

1	Transitionssysteme und Verifikation	3
1.1	Transitionssysteme	3
1.2	Produkte von Transitionssystemen	9
1.3	Automaten und reguläre Sprachen	13
1.4	Kripkestrukturen	19
1.4.1	Verifikation und Model-Checking	19
1.4.2	Transitionssysteme in Form von <u>Kripke-Strukturen</u>	21
1.4.3	Kripke-Strukturen von Programmen	25
1.4.4	<u>Wechselseitiger Ausschluss</u>	27
1.5	Temporale Logik	31
1.5.1	Syntax und Semantik von <u>LTL-Formeln</u>	34
1.5.2	Syntax und Semantik von <u>CTL-</u> und CTL*-Formeln	37
1.5.3	<u>Faire Kripke-Struktur</u>	41
1.5.4	<u>CTL-Model-Checking</u>	42



Logic in Computer Science

Modelling and reasoning about systems

Michael R A Huth Mark D Ryan



Universität Hamburg / FB Informatik

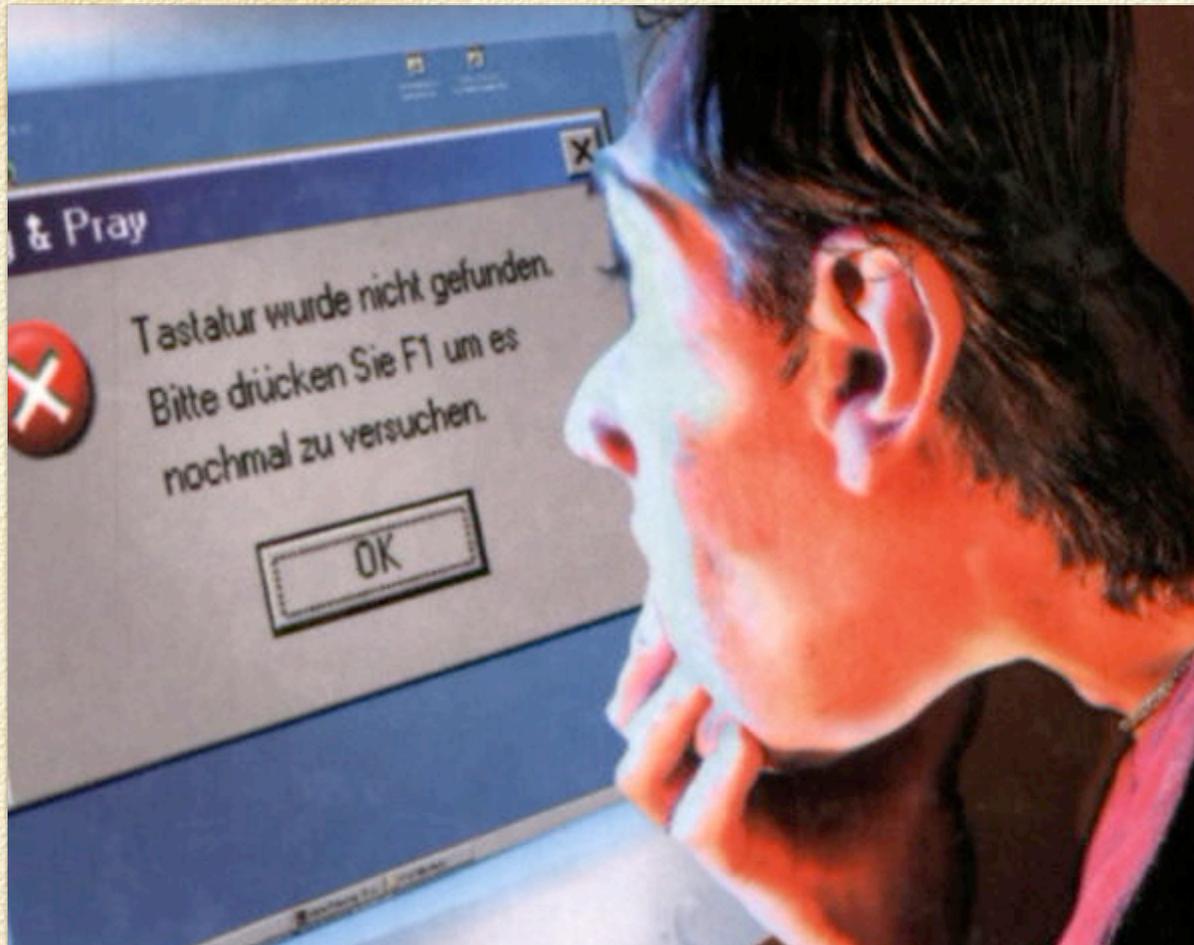


07583419 5

L
HUT
29351

Ariane 5

4. Juni 1996, Kourou / Frz. Guyana, ESA



117

0800 Antam started
 1000 " stopped - antam ✓

			{ 1.2700	9.037 847 025
	13 ⁰⁰ (032)	MP - MC	1.582147000	9.037 846 995 connect
	(033)	PRO 2	2.130476415	4.615925059(-2)
		connect	2.130676415	

Relays 6-2 in 033 failed special speed test
 in relay .. 11.000 test .

Relays changed

1100 Started Cosine Tape (Sine check)
 1525 Started Multi Adder Test.

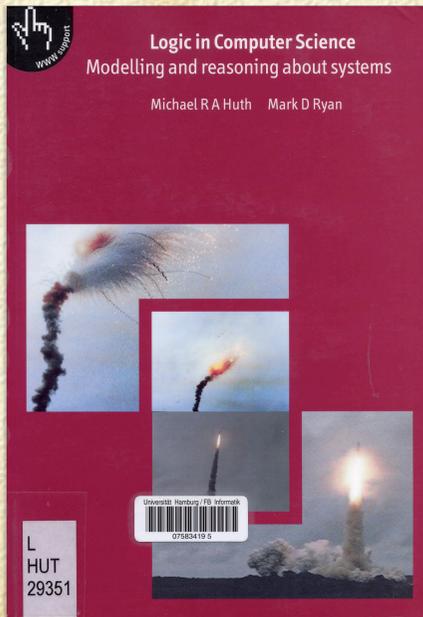
1545  Relay #70 Panel F
 (moth) in relay.

First actual case of bug being found.

~~1630~~ Antam started.
 1700 closed down.

Relay
2145
Relay 3370

Die Computer-Pionierin Grace Hopper hat Überlieferungen zufolge diesen Begriff zuerst verwendet. Sie hatte am 9. September 1945 eine Motte in einem Relais des Computers Mark II Aiken Relay Calculator gefunden, und das Tier führte zu einer Fehlfunktion. Grace Hopper entfernte die Motte und schrieb in ihr Logbuch: „**First actual case of bug being found**“, dazu klebte sie die im Relais gefundene Motte.



Ariane 5 – 501

4. Juni 1996, Kourou / Frz. Guyana, ESA

Jungferflug der neuen europäischen Trägerrakete (Gewicht: 740 t, Nutzlast 7 – 18 t) mit 4 Cluster-Satelliten; Entwicklungskosten in 10 Jahren: DM 11 800 Millionen

Wirkung: 37 Sekunden nach Zünden der Rakete (30 Sekunden nach Liftoff) erreichte Ariane 5 in 3700 m Flughöhe eine Horizontal-Geschwindigkeit von 32768.0 (interne Einheiten).

Dieser Wert lag etwa **fünfmal höher** als bei Ariane 4.

Die Umwandlung in eine ganze Zahl führte zu einem **Überlauf**, der jedoch nicht abgefangen wurde. Der Ersatzrechner (Redundanz !) hatte das gleiche Problem schon 72 msec vorher und schaltete sich sofort ab. Daraus resultierte, dass Diagnose-Daten zum Hauptrechner geschickt wurden, die dieser als Flugbahndaten interpretierte.

Daraufhin wurden unsinnige Steuerbefehle an die seitlichen, schwenkbaren Feststoff-Triebwerke, später auch an das Haupttriebwerk gegeben, um die großen Flugabweichungen (über 20 Grad) korrigieren zu können. Die Rakete drohte jedoch auseinanderzubrechen und **sprengte sich selbst (39 sec)**.

Ein intensiver Test des Navigations- und Hauptrechners wurde nicht unternommen, da die Software bei Ariane 4 erprobt war.

Schaden:

DM 250 Millionen Startkosten

DM 850 Millionen Cluster-Satelliten

DM 600 Millionen für nachfolgende Verbesserungen

Verdienstausschlag für 2 bis 3 Jahre

Der nächste Testflug konnte erst 17 Monate später durchgeführt werden – 1. Stufe beendete vorzeitig das Feuern.

Der erste kommerzielle Flug fand im Dezember 1999 statt.

Der Versuch, neue 4 Cluster-Satelliten zu starten, gelang im Juli und August 2000 mit zwei russischen Trägerraketen

Ursachen:

Der problematische Programmteil wurde nur für die Vorbereitung beim Start und den Start selbst gebraucht. Er sollte nur während einer Übergangszeit aktiv sein, aus Sicherheitsgründen: 50 sec, bis die Bodenstation bei einer Startunterbrechung die Kontrolle übernommen hätte.

Trotz des ganz anderen Verhaltens der Ariane 5 wurde dieser Wert nicht neu gesetzt.

Nur bei 3 von 7 Variablen wurde ein Überlauf geprüft – für die anderen 4 Variablen existierten Beweise, dass die Werte klein genug bleiben würden (Ariane 4).

Diese Beweise galten jedoch nicht für die Ariane 5 und wurden dafür auch gar nicht nachvollzogen.
Problem der Wiederverwendung von Software !

Mariner 1

22. Juli 1962, Cape Canaveral/Florida
Start der ersten amerikanischen Venussonde Mariner 1
Trägerrakete Atlas-Agena B (NASA, 15. AAB-Start)

Ausschnitt aus dem FORTRAN-Programm zur Steuerung der Flugbahn der Trägerrakete:

```
DO 5 K = 1. 3  
  ...  
5 CONTINUE
```

Entscheidender Fehler:

Punkt statt Komma!

Wirkung:

DO 5 K = 1.3

Wertzuweisung an eine nicht deklarierte Variable

Kein Durchlauf der (nicht vorhandenen) Schleife

Folge:

Abweichung der Trägerrakete von der vorgesehenen Flugbahn
Zerstörung der Rakete nach 290 Sekunden
Kosten dieser Wertzuweisung: \$ 18.5 Millionen

Ursache:

Schlechte Programmiersprache wegen Blanks (Zwischenräume) in Namen und Zahlen erlaubt
Variablen-Deklaration nicht notwendig.
Strukturierte Schleife (END DO) nicht möglich.

AT&T-Telefonnetz

15. Januar 1990:

70 Millionen von 138 Millionen Ferngesprächen innerhalb USA konnten 9 Stunden lang nicht vermittelt werden.

Schaden:

\$ 75 Millionen bei AT&T (ohne Folgeschäden)

Mehrere \$ 100 Millionen bei den Kunden (Versandhandel, Transportunternehmen, Reisebüros usw.)

Ursache:

Eine Schaltzentrale in New York setzte sich nach einer Fehlfunktion in den RESET-Modus:

- Ausfall-Meldung an alle anderen Zentralen
- Neubesetzung interner Tabellen (Reset)
- OK-Meldung an alle anderen Zentralen
- Weiterleiten neuer Ferngespräche

Alle Zentralen mußten daraufhin ihre Tabellen ändern.

Bei 3 Schaltzentralen kamen aber kurz nach der OK-Meldung mehrere neue Gespräche an

Verarbeitung der Meldung und der neuen Gespräche zerstörte Daten, was zum Rechnerausfall führte

Im Schneeball-System wurden 9 Stunden lang alle Zentralen lahmgelegt (zu 50% im RESET-Modus)

Notlösung: keine OK-Meldungen mehr verschicken

Eigentlicher Fehler wurde eine Woche später gefunden:

Der break-Befehl von C wurde falsch eingesetzt.

Existierte seit Programm-Optimierung 4 Wochen vorher.

1000 Zeilen	15 Blatt Papier
100 000 Zeilen	15 cm hoher Stapel
10 Millionen Zeilen	15 m hoher Stapel
200 Millionen Zeilen	300 m hoher Stapel

TeX 82

Handy

Hubble Bodensoftware

Luftraum-Überwachung

Space Shuttle, auch IIS

B-2 Stealth Bomber

Windows 95

Windows NT 4.0

Windows 2000

SDI, auch NMD (Schätzung)

Telefonsysteme USA

Programmieraufwand (kaum sprachabhängig)

mittlere Programmgröße 50 Zeilen pro Tag

große Programmgröße 10 – 20 Zeilen pro Tag

kritische Software 1 – 2 Zeilen pro Tag

Fehlerhäufigkeit

Normale Software 25 Fehler pro 1000 Zeilen

Wichtige Software 2 – 3 Fehler pro 1000 Zeilen

Medizinische Software 0.2 Fehler pro 1000 Zeilen

Space Shuttle Software < 0.1 Fehler pro 1000 Zeilen

25 – 100 Millionen Zeilen

200 Millionen Zeilen

Test – Verifikation – Validierung

Tests sehr zeitaufwendig

25 – 185 Stunden Testzeit für 1000 Zeilen

Keine Interpolation möglich (diskrete Werte)

nur die Anwesenheit von Fehlern ist testbar, nie deren Abwesenheit

Fehlende Programmteile nicht testbar

Fehler in nebenläufigen Programmen nicht reproduzierbar

Therac-25 Beschleuniger

(10 Geräte seit 1983 in USA und Kanada im Einsatz)

Software errechnet und steuert Bestrahlungs-Stärke entweder als Elektronenstrahlung (bis 25 MeV) oder als Röntgenstrahlung (mit 100-facher Elektronenstrahlung, Wolfram-Scheibe im Strahl, absorbiert Elektronen, Röntgenstrahlen entstehen)

Juni 1985, Georgia/USA

Elektronenstrahl brannte Loch durch Schulter

Krankenhaus lehnte Verschulden ab:

"Computer-gesteuert", "Hat schon immer funktioniert"

März 1986, Texas/USA

Patient erhielt zweimal Überdosis, Tod September 1986

Hersteller untersuchte das Problem, fand aber nichts

April 1986, Texas/USA

Fehler in derselben Klinik, Patient starb im Mai 1986

Zufällig gefundene Ursache:

Operator gab den Strahltyp falsch ein ("X-Ray")

Maschine schaltete um auf 100-fache Energie

Operator korrigierte Fehler in weniger als 8 Sekunden

Maschine nahm nur Wolfram-Scheibe aus Strahl heraus

Sicherheitsmechanismus schaltete nach 0.3 Sekunden ab

Notlösung: Abmontieren der Korrektur-Taste (Up-Key)

Korrektur im November 1986 (über 20 Modifikationen)

Space Shuttle 1 (Columbia), April 1981:

1. Startversuch der US-Raumfähre: Abbruch des Starts, da die 4 (alten) IBM-Bordrechner zu unterschiedlichen Ergebnissen kamen und sich nicht synchronisierten
Ursache: Eine Verzögerungsbedingung war zwei Jahre zuvor von 50 ms auf 80 ms geändert worden
In 2000 Stunden Testzeit wurden über 200 Fehler gefunden
Beim 1. Flug traten 24 weitere Fehler auf (Fehler-Manual!)

<http://www-aix.gsi.de/~giese/swr/allehtml.html>

Model Checking

Literatur:

E.M. Clarke et al.: *Model Checking*, The MIT Press, Cambridge, 1999, [CGP99]

B. Bèrard et al.: *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer, Berlin, 1999, [BBF99]

C. Girault, R. Valk: *Petri Nets for Systems Engineering, Part III: Verification*, Springer, Berlin, 2003, [GV03]

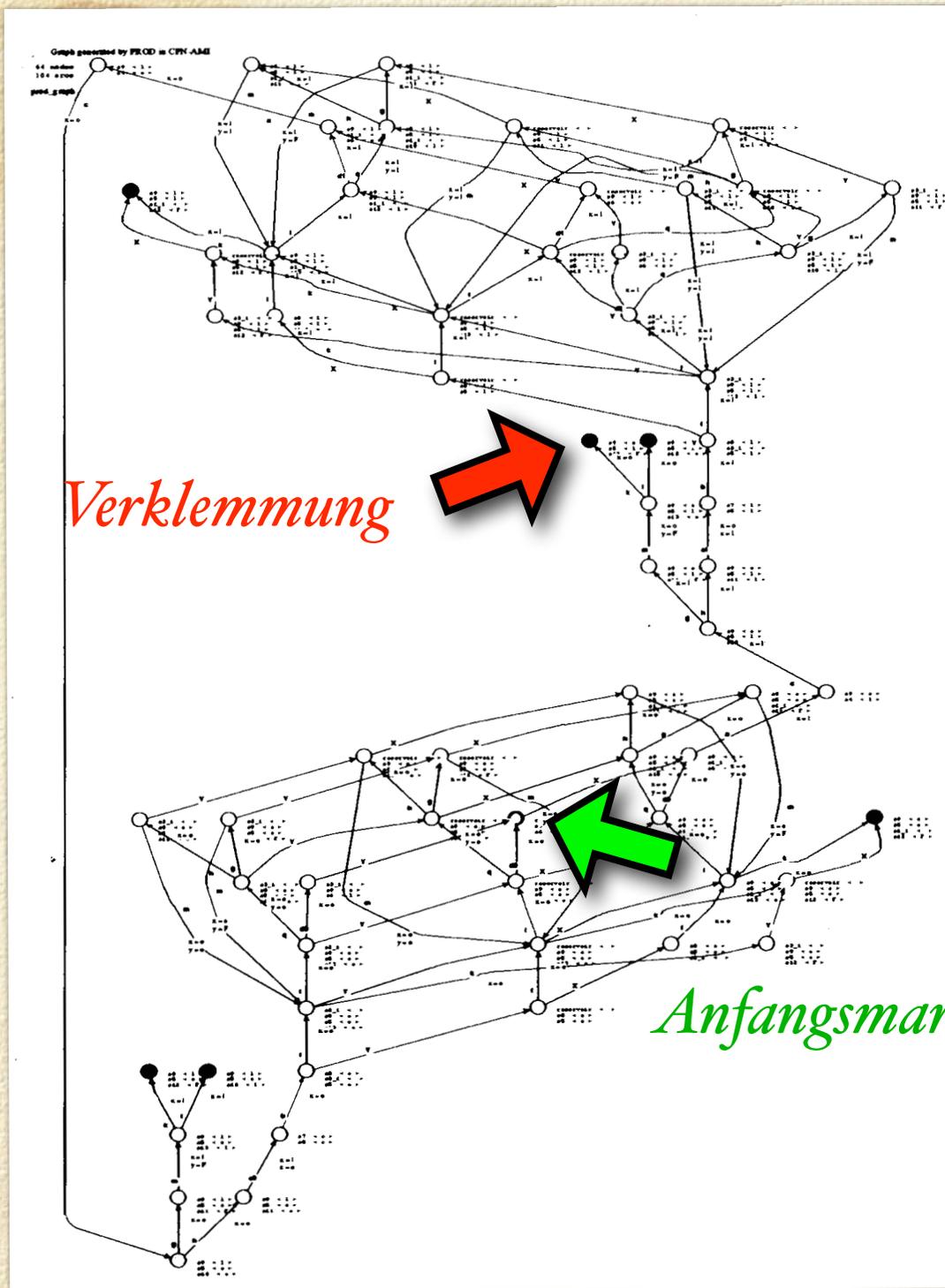
M. Huth, M. Ryan: *Logic in Computer Science*, Cambridge Univ. Press, 2004, [HR04]

C. Baier, J.-P. Katoen: *Principles of Model Checking*, MIT Press, 2008, [BK08]

1.4 Kripkestrukturen

1.4.1 Verifikation und Model-Checking

*Transitionssystem
eines
Alternierbitprotokolls
(aus einem Petrinetz)*



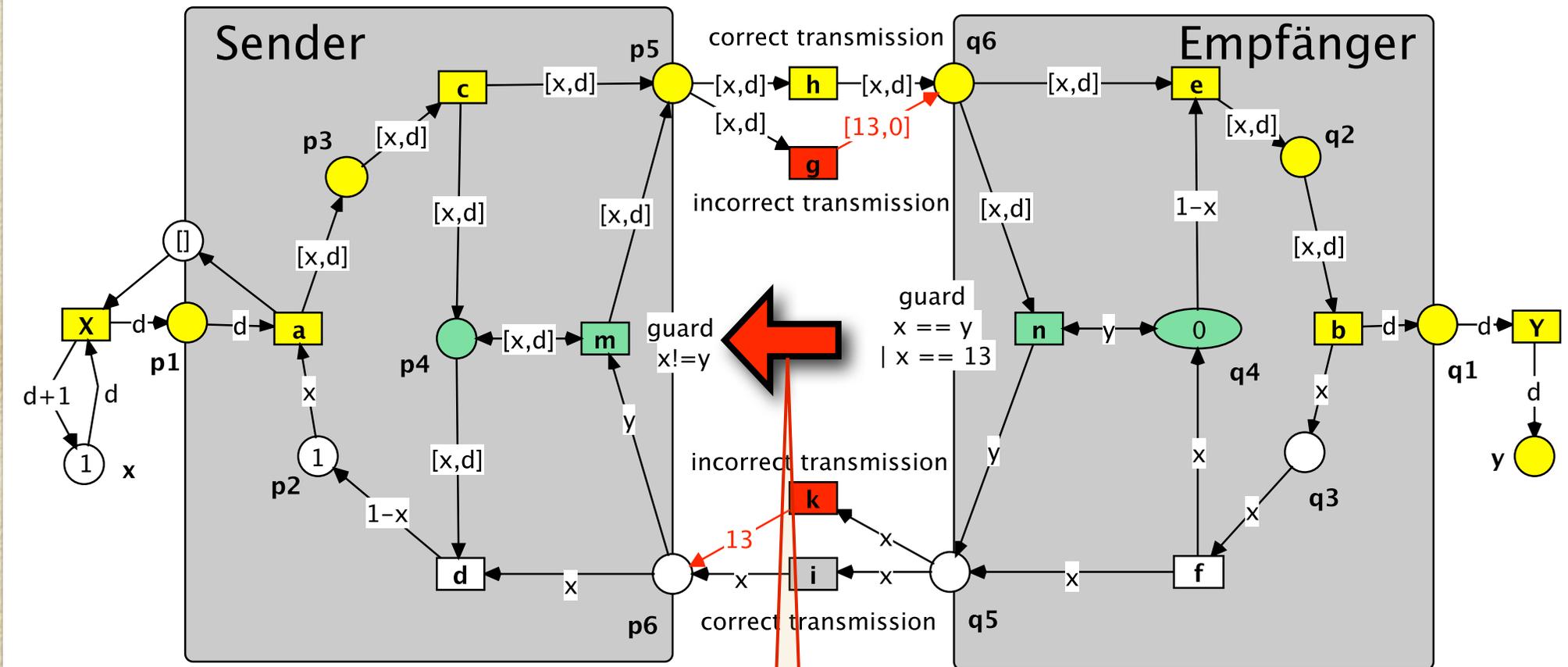


Abbildung 2.60: Das Alternierbitprotokoll abp-5

Hier lag der Fehler!

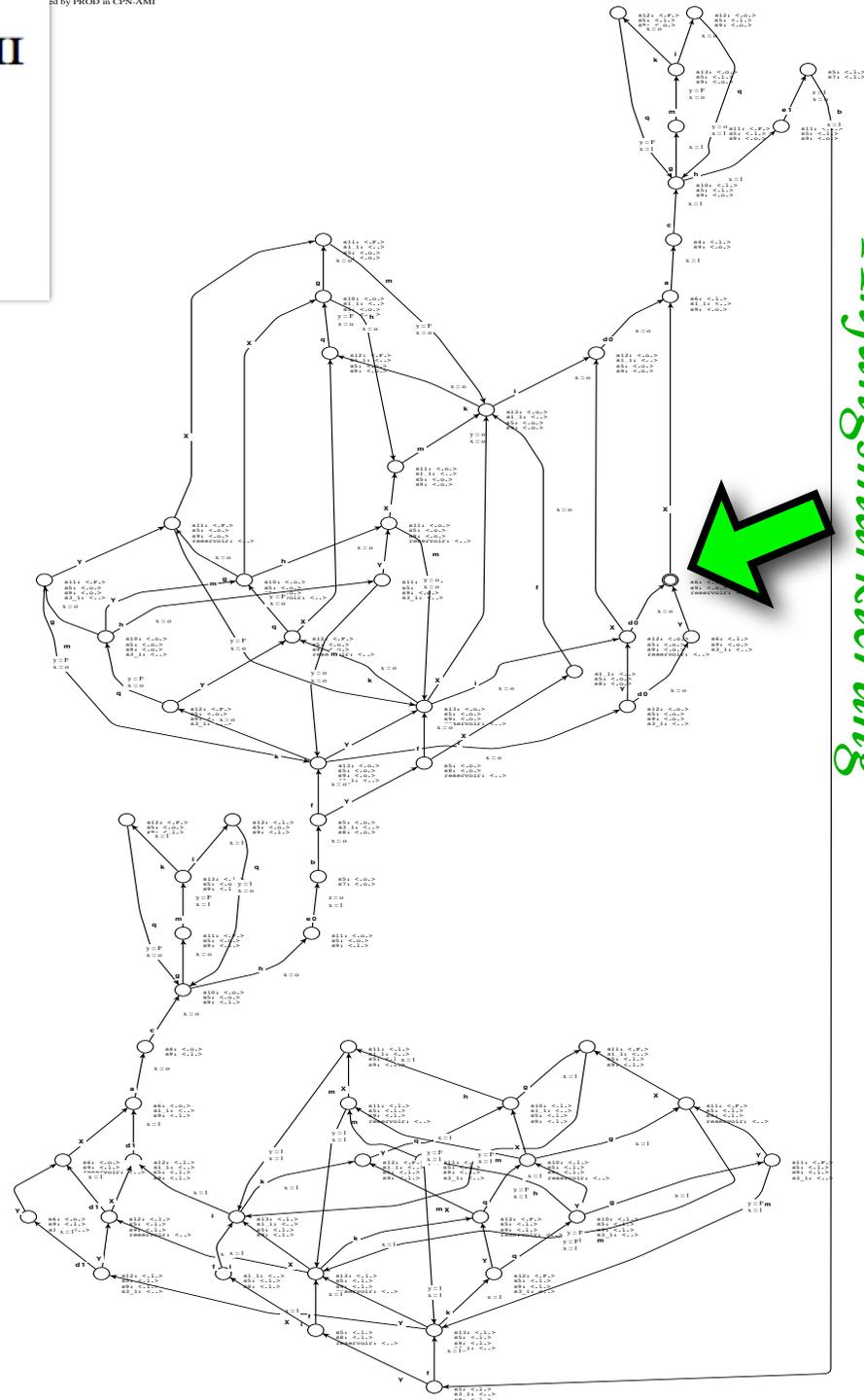
Graph generated by PROD in CPN-AMI

64 nodes
108 arcs

prod_graph

*Transitionssystem
eines
Alternierbitprotokolls
(aus einem Petrinetz)*

*keine
Verklemmung*



Anfangsmarkierung

1.4.2 Transitionssysteme in Form von Kripke-Strukturen

Definition 1.20 Sei $TS = (S, A, tr, S^0, S^F)$ ein Transitionssystem. Eine **Zustandsetikettenfunktion** ist eine Abbildung $E_S : S \rightarrow \mathcal{P}(AP)$, wobei AP eine Menge von atomaren Aussagen ist.

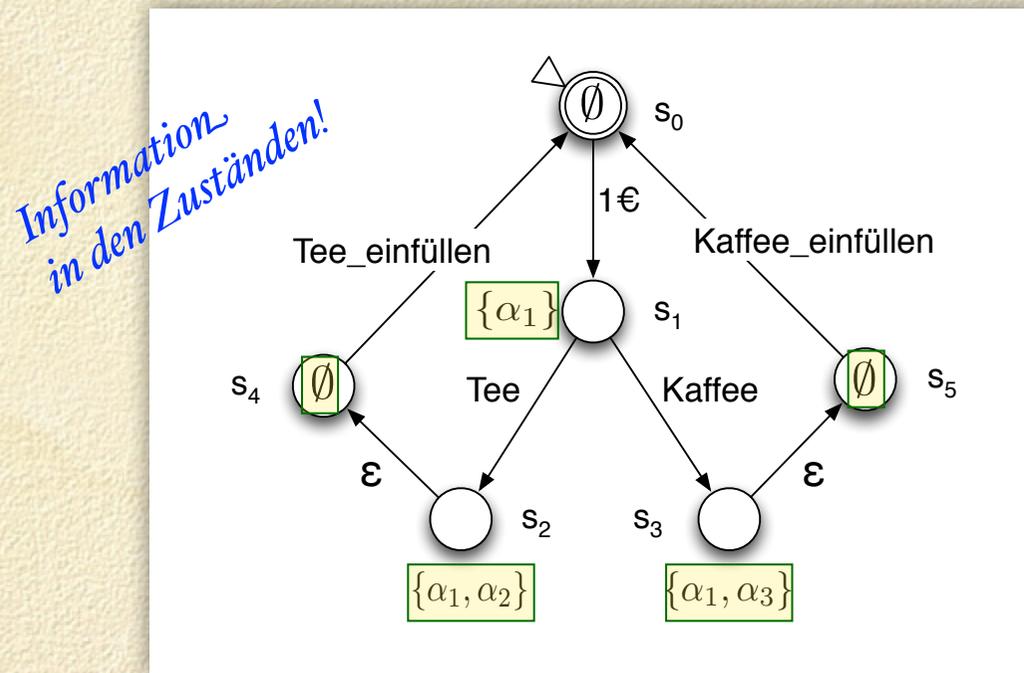


Abb. 1.12

$AP := \{\alpha_1, \alpha_2, \alpha_3\}$ durch $\alpha_1 = \text{„1€ gezahlt“}$,

$\alpha_2 = \text{„Tee gewählt“}$ und $\alpha_3 = \text{„Kaffee gewählt“}$

1.4.2 Transitionssysteme in Form von Kripke-Strukturen

Definition 1.20 Sei $TS = (S, A, tr, S^0, S^F)$ ein Transitionssystem. Eine **Zustandsetikettenfunktion** ist eine Abbildung $E_S : S \rightarrow \mathcal{P}(AP)$, wobei AP eine Menge von atomaren Aussagen ist.

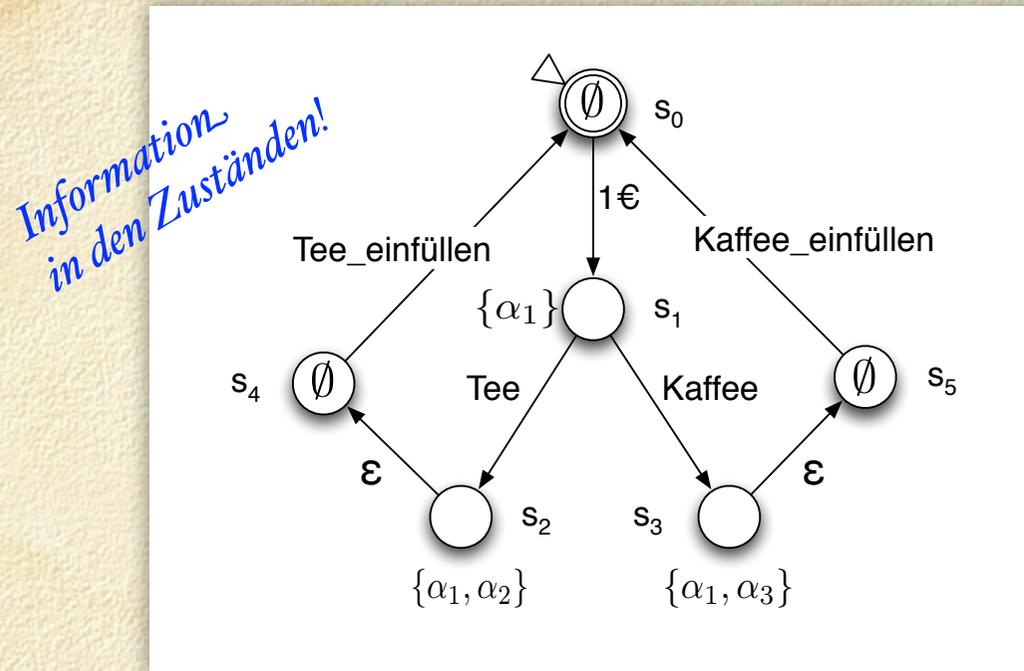


Abb. 1.12

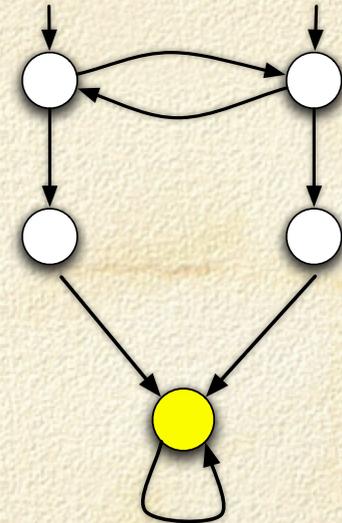
Definition 1.22 (Kripke-Struktur 1)

Eine Kripke-Struktur ist ein endliches Transitionssystem $M := (S, A, tr, S_0, \{\})$ ¹¹ mit einelementiger Aktionenmenge (also $|A| = 1$), linkstotaler¹² Transitionsrelation tr und zusätzlich einer Zustandsetikettenfunktion $E_S : S \rightarrow \mathcal{P}(AP)$.

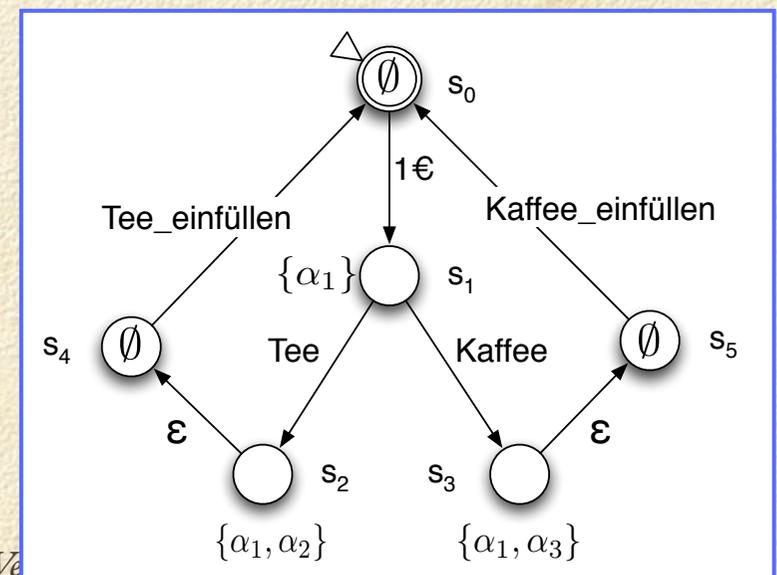
Definition 1.23 (Kripke-Struktur 2)

Eine Kripke-Struktur $M := (S, S_0, R, E_S)$ besteht aus

- einer endlichen Zustandsmenge S ,
- einer Menge $S_0 \subseteq S$ von Anfangszuständen,
- einer **linkstotalen** (Transitions-)Relation $R \subseteq S \times S$ und
- einer Zustandsetikettenfunktion $E_S : S \rightarrow \mathcal{P}(AP)$, die jedem Zustand s eine Menge $E_S(s) \subseteq AP$ von aussagenlogischen atomaren Formeln zuordnet (die in diesem Zustand gelten).



$$\forall s \in S \exists s' \in S : (s, s') \in R$$



$E_S(3) = \{\neg Start, Close, \neg Heat, \neg Error\}$ in der Kripke-Struktur von Abb. 5.5.

Kripke-Struktur

Mikrowellenofen

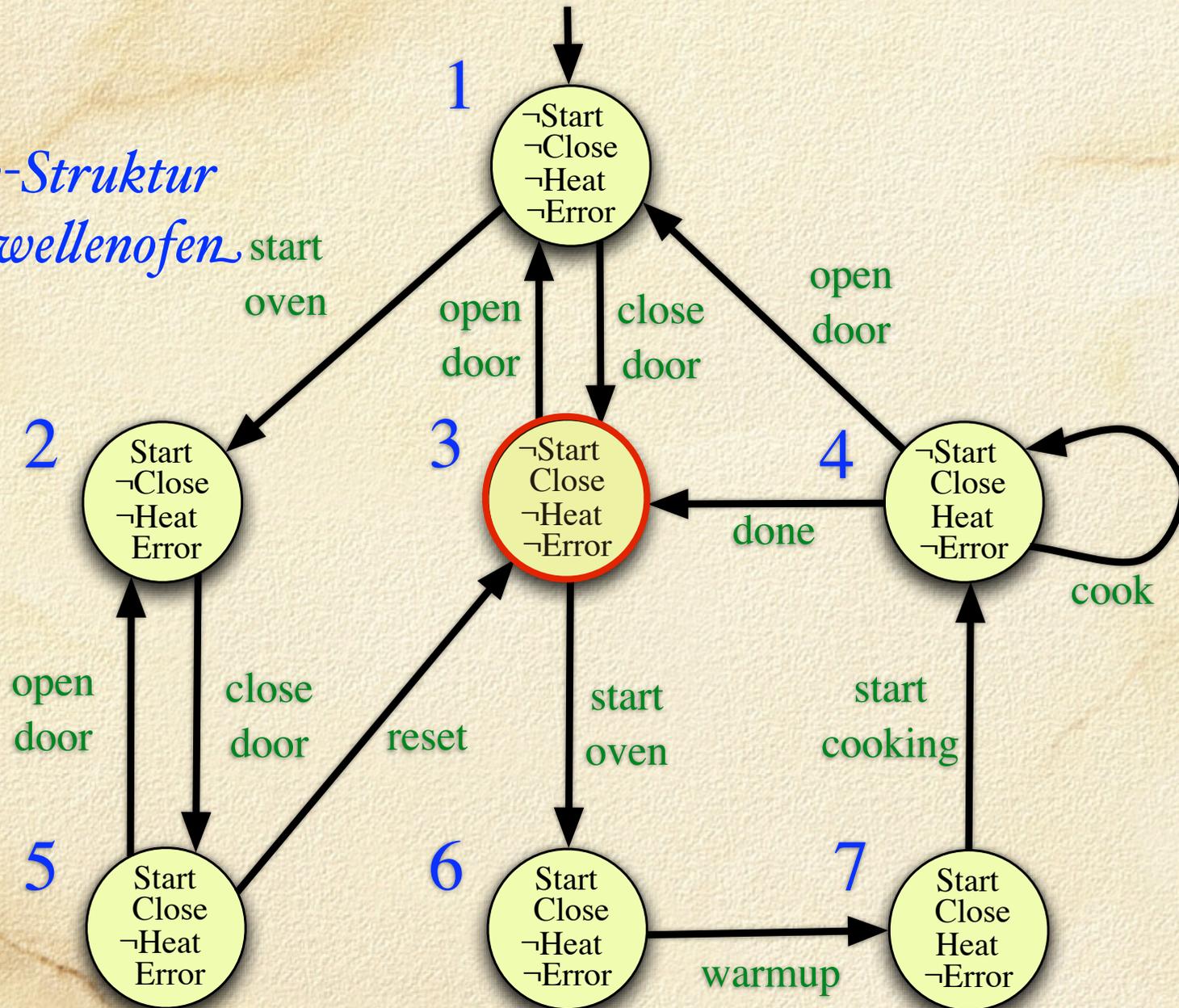


Abb. 1.16

Die Kripke-Struktur kann mittels Prädikaten erster Ordnung repräsentiert werden.

Belegung

Zustand: $s : V \rightarrow D$ ist eine Abbildung von der Menge der Variablen in eine Wertemenge. Z.B. für die Variablen Menge $V = \{v_1, v_2, v_3\}$, wird der Zustand $s = \langle v_1 \leftarrow 2, v_2 \leftarrow 3, v_3 \leftarrow 5 \rangle$ durch die zugehörige Formel $(v_1 = 2) \wedge (v_2 = 3) \wedge (v_3 = 5)$ repräsentiert. S_0 bezeichnet die Formel für den Anfangszustand.

Transition: Beziehung zwischen Werten der Variablen vor der Transition (Menge V) und Werten der Variablen nach der Transition (Menge V'). Wenn R eine Transitionsrelation ist, so bezeichnet $\mathcal{R}(V, V')$ die entsprechende Formel.

$$M := (S, S_0, \mathcal{R}, E_S)$$

Definition 1.26 Eine Kripke-Struktur in logischer Darstellung ist ein Tupel $M := (S, S_0, R, E_S)$ mit:

1. S Menge der Belegungen,
2. $S_0 \subseteq S$ Menge von Zuständen, die Anfangsbedingung S_0 erfüllen,
3. $\forall s, s' \in S : R(s, s') \leftrightarrow \mathcal{R}(V, V')$ mit Belegung s für V und s' für V' ,
4. $E_S(s)$ Menge der Formeln, die bei Belegung s gelten.

$(v \in E_S(s) \text{ gdw. } s(v) = \text{wahr})$

*Belegung = Werte
der Variablen*

Beispiel 1.27 (System mit Kripke-Struktur)

Variablenmenge: $V = \{x, y\}$,

Wertemenge: $D = \{0, 1\}$,

System: $x := (x + y) \bmod 2$,

Anfangszustand: $x = 1, y = 1$,

Das System kann mit zwei Prädikatenformeln beschrieben werden:

$$\mathcal{S}_0(x, y) \equiv x = 1 \wedge y = 1$$

$$\mathcal{R}(x, y, x', y') \equiv x' = (x + y) \bmod 2 \wedge y' = y$$

$$S_0(x, y) \equiv x = 1 \wedge y = 1$$

$$\mathcal{R}(x, y, x', y') \equiv x' = (x + y) \bmod 2 \wedge y' = y$$

Daraus die Kripke-Struktur: $M = (S, S_0, R, L)$

$$S = D \times D,$$

$$S_0 = \{(1, 1)\},$$

$$R = \{((1, 1), (0, 1)), ((0, 1), (1, 1)), ((1, 0), (1, 0)), ((0, 0), (0, 0))\},$$

$$L((1, 1)) = \{x = 1, y = 1\}, \dots$$

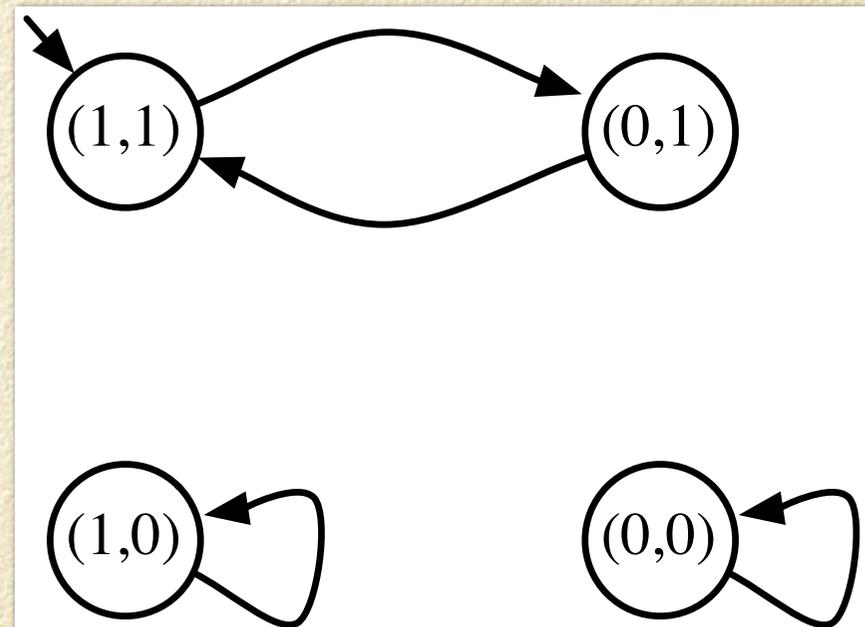


Abb. 1.13

1.4.3 Kripke-Strukturen von Programmen

Sequentielle Programme als Kripke-Strukturen

- a) Zuweisung
- b) Skip
- c) Hintereinanderausführung
- d) bedingte Anweisung
- e) Schleife

Zur Kennzeichnung von Zuständen erhalten die Programme Zeilennummern.

Für Zeilennummern gibt es die Variable pc („Befehlszähler“). Mit $pc = \perp$ ist gemeint, dass das Programm nicht aktiv ist.

Prädikat $same(Y) \equiv \forall y \in Y : (y' = y) \quad (Y \subseteq V)$

Prädikat $pre(V)$ beschreibt die Anfangsbelegung der Variablen V .

Anfangszustand: $\mathcal{S}_0 \equiv pre(V) \wedge pc = m$, mit m als Startzeilennummer.

Anfangswert $x = 1, y = 2$.

$l_1 : x := 2 \cdot y;$

$l_2 : y := y - 1;$

l_3

$\mathcal{S}_0 \equiv (x = 1 \wedge y = 2 \wedge pc = l_1)$

Beispiel 1.28 auf Seite 28

Die Prozedur $\mathcal{C}(l, P, l')$ liefert für eine Programmbeschreibung P die Repräsentation \mathcal{R} der Transitionen des Programms als Disjunktion der prädikatenlogischen Formeln.

l, l' Zeilennummer jeweils vor und nach der Anweisung

pc, pc' Befehlszähler - Variable

Anfangswert $x = 1, y = 2$.

$l_1 : x := 2 \cdot y;$

$l_2 : y := y - 1;$

l_3

$\mathcal{S}_0 \equiv (x = 1 \wedge y = 2 \wedge pc = l_1)$

Beispiel 1.28 auf Seite 28

Zuweisung:

$$C(l, v \leftarrow e, l') \equiv pc = l \wedge pc' = l' \wedge v' = e \wedge \text{same}(V \setminus \{v\})$$

Hintereinanderausführung:

$$C(l, (P_1; l'' : P_2), l') \equiv C(l, P_1, l'') \vee C(l'', P_2, l')$$



Anfangswert $x = 1, y = 2$.

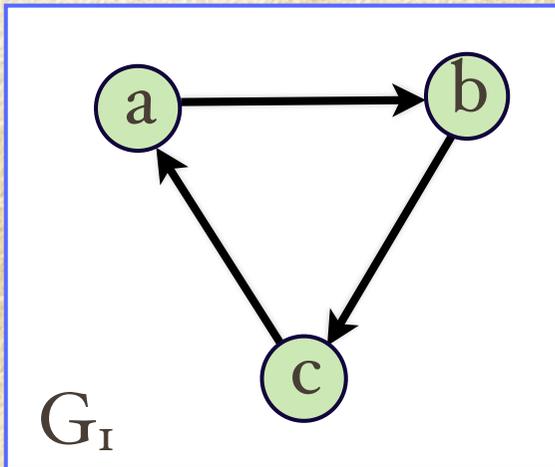
$l_1 : x := 2 \cdot y;$

$l_2 : y := y - 1;$

l_3

$$\mathcal{S}_0 \equiv (x = 1 \wedge y = 2 \wedge pc = l_1)$$

Beispiel 1.28 auf Seite 28



$$G_1 = \{(x,y) \mid (x,y) = (a,b) \vee (x,y) = (b,c) \vee (x,y) = (c,a) \}$$

$$G_2 = \{(x,y) \mid (x,y) = (a,b) \wedge (x,y) = (b,c) \wedge (x,y) = (c,a) \}$$

Beispiel 5.9 (ein kleines sequentielles Programm)

Anfangswert $x = 1, y = 2$.

$l_1 : x := 2 \cdot y;$

$l_2 : y := y - 1;$

l_3

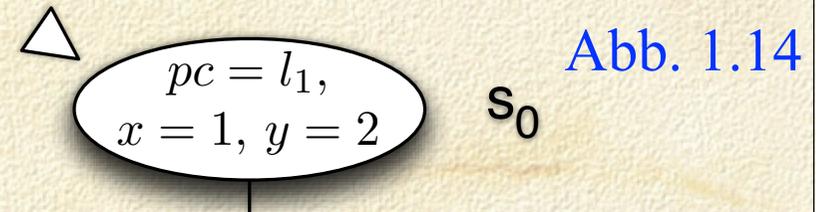


Abb. 1.14

Anfangszustand: $S_0 \equiv (x = 1 \wedge y = 2 \wedge pc = l_1)$

Beispiel 1.28 auf Seite 28

Beispiel 1.28 (ein kleines sequentielles Programm)

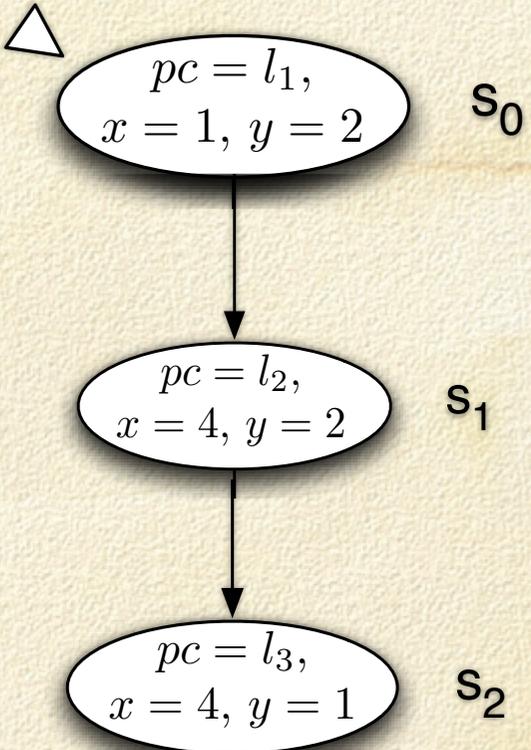
Anfangswert $x = 1, y = 2$.

$l_1 : x := 2 \cdot y;$

$l_2 : y := y - 1;$

l_3

Abb. 1.14



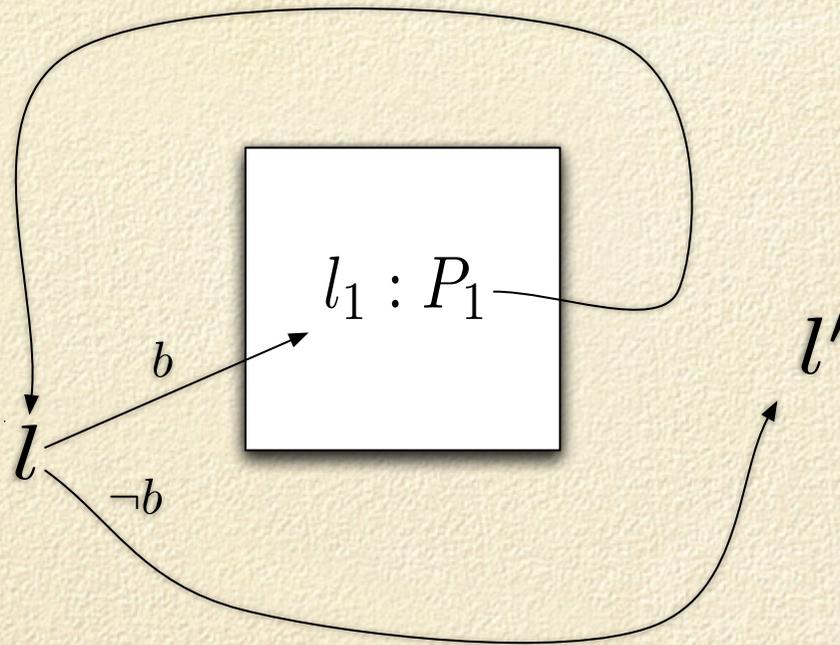
Transitionsrelation: $\mathcal{R} \equiv \mathcal{C}(l_1, x \leftarrow 2 \cdot y, l_2) \vee \mathcal{C}(l_2, y \leftarrow y - 1, l_3)$

mit $\mathcal{C}(l_1, x \leftarrow 2 \cdot y, l_2) \equiv pc = l_1 \wedge pc' = l_2 \wedge x' = 2 \cdot y \wedge y' = y$

und $\mathcal{C}(l_2, y \leftarrow y - 1, l_3) \equiv pc = l_2 \wedge pc' = l_3 \wedge y' = y - 1 \wedge x' = x$

Schleifen-Anweisung:

$$\begin{aligned} \mathcal{C}(l, \text{ while } b \text{ do } l_1 : P_1 \text{ endwhile, } l') &\equiv \\ & (pc = l \wedge pc' = l_1 \wedge b \wedge \text{same}(V)) \vee \\ & (pc = l \wedge pc' = l' \wedge \neg b \wedge \text{same}(V)) \vee \\ & \mathcal{C}(l_1, P_1, l) \end{aligned}$$

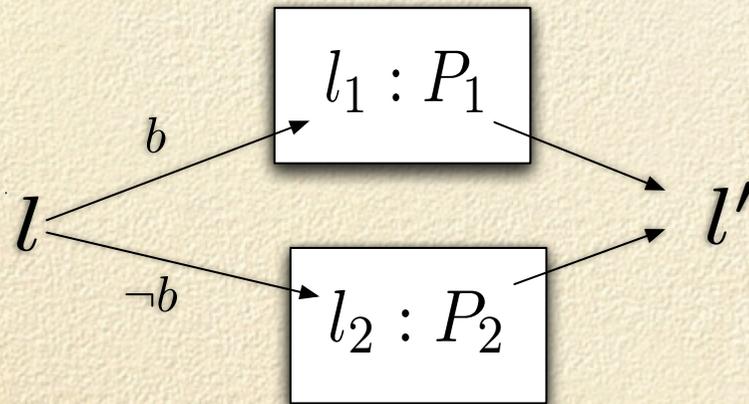


Skip:

$$C(l, \text{skip}, l') \equiv pc = l \wedge pc' = l' \wedge \text{same}(V)$$

Bedingte Anweisung:

$$C(l, \text{if } b \text{ then } l_1 : P_1 \text{ else } l_2 : P_2 \text{ endif}, l') \equiv \\ (pc = l \wedge pc' = l_1 \wedge b \wedge \text{same}(V)) \vee \\ (pc = l \wedge pc' = l_2 \wedge \neg b \wedge \text{same}(V)) \vee \\ C(l_1, P_1, l') \vee C(l_2, P_2, l')$$



Nebenläufige Programme als Kripke-Strukturen

Programmbeschreibung:

$$P = \mathbf{cobegin} P_1 \parallel P_2 \parallel \dots \parallel P_n \mathbf{coend}$$

Mit Zeilennummern:

$$P^{\mathcal{L}} = \mathbf{cobegin} l_1 : P_1^{\mathcal{L}} l'_1 \parallel \dots \parallel l_n : P_n^{\mathcal{L}} l'_n; \mathbf{coend}$$

Daraus die Formeln, wobei $PC = \{pc, pc_i \mid pc_i\text{-Befehlszähler von } P_i\}$):

$$S_0(V, PC) \equiv pre(V) \wedge pc = m \wedge \bigwedge_{i=1}^n (pc_i = \perp)$$

P_i sind also am Anfang nicht aktiv. Damit ergibt sich folgende Repräsentation:

$P^{\mathcal{L}} = \mathbf{cobegin} \ l_1 : P_1^{\mathcal{L}} \ l'_1 \parallel \dots \parallel l_n : P_n^{\mathcal{L}} \ l'_n ; \mathbf{coend}$

$\mathcal{C}(l, P^{\mathcal{L}}, l') \equiv$

$(pc = l \wedge pc'_1 = l_1 \wedge \dots \wedge pc'_n = l_n \wedge pc' = \perp) \vee$

Initialisierung

$(pc = \perp \wedge pc_1 = l'_1 \wedge \dots \wedge pc_n = l'_n \wedge pc' = l' \wedge \bigwedge_{i=1}^n (pc'_i = \perp)) \vee$

Termination

$(\bigvee_{i=1}^n (\mathcal{C}(l_i, P_i, l'_i) \wedge \mathit{same}(V \setminus V_i) \wedge \mathit{same}(PC \setminus \{pc_i\})))$

Transition von P_i

await b

warte bis b gilt!

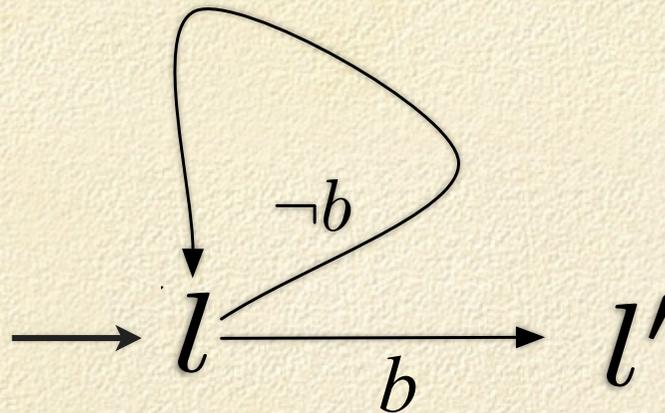
await-Anweisung:

$$\begin{aligned} C(\underline{l}, \text{await}(b), \underline{l'}) &\equiv \\ (pc_i = l \wedge pc'_i = \underline{l} \wedge \neg b \wedge \text{same}(V_i)) & \\ \vee (pc_i = l \wedge pc'_i = \underline{l'} \wedge \underline{b} \wedge \text{same}(V_i)) & \end{aligned}$$

“busy waiting”

b = false

b = true



Beispielprogramm für wechselseitigen Ausschluss

$P = m : \text{cobegin } P_0 \parallel P_1 \text{ coend}$



Parallele Programme oder Prozesse können zum konsistenten Schreiben auf gemeinsame Daten einen „kritischen Abschnitt“ enthalten, der nicht überlappend ausgeführt werden darf. Dies kommt im „nicht-kritischen Abschnitt“ nicht vor.

Die Programme sollen dabei (für den Fall zweier Prozesse P und Q) folgende Eigenschaften erfüllen:

A) Die Befehlszähler von P und Q sind **nie gleichzeitig** in ihren kritischen Abschnitten. *Markierungs-Invarianz*
safety property **Es gilt immer ...**

B) Meldet der Prozess P oder Q den Wunsch zum Eintritt in den kritischen Abschnitt an ($wantP = True$ oder $wantQ = True$), so **kann** er nach einer gewissen endlichen Zeit **tatsächlich** in seinen kritischen Abschnitt **eintreten**.

Lebendigkeits-Invarianz
liveness property

Es gilt später einmal ...

allgemeines Programm-Schema

Initialisierung

$m : \mathbf{cobegin} P \parallel Q \mathbf{coend}$

wobei

$P: l_0 : \mathbf{while} \text{ True } \mathbf{do}$

$p_i : \text{non-critical section};$

Eintrittsprotokoll

$p_j : \text{critical section};$

Austrittsprotokoll

$\mathbf{endwhile} l'_0$

$Q: l_1 : \mathbf{while} \text{ True } \mathbf{do}$

$q_i : \text{non-critical section};$

Eintrittsprotokoll

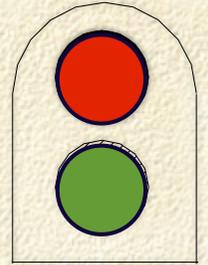
$q_j : \text{critical section};$

: Austrittsprotokoll

$\mathbf{endwhile} l'_1$

wechselseitiger Ausschluss I

(nach Dijkstra)



Ampel = grün: {rot, grün},
m : **cobegin** *P* || *Q* **coend**
wobei

P: l_0 : **while** True **do**
 ➔ p_0 : non-critical section;
 p_2 : **await**(*Ampel* = grün);
 p_3 : *Ampel* := rot;
 p_4 : critical section;
 p_5 : *Ampel* := grün;
 endwhile l'_0

Q: l_1 : **while** True **do**
 ➔ q_0 : non-critical section;
 q_2 : **await**(*Ampel* = grün);
 q_3 : *Ampel* := rot;
 q_4 : critical section;
 q_5 : *Ampel* := grün;
 endwhile l'_1

kein WA!

wechselseitiger Ausschluss 2

(nach Dijkstra)

$wantP = wantQ = False : boolean,$
 $m : cobegin P || Q coend$
wobei

$P: l_0 : while True do$
  $p_0 : non-critical\ section;$
 $p_1 : wantP := True;$
 $p_3 : await(wantQ = False)$
  $p_4 : critical\ section;$
 $p_5 : wantP := False;$
 $endwhile l'_0$

$Q: l_1 : while True do$
  $q_0 : non-critical\ section;$
 $q_1 : wantQ := True;$
 $q_3 : await(wantP = False)$
  $q_4 : critical\ section;$
 $q_5 : wantQ := False;$
 $endwhile l'_1$

nicht verklemmungsfrei!

wechselseitiger Ausschluss 3

(nach Dijkstra)

$wantP = wantQ = False : boolean,$
 $m : cobegin P || Q coend$
wobei

$P:$ $l_0 : while True do$
 $p_0 : non\text{-}critical\ section;$
 $p_1 : wantP := True;$
 $p_3 : while wantQ = True do$
 $\quad wantP := False;$
 $\quad wantP := True$
 $endwhile$
 $p_4 : critical\ section;$
 $p_5 : wantP := False;$
 $endwhile\ l'_0$

$Q:$ $l_1 : while True do$
 $q_0 : non\text{-}critical\ section;$
 $q_1 : wantQ := True;$
 $p_3 : while wantP = True do$
 $\quad wantQ := False;$
 $\quad wantQ := True$
 $endwhile$
 $q_4 : critical\ section;$
 $q_5 : wantQ := False;$
 $endwhile\ l'_1$

livelock! „after you - after you“ - Effekt

↯wechselseitiger Ausschluss 2 + tie break rule

(nach Dijkstra)

Wer sich zuletzt anmeldet, muss warten!

➡ neue Variable last
mit Werten 1 und 2

wantP = wantQ = False : boolean,
m : cobegin P||Q coend
wobei

P: l₀ : while True do
➡ p₀ : non-critical section;
p₁ : wantP := True;
p₃ : await(wantQ = False)
p₄ : critical section;
p₅ : wantP := False;
endwhile l'₀

Q: l₁ : while True do
➡ q₀ : non-critical section;
q₁ : wantQ := True;
q₃ : await(wantP = False)
q₄ : critical section;
q₅ : wantQ := False;
endwhile l'₁

nicht verklemmungsfrei!

wechselseitiger Ausschluss 4

(nach Dijkstra/Dekker/Peterson)

$wantP = wantQ = False : boolean,$
 $last = 1 \vee last = 2 : integer,$
 $m : cobegin P || Q coend$
wobei

$P: l_0 : while True do$
 ~~$p_0 : non\ critical\ section;$~~
 $p_1 : wantP := True;$
 $p_2 : last := 1;$
 $p_3 : await(wantQ = False$
 $\vee last = 2);$
 ~~$p_4 : critical\ section;$~~
 $p_5 : wantP := False;$
 $endwhile l'_0$

$Q: l_1 : while True do$
 ~~$q_0 : non\ critical\ section;$~~
 $q_1 : wantQ := True;$
 $q_2 : last := 2;$
 $q_3 : await(wantP = False$
 $\vee last = 1);$
 ~~$q_4 : critical\ section;$~~
 $q_5 : wantQ := False;$
 $endwhile l'_1$

ok! Prüfen durch Model Checking

$wantP = wantQ = False : boolean,$
 $last = 1 \vee last = 2 : integer,$
 $m : \mathbf{cobegin} P \parallel Q \mathbf{coend}$
 wobei

$P: l_0 : \mathbf{while} \text{ True do}$
 ~~$[p_0 : \text{non-critical section};]$~~
 $p_1 : wantP := True;$
 $p_2 : last := 1;$
 $p_3 : \mathbf{await}(wantQ = False$
 $\quad \vee last = 2);$
 ~~$[p_4 : \text{critical section};]$~~
 $p_5 : wantP := False;$
 $\mathbf{endwhile} l'_0$

$Q: l_1 : \mathbf{while} \text{ True do}$
 ~~$[q_0 : \text{non-critical section};]$~~
 $q_1 : wantQ := True;$
 $q_2 : last := 2;$
 $q_3 : \mathbf{await}(wantP = False$
 $\quad \vee last = 1);$
 ~~$[q_4 : \text{critical section};]$~~
 $q_5 : wantq := False;$
 $\mathbf{endwhile} l'_1$

Im folgenden ist das (reduzierte) Programm in der zuvor eingeführten Notation dargestellt.

$PC = \{pc, pc_0, pc_1\}, \quad V = V_0 = V_1 = \{wantP, wantQ, last\}$

pc_0 nimmt die Werte $\{p_1, p_2, p_3, p_5\}$ an und pc_1 die Werte $\{q_1, q_2, q_3, q_5\}$.

Der Anfangszustand ist:

$$S_0(V, PC) \equiv pc = m \wedge pc_0 = \perp \wedge pc_1 = \perp$$

Die Transitionsrelation ist repräsentiert durch $\mathcal{R}(V, PC, V', PC')$ als Disjunktion der folgenden Formeln:

- $pc = m \wedge pc'_0 = l_0 \wedge pc'_1 = l_1 \wedge pc' = \perp$
- $pc_0 = l'_0 \wedge pc_1 = l'_1 \wedge pc' = m' \wedge pc'_0 = \perp \wedge pc'_1 = \perp$
- $\mathcal{C}(l_0, P, l'_0) \wedge \text{same}(V \setminus V_0) \wedge \text{same}(PC \setminus \{pc_0\})$ also $\mathcal{C}(l_0, P, l'_0) \wedge \text{same}(pc, pc_1)$
- $\mathcal{C}(l_1, Q, l'_1) \wedge \text{same}(V \setminus V_1) \wedge \text{same}(PC \setminus \{pc_1\})$ also $\mathcal{C}(l_1, Q, l'_1) \wedge \text{same}(pc, pc_0)$

$wantP = wantQ = False : boolean,$
 $last = 1 \vee last = 2 : integer,$
 $m : \mathbf{cobegin} P \parallel Q \mathbf{coend}$
 wobei

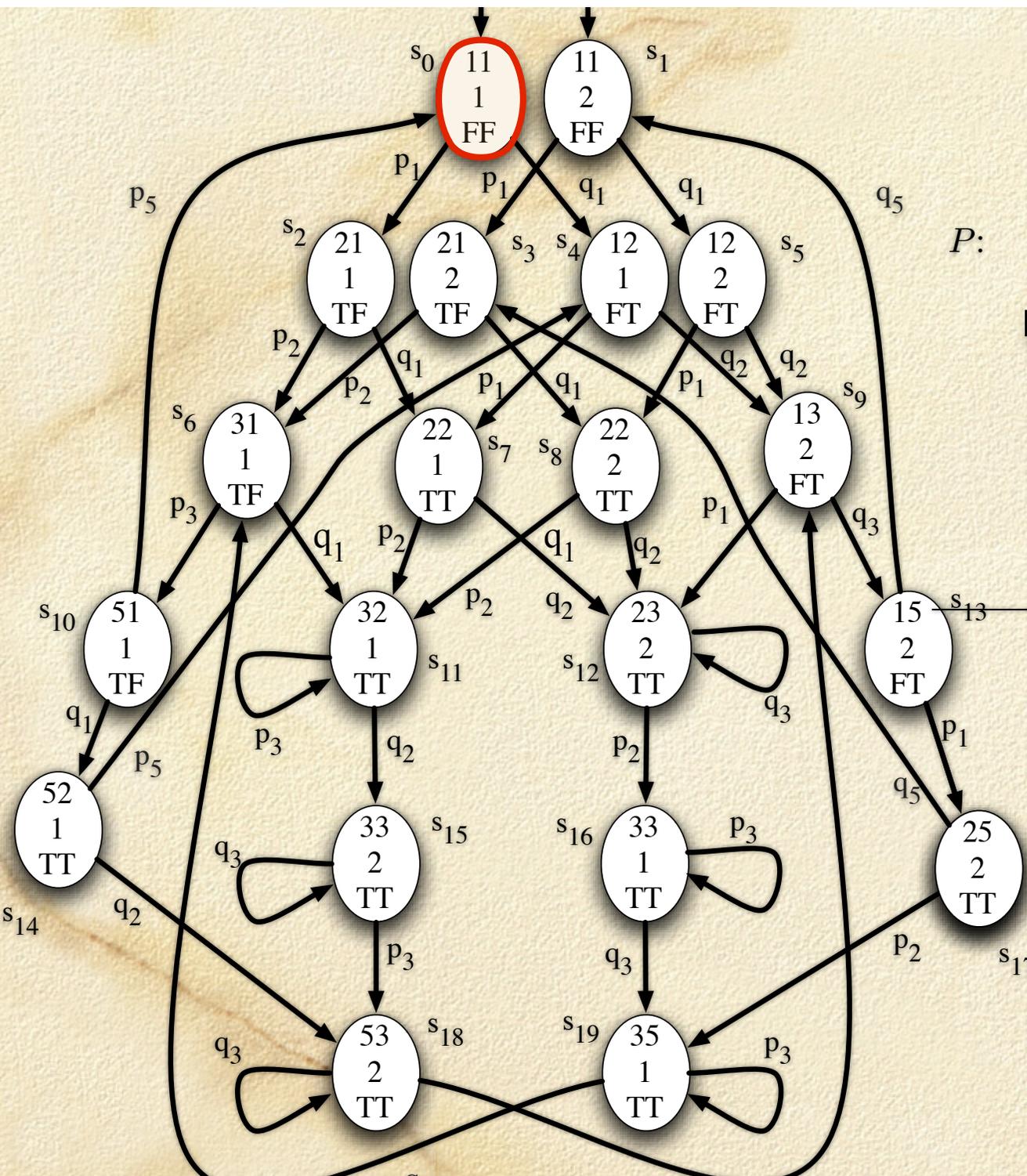
$P: l_0 : \mathbf{while} \text{ True do}$
 ~~$[p_0 : \text{non-critical section};]$~~
 $p_1 : wantP := True;$
 $p_2 : last := 1;$
 $p_3 : \mathbf{await}(wantQ = False$
 $\quad \vee last = 2);$
 ~~$[p_4 : \text{critical section};]$~~
 $p_5 : wantP := False;$
 $\mathbf{endwhile} l'_0$

$Q: l_1 : \mathbf{while} \text{ True do}$
 ~~$[q_0 : \text{non-critical section};]$~~
 $q_1 : wantQ := True;$
 $q_2 : last := 2;$
 $q_3 : \mathbf{await}(wantP = False$
 $\quad \vee last = 1);$
 ~~$[q_4 : \text{critical section};]$~~
 $q_5 : wantq := False;$
 $\mathbf{endwhile} l'_1$

Dabei ist $\mathcal{C}(l_0, P, l'_0)$ die Disjunktion von folgenden Formeln:

- $pc_0 = l_0 \wedge pc'_0 = p_1 \wedge True \wedge same(V)$ (Schleifen-Anweisung)
- $pc_0 = p_1 \wedge pc'_0 = p_2 \wedge wantP' = True \wedge same(V \setminus \{wantP\})$ (Zuweisung)
- $pc_0 = p_2 \wedge pc'_0 = p_3 \wedge last' = 1 \wedge same(V \setminus \{last\})$ (Zuweisung)
- $pc_0 = p_3 \wedge pc'_0 = p_3 \wedge \neg(wantQ = False \vee last = 2) \wedge same(V)$ (wait-Anweisung)
- $pc_0 = p_3 \wedge pc'_0 = p_5 \wedge (wantQ = False \vee last = 2) \wedge same(V)$ (wait-Anweisung)
- $pc_0 = p_5 \wedge pc'_0 = l_0 \wedge wantP' = False \wedge same(V \setminus \{wantP\})$ (Zuweisung)

$\mathcal{C}(l_1, Q, l'_1)$ ist entsprechend definiert.



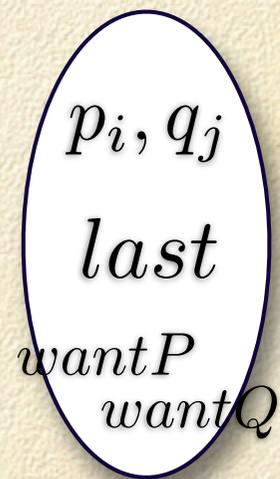
$wantP = wantQ = False$: boolean,
 $last = 1 \vee last = 2$: integer,
 m : cobegin $P \parallel Q$ coend
wobei

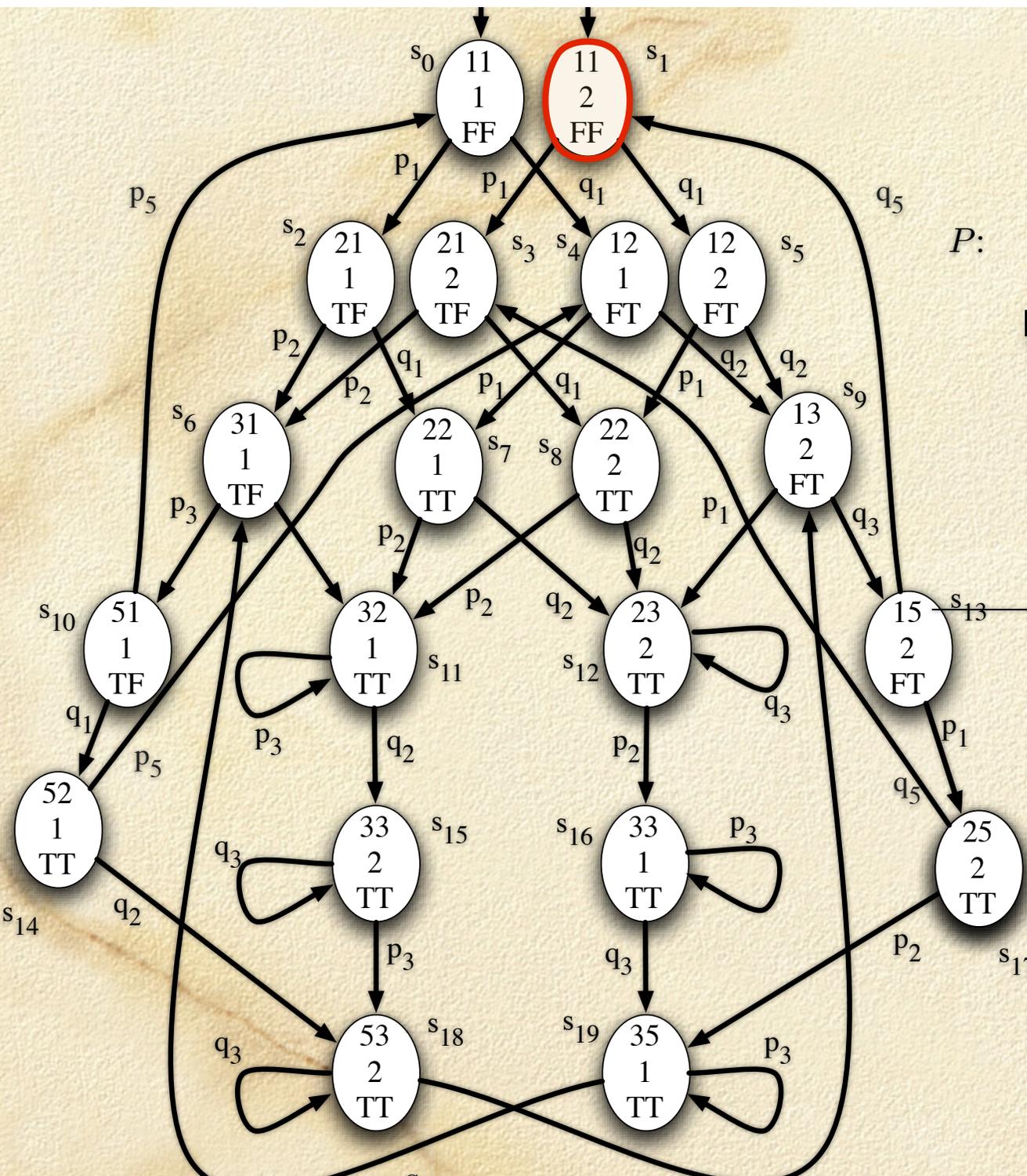
P :

```

 $l_0$  : while True do
  [  $p_0$  : non critical section; ]
   $\rightarrow$   $p_1$  :  $wantP := True$ ;
   $p_2$  :  $last := 1$ ;
   $p_3$  : await ( $wantQ = False$ 
               $\vee last = 2$ );
  [  $p_4$  : critical section; ]
   $p_5$  :  $wantP := False$ ;
endwhile  $l'_0$ 

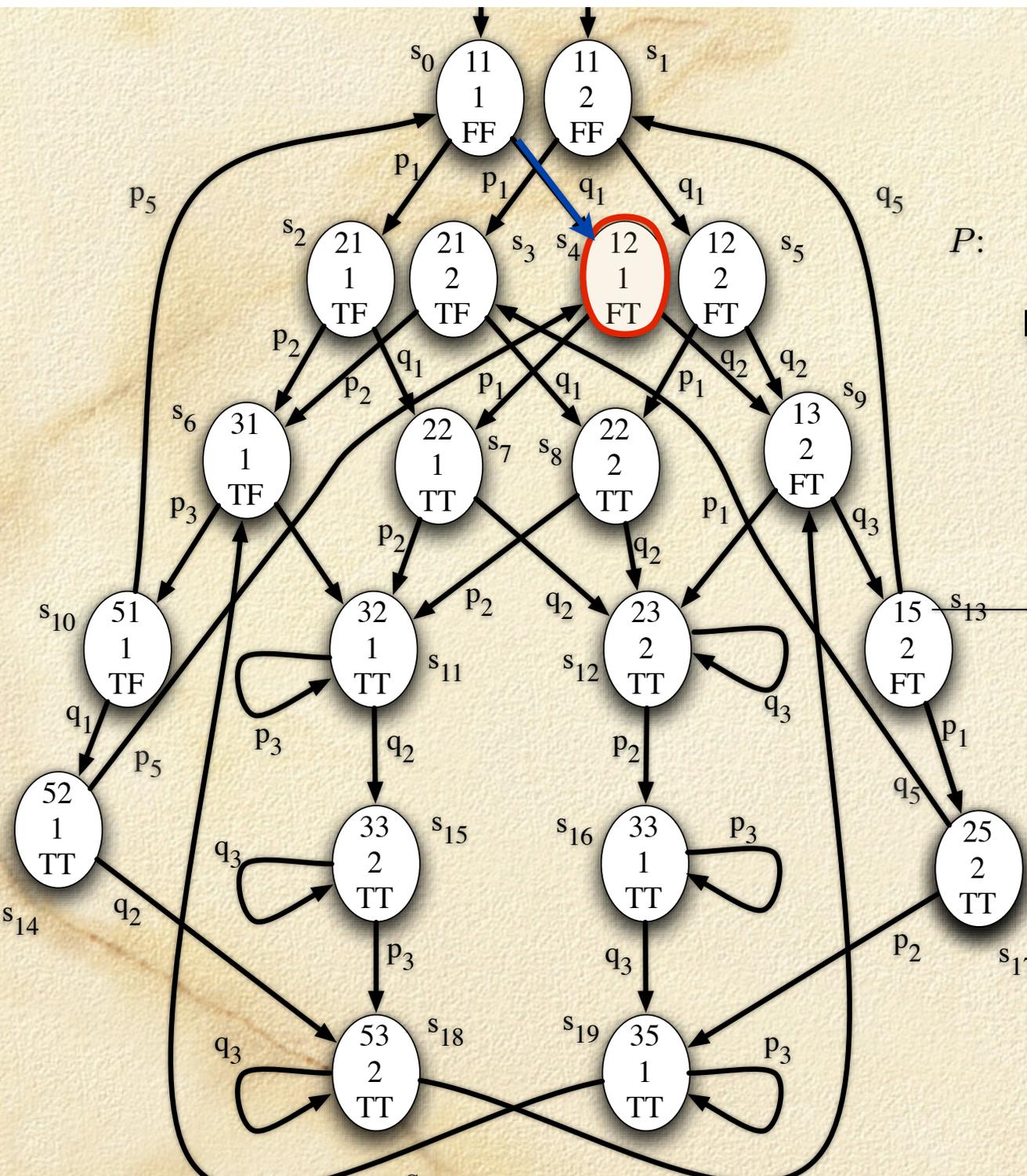
```





$wantP = wantQ = False$: boolean,
 $last = 1 \vee last = 2$: integer,
 m : cobegin $P \parallel Q$ coend
 wobei
 P : l_0 : while True do
 [p_0 : non-critical section;]
 \rightarrow p_1 : $wantP := True$;
 p_2 : $last := 1$;
 p_3 : await ($wantQ = False$
 $\vee last = 2$);
 [p_4 : critical section;]
 p_5 : $wantP := False$;
 endwhile l'_0





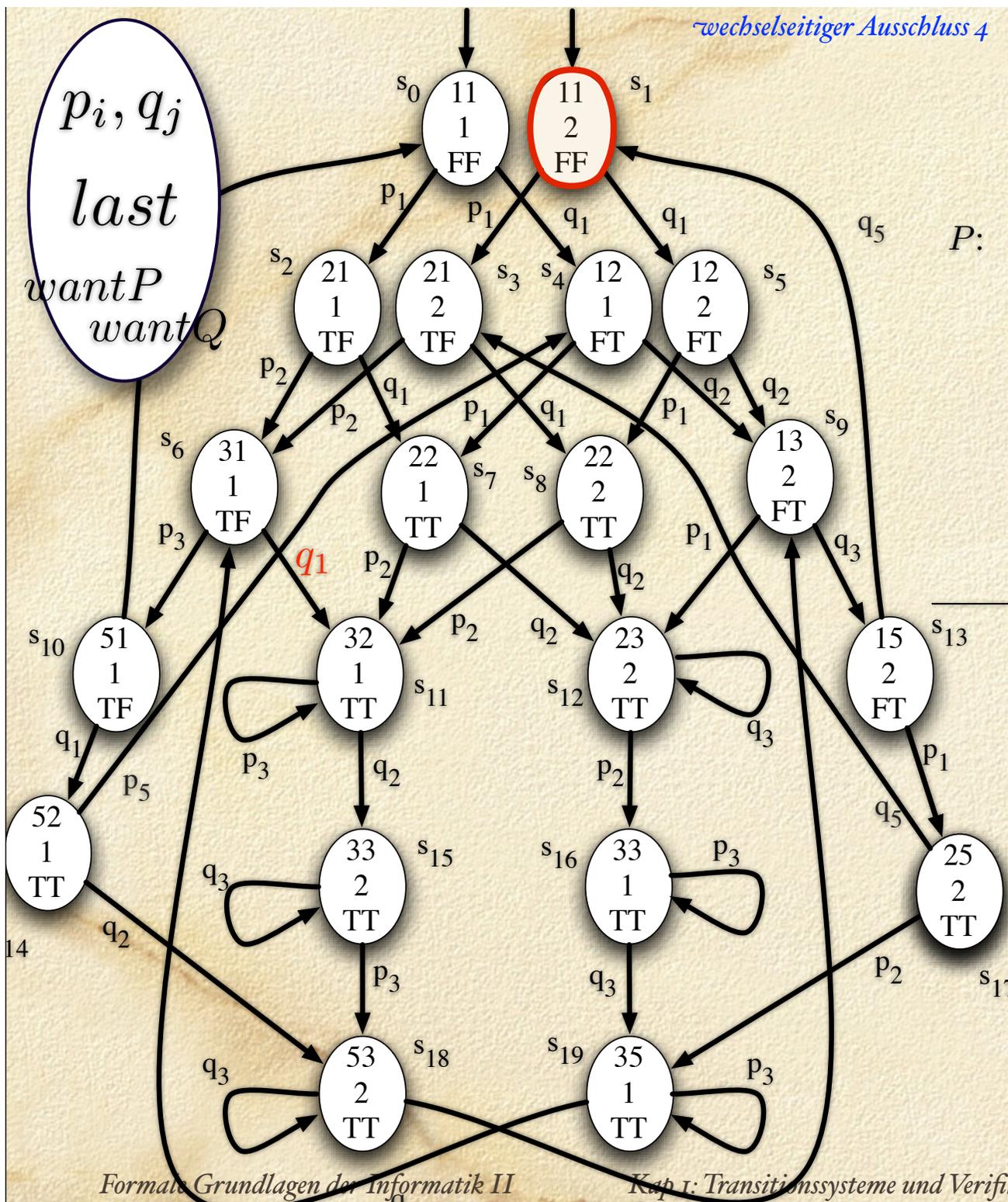
$wantP = wantQ = False$: boolean,
 $last = 1 \vee last = 2$: integer,
 m : **cobegin** $P||Q$ **coend**
 wobei

P : l_0 : **while** True **do**
 ~~p_0 : non-critical section;~~
 $\rightarrow p_1$: $wantP := True$;
 p_2 : $last := 1$;
 p_3 : **await** ($wantQ = False$
 $\vee last = 2$);
 ~~p_4 : critical section;~~
 p_5 : $wantP := False$;
endwhile l'_0



Die Programme sollen dabei (für den Fall zweier Prozesse P und Q) folgende Eigenschaften erfüllen:

- A) Die Befehlszähler von P und Q sind **nie gleichzeitig** in ihren kritischen Abschnitten.
- B) Meldet der Prozess P oder Q den Wunsch zum Eintritt in den kritischen Abschnitt an ($wantP = True$ oder $wantQ = True$), so **kann** er nach einer gewissen endlichen Zeit **tatsächlich** in seinen kritischen Abschnitt **eintreten**.



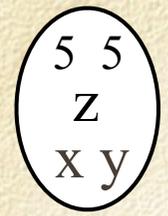
p_i, q_j
 $last$
 $wantP$
 $wantQ$

$wantP = wantQ = False : boolean,$
 $last = 1 \vee last = 2 : integer,$
 $m : cobegin P || Q coend$
wobei

```

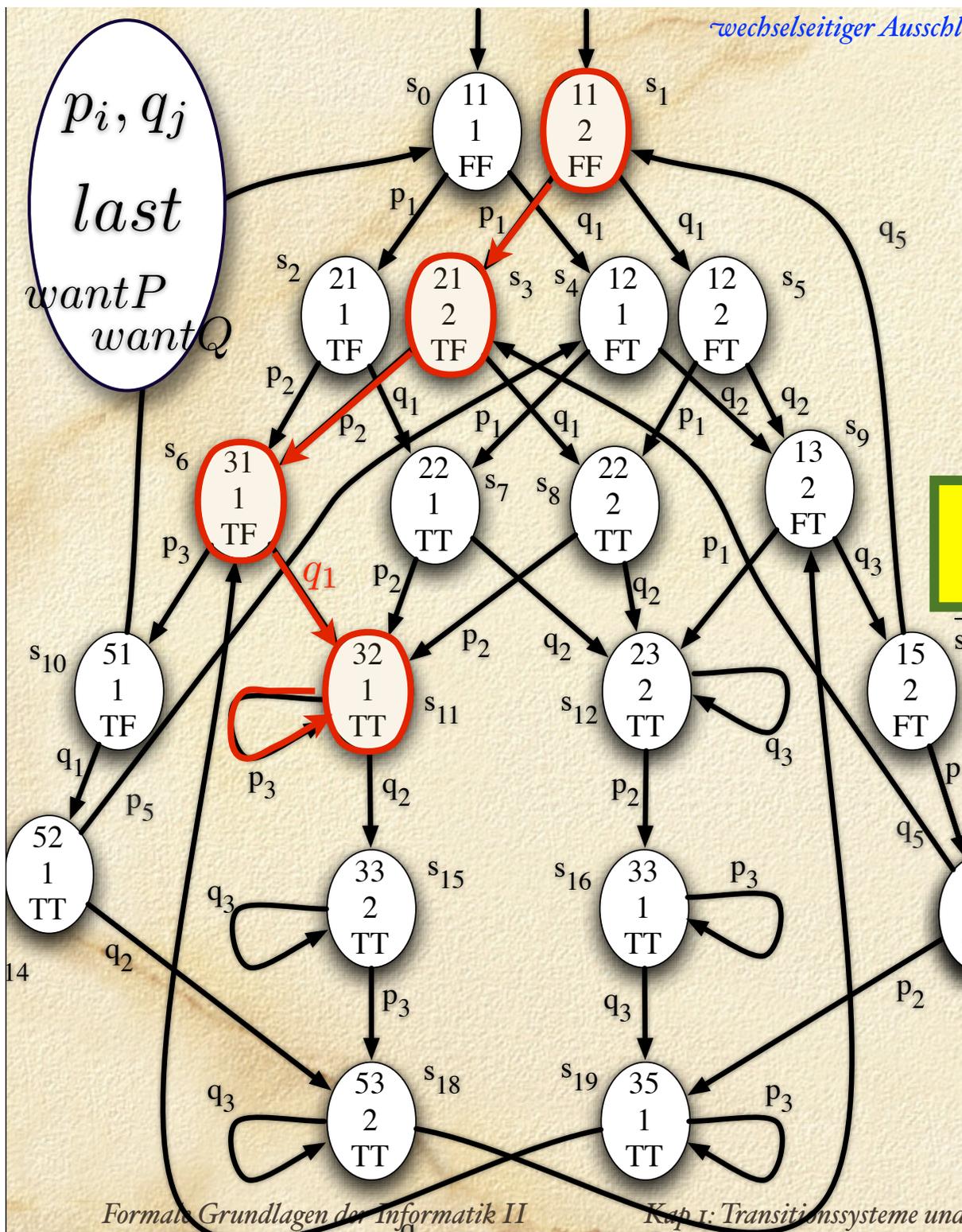
P: l0 : while True do
  [ p0 : non critical section; ]
  p1 : wantP := True;
  p2 : last := 1;
  p3 : await (wantQ = False
            ∨ last = 2);
  [ p4 : critical section; ]
  p5 : wantP := False;
endwhile l'0
    
```

Gilt A? nie
 $\neg(p5 \wedge q5)$
Gilt B?



Die Programme sollen dabei (für den Fall zweier Prozesse P und Q) folgende Eigenschaften erfüllen:

- A) Die Befehlszähler von P und Q sind **nie gleichzeitig** in ihren kritischen Abschnitten.
- B) Meldet der Prozess P oder Q den Wunsch zum Eintritt in den kritischen Abschnitt an ($wantP = True$ oder $wantQ = True$), so **kann** er nach einer gewissen endlichen Zeit **tatsächlich** in seinen kritischen Abschnitt **eintreten**.



p_i, q_j
 $last$
 $wantP$
 $wantQ$

$wantP = wantQ = False : boolean,$
 $last = 1 \vee last = 2 : integer,$
 $m : cobegin P || Q coend$
 wobei
 $P: l_0 : while True do$
 ~~$p_0 : non\ critical\ section;$~~
 $p_1 : wantP := True;$
 $p_2 : last := 1;$
 $p_3 : await (wantQ = False$
 $\vee last = 2);$

temporale Logik

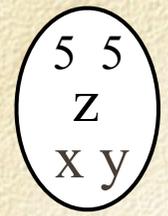
Gilt A?

$\square \neg (p_5 \wedge q_5)$

immer gilt Gilt B?

$\square (p_1 \Rightarrow \diamond p_5)$

später gilt p_5



verschleppungsfreie
 Schaltregel!!

(finite delay property)