Verzweigungslogik

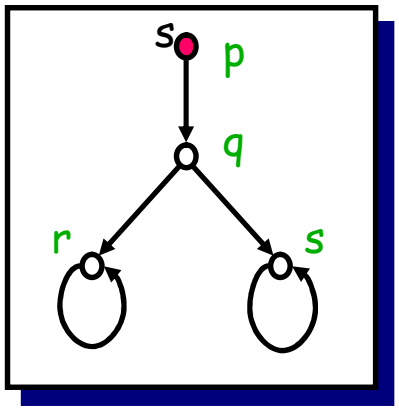
H. Peter Gumm

Philipps-Universität Marburg

Sommersemester 2007

Trennende Formeln ?

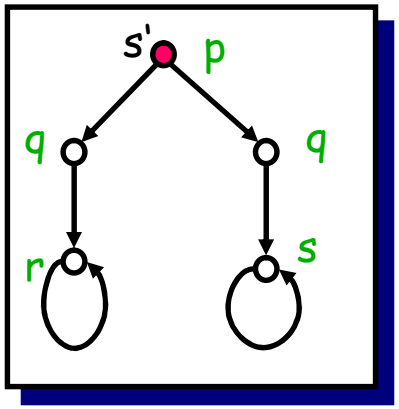
- Gibt es eine Logik, die **s** und **s'** in den beiden Figuren unterscheidet ?
  - In LTL unmöglich
  - In LTL kann man nur etwas aussagen über **alle** Pfade, die von einem Zustand ausgehen (**A**-Aussage)



AX EX r

A Auf jedem Pfad gilt, dass
X zum nächsten Zeitpunkt,
E ein Pfad existiert, auf dem
X zum nächsten Zeitpunkt
r gilt

- In CTL kann eine temporale Aussage über den Zustand **s** entweder etwas aussagen über
  - **alle Pfade**, die von **s** ausgehen (**A**-Aussage) oder über
  - **mindestens einen** Pfad, der von **s** ausgeht (**E**-Aussage)
  - Beispiele:
    - AX EX r
    - EX AX ¬r

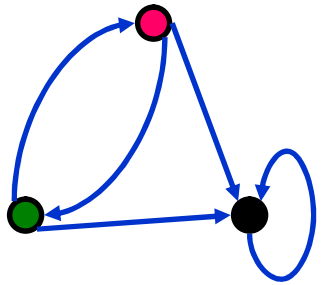


EX AX ¬r

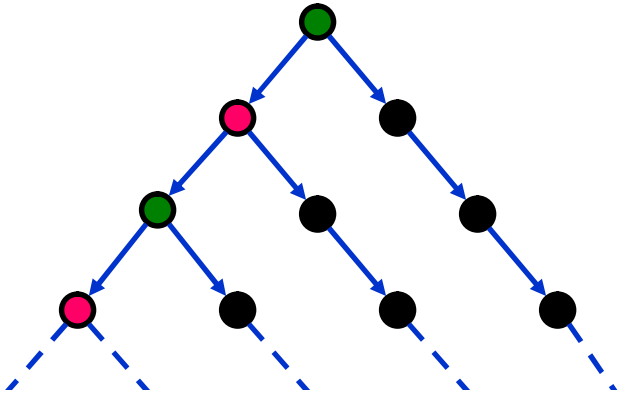
E Es gibt einen Pfad, auf dem
X zum nächsten Zeitpunkt auf
A allen ausgehenden Pfaden
X zum nächsten Zeitpunkt
¬r gilt

Berechnungsbäume

- Die in einem Zustand *s* beginnenden möglichen Berechnungen kann man in einen unendlichen Baum „entfalten" (engl.: *to unfold*):
  - *s* ist die Wurzel
  - Söhne sind die Knoten s' mit (s,s') $\in$ R.
  - Es entsteht ein unendlicher „Berechnungsbaum"
- Jeder Pfad in dem Baum entspricht einer Berechnung - und umgekehrt
- Der Baum verdeutlicht aber noch mehr Informationen – z.B. die möglichen Verzweigungen der Berechnung



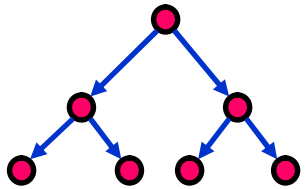
Kripke-Struktur
mit 3 Zuständen



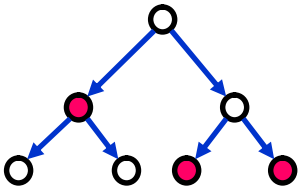
Zugehöriger Berechnungsbaum
- Nur die ersten 4 Ebenen dargestellt

Pfad-Quantoren - Pfadoperatoren

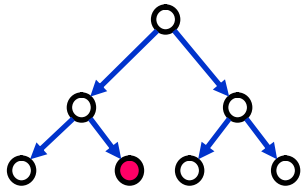
- Pfadoperatoren wie in LTL
  - X, F , G, U
- Pfad-Quantoren:
  - A – für alle Pfade
  - E – es existiert ein Pfad
- CTL-Formeln sind Kombinationspaare:
  - Pfadquantor + LTL-Operator
- Grundlegende Operatoren
  - AX, EX, AF, EF, AG, EG, AU, EU
  - Logische Operatoren $\neg$, $\wedge$, $\vee$, true, false
- Intendierte Bedeutung:
  - $s \models AF\ p$: Jede in s startende Berechnung erfüllt Fp
  - $s \models E[p\ U\ q]$: Es gibt eine in s startende Berechnung, die pUq erfüllt
- CTL Formeln können beliebig geschachtelt werden
  - AG AF p
  - AG E[p U AG q]
  - Aber nicht : A G[p U G q]
- Interpretation am besten über Berechnungsbäumen



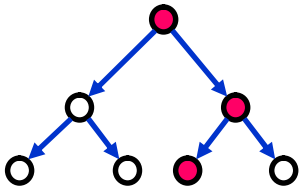
AG rot



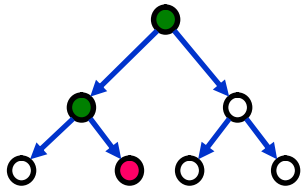
AF rot



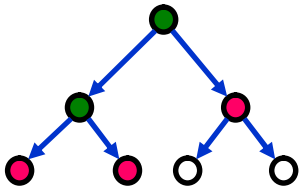
EF rot



EG rot



E(grün U rot)



A(grün U rot)

Computation Tree Logic - Syntax

```
Q      ::= A | E

SProp  ::= P
         | QX SProp
         | QF Sprop
         | QG Sprop
         | Q [ SProp U SProp ]
         | Q [ SProp W SProp ]
         | SProp ∧ SProp
         | ¬ SProp
         | true | false
```

A – für alle Pfade,
E - es existiert ein Pfad

X – next time
F – sometimes
G – always
U – until
W – waits for, weak until, unless

Immer nur Kombinationen erlaubt:
Pfad-Quantor **A** oder **E** mit temporalem Operator **X, U, F, G, W**.

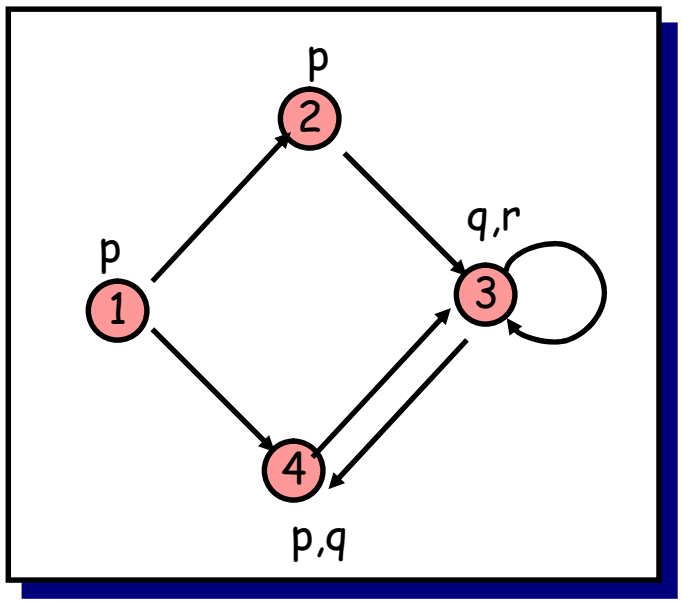
Mögliche Kombinationen sind:
AX .... , **AG** ... , **AF** ... , **A[** ... **U** ... **]**, **A[** ... **W** ... **]**,
EX .... , **EG** ... , **EF** ... , **E[** ... **U** ... **]**, **E[** ... **W** ... **]**.

Beispiel

CTL-Ausdrücke spezifizieren **Zustände s** einer Kripke-Struktur:
alle (**A**) von s ausgehenden, oder
mindestens eine (**E**) in s ausgehende Berechnung
erfüllt eine Pfad-Eigenschaft $X\varphi$, $F\varphi$, $G\varphi$, $\varphi\ U\ \psi$, oder $\varphi\ W\ \psi$.

In φ und ψ können wieder CTL-Ausdrücke und logische Operatoren vorkommen.

Die folgenden CTL-Aussagen sind in der links dargestellten Kripke Struktur richtig:



$1 \models \mathbf{AF}q$	Jeder von 1 ausgehende Pfad erfüllt irgendwann q
$3 \models \neg\, \mathbf{EG}p$	Es gibt keinen von 3 ausgehenden Pfad, der immer p erfüllt
$\models \mathbf{AG\ AF}r$	Jeder Pfad erfüllt unendlich oft r
$\models \mathbf{AG}\ (p \vee q)$	In jedem erreichbaren Zustand gilt $p \vee q$
$\models \mathbf{A}\ [\ p\ \mathbf{U}\ q\ ]$	Auf jedem Pfad gilt p**U**q.

Formale Semantik von CTL

Die Semantik von CTL-Formeln ist für Zustände definiert.

Sei $(S,\rightarrow,L)$ eine Kripke-Struktur, $s \in S$
seien φ, ψ CTL-Formeln und $p \in AP$.

$$
\begin{aligned}
Q &::= A \mid E \\
SProp &::= AP \\
&\mid Q\,X\,SProp \\
&\mid Q\,F\,Sprop \\
&\mid Q\,G\,Sprop \\
&\mid Q\,[\,SProp\ U\ SProp\,] \\
&\mid Q\,[\,SProp\ W\ SProp\,] \\
&\mid SProp \wedge SProp \\
&\mid \neg\ SProp \\
&\mid true \mid false
\end{aligned}
$$

Syntax von CTL:

$$
\begin{aligned}
s \models p &:\Leftrightarrow p \in L(s) \\
s \models E\,X\,\varphi &:\Leftrightarrow \exists s' \in S.\ s \rightarrow s' \wedge s' \models \varphi \\
s \models EG\,\varphi &:\Leftrightarrow \exists \sigma \in Paths(s).\ \forall k \in \mathbb{N}.\ \sigma(k) \models \varphi. \\
s \models E[\,\varphi\ U\ \psi\,] &:\Leftrightarrow \exists \sigma \in Paths(s).\ \exists k \in \mathbb{N}.\ \sigma(k) \models \psi \wedge \forall i<k.\ \sigma(i) \models \varphi.
\end{aligned}
$$

$\sigma(i) \models \varphi$ und
nicht $\sigma^i \models \varphi$!!!

Zusätzliche Operatoren

- Die anderen CTL-Operatoren definiert man als Abkürzungen

$EF\ \varphi$	:=	$E [true\ U\ \varphi]$

$AX\ \varphi$	:=	$\neg EX \neg \varphi$
$AF\ \varphi$	:=	$\neg EG \neg \varphi$
$AG\ \varphi$	:=	$\neg EF \neg \varphi$
$A [\varphi\ U\ \psi]$	:=	$\neg EG\neg\psi \ \wedge \neg E [\neg\psi\ U\ (\neg\varphi \wedge\neg\psi)]$

$E [\ \varphi\ W\ \psi\ ]$	:=	$\neg A [(\varphi \wedge\neg\psi)\ U\ (\neg\varphi \wedge\neg\psi)]$
$A [\ \varphi\ W\ \psi\ ]$	:=	$\neg E [(\varphi \wedge\neg\psi)\ U\ (\neg\varphi \wedge\neg\psi)]$

Einige Tautologien

- Zwei CTL-Formeln φ und ψ heißen semantisch äquivalent

  $$\varphi \equiv_{CTL} \psi$$

  falls in jedem Zustand s jeder Kripke-Struktur gilt:

  $$s \models \varphi \quad \text{gdw.} \quad s \models \psi.$$

- Wichtige semantische Äquivalenzen sind:

$$\neg AX\,\varphi \equiv EX\,\neg\varphi$$

$$\neg AF\,\varphi \equiv EG\,\neg\varphi$$
$$\neg EF\,\varphi \equiv AG\,\neg\varphi$$

$$EF\,\varphi \equiv E[\,\text{true}\ U\ \varphi\,]$$
$$AF\,\varphi \equiv A[\,\text{true}\ U\ \varphi\,]$$
$$EG\,\varphi \equiv E[\,\varphi\ W\ \text{false}\,]$$
$$AG\,\varphi \equiv A[\,\varphi\ W\ \text{false}\,]$$

Fixpunktgleichungen

- Die folgenden Fixpunktgleichungen werden wir später zum Model Checking heranziehen
- Die linken Seiten sind entweder als größte oder als kleinste Lösungen dieser Gleichungen definiert

$$
\begin{aligned}
AF\ \varphi &\equiv \varphi \vee AX\ AF\ \varphi \\
AG\ \varphi &\equiv \varphi \wedge AX\ AG\ \varphi \\
A[\varphi\ U\ \psi] &\equiv \psi \vee (\varphi \wedge AX\ A[\varphi\ U\ \psi])
\end{aligned}
$$

$$
\begin{aligned}
EF\ \varphi &\equiv \varphi \vee EX\ EF\ \varphi \\
EG\ \varphi &\equiv \varphi \wedge EX\ EG\ \varphi \\
E[\varphi\ U\ \psi] &\equiv \psi \vee (\varphi \wedge EX\ E[\varphi\ U\ \psi])
\end{aligned}
$$

Distributivgesetze

- Distributivgesetze

$$AG(\phi \wedge \psi) \equiv AG(\phi) \wedge AG(\psi)$$

$$EF(\phi \vee \psi) \equiv EF(\phi) \vee EF(\psi)$$

- Jedoch:

$$EG(\phi \wedge \psi) \neq EG(\phi) \wedge EG(\psi)$$
$$AF(\phi \wedge \psi) \neq AF(\phi) \wedge AF(\psi)$$
$$AF(\phi \vee \psi) \neq AF(\phi) \vee AF(\psi)$$
$$EG(\phi \vee \psi) \neq EG(\phi) \vee EG(\psi)$$

Positive Normalform

- Jede CTL-Formel lässt sich in eine Form bringen, in der Negationen nur vor Propositionen $p \in AP$ stehen.

- Algorithmus:
  Wende folgende Gleichungen von links nach rechts an:

  - $\neg true = false$
  - $\neg false = true$
  - $\neg\neg\phi = \phi$
  - $\neg(\phi \vee \psi) = \neg\phi \wedge \neg\psi$
  - $\neg(\phi \wedge \psi) = \neg\phi \vee \neg\psi$

  - $\neg EG\phi = AF\neg\phi$
  - $\neg EF\phi = AG\neg\phi$
  - $\neg AG\phi = EF\neg\phi$
  - $\neg AF\phi = EG\neg\phi$

  - $\neg AX\phi = EX\neg\phi$
  - $\neg EX\phi = EX\neg\phi$

Aber was soll man mit AU, EU machen ?

$\neg\ A[\phi\ U\ \psi\ ] = E[\ ???\ ]$
$\neg\ E[\phi\ U\ \psi\ ] = A[\ ???\ ]$

Releases

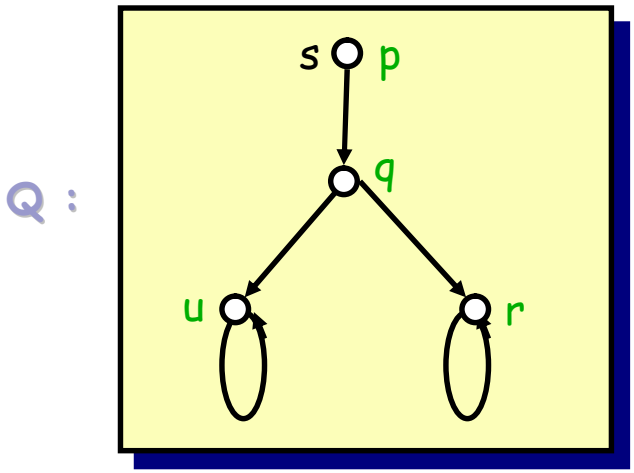
- Wir definieren weitere Operatoren AR und ER durch
  - $A[\phi \; R \; \psi] \;:=\; \neg E[\neg\phi \; U \; \neg\psi]$
  - $E[\phi \; R \; \psi] \;:=\; \neg A[\neg\phi \; U \; \neg\psi \;]$
- Damit gilt
  - $\neg A[\phi \; R \; \psi] = E[\neg\phi \; U \; \neg \; \psi]$
  - $\neg E[\phi \; R \; \psi] = A[\neg\phi \; U \; \neg \; \psi]$
  - $\neg A[\phi \; U \; \psi] = E[\neg\phi \; R \; \neg \; \psi]$
  - $\neg A[\phi \; U \; \psi] = E[\neg\phi \; R \; \neg \; \psi]$
- Unter Verwendung von AR, ER ist positive Normalform möglich

Trennschärfe von CTL

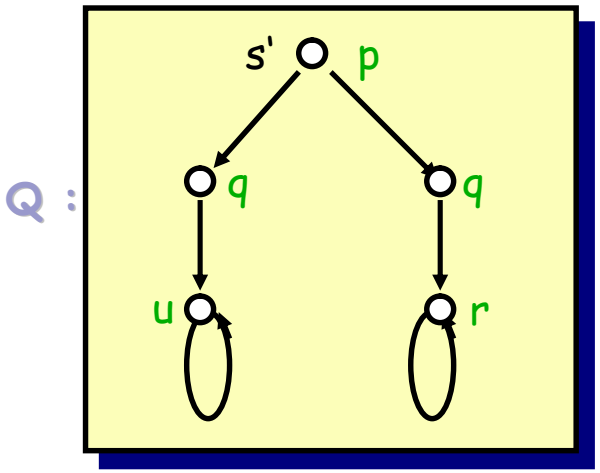
Die Punkte s und s' des folgenden Beispiels können wir mit CTL trennen:

$$\psi := AX\,(AX\,u \vee AX\,r\,).$$

$s \models \neg\psi$



$s' \models \psi$



Mit LTL können wir s und s' nicht unterscheiden, da Traces(s)=Traces(s')

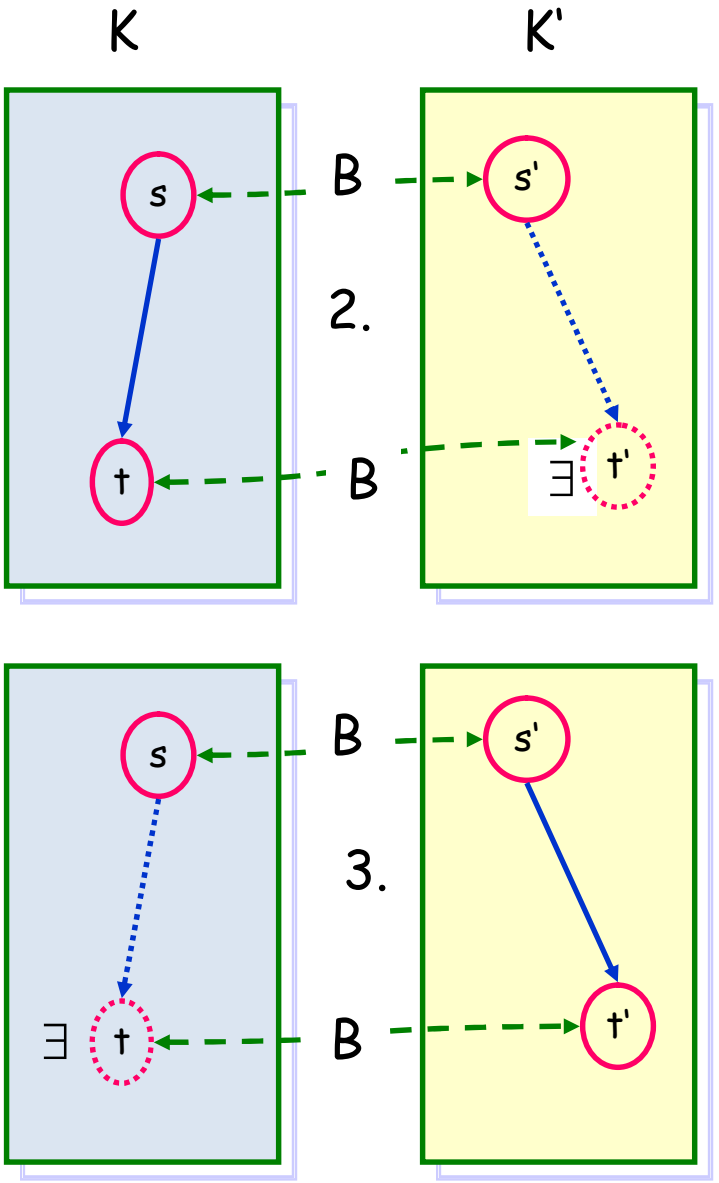
Bisimulation

K=(S, $\rightarrow$,L) und K'=(S', $\rightarrow$',L') Kripke-Strukturen.

Eine zweistellige Relation $B \subseteq S \times S'$
heißt Bisimulation, falls

$\forall s \in S. \forall s' \in S'. s\ B\ s' \Rightarrow$

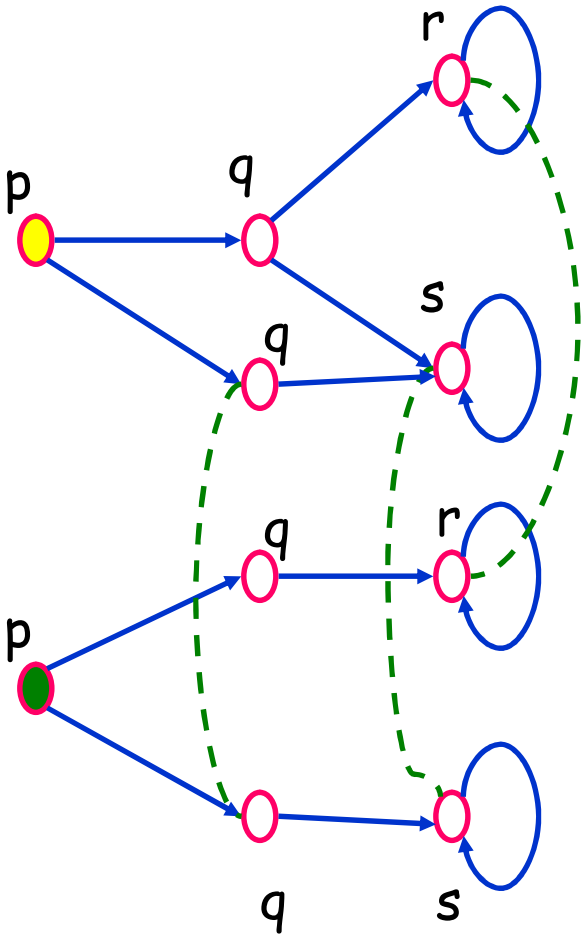
1. $L(s) = L'(s')$,
2. $\forall t \in S. s \rightarrow t \Rightarrow \exists t' \in S'. s' \rightarrow' t' \wedge t\ B\ t'$
3. $\forall t' \in S'. S' \rightarrow' t' \Rightarrow \exists t \in S. s \rightarrow t \wedge t\ B\ t'$



- B heißt links-total, falls zu jedem $s \in S$ ein $s' \in S'$ existiert mit $s\ B\ s'$.
- Rechts-total analog.
- Total := links.-total + rechts-total

Grösste Bisimulation

- Seien K und K' Kripke-Strukturen über P.
  - $\varnothing \subseteq B \times B'$ ist immer eine Bisimulation zwischen K und K'.
  - Ist $(B_i)_{i \in I}$ eine Familie von Bisimulationen zwischen K und K', dann auch $\cup_{i \in I} B_i$.
- Folgerung:
  - Zwischen zwei P-Kripke-Strukturen K und K' gibt es immer eine größte Bisimulation. Diese nennen wir $\sim_{K,K'}$.
  - Zustände s, s' mit $(s, s') \in \sim_{K,K'}$ heißen bisimilar.
  - Die größte Bisimulation zwischen K und K heißt $\sim_K$.
  - $\sim_K$ ist immer eine Äquivalenzrelation.



Kripke-Struktur mit größter Bisimulation

Bisimilar $\Rightarrow$ Spur-Äquivalent

Seien s und s' in der Bisimulation B

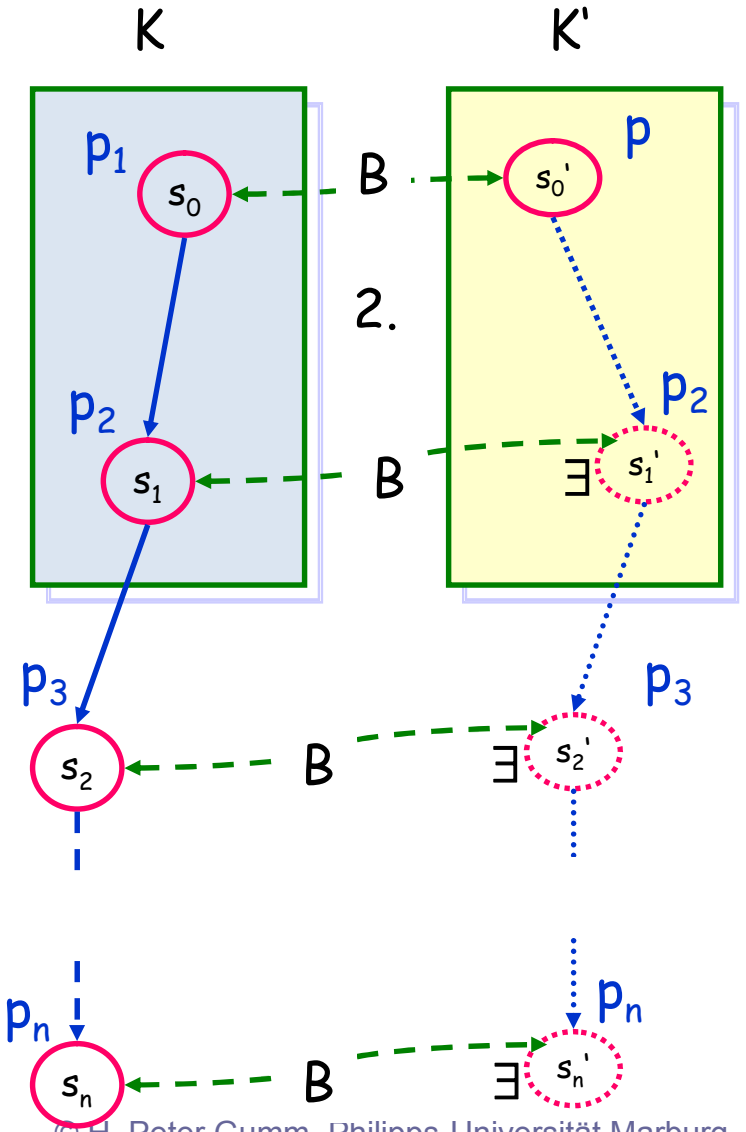
$\Rightarrow$ zu jedem Pfad $\sigma = (s_0,\ldots,s_n,\ldots)$, der in $s=s_0$ beginnt,

gibt es entsprechenden Pfad $\sigma' = (s_0',\ldots,s_n',\ldots)$,

der in $s' = s_0$ beginnt, so dass s_i B s_i' für jedes $i \in \mathbb{N}$

<u>Folgerung</u>: $\forall\, i \in \mathbb{N}.\ L(s_i) = L(s_i')$,

also sind s und s' spur-äquivalent.



Bisimilar $\Rightarrow$ CTL-Äquivalent

Lemma: Bisimilare Zustände sind CTL-äquivalent

Beweis: Seien $s \sim s'$ und $s \models \psi$. Zeige: $s' \models \psi$.
Induktion über den Aufbau von ψ:

1. $\psi \in AP$: $\psi \in L(s)$, wegen $s \sim s'$ also auch $\psi \in L(s')$, somit $s' \models \psi$.

2. $\psi = EX\,\varphi$: Es gibt ein t mit $s \to t$ und $t \models \varphi$. Also gibt es t' mit $s' \to t'$ und $t \sim t'$. Nach Ind.Vorr. also $t' \models \varphi$, also $s' \models EX\varphi$, d.h. $s' \models \psi$.

3. $\psi = EG\,\varphi$: Es ex. Pfad $s=s_0,s_1,\ldots,s_n,\ldots$ mit $s_i \to s_{i+1}$ und $s_i \models \varphi$ für alle $i \in \mathbb{N}$. Bisimulation liefert Pfad $s'=s_0',s_1',\ldots,s_n',\ldots$ mit $s_i' \to s_{i+1}'$ und $s_i \sim s_i'$. Ind.Vorr. liefert $s_i' \models \varphi$ für jedes i, also $s_0' \models EG\varphi$, somit $s' \models \psi$.

4. $\psi = E[\varphi\ U\ \phi]$: $\exists\, n \in \mathbb{N}$ und Zustandsfolge $s=s_0, s_1, \ldots, s_n$ mit $s_n \models \phi$ und $s_i \models \varphi$ für alle $i<n$. Bisimulation und Ind.-Vorr. liefern entsprechende Folge $s'=s_0', s_1', \ldots, s_n'$ mit $s_i' \models \varphi$ für alle $i<n$ und $s_n' \models \phi$. Da diese zum unendlichen Pfad erweitert werden kann, folgt $s' \models E[\varphi\ U\ \phi]$.

5. $\psi = \neg\varphi$: Aus $s' \models \varphi$ würde nach Ind.Vor. $s \models \varphi$ folgen. Daher $s' \models \neg\varphi$.

6. $\psi = \varphi_1 \wedge \varphi_2$: klar.

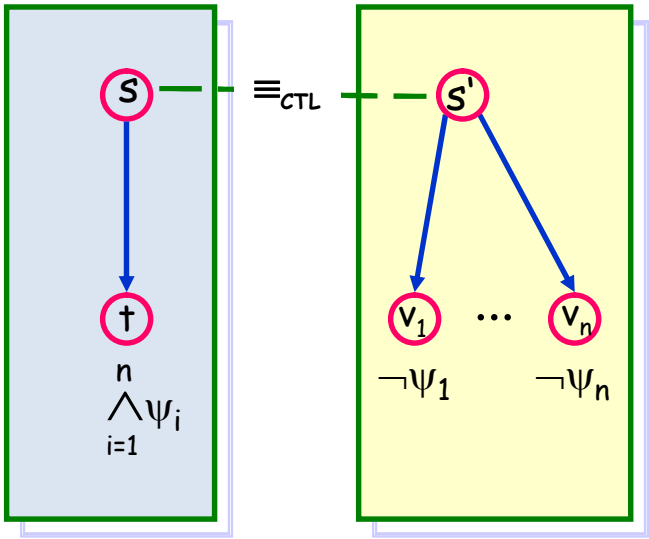
Bild-endlich $\Rightarrow$ $\equiv_{CTL} = \sim$

Satz(Hennessy,Milner): Ist K bild-endliche Kripke-Struktur, so ist $\equiv_{CTL}$ die größte Bisimulation.

Beweis: Es reicht zu zeigen, dass $\equiv_{CTL}$ eine Bisimulation ist.

Seien $s \equiv_{CTL} s'$.

1. $p \in L(s) \Leftrightarrow s \models p \Leftrightarrow s' \models p \Leftrightarrow p \in L(s')$.
   Folglich $L(s)=L(s')$.
2. Sei $s \to t$ und $s \equiv_{CTL} s'$. Gesucht: Ein t' mit $s' \to t'$ und $t \equiv_{CTL} t'$. Sei $N(s) := \{ v \mid s' \to v \} = \{ v_1, .., v_n \}$.
   Wenn $t \equiv_{CTL} v_i$, für keines der v_i gelten würde, so hätten wir für jedes der v_i eine CTL-Formel ψ_i mit $t \models \psi_i$ aber $v_i \models \neg\psi_i$.
   Somit auch
   $s \models EX\ \psi_1 \wedge \psi_2 \wedge ... \wedge \psi_n$, aber $s' \models \neg\ EX\ \psi_1 \wedge \psi_2 \wedge ... \wedge \psi_n$,
   im Widerspruch zu $s \equiv_{CTL} s'$.
3. Die dritte Bedingung ist symmetrisch.



Folgerung: In einer bild-endlichen Kripke-Struktur sind zwei Zustände genau dann CTL-äquivalent, wenn sie bisimilar sind.

CTL ist trennschärfer als LTL

Satz: Können wir zwei Zustände einer bild-endlichen Kripke-Struktur durch eine LTL-Formel trennen, dann können wir sie auch durch eine CTL-Formel trennen.

Beweis: s, s' in LTL unterscheidbar

$\Rightarrow$ s, s' nicht spur-äquivalent

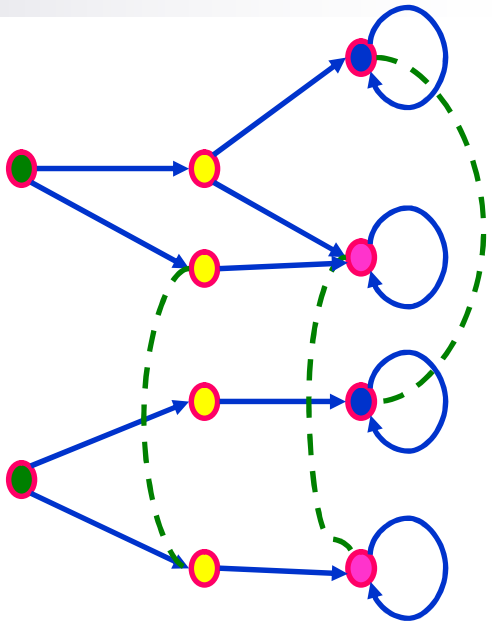
$\Rightarrow$ s, s' nicht CTL-äquivalent

$\Rightarrow$ es gibt CTL-Formel ψ mit

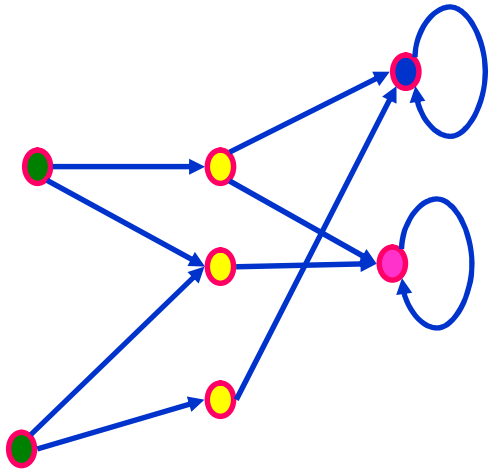
$$s \models \psi, s' \models \neg\psi$$

Vorsicht: Aus dem Satz kann man nicht schließen, daß jede LTL Formel auch durch eine CTL-Formel *ausdrückbar* wäre.

Zustandsminimierung

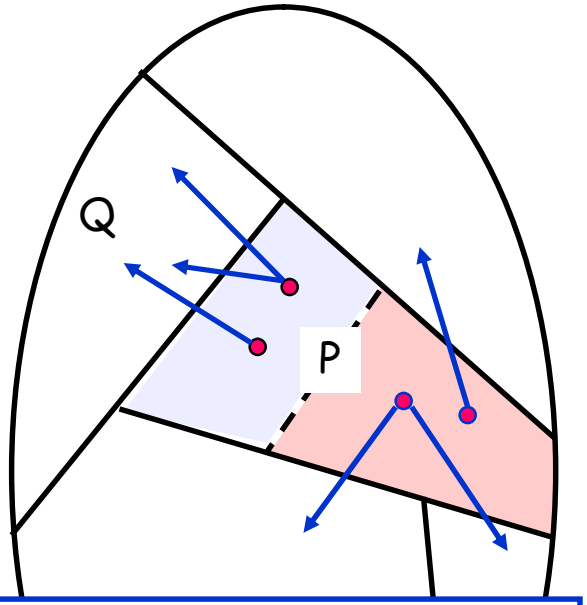


- Größte Bisimulation ~ auf einer Kripke-Struktur K=(S,R,AP) ist Äquivalenzrelation
  - reflexiv,
  - symmetrisch,
  - transitiv
- Faktormenge S/~ ist Menge aller Äquivalenzklassen:
  - $[s]_\sim := \{ s' \in S \mid s \sim s' \}$ -- Äquivalenzklasse von a
  - $S/\sim := \{ [s]_\sim \mid a \in S \}$ -- Faktormenge
- Auf Faktormenge S/~ erkläre Kripke-Struktur $K_\sim$:
  - $([s]_\sim , [s']_\sim) \in R_\sim :\Leftrightarrow \exists t \in [s]_\sim . \exists t' \in [s']_\sim . (t,t') \in R$ --
  - $L_\sim([a]_\sim) := L(a)$.
- Es gibt eine totale Bisimulation B zwischen K und $K_\sim$
  - $B := \{ (s, [s]_\sim) \mid s \in S \}$



Berechnung der größten Bisimulation

- Analog zur Berechnung des Minimalautomaten
  - B_0 gegeben durch Kern L
    - $s\ B_0\ s' :\Leftrightarrow L(s)=L(s')$
  - Für je zwei Klassen P,Q von B_i zerlege P_0 in
    - $\{ s \in P_0 \mid \exists\, s' \in Q.\ s \rightarrow s' \}$
    - $\{ s \in P_0 \mid \neg\exists\, s' \in Q.\ s \rightarrow s' \}$
  - Ergebnis ist B_{i+1}
  - Wiederhole bis $B_i = B_{i+1}$
  - Das Ergebnis ist die größte Bisimulation



Gegeben zwei Klassen P,Q
Zerteile P in
- die Zustände mit Transition nach Q
- die Zustände ohne Transition nach Q

Reduzierung des Zustandsraumes

Aus einer Kripke-Struktur K entsteht durch Minimierung eine kleinere Kripke-Struktur K/~

Beispiel: Bakery Algorithmus für wechselseitigen Ausschluss.
Hier mit zwei Prozessen. Zustandsraum ist unendlich !!

```
P0:
while(true){
   // Unkritisch
   x0 = x1 + 1;
   wait(x1==0||x0 < x1)
   // Kritischer Bereich
   x0 = 0;
}
```

```
P1:
while(true){
  // Unkritisch
  x1 = x0 + 1;
  wait(x0==0||x1 < x0)
  //Kritischer Bereich
  x1 = 0;
}
```

Idee: Beim Eintritt in die Bäckerei ziehe Nummer. Diese ist eins größer als alle bisher gezogenen. Wenn fertig, werfe Nummer weg. Der Kunde mit der kleinsten Nummer wird zuerst bedient.

Bakery

```
Pi:
while(true){
    // Unkritisch
0: x[i] = x[1-i] + 1;
1: wait(x[1-i]==0|x[i]<x[1-i])
    // Kritischer Bereich
2:   x[i] = 0;
}
```

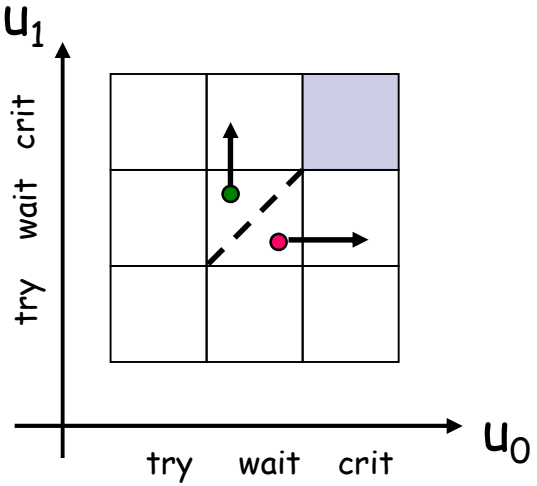
- Unendlicher Zustandsraum
- x_i, x_{1-i} können sich beliebig hochschaukeln
- Immerhin kann man mit SMV beweisen:

  $G\,(x_i<k) \Rightarrow G\,(\neg u_0.\text{kritisch} \vee \neg u_1.\text{kritisch})$

```
1  MODULE main
2  VAR
3    x0 : 0..99;
4    x1 : 0..99;
5    u0: process user(x0,x1);
6    u1: process user(x1,x0);
7  ASSIGN
8    init(x0):=0;
9    init(x1):=0;
10 LTLSPEC -- Sicherheit
11   G(x1<98) -> G(!u0.critical | !u1.critical)
12 LTLSPEC -- Lebendigkeit
13   G (u1.trying -> F u1.critical)
14
15 MODULE user(ich,du)
16 VAR
17   pc : 0 .. 2;
18 ASSIGN
19   init(pc) := 0;
20
21   next(pc) := case
22     trying
23     | waiting & !(du=0 | ich<du): 1;
24     TRUE                  :(pc+1) mod 3;
25   esac;
26
27   next(ich) := case
28     trying    : (du+1) mod 100;
29     critical  : 0;
30     TRUE      : ich;
31   esac;
32 DEFINE
33   trying   := pc=0;
34   waiting  := pc=1;
35   critical := pc=2;
36 FAIRNESS
37   running
```

Bisimulation

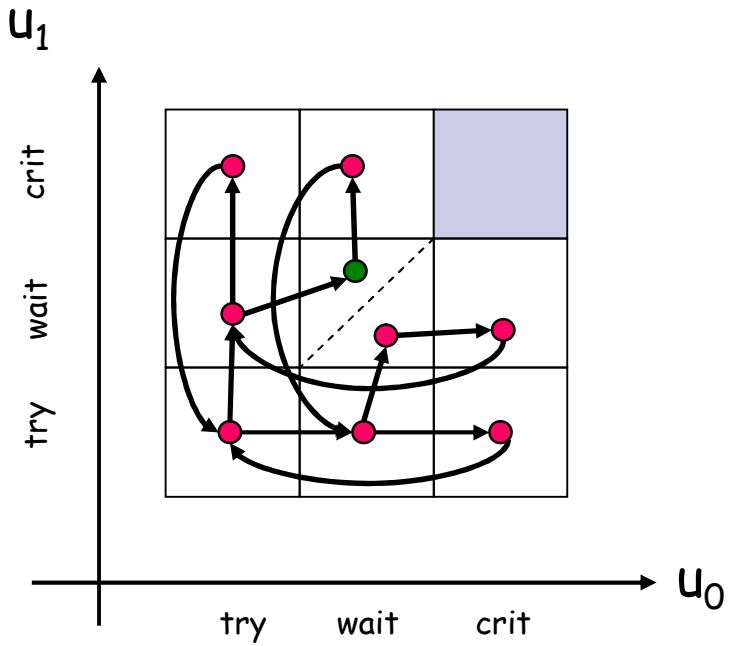
- Bakery=(S,I,→,L,AP) mit
  - S = {(u_0.pc, u_1.pc, x_0,x_1)∈ 3×3×ℕ×ℕ }
  - AP = { u_0.trying, u_1.trying, u_0.critical, u_1.critical }
    - nur diese sind für die interessierenden Eigenschaften notwendig
  - → : SMV-Spezifikation
  - L,I : siehe DEFINE-Klauseln und init-Klauseln



- Bisimulation
  - $s \sim s' \Rightarrow L(s) = L(s')$
    - also Unterteilung in mindestens 9 Klassen:
      - u_i.trying,u_i.critical, u_i.waitng := ¬(u_i.trying ∨ u_i.critical)
    - hoffen, dass die Klasse (u_0.critical∧ u_1.critical) leer ist
  - Umkehrung gilt nicht: $s \sim s' \Leftrightarrow L(s)=L(s')$ definiert keine Bisimulation, denn
    - $s=(1,1,1,2) \models u_0.wait \wedge u_1.wait$ und $s'=(1,1,2,1) \models u_0.wait \wedge u_1.wait$
    - aber: $s \rightarrow t \in u_1.crit$, dagegen $s' \not\rightarrow t' \in u_1.crit$
  - Also Aufteilung der Klasse (u_0.wait ∧ u_1.try)
    - Zustände x mit $x \rightarrow t \in u_1.crit$
    - andere (Zustände y mit $y \rightarrow u_0.crit$)

Bakery/~

- Wir faktorisieren nach der größten Bisimulation und erhalten als Quotient ein endliches System
- Eine Klasse ist leer, eine Klasse wird aufgeteilt
  - Das resultierende System hat 9 Zustände



Diese Zustandsmengen sind die Klassen der größten Bisimulation:

	(pc_0,pc_1,x_0,x_1)
(t,t)	$= (0, 0, 0, 0)$
(t,w)	$= (0, 1, 0, x_1 > 0)$
(t,c)	$= (0, 2, 0, x_1 > 0)$
(w,t)	$= (1, 0, x_0 > 0, 0)$
$(w,w)_a$	$= (1, 1, x_1 > x_0 > 0)$
$(w,w)_b$	$= (1, 1, x_0 > x_1 > 0)$
(w,c)	$= (1, 2, x_0 > x_1 > 0)$
(c,t)	$= (2, 0, x_0 > 0, 0)$
(c,w)	$= (2, 1, x_1 > x_0 > 0)$

Häufig verwendete CTL-Formeln

- *Sicherheit*: in jedem erreichbaren Zustand gilt ψ

  $AG(\psi)$

- *Lebendigkeit*: ein Zustand mit ψ ist erreichbar:

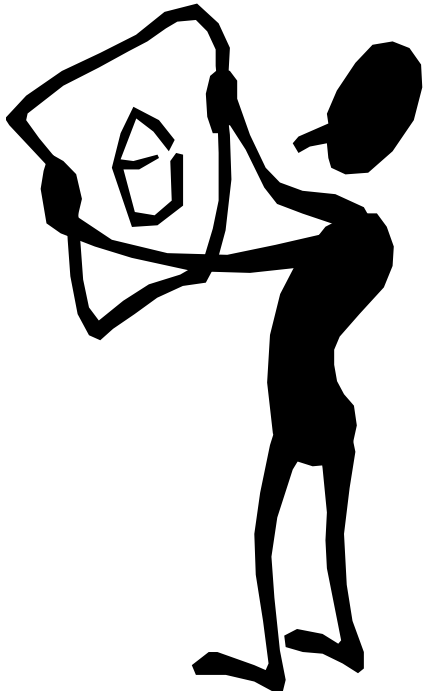
  $EF(\psi)$

- *Response* : φ zieht unweigerlich ψ nach sich

  $AG(\varphi \Rightarrow AF\ \psi)$

- *Non-blocking* : φ eröffnet di Möglichkeit, ψ zu erreichen

  $AG(\varphi \Rightarrow EF\ \psi)$



CTL in SMV

- In SMV kann man mit dem Schlüsselwort

  SPEC

  beliebige CTL-Formel φ spezifizieren.

- **SPEC** φ heißt:

  In jedem Anfangszustand ist φ wahr.
  Häufig sind die Formeln von der Art:

  $AG\ \varphi$.

- Mit dem Schlüsselwort **LTLSPEC** spezifiziert man LTL-Formeln ψ

- **LTLSPEC** ψ heißt:

  In jedem Anfangszustand gilt:

  $A\ \psi$

UltraEdit-32 - [C:\Dokumente und Einstellungen\Peter\Desktop\NeueVo...

File Edit Search Project View Format Column Macro Advanced Window Help

SimplePetereson.smv

```
ASSIGN
  init(turn)   := 0;
  init(flag[0]) := FALSE;
  init(flag[1]) := FALSE;

SPEC  -- Sicherheit - CTL
  AG !(P0.critical & P1.critical)

SPEC -- Lebendigkeit - CTL
  AG AF P0.critical & AG AF P1.critical;

LTLSPEC  -- Sicherheit - LTL
  G !(P0.critical & P1.critical)

LTLSPEC  -- Lebendigkeit - LTL
  G F P0.critical & G F P1.critical;

LTLSPEC  -- No deadlock
  G (P0.waiting & P1.waiting -> F (!P0.waiting | !P1.waiting));
```

For Help, press F1 | Ln 33, Col. 26, C0 | DOS | Mod: 22.03.2003 11:13:12 | File Size:

Spezifikationsmuster in CTL

- Gewisse Muster tauchen in Spezifikationen immer wieder auf. Sie lassen sich in CTL formulieren
- Wir benutzen hier W (weak until, unless) mit der Definition

$$p\ W\ q \quad :\Leftrightarrow \quad (G\ p) \vee (p\ U\ q).$$

Universalität	p gilt immer
Global	AG(p)
vor r	A[(p ∨ AG(¬r)) W r]
nach q	AG(q ⇒ AG(p))
zwischen q und r	AG(q∧¬r ⇒ A[(p∨AG(¬r)) W r])

Existenz	p gilt irgendwann
Global	AF(p)
vor r	A[¬r W (p ∧ ¬r)]
nach q	A[¬q W (q ∧ AF(p))]
zwischen q und r	AG(q∧¬r ⇒ A[¬r W (p ∧ ¬r)])

Spezifikationsmuster in CTL

- Wichtig sind Präzedenz und Response-Eigenschaften

Präzedenz	s vor p
Global	$A[\neg p \; W \; s]$
vor r	$A[(\neg p \vee AG(\neg r)) \; W \; (s \vee r)]$
nach q	$A[\neg q \; W \; (q \wedge A[\neg p \; W \; s])]$
zwischen q und r	$AG(q \wedge \neg r \Rightarrow A[(\neg p \vee AG(\neg r)) \; W \; (s \vee r)])$

Response	s reagiert auf p
Global	$AG(p \Rightarrow AF \; s)$
vor r	$A[((p \Rightarrow A[\neg r \; U \; (s \wedge \neg r)]) \vee AG(\neg r)) \; W \; r]$
nach q	$A[\neg q \; W \; (q \wedge AG(p \Rightarrow AF(s))]$
zwischen q und r	$AG(q \wedge \neg r \Rightarrow A[((p \Rightarrow A[\neg r \; U \; (s \wedge \neg r)]) \vee AG(\neg r)) \; W \; r])$

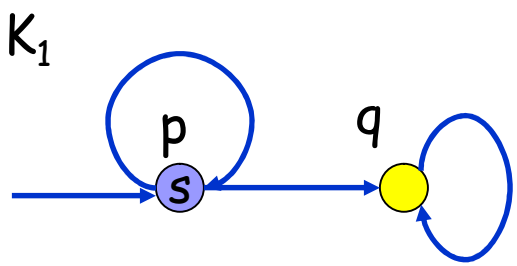
Äquivalenz von CTL und LTL-Formeln

- LTL kann man nicht unmittelbar mit CTL vergleichen
  - LTL-Formeln φ gelten für Pfade
  - CTL-Formeln Ψ gelten für Zustände
- Aus einer LTL-Formel φ gewinnen wir die Formel $A\varphi$ mit der Semantik
  - $s \models A\varphi :\Leftrightarrow$ Für alle $\sigma \in$ Paths(s). $\sigma \models \varphi$
- Seien φ aus LTL und Ψ aus CTL. Wir definieren
  - $\varphi \equiv \Psi \;:\Leftrightarrow$ Für jede Kripke-Struktur und jedes $s \in S$ gilt

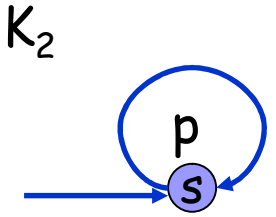
$$s \models A\varphi \Leftrightarrow s \models \Psi$$

Ausdrückbarkeit von CTL in LTL ?

- AG EF p lässt sich nicht in LTL ausdrücken
  - Beweis: Jede LTL-Formel, die in K_1 gilt, gilt auch in K_2 ,
  - wieso … ?



$s \models AG\ EF\ q$



$s \not\models AG\ EF\ q$

  - Andererseits AG EF q in K_1, nicht aber in K_2.
  - Daher kann AG EF p nicht äquivalent zu einer LTL-Formel sein.

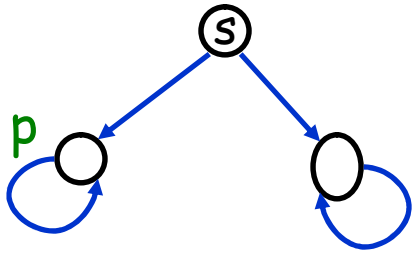
Satz von Clarke u. Draghicescu

Gegeben: CTL-Formel Ψ
Gefragt: Gibt es eine LTL-Formel φ mit $\Psi \equiv \varphi$?

- Sei ψ die LTL-Formel, die aus Ψ entsteht, wenn man alle Pfad-Quantoren weglässt

Satz (Clarke,Draghicescu): Entweder gilt $\Psi \equiv \psi$, oder es gibt keine zu Ψ äquivalente LTL-Formel.

- Anwendung: EG AF p ist nicht in LTL ausdrückbar

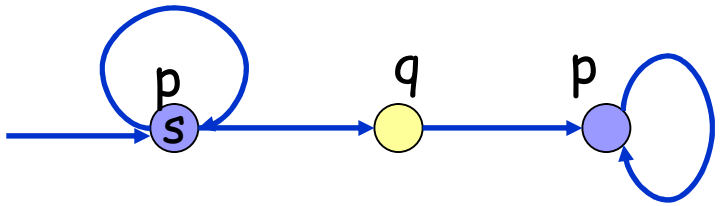


$s \models EG\ AF\ p$

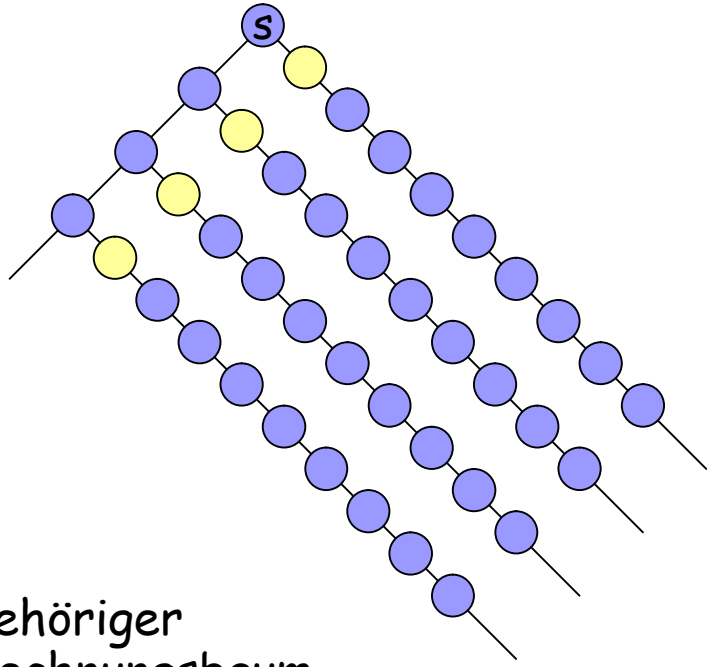
$s \not\models A\ GF\ p$

Ausdrückbarkeit von LTL in CTL

- Kann man die LTL-Formel FG p durch eine äquivalente CTL-Formel ausdrücken ?
- Erster Versuch: AF AG p
- AF AG p $\Rightarrow$ A FG p gilt immer. Was ist mit der Rückrichtung ?



Kripke-Struktur (L(●) = p)



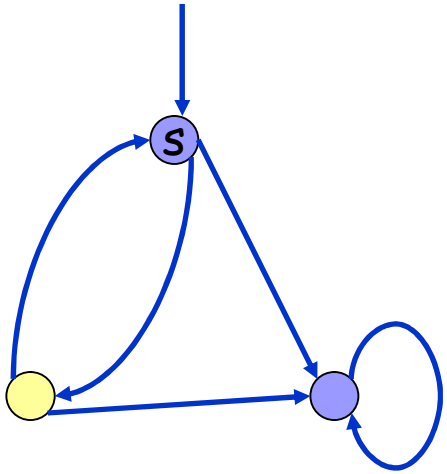
zugehöriger Berechnungsbaum (bisimilare Kripke-Struktur)

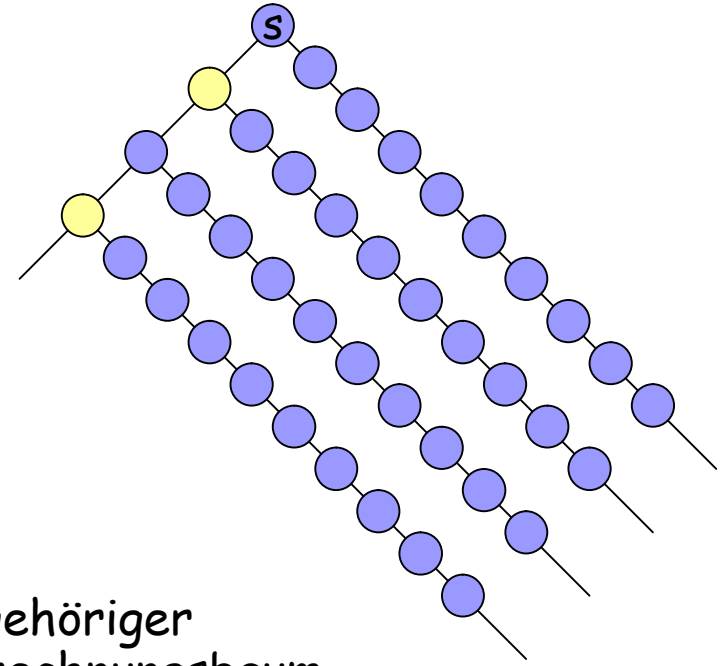
$s \models A\,FG\,p$ aber $s \models \neg\, AF\,AG\,p$

Folgerung: $A\,FG\,p \not\Rightarrow AF\,AG\,p$

Ausdrückbarkeit (3)

- Kann man die LTL-Formel FG p durch eine äquivalente CTL-Formel ausdrücken ?
- Zweiter Versuch: AF EG p





Kripke-Struktur (L( ) = p)

zugehöriger Berechnungsbaum (bisimilare Kripke-Struktur)

$s \models AF\ EG\ p$ aber $s \not\models A\ FG\ p$

Folgerung: $AF\ EG\ p \not\Rightarrow A\ FG\ p$

CTL*

CTL* steht für Extended Computation Tree Logic. Neben Aussagen über Zustände gibt es auch Aussagen über Pfade. Wir definieren, ausgehend wieder von einer Menge P von atomaren Ausagen durch wechselseitige Rekursion :

Zustandseigenschaften

$$
\begin{aligned}
\mathit{SProp} ::= {} & P \\
\mid {} & A\ \mathit{PProp} \\
\mid {} & E\ \mathit{PProp} \\
\mid {} & \mathit{SProp} \wedge \mathit{SProp} \\
\mid {} & \neg\ \mathit{SProp}
\end{aligned}
$$

Pfadeigenschaften

$$
\begin{aligned}
\mathit{PProp} ::= {} & \textbf{init}\ \mathit{SProp} \\
\mid {} & X\ \mathit{PProp} \\
\mid {} & \mathit{PProp}\ U\ \mathit{PProp} \\
\mid {} & \mathit{PProp} \wedge \mathit{PProp} \\
\mid {} & \neg\ \mathit{PProp}
\end{aligned}
$$

Die restlichen logischen Konnektoren, **true**, **false**, $\neg,\vee,\wedge,\Rightarrow,\Leftrightarrow$ fassen wir wie üblich als Abkürzungen auf. Ebenso seien die temporalen Operatoren **F**, **G** und **W** wie üblich definiert. "**init**" wird üblicherweise weggelassen.

Neu gegenüber CTL ist, daß die Pfadeigenschaften geschachtelt und boolesch kombiniert werden können. So ist in CTL* z.B. $A(pUq \wedge Xr)$ erlaubt, in CTL nicht.

Semantik von CTL*

Sei K=(S,R,L) eine beliebige Kripke-Struktur.
Für einen beliebigen Zustand **s** $\in S$, einen Pfad $\sigma = (s_0,\ldots,s_n,\ldots)$ sowie
für **p** $\in P$, $\varphi, \psi \in$ ***PProp*** definieren wir :

CTL* Syntax :

PProp | **init** SProp
| **X** PProp
| PProp **U** PProp
| PProp $\wedge$ PProp
| $\neg$ PProp

SProp ::= AP
| **A** PProp
| **E** PProp
| SProp $\wedge$ SProp
| $\neg$ SProp

Definition der CTL* Semantik :

$\sigma \models$ **init p** gdw. $s_0 \models$ **p**

$\sigma \models$ **X** φ gdw. $\sigma^1 \models \varphi$

$\sigma \models \varphi$ **U** ψ gdw. $\exists\, k \in Nat.\ \sigma^k \models \psi \wedge \forall i < k.\ \sigma^i \models \varphi.$

s $\models$ **p** gdw. **p** $\in$ **L** (**s**)

s $\models$ **A** φ gdw. für alle Pfade σ mit $\sigma(0)$= **s** gilt $\sigma \models \varphi$

s $\models$ **E** φ gdw. es gibt einen Pfad σ mit $\sigma(0)$= **s** und $\sigma \models \varphi$

Die Semantik der Booleschen Junktoren definieren wir wie üblich.

Verhältnis CTL*, CTL, LTL

CTL* beinhaltet sowohl LTL, als auch CTL.
LTL und CTL sind in ihrer Ausdruckskraft unvergleichbar.

Es gibt in LTL für die Praxis nützliche Ausdrücke, für die es keinen äquivalenten CTL-Ausdruck gibt, z.B.

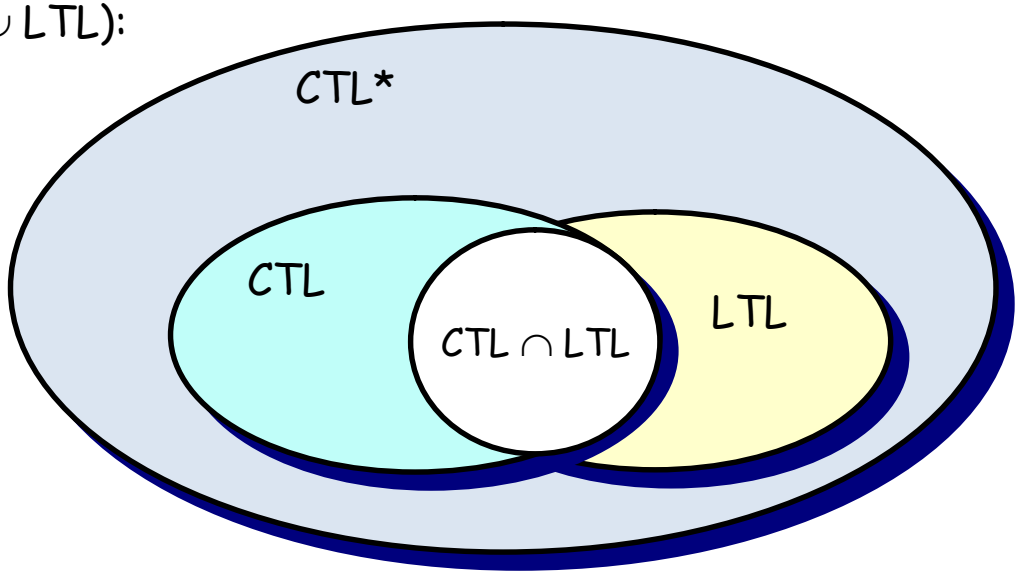
$$\mathbf{A\,[\,GFp \Rightarrow Fq\,]}$$

Dennoch kann CTL mehr Zustände trennen als LTL.

In CTL*, aber nicht ausdrückbar in (CTL ∪ LTL):
E[GF p]

In CTL aber nicht ausdrückbar in LTL:
AG EF p

In LTL aber nicht ausdrückbar in CTL:
A [GFp ⇒ Fq]



Aufgaben

1. Definieren Sie EF, EW, AX, AF, AG, AU und AW
   analog wie EX, EG und EU und zeigen Sie die Äquivalenzen

   $EF\ \varphi \equiv E[\text{true } U\ \varphi]$

   $AX\ \varphi \equiv \neg EX \neg \varphi$

   $AF\ \varphi \equiv \neg EG \neg \varphi$

   $AG\ \varphi \equiv \neg EF \neg \varphi$

   $A[\varphi\ U\ \psi] \equiv \neg EG\neg\psi \wedge \neg E[(\varphi \wedge \neg\psi)\ U\ (\neg\varphi \wedge \neg\psi)]$

   $E[\varphi\ W\ \psi] \equiv \neg A[(\varphi \wedge \neg\psi)\ U\ (\neg\varphi \wedge \neg\psi)]$

   $A[\varphi\ W\ \psi] \equiv \neg A[(\varphi \wedge \neg\psi)\ U\ (\neg\varphi \wedge \neg\psi)]$
2. Zeigen Sie: In LTL gilt FX p $\equiv_{LTL}$ XFp, in CTL gilt jedoch nicht: AF AX p $\equiv_{CTL}$ AX AF p.
3. Zeigen Sie, dass $A(\varphi\ U\ AX\psi) \Rightarrow A(\varphi\ U\ X\psi)$ gilt, nicht aber die Umkehrung.
4. Ist K=(S,R,AP) Kripke Struktur mit größter Bisimulation
   - Zeigen Sie: ~ ist eine Äquivalenzrelation
   - Zeigen Sie, dass für die minimale Kripke-Struktur $K_{\sim}$ gilt:
     - B = (s,[s]~) ist eine totale Bisimulation
     - B ist die grösste Bisimulation zwischen K und $K_{\sim}$.
5. Definieren Sie die Relation $\equiv_{LTL}$ analog zu $\equiv_{CTL}$. Wir haben gezeigt, dass $\equiv_{CTL}$ eine Bisimulation ist.
   - Was geht schief, wenn Sie den Beweis analog für $\equiv_{LTL}$ führen wollen ?
   - Finden Sie ein Kripke-System und Zustände s, s', die LTL-äquivalent sind, nicht aber bisimilar.

Aufgabe 7

a) Implementieren, spezifizieren und verifizieren Sie den Bakery-Algorithmus für drei User.

b) Die nebenstehende Variante des Bakery-Algorithmus (für zwei User) erlaubt diesen, entweder lustlos in der Bäckerei herumzulungern (idle) oder zu versuchen, den kritischen Bereich zu erreichen. Berechnen Sie bezüglich AP={u_0.trying, u_0.critical, u_1.trying, u_1.critical} die größte Bisimulation und skizzieren Sie den Quotienten (Hinweis: dieser hat 16 Zustände).

```
1  MODULE main
2  VAR
3    x0 : 0..99;
4    x1 : 0..99;
5    u0: process user(x0,x1);
6    u1: process user(x1,x0);
7  ASSIGN
8    init(x0):=0;
9    init(x1):=0;
10 LTLSPEC -- Sicherheit
11   G(x1<98) -> G(!u0.critical | !u1.critical)
12 LTLSPEC -- Lebendigkeit
13   G (u1.trying -> F u1.critical)
14
15 MODULE user(ich,du)
16 VAR
17   pc : 0 .. 3;
18 ASSIGN
19   init(pc) := 0;
20
21   next(pc) := case
22     idle                          : {0,1};
23     waiting & !(du=0 | ich<du): 2;
24     TRUE                          :(pc+1) mod 4;
25   esac;
26   next(ich) := case
27     trying     : (du+1) mod 100;
28     critical   : 0;
29       1        : ich;
30   esac;
31 DEFINE
32   idle     := pc=0;
33   trying   := pc=1;
34   waiting  := pc=2;
35   critical := pc=3;
36 FAIRNESS
37   running
```