

Model-Checking: Einführung



Universität Paderborn
Fakultät EIM, AG-Engels
Wintersemester 2004/05

Andreas Koop

Überblick

- Motivation
- Grundlagen
 - Modellierung
 - Spezifikation von Eigenschaften
 - Verifikation
- Werkzeuge
 - SMV
 - UML Model-Checker...
- Zusammenfassung und Ausblick

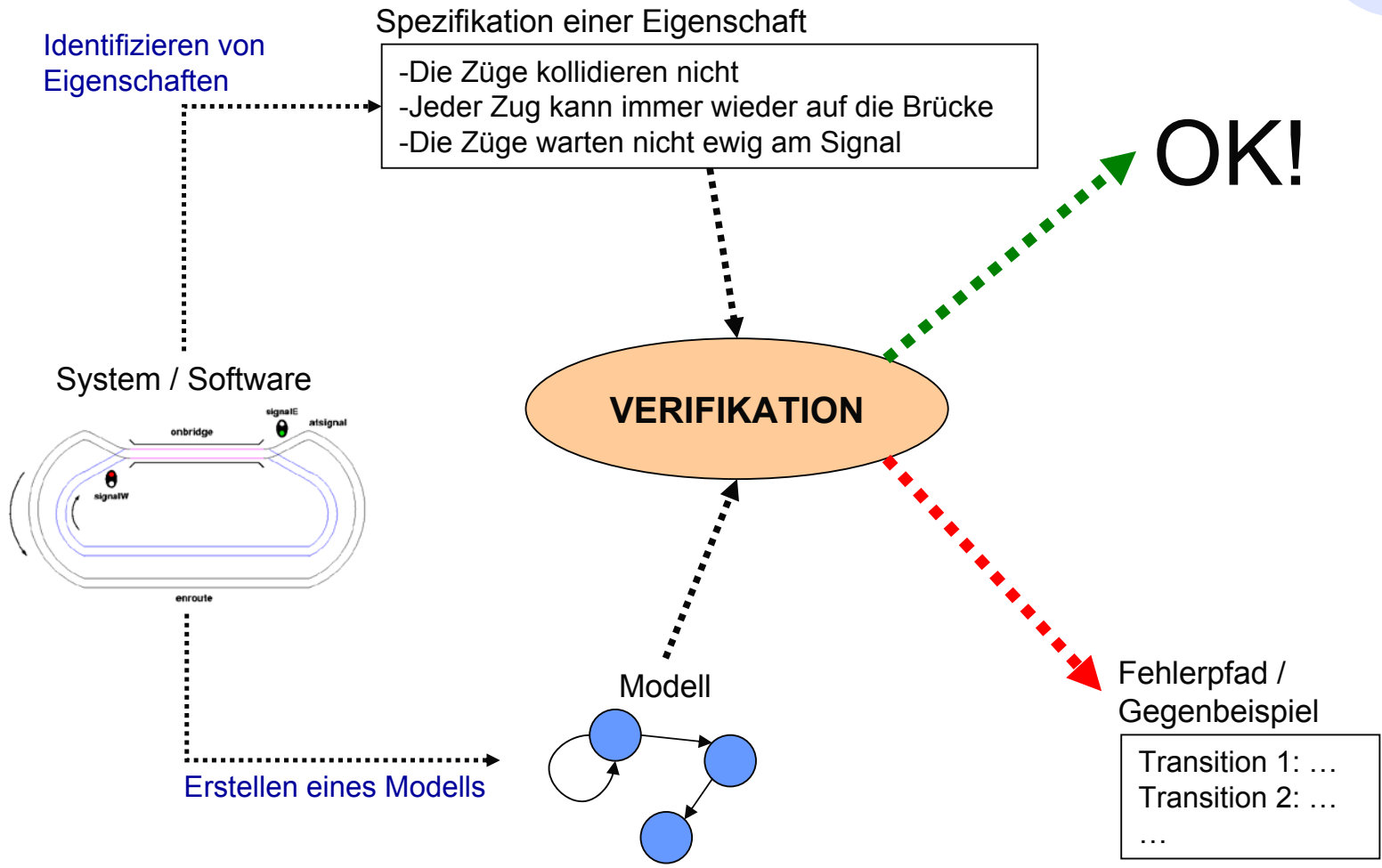
Motivation

- *Wie kann man sicherstellen, dass ein System sich wie gewünscht verhält?*
- **Nicht erkannte Fehler:**
 - **Softwarefehler in der Ariane 5 (Oktober 1996)**
40 Sekunden nach dem Start explodierte die Rakete aufgrund eines Rechenfehlers (Variablenüberlauf) → \$7 Milliarden Schaden
 - **Logikfehler im Airbus A320, Warschau (September 1993)**
Wegen Aquaplaning gaben die Sensoren am Fahrwerk bei der Landung dem Computer die Meldung weiter, das Flugzeug würde noch fliegen. Daher wurde nicht rechtzeitig gebremst und die Maschine prallte gegen einen Hügel → 2 Tote

Verifikationsmethoden

- Tests
 - ...sind noch lange kein Beweis für die Korrektheit eines Systems
- Simulation
 - Oft aufwendig
 - Nicht alle Ergebnisse lassen sich auf die Realität übertragen
- Theorembeweis: HOARE...
 - Sehr aufwendige Methode
 - Viel Erfahrung und Expertenwissen nötig
 - Nicht automatisierbar
- Model-Checking
 - Automatisierbar

Grundlagen: Was ist Model-Checking?



Grundlagen: Model-Checking Ansatz

- **Problem:** Sei M ein Modell, φ eine Spezifikation.
Erfüllt das Modell die Spezifikation?

Formal: Gilt $M \models \varphi$?

- **Vorgehensweise**

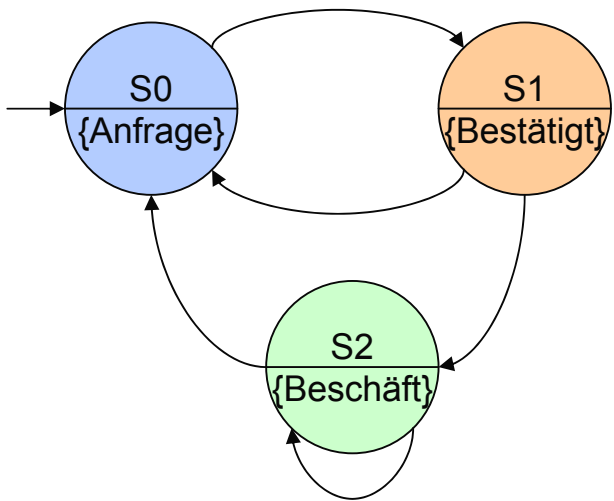
1. **Modelliere** System mittels *Transitionssystem* M
2. **Spezifiziere** gewünschte Eigenschaft φ
3. **Verifiziere:** $M \models \varphi$?

- **Mögliche Antworten**

- Eigenschaft φ gilt (im Modell)
- Eigenschaft φ gilt nicht! → Gegenbeispiel!
- Keine! OutOfMemory / TimeLimitExceeded

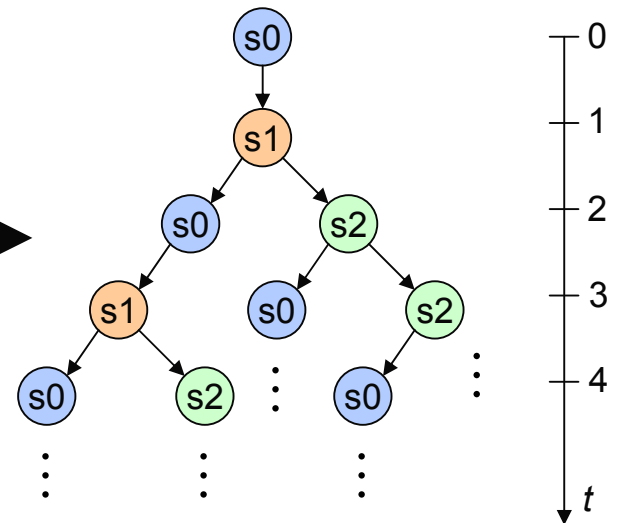
Modell: Kripke-Struktur

- Quadrupel $M = (S, R, L, s_0)$ über den Aussagen $P = \{p_1, \dots, p_n\}$
 - mit endlicher Zustandsmenge S
 - einer Übergangsrelation $R \subseteq S \times S$
 - einer Beschriftungsfunktion $L : S \rightarrow 2^P$, die jedem Zustand eine Menge von Aussagen aus P zuordnet
 - einem Anfangszustand $s_0 \in S$



Abwicklung
=====

Berechnungsbaum

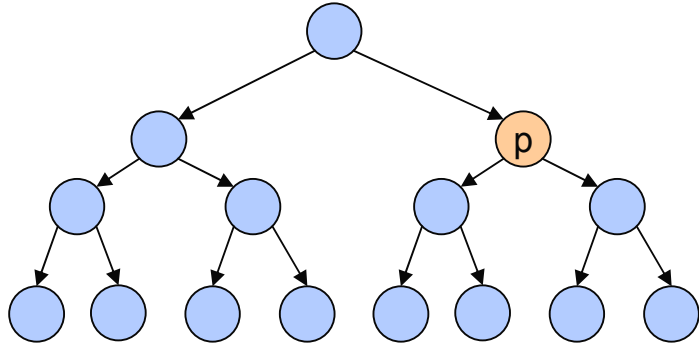


Spezifikation: Temporale Logik CTL

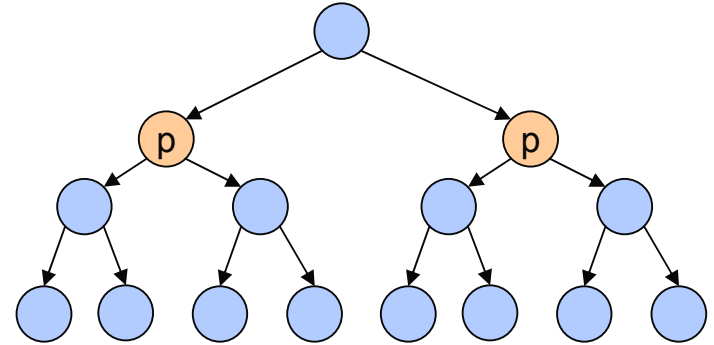
- Formalismus zur Beschreibung von Berechnungsbaum-Eigenschaften (*Computation Tree Logic*)
- Erweiterung der Aussagenlogik um weitere Operatoren:
 - **Pfadquantoren:**
 - **E p (Exists):** Aussage p gilt auf *mindestens* einem Berechnungspfad
 - **A p (Always):** Aussage p gilt auf *allen* Berechnungspfaden → immer!
 - **Temporaloperatoren:**
 - **X p (NeXt):** im *nächsten, direkt folgenden* Zustand gilt p
 - **F p (Future):** *irgenwann* wird ein Zustand erreicht, in dem p gilt
 - **G p (Globally):** in *jedem erreichbaren* Zustand gilt p
 - **p1 U p2 (Until):** p1 gilt in *allen* Zuständen eines Pfades bis ein Zustand erreicht wird, in dem p2 gilt

Spezifikation: Grundlegende CTL-Formeln (1)

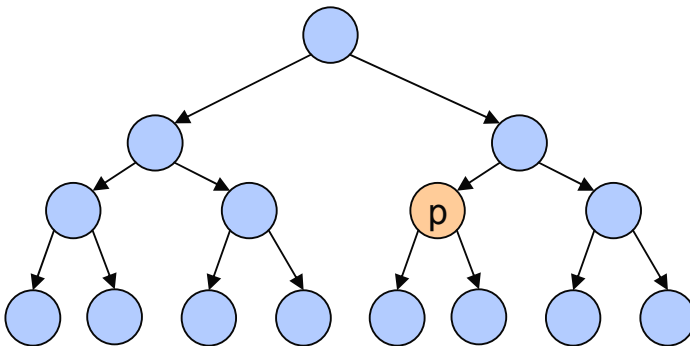
EX p: in *einem* nächsten Zustand gilt p



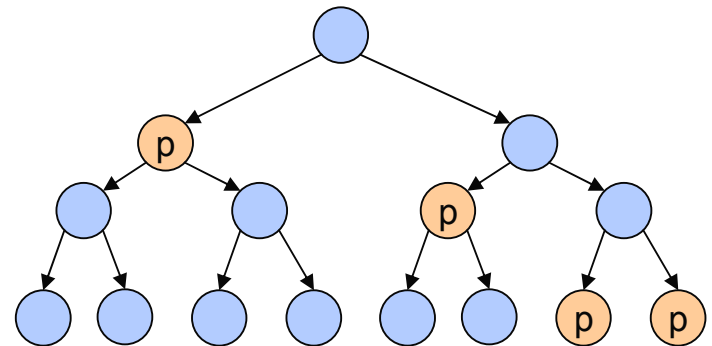
AX p: in *allen* nächsten Zuständen gilt p



EF p: es gibt *einen* Pfad, auf dem *irgendwann* p gilt

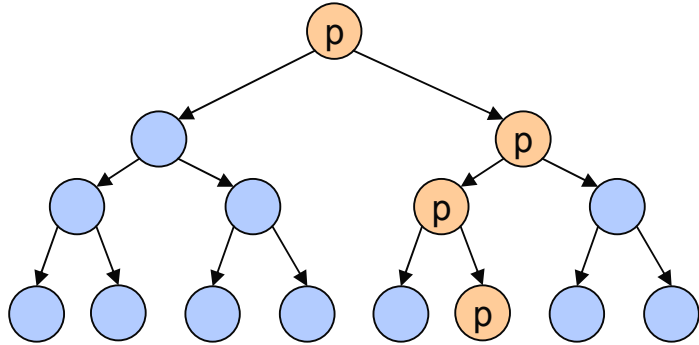


AF p: auf *allen* Pfaden gilt *irgendwann* p

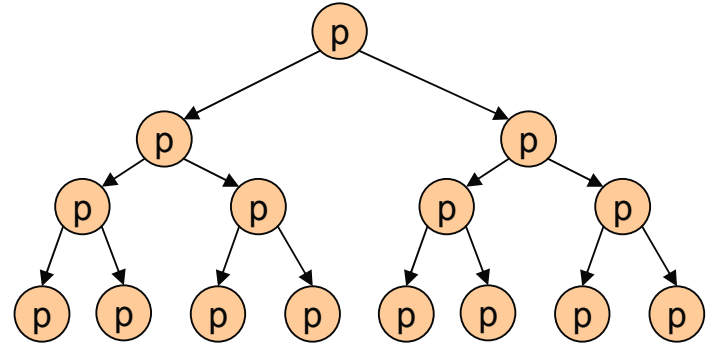


Spezifikation: Grundlegende CTL-Formeln (2)

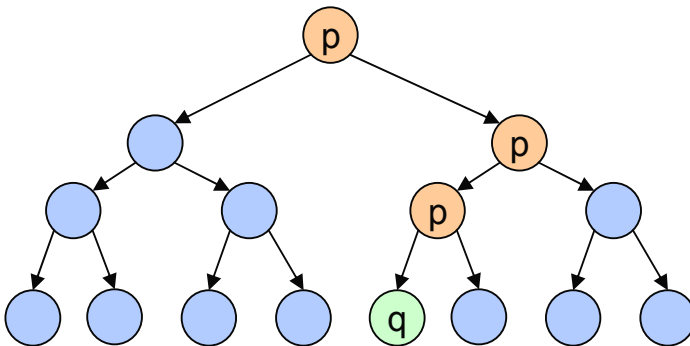
EG p: es gibt *einen* Pfad, auf dem *immer* p gilt



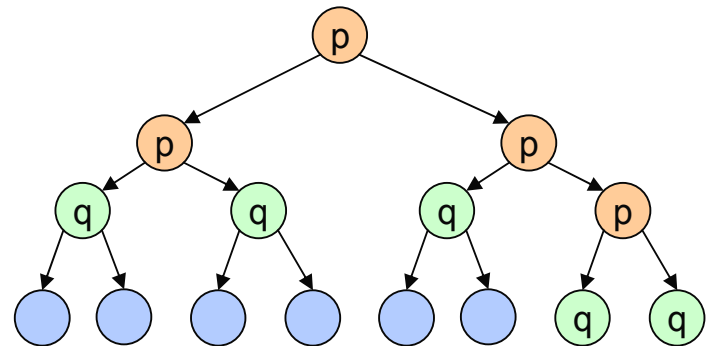
AG p: auf *allen* Pfaden gilt *immer* p



E(pUq): es gibt *einen* Pfad, auf dem immer p gilt bis *irgendwann* q gilt

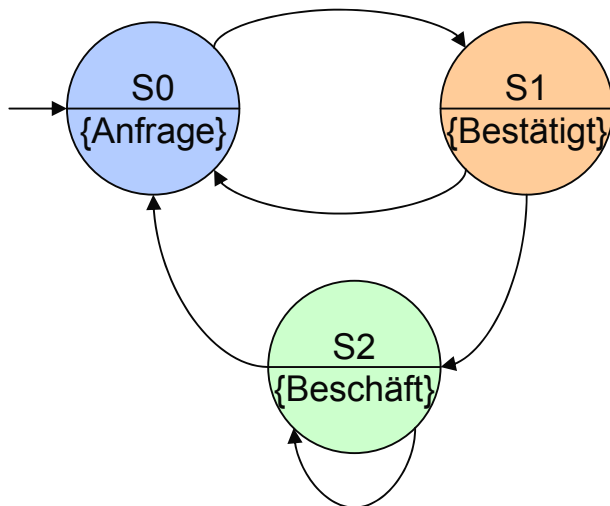


A(pUq): auf *allen* Pfaden gilt immer p bis *irgendwann* q gilt

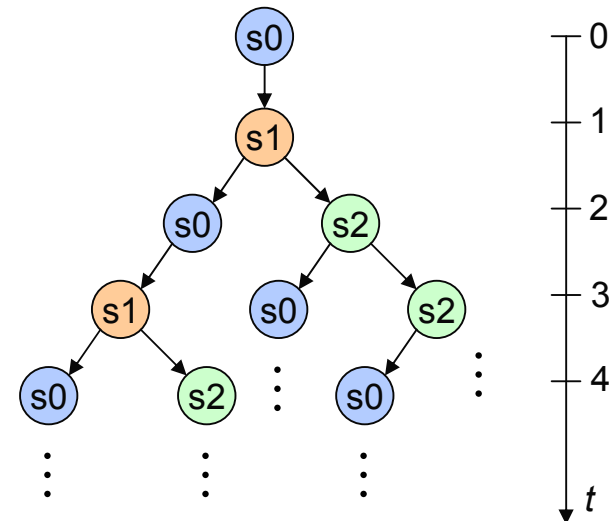


Spezifikation: CTL Beispiel

- **AG**(Anfrage => **AF** Bestätigt): „Jede Anfrage wird irgendwann bestätigt.“ (ist erfüllt)
- **AG**(Beschäftigt): „Das System ist immer beschäftigt.“ (ist nicht erfüllt)
- **AG**(**AF** Beschäftigt): „Das System kann von überall in den *Beschäftigt*-Zustand gelangen.“ (ist erfüllt)

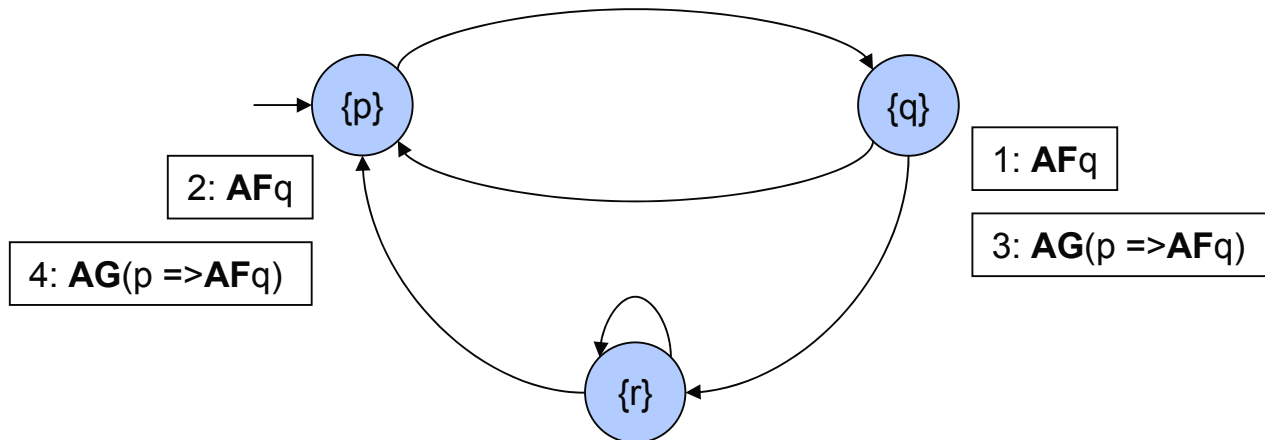


Berechnungsbaum



Verifikation: Markierungsalgorithmus

- Verfolge die Teilformeln ψ von φ gemäß Aufbau von φ
 - Markiere Zustand s mit ψ , falls $(M, s) \models \psi$
 - Prüfe, ob der Anfangszustand mit φ markiert ist
-
- Beispiel
 - Sei $\varphi = \mathbf{AG}(p \Rightarrow \mathbf{AF}q)$
 - Teilformeln: p , q , $\mathbf{AF}q$, $\mathbf{AG}(p \Rightarrow \mathbf{AF}q)$



Verifikation: State-space-explosion

- Bei nebenläufigen Prozessen geht die Anzahl der Zustände der Teilsysteme als kartesisches Produkt in die Anzahl der Zustände des Berechnungsbaums ein
- Beispiel:

```
Process 0: Repeat
  00 non-critical-section
  01 wait unless turn = 0
  10 critical section 1
  11 turn:= 1

Process 1: Repeat
  00 non-critical-section 0
  01 wait unless turn = 1
  10 critical section 0
  11 turn:= 0
```

- 4 Teilzustände, 1 Boolesche Variable
→ $4 \times 4 \times 2 = 32$ Konstellationen
- 20 Prozesse à 10 Zustände
→ 10^{20} Situationen im Berechnungsbaum

→ Notwendigkeit von effizienten Datenstrukturen und Algorithmen

Verifikation: Effizientere Model-Checking Verfahren

- Symbolisches Model-Checking
 - Beschreibung der Zustände und Verhalten eines Systems durch symbolische Formeln, sogenannte Binary Decision Diagrams
 - Symmetrien in Berechnungsbaum werden mit Hilfe der BDDs geschickt zusammengefasst
 - Kein expliziter Aufbau des Berechnungsbaums, sondern Überprüfung einer Eigenschaft auf einer Zustandsmenge
- On-the-fly Model-Checking
 - Expliziter, aber nur teilweiser Aufbau des Berechnungsbaums
 - Effiziente Speicherung und Codierung schon besuchter Zustände
- Abstraktion

Werkzeuge

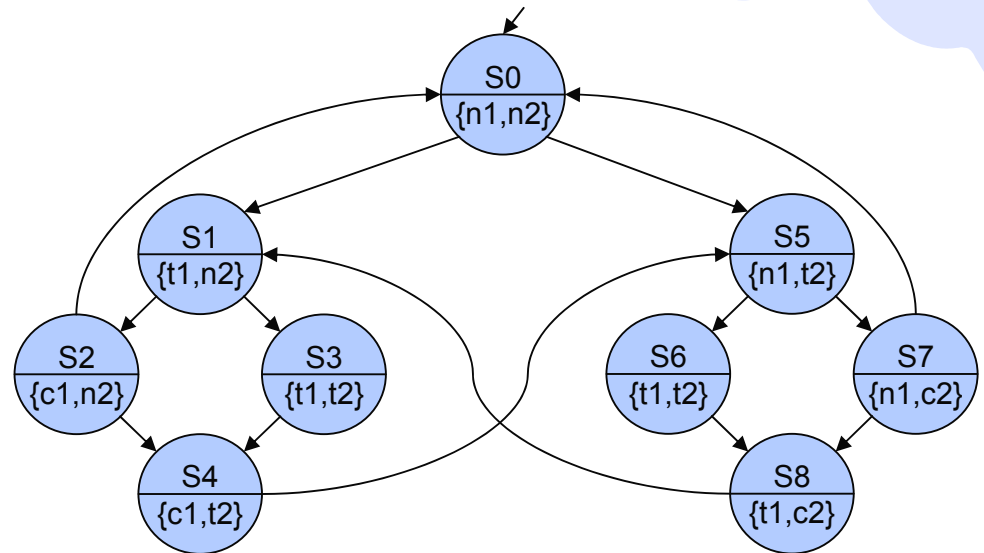
- Symbolic Model Verifier (SMV) (CMU `92)
 - Symbolischer Model-Checker
 - Modellierung mit eigener Eingabesprache (intern BDDs)
 - Spezifikation in Form von CTL-Syntax (alles in einer Datei)
- SPIN (Bell Labs 80er)
 - On-the-fly Model-Checker
 - Softwarepaket zur formalen Überprüfung von verteilten Systemen, meist Datenkontrollprotokollen
 - Modellierung in der PROMELA-Sprache (C-artige Syntax)
 - Spezifikation erfolgt mittels LTL (Linear Temporal Logic)

SMV: Ein Beispiel

MUTEX-Protokoll

```
Process 0: Repeat
  00 non-critical-section
  01 wait unless turn = 0
  10 critical section 1
  11 turn:= 1

Process 1: Repeat
  00 non-critical-section 0
  01 wait unless turn = 1
  10 critical section 0
  11 turn:= 0
```



- n_i (non-critical): Prozess i außerhalb des kritischen Abschnitts
- t_i (trying): Prozess i wartet auf Zugang zum kritischen Abschnitt
- c_i (critical): Prozess i befindet sich im kritischen Abschnitt

SMV: Modellierung durch Module

mutex.smv

```
MODULE main
VAR
  turn : boolean;
  proc1 : process proc(proc2.state, turn, 0);
  proc2 : process proc(proc1.state, turn, 1);
ASSIGN
  init(turn) := 0;

MODULE proc(other-state, turn, myturn)
VAR
  state : {n,t,c};
ASSIGN
  init(state) := n;
  next(state) :=
    case
      (state = n) : {t,n};
      (state = t) & (other-state = n) : c;
      (state = t) & (other-state = t) & (turn = myturn) : c;
      (state = c) : {c,n};
    1 : state;
  esac;
  next(turn) :=
    case
      turn = myturn & state = c : !turn;
    1 : turn;
  esac;
```

SMV: Spezifikation und Ausgabe

■ Spezifikation

- **Sicherheit:** Zu jedem Zeitpunkt darf sich höchstens ein Prozess im kritischen Abschnitt befinden

```
SPEC AG! ((proc1.state = c) & (proc2.state = c))
```

- **Lebendigkeit:** Die Anforderung zum Eintritt in den kritischen Bereich wird irgendwann schließlich erfüllt

```
SPEC AG (proc1.state = t) -> AF (proc1.state = c)
SPEC AG (proc2.state = t) -> AF (proc2.state = c)
```

■ Verifikation:

```
[kopyy@localhost example]$ smv mutex.smv
-- specification AG (! (proc1.state = c & proc2.state = c)) is true
-- specification AG (proc1.state = t -> AF proc1.state = c) is true
-- specification AG (proc2.state = t -> AF proc2.state = c) is true

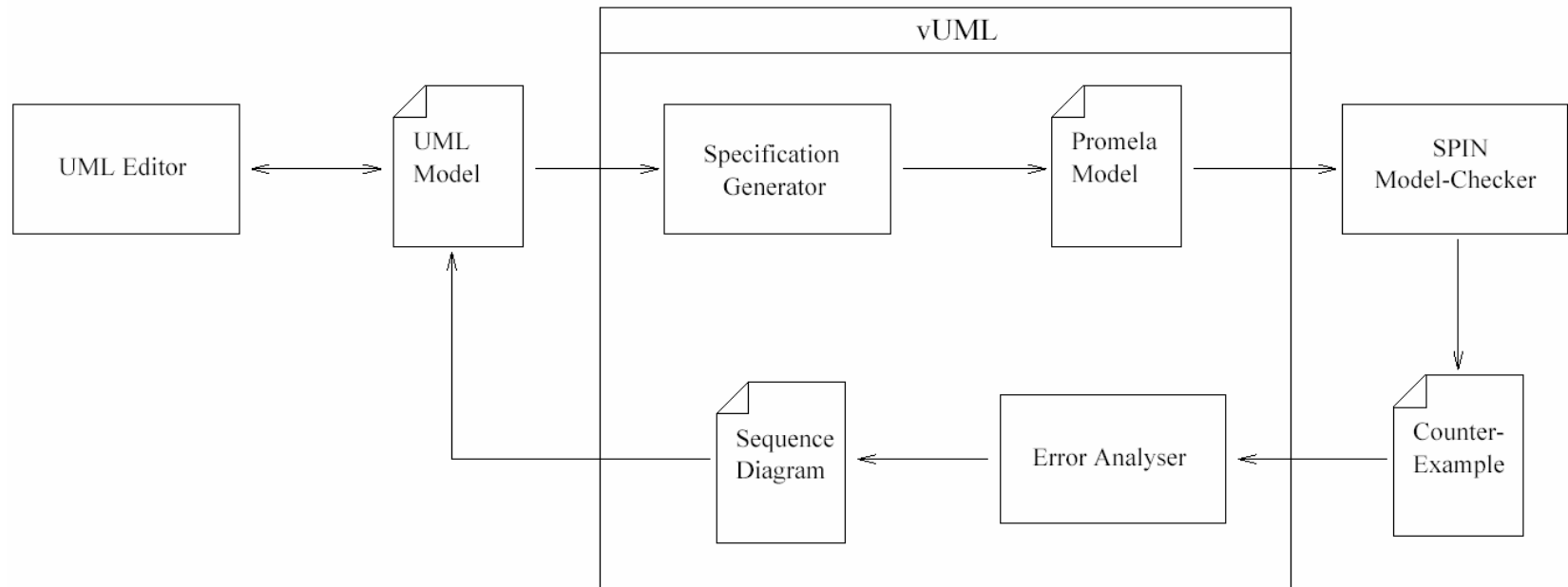
resources used:
user time: 0.01 s, system time: 0 s
BDD nodes allocated: 1117
Bytes allocated: 1245184
BDD nodes representing transition relation: 77 + 6
```

Werkzeuge: UML-Model-Checker

- Hugo/RT
 - Verifikation von Statecharts
 - Anforderung mittels Sequenz- bzw. Kollaborationsdiagramm
 - SPIN als Backend
- vUML
 - Verifikation von Statecharts
 - Vordefinierte Anforderung: Detektion von Deadlocks, Unerreichbaren Zuständen, usw.
 - SPIN als Backend
- „Noname“ (Osaka/Japan)
 - Modell als Klassendiagramm, Statechart
 - Anforderung mittels Sequenzdiagramm
 - SMV als Backend


Werkzeug: vUML

- Verifikation von UML-Statecharts
- Übersetzung in die SPIN Modellierungssprache PROMELA
- Gegenbeispiel in Form eines UML-Sequenzdiagramms
- Detektion von Deadlocks, Unerreichbaren Zuständen



Zusammenfassung

- Model-Checking ist ein *automatisiertes Verfahren* zur *Überprüfung* von Eigenschaften eines Modells
- Als mathematisches Modell werden Kripke-Strukturen verwendet
- Eigenschaften des Modells lassen sich mittels temporaler Logiken (z.B. CTL) spezifizieren
- Es existieren Werkzeuge zur Überprüfung von UML-Statecharts

-  Die Ergebnisse der Verifikation gelten nur für das Modell und dürfen nicht bedenkenlos auf das reale System übertragen werden.

Vielen Dank für Ihr Interesse

?

?

Fragen?

?

?

?

?

?

?

?