

Algorithmen II

Peter Sanders

Übungen:

Moritz Kobitzsch und Dennis Schieferdecker

Institut für Theoretische Informatik, Algorithmik II

Web:

http://algo2.iti.kit.edu/AlgorithmenII_WS11.php

4 Anwendungen von DFS

Tiefensuchschema für $G = (V, E)$

unmark all nodes; **init**

foreach $s \in V$ **do**

if s is not marked **then**

mark s

// make s a root and grow

root(s)

// a new DFS-tree rooted at it.

DFS(s, s)

Procedure **DFS**($u, v : \text{NodeId}$)

// Explore v coming from u .

foreach $(v, w) \in E$ **do**

if w is marked **then** **traverseNonTreeEdge**(v, w)

else **traverseTreeEdge**(v, w)

mark w

DFS(v, w)

backtrack(u, v) // return from v along the incoming edge

DFS Nummerierung

init: $\text{dfsPos} = 1 : 1..n$

root(s): $\text{dfsNum}[s] := \text{dfsPos}++$

traverseTreeEdge(v, w): $\text{dfsNum}[w] := \text{dfsPos}++$

$$u \prec v \Leftrightarrow \text{dfsNum}[u] < \text{dfsNum}[v] .$$

Beobachtung:

Knoten auf dem Rekursionsstapel sind bzgl., \prec sortiert

Fertigstellungszeit

init: finishingTime=1 : 1..n

backtrack(u, v): finishTime[v] := finishingTime++

Starke Zusammenhangskomponenten

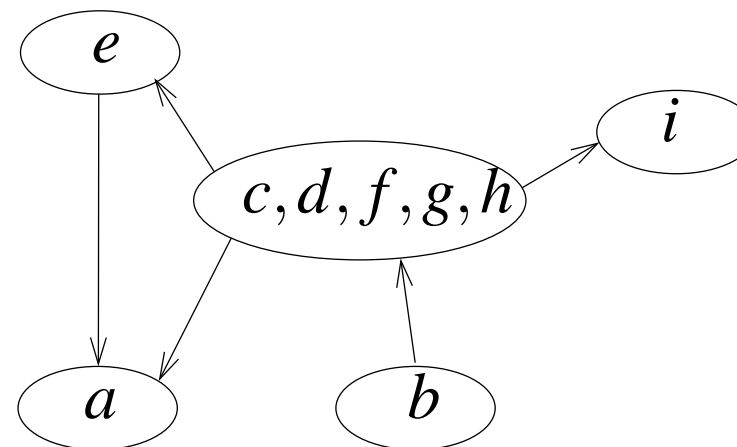
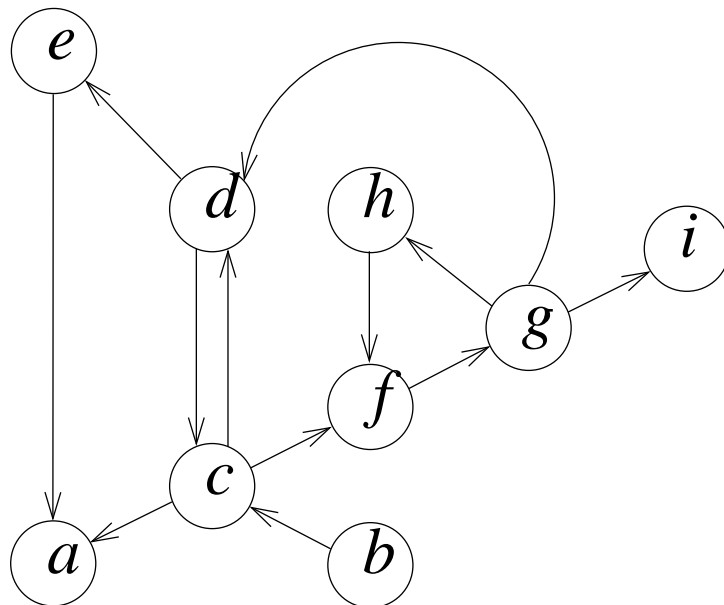
Betrachte die Relation $\overset{*}{\leftrightarrow}$ mit

$u \overset{*}{\leftrightarrow} v$ falls \exists Pfad $\langle u, \dots, v \rangle$ und \exists Pfad $\langle v, \dots, u \rangle$.

Beobachtung: $\overset{*}{\leftrightarrow}$ ist Äquivalenzrelation

Übung

Die Äquivalenzklassen von $\overset{*}{\leftrightarrow}$ bezeichnet man als **starke Zusammenhangskomponenten**.



Starke Zusammenhangskomponenten – Abstrakter Algorithmus

$G_c := (V, \emptyset = E_c)$

foreach edge $e \in E$ **do**

invariant SCCs of G_c are known

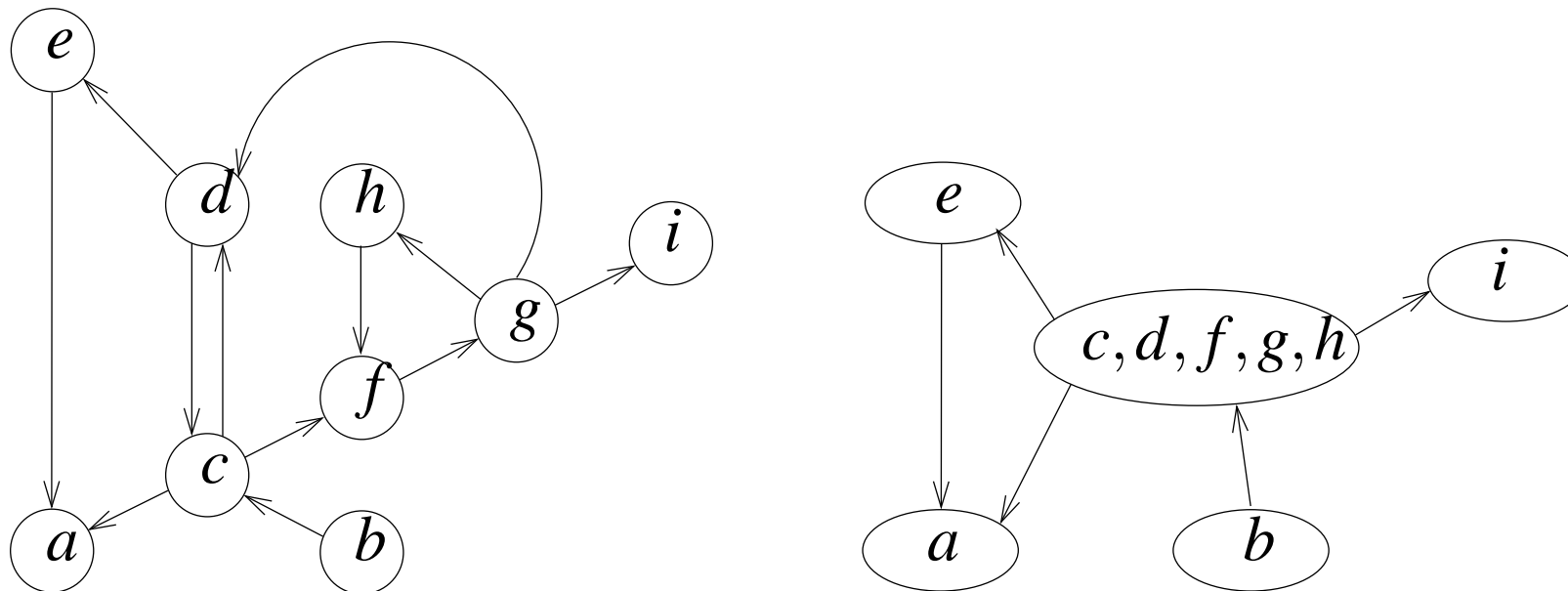
$E_c := E_c \cup \{e\}$

Schrumpfggraph

$$G_c^s = (V^s, E_c^s)$$

Knoten: SCCs von G_c .

Kanten: $(C, D) \in E_c^s \Leftrightarrow \exists (c, d) \in E_c : c \in C \wedge d \in D$



Beobachtung: Der Schrumpfggraph ist azyklisch

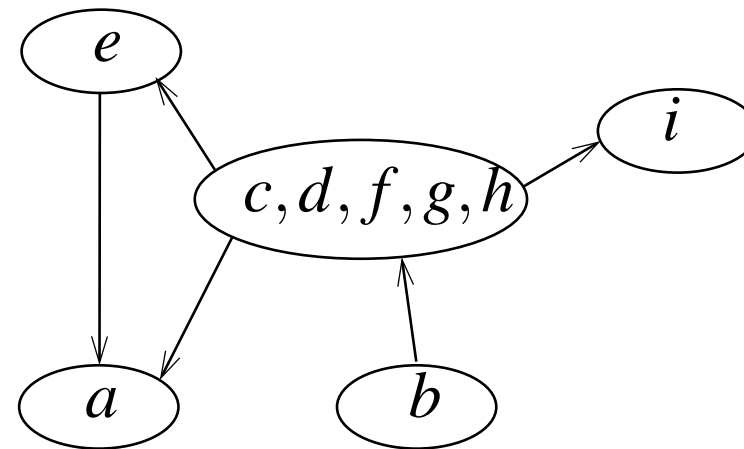
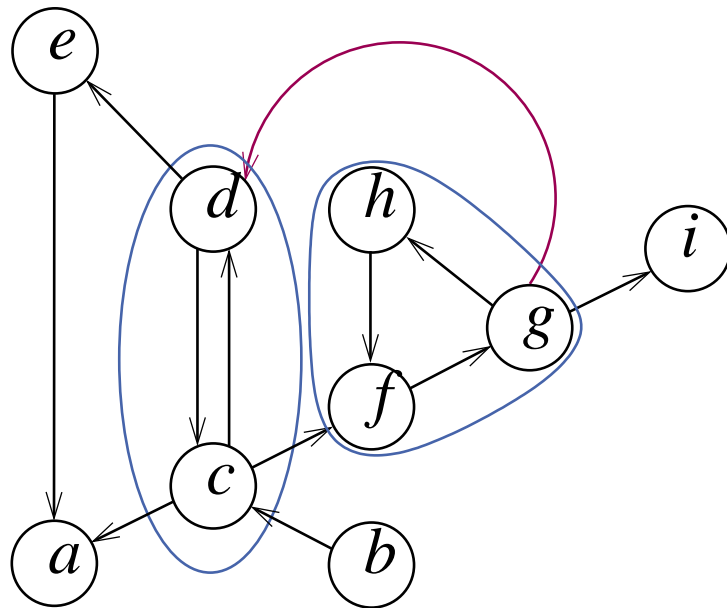
Auswirkungen einer neuen Kante e auf G_c, G_c^s

SCC-intern: Nichts ändert sich

zwischen zwei SCCs:

Kein Kreis: Neue Kante in G_c^s

Kreisschluss: SCCs auf Kreis kollabieren.



Konkreter: SCCs mittels DFS

[Cheriyān/Mehlhorn 96, Gabow 2000]

V_c = markierte Knoten

E_c = bisher explorierte Kanten

Aktive Knoten: markiert aber nicht finished.

SCCs von G_c :

nicht erreicht: Unmarkierte Knoten

offen: enthält aktive Knoten

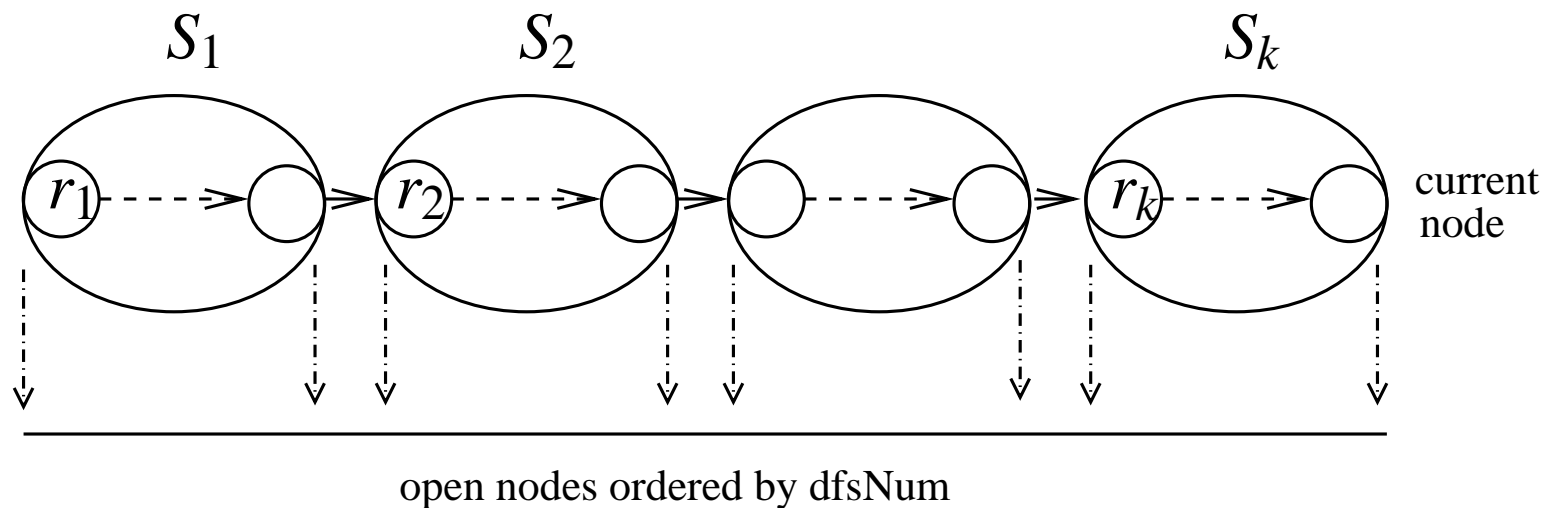
abgeschlossen: alle Knoten finished

component[w] gibt Repräsentanten einer SCC an.

Knoten von offenen (abgeschl.) Komponenten heißen offen (abgeschl.)

Invarianten von G_C

1. Kanten von abgeschlossenen Knoten gehen zu abgeschlossenen Knoten
2. Offene Komponenten S_1, \dots, S_k bilden Pfad in G_C^S .
3. Repräsentanten partitionieren die offenen Komponenten bzgl. ihrer dfsNum.



Lemma: Abgeschlossene SCCs von G_c sind SCCs von G

Betrachte abgeschlossenen Knoten v

und beliebigen Knoten w

in der SCC von v bzgl. G .

z.Z.: w ist abgeschlossen und

in der gleichen SCC von G_c wie v .

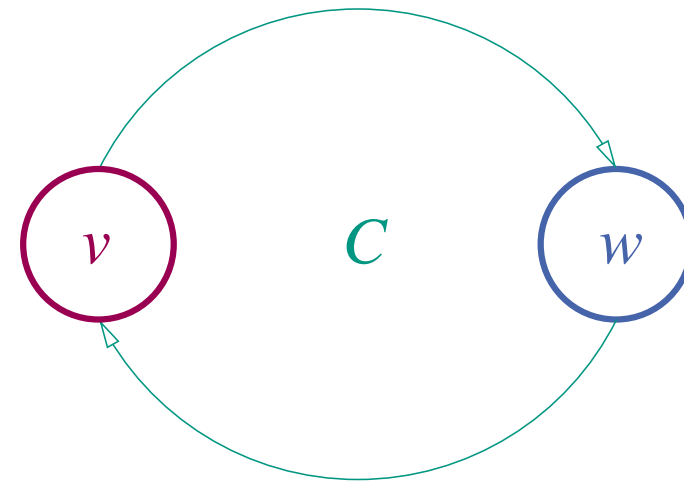
Betrachte Kreis C durch v, w .

Inv. 1: **Knoten** von C sind abgeschlossen.

Abgeschl. Knoten sind finished.

Kanten aus finished Knoten wurden exploriert.

Also sind alle **Kanten** von C in G_c .



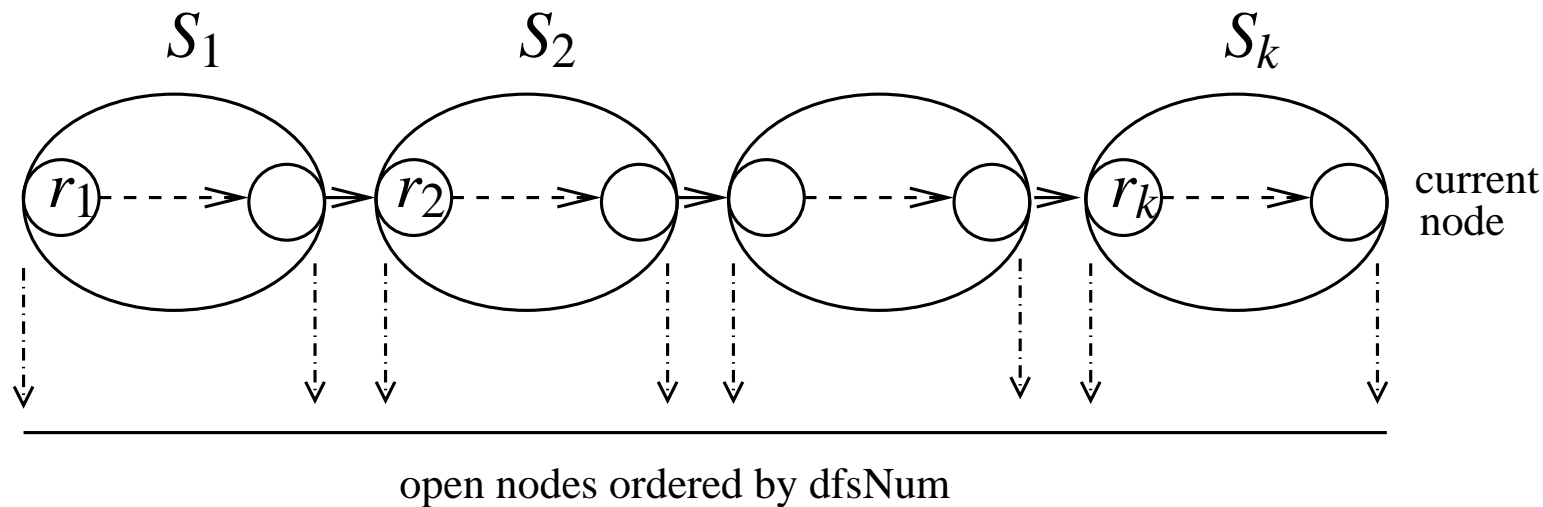
□

Repräsentation offener Komponenten

Zwei Stapel aufsteigend sortiert nach dfsNum

oReps: Repräsentanten offener Komponenten

oNodes: Alle offenen Knoten



init

component : NodeArray **of** NodeId // SCC representatives

oReps= $\langle \rangle$: Stack **of** NodeId // representatives of open SCCs

oNodes= $\langle \rangle$: Stack **of** NodeId // all nodes in open SCCs

Alle Invarianten erfüllt.

(Weder offene noch geschlossene Knoten)

$\text{root}(s)$

$\text{oReps.push}(s)$

$\text{oNodes.push}(s)$

// new open

// component

$\{s\}$ ist die einzige offene Komponente.

Alle Invarianten bleiben gültig

`traverseTreeEdge(v, w)`

`oReps.push(w)`

`oNodes.push(w)`

$\{w\}$ ist neue offene Komponente.

$\text{dfsNum}(w) >$ alle anderen.

\rightsquigarrow Alle Invarianten bleiben gültig

// new open

// component

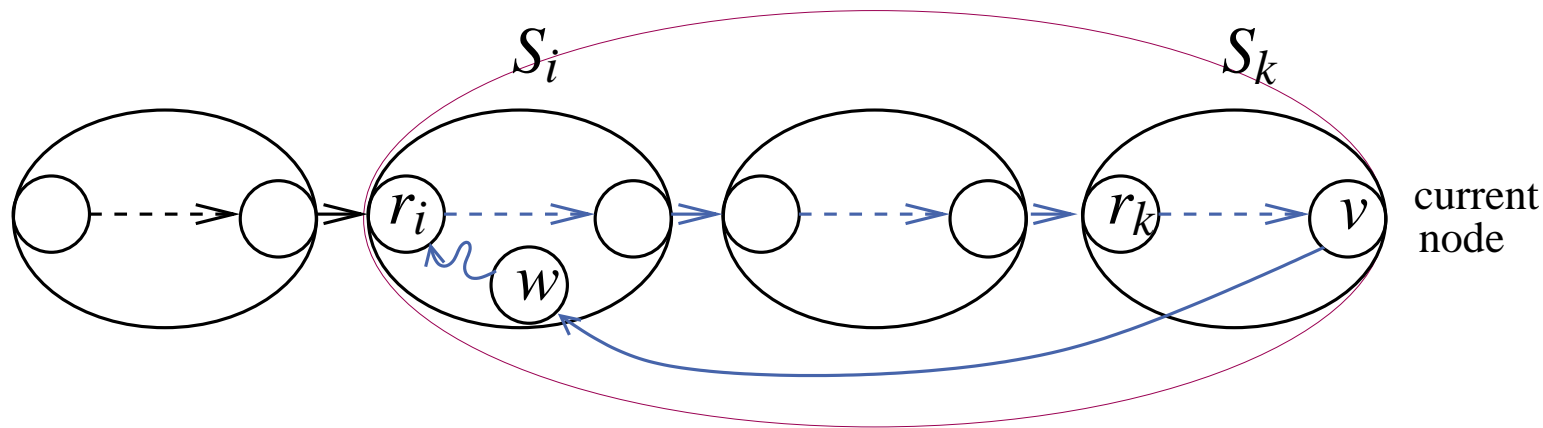
traverseNonTreeEdge(v, w)

if $w \in \text{oNodes}$ **then**

while $w \prec \text{oReps.top}$ **do** oReps.pop

$w \notin \text{oNodes} \rightsquigarrow w$ is abgeschlossen $\overset{\text{Lemma}(*)}{\rightsquigarrow}$ Kante uninteressant

$w \in \text{oNodes}$: kollabiere offene SCCs auf **Kreis**



backtrack(u, v)

if $v = \text{oReps.top}$ **then**

oReps.pop

// close

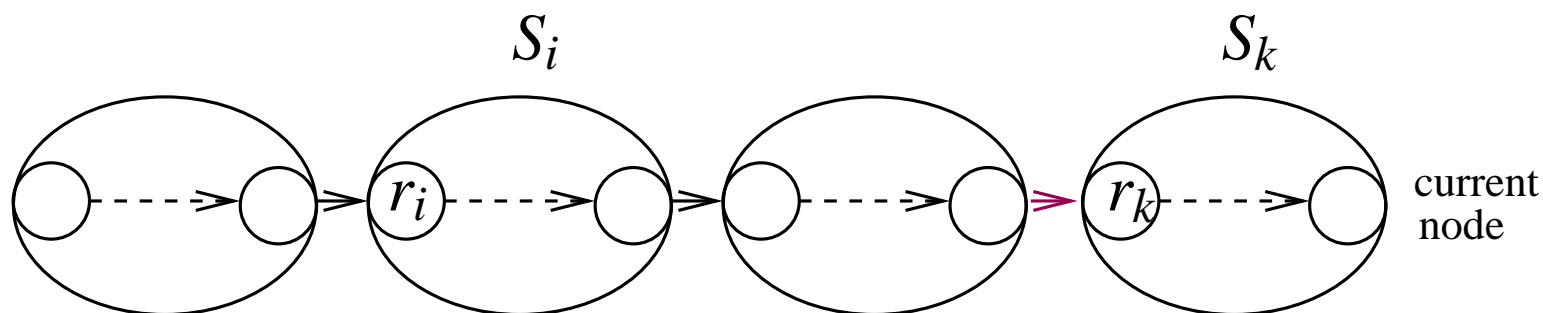
repeat

// component

$w := \text{oNodes.pop}$

component[w] := v

until $w = v$



z.Z. Invarianten bleiben erhalten...

backtrack(u, v)

if $v = \text{oReps.top}$ **then**

oReps.pop

repeat

$w := \text{oNodes.pop}$

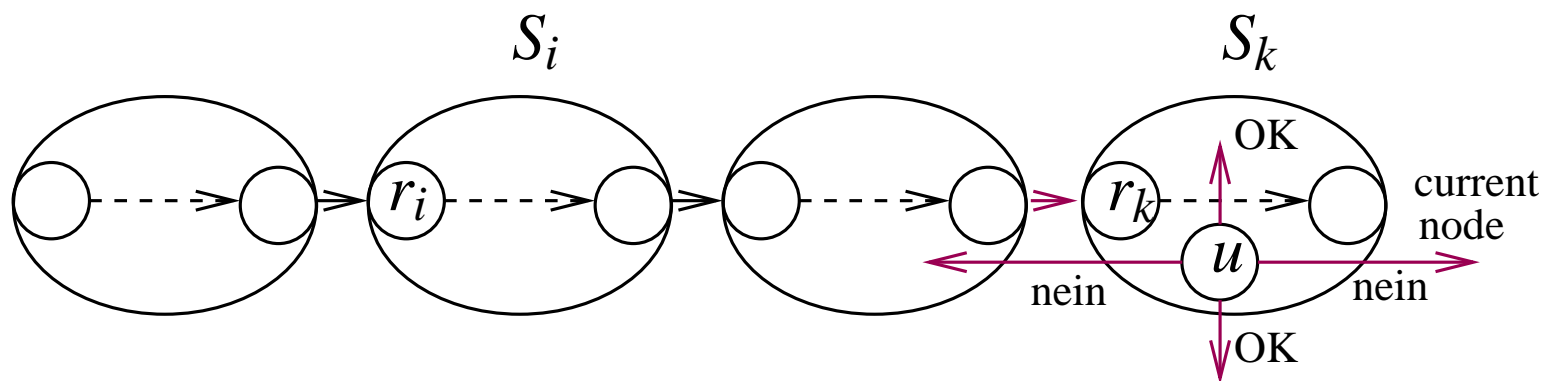
component[w] := v

until $w = v$

// close

// component

Inv. 1: Kanten von abgeschlossenen Knoten gehen zu abgeschlossenen Knoten.



backtrack(u, v)

if $v = \text{oReps.top}$ **then**

oReps.pop

repeat

$w := \text{oNodes.pop}$

component[w] := v

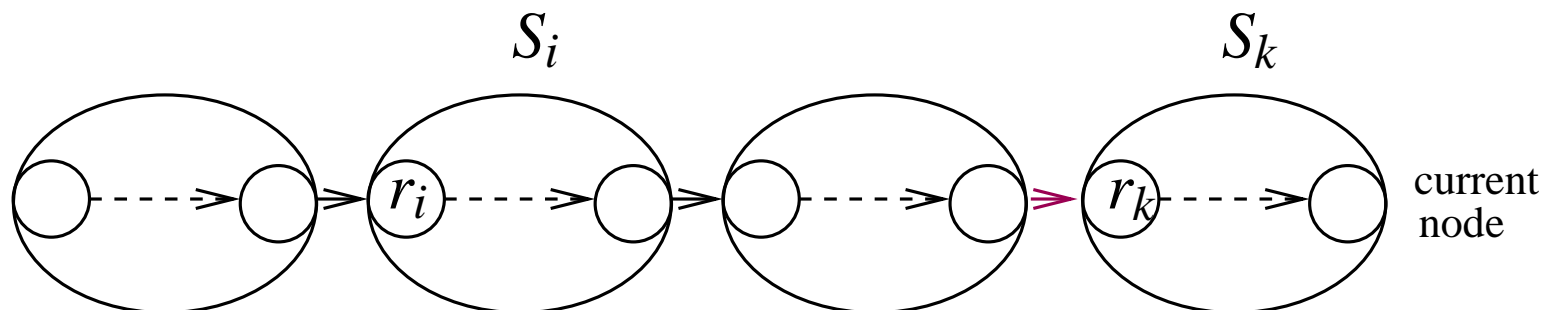
until $w = v$

// close

// component

Inv. 2: Offene Komponenten S_1, \dots, S_k bilden Pfad in G_c^s

OK. (S_k wird ggf. entfernt)

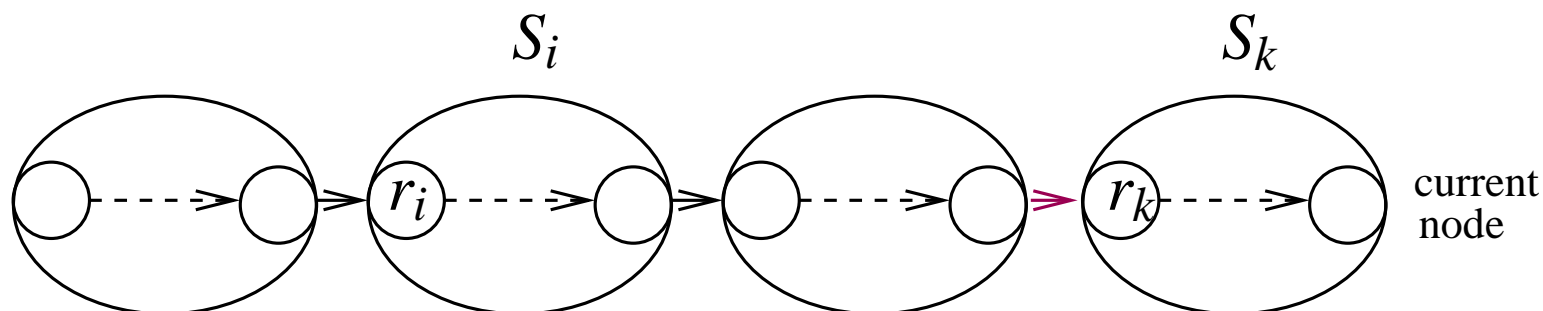


```

backtrack( $u, v$ )
  if  $v = \text{oReps.top}$  then
    oReps.pop // close
  repeat // component
     $w := \text{oNodes.pop}$ 
    component[ $w$ ] :=  $v$ 
  until  $w = v$ 
    
```

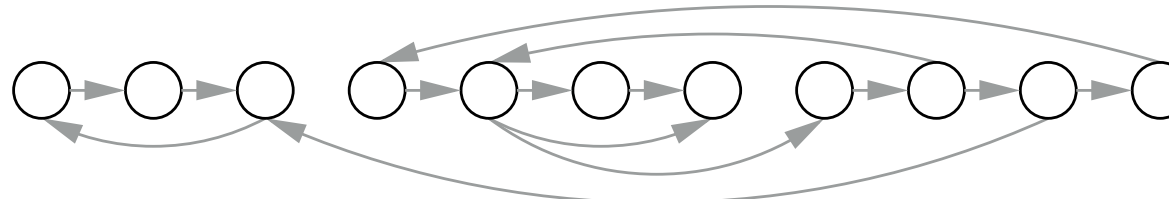
Inv. 3: Repräsentanten partitionieren die offenen Komponenten bzgl. ihrer dfsNum.

OK. (S_k wird ggf. entfernt)

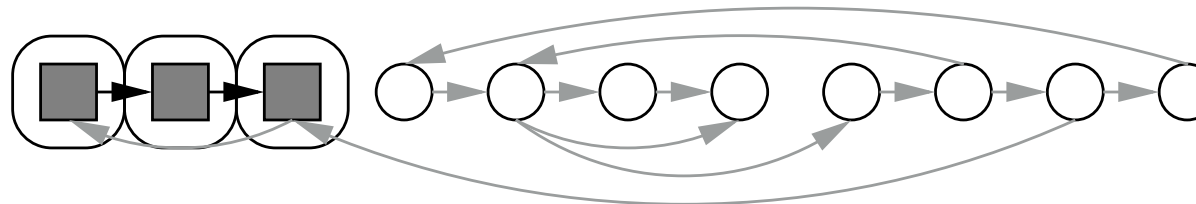


Beispiel

a b c d e f g h i j k



root(a) traverse(a,b) traverse(b,c)



unmarked marked finished



nonrepresentative node



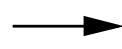
representative node



nontraversed edge



closed SCC

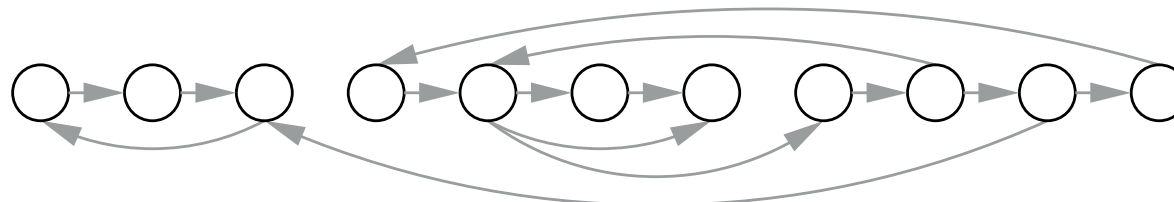


traversed edge

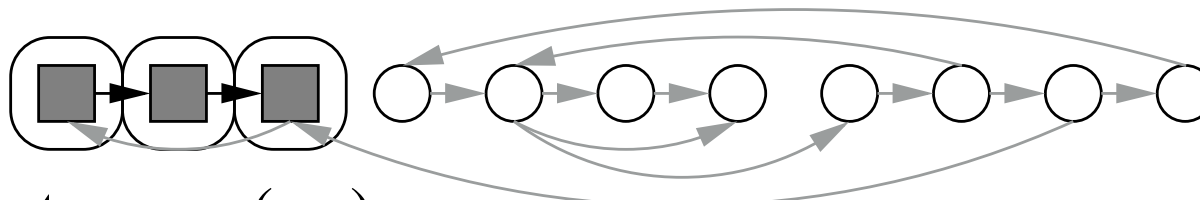


open SCC

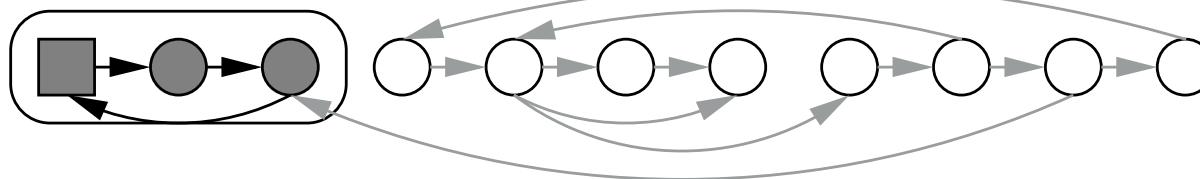
a b c d e f g h i j k



root(a) traverse(a,b) traverse(b,c)



traverse(c,a)



unmarked marked finished



nonrepresentative node



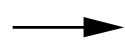
representative node



nontraversed edge



closed SCC

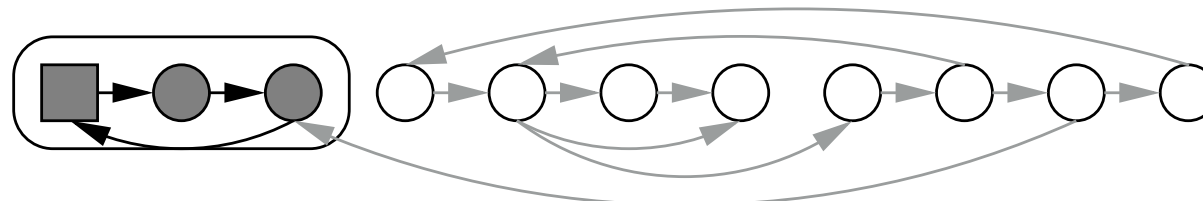


traversed edge

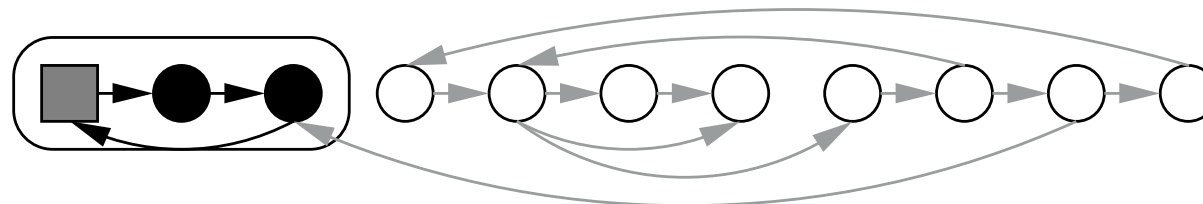


open SCC

a b c d e f g h i j k



backtrack(b,c) backtrack(a,b)



unmarked marked finished



nonrepresentative node



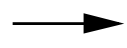
representative node



nontraversed edge



closed SCC

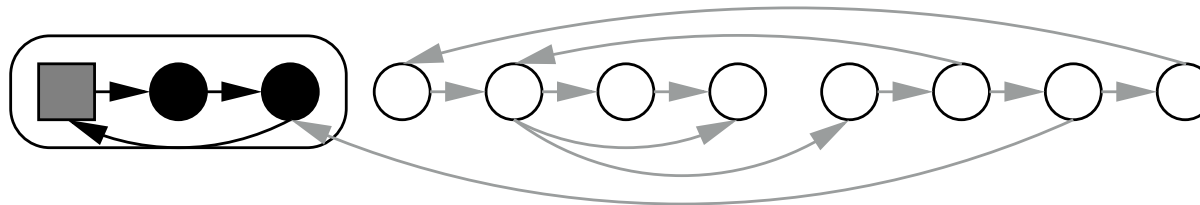


traversed edge

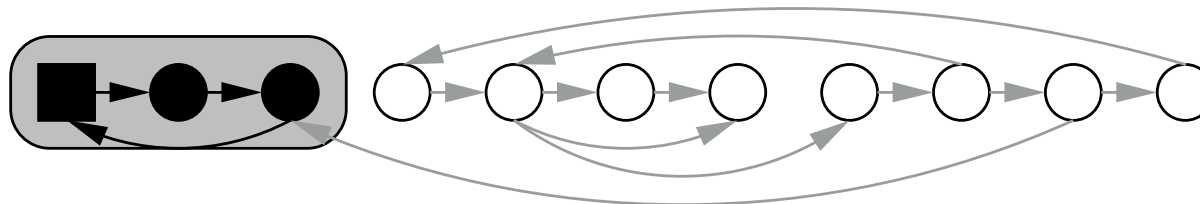


open SCC

a b c d e f g h i j k



backtrack(a,a)



unmarked marked finished



nonrepresentative node

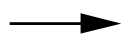


representative node



nontraversed edge

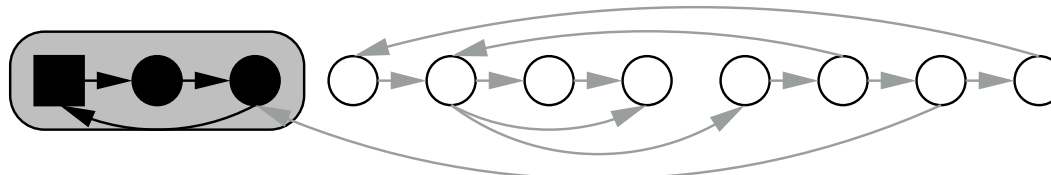
closed SCC



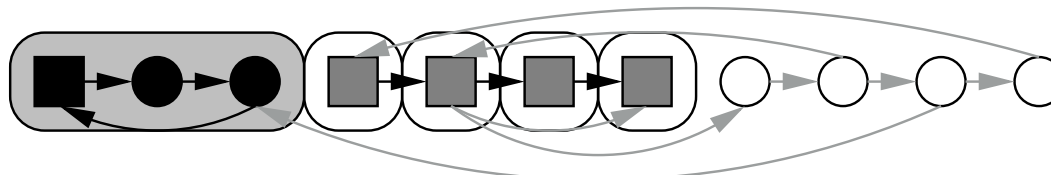
traversed edge

open SCC

a b c d e f g h i j k



root(d) traverse(d,e) traverse(e,f) traverse(f,g)



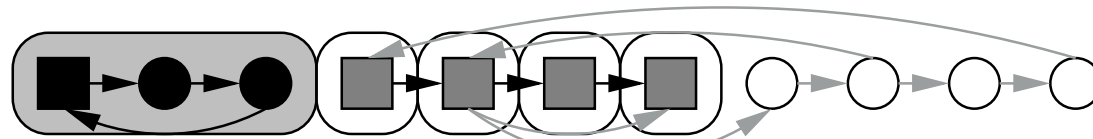
unmarked marked finished

○ ● ● nonrepresentative node
 ■ ■ representative node

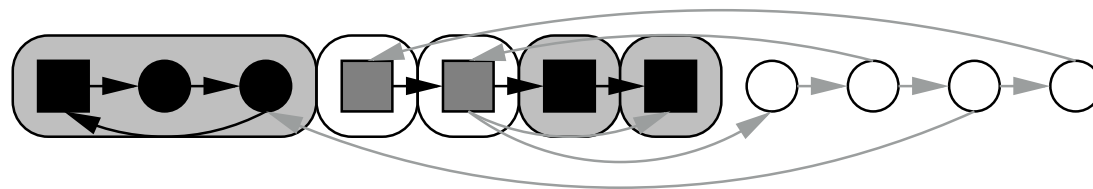
→ nontraversed edge closed SCC

→ traversed edge open SCC

a b c d e f g h i j k



backtrack(f,g) backtrack(e,f)



unmarked marked finished



nonrepresentative node

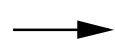


representative node



nontraversed edge

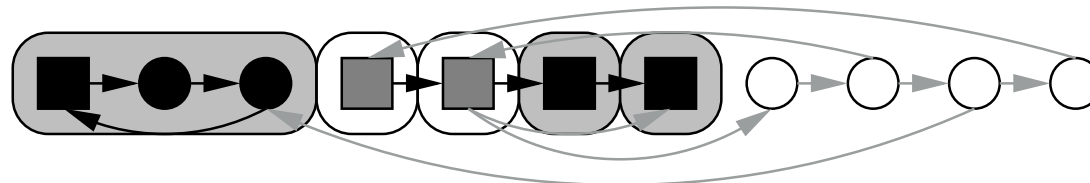
closed SCC



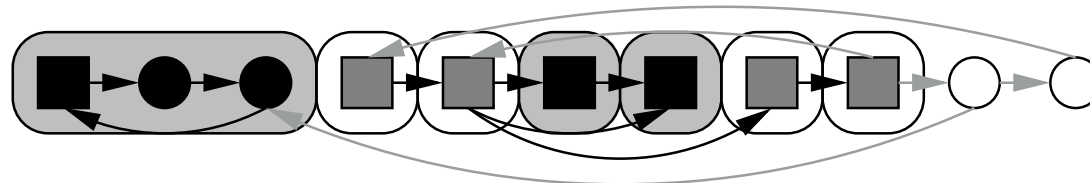
traversed edge

open SCC

a b c d e f g h i j k



traverse(e,g) traverse(e,h) traverse(h,i)



unmarked marked finished



nonrepresentative node



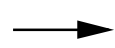
representative node



nontraversed edge



closed SCC

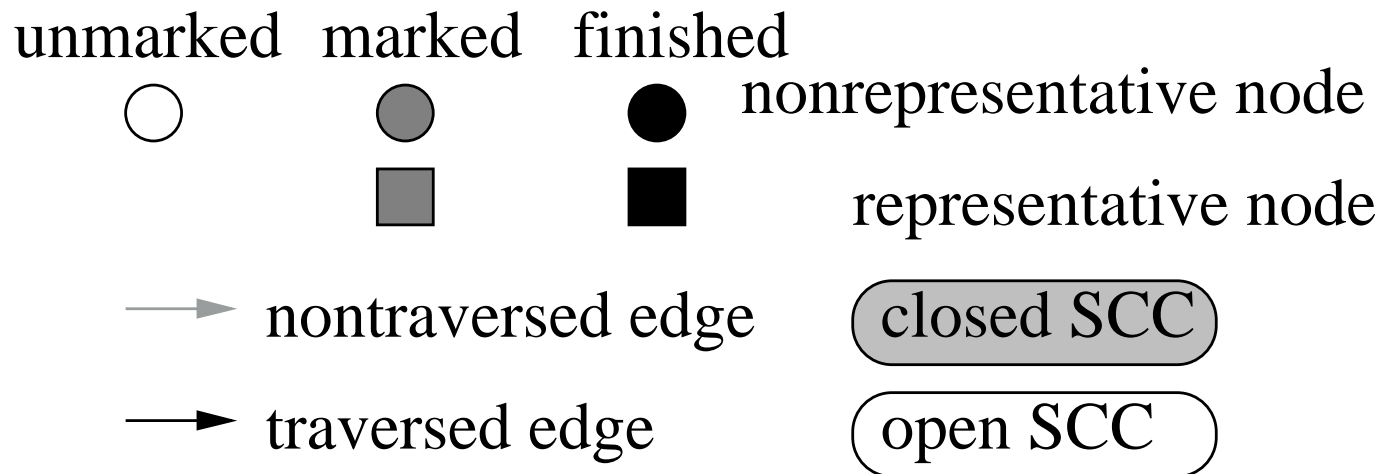
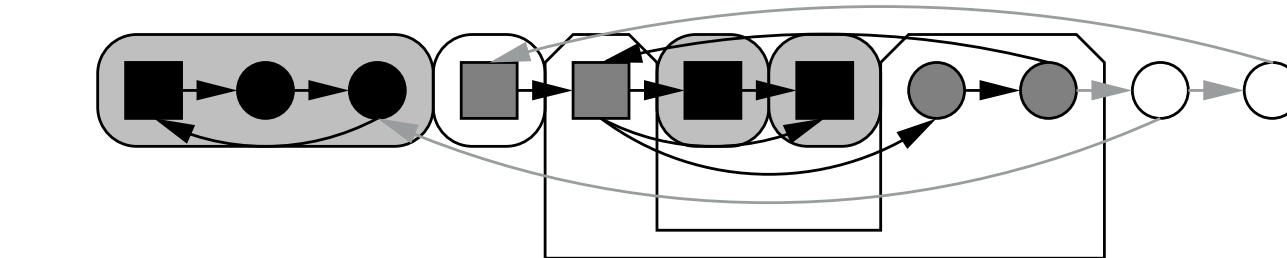
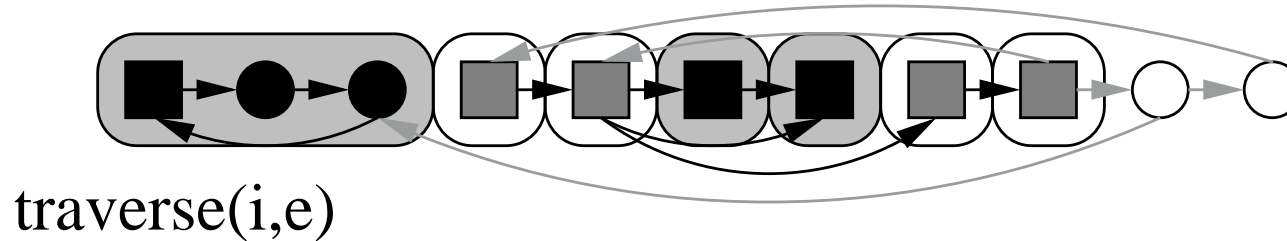


traversed edge

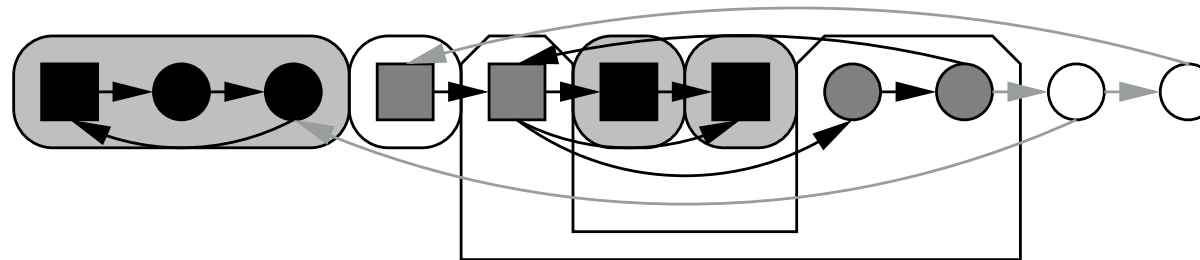


open SCC

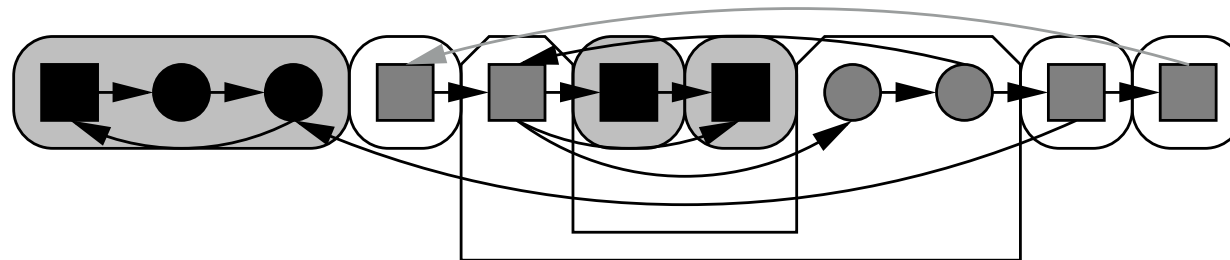
a b c d e f g h i j k



a b c d e f g h i j k



traverse(i,j) traverse(j,c) traverse(j,k)



unmarked marked finished



nonrepresentative node



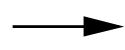
representative node



nontraversed edge



closed SCC

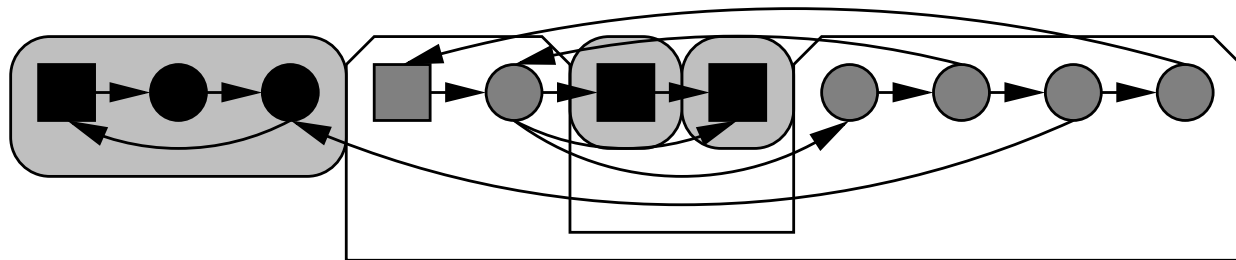
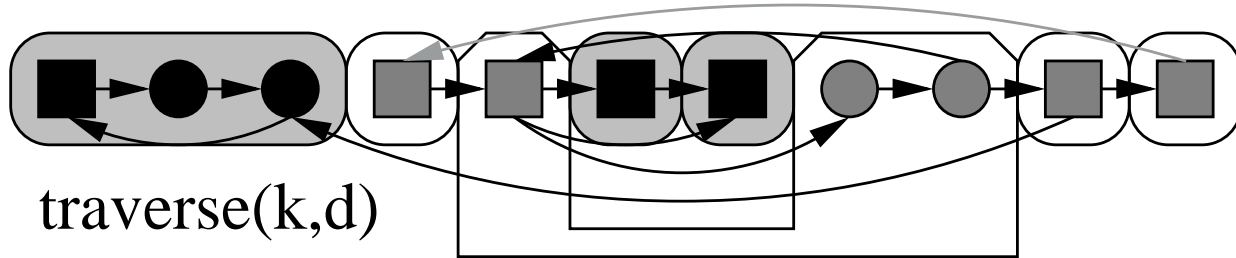


traversed edge



open SCC

a b c d e f g h i j k



unmarked marked finished



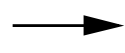
nonrepresentative node



representative node



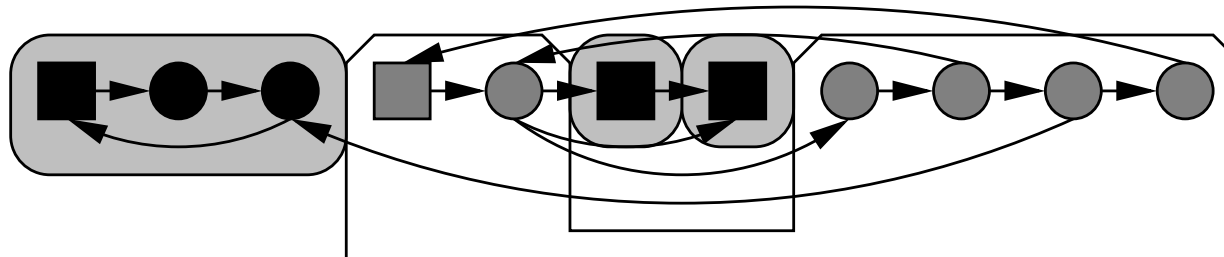
nontraversed edge



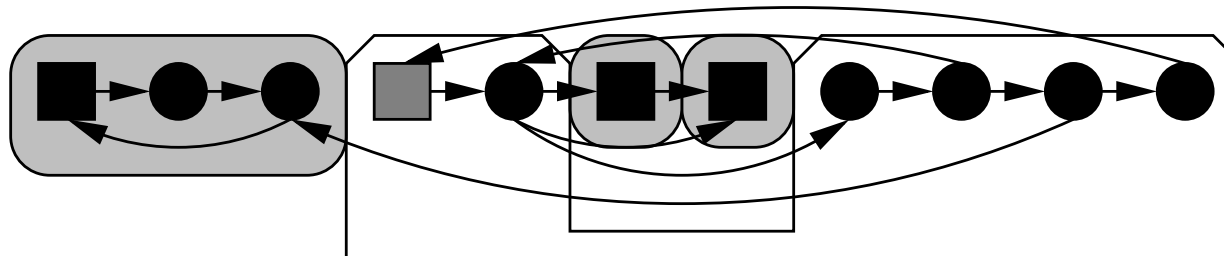
traversed edge



a b c d e f g h i j k



backtrack(j,k) backtrack(i,j) backtrack(h,i)
backtrack(e,h) backtrack(d,e)



unmarked marked finished



nonrepresentative node



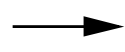
representative node



nontraversed edge



closed SCC

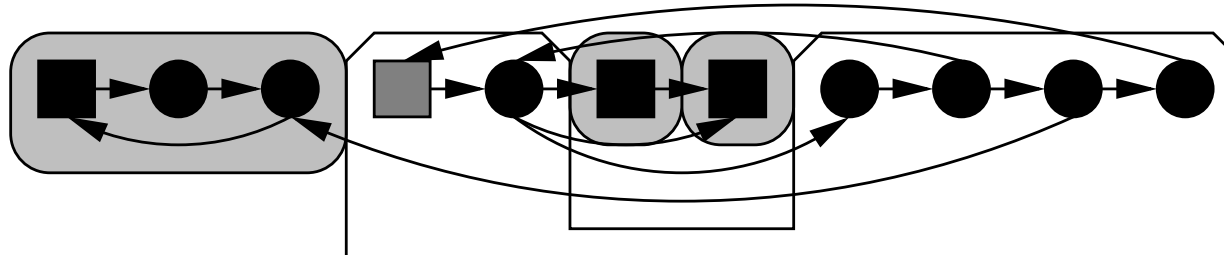


traversed edge

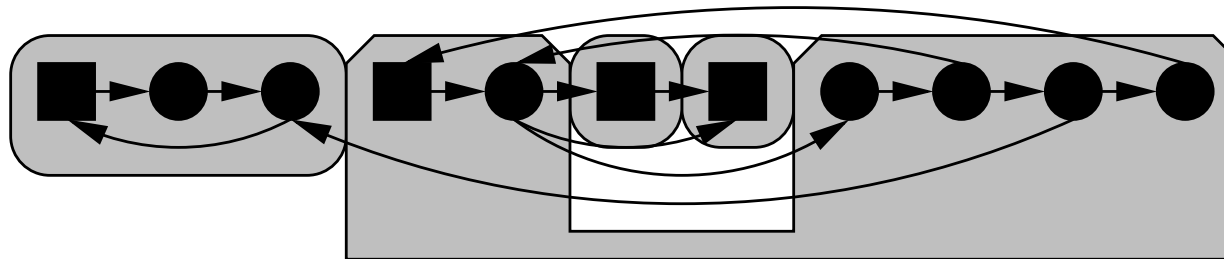


open SCC

a b c d e f g h i j k



backtrack(d,d)



unmarked marked finished



nonrepresentative node



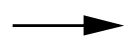
representative node



nontraversed edge



closed SCC



traversed edge



open SCC

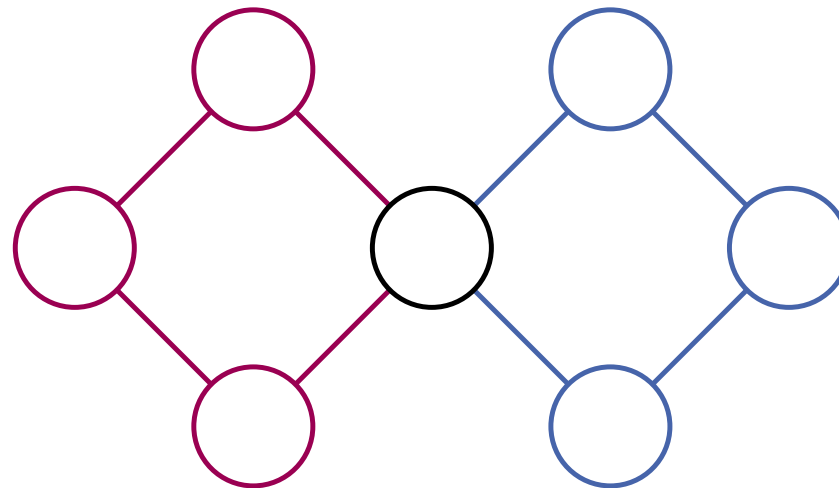
Zusammenfassung: SCC Berechnung

- Einfache Instantiierung des DFS-Musters
- Nichttrivialer Korrektheitsbeweis
- Laufzeit $O(m + n)$: (Jeweils max. n push/pop Operationen)
- Ein einziger Durchlauf

2-zusammenhängende Komponenten (ungerichtet)

Bei entfernen eines Knotens bleibt die Komponente
zusammenhängend.

(Partitionierung der **Kanten**)



Geht in Zeit $O(m + n)$ mit Algorithmus ähnlich zu SCC-Algorithmus



Mehr DFS-basierte Linearzeitalgorithmen

- 3-zusammenhängende Komponenten
- Planaritätstest
- Einbettung planarer Graphen