



Verzweigungslogik

H. Peter Gumm

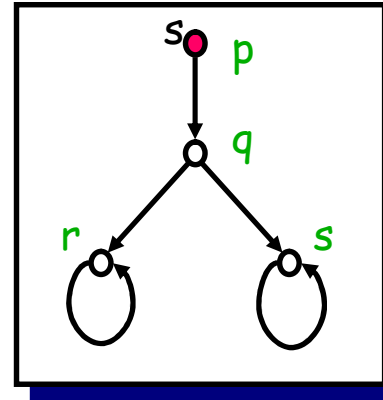
Philipps-Universität Marburg

Sommersemester 2007



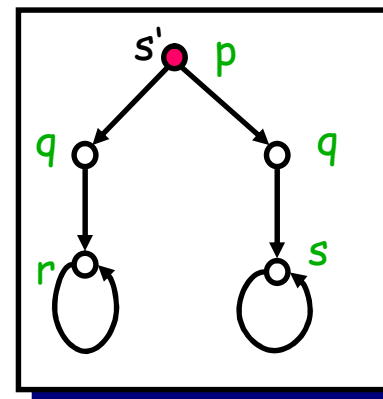
Trennende Formeln ?

- 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)
- 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:
 - $AX EX r$
 - $EX AX \neg r$



$AX EX r$

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



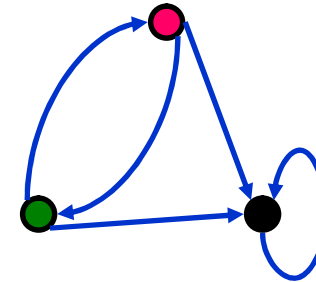
$EX AX \neg r$

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

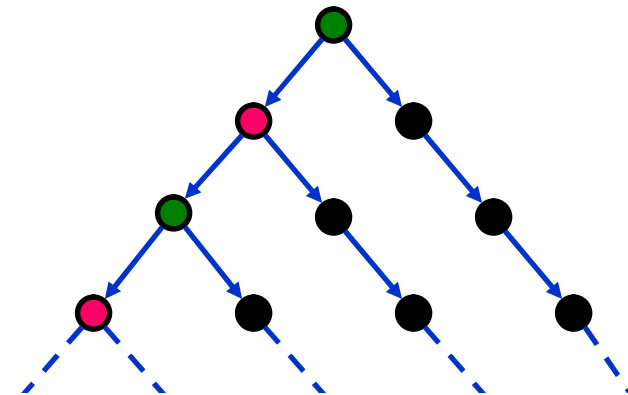


Berechnungsbäume

- 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

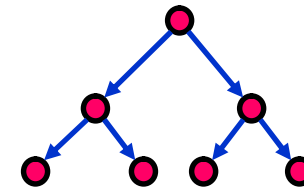


Zugehöriger Berechnungsbaum
- Nur die ersten 4 Ebenen dargestellt

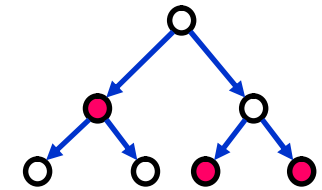


Pfad-Quantoren - Pfadoperatoren

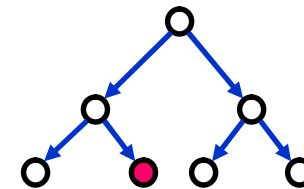
- Pfadoperatoren wie in LTL
 - X, F, G, U
- Pfad-Quantoren:
 - A - für alle Pfade
 - E - es existiert ein Pfad
- CTL-Formeln sind Kombinationspaare:
 - Pfadquantor + LTL-Operator
- Grundlegende Operatoren
 - $AX, EX, AF, EF, AG, EG, AU, EU$
 - Logische Operatoren $\neg, \wedge, \vee, \text{true}, \text{false}$
- 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 $p U q$ erfüllt
- CTL Formeln können beliebig geschachtelt werden
 - $AG AF p$
 - $AG E[p U AG q]$
 - **Aber nicht** : $AG[p U G q]$
- Interpretation am besten über Berechnungsbäumen



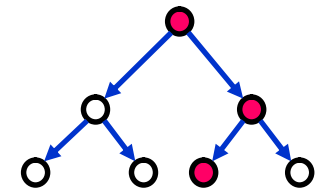
AG rot



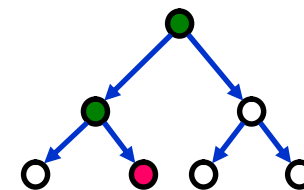
AF rot



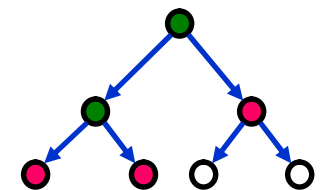
EF rot



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 **a**lle Pfade,
E - es **e**xistiert ein Pfad

X - **n**ext time
F - **s**ometimes
G - **a**lways
U - **u**ntil
W - **w**aits for, **w**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** ...].

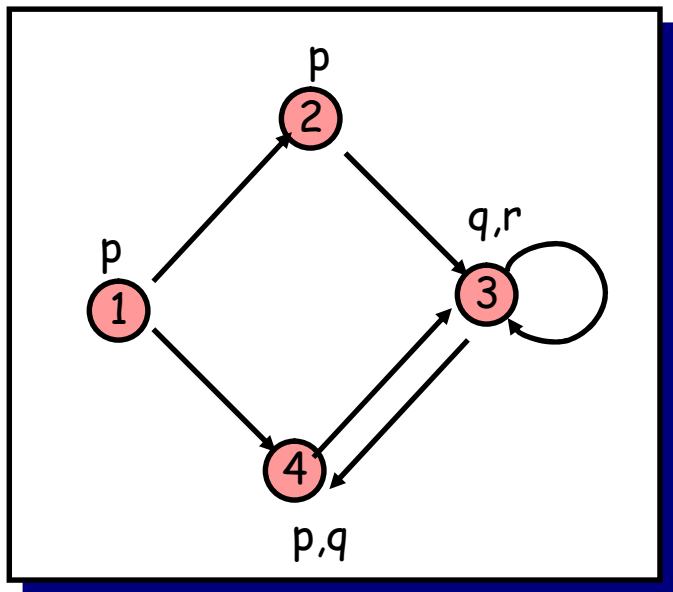


Beispiel

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:



$$1 \models \mathbf{AF}q$$

$$3 \models \neg \mathbf{EG}p$$

$$\models \mathbf{AG} \mathbf{AF}r$$

$$\models \mathbf{AG} (p \vee q)$$

$$\models \mathbf{A} [p \mathbf{U} q]$$

Jeder von 1 ausgehende Pfad erfüllt irgendwann q

Es gibt keinen von 3 ausgehenden Pfad, der immer p erfüllt

Jeder Pfad erfüllt unendlich oft r

In jedem erreichbaren Zustand gilt $p \vee q$

Auf jedem Pfad gilt $p \mathbf{U} q$.



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 $p \in AP$.

```

Q ::= A | E

SProp ::= AP
        | QX SProp
        | QF SProp
        | QG SProp
        | Q [ SProp U SProp ]
        | Q [ SProp W SProp ]
        | SProp ^ SProp
        | ¬ SProp
        | true | false

```

$s \models p \iff p \in L(s)$
 $s \models EX \varphi \iff \exists s' \in S. s \rightarrow s' \wedge s' \models \varphi$
 $s \models EG \varphi \iff \exists \sigma \in \text{Paths}(s). \forall k \in \mathbb{N}. \sigma(k) \models \varphi.$
 $s \models E[\varphi U \psi] \iff \exists \sigma \in \text{Paths}(s). \exists k \in \mathbb{N}. \sigma(k) \models \psi \wedge \forall i < k. \sigma(i) \models \varphi.$

Syntax von CTL:

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



Zusätzliche Operatoren



- Die anderen CTL-Operatoren definiert man als Abkürzungen

$$EF \varphi \quad := \quad E [\text{true} \cup \varphi]$$

$$AX \varphi \quad := \quad \neg EX \neg \varphi$$

$$AF \varphi \quad := \quad \neg EG \neg \varphi$$

$$AG \varphi \quad := \quad \neg EF \neg \varphi$$

$$A [\varphi \cup \psi] \quad := \quad \neg EG \neg \psi \wedge \neg E [\neg \psi \cup (\neg \varphi \wedge \neg \psi)]$$

$$E [\varphi \text{ W } \psi] \quad := \quad \neg A [(\varphi \wedge \neg \psi) \cup (\neg \varphi \wedge \neg \psi)]$$

$$A [\varphi \text{ W } \psi] \quad := \quad \neg E [(\varphi \wedge \neg \psi) \cup (\neg \varphi \wedge \neg \psi)]$$



Einige Tautologien

- Zwei CTL-Formeln φ und ψ heißen **semantisch äquivalent**

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

falls in jedem Zustand s jeder Kripke-Struktur gilt:

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

- Wichtige semantische Äquivalenzen sind:

$$\neg \text{AX } \varphi \quad \equiv \quad \text{EX } \neg \varphi$$

$$\neg \text{AF } \varphi \quad \equiv \quad \text{EG } \neg \varphi$$

$$\neg \text{EF } \varphi \quad \equiv \quad \text{AG } \neg \varphi$$

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

$$\text{AF } \varphi \quad \equiv \quad \text{A} [\text{true U } \varphi]$$

$$\text{EG } \varphi \quad \equiv \quad \text{E} [\varphi \text{ W false}]$$

$$\text{AG } \varphi \quad \equiv \quad \text{A} [\varphi \text{ W false}]$$



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

$$\begin{array}{ll} \text{AF } \varphi & \equiv \varphi \vee \text{AX } \text{AF } \varphi \\ \text{AG } \varphi & \equiv \varphi \wedge \text{AX } \text{AG } \varphi \\ \text{A}[\varphi \text{ U } \psi] & \equiv \psi \vee (\varphi \wedge \text{AX } \text{A}[\varphi \text{ U } \psi]) \\ \\ \text{EF } \varphi & \equiv \varphi \vee \text{EX } \text{EF } \varphi \\ \text{EG } \varphi & \equiv \varphi \wedge \text{EX } \text{EG } \varphi \\ \text{E}[\varphi \text{ U } \psi] & \equiv \psi \vee (\varphi \wedge \text{EX } \text{E}[\varphi \text{ U } \psi]) \end{array}$$



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:
 - $\neg \text{true} = \text{false}$
 - $\neg \text{false} = \text{true}$
 - $\neg \neg \phi = \phi$
 - $\neg(\phi \vee \psi) = \neg \phi \wedge \neg \psi$
 - $\neg(\phi \wedge \psi) = \neg \phi \vee \neg \psi$

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

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

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

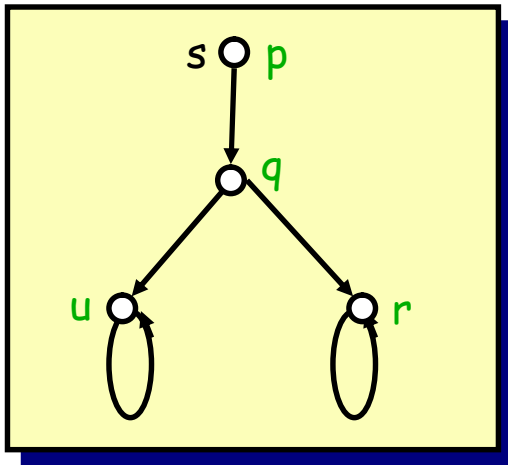


Die Punkte s und s' des folgenden Beispiels können wir mit CTL trennen:

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

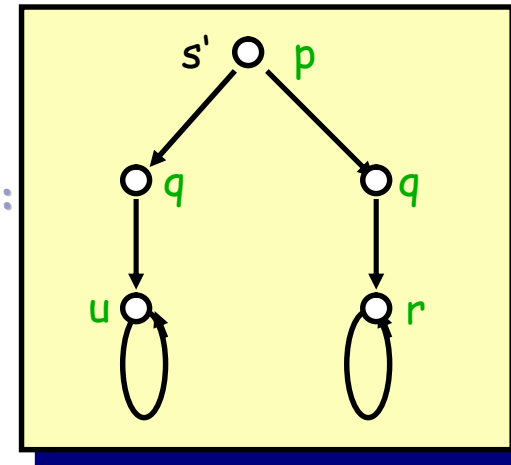
$$s \models \neg \psi$$

Q :



$$s' \models \psi$$

Q :



Mit LTL können wir s und s' nicht unterscheiden, da $\text{Traces}(s) = \text{Traces}(s')$



Bisimulation

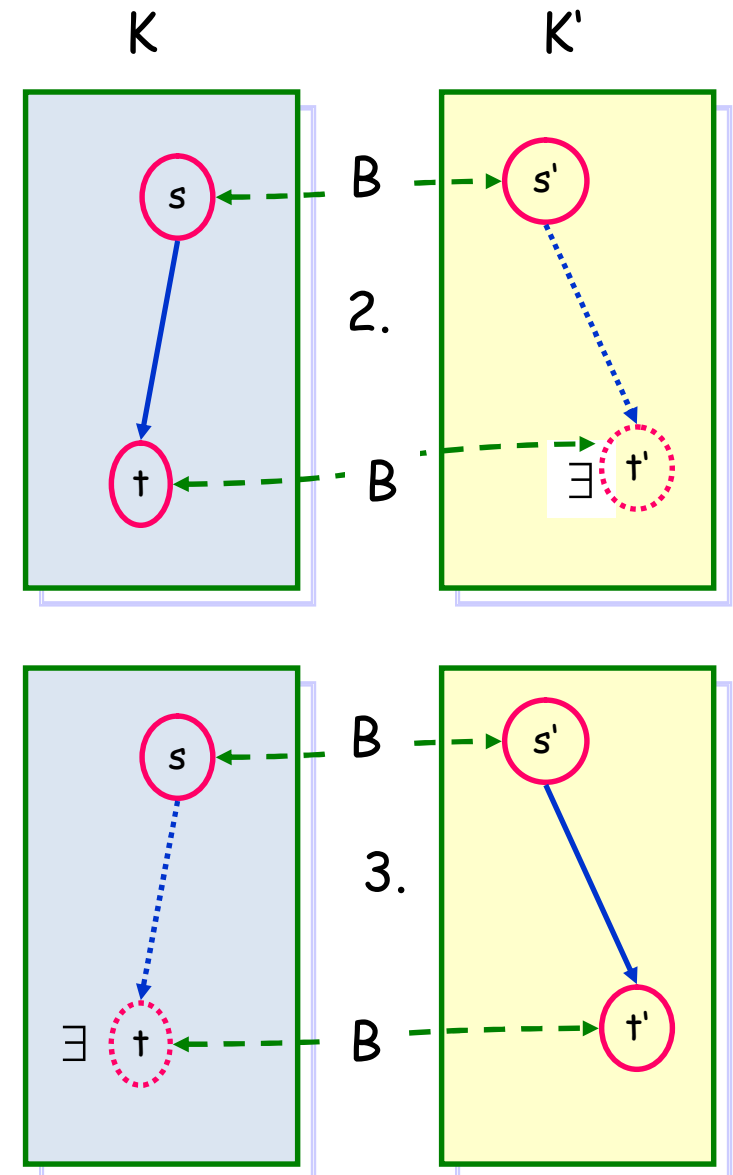
$K=(S, \rightarrow, L)$ und $K'=(S', \rightarrow', L')$ Kripke-Strukturen.

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. $\forall t' \in S'. S' \rightarrow' t' \Rightarrow \exists t \in S. s \rightarrow t \wedge t B t'$

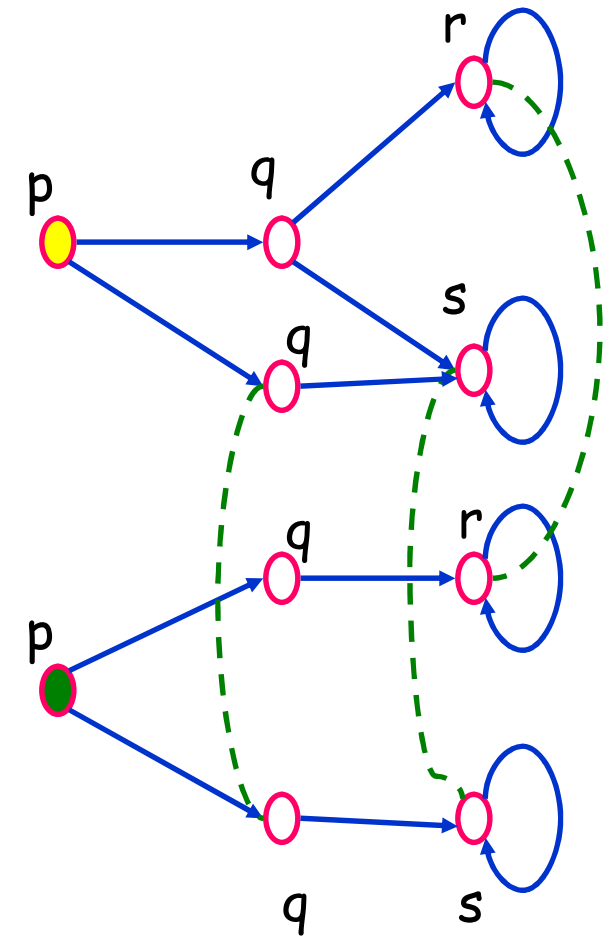
- B heißt **links-total**, falls zu jedem $s \in S$ ein $s' \in S'$ existiert mit $s B s'$.
- **Rechts-total** analog.
- **Total** := links.-total + rechts-total





Grösste Bisimulation

- Seien K und K' Kripke-Strukturen über P .
 - $\emptyset \subseteq B \times B'$ 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
größer Bisimulation

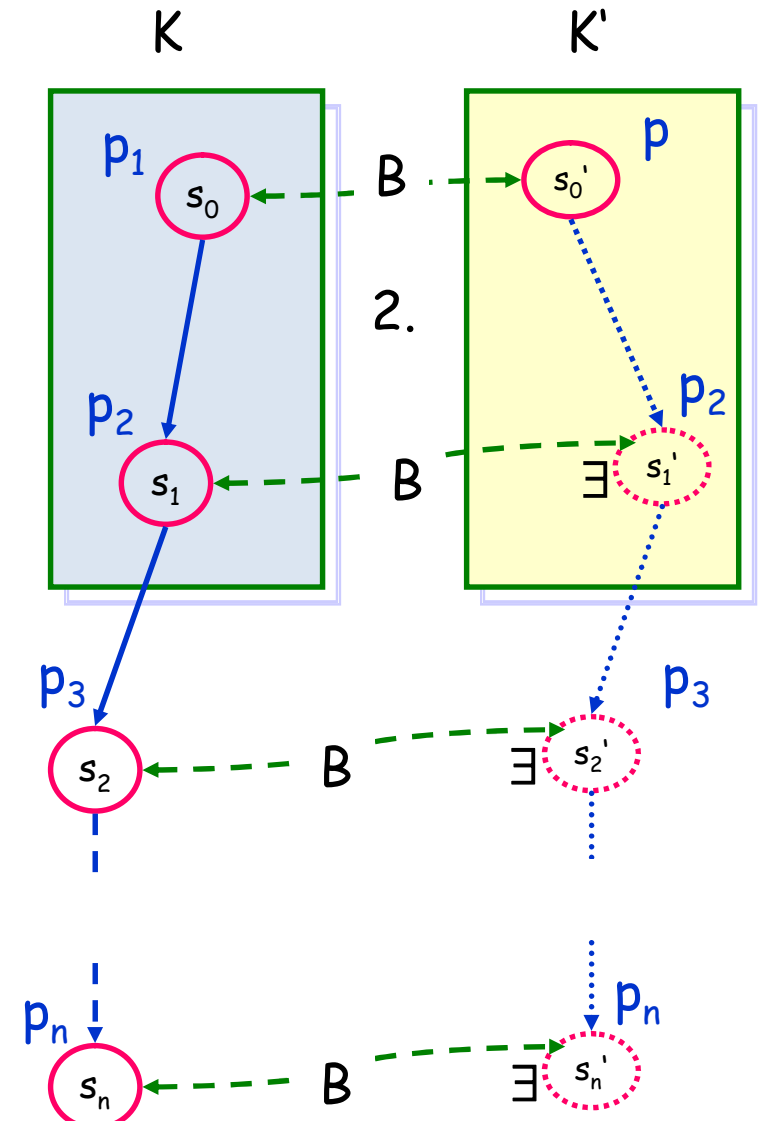


Bisimilar \Rightarrow Spur-Äquivalent

Seien s und s' in der Bisimulation B

\Rightarrow zu jedem Pfad $\sigma = (s_0, \dots, s_n, \dots)$, der in $s=s_0$ beginnt, gibt es entsprechenden Pfad $\sigma' = (s'_0, \dots, s'_n, \dots)$, der in $s' = s'_0$ beginnt, so dass $s_i B s'_i$ für jedes $i \in \mathbb{N}$

Folgerung: $\forall i \in \mathbb{N}. L(s_i) = L(s'_i)$,
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. $\psi = EX \phi$: Es gibt ein t mit $s \rightarrow t$ und $t \models \phi$. Also gibt es t' mit $s' \rightarrow t'$ und $t \sim t'$.
Nach Ind.Vorr. also $t' \models \phi$, also $s' \models EX\phi$, d.h. $s' \models \psi$.
3. $\psi = EG \phi$: Es ex. Pfad $s=s_0, s_1, \dots, s_n, \dots$ mit $s_i \rightarrow s_{i+1}$ und $s_i \models \phi$ für alle $i \in \mathbb{N}$.
Bisimulation liefert Pfad $s'=s'_0, s'_1, \dots, s'_n, \dots$ mit $s'_i \rightarrow s'_{i+1}$ und $s_i \sim s'_i$.
Ind.Vorr. liefert $s'_i \models \phi$ für jedes i , also $s'_0 \models EG\phi$, somit $s' \models \psi$.
4. $\psi = E[\phi \cup \phi]$: $\exists n \in \mathbb{N}$ und Zustandsfolge $s=s_0, s_1, \dots, s_n$ mit $s_n \models \phi$ und $s_i \models \phi$ für alle $i < n$. Bisimulation und Ind.-Vorr. liefern entsprechende Folge $s'=s'_0, s'_1, \dots, s'_n$ mit $s'_i \models \phi$ für alle $i < n$ und $s'_n \models \phi$.
Da diese zum unendlichen Pfad erweitert werden kann, folgt $s' \models E[\phi \cup \phi]$.
5. $\psi = \neg\phi$: Aus $s' \models \phi$ würde nach Ind.Vorr. $s \models \phi$ folgen. Daher $s' \models \neg\phi$.
6. $\psi = \phi_1 \wedge \phi_2$: klar.



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

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

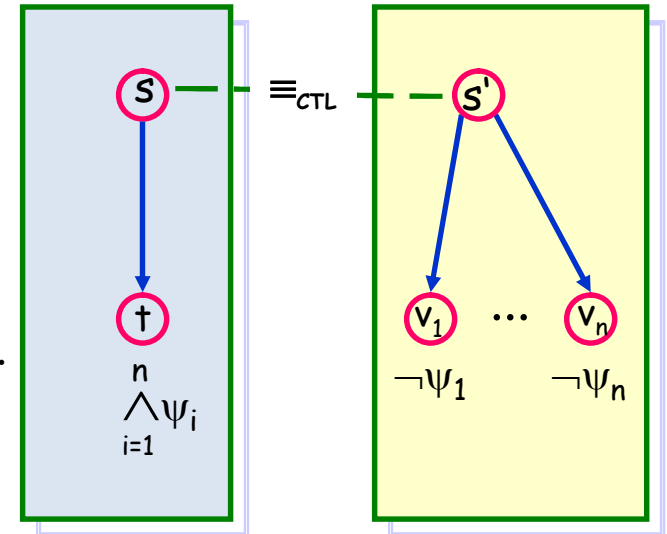
Beweis: Es reicht zu zeigen, dass \equiv_{CTL} eine Bisimulation ist.

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

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 $s \rightarrow t$ und $s \equiv_{\text{CTL}} s'$. Gesucht: Ein t' mit $s' \rightarrow t'$ und $t \equiv_{\text{CTL}} t'$. Sei $N(s) := \{v \mid s' \rightarrow v\} = \{v_1, \dots, v_n\}$.
Wenn $t \equiv_{\text{CTL}} v_i$, 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
 $s \models \text{EX } \psi_1 \wedge \psi_2 \wedge \dots \wedge \psi_n$, aber $s' \models \neg \text{EX } \psi_1 \wedge \psi_2 \wedge \dots \wedge \psi_n$,
im Widerspruch zu $s \equiv_{\text{CTL}} s'$.

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

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

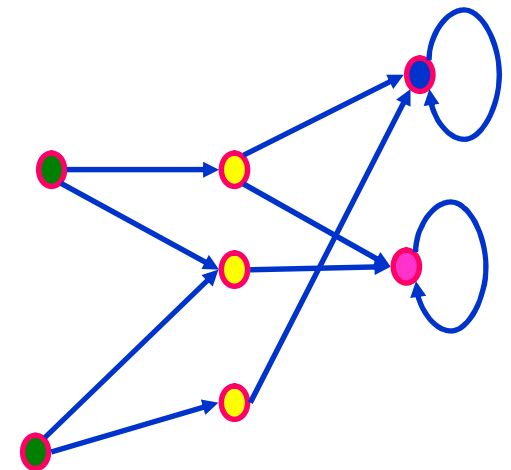
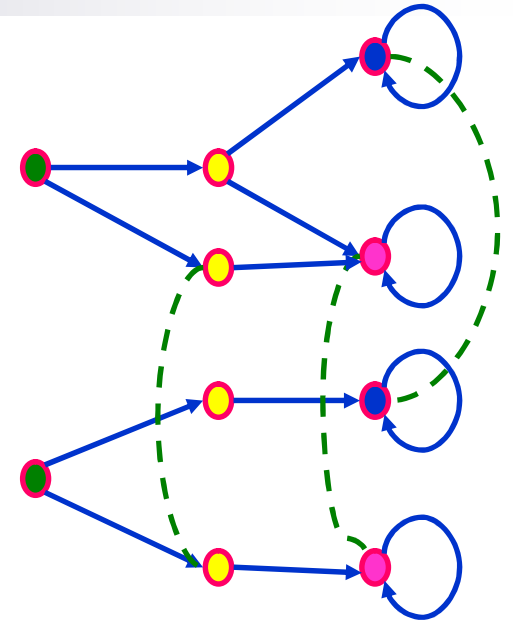


Vorsicht: Aus dem Satz kann man nicht schließen, daß jede LTL Formel auch durch eine CTL-Formel *ausdrückbar* wäre.



Zustandsminimierung

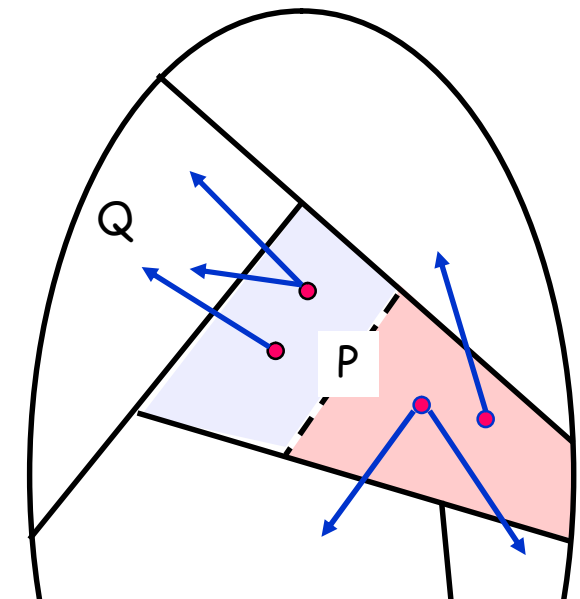
- Größte Bisimulation \sim auf einer Kripke-Struktur $K=(S,R,AP)$ ist Äquivalenzrelation
 - reflexiv,
 - symmetrisch,
 - transitiv
- Faktormenge S/\sim ist Menge aller Äquivalenzklassen:
 - $[s]_{\sim} := \{ s' \in S \mid s \sim s' \}$ -- Äquivalenzklasse von a
 - $S/\sim := \{ [s]_{\sim} \mid a \in S \}$ -- Faktormenge
- Auf Faktormenge S/\sim 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/\sim

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
```



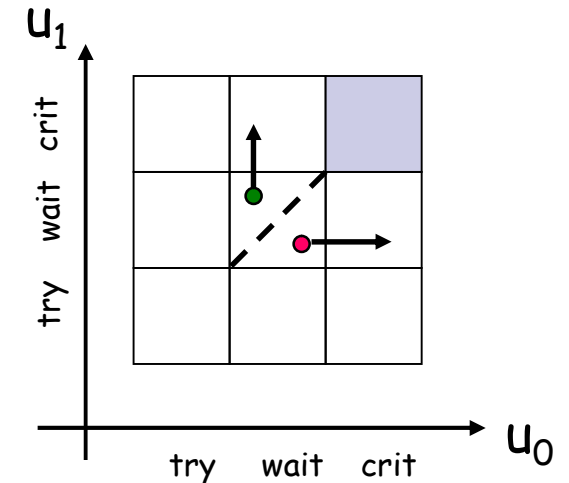
Bisimulation

■ Bakery=(S,I,→,L,AP) mit

- $S = \{(u_0.pc, u_1.pc, x_0, x_1) \in 3 \times 3 \times \mathbb{N} \times \mathbb{N}\}$
- $AP = \{u_0.trying, u_1.trying, u_0.critical, u_1.critical\}$
 - nur diese sind für die interessierenden Eigenschaften notwendig
- \rightarrow : 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:
 - $u_i.trying, u_i.critical, u_i.waitng := \neg(u_i.trying \vee u_i.critical)$
 - hoffen, dass die Klasse $(u_0.critical \wedge u_1.critical)$ 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 $(u_0.wait \wedge u_1.try)$
 - Zustände x mit $x \rightarrow t \in u_1.crit$
 - andere (Zustände y mit $y \rightarrow u_0.crit$)

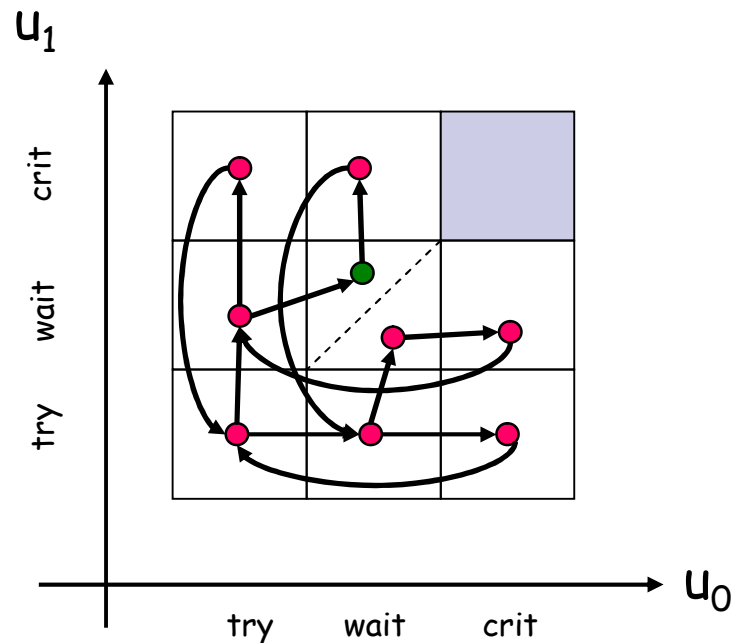


- : $(1,1,x_1 < x_0)$
- : $(1,1,x_0 < x_1)$



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:

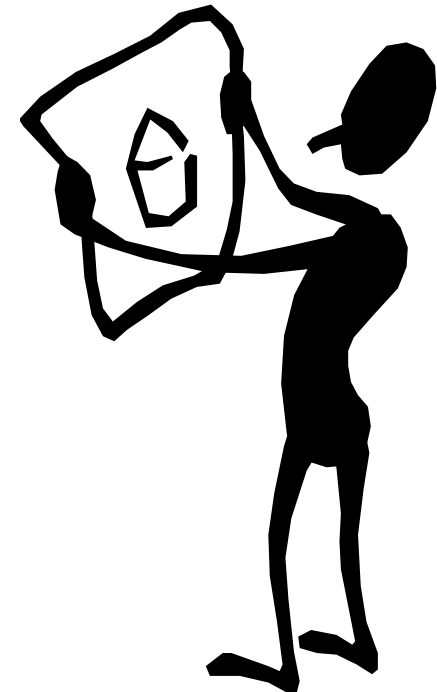
(pc_0, pc_1, x_0, x_1)

(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 die 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 φ .

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

- LTLSPEC** ψ heißt:

In jedem Anfangszustand gilt:

A ψ

```
UltraEdit-32 - [C:\Dokumente und Einstellungen\Peter\Desktop\NeueVo...
File Edit Search Project View Format Column Macro Advanced Window Help
SimplePeterson.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
$$p W q \iff (G p) \vee (p U q).$$

Universalität	p gilt immer
Global	$AG(p)$
vor r	$A[(p \vee AG(\neg r)) W r]$
nach q	$AG(q \Rightarrow AG(p))$
zwischen q und r	$AG(q \wedge \neg r \Rightarrow A[(p \vee AG(\neg r)) W r])$

Existenz	p gilt irgendwann
Global	$AF(p)$
vor r	$A[\neg r W (p \wedge \neg r)]$
nach q	$A[\neg q W (q \wedge AF(p))]$
zwischen q und r	$AG(q \wedge \neg r \Rightarrow A[\neg r W (p \wedge \neg r)])$





Spezifikationsmuster in CTL

- Wichtig sind Präzedenz und Response-Eigenschaften

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

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





Äquivalenz von CTL und LTL-Formeln

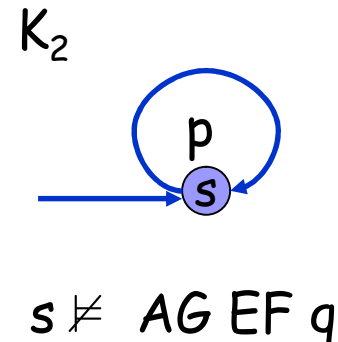
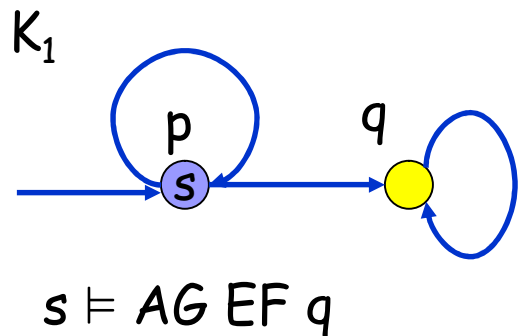
- LTL kann man nicht unmittelbar mit CTL vergleichen
 - LTL-Formeln φ gelten für Pfade
 - CTL-Formeln Ψ gelten für Zustände
- Aus einer LTL-Formel φ gewinnen wir die Formel $A\varphi$ mit der Semantik
 - $s \models A\varphi :\Leftrightarrow$ Für alle $\sigma \in \text{Paths}(s)$. $\sigma \models \varphi$
- 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 ... ?



- 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

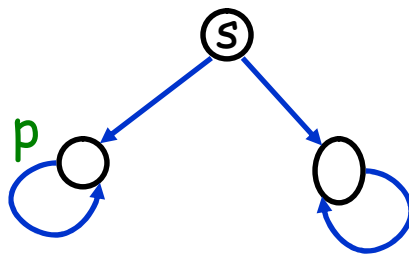
Gegeben: CTL-Formel Ψ

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



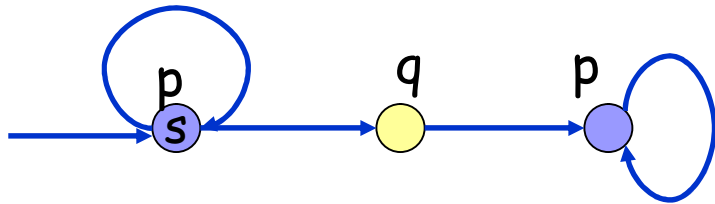
$s \models EG\ AF\ p$

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



Ausdrückbarkeit von LTL in CTL

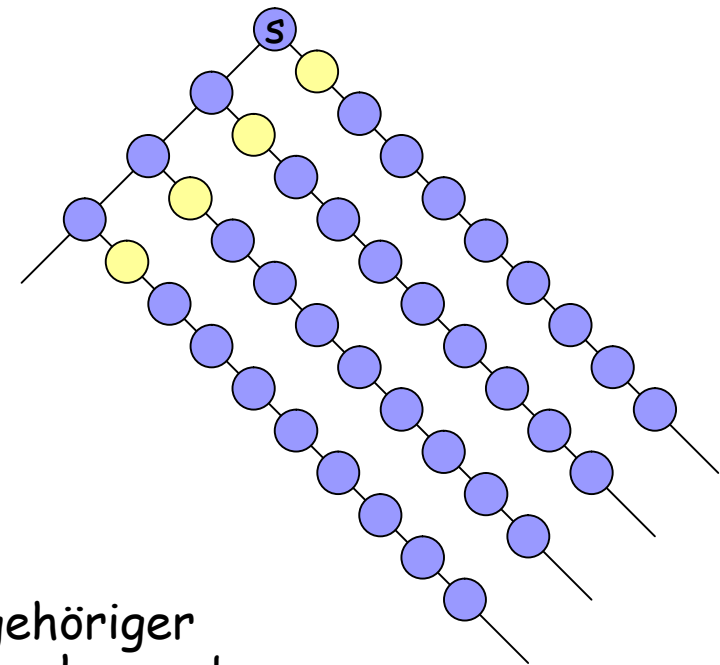
- Kann man die LTL-Formel $FG p$ durch eine äquivalente CTL-Formel ausdrücken?
- **Erster Versuch:** $AF AG p$
- $AF AG p \Rightarrow A FG p$ gilt immer. Was ist mit der Rückrichtung?



Kripke-Struktur ($L(\bullet) = p$)

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

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

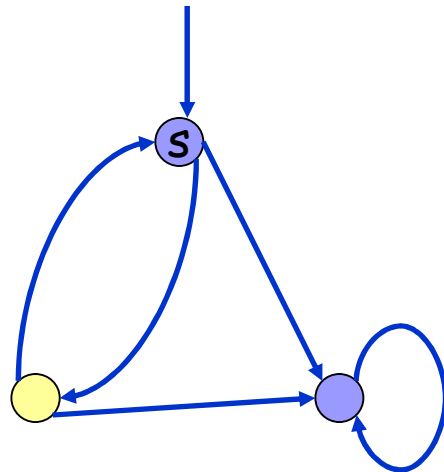


zugehöriger
Berechnungsbaum
(bisimilare Kripke-Struktur)



Ausdrückbarkeit (3)

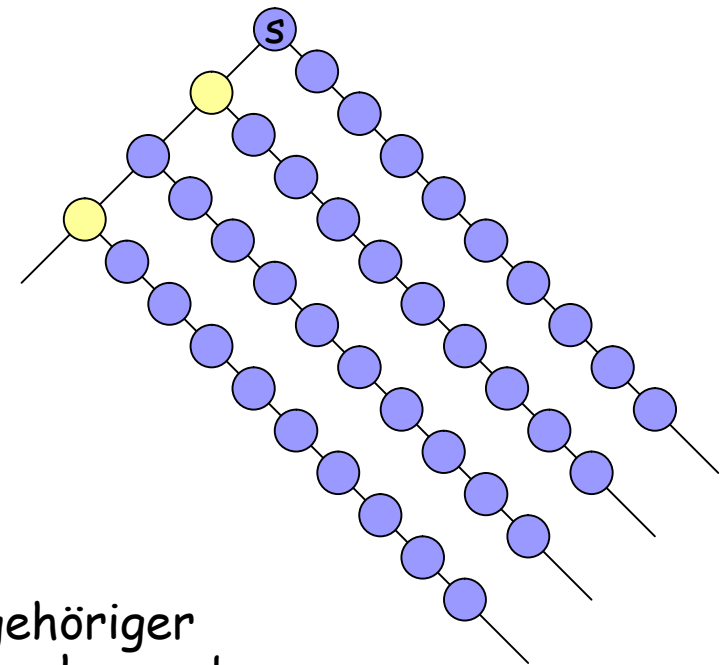
- Kann man die LTL-Formel $FG p$ durch eine äquivalente CTL-Formel ausdrücken ?
- **Zweiter Versuch:** $AF EG p$



Kripke-Struktur ($L(\bullet) = p$)

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

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



zugehöriger
Berechnungsbaum
(bisimilare Kripke-Struktur)



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 Aussagen durch wechselseitige Rekursion :

Zustandseigenschaften

```
SProp ::= P
        | A PProp
        | E PProp
        | SProp  $\wedge$  SProp
        |  $\neg$  SProp
```

Pfadeigenschaften

```
PProp ::= init SProp
        | X PProp
        | PProp  $\cup$  PProp
        | PProp  $\wedge$  PProp
        |  $\neg$  PProp
```

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 $\sigma=(s_0,\dots,s_n,\dots)$ sowie für $p \in P$, $\varphi, \psi \in PProp$ definieren wir :

CTL* Syntax :

$PProp$ | $init$ $SProp$
| X $PProp$
| $PProp \cup PProp$
| $PProp \wedge PProp$
| \neg $PProp$

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

Definition der CTL* Semantik :

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

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

$\sigma \models \varphi \cup \psi$ gdw. $\exists k \in \text{Nat. } \sigma^k \models \psi \wedge \forall i < k. \sigma^i \models \varphi.$

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

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

$s \models E \varphi$ 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.

$$A [GFp \Rightarrow Fq]$$

Dennoch kann CTL mehr Zustände trennen als LTL.

In CTL*, aber nicht ausdrückbar in $(CTL \cup LTL)$:

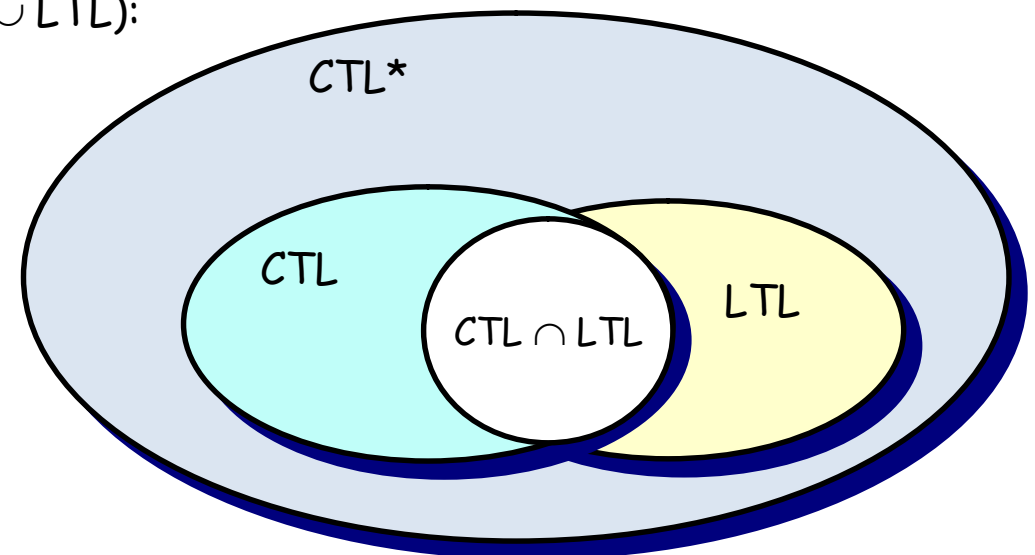
$$E[GF p]$$

In CTL aber nicht ausdrückbar in LTL:

$$AG EF p$$

In LTL aber nicht ausdrückbar in CTL:

$$A [GFp \Rightarrow Fq]$$





Aufgaben

- Definieren Sie EF, EW, AX, AF, AG, AU und AW analog wie EX, EG und EU und zeigen Sie die Äquivalenzen
$$\begin{aligned}EF \varphi &\equiv E [\text{true} \cup \varphi] \\AX \varphi &\equiv \neg EX \neg \varphi \\AF \varphi &\equiv \neg EG \neg \varphi \\AG \varphi &\equiv \neg EF \neg \varphi \\A [\varphi \cup \psi] &\equiv \neg EG \neg \psi \wedge \neg E [(\varphi \wedge \neg \psi) \cup (\neg \varphi \wedge \neg \psi)] \\E [\varphi \cup \psi] &\equiv \neg A [(\varphi \wedge \neg \psi) \cup (\neg \varphi \wedge \neg \psi)] \\A [\varphi \cup \psi] &\equiv \neg A [(\varphi \wedge \neg \psi) \cup (\neg \varphi \wedge \neg \psi)]\end{aligned}$$
- Zeigen Sie: In LTL gilt $FX p \equiv_{LTL} XFp$, in CTL gilt jedoch **nicht**: $AF AX p \equiv_{CTL} AX AF p$.
- Zeigen Sie, dass $A(\varphi \cup AX\psi) \Rightarrow A(\varphi \cup X\psi)$ gilt, nicht aber die Umkehrung.
- Ist $K=(S,R,AP)$ Kripke Struktur mit größter Bisimulation
 - Zeigen Sie: \sim ist eine Äquivalenzrelation
 - Zeigen Sie, dass für die minimale Kripke-Struktur K_{\sim} gilt:
 - $B = (s, [s]_{\sim})$ ist eine totale Bisimulation
 - B ist die grösste Bisimulation zwischen K und K_{\sim} .
- 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 $AP = \{u_0.\text{trying}, u_0.\text{critical}, u_1.\text{trying}, u_1.\text{critical}\}$ 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
```