Formale Grundlagen der Informatik 3

Temporale Logik CTL, Explizites Model Checking

Prof. Stefan Katzenbeisser
Security Engineering Group
Technische Universität Darmstadt

skatzenbeisser@acm.org
http://www.seceng.informatik.tu-darmstadt.de

Model Checking:
Turing Award 2007

E. Clarke, A. Emerson, J. Sifakis

```
byte n = 0;
active proctype P() {
  n = 1;
}
active proctype Q() {
  n = 2;
}
```

Programm / „Modell"

```
„P und Q sind nie gleichzeitig
in einem kritischen Abschnitt"
```

Eigenschaft / Spezifikation

Model Checker

Gegenbeispiel

Model Checking
Modell und Spezifikation

```
byte n = 0;
active proctype P() {
  n = 1;
}
active proctype Q() {
  n = 2;
}
```

Programm / „Modell“

„P und Q sind nie gleichzeitig in einem kritischen Abschnitt“

Eigenschaft / Spezifikation

Model Checker

✔

Gegenbeispiel

Modell:

- Abstrakte Repräsentation des Systems
- Formalisiert durch eine Kripke-Struktur

Spezifikation:

- Eigenschaft, die das System erfüllen soll
- Meist formalisiert durch eine logische Formel

Wiederholung: Kripke Strukturen

Angelehnt an endliche Automaten, aber ...

- Modellierung **reaktiver Systeme**, die keinen „Endzustand" kennen
- Relevant sind primär Zustandsänderungen, nicht Eingaben
- Zustände werden mit **„atomaren" Eigenschaften** versehen

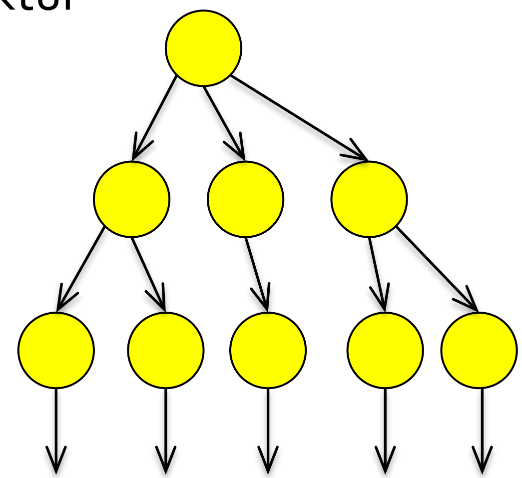
Eine Kripke-Struktur $\mathcal{M} = (S, I, R, L)$ über einer Menge P besteht aus

- einer endlichen Menge an Zuständen S und Anfangszuständen $I \subseteq S$
- einer Übergangsrelation $R \subseteq S \times S$
- und einer Abbildung $L : S \to 2^P$

Eine Kripke-Struktur nennt man **total**, wenn es für jeden Zustand $s \in S$ einen Zustand $s' \in S$ gibt mit $(s, s') \in R$

Repräsentation der Spezifikation: Temporale Logik CTL

- Ausgangspunkt: Computation Tree einer Kripke-Struktur
- CTL = „Computation Tree Logic"
- Zeitlogik erlaubt Aussagen über das **temporale Verhalten** eines Systems



- Erweitert Aussagenlogik um Pfad- und Zustandsquantoren
- Formeln werden immer **bezüglich einer Kripke-Struktur** und einem **Anfangszustand** ausgewertet
- Wir schreiben: $\mathcal{M}, s \models \varphi$
  wenn Formel φ im Zustand s der Kripke-Struktur $\mathcal{M}$ erfüllt ist.

Temporale Logik CTL

Pfadquantoren: betreffen **Pfade ab einem bestimmten Zustand**

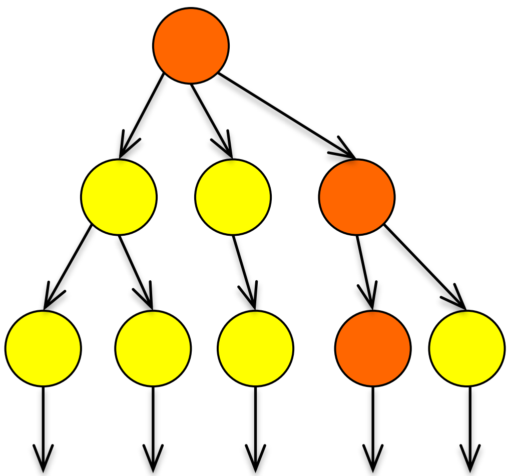
- **A** … „für jeden Pfad gilt" (**A**LL)
- **E** … „es gibt einen Pfad auf dem gilt" (**E**XISTS)

Zustandsquantoren: betreffen **einen bestimmten Pfad**

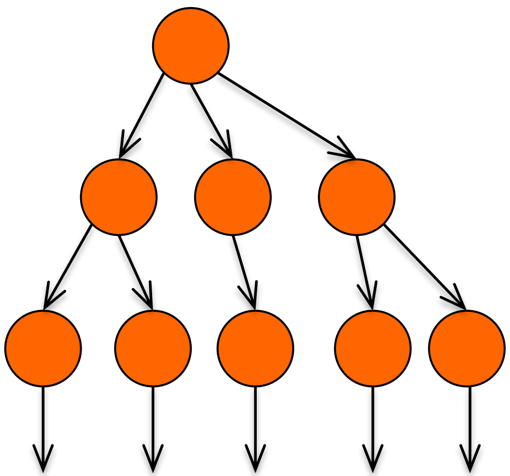
- **X** … „im nächsten Zustand gilt" (NE**X**T)
- **F** … „in der Zukunft gilt irgendwann" (**F**UTURE)
- **G** … „in der Zukunft gilt immer" (**G**LOBAL)
- **U** … „es gilt eine Eigenschaft bis eine andere gilt" (**U**NTIL)

In CTL dürfen Pfad- und Zustandsquantoren nur paarweise auftreten:
AX,AF,AG,AU,EX,EF,EG, EU

Temporale Logik CTL
Pfadquantoren

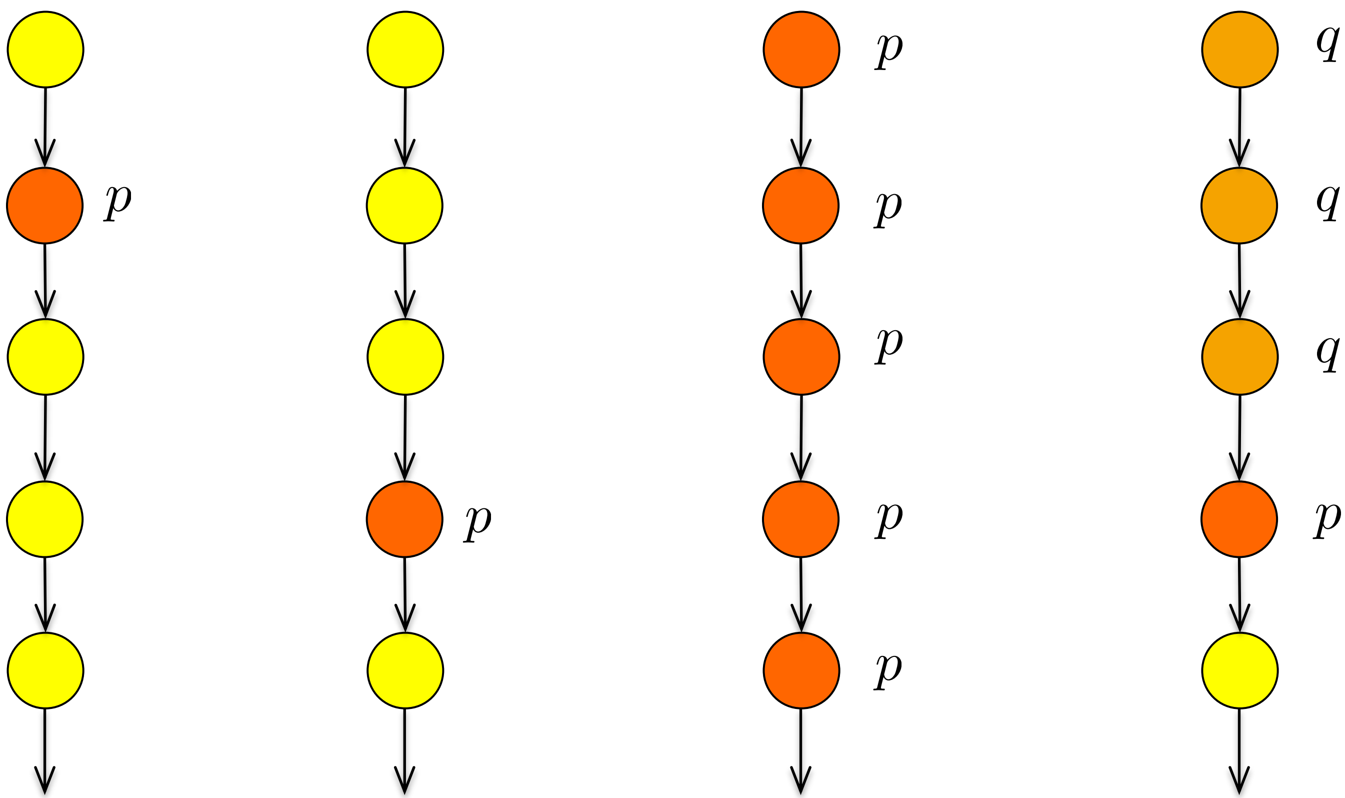


E: „es gibt einen Pfad auf dem gilt" (**E**XISTS)



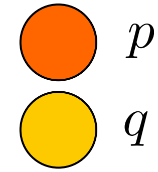
A: „für jeden Pfad gilt" (**A**LL)

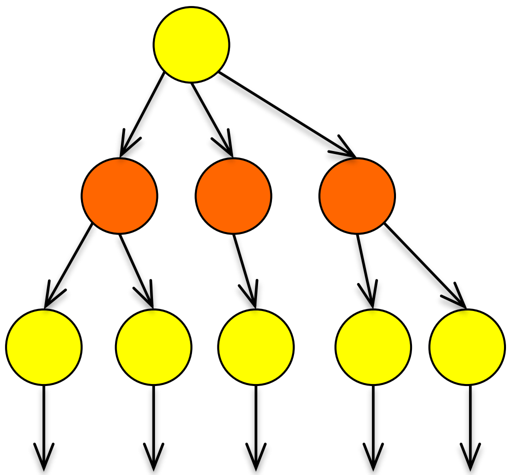
Temporale Logik CTL
Zustandsquantoren



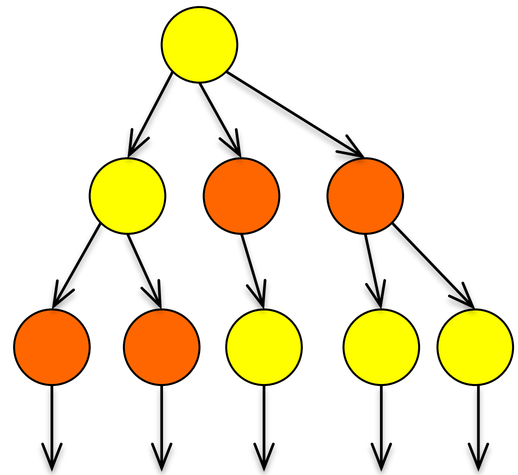
$\mathbf{X}p$ $\mathbf{F}p$ $\mathbf{G}p$ $q\mathbf{U}p$

Temporale Logik CTL
Beispiele (1)

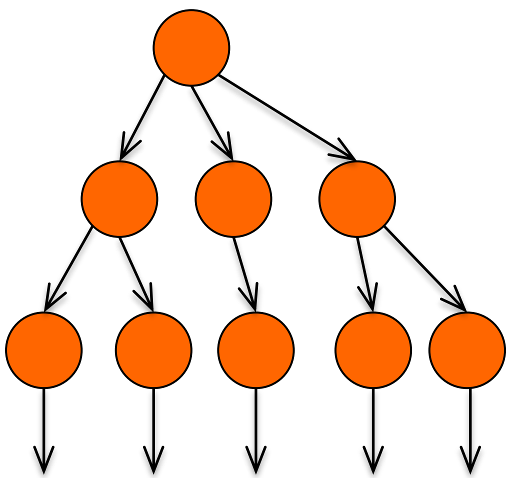




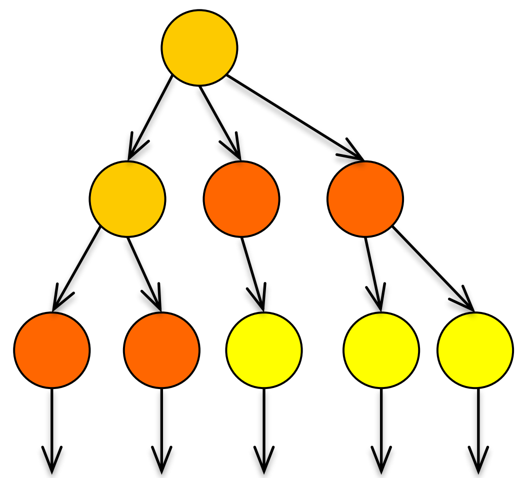
$\mathbf{AX}p$



$\mathbf{AF}p$

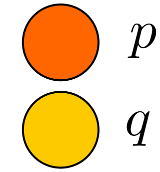


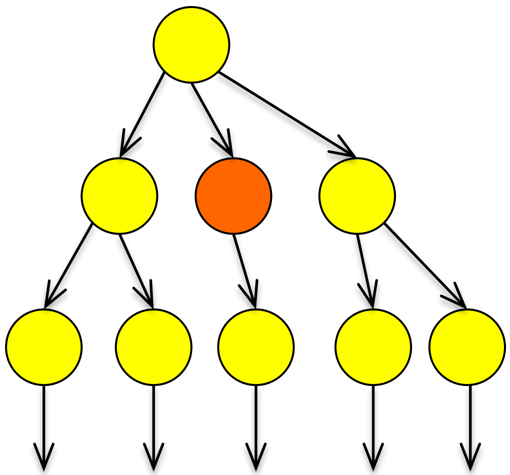
$\mathbf{AG}p$



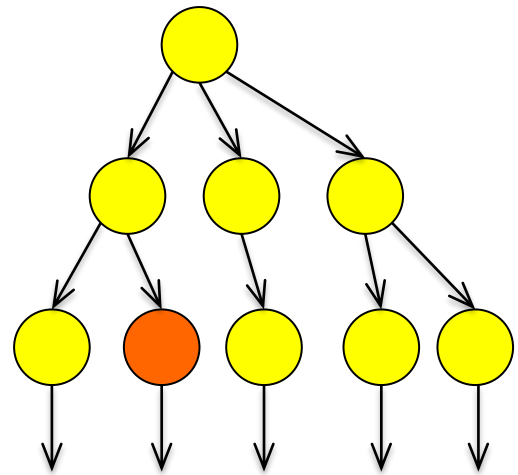
$\mathbf{A}q\mathbf{U}p$

Temporale Logik CTL
Beispiele (2)

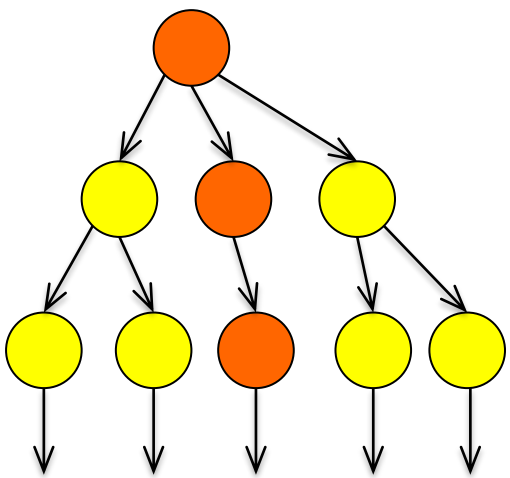




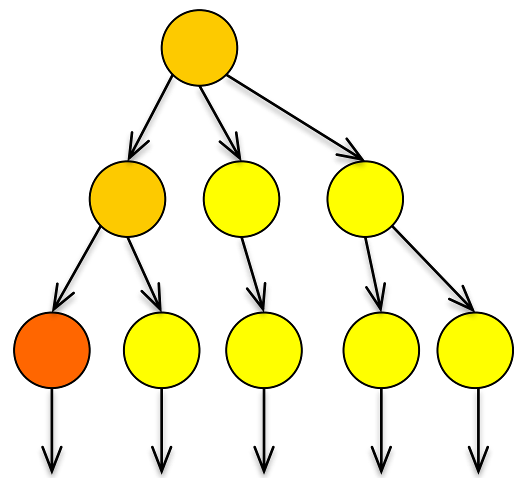
$\mathbf{EX}p$



$\mathbf{EF}p$



$\mathbf{EG}p$



$\mathbf{E}q\mathbf{U}p$

Temporale Logik CTL
Syntax

Syntax:

Die Menge der CTL-Formeln ist die kleinste Menge für die gilt:

- Atomare Eigenschaften $p \in P$ sowie die Konstanten $\top, \bot$ sind CTL-Formeln.
- Sind φ und ψ CTL-Formeln, dann sind auch
  $\neg\varphi,\ \varphi \vee \psi,\ \varphi \wedge \psi,\ \varphi \rightarrow \psi,$
  $\mathbf{AX}\varphi,\ \mathbf{EX}\varphi,\ \mathbf{A}\varphi\mathbf{U}\psi,\ \mathbf{E}\varphi\mathbf{U}\psi,\ \mathbf{AG}\varphi,\ \mathbf{EG}\varphi,\ \mathbf{AF}\varphi,\ \mathbf{EF}\varphi$
  CTL-Formeln.

Beispiele:

$\mathbf{EG}r$

$\mathbf{A}r\mathbf{U}q$

$\mathbf{AGAF}p$

$\mathbf{EF}(\mathbf{EG}p \rightarrow \mathbf{AF}r)$

Keine CTL-Formeln:

$\mathbf{FG}r$

$\mathbf{A}\neg\mathbf{G}p$

$\mathbf{EF}(r\mathbf{U}q)$

$\mathbf{AEF}r$

Temporale Logik CTL
Semantik (1)

- CTL-Formeln werden über Kripke-Strukturen $\mathcal{M} = (S, I, R, L)$ ausgewertet.
- Wir schreiben: $\mathcal{M}, s \models \varphi$
  wenn Formel φ im Zustand s der Kripke-Struktur $\mathcal{M}$ erfüllt ist.

Semantik, Teil 1:

- $\mathcal{M}, s \models \top,\ \mathcal{M}, s \not\models \bot$ für alle Zustände $s \in S$ (Konstanten)
- $\mathcal{M}, s \models p$ falls $p \in L(s)$ (Atomare Aussagen)
- $\mathcal{M}, s \models \neg\varphi$ falls $\mathcal{M}, s \not\models \varphi$ (Negation)
- $\mathcal{M}, s \models \varphi \wedge \psi$ falls $\mathcal{M}, s \models \varphi$ und $\mathcal{M}, s \models \psi$ (Konjunktion)
- $\mathcal{M}, s \models \varphi \vee \psi$ falls $\mathcal{M}, s \models \varphi$ oder $\mathcal{M}, s \models \psi$ (Disjunktion)
- $\mathcal{M}, s \models \varphi \rightarrow \psi$ falls $\mathcal{M}, s \not\models \varphi$ oder $\mathcal{M}, s \models \psi$ (Implikation)

Temporale Logik CTL
Semantik (2)

- Für eine Kripke-Struktur $\mathcal{M} = (S, I, R, L)$ schreiben wir $s \to s'$ falls $(s, s') \in R$
- Ein Pfad $\pi = s_0 s_1 s_2 s_3 \ldots$ ist eine Sequenz $s_0 \to s_1 \to s_2 \to s_3 \to \ldots$

Semantik, Teil 2:

- $\mathcal{M}, s \models \mathbf{AX}\varphi$ falls für **alle** Zustände s' mit $s \to s'$ gilt: $\mathcal{M}, s' \models \varphi$
- $\mathcal{M}, s \models \mathbf{EX}\varphi$ falls **ein** Zustand s' mit $s \to s'$ **existiert**, sodass: $\mathcal{M}, s' \models \varphi$

- $\mathcal{M}, s \models \mathbf{AG}\varphi$ falls für **alle** Pfade $\pi = s_0 s_1 s_2 s_3 \ldots$ mit $s_0 = s$ gilt: $\mathcal{M}, s_i \models \varphi$ für alle $i = 0, 1, 2, \ldots$
- $\mathcal{M}, s \models \mathbf{EG}\varphi$ falls **ein** Pfad $\pi = s_0 s_1 s_2 s_3 \ldots$ mit $s_0 = s$ **existiert**, sodass: $\mathcal{M}, s_i \models \varphi$ für alle $i = 0, 1, 2, \ldots$

Temporale Logik CTL
Semantik (3)

Semantik, Teil 3:

- $\mathcal{M}, s \models \mathbf{AF}\varphi$ falls für **alle** Pfade $\pi = s_0 s_1 s_2 s_3 \ldots$ mit $s_0 = s$ **ein** Zustand s_i mit $i \in \{0, 1, 2, \ldots\}$ existiert, sodass $\mathcal{M}, s_i \models \varphi$
- $\mathcal{M}, s \models \mathbf{EF}\varphi$ falls **ein** Pfad $\pi = s_0 s_1 s_2 s_3 \ldots$ mit $s_0 = s$ und **ein** Zustand s_i auf dem Pfad **existiert**, sodass $\mathcal{M}, s_i \models \varphi$

- $\mathcal{M}, s \models \mathbf{A}\varphi\mathbf{U}\psi$ falls für **alle** Pfade $\pi = s_0 s_1 s_2 s_3 \ldots$ mit $s_0 = s$ **ein** Zustand s_i mit $i \in \{0, 1, 2, \ldots\}$ existiert, sodass $\mathcal{M}, s_i \models \psi$ und für **alle** Zustände s_j mit $0 \leq j < i$ gilt: $\mathcal{M}, s_j \models \varphi$
- $\mathcal{M}, s \models \mathbf{E}\varphi\mathbf{U}\psi$ falls ein Pfad $\pi = s_0 s_1 s_2 s_3 \ldots$ mit $s_0 = s$ **existiert**, auf dem es **einen** Zustand s_i mit $i \in \{0, 1, 2, \ldots\}$ gibt, sodass $\mathcal{M}, s_i \models \psi$ und für **alle** Zustände s_j mit $0 \leq j < i$ gilt: $\mathcal{M}, s_j \models \varphi$

Modellierung in CTL
Beispiele

CTL eignet sich zur Modellierung temporaler Eigenschaften von Systemen:

- Ein Zustand ist erreichbar, in dem „ready" gilt, aber nicht „busy"

  $$\mathbf{EF}(ready \wedge \neg busy)$$

- Auf jeden Request folgt immer irgendwann ein acknowledgement:

  $$\mathbf{AG}(requested \rightarrow \mathbf{AF}\, ack)$$

- Ein Prozess befindet sich unendlich oft im Zustand „idle"

  $$\mathbf{AG}\ \mathbf{AF}\ idle$$

- Aus jedem Zustand kann ein Zustand mit Parameter „restart" erreicht werden

  $$\mathbf{AG}\ \mathbf{EF}\ restart$$

- Jeder Prozess wird irgendwann permanent mit einem Deadlock enden

  $$\mathbf{AF}\ \mathbf{AG}\ deadlock$$

CTL
Erfüllbarkeit, Gültigkeit

- Eine CTL-Formel φ ist **erfüllbar**, wenn es **eine** Kripke-Struktur $\mathcal{M}$ und **einen** Anfangszustand s gibt mit $\mathcal{M}, s \models \varphi$
- Eine CTL- Formel φ ist **gültig**, wenn für **alle** Kripke-Strukturen $\mathcal{M}$ und Anfangszustände s gilt $\mathcal{M}, s \models \varphi$

Beispiele: Sind die folgenden CTL-Formeln erfüllbar und/oder gültig?

$$\mathbf{AGAF}p$$

$$\mathbf{AG}(\bot \to \mathbf{EF}p)$$

$$\mathbf{AG}(\neg p) \wedge \mathbf{EF}p$$

Eigenschaften von CTL

- CTL-Formeln können mit den folgenden Identitäten transformiert werden:

$$
\begin{aligned}
&\mathbf{AX}\varphi = \neg\mathbf{EX}(\neg\varphi)\\
&\mathbf{AG}\varphi = \neg\mathbf{EF}(\neg\varphi)\\
&\mathbf{EF}\varphi = \cancel{\neg}(\mathbf{E}\top\mathbf{U}\varphi)\\
&\mathbf{EG}\varphi = \neg\mathbf{AF}(\neg\varphi)\\
&\mathbf{A}(\varphi\mathbf{U}\psi) = \neg((\mathbf{E}\neg\psi\mathbf{U}(\neg\varphi \wedge \neg\psi)) \vee \mathbf{EG}\neg\psi)
\end{aligned}
$$

- Alle Quantoren können daher auf die folgenden Paare zurückgeführt werden: **EX,AF,EU**
- Andere minimale Quantorenmengen existieren ebenfalls (z.B. **EG,EU,EX**)

Model Checking CTL (1)

Model-Checking Problem für CTL

Eingabe: Kripke-Struktur $\mathcal{M} = (S, I, R, L)$ und CTL Formel φ

Ausgabe: alle Zustände $s \in S$ mit $\mathcal{M}, s \models \varphi$

- „Markierungsalgorithmus“: markiere alle Zustände in denen Formel gilt
- Ist der Anfangszustand markiert: Formel ist auf der Kripke-Struktur erfüllt
- Effiziente Algorithmen zur Lösung des Problems existieren

$$\mathbf{AG}(requested \to \mathbf{AF}\,ack)$$

1. Iteration über alle Teilformeln

2. Markierung aller Zustände mit dort gültigen Teilformeln (Farben)

Model Checking CTL (2)

function SAT(φ **)** // markiere alle Zustände in denen φ gilt

begin

case // Rekursion über die Struktur von φ

φ **is** $\top$ **:** markiere alle Zustände S mit φ

φ **is** $\bot$ **:** markiere keine Zustände

φ **is** $\neg\psi$ **:** SAT(ψ) ;
markiere alle Zustände mit φ die nicht mit ψ markiert sind

φ **is** $\varphi_1 \vee \varphi_2$ **:** SAT(φ_1); SAT(φ_2);
markiere alle Zustände mit φ die mit φ_1 **oder** φ_2 markiert sind

φ **is** $\varphi_1 \wedge \varphi_2$ **:** SAT(φ_1); SAT(φ_2);
markiere alle Zustände mit φ die mit φ_1 **und** φ_2 markiert sind

φ **is** $\varphi_1 \rightarrow \varphi_2$ **:** SAT($\neg\varphi_1$); SAT(φ_2);
markiere alle Zustände mit φ die mit $\neg\varphi_1$ oder φ_2 markiert sind

Model Checking CTL (3)

function SAT(φ) , continued

...

φ **is** $\mathbf{AX}\psi$ **:** SAT($\neg\mathbf{EX}\neg\psi$) // Regeln zum Vereinfachen der

φ **is** $\mathbf{EF}\psi$ **:** SAT($\mathbf{E}\top\mathbf{U}\psi$) // CTL-Formeln

φ **is** $\mathbf{EG}\psi$ **:** SAT($\neg\mathbf{AF}\neg\psi$)

φ **is** $\mathbf{AG}\psi$ **:** SAT($\neg\mathbf{EF}\neg\psi$)

φ **is** $\mathbf{A}\varphi_1\mathbf{U}\varphi_2$ **:** SAT($\neg((\mathbf{E}\neg\varphi_2\mathbf{U}(\neg\varphi_1 \wedge \neg\varphi_2)) \vee \mathbf{EG}\neg\varphi_2)$)

φ **is** $\mathbf{EX}\psi$ **:** SAT_{EX}(ψ) // spezielle Routinen für **EX,EU,AF**

φ **is** $\mathbf{E}\varphi_1\mathbf{U}\varphi_2$ **:** SAT_{EU}(φ_1, φ_2)

φ **is** $\mathbf{AF}\psi$ **:** SAT_{AF}(ψ)

end

Model Checking CTL (4)
EX

function SAT$_{EX}$(φ) // markiere alle Zustände in denen $\varphi = \mathbf{EX}\psi$ gilt

begin

SAT(ψ)

markiere alle Zustände $s \in S$ mit φ für die es einen Zustand s_0 gibt
mit $s \to s_0$ und der bereits mit ψ markiert ist

end

Idee:

- Markiere zuerst alle Zustände in denen ψ gilt
- In allen „Vorgängern" muss dann natürlich $\varphi = \mathbf{EX}\psi$ gelten!

Model Checking CTL (5)
AF

function SAT$_{AF}$(φ) // markiere alle Zustände in denen $\varphi = \mathbf{AF}\psi$ gilt

begin

SAT(ψ); Y ist die Menge aller Zustände die mit ψ markiert wurden; $X = S$;

repeat until $X = Y$

$X := Y$; füge zu Y Zustände $s \in S$ hinzu für die für **alle** Zustände s_0 mit $s \to s_0$ gilt $s_0 \in Y$

end repeat

markiere alle Zustände in Y mit φ

end

Idee:

- Markiere zuerst alle Zustände in denen ψ gilt
- Füge jeweils Knoten hinzu bei denen **alle** Nachfolger markiert sind
- Fixpunktiteration: Terminiere den Prozess wenn keine neue Markierung gefunden

Model Checking CTL (6)
EU

function $\mathbf{SAT}_{\mathbf{EU}}$**(** φ **)** // markiere alle Zustände in denen $\varphi = \mathbf{E}\varphi_1\mathbf{U}\varphi_2$ gilt

begin

SAT(φ_1); SAT(φ_2);

Y ist die Menge aller Zustände die mit φ_2 markiert wurden; $X = S$;

repeat until $X = Y$

$X := Y$; Füge zu Y Zustände $s \in S$ hinzu die mit φ_1 markiert sind und für die es einen Zustand s_0 gibt mit $s \to s_0$ und $s_0 \in Y$

end repeat

markiere alle Zustände in Y mit φ

end

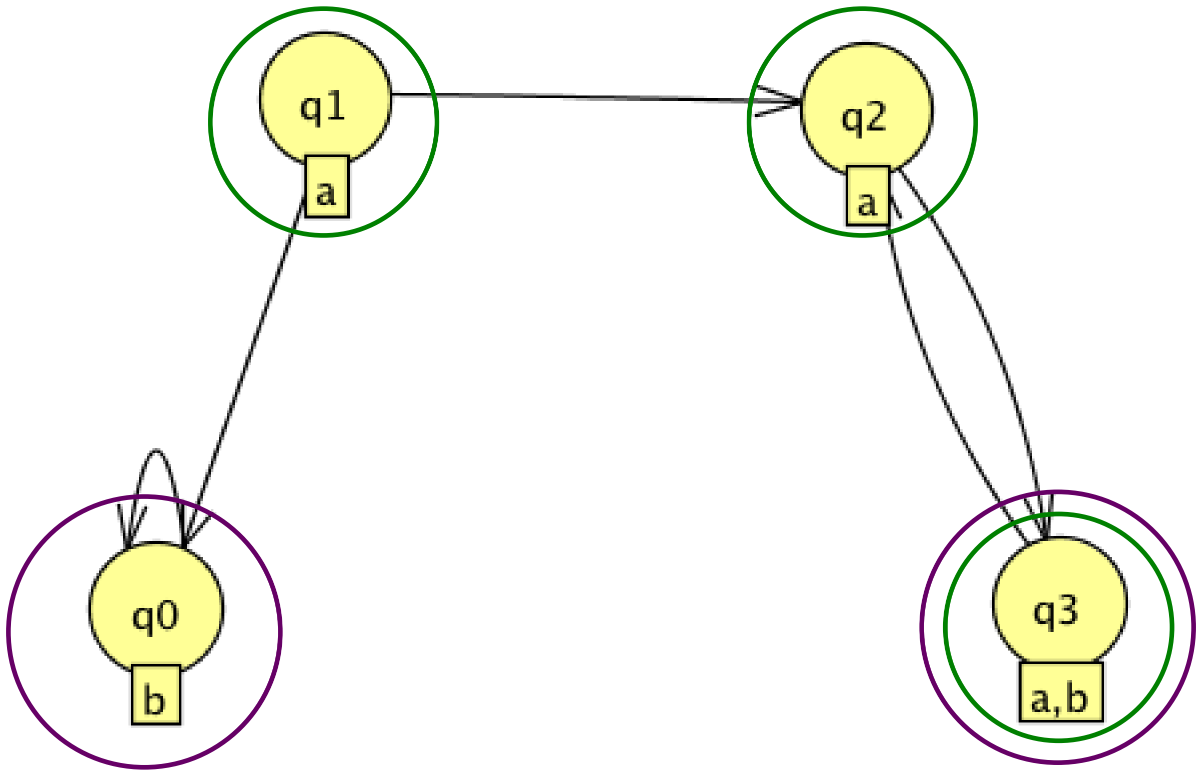
Idee:

- Markiere zuerst alle Zustände mit φ in denen φ_2 gilt
- Fixpunktiteration: Markiere alle Vorgänger wenn sie mit φ_1 markiert sind

Model Checking CTL
Beispiel (1)

$\mathbf{EG}a \wedge \mathbf{AF}b$

1. Schritt: grüne & violette Markierung

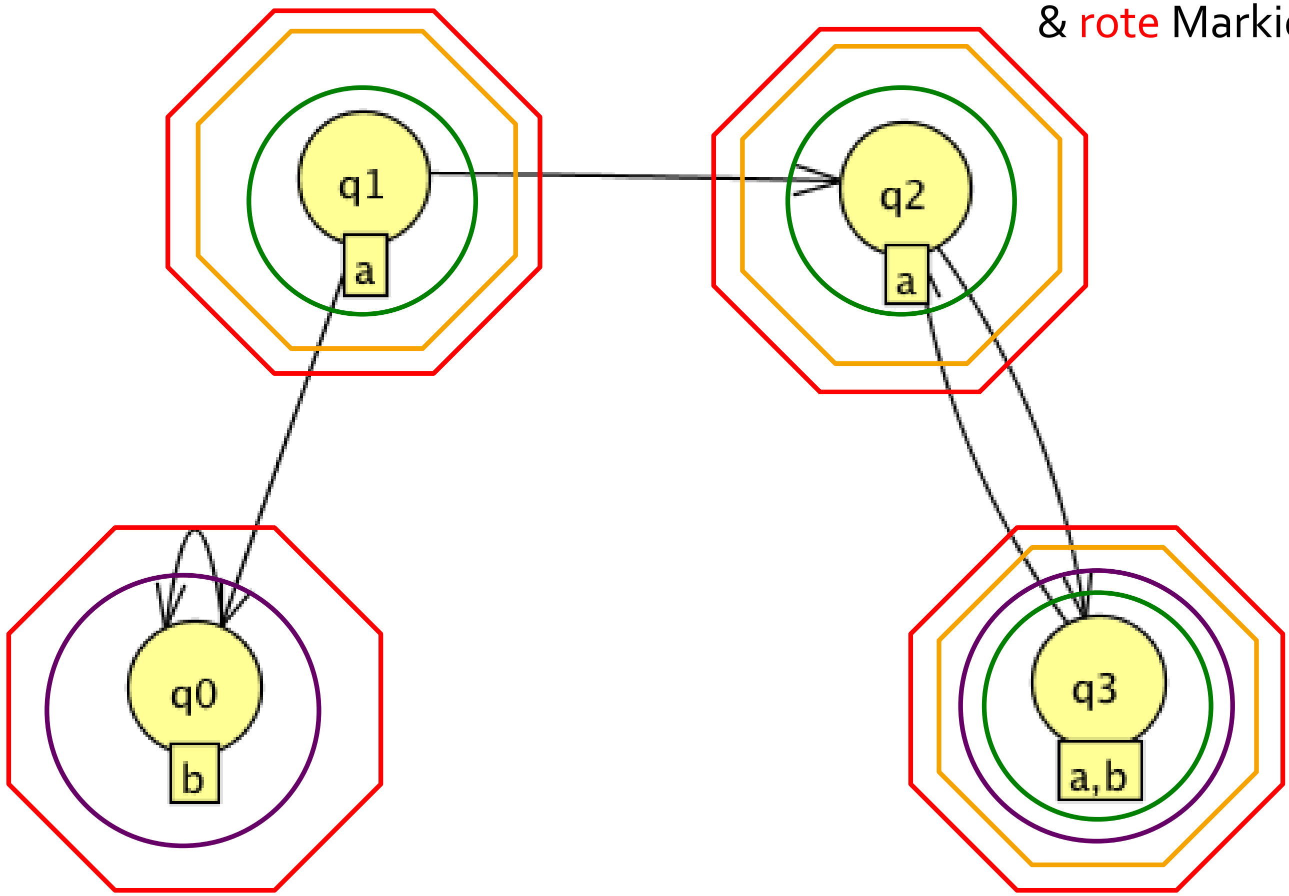


Model Checking CTL
Beispiel (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\mathbf{EG}a \wedge \mathbf{AF}b$$

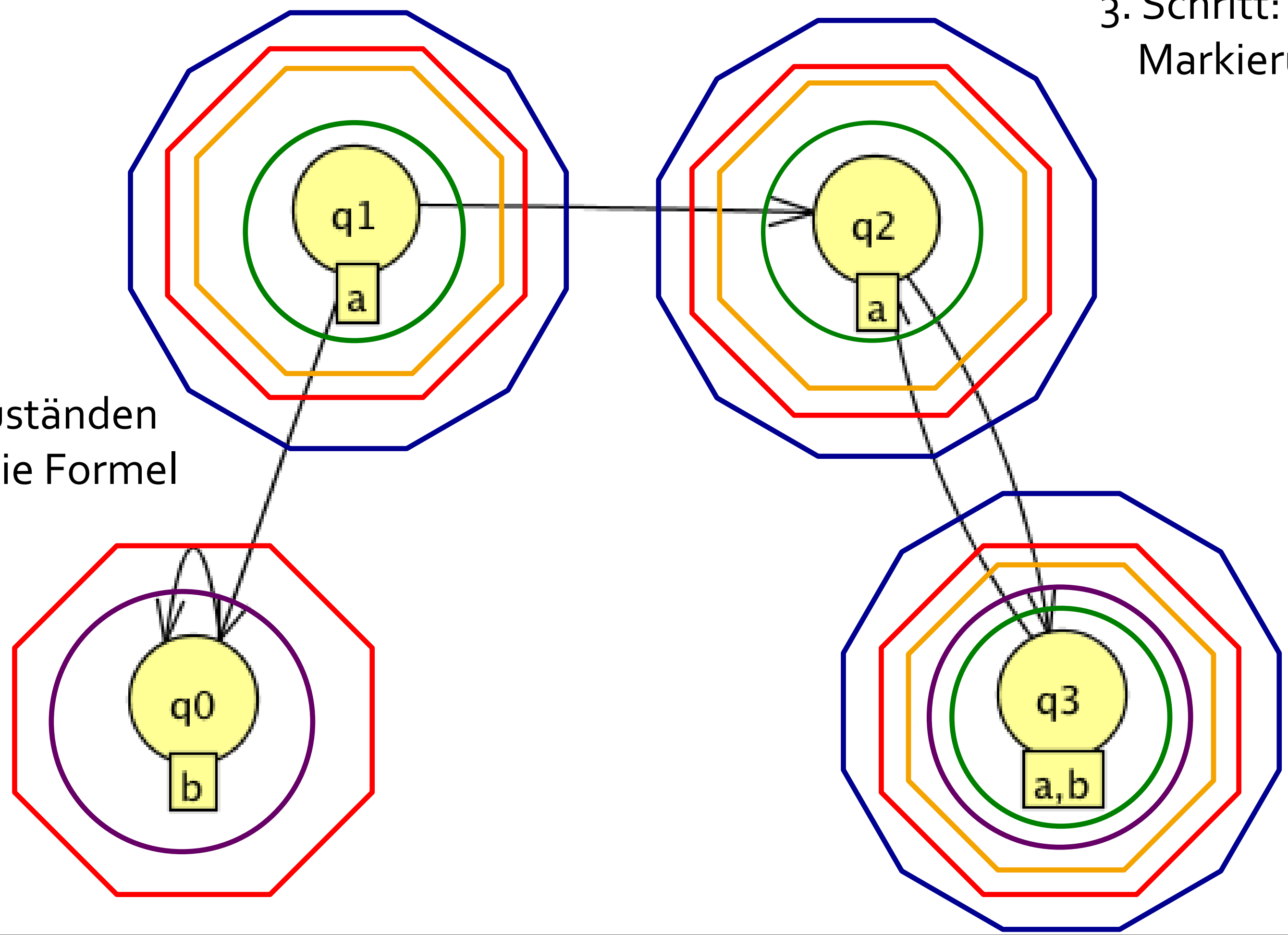
2. Schritt: orange
& rote Markierung



SECENG
SECURITY ENGINEERING

Model Checking CTL
Beispiel (3)

$\mathbf{EG}a \wedge \mathbf{AF}b$



3. Schritt: blaue Markierung

Lösung: in den Zuständen q1, q2 und q3 ist die Formel erfüllt.

SMV

- *Symbolic Model Verifier* (Ken McMillan, 1993)
- Neue Implementierung: NuSMV http://nusmv.irst.itc.it

- Modell wird in spezieller Eingabesprache spezifiziert
- Spezifikationen in CTL
- Interne Darstellung des Modells durch effiziente Datentypen (OBDD)

- Erlaubt Spezifikation von:
  - synchronen und asynchronen Systemen
  - Nichtdeterminismus (z.B. unbekannte Eingaben)
  - Endliche Datentypen (integers, enums, Boolesche Werte)

Beispiel: Mutual Exclusion (1)

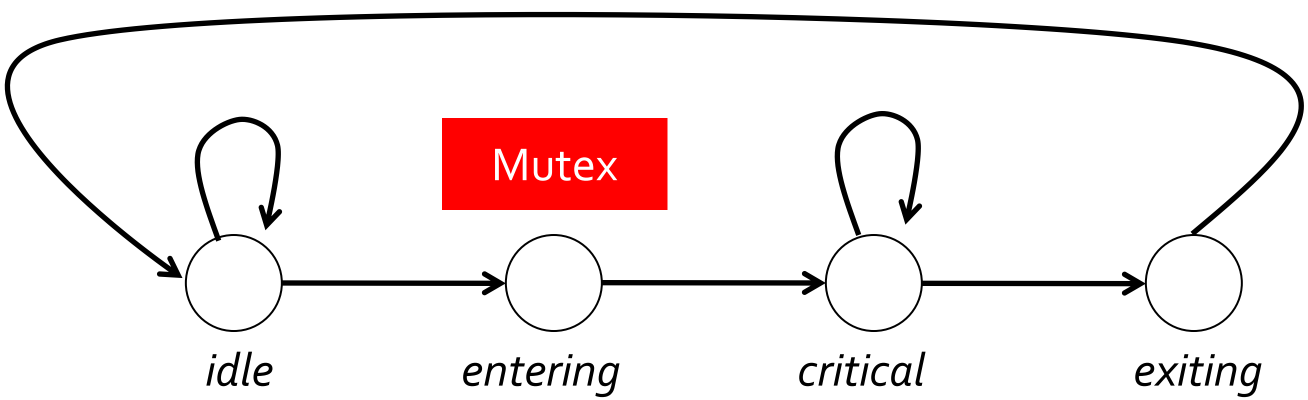
- Zwei Prozesse greifen auf die gleiche Ressource zu
- *Critical Sections:* Teile der beiden Prozesse, die nicht zugleich aktiv sein dürfen

- Synchronisation durch **Mutex**
  - Binäres Register
  - Falls Wert gleich 1, darf ein Prozess auf die Ressource zugreifen; gleichzeitig wird Register auf 0 gesetzt
  - Änderungen des Registers sind *atomar*!

- Gewünschte Eigenschaften:
  - Es befindet sich immer nur ein Prozess in der „critical section"
  - Jeder Prozess, der Ressource benötigt, bekommt sie schließlich auch

Beispiel: Mutual Exclusion (2)

- Zwei Prozesse greifen auf die gleiche Ressource zu
- *Critical Sections:* Teile der beiden Prozesse, die nicht zugleich aktiv sein dürfen



Beispiel: Mutual Exclusion (3)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
MODULE user(mutex)
VAR
   state : {idle, entering, critical, exiting};
ASSIGN
   init(state) := idle;
   next(state) :=
     case
        state = idle : {idle, entering};
        state = entering & mutex : critical;
        state = critical : {critical, exiting};
        state = exiting : idle;
        1 : state;
     esac;
   next(mutex) :=
     case
        state = entering : 0;
        state = exiting : 1;
        1 : mutex;
     esac;
FAIRNESS
   running
```

Anfangszustand

Implementierung des vereinfachten Modells eines Mutex als „state machine"

„Fairness"-Bedingung: wir betrachten nur Abläufe in denen Der Prozess „immer wieder" (unendlich oft) Rechenzeit bekommt.

SecEng Security Engineering

Beispiel: Mutual Exclusion (4)

```
MODULE main
VAR
   mutex: boolean;
   proc1 : process user(mutex);
   proc2 : process user(mutex);
ASSIGN
   init(mutex) := 1;
SPEC
   AG !(proc1.state = critical & proc2.state = critical)
SPEC
   AG (proc1.state = entering -> AF (proc1.state = critical))
```

Definition zweier paralleler Prozesse

es dürfen nicht beide Prozesse gleichzeitig in der „critical section“ sein

kein „starvation“: jeder Prozess der in die „critical section“ eintreten möchte kann dies irgendwann auch

Beispiel: Mutual Exclusion (5) Gegenbeispiel

```
-- specification AG !(proc1.state = critical & proc2.state = critical)  is true
-- specification AG (proc1.state = entering -> AF proc1.state = critical)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  semaphore = 1
  proc1.state = idle
  proc2.state = idle
-> Input: 1.2 <-
  _process_selector_ = proc1
  running = 0
  proc2.running = 0
  proc1.running = 1
-- Loop starts here
-> State: 1.2 <-
  proc1.state = entering
-> Input: 1.3 <-
  _process_selector_ = proc2
  running = 0
  proc2.running = 1
  proc1.running = 0
  ...
```

Angabe eines Pfades, der die zweite Spezifikation verletzt; es kann also zu „starvation“ kommen!

Ausblick

- SMV war der erste Model-Checker, der auf brauchbare Modellgrößen anwendbar war

- Nächste 3 Wochen:
    - Model Checker SPIN
    - Anwendungen von SPIN
    - Syntax der Modelle: PROMELA
    - Spezifikationen in anderer Zeitlogik LTL
    - Kurzer Abriss der Theorie zu LTL und „Büchi-Automaten"

- SPIN ist Basis des ersten Labs