CTL Model Checking

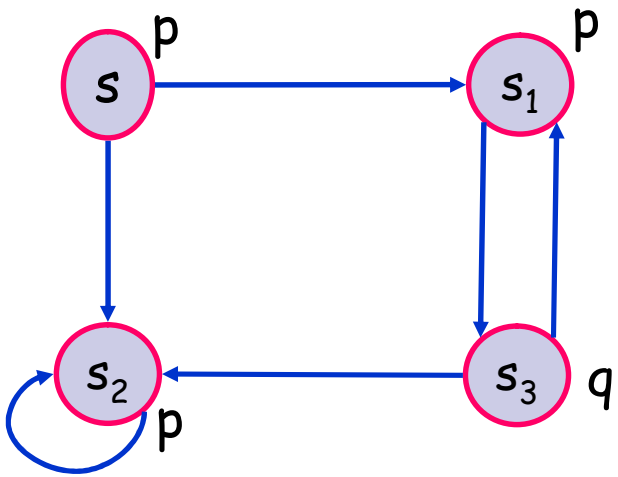
H. Peter Gumm

Philipps-Universität Marburg

Sommersemester 2003

Aufgabenstellung

- $K=(S,S_0,\rightarrow,L)$ Kripke-Struktur
  - $s \in S$
  - ψ CTL-Formel
- Gilt für ein $s \in S_0$: $s \models \psi$?



$s \models E[\, p \; U \; q \,]$

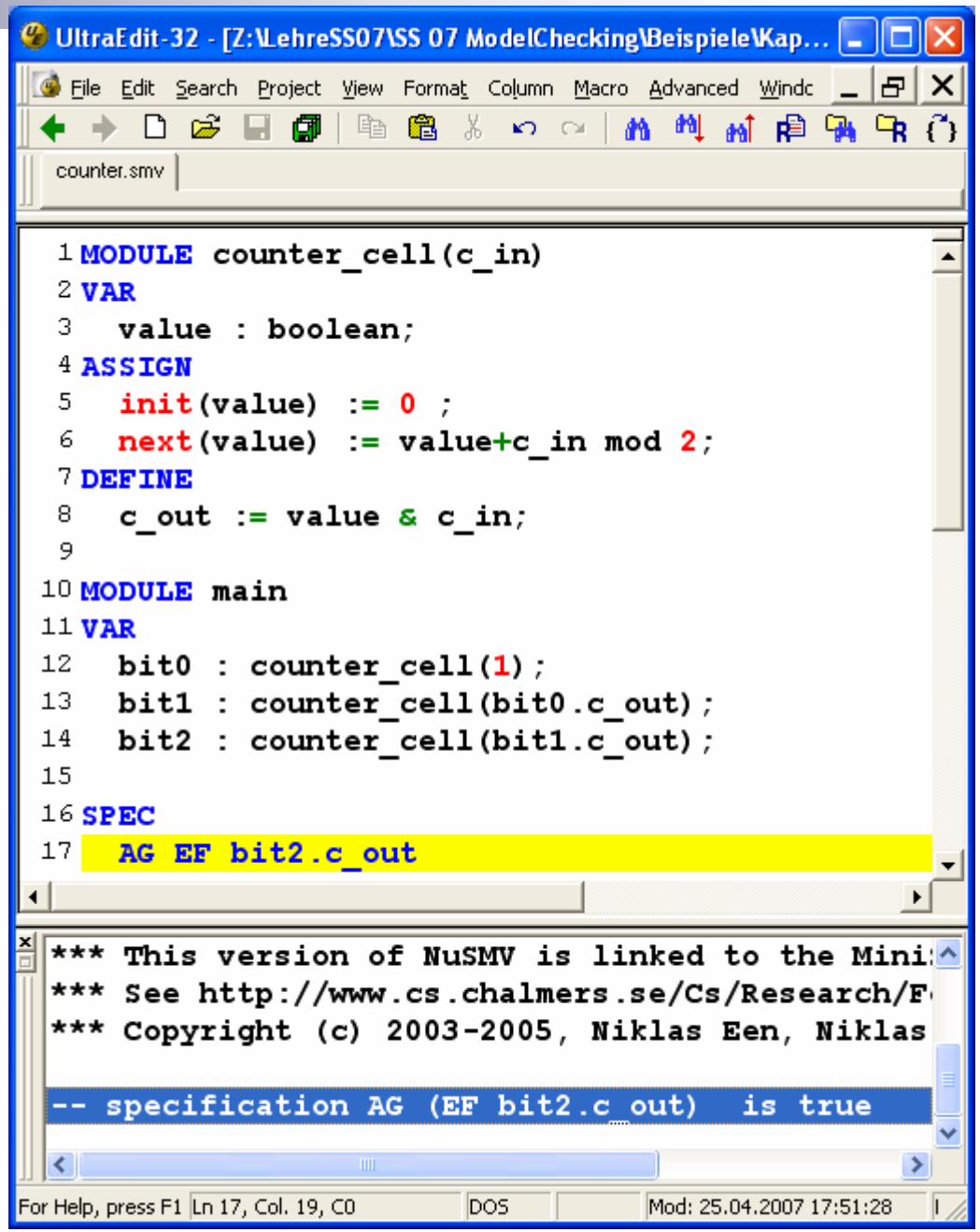
© H. Peter Gumm, Philipps-Universität Marburg

Im Falle von SMV

- Kripke-Struktur: SMV-Programm
- $S_0 \subseteq S$ Anfangszustände
- **SPEC** ψ bedeutet:

Gilt für alle $s \in S_0$:

$s \models \psi$



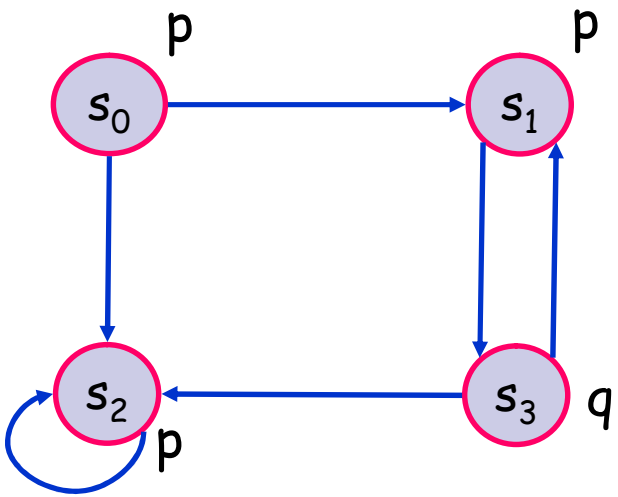
© H. Peter Gumm, Philipps-Universität Marburg

Allgemeineres Problem ist leichter:

- K=(S,R,L) Kripke Struktur
- ψ CTL-Formel
- Bestimme $[\psi] := \{ s \in S \mid \text{mit } s \models \psi \}$

Beispiel: Finde alle Zustände s mit

$s \models E[\, p \; U \; q \,]$



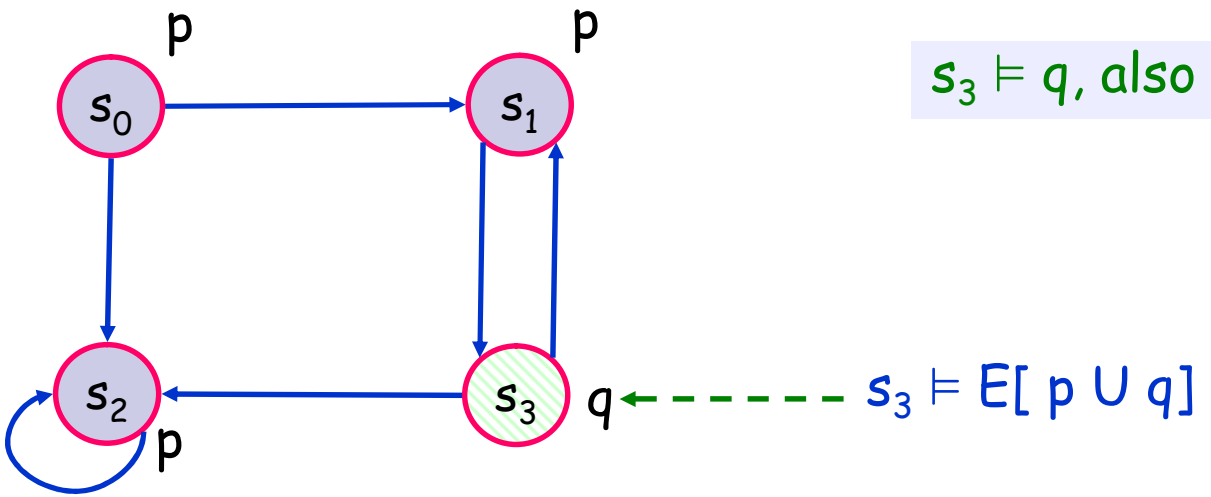
© H. Peter Gumm, Philipps-Universität Marburg

Allgemeineres Problem ist leichter

- K=(S,R,L) Kripke Struktur
- ψ CTL-Formel
- Bestimme $[\psi] := \{ s \in S \mid \text{mit } s \models \psi \}$

Beispiel: Finde alle Zustände s mit

$s \models E[\, p \; U \; q \,]$



$s_3 \models q$, also

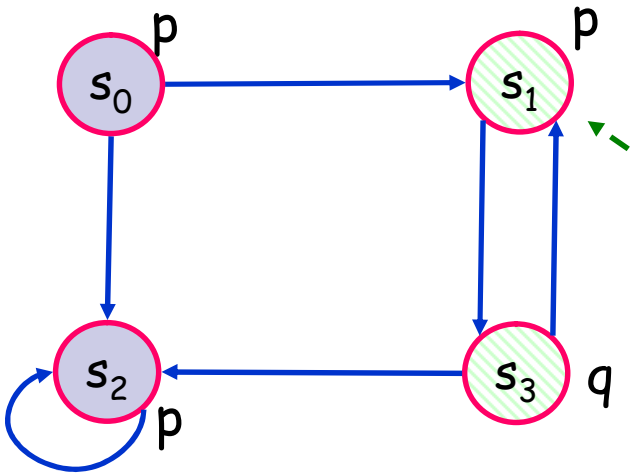
© H. Peter Gumm, Philipps-Universität Marburg

Allgemeineres Problem ist leichter:

- K=(S,R,L) Kripke Struktur
- ψ CTL-Formel
- Bestimme $[\psi] := \{ s \in S \mid \text{mit } s \models \psi \}$

Beispiel: Finde alle Zustände s mit

$s \models E[\, p \; U \; q \,]$



$s_1 \; R \; s_3$.
$s_1 \models p$ und
$s_3 \models E[pUq]$, also

$s_1 \models E[\, p \; U \; q]$
$s_3 \models E[\, p \; U \; q]$

© H. Peter Gumm, Philipps-Universität Marburg

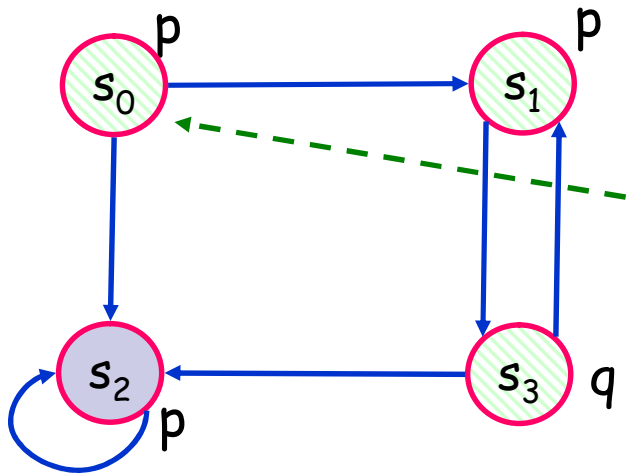
Allgemeineres Problem ist leichter:

- K=(S,R,L) Kripke Struktur
- ψ CTL-Formel
- Bestimme $[\psi] := \{ s \in S \mid \text{mit } s \models \psi \}$

Beispiel: Finde alle Zustände s mit

$s \models E[\, p \; U \; q \,]$

$[\, E[pUq] \,] = \{s_0, s_1, s_3\}$

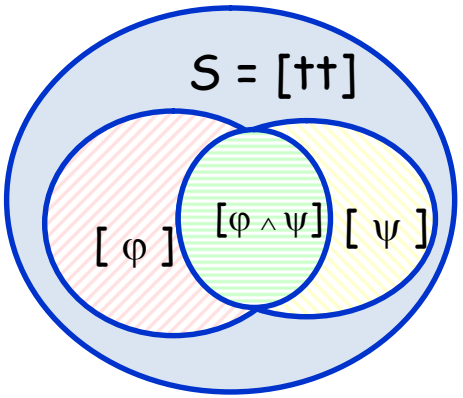


s_0 R s_1 .
$s_0 \models$ p und
$s_1 \models E[pUq]$, also

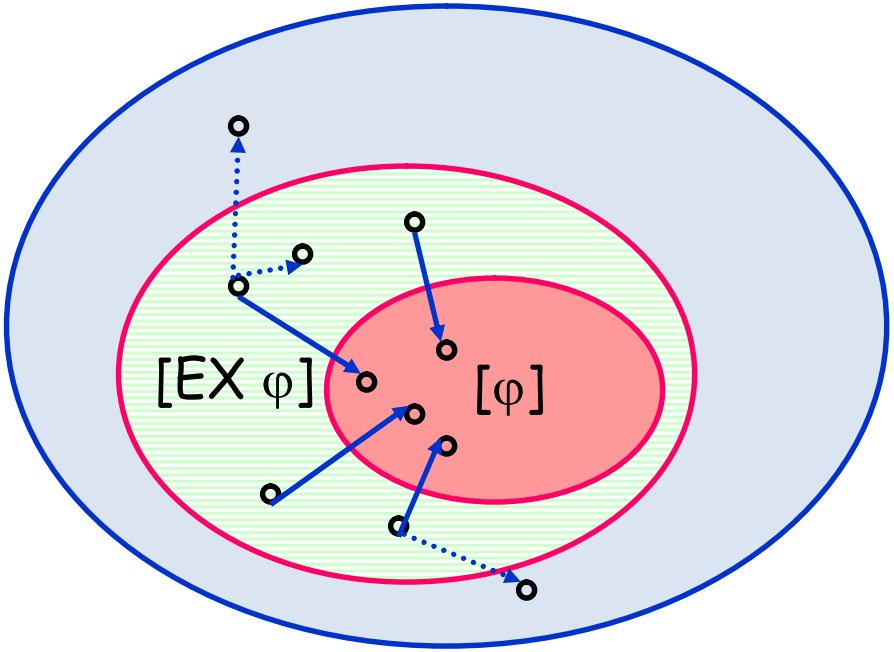
$s_0 \models E[\, p \; U \; q]$
$s_1 \models E[\, p \; U \; q]$
$s_3 \models E[\, p \; U \; q]$

© H. Peter Gumm, Philipps-Universität Marburg

Strategie



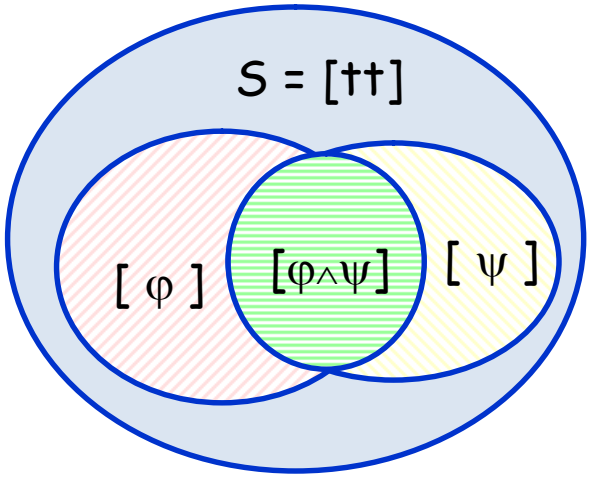
- Gegeben eine CTL-Formel φ
  - Für alle Teilformeln φ_i von φ bestimme $[\varphi_i]$
  - Bestimme [φ] aus den $[\varphi_i]$
- Beispiele:
  - $[\varphi \wedge \psi] = [\varphi] \cap [\psi]$
  - $[\neg\varphi] = S-[\varphi]$
  - $[\, EX\, \varphi\,] = R^{-1}[\varphi]$
  - .. aber …
  - $[\, E[\varphi U \psi]\,] = ?$
  - $[\, EG\, \psi\,] = ?$
  - $[\, AX\, \varphi\,] = ?$
  - $[\, AF\, \varphi\,] = ?$
  - …

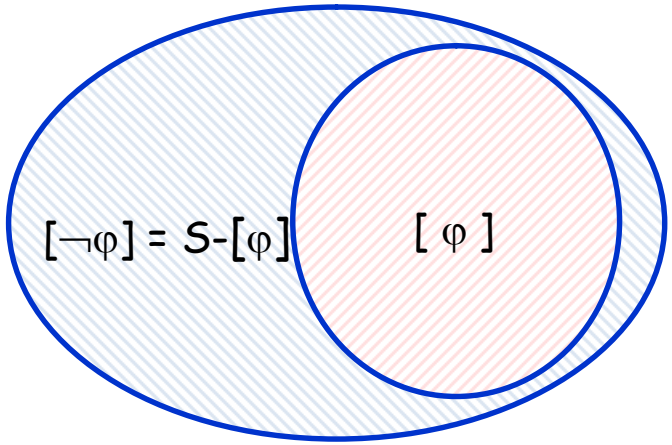


© H. Peter Gumm, Philipps-Universität Marburg

Reduktion

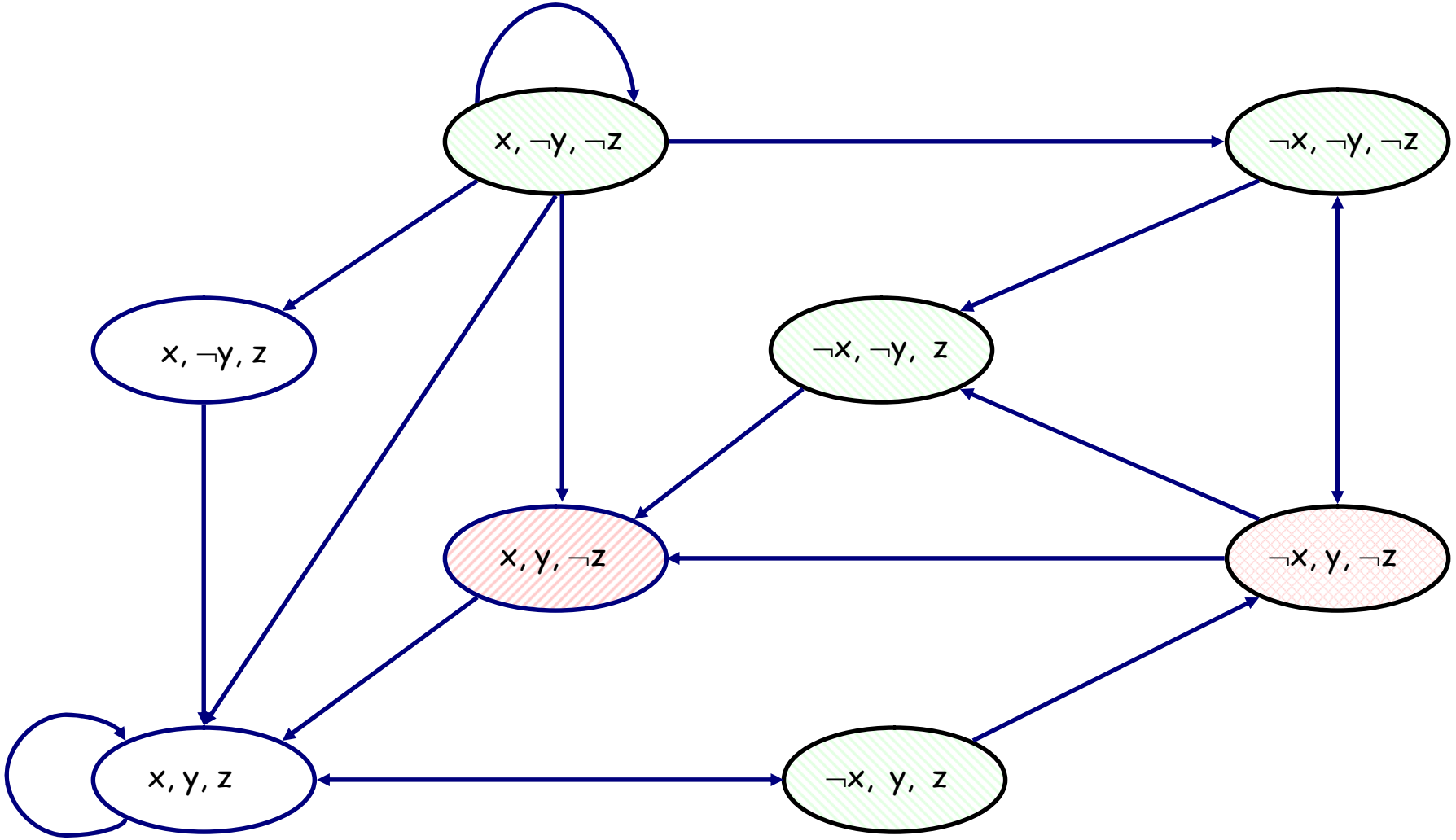
- Nur drei CTL-Operatoren nötig :
  - tt, ¬, ∧, EX, EG, EU.
  - Rückführung der anderen Operatoren:
    - EF ψ = E[tt Uψ]
    - AX ψ = ¬ EX ¬ψ
    - AF ψ = ¬ EG ¬ψ
    - AG ψ = ¬E[tt U ¬ψ]
    - A[φUψ] = ¬EG¬ψ ∧ ¬E[¬ψ U (¬φ ∧ ¬ψ)]
- Es genügt darzulegen:
  - Gegeben [φ], [ψ] ...
  - ... wie berechnet man daraus
    - [tt], [¬φ], [φ∧ψ] ... klar.
    - [EXφ], [EGφ], E[φUψ] ... ?





© H. Peter Gumm, Philipps-Universität Marburg

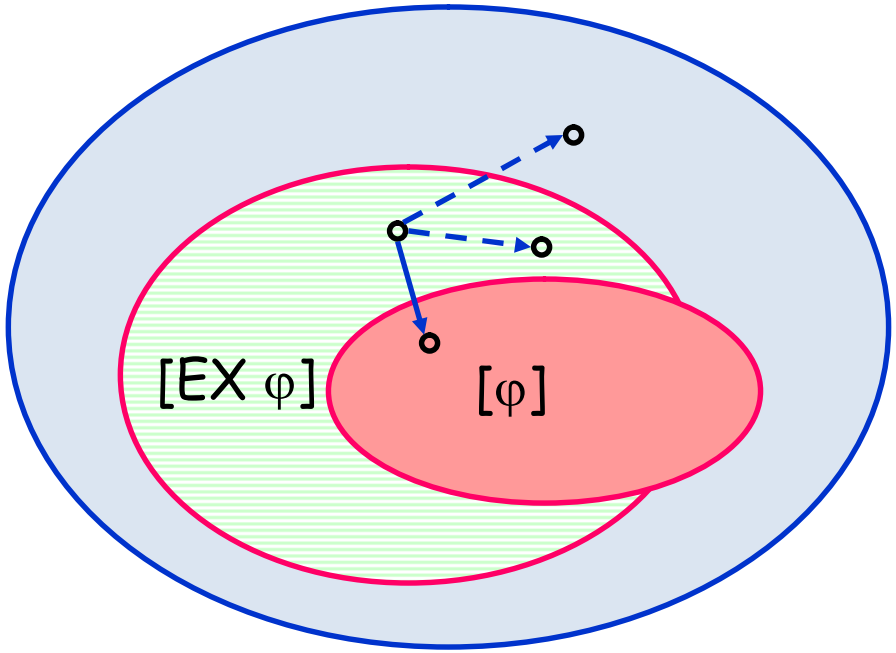
EX (y∧¬z)



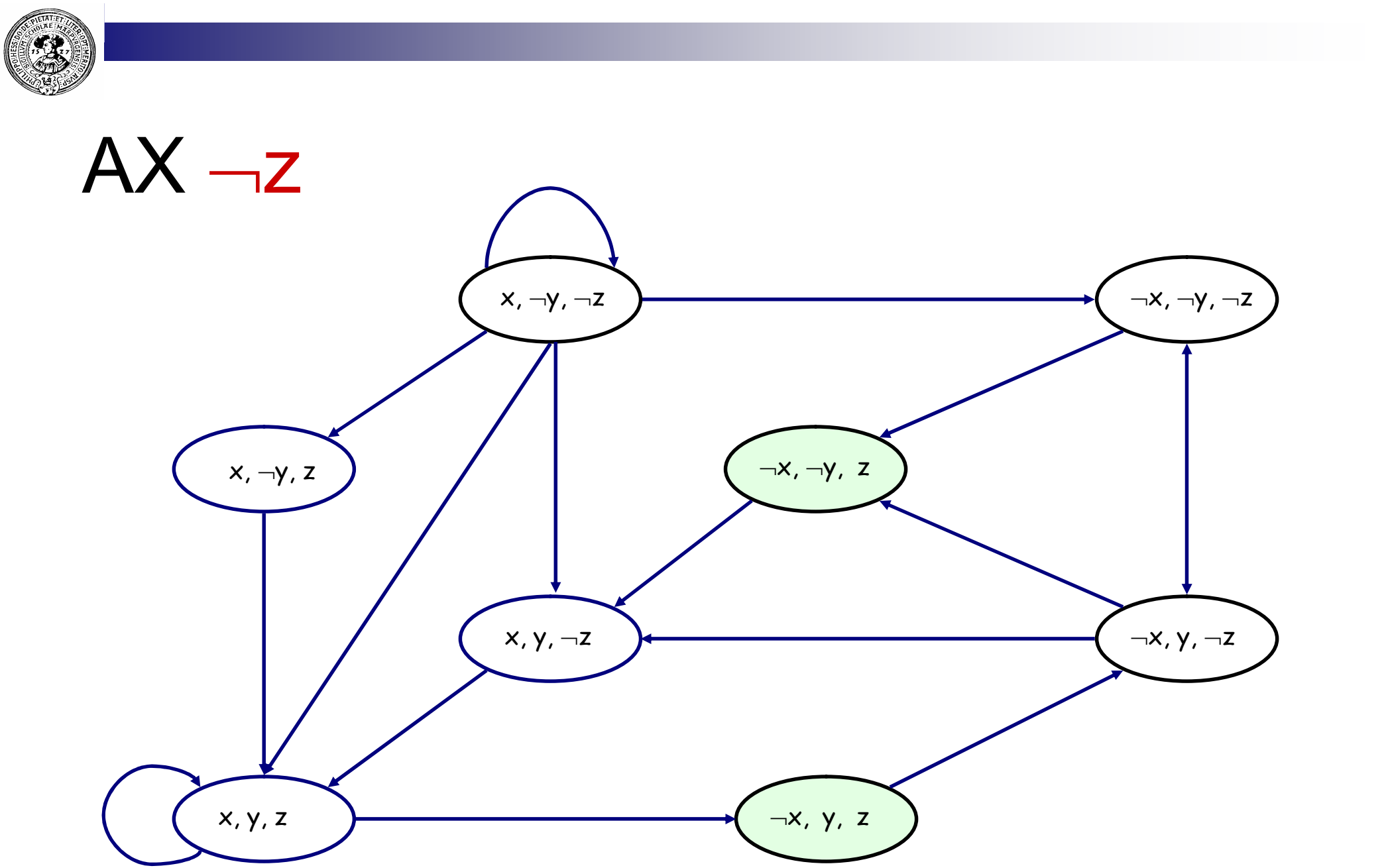
© H. Peter Gumm, Philipps-Universität Marburg

EX φ

- $[\ EX\ \varphi\ ] = R^{-1}[\varphi] = \{\ s \in S \mid \exists s' \in [\varphi].\ s\ R\ s'\ \}$



© H. Peter Gumm, Philipps-Universität Marburg



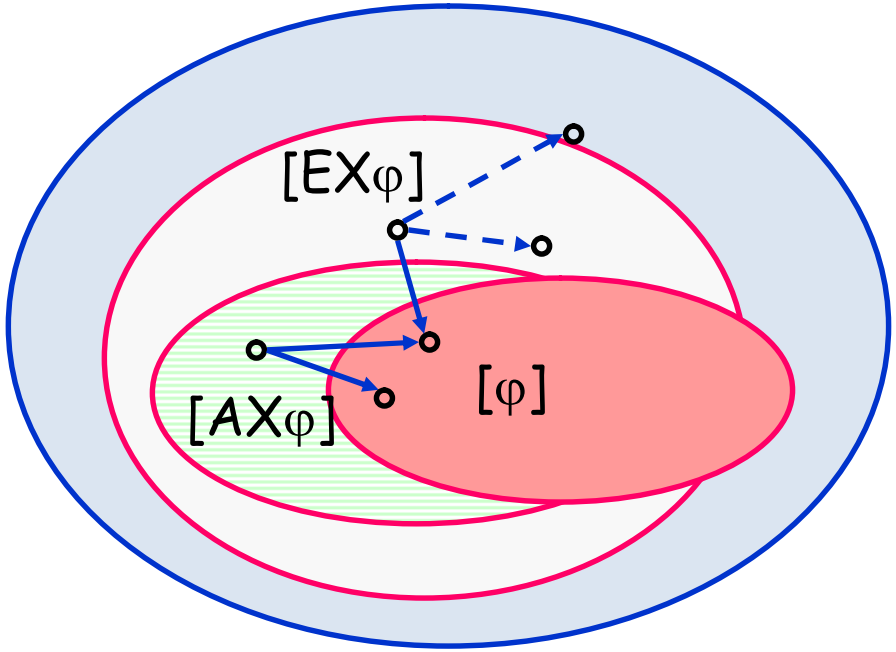
© H. Peter Gumm, Philipps-Universität Marburg

AX φ

- $[\ AX\ \varphi\ ] = \{\ s \in S \mid \forall s' \in S.\ s\ R\ s' \Rightarrow s' \in [\varphi]\ \}$

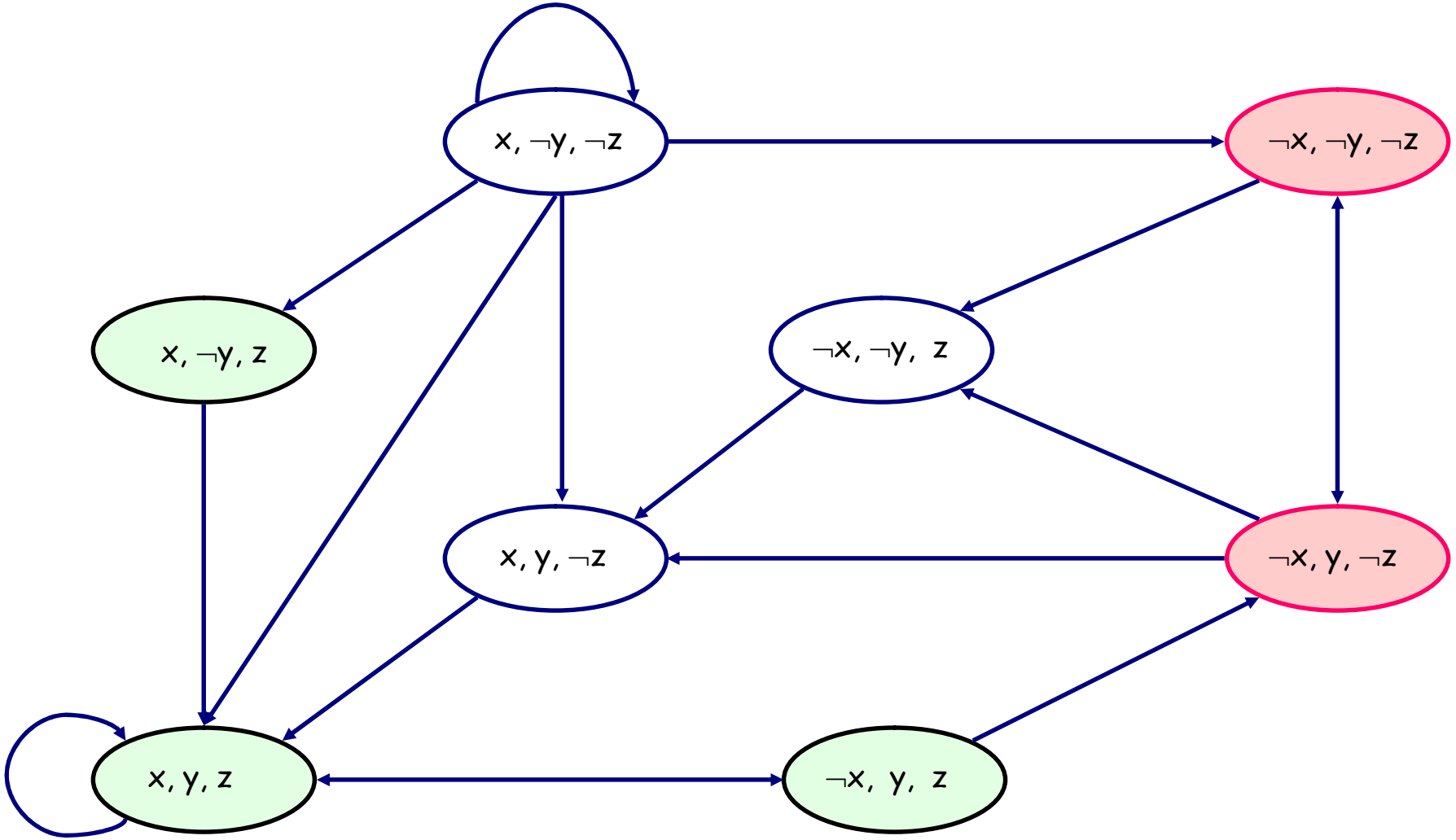
  $= \{\ s \in S \mid R[s] \subseteq [\varphi]\ \}$

  $= wp(\ R, [\varphi]\ \}$ -- „weakest precondition"



© H. Peter Gumm, Philipps-Universität Marburg

E[z U ¬x∧¬z]



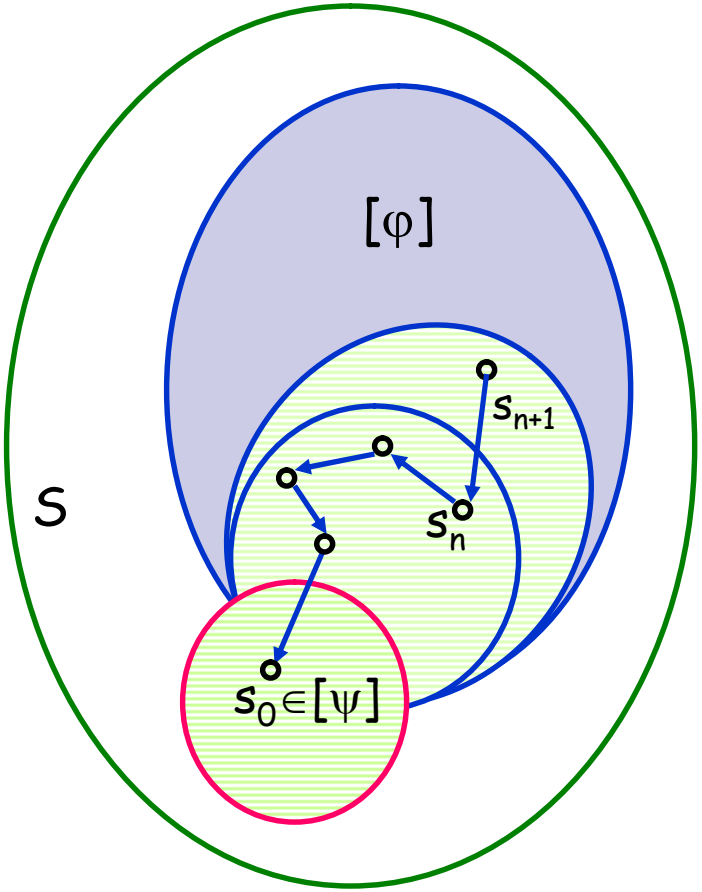
© H. Peter Gumm, Philipps-Universität Marburg

$E[\varphi\ U\ \psi]$

- $s \in [\ E[\varphi U \psi]\ ] \Leftrightarrow$

  $\exists n \geq 0.\ \exists s_n,\ldots,s_0.$
  $s_0 \in [\psi],\ s_i \in [\varphi]$ für $i > 0$, $s_{i+1} R\ s_i$.

- Konstruktionsmethode
  - $S_0 = [\psi]$
    - Füge jeden Vorgänger aus $[\varphi]$ hinzu:
  - $S_{n+1} = S_n \cup ([\varphi] \cap R^{-1}[S_n])$
  - $[\ E[\varphi\ U\ \psi]\ ] = \cup\{\ S_n \mid n \in \mathbb{N}\ \}$
  - Wenn wir S als endlich voraussetzen, stoppt dieser Prozess



© H. Peter Gumm, Philipps-Universität Marburg

Prozedurales Programm

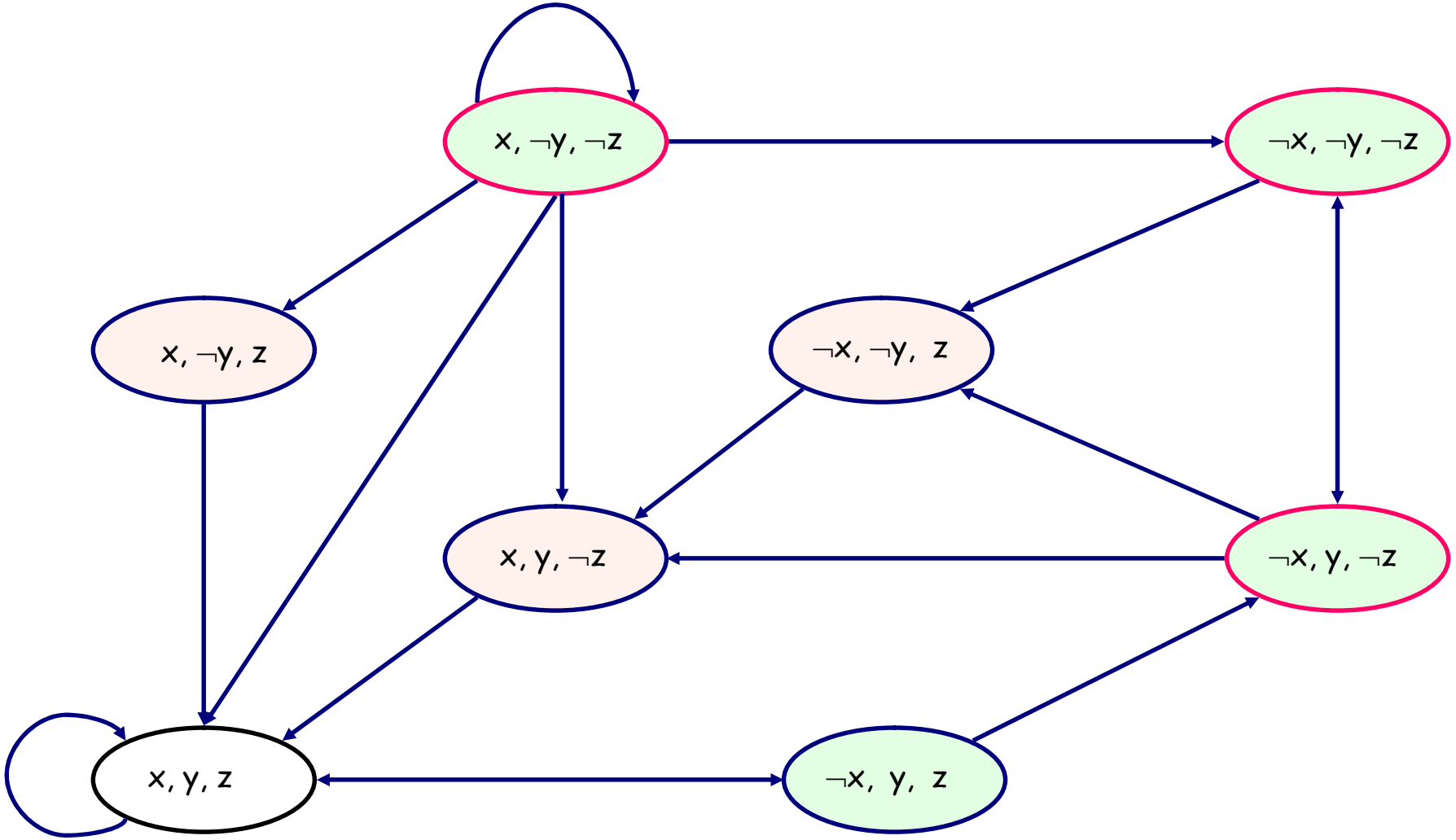
```
void CheckEU(α, β) {
        T := { s | β ∈ label(s) } ;
        for all s ∈ T
                add „E[α U β]" to label(s);
        while ( T≠ ∅ ){
                choose s ∈ T;
                T := T- {s};
                for all t with t R s und „E[α U β]" ∉ label(s) {
                        if ( α ∈ label(t) ) {
                                add „E[α U β]" to label(t);
                                T := T∪{t}
                        }
                }
        }
}
```

Wir können annehmen, daß für alle Zustände **s** gilt:
$\alpha \in$ **label(s)** gdw. **s** $\models_L \alpha$ und
$\beta \in$ **label(s)** gdw. **s** $\models_L \beta$.

Invariante:
s $\in T \Rightarrow$ **s** $\models E[\alpha \, U \, \beta]$

© H. Peter Gumm, Philipps-Universität Marburg

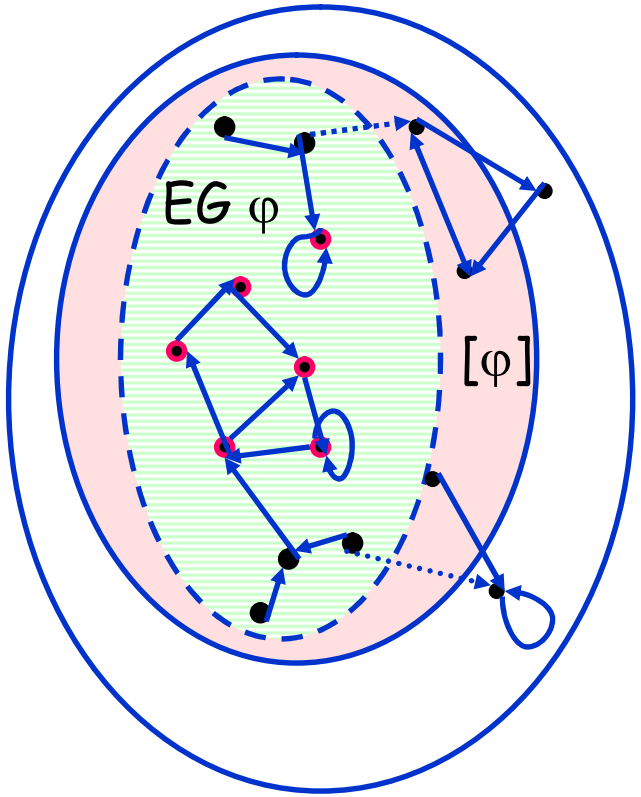
EG (¬x∨¬y∨¬z)



© H. Peter Gumm, Philipps-Universität Marburg

EG φ

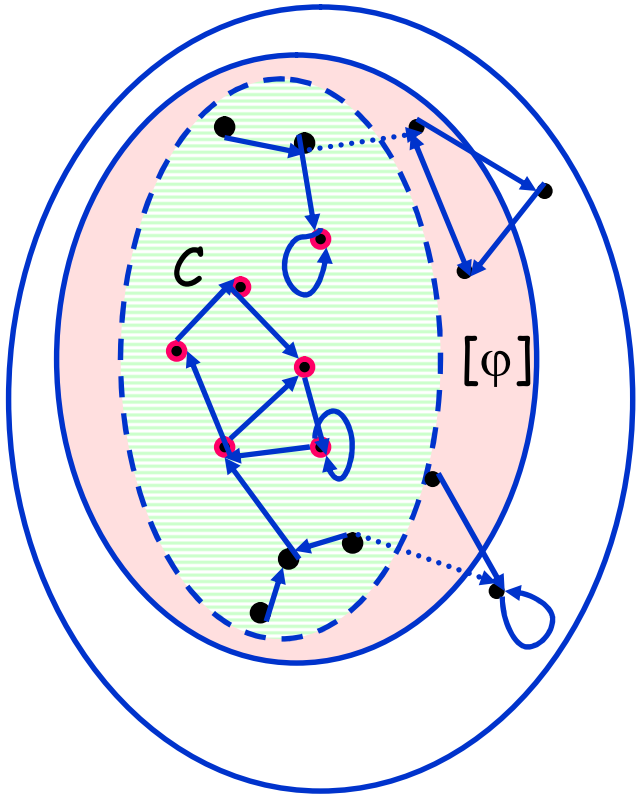
- Kripke-Struktur endlich
  - Jede Berechnung σ hat eine Schleife
  - Aus $s \models EG\ \varphi$ folgt:
    - Es gibt eine Berechnung der Art
      $s=s_0, \ldots, s_{k-1}, s_k, s_{k+1}, \ldots, s_{k+r}, s_k, s_{k+1}, \ldots$ mit $s_i \in [\varphi]$.
  - Strategie
    - Nur Zustände in $[\varphi]$ betrachten
    - Finde Zustände $c \in [\varphi]$ für die es Schleife
      $c \rightarrow s_1 \rightarrow \ldots s_n \rightarrow c$ mit $s_i \in [\varphi]$ gibt
    - Finde alle Zustände s, die (in $[\varphi]$)
      zu einem solchen Zustand führen



© H. Peter Gumm, Philipps-Universität Marburg

Berechnung von EG φ

- Approximation von oben:
  - $C_0 := [\varphi]$
  - Entferne s aus C_n, falls kein Nachfolger in C_n
    $C_{n+1} := C_n - \{s \in C_n \mid R[s] \cap C_n = \emptyset\}$
  - Übrig bleibt:
    $EG\varphi = \cap \{ C_n \mid n \in N \}$
  - Jedes $s \in EG\ \varphi$ hat Nachfolger in $EG\varphi$



© H. Peter Gumm, Philipps-Universität Marburg

Der komplette Algorithmus - funktional

- Zur Berechnung von [p] dient die folgende Funktion eval mit Hilfsfunktionen tauEX, tauEG und tauEU :

```
function eval(p)
case
        p ∈ AP        : { s | p ∈ L(s) }
        p = ¬q        : S-eval(q)
        p = q ∨ r     : eval(q) ∪ eval(r)
        p = EXq       : tauEX(eval(q))
        p = E[q U r]  : tauEU(eval(q), eval(r))
        p = EG q      : tauEG(eval(q))
```

```
tauEX(Q) { return R^-1(Q) }
```

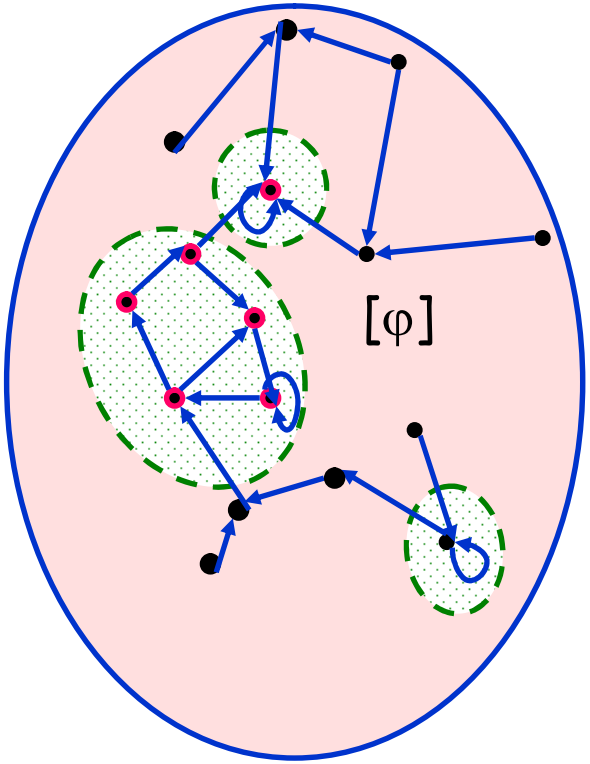
```
function tauEU(Q,Y) {
    Y' :=Y ∪(Q ∩ tauEX(Y))
    if (Y'=Y)  return(Y)
        else return tauEU(Q,Y')
```

```
tauEG(Y) {
    Y' := Y ∩ tauEX(Y)
    if (Y'=Y) return(Y)
    else return tauEG(Y')
}
```


© H. Peter Gumm, Philipps-Universität Marburg

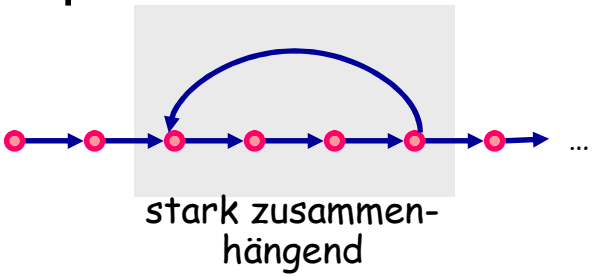
Zusammenhangskomponenten

- s, s' *stark zusammenhängend* (strongly connected), falls es einen Pfad von s zu s' gibt, und umgekehrt.
  - s, s' stark zusammenhängend, falls $(s,s') \in R^+ \cap (R^{-1})^+$
  - $R^+ \cap (R^{-1})^+$ ist symmetrisch und transitiv
    - lässt sich mit Warshall in $O(n^3)$ berechnen
- *Starke Zusammenhangskomponente* (SCC) ist maximale Teilmenge $C \subseteq S$ so dass je zwei $s,s' \in C$ stark zusammenhängend sind.
  - $SC := \Delta \cup (R^+ \cap (R^{-1})^+)$ ist Äquivalenzrelation
    - Jede starke Zusammenhangskomponente ist Äquivalenzklasse von SC
    - Jede Äquivalenzklasse von SC mit mehr als einem Element ist SCC
    - einelementige Äquivalenzklasse $\{ s \}$ von SC ist scc gdw. $s \rightarrow s$.
- S lässt sich disjunkt zerlegen in $S = C_1 \cup C_2 \cup \ldots \cup C_n \cup Rest$
  - $C_1,\ldots,C_n$ sind SCCs,
    - $C_1 \cup C_2 \cup \ldots \cup C_n = \{ s \in S \mid s\, R^+\, s \}$
    - $Rest = \{ s \in S \mid (s,s) \notin R^+ \}$

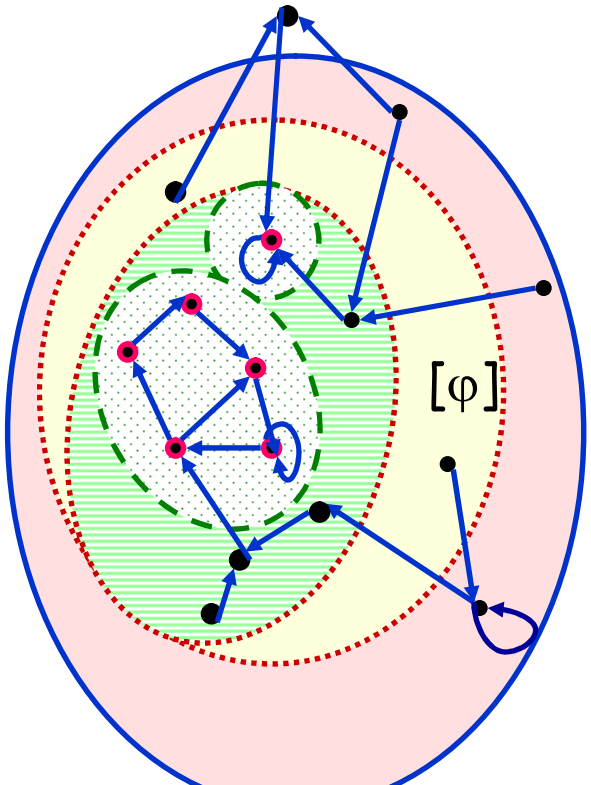


© H. Peter Gumm, Philipps-Universität Marburg

Linearer Algorithmus für EG φ



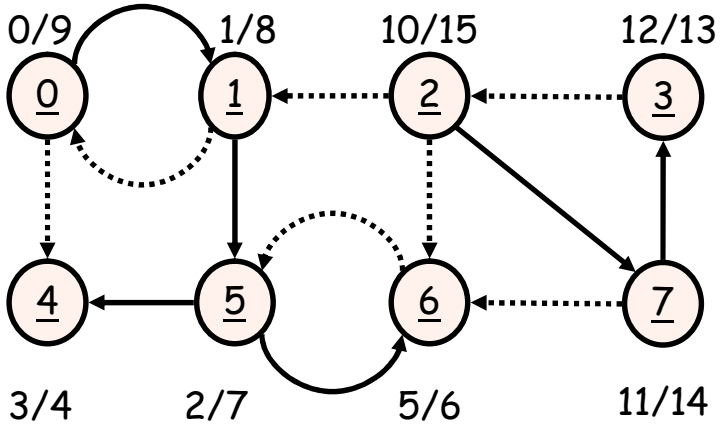
- Wird ein Zustand einer Kripke-Struktur zweimal von dem gleichen Pfad erreicht, so muss er zu einer Schleife, also auch zu einer SCC gehören.
- In einer endlichen Kripke-Struktur muss jeder Pfad ab einem bestimmten Zeitpunkt komplett in einer SCC verlaufen.
  - Beweis: Es kann nur endlich viele Zustände geben, die von der Folge nur einmal besucht werden. Nachdem der letzte besucht wurde, befindet sich die Folge immer in der Vereinigung der SCCs.
- Für die Betrachtung von EGφ interessieren nur Pfade in der Teilstruktur, die aus allen Zuständen aus [φ] und der darauf eingeschränkten Relation → besteht. SCC[φ] sei die Vereinigung aller SCCs in [φ].
- Ein Zustand s erfüllt EGφ ⇔ in der auf [φ] eingeschränkten Kripke-Struktur gibt es einen endlichen Pfad $s=s_0,\ldots,s_n \in SCC[\varphi]$
- Um [EG φ] zu berechnen, berechne in dem Teilgraphen [φ]
  - die (Vereinigung der) SCCs $C_1,\ldots,C_n \subseteq [\varphi]$
  - $s \in EG\ \varphi$ ⇔ Es gibt in [φ] einen Pfad von s zu einem C_i.
- Es gibt einen Algorithmus von Tarjan, der die *SCCs* in linearer Zeit, O(n+m), berechnet. Dieser besteht i.W. aus zwei Tiefensuchen.
  - n = |S| und m = |R|



© H. Peter Gumm, Philipps-Universität Marburg

Tiefensuche

- Besuche alle Knoten eines Graphen
  - Besuchte Knoten werden markiert
  - Kante heißt Baumkante, wenn sie zu einem bisher noch nicht besuchten Knoten führt
- discovery time d, finishing time f
  - Zeitpunkte wenn Knoten expandiert/verlassen wird
  - finish-time festhalten
- Die Baumkanten bilden einen Wald
  - f(vater) > f(sohn)
  - Intervalle [d(k),f(k)] sind entweder geschachtelt oder disjunkt



dt/ft : discovery time / finishing time

Baumkanten durchgezogen

Knoten absteigend geordnet nach ft:

2, 7, 3, 0, 1, 5, 6, 4

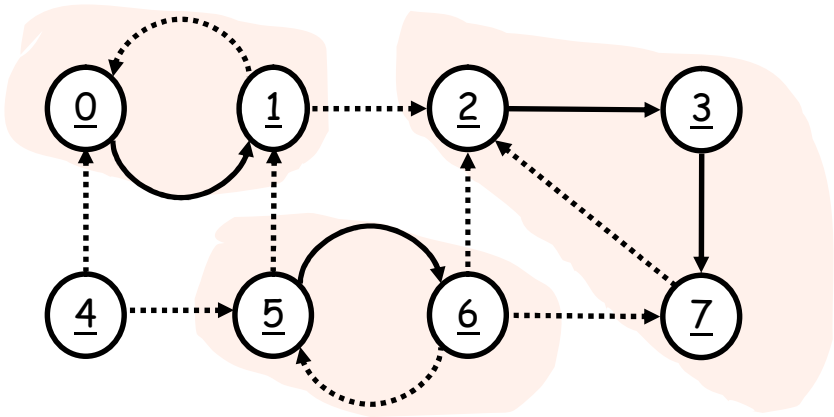
© H. Peter Gumm, Philipps-Universität Marburg

Tarjans Algorithmus

- SCC in einem beliebigen Graphen G=(V,E)
  - Durchlaufe den Graphen depth-first
    - Merke die finish-time jedes Knoten
  - Bilde den reversen Graphen G'=(V,E')
    - Transposition der Adjazenzmatrix
  - Ordne Knoten absteigend nach finish-time
  - Führe Tiefensuche in G' durch
  - Die Bäume des entstandenen DFS-Waldes sind genau die SCCs
- Für Model-Checking :
  - entferne die ein-elementigen Klassen ohne Selbstloops.

DFS im reversen Graphen mit Knotenreihenfolge:

2, 7, 3, 0, 1, 5, 6, 4

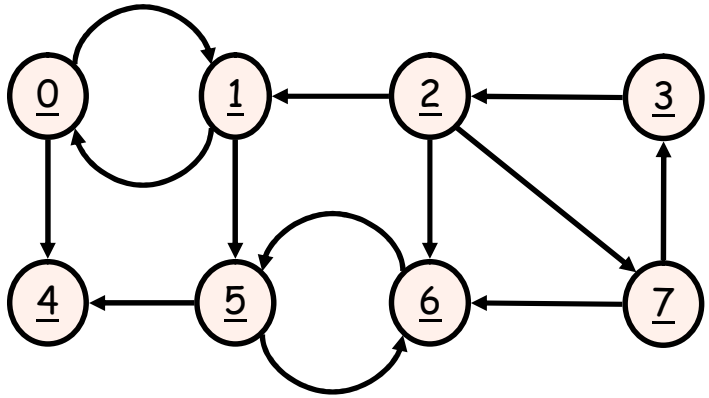


DFS-Wald = SCC+isoierte Punkte

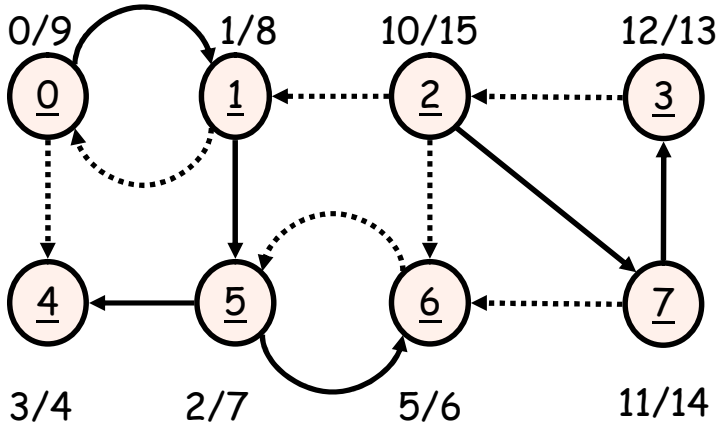
© H. Peter Gumm, Philipps-Universität Marburg

Tarjan – auf einen Blick

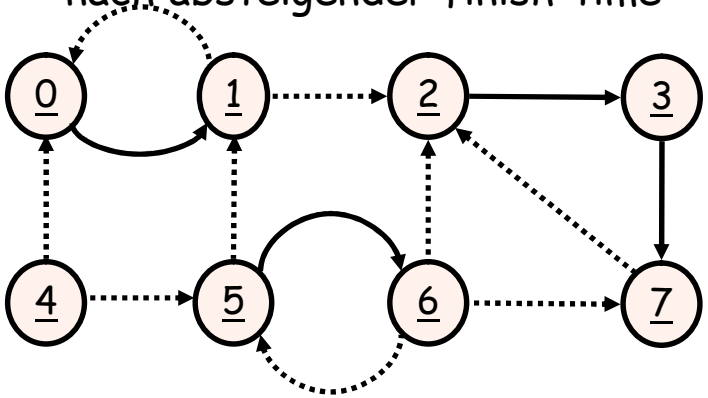
1. Ausgangsgraph



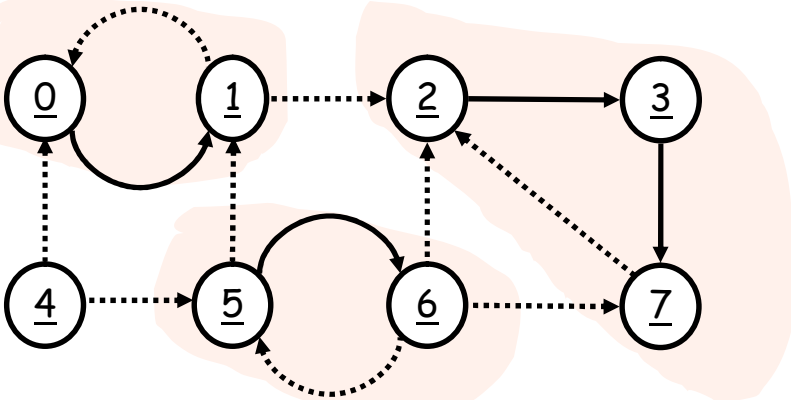
2. Nach erster DFS mit discover time und finish time. Baumkanten durchgezogen



3. DFS im reversen Graphen nach absteigender finish-time



4. DFS-Wald = SCC + isolierte Punkte



© H. Peter Gumm, Philipps-Universität Marburg

Tarjan

- Erste Tiefensuche
  - berechne finishing times
- Ordne Knoten ein nach absteigender finishing time
- Transponiere Graphen
- erneute Tiefensuche

Komplexität für Graphen mit n Knoten und m Kanten:

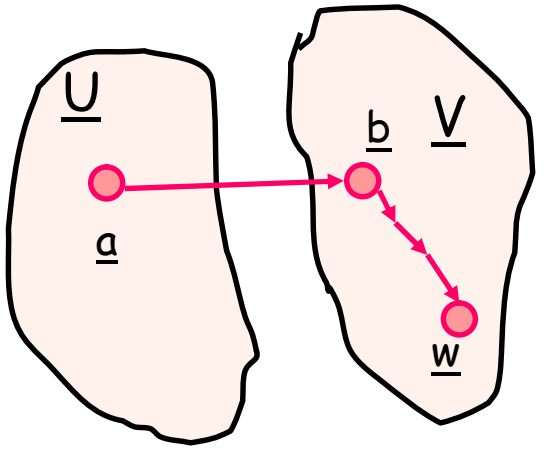
$O(n) + O(n) + O(n+m) + O(n) =$

$O(n+m)$

```
7   int[] nodes  = {0,1,2,3,4,5,6,7};
8   int[][] succ = {{1,4},{0,5},{1,6,7},{2},{},{4,6},{5},
9   // ------------------------------------------------
10  int[] ordNodes = new int[nodes.length];
11  private boolean[] seen = new boolean[nodes.length];
12  private int[]     fin  = new int [nodes.length];
13  private int       time   = 0;
14  // ------------------------------------------------
15  void scc(){
16    dfSearch(succ);
17    for(int k:nodes)ordNodes[nodes.length-1-fin[k]]=k;
18    nodes = ordNodes;
19    dfSearch(transpose(succ));
20  }
21  void dfSearch(int[][] kanten){
22     for(int k : nodes){ seen[k] = false; }
23     for(int k : nodes)
24       if(!seen[k])dfVisit(k,kanten);
25  }
26  void dfVisit(int k, int[][] kanten){
27    seen[k] = true;
28    System.out.print(" "+k);
29    for(int next: kanten[k])
30      if(!seen[next]) dfVisit(next,kanten);
31    fin[k] = time++;
32  }
```


© H. Peter Gumm, Philipps-Universität Marburg

Korrektheit



- Für beliebige Teilmengen $U \subseteq S$ definiere

  $f(U) = \max\{ f(u) \mid u \in U \}$.

- Lemma: Sind U und V zwei SCCs und $a \rightarrow b$ eine Kante mit $a \in U$ und $b \in V$. Dann gilt $f(V) < f(U)$.

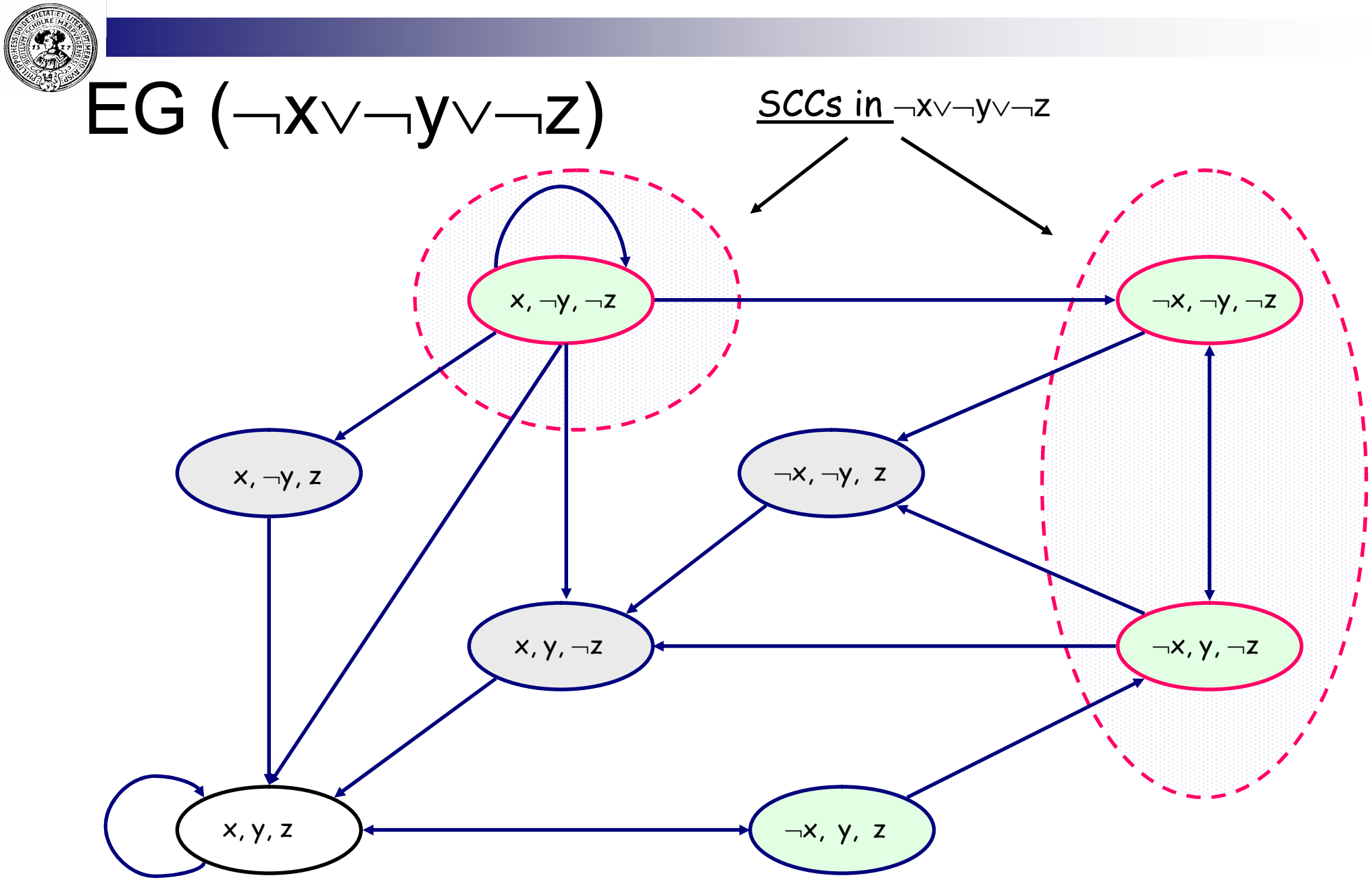
- Sind x und y in der gleichen SCC, dann sind x und y auch im gleichen Baum jeder DFS-Suche
  - Klar, es gibt einen Weg von x nach y und einen von y nach x.

- Sind a und b im gleichen Baum der zweiten DFS, dann auch in der gleichen SCC

  - Angenommen, das wäre nicht so. Dann müssten zwei verschiedene SCCs U und V existieren, und eine Baumkante der zweiten DFS zwischen U und V. Wir können V so wählen, dass w, die Wurzel des DFS Baumes, darin enthalten ist.

  - Nach dem Lemma gilt $f(V) < f(U)$, insbesondere also $f(w) < f(V) < f(U)$. Somit gibt es ein Element u in U mit $f(w) < f(u)$. Das ist aber ein Widerspruch zur Wahl von w.

© H. Peter Gumm, Philipps-Universität Marburg



© H. Peter Gumm, Philipps-Universität Marburg

Fixpunkte

- Bekannte Gleichungen für CTL-Operatoren
  - $EF\varphi = \varphi \vee EX\ EF\varphi$
  - $EG\ \varphi = \varphi \wedge EX\ EG\ \varphi$
  - $E[\varphi U \psi] = \psi \vee (\varphi \wedge EX\ E[\varphi U \psi])$
  - $AF\varphi = \varphi \vee AX\ AF\varphi$
  - $AG\ \varphi = \varphi \wedge AX\ AG\varphi$
  - $A[\varphi U \psi] = \psi \vee (\varphi \wedge AX\ A[\varphi U \psi])$
- Lassen sie sich durch diese Gleichungen definieren ?
  - Erinnert an rekursive Definitionen
  - Was sind die Basisfälle ?
- Die CTL-Operatoren erfüllen also Fixpunkt Gleichungen
  - $z = \varphi \vee EX\ z$
  - $z = \varphi \wedge EX\ z$
  - $z = \psi \vee (\varphi \wedge EX\ z)$
  - $z = \varphi \vee AX\ z$
  - $z = \varphi \wedge AX\ z$
  - $z = \psi \vee (\varphi \wedge AX\ z)$
- Es gibt weitere Lösungen:
  - true erfüllt auch erste und vierte Gleichung:
    - $\varphi \vee EX\ true = \varphi \vee true = true.$
  - false erfüllt Gleichung 2 und 5

© H. Peter Gumm, Philipps-Universität Marburg

Funktionale

Sei S eine Menge und $\tau : \wp(S) \to \wp(S)$ eine Abbildung. τ heißt auch ***Funktional***, weil man es als Abbildung des *Funktionenraumes* $[S \to 2]$ in $[S \to 2]$ auffassen kann.

τ heißt ***monoton***, falls für alle $U,V \subseteq S$ gilt : $U \subseteq V \Rightarrow \tau(U) \subseteq \tau(V)$.

$W \subseteq S$ heißt ***Fixpunkt*** von τ, falls $\tau(W) = W$.

Satz (Tarski): Jedes monotone Funktional τ hat einen Fixpunkt.

Beweis: Sei $F = \cup\{ V \subseteq S \mid \tau(V) \supseteq V \}$. Für alle V mit $\tau(V) \supseteq V$ gilt also $F \supseteq V$. Wegen Monotonie gilt für alle V mit $\tau(V) \supseteq V$ auch $\tau(F) \supseteq \tau(V)$. Folglich gilt

$$\tau(F) \supseteq \cup\{ \tau(V) \mid \tau(V) \supseteq V\} \supseteq \cup\{ V \mid \tau(V) \supseteq V\} = F$$

Somit $\tau(F) \supseteq F$ und wegen Monotonie wieder $\tau(\tau(F)) \supseteq \tau(F)$, woraus $\tau(F) \subseteq F$ folgt. Also $\tau(F)=F$.

Ist S endlich, so hat jedes monotone Funktional einen größten Fixpunkt $\cap \tau^n(S)$ und einen kleinsten Fixpunkt $\cup \tau^n(\emptyset)$.

Beweis: Betrachte die Folge $\emptyset \subseteq \tau(\emptyset) \subseteq \tau(\tau(\emptyset)) \subseteq \ldots \subseteq \tau^n(\emptyset) \subseteq \ldots$ Da S endlich ist, können diese Mengen nicht jedesmal echt größer werden. Es gibt also ein $n \in \mathbb{N}$ mit $\tau^n(\emptyset) = \tau^{n+1}(\emptyset)$. Es folgt, dass $\tau^n(\emptyset)$ der kleinste Fixpunkt von τ ist. Analog ist $\cap_{n \in \mathbb{N}} \tau^n(S)$ der größte Fixpunkt von τ.

Kleinster Fixpunkt von τ : $\mu y.\tau(y)$ Größter Fixpunkt von τ : $\nu y.\tau(y)$

© H. Peter Gumm, Philipps-Universität Marburg

EF φ als kleinster Fixpunkt

Ausgangspunkt : $\tau(Y) = \varphi \vee \mathbf{EX}\ Y$

transformiert : $\underline{\tau}(Y) = [\varphi] \cup \mathbf{R}^{-1}(Y)$.

EF φ ist <u>kleinster</u> Fixpunkt von $\tau = \lambda y.\ \varphi \vee \mathbf{EX}\ y$.

Daß **EF** φ überhaupt ein Fixpunkt von τ ist, folgt aus den Fixpunktidentitäten von CTL
$\underline{\tau}(\mathbf{EF}\ \varphi) = \varphi \vee \mathbf{EX}\ \mathbf{EF}\ \varphi = \mathbf{EF}\ \varphi$.

Sei Y nun irgendein Fixpunkt von τ. Dann gilt: $[\varphi] \subseteq Y$ und $[\mathrm{EX}\ Y] \subseteq Y$, also $\mathrm{EX}(\mathrm{EX}(\ldots(\mathrm{EX}\ \varphi)\ldots))] \subseteq Y$

Sei $s_0 \in [\mathbf{EF}\ \varphi]$, dann gibt es $s_0 \to \ldots \to s_n$ mit $s_n \models \varphi$. Es folgt $s_0 \in [\mathbf{EX}\ (\mathbf{EX}\ \ldots\ (\mathbf{EX}\ \varphi))] \subseteq y$.

© H. Peter Gumm, Philipps-Universität Marburg

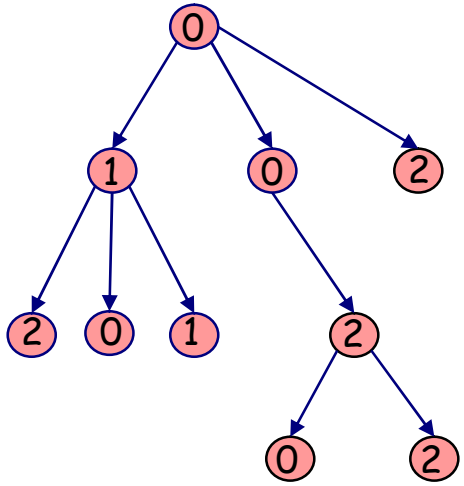
Erinnerung: Alphabete, Worte

- Sei Σ eine Menge
  - genannt Alphabet
  - endlich oder unendlich
- Σ^* = Menge aller Σ-Worte
  - Menge aller endlichen Listen von Elt. aus Σ
  - ε = leeres Wort
  - $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ Menge aller nichtleeren Worte
- Für $e \in \Sigma$ und $u \in \Sigma^*$ sei
  - $e \cdot u$ das Wort mit ersten Zeichen e und Rest u
  - $u \cdot e$ das Wort mit Anfangswort u und letztem Zeichen e.
- $u \preccurlyeq v \;:\Leftrightarrow\;$ u Präfix von v
  - $\varepsilon \preccurlyeq v$
  - $e \cdot u \preccurlyeq v :\Leftrightarrow v = e \cdot w$ und $u \preccurlyeq w$

© H. Peter Gumm, Philipps-Universität Marburg

Bäume, Äste

- Ein Σ-Baum ist eine Teilmenge $B \subseteq \Sigma^+$ mit
  - $\forall u \in \Sigma^+ . \forall e \in \Sigma,\ u \cdot e \in B, \Rightarrow u \in B$
    - Äquivalent: B ist *Präfix-abgeschlossen*
  - $e, e' \in \Sigma \cap B \Rightarrow e = e'$
    - Wurzel: Eindeutiges Wort $w \in B$ der Länge 1.
- Knoten: Jedes $u \in B$
  - Idee: Jedes Wort ist Adresse eines Knotens
  - Sohn eines Knoten $u \in B$: Jedes Wort $u \cdot e \in B$
  - Verzweigungsgrad eines Knoten K: Anzahl der Söhne
- Ast: Bezüglich $\preccurlyeq$ linear geordnete Teilmenge
  - Teilmenge $A \subseteq B$ mit $\forall u, v \in A.\ u \preccurlyeq v$ oder $v \preccurlyeq u$
  - Ist der Baum B endlich, so gibt es in jedem Ast ein maximales Wort a mit $\forall u \in A.\ u \preccurlyeq a$.



B = { 0, 01, 00, 02, 012, 010, 011, 002, 0020, 0022 }

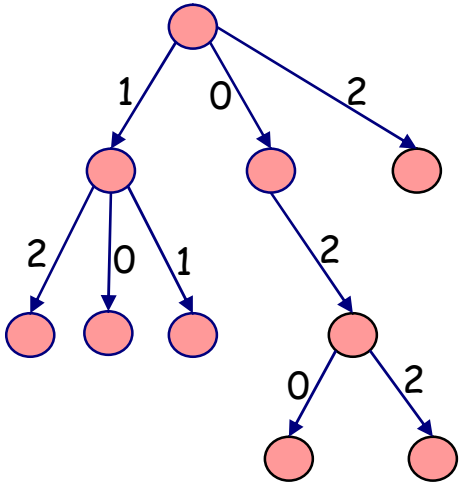
A = { 0, 00, 002 } ist ein Ast

A = { $w \in B \mid w \preccurlyeq 0020$ }

© H. Peter Gumm, Philipps-Universität Marburg

Varianten von Bäumen

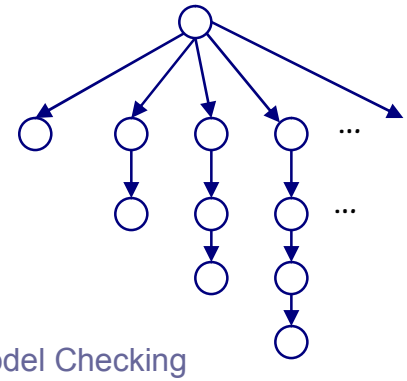
- Bäume mit Kantenlabel:
  - $B \subseteq \Sigma^*$ mit
    - $u \preccurlyeq v \in B \Rightarrow u \in B$
  - Jetzt gilt immer $\varepsilon \in B$
  - ε ist Adresse der Wurzel
- Alles andere wie bisher



B = { ε, 0, 01, 00, 02, 012, 010, 011, 002, 0020, 0022 }

A = { ε, 0, 02, 020 } ist ein Ast

A = { w∈ B | w ≼ 020 }

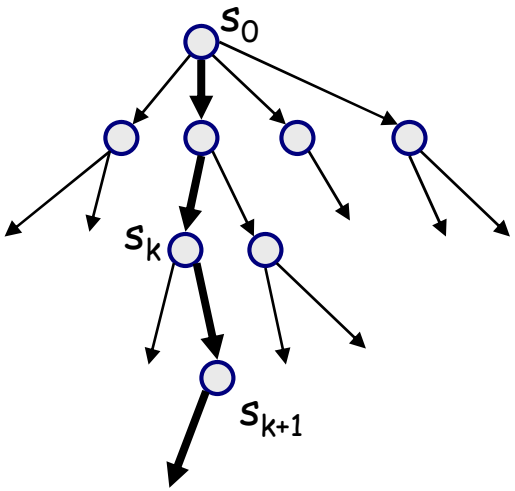


Ein unendlicher Baum mit nur endlichen Ästen aber unendlichem Verzweigungsgrad $\Sigma = \mathbb{N}$

© H. Peter Gumm, Philipps-Universität Marburg

Lemma von König

- Jeder unendliche, endlich verzweigende Baum enthält einen unendlichen Ast.



Beweisskizze (benötigt Auswahlaxiom): Wir „konstruieren" einen solchen Ast. Er beginnt mit der Wurzel s_0. Da der Baum unendlich ist, gibt es unendlich viele endliche Pfade, die mit s_0 beginnen.

Angenommen, wir haben bereits $s_0,\ldots,s_k$ so konstruiert, so dass es unendlich viele Pfade gibt, die mit der Folge $s_0,\ldots,s_k$ beginnen. Da s_k nur endlich viele Söhne hat, muss es einen Sohn geben, durch den unendlich viele dieser Pfade durchlaufen. Dieser Sohn sei s_{k+1}.

Wir haben also gezeigt, dass wir induktiv den Pfad immer noch um einen Sohn verlängern können, erhalten also unseren unendlichen Ast.

© H. Peter Gumm, Philipps-Universität Marburg

AF φ als kleinster Fixpunkt

- Ist K bild-endlich, so ist AF φ der kleinste Fixpunkt des Funktionals $\tau(Y) = \varphi \vee AX(Y)$
- 1. Wegen der Gleichung $AF\varphi = \varphi \vee AX\, AF\varphi$ ist $AF\varphi$ ein Fixpunkt.
- 2. Sei Y irgendein Fixpunkt, also $Y = \varphi \vee AX(Y)$, dann müssen wir zeigen: $AF\, \varphi \subseteq Y$, oder, äquivalent: $\forall\, s \in S.\ s \notin Y \Rightarrow s \notin AF\varphi$.
- Lemma: $\forall n \in \mathbb{N}. \forall s \in S.\ s \notin Y \Rightarrow \exists\, s = s_0 \to \ldots \to s_n. \forall\, i \leq n.\ s_i \not\models \varphi$.
  Beweis: Induktion über n.
  - n=0 klar.
  - $n \to n+1$: $s \notin Y \Rightarrow s \notin \varphi \vee AX(Y)$, also $s \not\models \varphi$ und $s \notin AX(Y)$.
    Es ex. also ein t mit $s \to t$ und $t \notin Y$.
    Mit Ind.Hyp. ex. $t = t_0 \to \ldots \to t_n$ mit $t_i \not\models \varphi$. Dann ist $s \to t_0 \to \ldots \to t_n$ der gesuchte Pfad der Länge n+1, der nirgends φ erfüllt.
- Aus $s \notin Y$ folgt also: Der Baum, der aus allen endlichen, in s startenden Pfaden besteht, die nirgends φ erfüllen, ist unendlich. Wegen der Voraussetzung bild-endlich garantiert König's Lemma einen unendlichen Pfad, somit $s \not\models AF\varphi$.

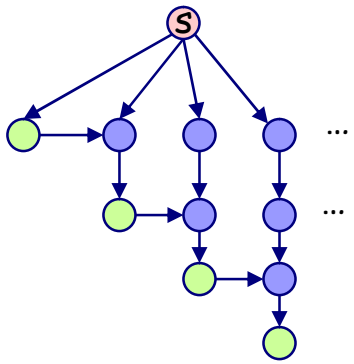
© H. Peter Gumm, Philipps-Universität Marburg

Notwendigkeit

- Bildendlichkeit ist notwendig für

$$AF\varphi = \mu Y.(\varphi \vee AX\ Y)$$

wie das Beispiel zeigt:



Der kleinste Fixpunkt von $\tau(Y)$ = grün $\vee$ AX(Y) besteht aus allen Punkten, aber $s \notin \cup \tau^n(\emptyset)$.

© H. Peter Gumm, Philipps-Universität Marburg

$E[\phi\ U\ \psi] = \mu Y.(\psi \vee (\phi \wedge EX\ Y))$

- Wegen der Gleichung
  $E[\phi\ U\ \psi] = \psi \vee (\phi \wedge EX\ E[\phi U \psi])$
  ist $E[\phi\ U\ \psi]$ ein Fixpunkt des Funktionals.

- Sei jetzt Y irgendein Fixpunkt, also
  $Y = \psi \vee (\phi \wedge EX\ Y)$
  dann müssen wir zeigen: $E[\phi\ U\ \psi] \subseteq Y$.

  Sei $s \in E[\phi U\ \psi]$, dann gibt es $s_1,\ldots,s_n$ mit $s \to s_1 \to \ldots \to s_n \models \psi$ und $s_i \models \phi$ für $i<n$. Es folgt $s_n \in [q] \subseteq Y$.

  - Aus $x \to y \in Y$ und $x \in [p]$ folgt $x \in Y$.
  - Daher gilt der Reihe nach: $s_n \in Y$, $s_{n-1} \in Y$ … $s \in Y$.

© H. Peter Gumm, Philipps-Universität Marburg

$EG\ \phi = \nu Y.(\phi \wedge EX\ Y)$

- Wegen $EG\phi = \phi \wedge EX\ EG\ \phi$ ist $EG\phi$ ein Fixpunkt.
- Sei Y ein beliebiger Fixpunkt, also $Y = \phi \wedge EX\ Y$. Wir müssen zeigen: $Y \subseteq EG\ \phi$.

  Ist $s \in Y$, dann gilt $s \models \phi$ und $\exists\ s_1.s \rightarrow s_1 \in Y$.
  Beginnend mit s können wir also einen unendlichen Pfad konstruieren, auf dem stets ϕ gilt.

  Folglich $s \models EG\ \phi$.

- Analog: $AG\ \phi = \nu\ Y.(\phi \wedge AX\ Y)$:

© H. Peter Gumm, Philipps-Universität Marburg

CTL-Operatoren als Fixpunkte

So wie für **EF** und **AF** können auch für die anderen CTL-Operatoren Fixpunktgleichungen hergeleitet werden. Es gilt :

$$\mathbf{EF}\ p = \mu y.\ (p \vee \mathbf{EX}\ y)$$

$$\mathbf{EG}\ p = \nu y.\ (p \wedge \mathbf{EX}\ y)$$

$$\mathbf{E}[p\ \mathbf{U}\ q] = \mu y.\ (q \vee (p \wedge \mathbf{EX}\ y))$$

Falls R **bild-endlich** ist, gilt :

$$\mathbf{EF}\ p = \cup_i\ (\lambda y.(p \vee \mathbf{EX}\ y)^i)(\varnothing)$$

$$\mathbf{EG}\ p = \cap_i\ (\lambda y.(p \wedge \mathbf{EX}\ y)^i)(S)$$

$$\mathbf{E}[p\ \mathbf{U}\ q] = \cup_i\ (\lambda y.(q \vee (p \wedge \mathbf{EX}\ y))^i)(\varnothing)$$

© H. Peter Gumm, Philipps-Universität Marburg

Die Berechnung der Fixpunkte

Kleinste (bzw. größte) Fixpunkte eines Funktionals τ sind einfach algorithmisch zu berechnen :

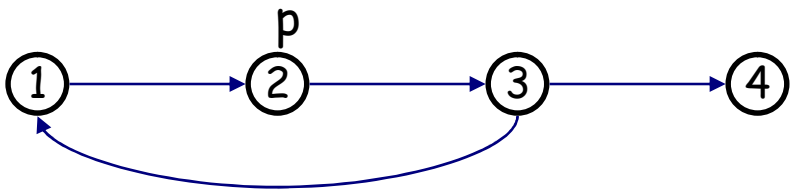
```
Y := false ;  {bzw.Y:=true}
do{
    Y' := Y ;
    Y := τ(Y)
}while (Y≠Y);
return Y;
```

Dieser Algorithmus terminiert, gdw. ein n existiert mit $\tau(\tau^n(\varnothing)) = \tau^n(\varnothing)$. (bzw. $\tau(\tau^n(S)) = \tau^n(S)$.)

© H. Peter Gumm, Philipps-Universität Marburg

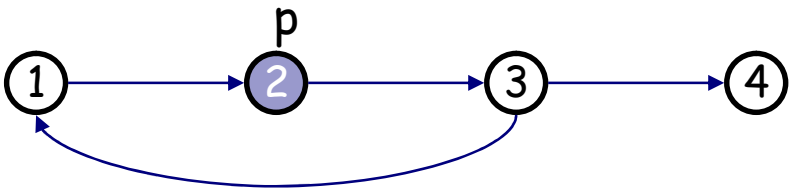
Beispiel: Berechnung von EF p

Es gilt: **EF** p = $\mu y.(p \vee \mathbf{EX}\ y)$, also
$\tau = \lambda y.(p \vee \mathbf{EX}\ y)$
$\approx \lambda y.(p \cup \mathbf{R}^{-1} y)$.

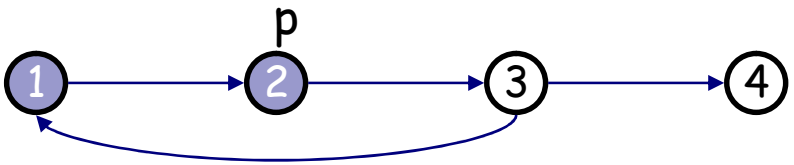


---

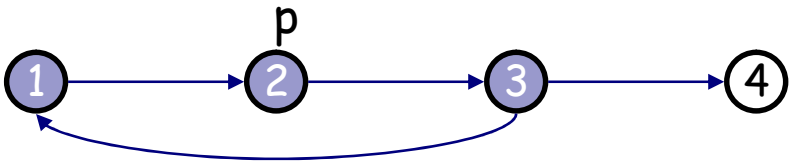
1. $\tau^1(\emptyset) = p \cup \mathbf{R}^{-1}(\emptyset) = p = \{ 2 \}$



2. $\tau^2(\emptyset) = p \cup \mathbf{R}^{-1}\{ 2 \} = \{ 1,2 \}$



3. $\tau^3(\emptyset) = p \cup \mathbf{R}^{-1}\{ 1,2 \} = \{ 1,2,3 \}$



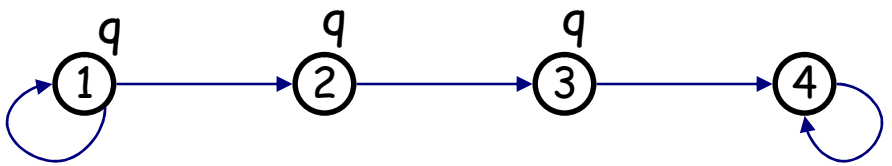
4. $\tau^4(\emptyset) = p \cup \mathbf{R}^{-1}\{ 1,2,3 \} = \tau^3(\emptyset)$

Fixpunkt erreicht !

© H. Peter Gumm, Philipps-Universität Marburg

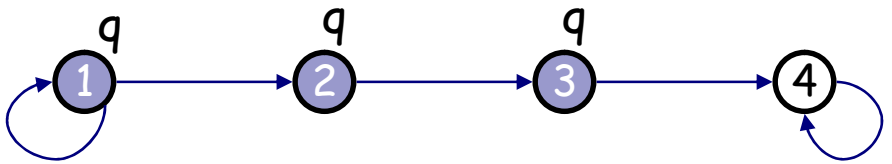
Berechnung von EG q

Es gilt: **EG** q = ν y. (q ∧ **EX** y), also
τ = λ y.(q ∧ **EX** y)
≈ λ y.(q ∩ **R**$^{-1}$ y).

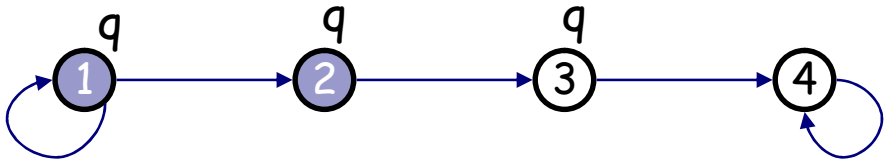


---

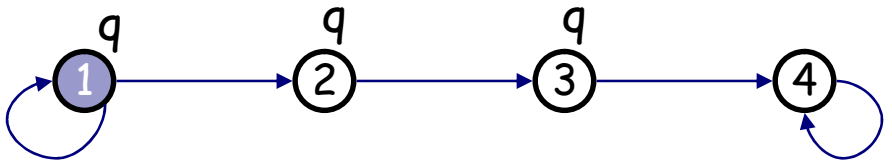
1. $\tau^1(S) = q \cap R^{-1}\{1,2,3,4\} = \{1,2,3\}$



2. $\tau^2(S) = q \cap R^{-1}\{1,2,3\} = \{1,2\}$



3. $\tau^3(S) = q \cap R^{-1}\{1,2\} = \{1\}$



4. $\tau^4(S) = q \cap R^{-1}\{1\} = \tau^3(S)$

Fixpunkt erreicht !

© H. Peter Gumm, Philipps-Universität Marburg

Die restlichen CTL-Operatoren

Die folgenden CTL-Operatoren haben wir ausführlich behandelt :

$$\mathbf{AF}\ p = \mu y.\ (p \vee \mathbf{AX}\ y)$$

$$\mathbf{EF}\ p = \mu y.\ (p \vee \mathbf{EX}\ y)$$

$$\mathbf{EG}\ p = \nu y.\ (p \wedge \mathbf{EX}\ y)$$

$$\mathbf{E}[p\ \mathbf{U}\ q] = \mu y.\ (q \vee (p \wedge \mathbf{EX}\ y))$$

Die fehlenden lassen sich ebenfalls als Fixpunkte definieren :

$$\mathbf{AG}\ p = \nu y.\ (p \wedge \mathbf{AX}\ y)$$

$$\mathbf{A}[p\ \mathbf{U}\ q] = \mu y.\ (q \vee (p \wedge \mathbf{AX}\ y))$$

© H. Peter Gumm, Philipps-Universität Marburg

Rückführung auf drei Operatoren

Allein mit Hilfe von **EX**, **E[U]** und **EG** lassen sich alle CTL-Operatoren darstellen :

$$\mathbf{AX}\ p = \neg\mathbf{EX}\ \neg p$$

$$\mathbf{AF}\ p = \neg\mathbf{EG}\ \neg p$$

$$\mathbf{EF}\ p = \mathbf{E}[\ tt\ \mathbf{U} p\ ]$$

$$\mathbf{AG}\ p = \neg\mathbf{EF}\ \neg p$$

$$\mathbf{A}[\ p\ \mathbf{U}\ q\ ] = \neg(\mathbf{E}[\neg q\ \mathbf{U}\ \neg p \wedge \neg q\ ] \vee \mathbf{EG}\ \neg q)$$

© H. Peter Gumm, Philipps-Universität Marburg

Fairness

Meist interessiert man sich nur für Berechnungen, die eine gewisse Fairness-Bedingung respektieren. In SMV kann man CTL-Eigenschaften angeben, die nicht dauernd verletzt sein sollen.

```
-- der Kanal ist nicht völlig kaputt
FAIRNESS  working
```

$\mathbf{A}\ \mathrm{GF}\ \mathrm{working}$

```
-- p kann sich immer wieder ändern
FAIRNESS (p & EX !p)|(!p & EX p)
```

$\mathbf{A}\ \mathrm{GF}\ (\ p \wedge \mathrm{EX} \neg p\ ) \vee (\neg p \wedge \mathrm{EX}\ p\ )$

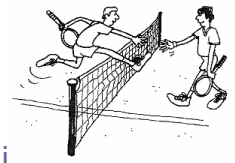
```
MODULE main
VAR
        p1 : myproc;
        ...
        pn : myproc;
```

```
MODULE myproc
...
FAIRNESS running
```

$\mathbf{A}\ (\mathrm{GF}\ p_1.\mathrm{running}\ ) \wedge \ldots \wedge (\mathrm{GF}\ p_n.\mathrm{running}\ )$

Leider kann man Fairness-Bedingungen in CTL nicht ausdrücken:

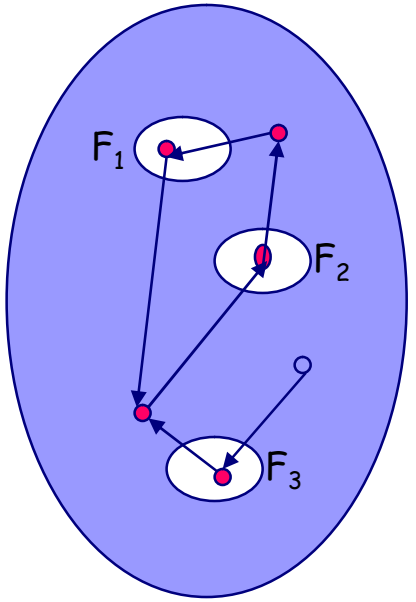
Es gibt keinen CTL-Ausdruck, der äquivalent zu $\mathbf{A}\ \mathrm{GF}\ \varphi$ ist.



© H. Peter Gumm, Philipps-Uni

Fairness

- Sei $F=(F_1,\ldots,F_n)$ Familie von Teilmengen $F_i \subseteq S$ der Kripke-Struktur $(S,\rightarrow,L)$.
- Ein Pfad σ heißt *F-fair*, falls er jede Fairness-Menge unendlich oft trifft.
- Falls es für jedes F_i eine logische Formel ψ_i gibt mit $F_i=[\psi_i]$, so ist σ genau dann F-fair, falls $\sigma \models GF\psi_i$ für jedes i.
  - da die ψ_i Zustandsmengen beschreiben, sind sie i.A. keine LTL-Formeln.
  - selbst wenn jedes ψ_i eine CTL-Formel ist, ist $GF\psi_i$ keine CTL-Formel
- Bedingte Pfad-Quantoren
  - $A_F\,\varphi$: auf allen F-fairen Pfaden gilt φ
  - $E_F\,\varphi$: es gibt einen F-fairen Pfad, der φ erfüllt.



Der gezeigte Pfad ist (F_1,F_2)-fair,
aber nicht (F_1,F_2,F_3)-fair.

© H. Peter Gumm, Philipps-Universität Marburg

CTL-Fairness

$C=\{c_1,\ldots, c_n\}$ sei endliche Menge von CTL-Ausdrücken.
Die C-fairen Pfadquantoren kann man in CTL* beschreiben:

$$\mathbf{A}_C\,\varphi \quad :\equiv \quad \mathbf{A}\ (\varphi \vee \mathbf{FG}\neg c_1 \vee \ldots \vee \mathbf{FG}\neg c_n)$$

Auf allen C-fairen Pfaden, gilt φ.

$$\mathbf{E}_C\,\varphi \quad :\equiv \quad \mathbf{E}\ (\varphi \wedge \mathbf{GF}c_1 \wedge \ldots \wedge \mathbf{GF}c_n)$$

Es gibt einen C-fairen Pfaden, auf dem φ gilt.

Wir müssen den Model-Checker Algorithmus modifizieren, um in Gegenwart von Fairnessbedingungen die gewünschte Semantik zu erhalten.

© H. Peter Gumm, Philipps-Universität Marburg

Reduktion

Sei $C=\{ c_1,\ldots,c_n\}$ eine Menge von CTL-Ausdrücken. Wir ersetzen AX, AF, AG, AU, sowie EX, EF, EG, EU durch die fairen Varianten A_CX, A_CF, A_CG, A_CU und E_CX, E_CF, E_CG, E_CU. Zum Glück gilt:

Alle fairen CTL-Operatoren lassen sich (in CTL) auf E_CG zurückführen :

$E_CX\ p = EX\ (\ p \wedge E_CG\ tt\ )$

$E_CF\ p = EF\ (\ p \wedge E_CG\ tt\ )$

$E_C\ [p\ U\ q\ ] = E(\ p\ U\ (q \wedge E_CG\ tt\ )\ )$

$A_CX\ p = \neg E_CX\ (\neg\ p\ )$

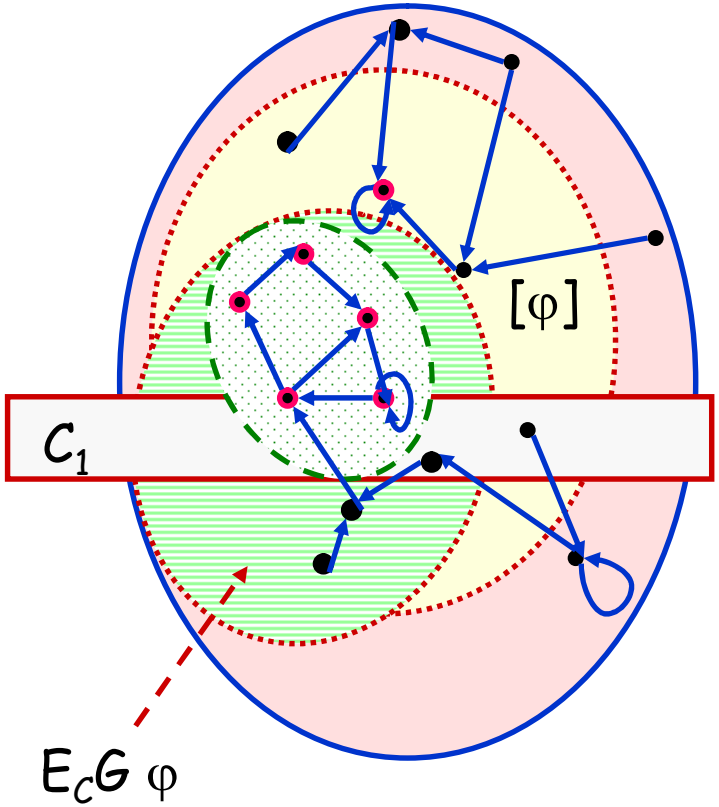
$A_CF\ p = \neg E_CG \neg p$

$A_C\ [p\ U\ q\ ] = \neg\ E_C[\neg q\ U\ \neg p \wedge \neg q] \wedge (\neg E_CG \neg q)$

© H. Peter Gumm, Philipps-Universität Marburg

$E_C G\ \varphi$

- Sei $C = \{ c_1, .., c_k\}$ die Menge der CTL-Fairnesseigenschaften.
- Jedes $c_r \in C$ bestimmt Teilmenge $C_r \subseteq S$.
- Bestimme nur die Zusammenhangs-komponenten K_i in $[\varphi]$, mit $(K_i \cap C_r) \neq \emptyset$ für alle $1 \leq r \leq k$.
- Es gilt: $s \in E_C G\ \varphi$
  $\Leftrightarrow$ es gibt einen Pfad in $[\varphi]$ zu einem der K_i.
  - Beweis: Sei $\sigma=(s_0,\ldots,s_n)$ ein Pfad in $[\varphi]$ mit $s_n \in K_i$. Da $K_i \cap C_j \neq \emptyset$ für alle $j \leq k$., lässt sich σ durch eine Schleife fortsetzen, die ganz in K_i verläuft und jedes C_j trifft. Es folgt, dass $s \in E_C G\varphi$.
  - Sei umgekehrt $s \in E_C G\varphi$, dann gibt es einen C-fairen Pfad, auf dem stets φ erfüllt ist. Er muss also irgendwann in eine Zusammenhangskomponente K_i hineinlaufen, wo er dauernd verbleibt. Weil der Pfad fair ist, muss K_i jedes C_j treffen.



© H. Peter Gumm, Philipps-Universität Marburg