

# Computergestützte Verifikation

29.4.03

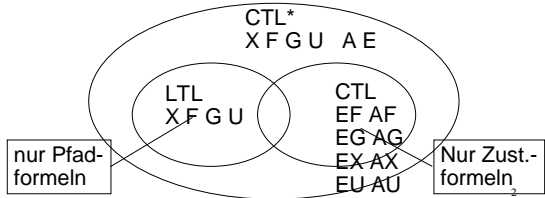
1

## CTL\* - Zusammenfassung

sehr ausdrucksstark

Keine effizienten Algorithmen bekannt

Es gibt effiziente Algorithmen für Fragmente von CTL\*



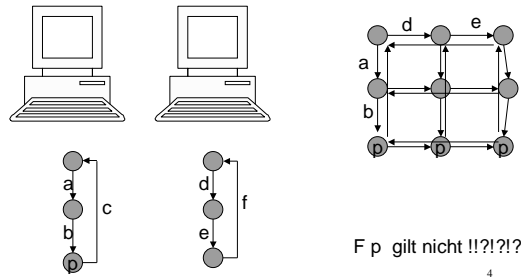
## Gegenbeispiele für A-Formeln

Gegenbeispiel für... = Zeugenpfad für...

AG $\phi$	EF $\neg \phi$
AF $\phi$	EG $\neg \phi$
(AX $\phi$ )	(EX $\neg \phi$ )
A( $\phi$ U $\psi$ )	{ EG $\neg \psi$ oder E( $\neg \psi$ U ( $\neg \phi \wedge \neg \psi$ ))

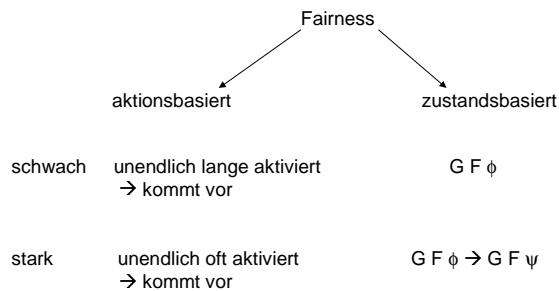
3

## Progress und Fairness



4

## Lösung: Fairnessannahmen



5

## Model Checking für finite state systems

explizit: Kapitel 3

symbolisch: Kapitel 4

- 3.1: Tiefensuche
- 3.2: LTL-Model Checking
- 3.3: CTL-Model Checking
- 3.4: Fairness
- 3.5: Reduktion durch Symmetrie
- 3.6: Partial Order Reduction
- 3.7: Tools

- 4.1: BDD-basiertes CTL-Model Checking
- 4.2: SAT-basiertes Model Checking
- 4.3: Tools

6

# Kapitel 3

## Explizites Model Checking

### 3.1. Tiefensuche

7

## Setting

Geg.: Gerichteter Graph  $[V, E]$

Ges.: stark zusammenhängende Komponenten

$v \sim v'$  gdw. Es gibt einen Weg von  $v$  nach  $v'$  und einen Weg von  $v'$  nach  $v$  in  $G$

$\sim$  ist Äquivalenzrelation; Klassen heißen SZK.

SZK können durch Tiefensuche ermittelt werden (Tarjan, '72)

## Einfache Tiefensuche

Annahme: Graph  $[V, E]$  zusammenhängend von  $v_0$

VAR schwarz, grau, weiss: Knotenmengen, dfs: Nat

schwarz := grau :=  $\emptyset$ , (weiss :=  $V$ ), maxdfs := 0  
dfs( $v_0$ ):

```
dfs(v)
  v.dfs = maxdfs; maxdfs += 1;
  (weiss := weiss \ {v}), grau := grau \cup {v};
  FOR ALL v' ([v,v'] ∈ E) DO
    IF v' ∈ weiss THEN
      dfs(v')
    END
  END
  END
  grau := grau \cup {v}; schwarz := schwarz \cup {v};
```

Invariant: weiss  $\cup$  grau  $\cup$  schwarz =  $V$ , schwarz  $\cup$  reach(grau) =  $V$

Ende: grau =  $\emptyset$

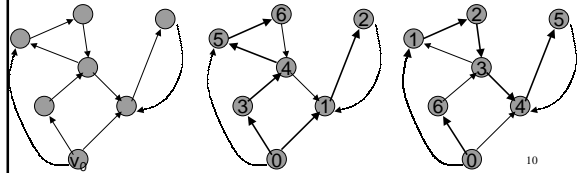
9

## Tiefensuchbaum

Tiefensuche definiert Numerierung der Knoten (dfs) und Tiefensuchbaum  $[V, T]$ :

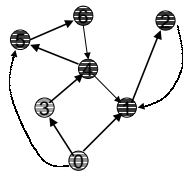
$[v, v'] \in T$  gdw. dfs( $v'$ ) wird von dfs( $v$ ) aus aufgerufen

Beispiel:



10

## SZK und Tiefensuchbaum



Jede SZK bildet einen zusammenhängenden Bereich im Tiefensuchbaum

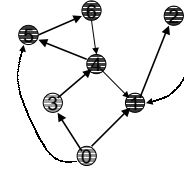
Wurzel des Teilbaums ist Knoten mit kleinster dfs.

$[v, v'] \in T \Rightarrow v.dfs < v'.dfs$

In jedem Teilbaum ist die Menge der Tiefensuchnummern lückenlos

11

## Klassifikation von E

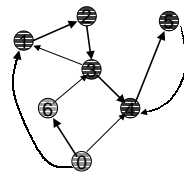


$[v, v']$  ist Baumkante, falls  $[v, v'] \in T$

$[v, v']$  ist Vorwärtskante, falls  $[v, v'] \in T^+ \setminus T$

$[v, v']$  ist Rückwärtskante, falls  $[v', v] \in T^+$

$[v, v']$  ist Querkante, sonst



$[v, v']$  in Quer  $\Rightarrow v.dfs > v'.dfs$

$[v, v']$  in Vorwärts  $\Rightarrow v.dfs \leq v'.dfs$

$[v, v']$  in Rückwärts  $\Rightarrow v.dfs > v'.dfs$

$[v, v']$  in Rückwärts  $\Rightarrow v \sim v'$

12

### Kriterium für Startknoten von SZK

$v.\text{lowlink} = \text{MIN}(v'.\text{dfs} \mid v' \text{ von } v \text{ erreichbar über beliebig viele Baumkanten, gefolgt von max. einer anderen Kante } [v, v'] \text{ mit } v \sim v')$

Satz:  $v$  ist genau dann Startknoten einer SZK wenn  $v.\text{lowlink} = v.\text{dfs}$

13

### Tarjans Algorithmus

VAR Tarj: Stack von Knoten, maxdfs: Nat, weiss: Knotenmenge  
 (weiss := V), maxdfs = 0; Tarj := empty stack  
 dfs( $v_0$ );

```

dfs(v):
  v.dfs = v.lowlink = maxdfs; maxdfs += 1;
  push(v, Tarj); {weiss := weiss ∪ {v}}
  FOR ALL  $v'$  ( $[v, v']$  in E) DO
    IF  $v'$  in weiss THEN
      dfs(v')
      v.lowlink = MIN(v.lowlink, v'.lowlink)  Baumkante
    ELSE
      IF  $v'$  on Tarj THEN
        v.lowlink = MIN(v.lowlink, v'.dfs)  andere Kante
      END
    END
  END
  IF v.lowlink = v.dfs THEN
    REPEAT
       $v^* = \text{pop}(\text{Tarj})$ 
    UNTIL  $v = v^*$ 
  } eine SZK
  END
  
```

14

### lowlink wird korrekt berechnet

... gefolgt von einer anderen Kante  $[v, v']$  mit  $v \sim v'$

```

...
ELSE
  IF  $v'$  on Tarj THEN
    v.lowlink = MIN(v.lowlink, v'.dfs)
  END
END
...
  
```

Fall 1:  $[v, v']$  Vorwärtskante: also  $v.\text{dfs} < v'.\text{dfs}$ , also trägt  $v'$  nicht zum Minimum bei

Fall 2:  $[v, v']$  Rückwärtskante: also  $v \sim v'$ ,  $v'$  vor  $v$  im Stack, also  $v'$  noch auf Tarjanstack

Fall 3:  $[v, v']$  Querkante: ok, siehe Tafelskizze

15

### Fazit Kapitel 3.1

Wir haben einen Algorithmus, der in  $O(|V| + |E|)$  die SZK eines gerichteten Graphen  $[V, E]$  bestimmt.

Dieser Algorithmus kann mit der Konstruktion des Transitionssystems verbunden werden:

```

FOR ALL  $[v, v']$  in E DO
  ...
  →
  FOR ALL commands  $g$  :  $g$  enabled in  $s$  DO
     $s' := \text{execute } g \text{ in } s$ 
    ...
    weiss  $\hat{=}$  noch nicht gesehen
  
```

16

## 3.2 LTL Model Checking

17

### Grundidee

LTL-Eigenschaft  $\hat{=}$  Menge derjenigen Pfade, die  $\phi$  erfüllen  
 $\rightarrow L_\phi^\omega$

Transitionssystem  $\rightarrow$  Menge derjenigen Pfade, die in TS realisiert werden können  
 $\rightarrow L_{\text{TS}}^\omega$

TS erfüllt  $\phi$  genau dann, wenn jeder Pfad in TS  $\phi$  erfüllt, d.h.

$$L_{\text{TS}}^\omega \subseteq L_\phi^\omega$$

18

## Wie kann man Inklusion entscheiden?

$$L_{TS}^\omega \subseteq L_\emptyset^\omega$$

$$\Leftrightarrow L_{TS}^\omega \cap \overline{L_\emptyset^\omega} = \emptyset$$

$$\Leftrightarrow L_{TS}^\omega \cap L_{-\emptyset}^\omega = \emptyset$$

- Agenda: 1. Automaten, die  $L_{TS}^\omega$  und  $L_{-\emptyset}^\omega$  akzeptieren  
 2. Konstruktion eines Schnittpunktsautomaten  
 3. Entscheidung, ob Automat die leere Sprache akzeptiert

Problem: Wir reden über unendliche Sequenzen!

19

## Büchi-Automaten

= endliche Automaten mit einem für unendliche Sequenzen geeigneten Akzeptierungskriterium

$$B = [X, Z, Z_0, \delta, F]$$

$X$  – Alphabet  
 $Z$  – Zustandsmenge  
 $Z_0$  – Anfangszustandsmenge  
 $\delta: Z \times X \rightarrow 2^Z$   
 $F = \{F_1, \dots, F_n\}, F_i \subseteq Z$  Akzeptierungsmengen

unendliche Sequenz  $\pi$  in  $X^\omega$

$B$  akzeptiert  $\pi$ : es ex. unendliche Sequenz  $\zeta = z_0 z_1 z_2 \dots$

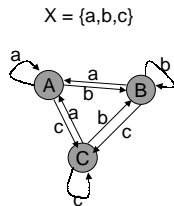
-  $z_0 \in Z_0, z_{i+1} \in \delta(z_i, x_i)$ ,

- Für jedes  $F_i \in F$ :  $\zeta$  enthält unendlich oft Elemente aus  $F_i$

$\rightarrow L_B$

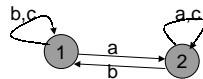
20

## Beispiele



$X = \{a, b, c\}$   
 $Z_0 = Z$   
 $F_1 = \{A, B\}$   
 $F_2 = \{A, C\}$

$L_B = \{\pi \mid \text{in } \pi \text{ unendlich oft a oder unendlich oft b und c}\}$



$Z_0 = \{1\}$   
 $F_1 = \{1\}$

$L_B = \{\pi \mid \text{nach jedem a in } \pi \text{ kommt irgendwann b}\}$

21

## Transitionssystem und Büchi-Automat

Eine Menge  $L^\omega$  von unendlichen Sequenzen heißt  $\omega$ -regulär, wenn es einen endlichen Büchi-Automaten gibt, der  $L^\omega$  akzeptiert.

Beobachtung: Jede Menge von Systemabläufen eines finite state Transitionssystems ist  $\omega$ -regulär.

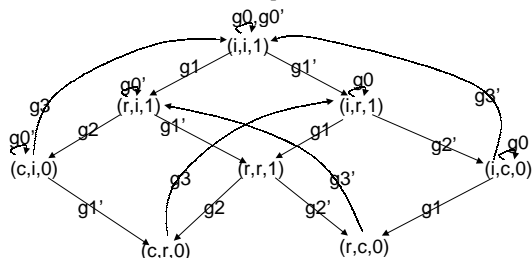
$X$  = Vektoren von booleschen Werten, die als Wahrheitswerte der elementaren Aussagen interpretiert werden

$$Z = S, Z_0 = I, \delta(z, x) = \begin{cases} \{z' \mid [z, z'] \in E\}, & \text{falls } x \text{ den Wahrheitswerten der Aussagen in } z \text{ entspricht} \\ \emptyset, & \text{sonst} \end{cases}$$

$F_1 = Z, F = \{F_1\}$

22

## Beispiele



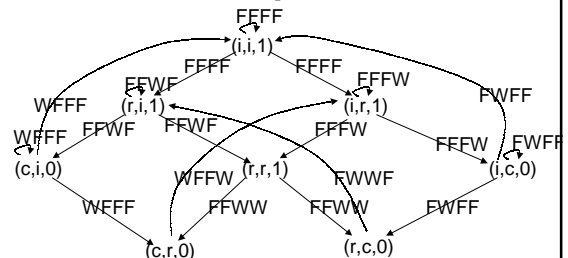
S:  $G(\text{pc1} \neq \text{"critical"} \vee \text{pc2} \neq \text{"critical"})$

L:  $G(\text{pc1} = \text{"request"} \Rightarrow F \text{pc1} = \text{"critical"})$

$G(\text{pc2} = \text{"request"} \Rightarrow F \text{pc2} = \text{"critical"})$

1. pc1 = crit
2. pc2 = crit
3. pc1 = req
4. pc2 = req

## Beispiele

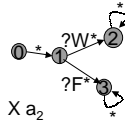


1. pc1 = crit
2. pc2 = crit
3. pc1 = req
4. pc2 = req

## Büchi-Automaten und LTL

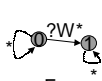
Satz: Zu jeder LTL-Formel  $\phi$  gibt es einen Büchi-Automaten mit  $\max 2^{|\phi|}$  Zuständen, der genau die unendlichen Sequenzen akzeptiert, die  $\phi$  erfüllen.

Beispiele:



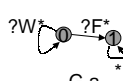
$X a_2$

$Z_0 = \{0\},$   
 $F = \{ \{2\} \}$



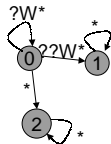
$F a_2$

$Z_0 = \{0\},$   
 $F = \{ \{1\} \}$



$G a_2$

$Z_0 = \{0\},$   
 $F = \{ \{0\} \}$



$a_2 U a_3$

$Z_0 = \{0\},$   
 $F = \{ \{1\} \}_{25}$