Formale Grundlagen der Informatik 3

Temporale Logik LTL, Büchi-Automaten, Model Checking LTL

Prof. Stefan Katzenbeisser
Security Engineering Group
Technische Universität Darmstadt

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

Wiederholung:
Model Checking

TECHNISCHE UNIVERSITÄT DARMSTADT

A.M. TURING AWARD

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

SecEng Security Engineering

Wiederholung: Temporäre Quantoren

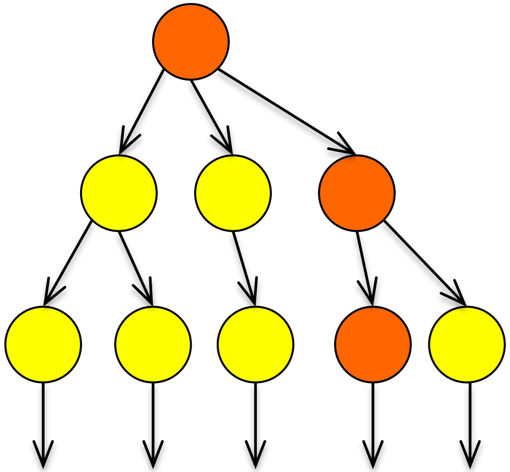
Pfadquantoren: betreffen **Pfade ab einem bestimmten Zustand**

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

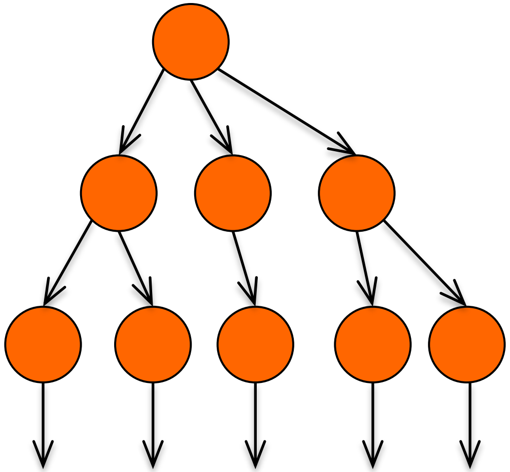
Zustandsquantoren: betreffen **einen bestimmten Pfad**

- **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)

Wiederholung: Pfadquantoren

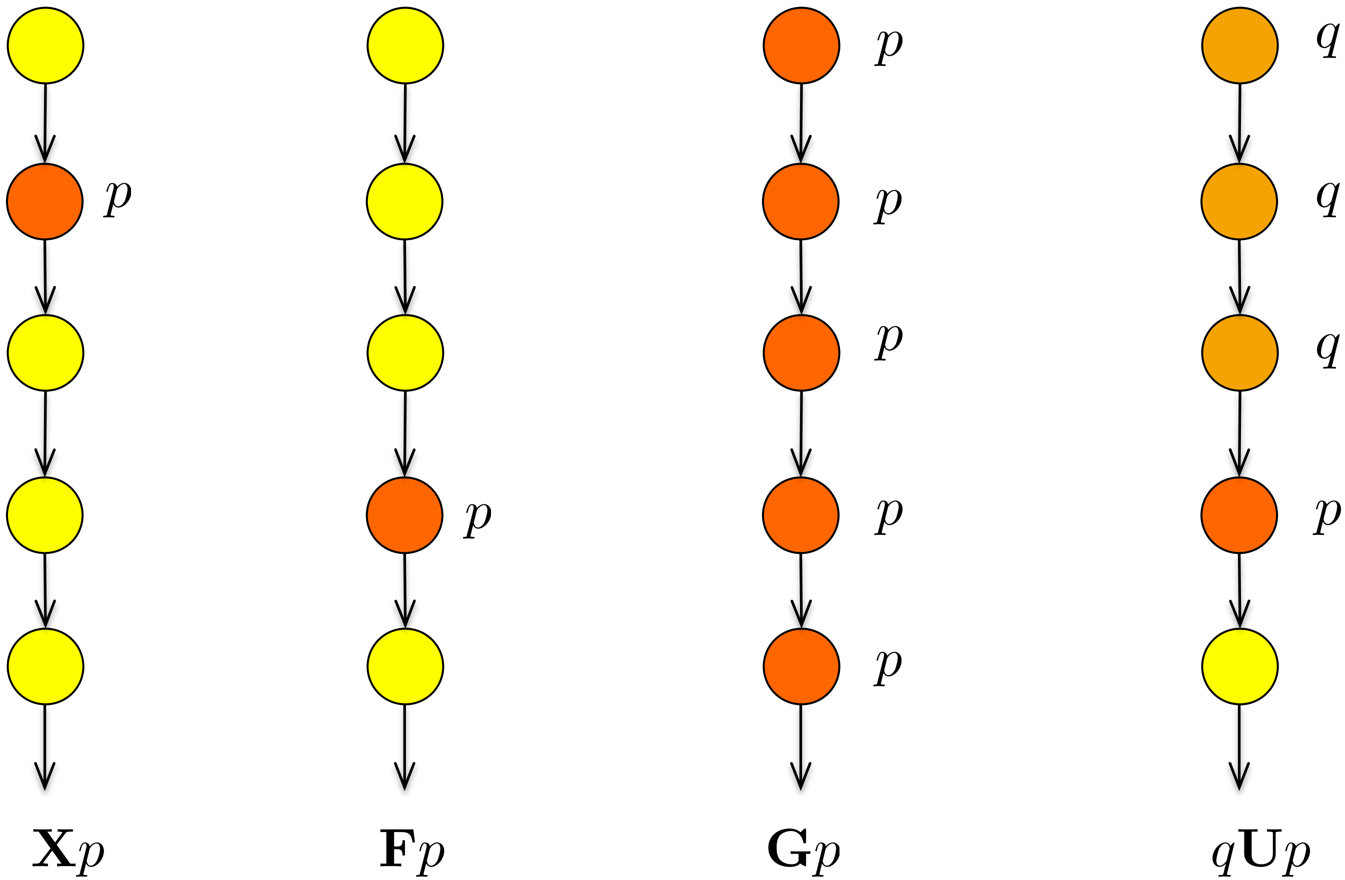


E: „es gibt einen Pfad auf dem gilt" (**E**XISTS)



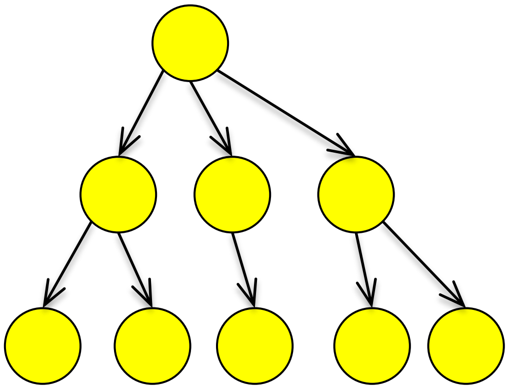
A: „für jeden Pfad gilt" (**A**LL)

Wiederholung: Zustandsquantoren

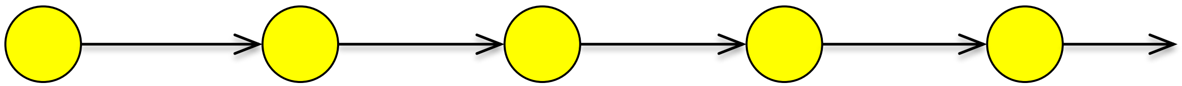


Temporale Logik CTL vs. LTL

- CTL basiert auf Computation Trees …
  … und macht Aussagen über einen gesamten Baum



- LTL = „Linear Time Logic"
  … macht nur Aussagen über **alle** Pfade, die in einem Zustand starten
- Struktur der Formel: $\mathbf{A}\varphi$ wobei φ **keine** Pfadquantoren mehr enthält



Temporale Logik LTL
Syntax

Syntax:

Die Menge der LTL-Formeln ist die Menge aller Formeln der Form $\mathbf{A}\varphi$ wobei φ eine Pfadformel ist. Die Menge der Pfadformeln ist die kleinste Menge für die gilt:

- Atomare Eigenschaften $p \in P$ sowie die Konstanten $\top, \bot$ sind Pfadformeln.
- Sind φ und ψ Pfadformeln, dann sind auch
  $\neg\varphi, \varphi \vee \psi, \varphi \wedge \psi, \varphi \rightarrow \psi, \mathbf{X}\varphi, \mathbf{F}\varphi, \mathbf{G}\varphi, \varphi\mathbf{U}\psi$
  Pfadformeln.

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

$\mathbf{A}(\mathbf{GF}p \rightarrow \mathbf{F}q)$

$\mathbf{A}(\mathbf{FG}p)$

Keine LTL-Formeln: $\mathbf{EG}r$

$\mathbf{AGAF}p$

Hinweis: Manche Autoren schreiben statt **G** das Symbol $\Box$ und statt **F** $\Diamond$

Temporale Logik LTL
Semantik der Pfadformel (1)

- Ausgangspunkt: Kripke-Struktur $\mathcal{M} = (S, I, R, L)$
- Pfadformeln werden über (unendlichen) Pfaden $\pi = s_0 s_1 s_2 s_3 \ldots$ ausgewertet.
- Wir schreiben: $\pi \models \varphi$ wenn φ über dem Pfad π erfüllt ist.

Semantik, Teil 1:

- $\pi \models \top, \pi \not\models \bot$ für alle Pfade π (Konstanten)
- $\pi \models p$ falls $p \in L(s_0)$ (Atomare Aussagen)
- $\pi \models \neg\varphi$ falls $\pi \not\models \varphi$ (Negation)
- $\pi \models \varphi \wedge \psi$ falls $\pi \models \varphi$ und $\pi \models \psi$ (Konjunktion)
- $\pi \models \varphi \vee \psi$ falls $\pi \models \varphi$ oder $\pi \models \psi$ (Disjunktion)
- $\pi \models \varphi \rightarrow \psi$ falls $\pi \not\models \varphi$ oder $\pi \models \psi$ (Implikation)

Temporale Logik LTL
Semantik der Pfadformel (2)

- Wiederholung: für einen Pfad $\pi = s_0 s_1 s_2 s_3 \ldots$
  schreiben wir π^i für den Pfad $s_i s_{i+1} s_{i+2} \ldots$

Semantik, Teil 2:

- $\pi \models \mathbf{G}\varphi$ falls für alle Indices $k \geq 0$ gilt $\pi^k \models \varphi$
- $\pi \models \mathbf{F}\varphi$ falls es einen Index $k \geq 0$ gibt mit $\pi^k \models \varphi$
- $\pi \models \mathbf{X}\varphi$ falls $\pi^1 \models \varphi$
- $\pi \models \varphi \mathbf{U} \psi$ falls es einen Index $k \geq 0$ gibt mit $\pi^k \models \psi$
  und für alle Indices $0 \leq j < k$ gilt $\pi^j \models \varphi$

Temporale Logik LTL
Semantik

Eine LTL-Formel $\mathbf{A}\varphi$ ist im Zustand s_0 einer Kripke-Struktur $\mathcal{M} = (S, I, R, L)$ erfüllt wenn für alle Pfade $\pi = s_0 s_1 s_2 s_3 \ldots$ die in s_0 starten gilt: $\pi \models \varphi$
Wir scheiben dann $\mathcal{M}, s_0 \models \mathbf{A}\varphi$

Analog zu CTL definieren wir die Gültigkeit und Erfüllbarkeit:

Eine LTL-Formel $\mathbf{A}\varphi$ ist gültig wenn für alle Kripke-Strukturen $\mathcal{M} = (S, I, R, L)$ und alle Zustände $s \in I$ gilt $\mathcal{M}, s \models \mathbf{A}\varphi$

Eine LTL-Formel $\mathbf{A}\varphi$ ist erfüllbar, wenn es eine Kripke-Struktur $\mathcal{M} = (S, I, R, L)$ und einen Zustand $s \in I$ gibt mit $\mathcal{M}, s \models \mathbf{A}\varphi$

Temporale Logik LTL
Beispiele

Sind die folgenden Ausdrücke

- erfüllbar?
- gültig?

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

$$\mathbf{A}(\mathbf{GF} p \rightarrow \mathbf{F} q)$$

$$\mathbf{A}(\mathbf{FG} p)$$

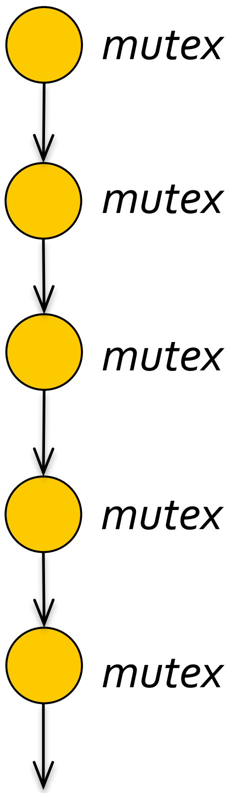
$$\mathbf{A}(\mathbf{GF} p)$$

$$\mathbf{A}(\mathbf{G} p \rightarrow \mathbf{F} p)$$

Temporale Logik LTL
Safety-Eigenschaften

- Formeln der Struktur $\mathbf{A}(\mathbf{G}\varphi)$ nennt man **„Safety-Eigenschaften"**
- Modellierung von Eigenschaften wie „Ein schlechtes Ereignis X tritt niemals ein"
- Beispiel: Sei *mutex* eine Variable, die gültig ist falls *zwei* Prozesse nicht gleichzeitig auf eine Ressource zugreifen.

  $\mathbf{A}(\mathbf{G}\, mutex)$



Temporale Logik LTL
Liveness-Eigenschaften

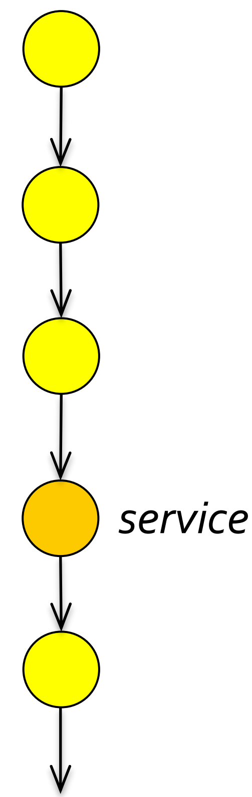
TECHNISCHE UNIVERSITÄT DARMSTADT

- Formeln der Struktur $\mathbf{A}(\mathbf{F}\varphi)$ nennt man **„Liveness-Eigenschaften"**

- Modellierung von Eigenschaften wie „Ein gutes Ereignis X tritt irgendwann einmal ein"

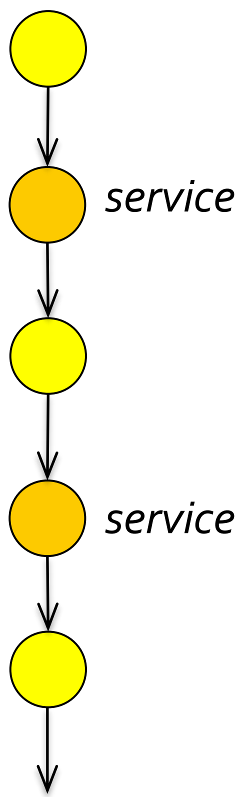
- Beispiel: Sei *service* eine Variable, die gültig ist falls ein bestimmtes Service angeboten wird.

  $\mathbf{A}(\mathbf{F}\,service)$



SECENG SECURITY ENGINEERING

Temporale Logik LTL
Kombination von Liveness und Safety



Liveness kann genutzt werden um Aussagen wie

„X ist unendlich oft erfüllt“

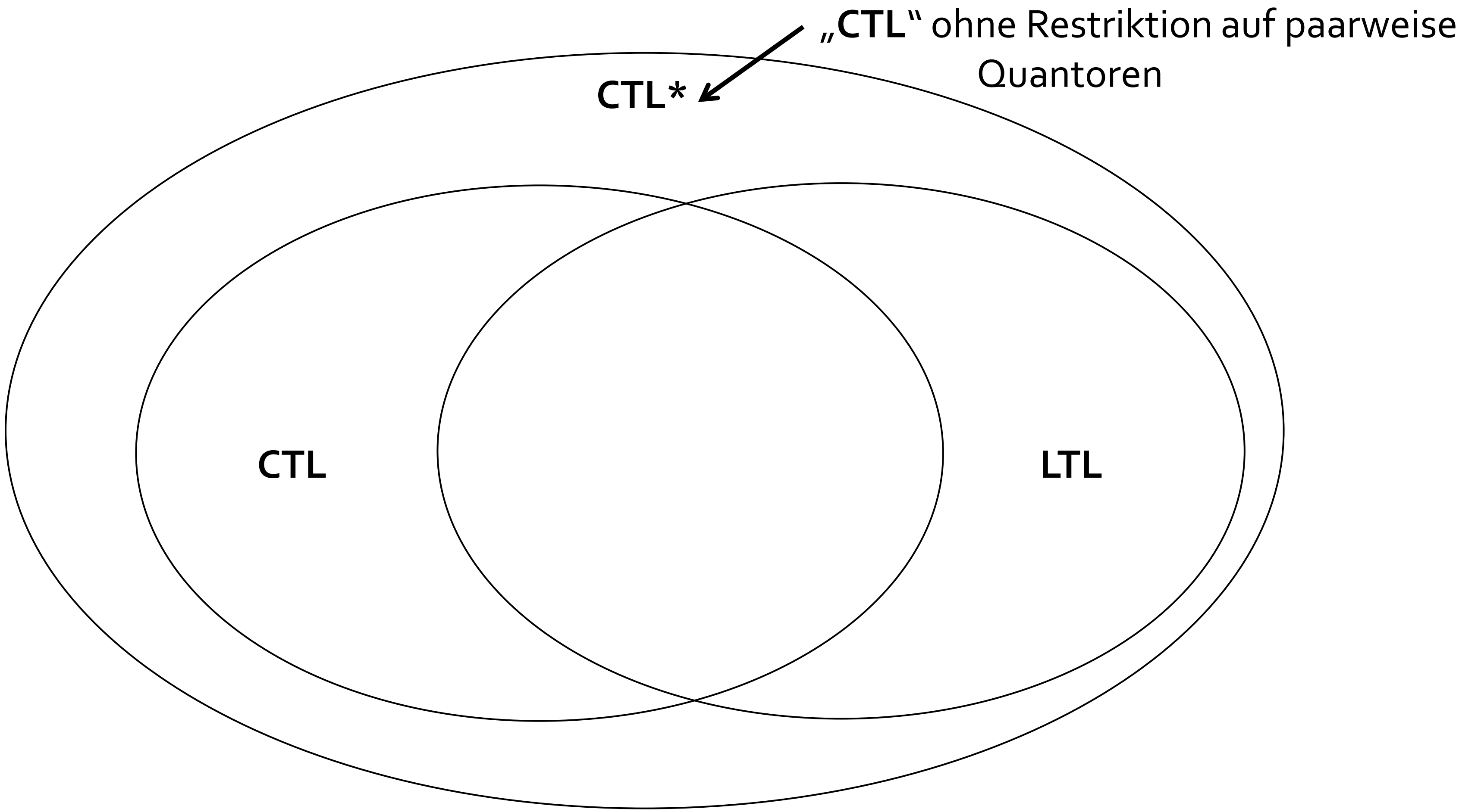
oder

„X erreicht immer wieder einen Zustand..“

zu modellieren

$$\mathbf{A}(\mathbf{GF}\varphi)$$

CTL, LTL und CTL*
Ausdrucksstärke (1)



CTL, LTL und CTL* Ausdrucksstärke (2)

- Es gibt keine CTL-Formel äquivalent zur LTL-Formel

  $\mathbf{A}(\mathbf{FG}p)$

- Es gibt keine LTL-Formel äquivalent zur CTL-Formel

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

- Die Formel
  $\mathbf{A}(\mathbf{FG}p) \vee \mathbf{AG}(\mathbf{EF}p)$
  ist in CTL*, aber weder in LTL noch in CTL ausdrückbar

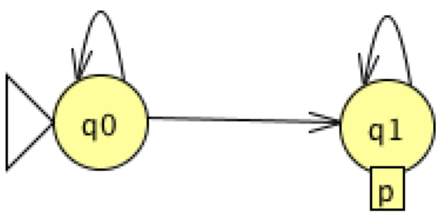
CTL, LTL und CTL*
Ausdrucksstärke (3)

Es gibt keine LTL-Formel äquivalent zur CTL-Formel

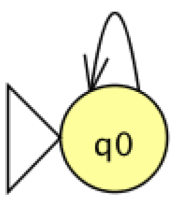
$$\mathbf{AG}(\mathbf{EF}p)$$

Beweisskizze: Nehmen wir das Gegenteil an, also es existiert eine LTL-Formel φ äquivalent zu $\mathbf{AG}(\mathbf{EF}p)$. Betrachte zwei Kripke-Strukturen

T1:



T2:



Es gilt $T1, q_0 \models \mathbf{AG}(\mathbf{EF}p)$ und damit nach Annahme $T1, q_0 \models \varphi$.
Alle Pfade in T2 sind auch Pfade in T1. Gilt eine LTL-Formel daher in T1, so gilt sie auch in T2. Daher impliziert $T1, q_0 \models \varphi$ auch $T2, q_0 \models \varphi$. Allerdings gilt: $T2, q_0 \not\models \mathbf{AG}(\mathbf{EF}\varphi)$. Dies ist ein Widerspruch!

Büchi-Automaten

- Endliche Automaten beschreiben Sprachen **endlicher** Wörter
  (die Menge der Wörter ist in der Regel **unendlich**)
- Endliche Automaten korrespondieren zu regulären Ausdrücken
- Büchi-Automaten akzeptieren **unendlich lange** Wörter
- Büchi-Automaten korrespondieren zu omega-regulären Ausdrücken
- Gleiche Struktur wie endliche Automaten, andere Akzeptanzbedingung

Wörter, Wiederholung

Wiederholung:

- Σ^* bezeichnet die Menge aller endlichen Wörter über endlicher Menge Σ
- Formal:

$$\Sigma^* = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} \{a_1 a_2 a_3 \ldots a_i | a_k \in \Sigma, 1 \leq k \leq i\}$$

- Jede Teilmenge $L \subseteq \Sigma^*$ bezeichnet man als „Sprache“

Reguläre Ausdrücke

- Die leere Menge und jedes Symbol $a \in \Sigma$ ist ein regulärer Ausdruck
- Sind x und y reguläre Ausdrücke, so sind es auch $xy, x|y, x^*$

Beispiele: $ab^*c, (a|bc^*), a^*ba^*, (a|b|c)^*$

Unendliche Wörter

- Σ^ω bezeichnet die Menge aller **unendlichen** Wörter über endlicher Menge Σ
- Formal: $\Sigma^\omega = \{a_1 a_2 a_3 a_4 \ldots | a_i \in \Sigma, i \geq 1\}$

Wichtig:

- Σ^* ist in der Regel **unendlich** groß, enthält Wörter **endlicher** Länge
- Σ^ω ist **unendlich** groß, enthält **nur unendlich** lange Wörter

Omega-reguläre Sprachen:

- Sind x und y (wobei y das leere Wort **nicht** enthält) **reguläre** Ausdrücke, so ist xy^ω ein omega-regulärer Ausdruck
- Sind x und y omega-reguläre Ausdrücke, so ist es auch $x|y$

Beispiele:

$$a^\omega, (ab)^\omega, (a|b)^\omega, (a^*ba)^\omega, a^*ba^\omega$$

Büchi-Automaten (1)

Ein Büchi-Automat $\mathcal{M} = (M, R, q, E, \Sigma)$ über einem endlichen Alphabet Σ besteht aus einer endlichen Menge M an Zuständen, einer Relation $R \subseteq M \times \Sigma \times M$, einem Startzustand $q \in M$ und einer Menge an Endzuständen $E \subseteq M$.

Wichtig: Ein Büchi-Automat unterscheidet sich von einem endlichen Automaten nur in der Art und Weise wie er Wörter akzeptiert.

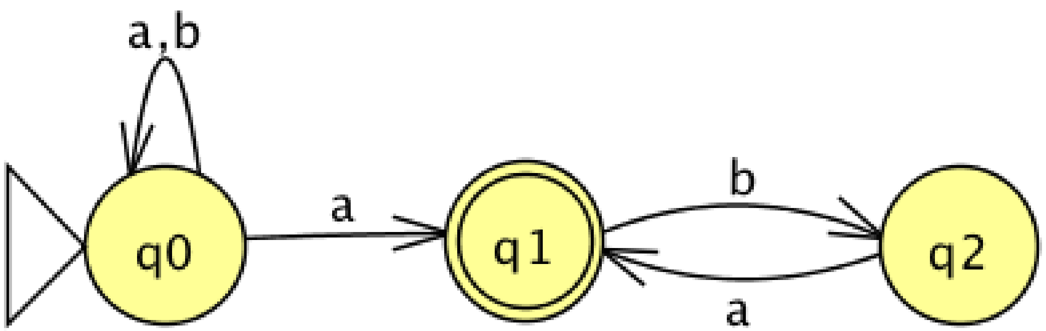
Ein unendliches Wort $w = w_0 w_1 \ldots \in \Sigma^\omega$ korrespondiert zu einem Ablauf eines Büchi-Automaten $\mathcal{M} = (M, R, q, E, \Sigma)$ wenn es eine Sequenz von Zuständen $m_0, m_1, m_2, \ldots \in M$ gibt mit $m_0 = q$ und $(m_{k-1}, w_{k-1}, m_k) \in R$ für alle $k \geq 1$. Der Automat akzeptiert dieses Wort falls mindestens ein Zustand $m \in E$ unendlich oft im Ablauf $m_0 m_1 m_2 \ldots$ vorkommt.

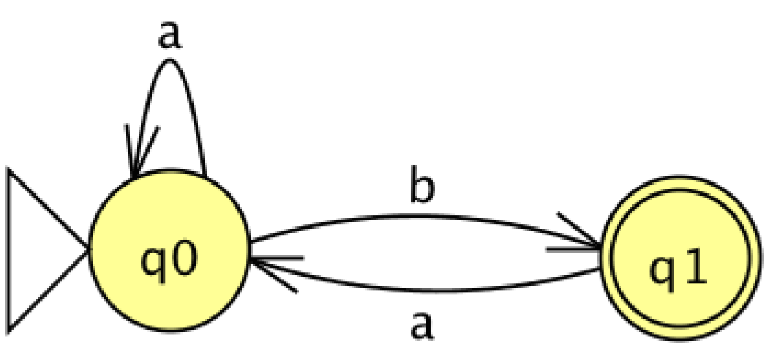
Büchi-Automaten (2)

Ein unendliches Wort $w = w_0 w_1 \ldots \in \Sigma^\omega$ korrespondiert zu einem Ablauf eines Büchi-Automaten $\mathcal{M} = (M, R, q, E, \Sigma)$ wenn es eine Sequenz von Zuständen $m_0, m_1, m_2, \ldots \in M$ gibt mit $m_0 = q$ und $(m_{k-1}, w_{k-1}, m_k) \in R$ für alle $k \geq 1$. Der Automat akzeptiert dieses Wort falls mindestens ein Zustand $m \in E$ unendlich oft im Ablauf $m_0 m_1 m_2 \ldots$ vorkommt.

- Bezeichnet $\inf(m_0 m_1 m_2 \ldots)$ die Menge aller Zustände $m \in M$, die in einem Ablauf $m_0 m_1 m_2 \ldots$ unendlich oft vorkommen.
- Ein Büchi-Automat akzeptiert ein Wort $w = w_0 w_1 \ldots \in \Sigma^\omega$ wenn es einen zugehörigen gültigen Ablauf $m_0 m_1 m_2 \ldots$ gibt mit $\inf(m_0 m_1 m_2 \ldots) \cap E \neq \emptyset$
- Jeder Büchi-Automat korrespondiert zu einer omega-Sprache $\mathcal{L}^\omega(\mathcal{M})$ aller akzeptierten Wörter
- Eine omega-Sprache für die ein Büchi-Automat existiert ist eine omega-reguläre Sprache

Büchi-Automaten
Beispiele





Büchi-Automaten
Entscheidbarkeit

Es ist entscheidbar ob die Sprache $\mathcal{L}^\omega(\mathcal{M})$ eines Büchi-Automaten $\mathcal{M} = (M, R, q, E, \Sigma)$ die leere Menge ist.

Idee:

- Betrachte Automaten als Graphen (Zustände: M, Kanten: R)
- Starke Zusammenhangskomponenten eines Graphen: jeder Knoten ist durch einen Pfad von jedem anderen erreichbar.
- Liegt ein Endzustand in einer starken Zusammenhangskomponente und ist dieser vom Startzustand erreichbar, so ist der Schnitt sicher nicht-leer!
- Komplexität: linear (Tarjan-Algorithmus)

Büchi-Automaten
Abschlusseigenschaften

TECHNISCHE UNIVERSITÄT DARMSTADT

Die Menge der omega-regulären Sprachen ist unter Vereinigung, Schnitt und Komplement abgeschlossen, d.h.

- sind die Sprachen $\mathcal{L}_1, \mathcal{L}_2$ omega-regulär, so sind es auch die Sprachen $\mathcal{L}_1 \cup \mathcal{L}_2$ und $\mathcal{L}_1 \cap \mathcal{L}_2$;
- ist die Sprache $\mathcal{L}_1$ omega-regulär, so ist es auch $\Sigma^\omega \setminus \mathcal{L}_1$.

Idee für Schnitt: „Produktautomatenkonstruktion"

Büchi-Automaten
Schnitt zweier Automaten

Idee für Schnitt: Produktautomatenkonstruktion

- „Parallelausführung" beider Automaten
- Beachten dass abwechselnd beide Automaten Endzustände besuchen
- Gegeben: $\mathcal{M}_1 = (M_1, R_1, q_1, E_1, \Sigma)$ und $\mathcal{M}_2 = (M_2, R_2, q_2, E_2, \Sigma)$
- Definiere einen neuen Büchi-Automaten mit Zuständen $M_1 \times M_2 \times \{1,2\}$
- Relation: $(m_1, m_2, 1) \to (m_1', m_2', 1)$ falls $m_1 \to m_1', m_2 \to m_2', m_1 \notin E_1$

  $(m_1, m_2, 1) \to (m_1', m_2', 2)$ falls $m_1 \to m_1', m_2 \to m_2', m_1 \in E_1$

  $(m_1, m_2, 2) \to (m_1', m_2', 2)$ falls $m_1 \to m_1', m_2 \to m_2', m_2 \notin E_2$

  $(m_1, m_2, 2) \to (m_1', m_2', 1)$ falls $m_1 \to m_1', m_2 \to m_2', m_2 \in E_2$

  Hierbei steht $\to$ für eine Kante mit einem (gleichen) Eingabesymbol
- Startzustand: $(q_1, q_2, 1)$
- Endzustände: $M_1 \times E_2 \times \{2\}$
- Der neue Automat akzeptiert die Sprache $\mathcal{L}^\omega(M_1) \cup \mathcal{L}^\omega(M_2)$

LTL Model Checking
Prinzipielle Idee: Testen ob $\mathcal{M} \models \varphi$

Modellierung aller möglichen Abläufe

Modell $\mathcal{M}$

Übersetzen in Büchi-Automaten $\mathcal{B}_{\mathcal{M}}$

φ **Spezifikation**

$\mathcal{B}_{\neg\varphi}$ **Übersetzen der Negation in Büchi-Aut.**

Modellierung der nicht erlaubten Abläufe

Bildung des Durchschnitts der beiden Automaten

$\mathcal{B}_{\mathcal{M}} \cap \mathcal{B}_{\neg\varphi}$

Test ob Sprache des Automaten nicht leer ist

Modellierung der „gültigen“ Abläufe (1)

Idee: Transformation der Kripke-Struktur in einen Büchi-Automaten

➔ modelliere „Abläufe“ von Labels

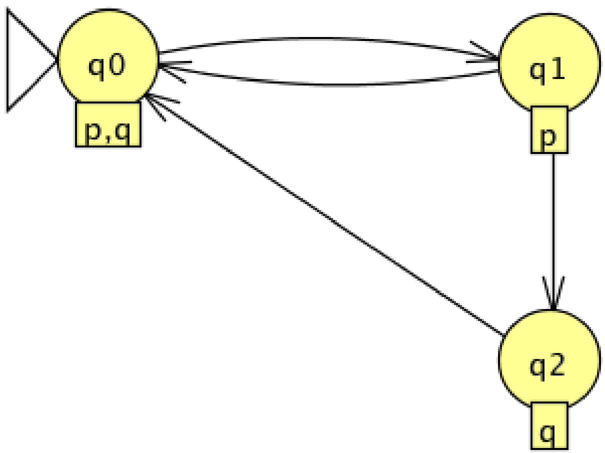
- Übernahme von Kanten und Knoten
- Füge neuen Startzustand hinzu
- Annotiere Kanten mit Labels des Zielknoten
- Lösche die Labels an den Knoten
- Alle Knoten sind Endzustände
- … fertig!

Modell $\mathcal{M}$

↓

Übersetzen in Büchi-Automaten $\mathcal{B}_{\mathcal{M}}$

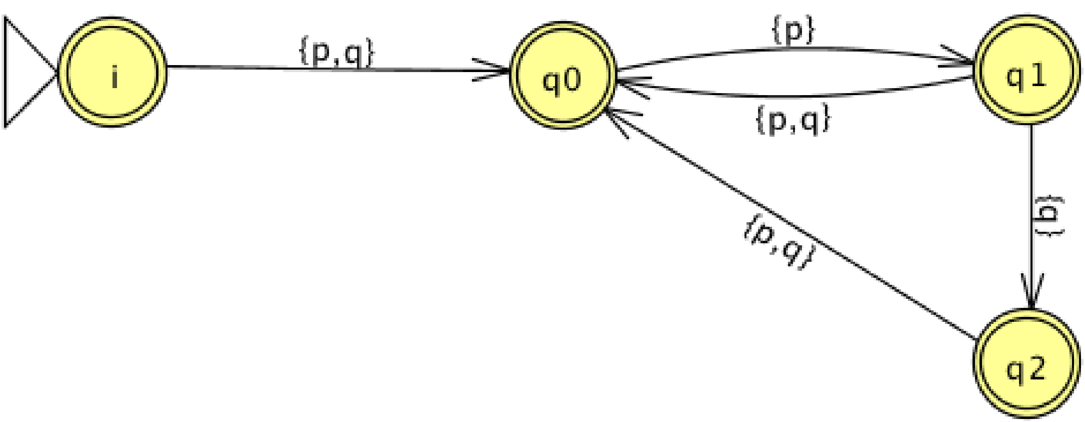
Modellierung der „gültigen“ Abläufe (2)
Beispiel



Modell $\mathcal{M}$

↓

Übersetzen in Büchi-Automaten $\mathcal{B}_{\mathcal{M}}$



Modellierung der „nicht erlaubten“ Abläufe

Idee: Transformation der LTL-Formel in einen Büchi-Automaten

LTL … repräsentiert eine Menge an unendlichen Pfaden, auf denen Formel erfüllt ist

Büchi-Automat … akzeptiert eine Menge an unendlichen Pfaden

Spezifikation φ

↓

Übersetzen der Negation in Büchi-Aut. $\mathcal{B}_{\neg\varphi}$

Zu jeder LTL-Formel existiert ein Büchi-Automat, der alle Pfade akzeptiert auf denen die Formel gültig ist.

Modellierung der „nicht erlaubten“ Abläufe

TECHNISCHE UNIVERSITÄT DARMSTADT

- LTL-Formeln sind üblicherweise „kurz“
- Büchi-Automaten können **exponentiell größer** sein

Spezifikation φ

↓

Übersetzen der Negation in Büchi-Aut. $\mathcal{B}_{\neg\varphi}$

Mehrere Schritte:

- Transformation in „Negations-Normalform“
- Transformation in einen generalisierten Büchi-Automat
- Konstruktion eines Büchi-Automaten

Transformation LTL – Büchi Automaten (1)
1. Schritt: Negationsnormalform

- Alle Negationen sollen direkt vor atomaren Aussagen stehen
- Dies ist mit Umformungsregeln möglich:

$$\neg\neg\varphi = \varphi$$

$$\neg(\varphi_1 \vee \varphi_2) = \neg\varphi_1 \wedge \neg\varphi_2$$ „Regeln nach De Morgan"

$$\neg(\varphi_1 \wedge \varphi_2) = \neg\varphi_1 \vee \neg\varphi_2$$

$$\neg\mathbf{G}\varphi = \mathbf{F}\neg\varphi$$

$$\neg\mathbf{F}\varphi = \mathbf{G}\neg\varphi$$

$$\neg\mathbf{X}\varphi = \mathbf{X}\neg\varphi$$

- Neuer Operator **R** (Releases): $\varphi_1\mathbf{R}\varphi_2$ ist erfüllt, falls φ_2 erst dann nicht mehr gilt, sobald einmal φ_1 gegolten hat (was **nicht** eintreten muss).
- Damit gilt: $\neg(\varphi_1\mathbf{U}\varphi_2) = (\neg\varphi_1)\mathbf{R}(\neg\varphi_2)$ $\qquad \mathbf{F}\varphi = \top\mathbf{U}\varphi$

  $\neg(\varphi_1\mathbf{R}\varphi_2) = (\neg\varphi_1)\mathbf{U}(\neg\varphi_2)$ $\qquad \mathbf{G}\varphi = \bot\mathbf{R}\varphi$

Transformation LTL – Büchi Automaten (2)
2. Schritt: Transformation in Graphen

- Formel enthält nun nur atomare Aussagen (ggf. negiert), Konjunktion, Disjunktion und Operatoren **X**, **U** und **R**
- Idee: rekursiver Algorithmus der Formel schrittweise expandiert und Koten eines Graphen erzeugt
- Struktur jedes Knoten:

Incoming:

Old:

New:

Next:

„Incoming": Vorgänger des Knoten im Automaten
„Old": bereits bearbeitete Formeln
„New": Formeln die noch bearbeitet werden müssen
„Next": Formeln für den nächsten Knoten

- Intuitiv: Knoten beschreibt Suffix π^i eines Pfades; π^i erfüllt alle Formeln in „Old" und „New", π^{i+1} erfüllt alle Formeln in „Next"

Transformation LTL – Büchi Automaten (3)
2. Schritt: Transformation in Graphen

TECHNISCHE UNIVERSITÄT DARMSTADT

Start des Algorithmus:

Incoming: init

Old:

New: φ

Next:

Algorithmus startet auf einem Knoten, der die gesamte Formel enthält; Vorgänger ist artifizieller Startzustand

Schrittweise Verfeinerung:

- Ist in einem Knoten das Feld „New“ nicht leer, so wird der Knoten verfeinert oder ersetzt; die daraus erhaltenen neuen Knoten werden wiederum durch den gleichen Algorithmus verfeinert.
- Dadurch schrittweiser Aufbau des Automaten

Transformation LTL – Büchi Automaten (4)
2. Schritt: Transformation in Graphen

Verfeinerungsregeln:

Old:	$\longrightarrow$	Old: p
New: p		New:
Next:		Next:

atomare Aussagen, ggf. negiert

Old:	$\longrightarrow$	Old: $\varphi_1 \wedge \varphi_2$
New: $\varphi_1 \wedge \varphi_2$		New: φ_1, φ_2
Next:		Next:

Alle hier „leeren" Felder werden kopiert!

Transformation LTL – Büchi Automaten (5)
2. Schritt: Transformation in Graphen

Verfeinerungsregeln:

Old: New: $\varphi_1 \vee \varphi_2$ Next:	$\longrightarrow$	Old: $\varphi_1 \vee \varphi_2$ New: φ_1 Next:	Old: $\varphi_1 \vee \varphi_2$ New: φ_2 Next:
Old: New: $\mathbf{X}\varphi$ Next:	$\longrightarrow$	Old: $\mathbf{X}\varphi$ New: Next: φ	

Alle hier „leeren" Felder werden kopiert!

SecEng Security Engineering

Transformation LTL – Büchi Automaten (6)
2. Schritt: Transformation in Graphen

Alle hier „leeren" Felder werden kopiert!

Verfeinerungsregeln:

Idee: $\varphi_1 \mathbf{U} \varphi_2 = \varphi_2 \vee (\varphi_1 \wedge \mathbf{X}(\varphi_1 \mathbf{U} \varphi_2))$

Old: New: $\varphi_1 \mathbf{U} \varphi_2$ Next:	$\longrightarrow$	Old: $\varphi_1 \mathbf{U} \varphi_2$ New: φ_2 Next:	Old: $\varphi_1 \mathbf{U} \varphi_2$ New: φ_1 Next: $\varphi_1 \mathbf{U} \varphi_2$

Idee: $\varphi_1 \mathbf{R} \varphi_2 = ((\varphi_1 \wedge \varphi_2) \vee (\varphi_2 \wedge \mathbf{X}(\varphi_1 \mathbf{R} \varphi_2)))$

Old: New: $\varphi_1 \mathbf{R} \varphi_2$ Next:	$\longrightarrow$	Old: $\varphi_1 \mathbf{R} \varphi_2$ New: φ_1, φ_2 Next:	Old: $\varphi_1 \mathbf{R} \varphi_2$ New: φ_2 Next: $\varphi_1 \mathbf{R} \varphi_2$

Transformation LTL – Büchi Automaten (7)
2. Schritt: Transformation in Graphen

Transformationsalgorithmus, informelle Beschreibung

- Starte mit einer leeren Menge an „fertigen" Knoten
- Falls der aktuell betrachtete Knoten ein leeres „New"-Feld besitzt:
  - Prüfe, ob bereits ein „fertiger" Knoten existiert der die gleichen „Old" und „Next"-Felder hat; in diesem Fall wird der aktuelle Knoten gelöscht und dessen Vorgänger dem „Incoming"-Feld des existierenden Knotens hinzugefügt
  - Falls kein solcher Knoten existiert wird der aktuelle Knoten als „fertig" markiert und ein neuer Knoten als Nachfolger erstellt, der im „New"-Feld die Formel des aktuellen „Next"-Feldes enthält („Old" und „Next" des neuen Knotens sind leer). Dieser Knoten wird rekursiv mit dem gleichen Algorithmus bearbeitet
- ...

Transformation LTL – Büchi Automaten (8)
2. Schritt: Transformation in Graphen

Transformationsalgorithmus, informelle Beschreibung, Fortsetzung

- …
- Sonst wird eine Formel im „New"-Feld ausgewählt
  - Ist diese Formel bereits im „Old"-Feld enthalten, so wird sie gelöscht und der Algorithmus auf den gleichen Knoten nochmals angewendet.
  - Ist die Formel nicht enthalten, so erfolgt eine Verfeinerung nach den vorhin beschriebenen Regeln
    - Wird ein Knoten neu angelegt, so werden zuerst alle Felder „Incoming", „Old", „New" und „Next" in den neuen Knoten kopiert und dann ggf. durch den Algorithmus angepasst.
    - Knoten mit Widersprüchen im „Old" oder „New"-Feld werden gelöscht.

Transformation LTL – Büchi Automaten (9)
3. Schritt: Transformation in Automaten

- *Alphabet des Automaten:*
  alle atomaren Eigenschaften

- *Zustände des Automaten:*
  alle durch den Algorithmus erzeugten „fertigen“ Knoten sowie der artifizieller Startzustand

- *Zustandsübergänge:*
  alle Übergänge die in den „Incoming“-Feldern generiert wurden; zusammen mit allen atomaren Eigenschaften die die (ggf. negierten) atomaren Eigenschaften im „Old“-Feld des Zielknotens erfüllen

- *Anfangszustand:* artifizieller Anfangszustand

Transformation LTL – Büchi Automaten (10)
3. Schritt: Transformation in Automaten

TECHNISCHE UNIVERSITÄT DARMSTADT

Die Konstruktion liefert einen generalisierten Büchi-Automaten, bei dem es mehrere Akzeptanzmengen gibt; dieser akzeptiert ein unendliches Wort, falls darin für **jede** Akzeptanzmenge mindestens ein akzeptierender Zustand unendlich oft vorkommt.

- *Akzeptanzmengen:*
  - Für jede Teilformel der Form $\varphi_1 \mathbf{U} \varphi_2$ wird eine Akzeptanzmenge erstellt.
  - Menge enthält alle Knoten, in denen φ_2 in „Old" oder $\varphi_1 \mathbf{U} \varphi_2$ nicht in „Old" vorkommt.
  - Dies garantiert dass irgendwann φ_2 gelten muss sobald $\varphi_1 \mathbf{U} \varphi_2$ gilt.

Generalisierte Büchi-Automaten können in Büchi-Automaten transformiert werden.

LTL Model Checking
Abschluss

Bildung des Durchschnitts der beiden Automaten

$\downarrow \mathcal{B}_{\mathcal{M}} \cap \mathcal{B}_{\neg\varphi}$

Test ob Sprache des Automaten nicht leer ist

- Konstruktion des Schnittautomaten ...
- ... und den Test auf die leere Menge wurden bereits besprochen!
- Komplexität?
- Gegenbeispiel: Konstruktion des Schnittautomaten, suche einen Zyklus durch einen akzeptierenden Zustand.