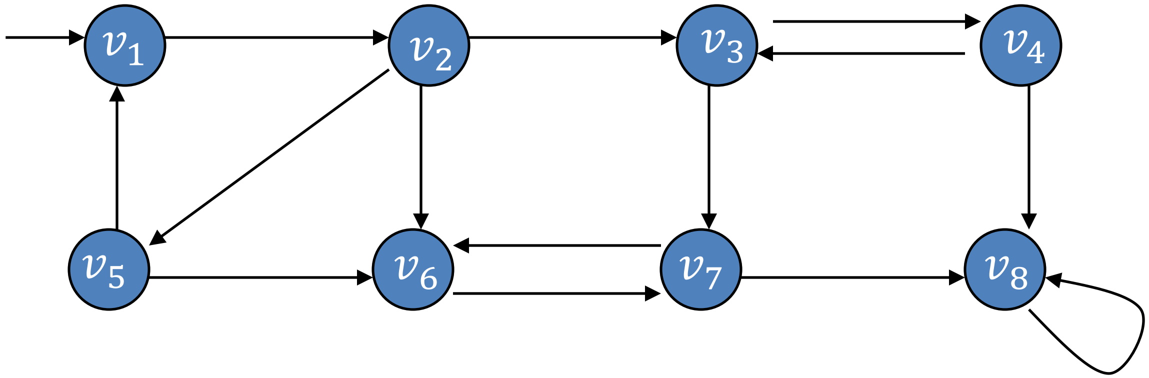
1. Call DFS(G) to compute finish[v] for each vertex in G.

2. Call Modified-DFS(G^T)
   - main loop of processes vertices in order of decreasing finish[v]

3. Each tree of Modified-DFS(G^T) is a SCC of G.

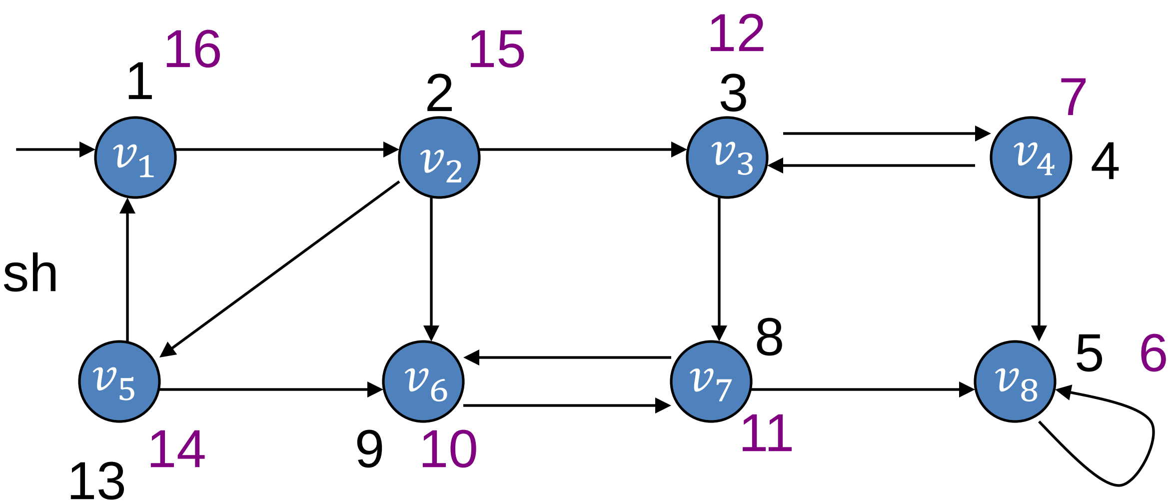
Example

Compute DFS Finish times



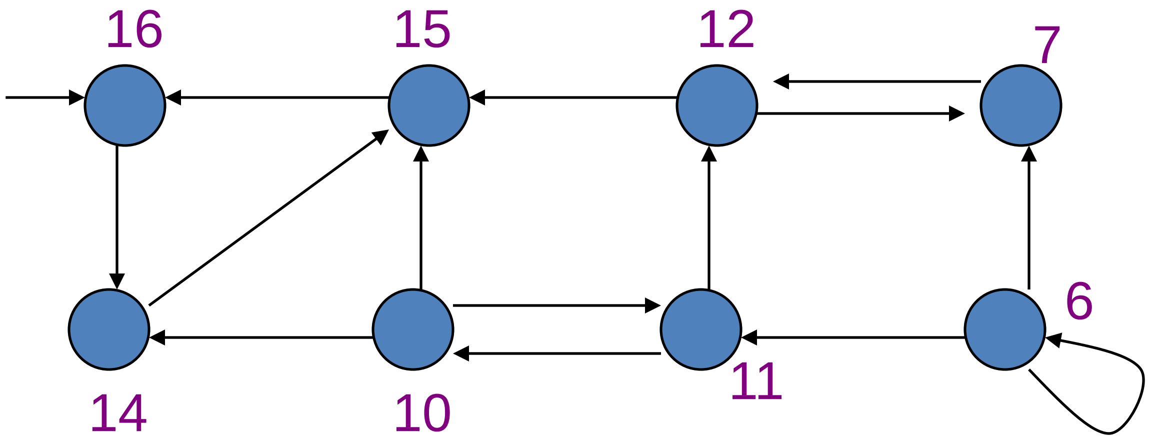
Example

DFS Finish times



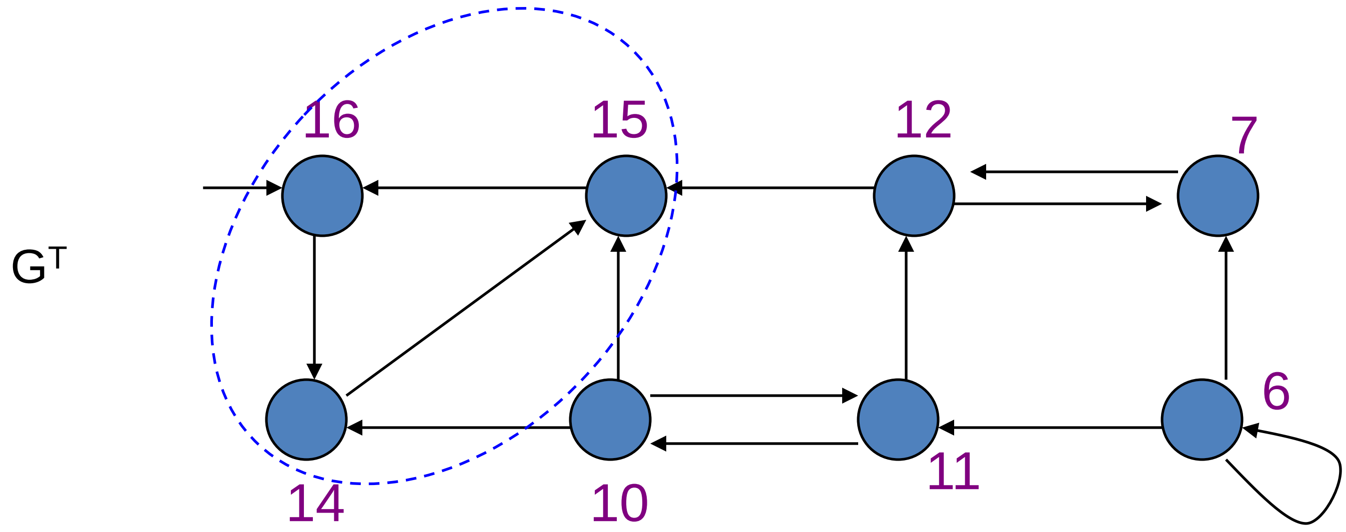
Example

GT



Swapped the direction of edges

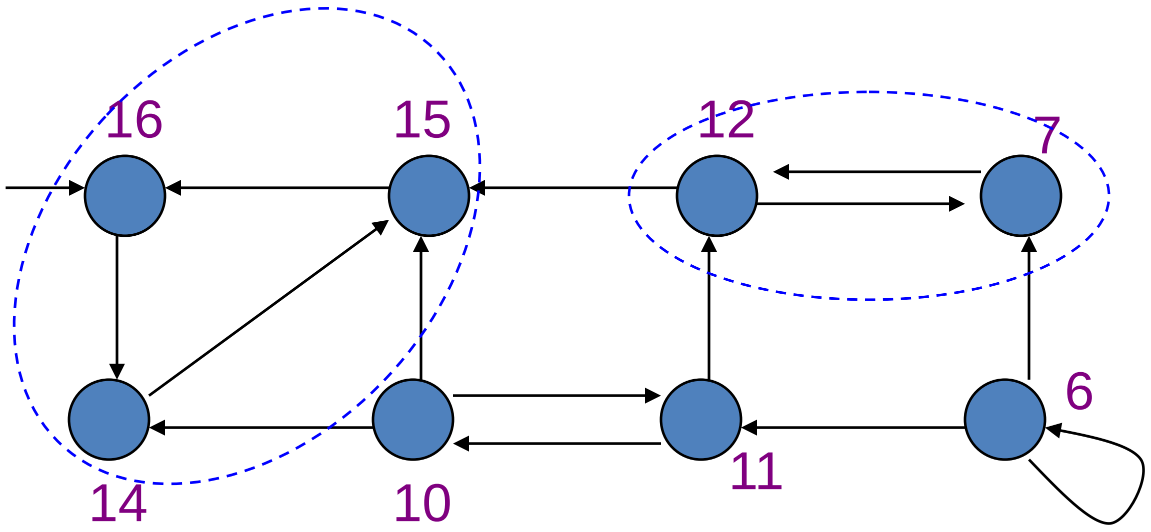
Example



DFS from decreasing finish times: every tree is an SCC.

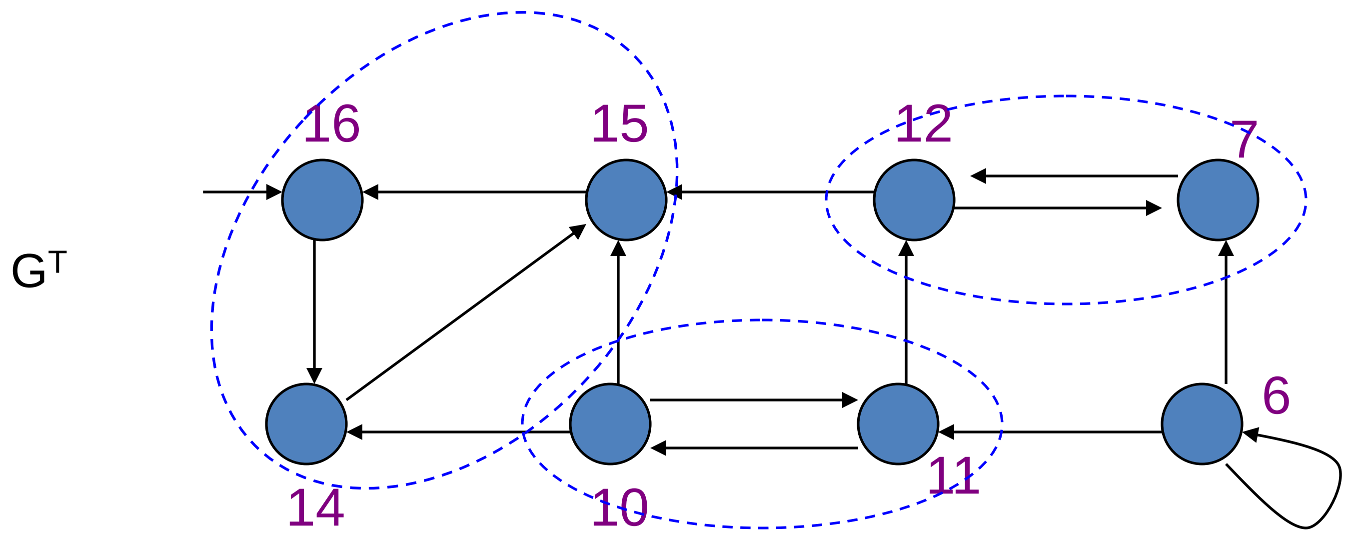
Example

G^T



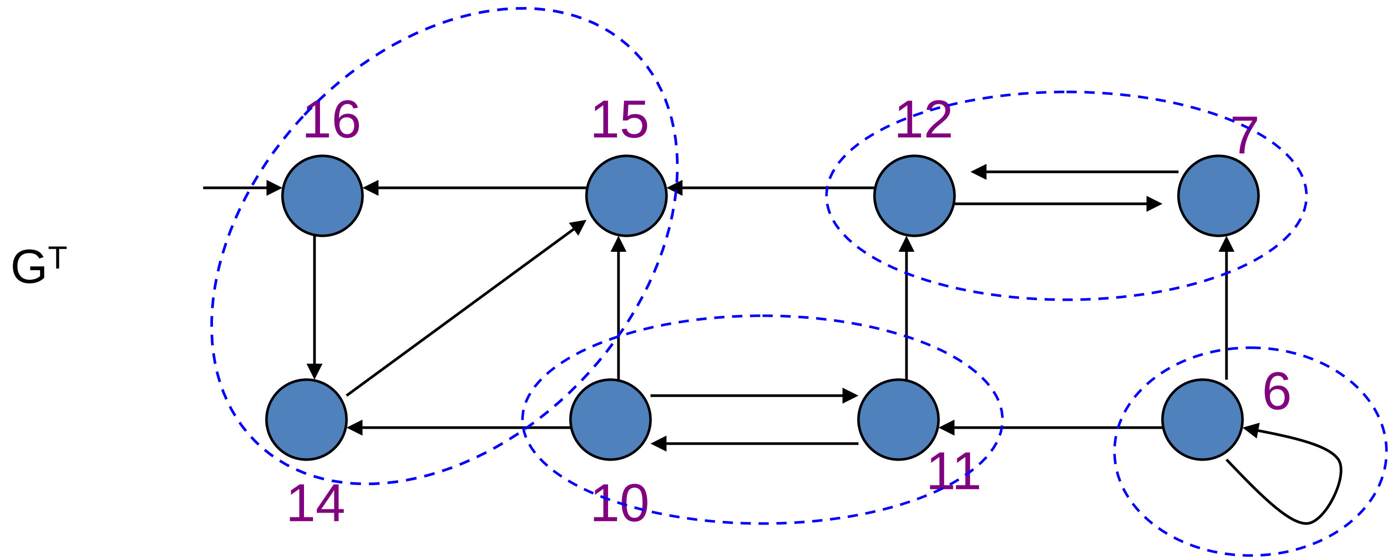
DFS from decreasing finish times: every tree is an SCC.

Example



DFS from decreasing finish times: every tree is an SCC.

Example



DFS from decreasing finish times: every tree is an SCC.

An algorithm for finding SCCs

1. Call DFS(G) to compute finish[v] for each vertex in G.
2. Call Modified-DFS(G^T)
   - main loop of processes vertices in order of decreasing finish[v]
3. Each tree of Modified-DFS(G^T) is a SCC of G.

ToDo What is the worst-case complexity in time and space?

An algorithm for finding SCCs

1. Call DFS(G) to compute finish[v] for each vertex in G.
2. Call Modified-DFS(G^T)
   - main loop of processes vertices in order of decreasing finish[v]
3. Each tree of Modified-DFS(G^T) is a SCC of G.

- What is the worst-case complexity in time and space?

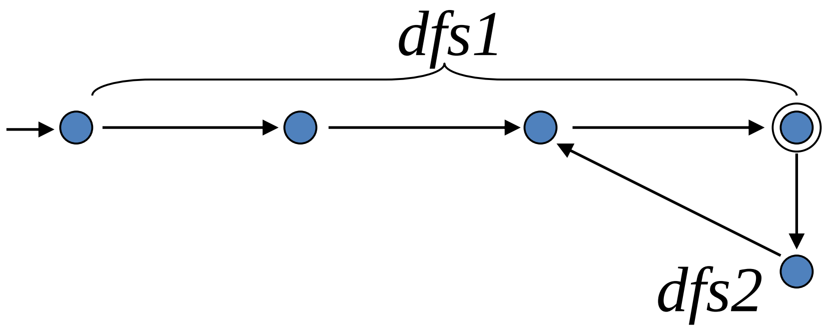
$O(|\mathcal{B}|)$

Double DFS-Algorithm

- Better alternative algorithm
  - Less memory
  - Saves the accepting path

Double DFS-Algorithm

- The first DFS finds a state f ∈ **F**
- The second DFS attempts to close a loop around it.



- The trick is:
  - how to avoid exploring the entire graph for each accepting state?

The Double-DFS algorithm

```
DFS1(s) {
  push(s,Stack1);
  hash(s,Table1);
  for each t ∈ Succ (s) do {
    if t ∉ Table1 then DFS1(t);
  }
  if s ∈ F then DFS2(s);
  pop(Stack1);
}
```

```
DFS2(s) {
  push(s, Stack2);
  hash(s, Table2) ;
  for each t ∈ Succ (s) do {
    if t is on Stack1 exit("not empty");
    else if t ∉ Table2 then DFS2(t)
  }
  pop( Stack2);
}
```

Upon finding an accepting cycle, Stack1, Stack2, t, determines a **witness:** an accepting cycle reached from an initial state.

The Double-DFS algorithm

Input: A
Initialize: Stack1:={} , Stack2:={}
Table1:={} , Table2:={}

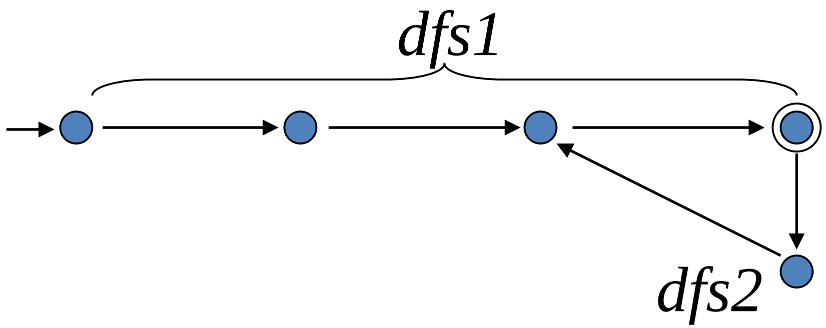
```
procedure Main() {
  for each s ∈ S_0 do {
    if  s ∉ Table1  then DFS1(s);
  }
  output("empty");
  exit;
}

DFS1(s)  {
   push(s,Stack1);
   hash(s,Table1);
   for each  t ∈ Succ (s) do {
     if t ∉ Table1 then DFS1(t);
   }
   if  s ∈ F then DFS2(s);
   pop(Stack1);
}
```

```
DFS2(s) {
  push(s, Stack2);
  hash(s, Table2) ;
  for each t ∈ Succ (s) do {
     if t is on Stack1 exit("not empty");
     else if t ∉ Table2 then DFS2(t)
   }
   pop( Stack2);
}
```

Upon finding an accepting cycle, Stack1, Stack2, t, determines a **witness:** an accepting cycle reached from an initial state.

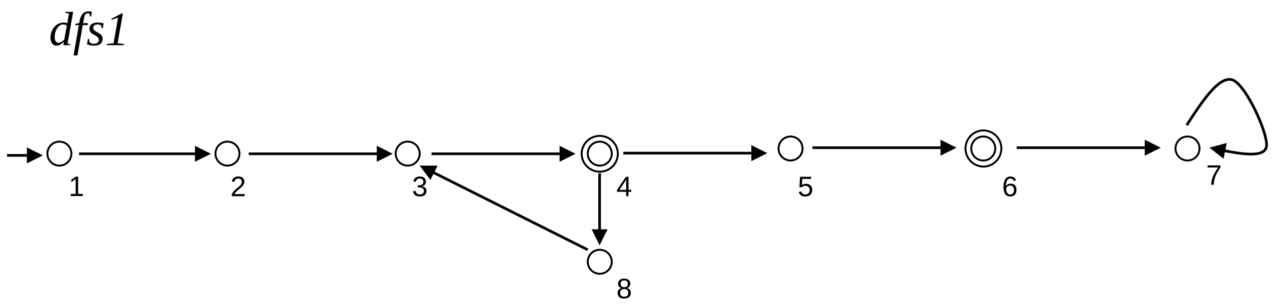
Double DFS-Algorithm

- The first DFS finds a state $f \in \mathbf{F}$
- The second DFS attempts to close a loop around it.



- The trick is:
  - how to avoid exploring the entire graph for each accepting state?

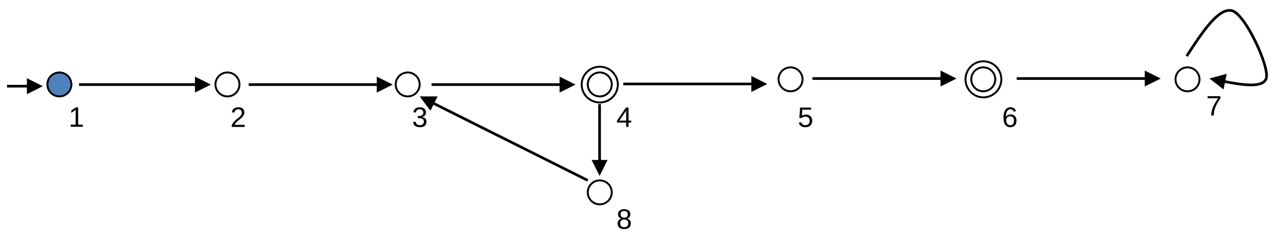
The Double DFS algorithm



Stack 1 = {}
Table 1 = {}

The Double DFS algorithm

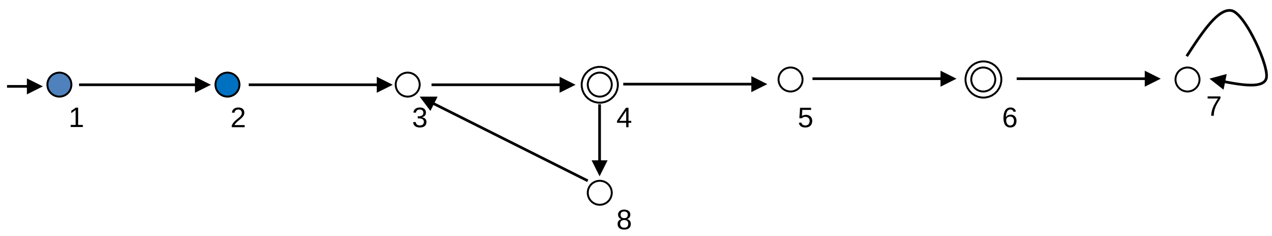
dfs1



Stack 1 = {1}
Table 1 = {1}

The Double DFS algorithm

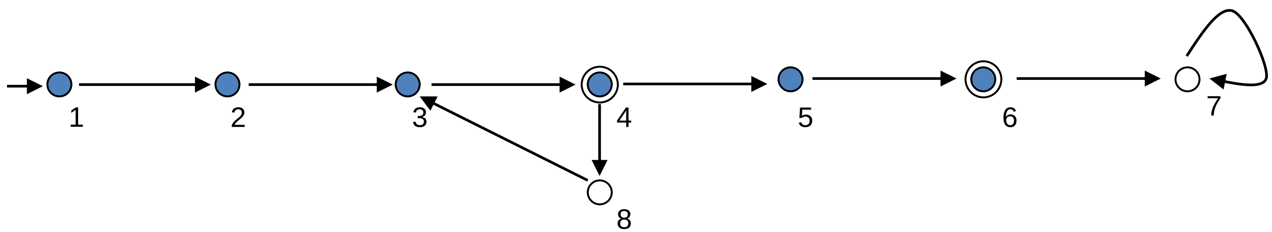
dfs1



Stack 1 = {1,2}
Table 1 = {1,2}

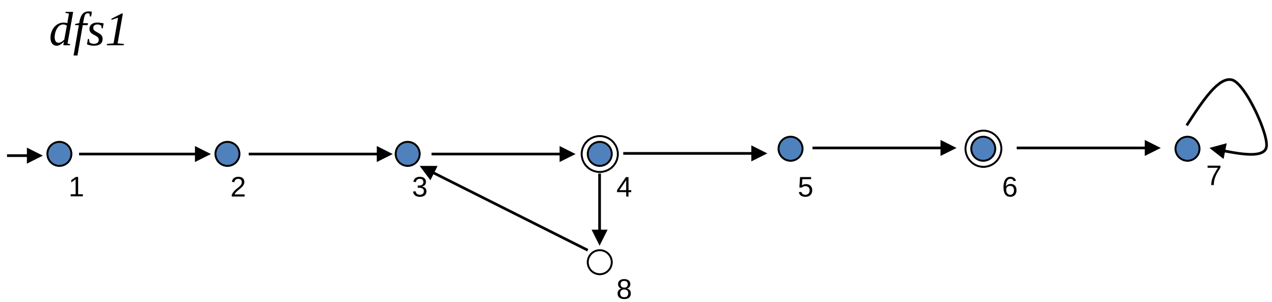
The Double DFS algorithm

dfs1



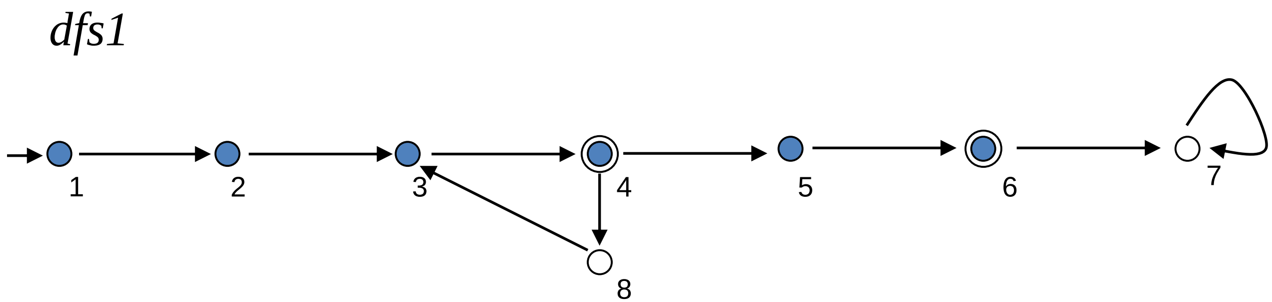
Stack 1 = {1 – 6}
Table 1 = {1 – 6}

The Double DFS algorithm



Stack 1 = {1 – 7}
Table 1 = {1 – 7}

The Double DFS algorithm

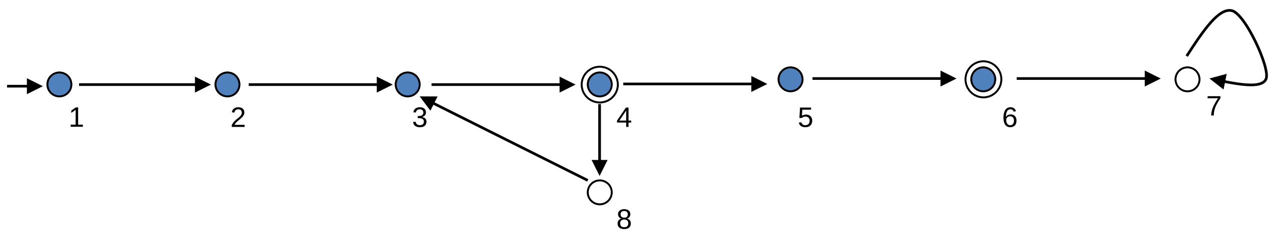


Stack 1 = {1 – 6}
Table 1 = {1 – 7}

For the first time we identified an accepting state for which all the successors were already explored. Now it's DFS2's turn to try to close the loop.

The Double DFS algorithm

dfs1



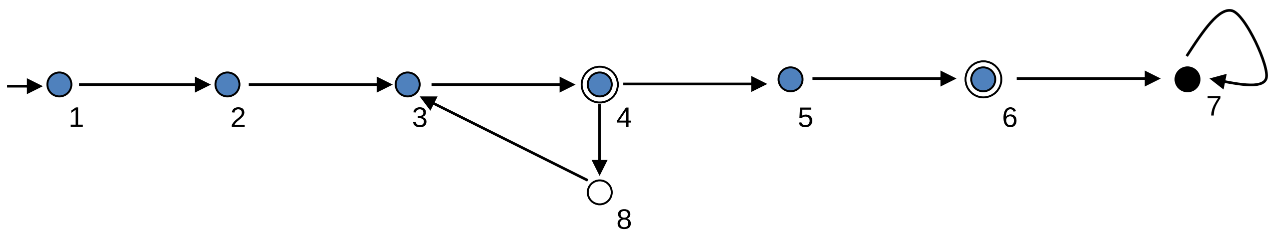
Stack 1 = {1 – 6}
Table 1 = {1 – 7}

Stack 2 = {6}
Table 2 = {6}

For the first time we identified an accepting state for which all the successors were already explored. Now it’s DFS2’s turn to try to close the loop.

The Double DFS algorithm

dfs2

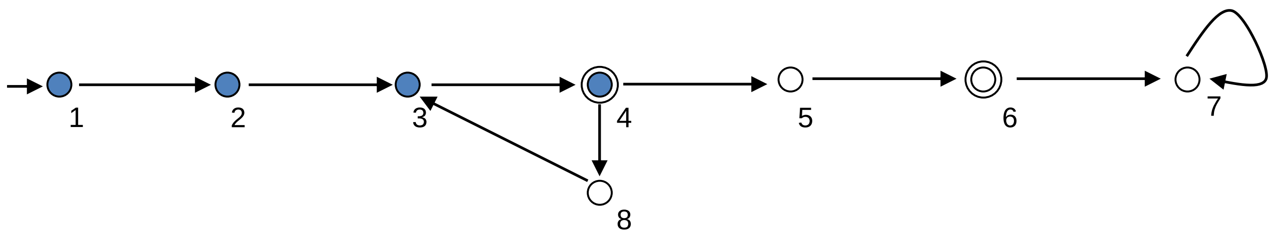


Stack 1 = {1 – 6}
Table 1 = {1 – 7}

Stack 2 = {6,7}
Table 2 = {6,7}

The Double DFS algorithm

dfs1



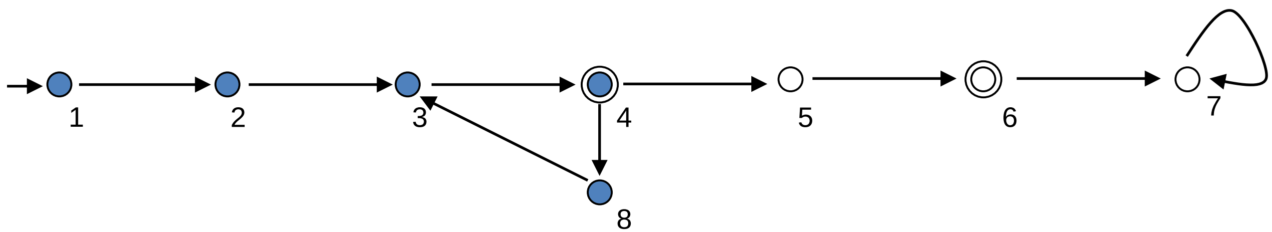
Stack 1 = {1 – 4}
Table 1 = {1 – 7}

Stack 2 = {}
Table 2 = {6,7}

Backtracking, 4 still has successors…

The Double DFS algorithm

dfs1

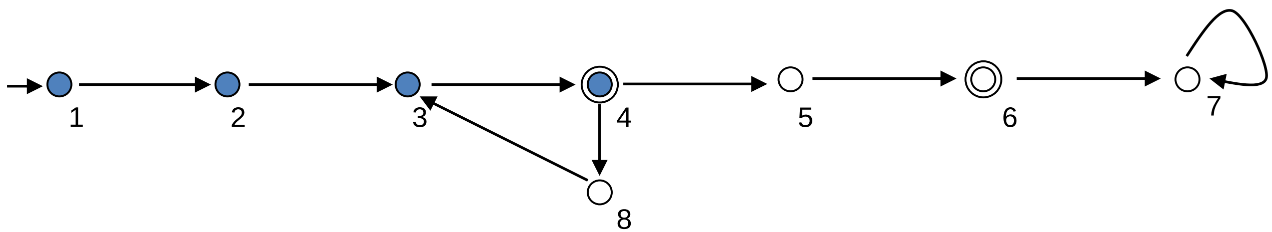


Stack 1 = {1 – 4,8}
Table 1 = {1 – 8}

Stack 2 = {}
Table 2 = {6,7}

The Double DFS algorithm

dfs1



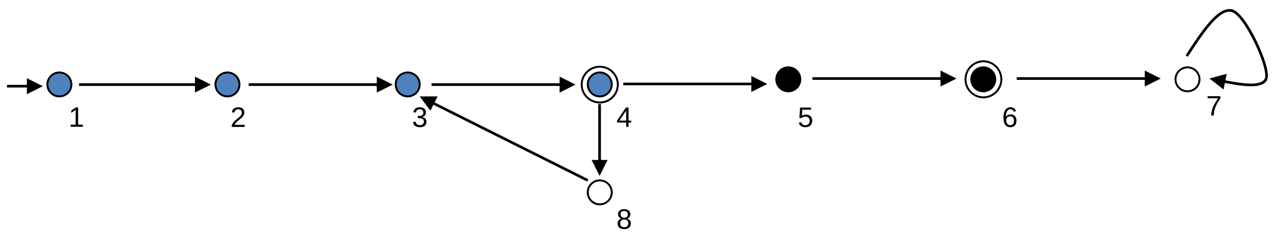
Stack 1 = {1 – 4}
Table 1 = {1 – 8}

Stack 2 = {}
Table 2 = {6,7}

Again we identified an accepting state for which all successors were already explored. Now it's DFS'2 turn to try to close the loop.

The Double DFS algorithm

dfs2



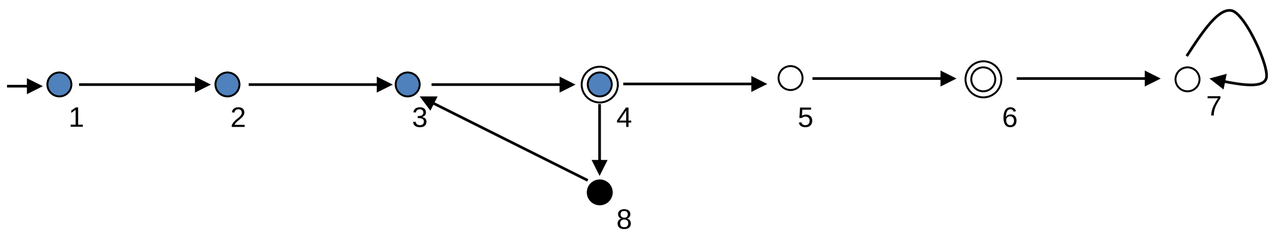
Stack 1 = {1 – 4}
Table 1 = {1 – 8}

Stack 2 = {4,5}
Table 2 = {4 - 7}

No point continuing to what is already in Table 2 (why?)

The Double DFS algorithm

dfs2



Stack 1 = {1 – 4}
Table 1 = {1 – 8}

Stack 2 = {4,8}
Table 2 = {4 – 8}

Bingo! Found a cycle !
(DFS2 progresses to node 3 which is on Stack1)

Correctness of the Double-DFS algorithm

- The Double-DFS algorithm outputs "empty" $\Leftrightarrow$ $\mathcal{L}(\mathcal{B})$ is empty.
- If Double-DFS outputs "not empty",
  then content of Stack1 + Stack2 + t is a word in $\mathcal{L}(\mathcal{B})$.

Complexity of Double-DFS

```
DFS1(s)  {
  push(s,Stack1);
  hash(s,Table1);
  for each  t ∈ Succ (s) do {
    if t ∉ Table1 then DFS1(t);
  }
  if  s ∈ F then DFS2(s);
  pop(Stack1);
}
```

```
DFS2(s) {
  push(s, Stack2);
  hash(s, Table2) ;
  for each t ∈ Succ (s) do {
    if t is on Stack1 exit(“not empty”);
    else if t ∉ Table2 then DFS2(t)
  }
  pop( Stack2);
}
```

ToDo What is the worst-case complexity in time and space?

Complexity of Double-DFS

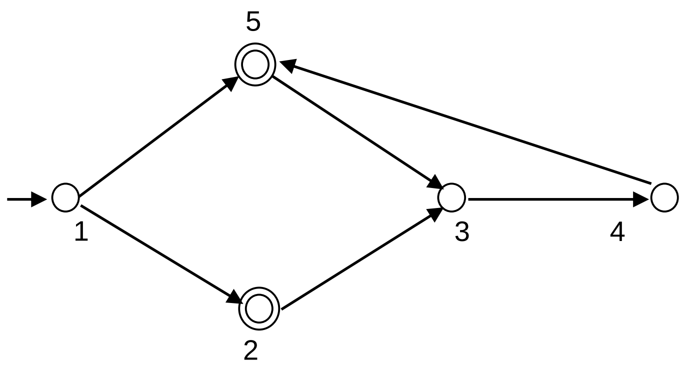
```
DFS1(s)  {
  push(s,Stack1);
  hash(s,Table1);
  for each  t ∈ Succ (s) do {
   if t ∉ Table1 then DFS1(t);
  }
  if  s ∈ F then DFS2(s);
  pop(Stack1);
}
```

```
DFS2(s) {
  push(s, Stack2);
  hash(s, Table2) ;
  for each t ∈ Succ (s) do {
    if t is on Stack1 exit("not empty");
    else if t ∉ Table2 then DFS2(t)
  }
  pop( Stack2);
}
```

- What is the worst-case complexity in time and space?
  - $O(|\mathcal{B}|)$

Double-DFS

ToDo Apply the DDFS algorithm:

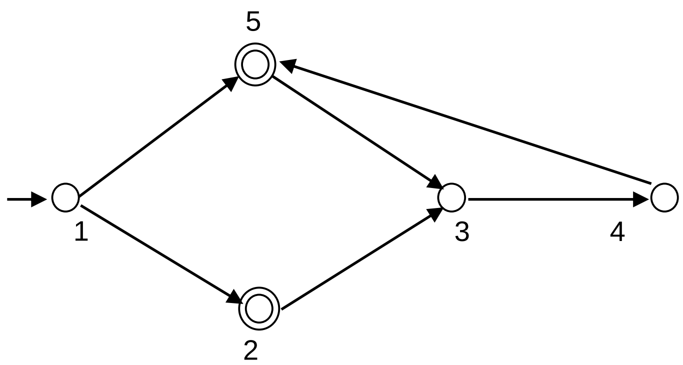



```
DFS1(s) {
  push(s,Stack1);
  hash(s,Table1);
  for each  t ∈ Succ (s) do {
    if t ∉ Table1 then DFS1(t);
  }
  if  s ∈ F then DFS2(s);
  pop(Stack1);
}
```

```
DFS2(s) {
  push(s, Stack2);
  hash(s, Table2) ;
  for each t ∈ Succ (s) do {
    if t is on Stack1 exit("not empty");
    else if t ∉ Table2 then DFS2(t)
  }
  pop( Stack2);
}
```

Double-DFS

- Apply the DDFS algorithm:



Stack 1 = {1,2,3,4,5} Stack 2 = {5} t=3
Table 1 = {1,2,3,4,5} Table 2 = {5}

```
DFS1(s)  {
  push(s,Stack1);
  hash(s,Table1);
  for each  t ∈ Succ (s) do {
    if t ∉ Table1 then DFS1(t);
  }
  if  s ∈ F then DFS2(s);
  pop(Stack1);
}
```

```
DFS2(s) {
  push(s, Stack2);
  hash(s, Table2) ;
  for each t ∈ Succ (s) do {
    if t is on Stack1 exit("not empty");
    else if t ∉ Table2 then DFS2(t)
  }
  pop( Stack2);
}
```

Outline

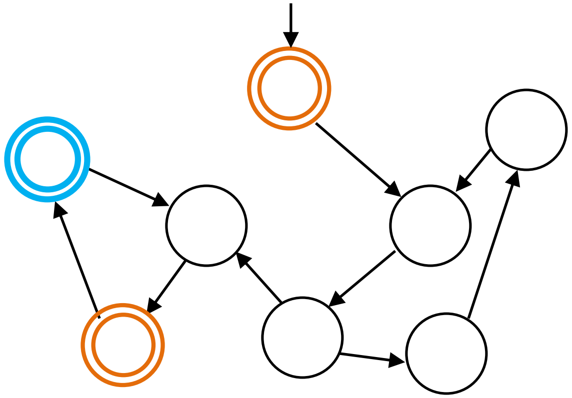
- Finite automata on finite words
- Automata on infinite words (Büchi automata)
- Deterministic vs non-deterministic Büchi automata
- Intersection of Büchi automata
- Checking emptiness of Büchi automata
- Generalized Büchi automata
- Automata and Kripke Structures
- Model checking using automata
- Translation of LTL to Büchi automata

Generalized Büchi automata

- Have several sets of accepting states

- $\mathcal{B} = (\mathbf{\Sigma}, \mathbf{Q}, \mathbf{\Delta}, \mathbf{Q}^0, \mathbf{F})$ is a generalized Büchi automaton:
  - $\mathbf{F} = \{P_1, \ldots, P_k\}$, where for every $1 \leq i \leq k$, $P_i \subseteq Q$

- A run ρ of $\mathcal{B}$ is accepting if for each $P_i \in \mathbf{F}$, $inf(\rho) \cap P_i \neq \emptyset$



Translation from Generalized Büchi to Büchi

- Given $\boldsymbol{\mathcal{B}} = (\boldsymbol{\Sigma}, \mathbf{Q}_1, \boldsymbol{\Delta}_1, \mathbf{Q}_1^0, \mathbf{F}_1)$ with $\mathbf{F} = \{P_1, \ldots, P_k\}$

- How does it work to construct a Büchi automaton $\boldsymbol{\mathcal{B}}$'

Translation from Generalized Büchi to Büchi

- $\mathcal{B} = (\mathbf{\Sigma},\mathbf{Q}_1,\mathbf{\Delta}_1,\mathbf{Q}_1{}^0,\mathbf{F}_1)$ with $\mathbf{F} = \{P_1, \dots, P_k\}$

- $\mathcal{B}' = (\mathbf{\Sigma}, \mathbf{Q} \times \{0,1,\dots,k\}, \mathbf{\Delta}', \mathbf{Q}^0 x 0, \mathbf{Q} \times k)$ with:

Translation from Generalized Büchi to Büchi

- $\mathcal{B} = (\boldsymbol{\Sigma}, \mathbf{Q}_1, \boldsymbol{\Delta}_1, \mathbf{Q}_1^0, \mathbf{F}_1)$ with $\mathbf{F} = \{P_1, \ldots, P_k\}$

- $\mathcal{B}' = (\boldsymbol{\Sigma}, \mathbf{Q} \times \{0,1,\ldots,k\}, \boldsymbol{\Delta}', \mathbf{Q}^0 x0, \mathbf{Q} \times k)$ with:

- The transition relation $\boldsymbol{\Delta}'$:
  $((q,x),a,(q',y)) \in \boldsymbol{\Delta}'$ when $(q,a,q') \in \boldsymbol{\Delta}$ and x and y are as follows:

  - If $q' \in P_i$ and $x=i$, then $y=i+1$ for $i<k$
  - If $x=k$, then $y=0$.
  - Otherwise, $x = y$.

Translation from Generalized Büchi to Büchi

- $\mathcal{B} = (\mathbf{\Sigma},\mathbf{Q}_1,\mathbf{\Delta}_1,\mathbf{Q}_1^0,\mathbf{F}_1)$ with $\mathbf{F} = \{P_1, \ldots, P_k\}$

- $\mathcal{B}' = (\mathbf{\Sigma}, \mathbf{Q} \times \{0,1,\ldots,k\}, \mathbf{\Delta}', \mathbf{Q}^0 x0, \mathbf{Q} \times k)$ with:

- The transition relation $\mathbf{\Delta}'$:
  $((q,x),a,(q',y)) \in \mathbf{\Delta}'$ when $(q,a,q') \in \mathbf{\Delta}$ and x and y are as follows:

  - If $q' \in P_i$ and $x=i$, then $y=i+1$ for $i<k$
  - If $x=k$, then $y=0$.
  - Otherwise, $x = y$.

- Size of $\mathcal{B}'$ = (size of $\mathcal{B}$) $\times$ (k+1)

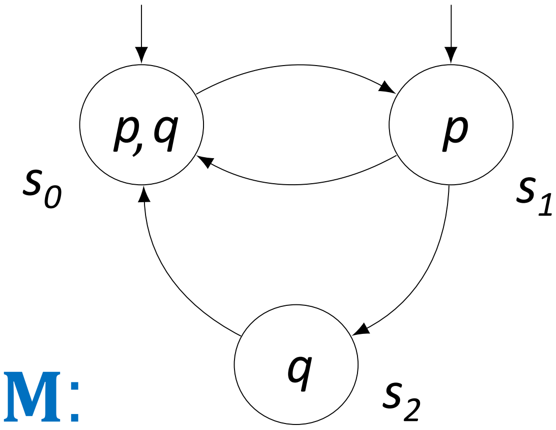
Outline

- Finite automata on finite words
- Automata on infinite words (Büchi automata)
- Deterministic vs non-deterministic Büchi automata
- Intersection of Büchi automata
- Checking emptiness of Büchi automata
- Generalized Büchi automata
- Automata and Kripke Structures
- Model checking using automata
- Translation of LTL to Büchi automata

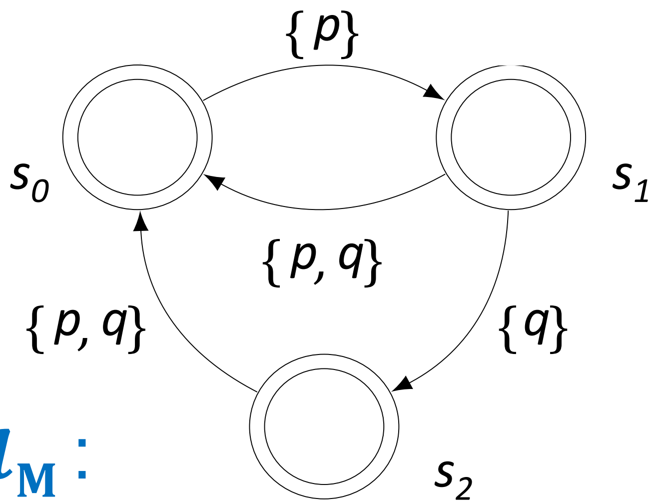
Kripke Structure **M** to Büchi Automaton $\mathcal{A}_{\mathbf{M}}$

- Move labels to transitions
- All states are accepting
- ToDo What about initial states?

M:



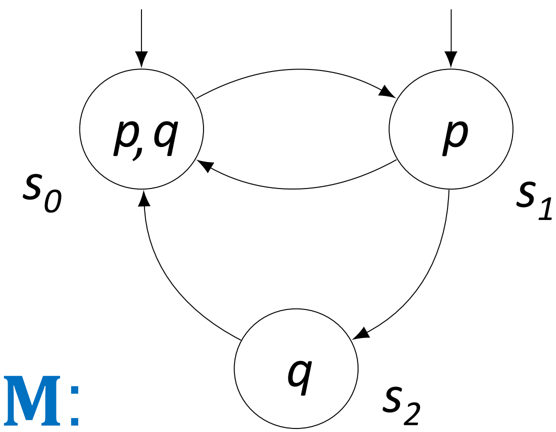
$\mathcal{A}_{\mathbf{M}}$:



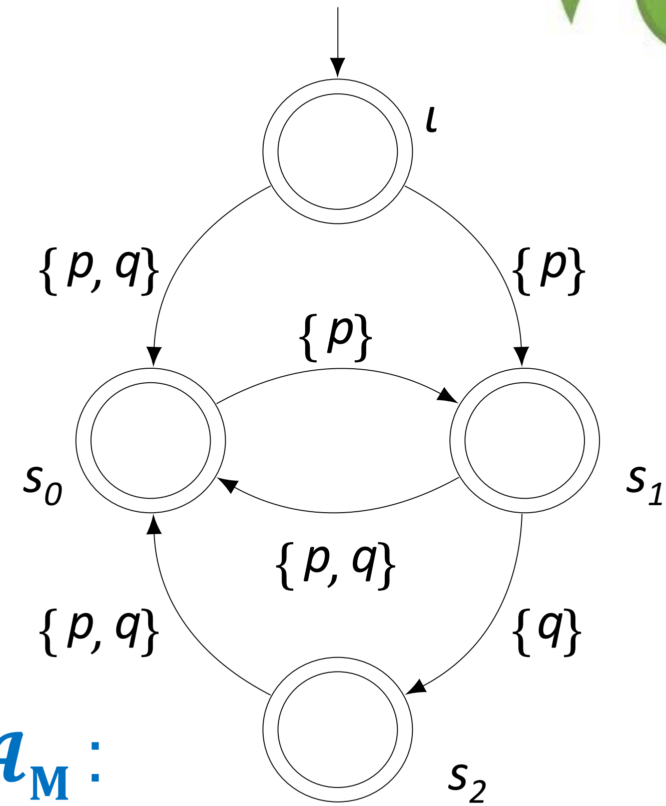
Kripke Structure **M** to Büchi Automaton $\mathcal{A}_{\mathbf{M}}$

- Move labels to transitions
- All states are accepting

M:



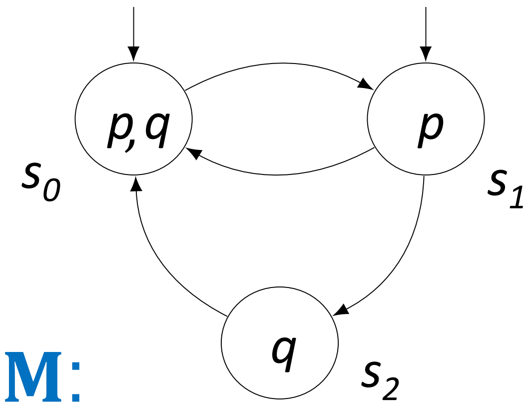
$\mathcal{A}_{\mathbf{M}}$:

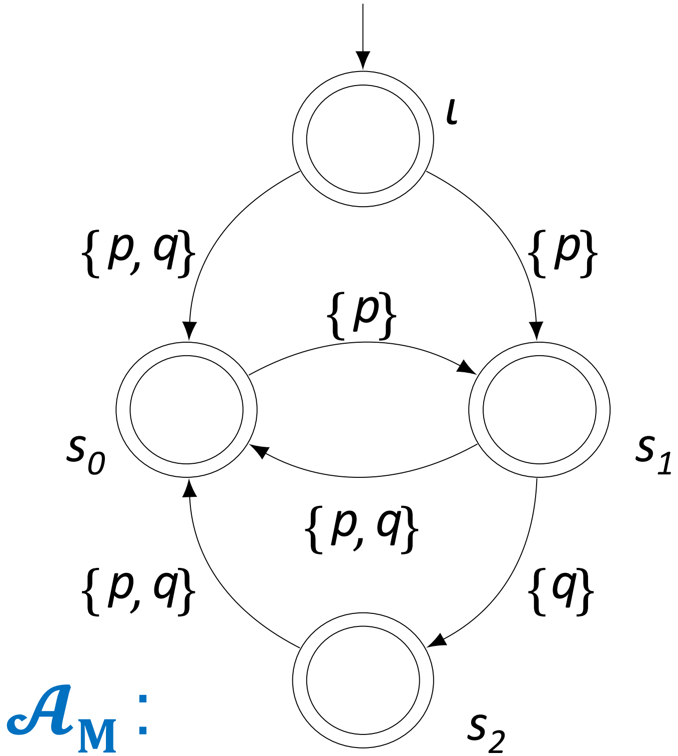


Automata and Kripke Structures

$\mathbf{M} = (\mathbf{S}, \mathbf{S}_0, \mathbf{R}, AP, \mathbf{L}) \;\Rightarrow\; \mathcal{A}_\mathbf{M} = (\mathbf{\Sigma}, \mathbf{S}\cup\{\iota\}, \mathbf{\Delta}, \{\iota\}, \mathbf{S}\cup\{\iota\})$,

where $\mathbf{\Sigma} = P(AP)$.



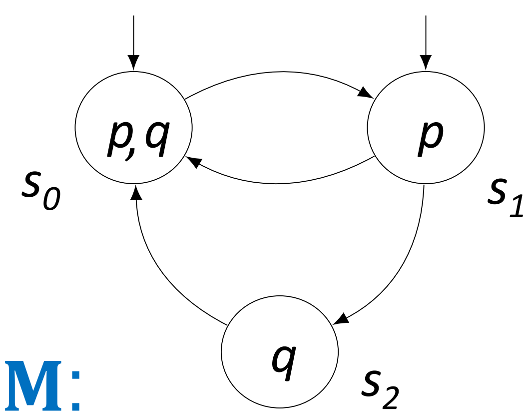


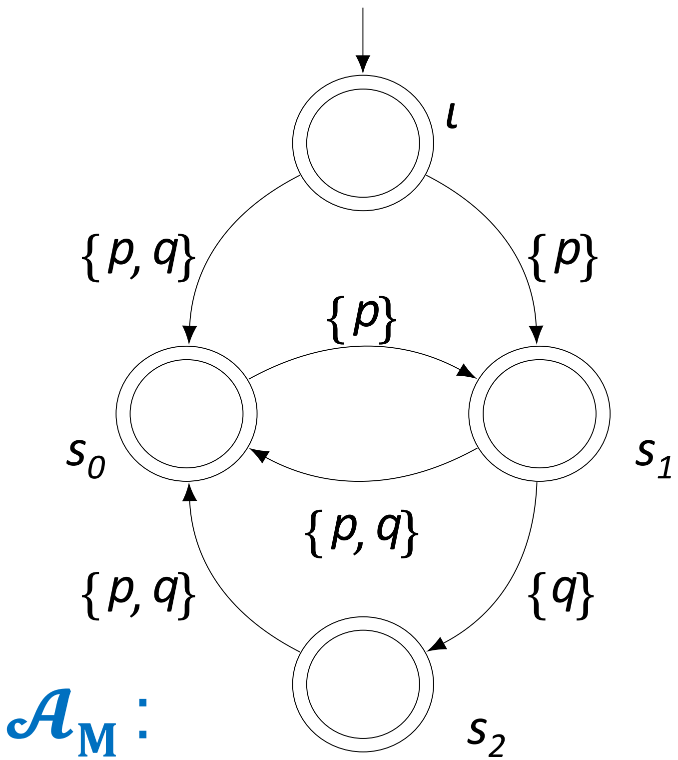
Automata and Kripke Structures

$\mathbf{M} = (\mathbf{S}, \mathbf{S}_0, \mathbf{R}, AP, \mathbf{L}) \;\Rightarrow\; \mathcal{A}_\mathbf{M} = (\Sigma, \mathbf{S}\cup\{\iota\}, \Delta, \{\iota\}, \mathbf{S}\cup\{\iota\})$,

where $\Sigma = P(AP)$.

- $(s,\alpha,s') \in \Delta$ for $s,s' \in \mathbf{S} \Leftrightarrow (s,s') \in R$ and $\alpha = \mathbf{L}(s')$





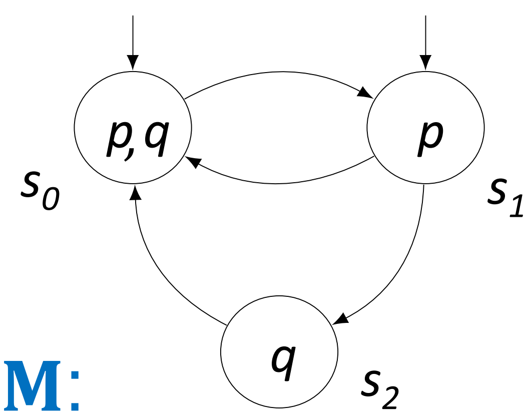
Automata and Kripke Structures

$\mathbf{M} = (\mathbf{S}, \mathbf{S}_0, \mathbf{R}, \mathrm{AP}, \mathbf{L}) \Rightarrow \mathcal{A}_{\mathbf{M}} = (\boldsymbol{\Sigma}, \mathbf{S} \cup \{\boldsymbol{\iota}\}, \boldsymbol{\Delta}, \{\boldsymbol{\iota}\}, \mathbf{S} \cup \{\boldsymbol{\iota}\})$,

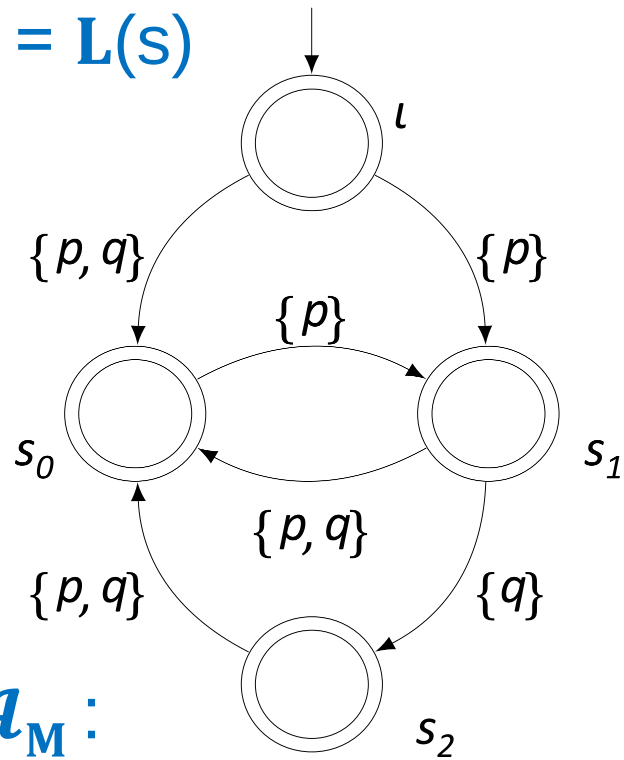
where $\boldsymbol{\Sigma} = P(\mathrm{AP})$.

- $(s,\alpha,s') \in \boldsymbol{\Delta}$ for $s,s' \in \mathbf{S} \Leftrightarrow (s,s') \in R$ and $\alpha = \mathbf{L}(s')$
- $(\boldsymbol{\iota},\alpha,s) \in \boldsymbol{\Delta} \Leftrightarrow s \in \mathbf{S}_0$ and $\alpha = \mathbf{L}(s)$

$\mathbf{M}$:



$\mathcal{A}_{\mathbf{M}}$:

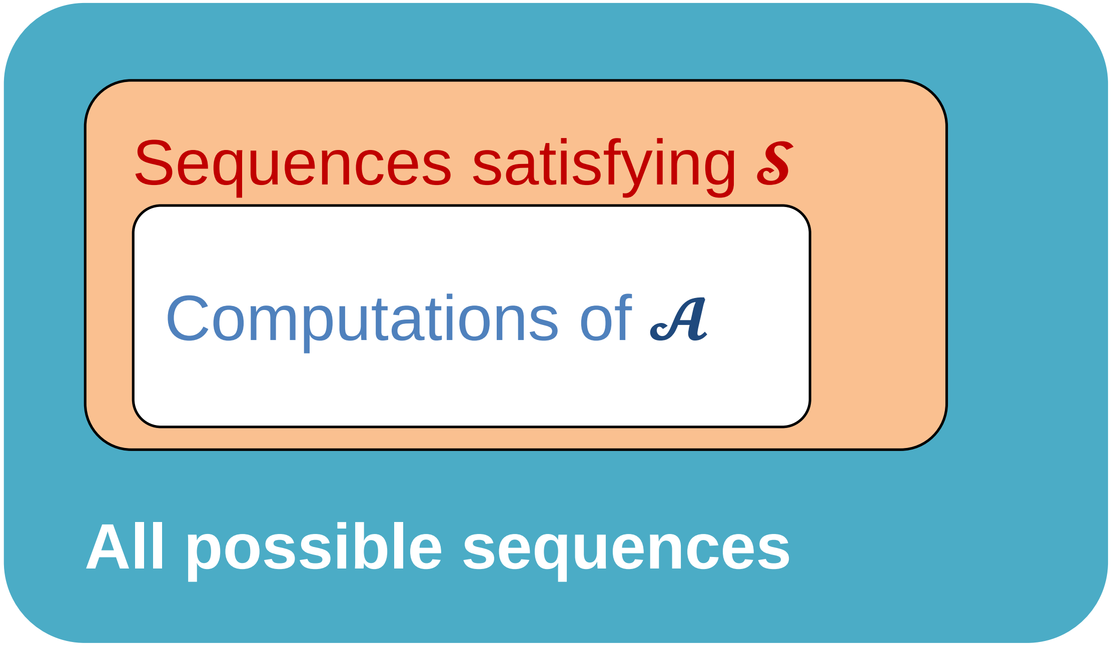


Outline

- Finite automata on finite words
- Automata on infinite words (Büchi automata)
- Deterministic vs non-deterministic Büchi automata
- Intersection of Büchi automata
- Checking emptiness of Büchi automata
- Generalized Büchi automata
- Automata and Kripke Structures
- Model checking using automata
- Translation of LTL to Büchi automata

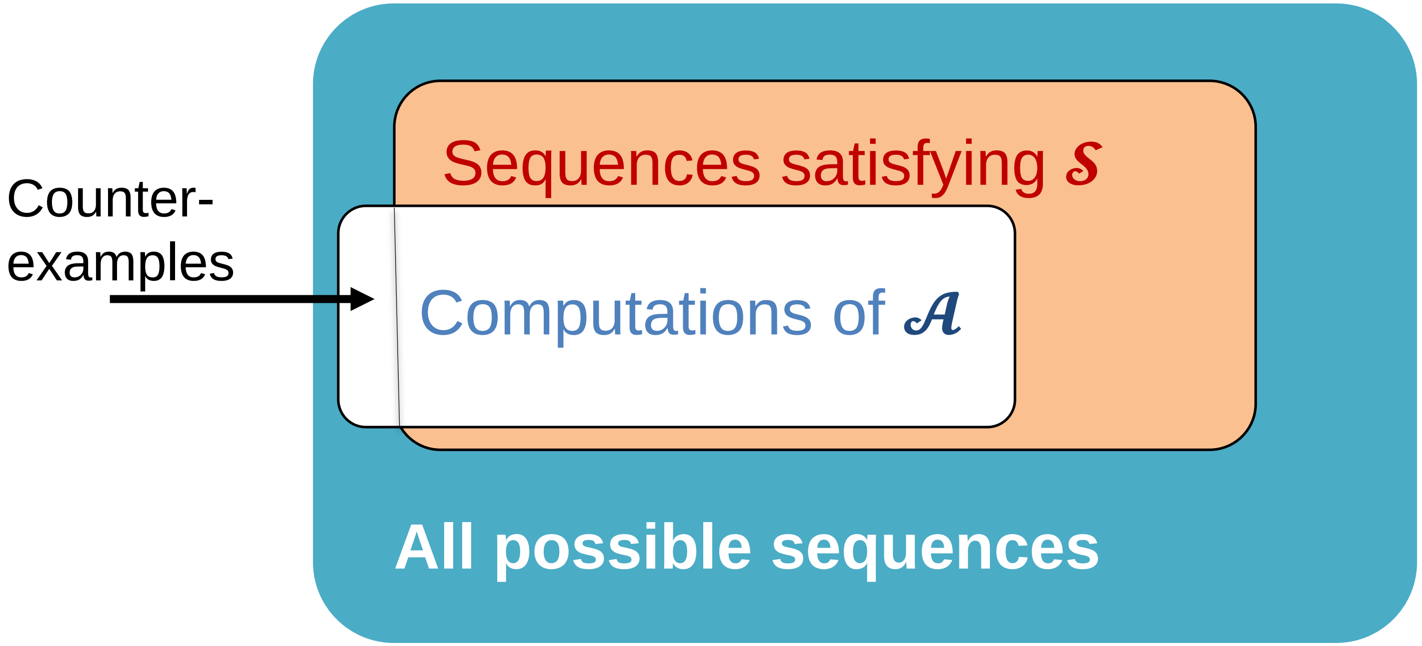
Model Checking when system $\mathcal{A}$ and spec $\mathcal{S}$ are given as Büchi automata

- $\mathcal{A}$ **satisfies** $\mathcal{S}$ if $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$



Model Checking when System $\mathcal{A}$ and Spec $\mathcal{S}$ are given as Büchi automata

- $\mathcal{A}$ **does not satisfy** $\mathcal{S}$ if $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{S})$

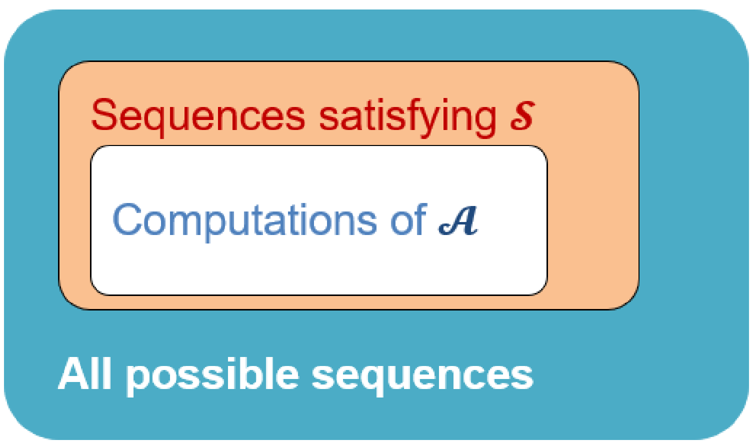


Model Checking when system $\mathcal{A}$ and spec $\mathcal{S}$ are given as Büchi automata

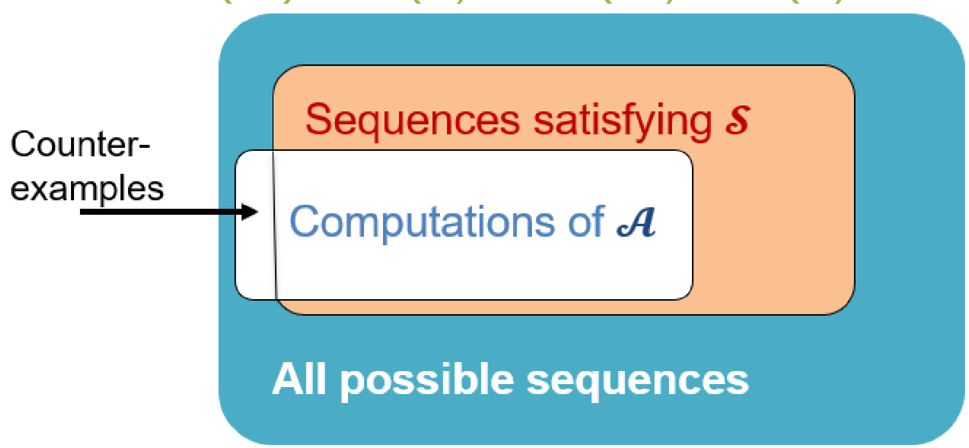
- Check whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$
- Equivalent:

$$\mathcal{L}(\mathcal{A}) \nsubseteq \mathcal{L}(\mathcal{S}) \;\equiv\; \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\overline{\mathcal{S}}) \neq \emptyset$$

$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$



$\mathcal{L}(\mathcal{A}) \nsubseteq \mathcal{L}(\mathcal{S}) \;\equiv\; \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\overline{\mathcal{S}}) \neq \emptyset$



Model Checking – suggested algorithm

very hard!

1. Complement $\mathcal{S}$. The resulting Büchi automaton is $\overline{\mathcal{S}}$
2. Construct the automaton $\mathcal{B}$ with $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\overline{\mathcal{S}})$
3. If $\mathcal{L}(\mathcal{B}) = \emptyset \Rightarrow \mathcal{A}$ satisfies $\mathcal{S}$
4. Otherwise, a word $v \cdot w^{\omega} \in \mathcal{L}(\mathcal{B})$ is a counterexample
   - a computation in $\mathcal{A}$ that does not satisfy $\mathcal{S}$

ToDo

How can we avoid building the complement of $\mathcal{S}$?

Model Checking of LTL

given an LTL property φ and a Kripke structure M
check whether $M \vDash \varphi$

next topic

1. Construct $\neg\varphi$
2. Construct a Büchi automaton $\mathcal{S}_{\neg\varphi}$
3. Translate M to an automaton $\mathcal{A}$.
4. Construct the automaton $\mathcal{B}$ with $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{S}_{\neg\varphi})$
5. If $\mathcal{L}(\mathcal{B}) = \emptyset \Rightarrow \mathcal{A}$ satisfies φ
6. Otherwise, a word $v \cdot w^{\omega} \in \mathcal{L}(\mathcal{B})$ is a counterexample
   - a computation in M that does not satisfy φ


THANK
YOU!