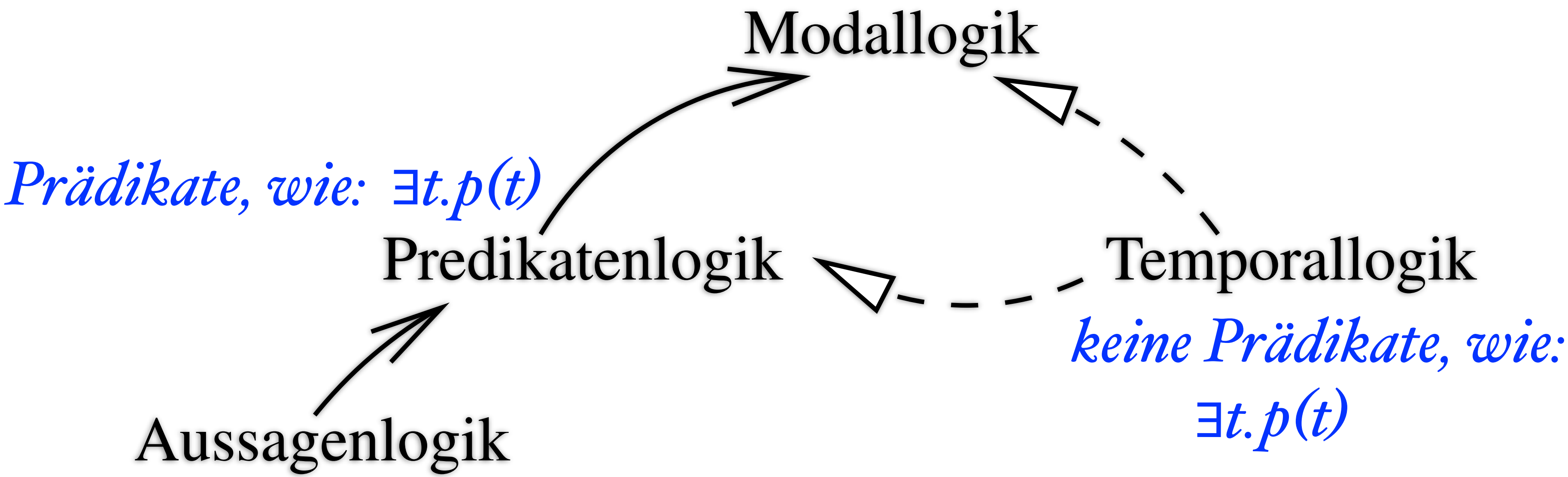
1.5 Temporale Logik



Beispiel 1.29 Spezifikation eines Aufzuges (Fragment)

I. Jede Anforderung des Aufzugs wird auch erfüllt.

II. Der Aufzug passiert keinen Stockwerk (SW) mit einer nicht erfüllten Anforderung.

Beispiel für physikalisches Bewegungsgesetz: $z(t) = -\frac{1}{2}gt^2$

I. Jede Anforderung des Aufzugs wird auch erfüllt.

$$\text{I. } \forall t, \forall n(app(n,t) \Rightarrow \exists t' > t \,.\, serv(n,t'))$$

$H(t)$ Position des Fahrstuhls zur Zeit t,

$app(n,t)$ offene Anforderung von Stockwerk n zur Zeit t,

$serv(n,t)$ Fahrstuhl bedient Stockwerk n

II. Der Aufzug passiert kein Stockwerk (SW) mit einer nicht erfüllten Anforderung.

II.

$$\forall t, \forall t' > t, \forall n \Bigg(\Big(app(n,t) \wedge H(t') \neq n \wedge \exists t_{trav} \,.\, t \leq t_{trav} \leq t' \wedge H(t_{trav}) = n \Big) \Rightarrow \Big(\exists t_{serv} \,.\, t \leq t_{serv} \leq t' \wedge serv(n, t_{serv}) \Big) \Bigg)$$

$H(t)$ Position des Fahrstuhls zur Zeit t,

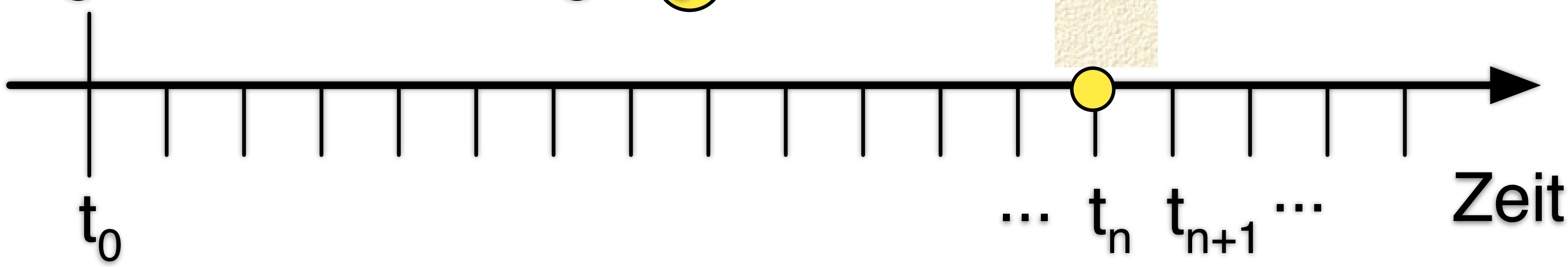
$app(n,t)$ offene Anforderung von Stockwerk n zur Zeit t,

$serv(n,t)$ Fahrstuhl bedient Stockwerk n

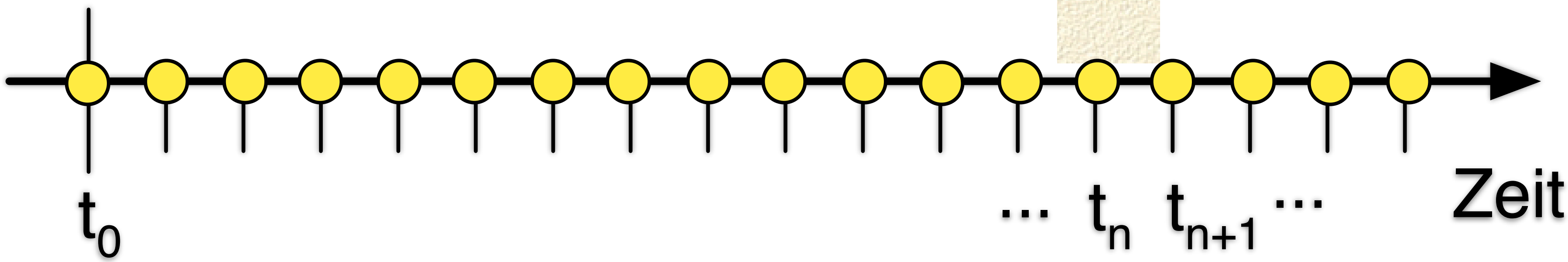
Temporale Logik für Informatik: Pnueli 1977

LTL

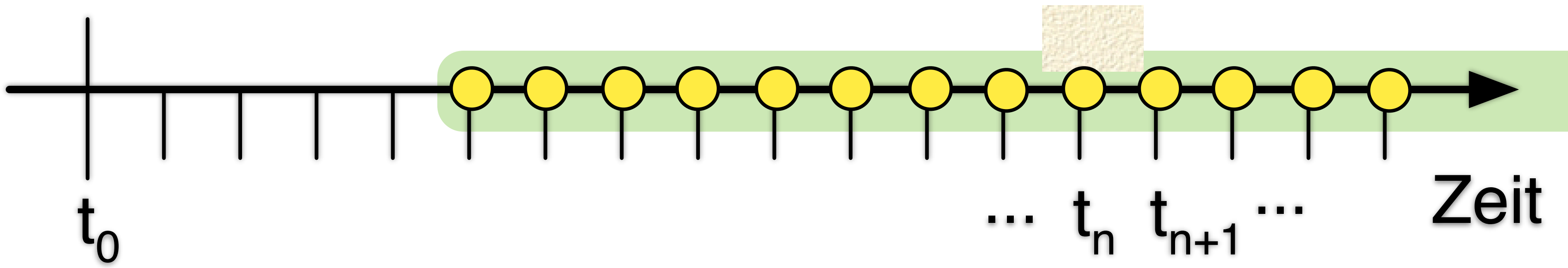
$\Diamond p$ irgendwann einmal gilt p



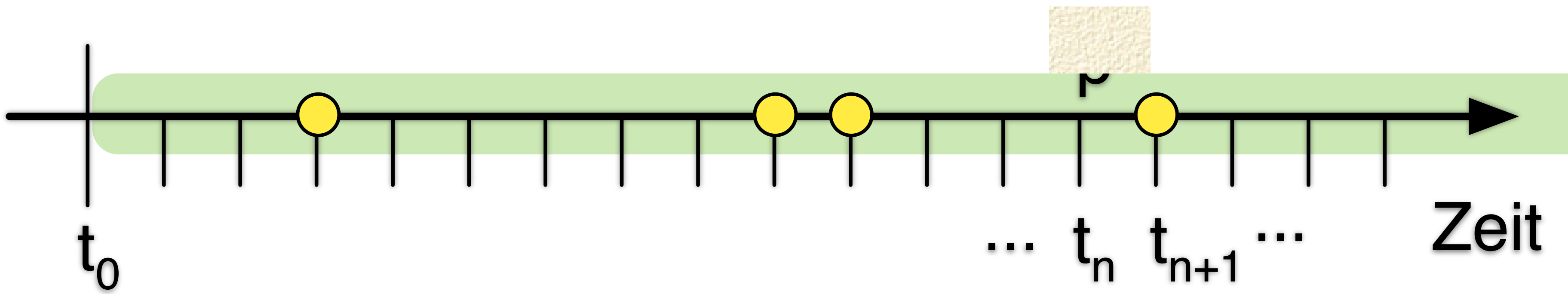
$\Box p$ von jetzt an gilt immer p



$\Diamond\Box p$ bedeutet?



$\Box\Diamond p$ bedeutet?



1.5.1 Syntax und Semantik von LTL-Formeln

$\Box p = Gp$ $\Diamond p = Fp$

$$G(((p \vee (\neg q))U(Xp)) \wedge (Fr)))$$

Definition 1.30 *Es sei AP eine Menge von aussagenlogischen Atomen. Dann wird die* Syntax von LTL-Formeln *wie folgt durch Backus-Naur-Form definiert:*

$$f ::= \mathbf{true}|\mathbf{false}|p|(\neg f)|(f \wedge f)|(f \vee f)|(Xf)|(Ff)|(Gf)|(fUf)$$

wobei $p \in AP$ *ist.*

$$G((p \vee \neg q)UXp) \wedge Fr)$$

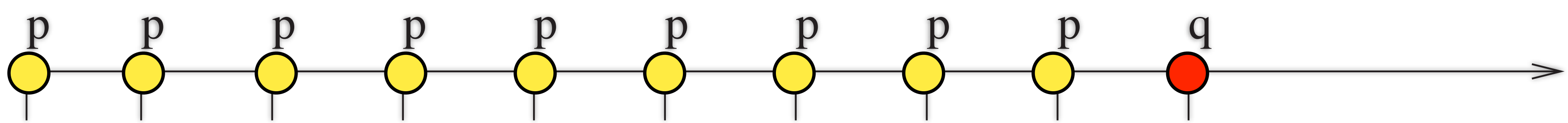
Semantik von LTL-Formeln

Xp „next time“: p gilt im zweitem Element A_1 der Folge (auch $\bigcirc p$ geschrieben),

Fp „eventually, in the future“: p gilt in einem Element einer gegebenen Folge (auch $\Diamond p$),

Gp „always, globally“: p gilt in allen Elementen einer gegebenen Folge (auch $\Box p$),

$p\,U\,q$ „until“: es gibt einen Element der gegebenen Folge, in dem q gilt und vor diesem Element gilt immer p.

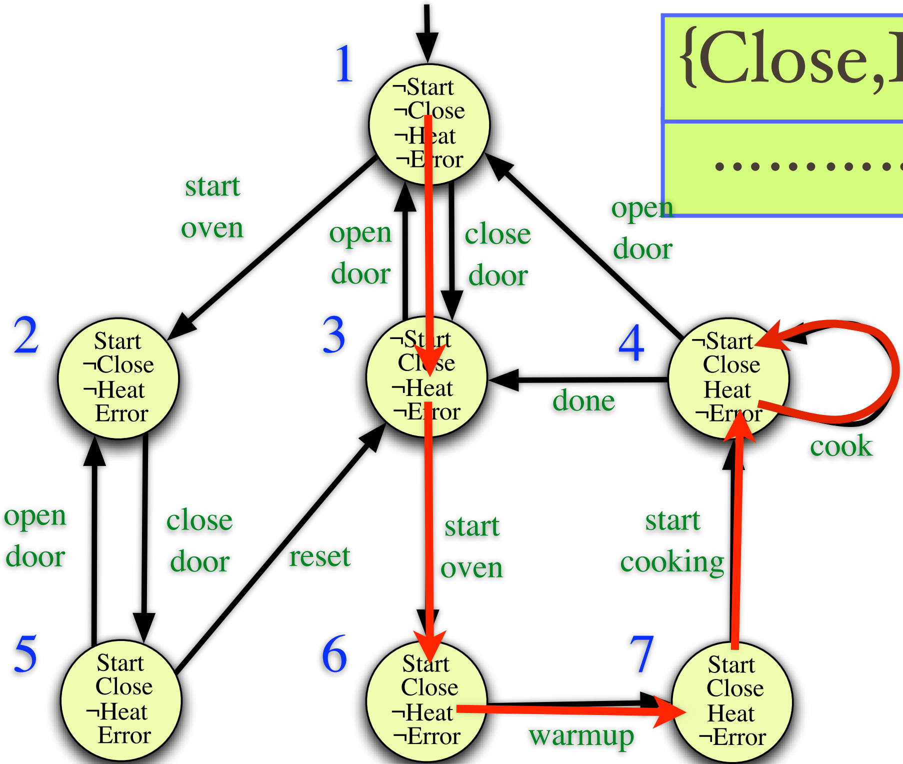


Definition 1.31 *Sei* $\alpha = A_0 A_1 A_2 \cdots \in \mathcal{P}(\mathcal{AP})^\omega$ *eine unendliche Folge von Aussagenmengen. Dann sei für* $i \in \mathbb{N}$ *die Folge* $\alpha^i = A_i A_{i+1} \cdots$ *der* i*-Suffix von* α*. In der folgenden induktiven Definition seien* $p \in AP$ *und* f*,* f_1*,* f_2 *LTL-Formeln.* $\alpha \models f$ *ist zu lesen als „für die Folge* α *gilt* f*" oder „*α *erfüllt* f*".*

$\alpha =$ ∅ {Close} {Start,Close} {Start,Close,Heat}

{Close,Heat} {Close,Heat} {Close,Heat}

....



$\alpha^3 =$ {Start,Close,Heat}

{Close,Heat}

{Close,Heat}

{Close,Heat}

....

Definition 1.31 *Sei* $\alpha = A_0 A_1 A_2 \cdots \in \mathcal{P}(\mathcal{AP})^\omega$ *eine unendliche Folge von Aussagenmengen. Dann sei für* $i \in \mathbb{N}$ *die Folge* $\alpha^i = A_i A_{i+1} \cdots$ *der* i*-Suffix von* α*. In der folgenden induktiven Definition seien* $p \in AP$ *und* f*,* f_1*,* f_2 *LTL-Formeln.* $\boxed{\alpha \models f}$ *ist zu lesen als „für die Folge* α *gilt* f*" oder „*$\boxed{\alpha \text{ erfüllt } f}$*".*

1. $\alpha \models \textbf{true}$.
2. $\alpha \not\models \textbf{false}$.
3. $\alpha \models p \quad \Leftrightarrow \quad p \in A_0$.

Konstante & elementare Aussagen

4. $\alpha \models \neg f \quad \Leftrightarrow \quad \alpha \not\models f$.
5. $\alpha \models f_1 \vee f_2 \quad \Leftrightarrow \quad \alpha \models f_1$ *oder* $\alpha \models f_2$.
6. $\alpha \models f_1 \wedge f_2 \quad \Leftrightarrow \quad \alpha \models f_1$ *und* $\alpha \models f_2$.

aussagenlogische Junktoren

7. $\alpha \models Xf \quad \Leftrightarrow \quad \alpha^1 \models f$.

nächster Schritt

8. $\alpha \models Ff \quad \Leftrightarrow \quad \exists k \geq 0 : \alpha^k \models f$.
9. $\alpha \models Gf \quad \Leftrightarrow \quad \forall k \geq 0 : \alpha^k \models f$.

irgendwann & immer

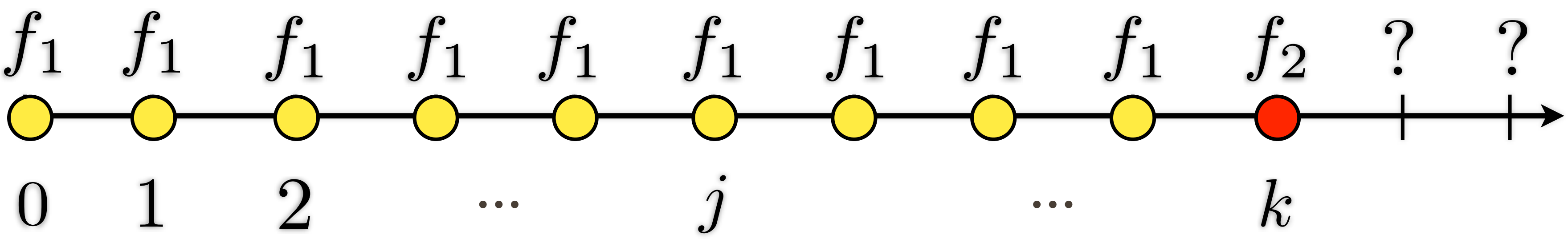
10. $\alpha \models f_1 U f_2 \quad \Leftrightarrow \quad \exists k \geq 0 \,.\, \alpha^k \models f_2$ *und für alle* $0 \leq j < k$ *gilt* $\alpha^j \models f_1$.

until

Definition 1.31 *Sei* $\alpha = A_0 A_1 A_2 \cdots \in \mathcal{P}(\mathcal{AP})^\omega$ *eine unendliche Folge von Aussagenmengen. Dann sei für* $i \in \mathbb{N}$ *die Folge* $\alpha^i = A_i A_{i+1} \cdots$ *der* i*-Suffix von* α*. In der folgenden induktiven Definition seien* $p \in AP$ *und* f, f_1, f_2 *LTL-Formeln.* $\boxed{\alpha \models f}$ *ist zu lesen als „für die Folge* α *gilt* f*" oder „*$\boxed{\alpha \text{ erfüllt } f}$*".*

10. $\alpha \models f_1 U f_2 \quad \Leftrightarrow \quad \exists k \geq 0 \,.\, \alpha^k \models f_2$ *und für alle* $0 \leq j < k$ *gilt* $\alpha^j \models f_1$.

until



- $F\, f \Leftrightarrow$ **true** $U\, f$

$L^\omega(f) := \{\alpha \in \mathcal{P}(AP)^\omega \mid \alpha \models f\}$ *heißt Menge der* α *erfüllenden Folgen über* AP *oder die* Sprache von α.

8.	$\alpha \models F f$	$\Leftrightarrow$	$\exists k \geq 0 : \alpha^k \models f.$
9.	$\alpha \models G f$	$\Leftrightarrow$	$\forall k \geq 0 : \alpha^k \models f.$

irgend wann &
immer

- $G\,f \Leftrightarrow \neg F\,\neg f.$
- $f_1 \wedge f_2 \Leftrightarrow \neg(\neg f_1 \vee \neg f_2),$
- $F\,f \Leftrightarrow \mathbf{true}\,U\,f,$

Eine LTL-Formel f gilt für eine unendliche Folgen $\pi = s_0 s_1 s_2 \cdots$ von Zuständen einer Kripke-Struktur $M := (S, S_0, R, E_S)$, wenn sie für die zugehörige Zustandsetikettenfolge $E_S(\pi) = E_S(s_0)E_S(s_1)E_S(s_2)\cdots \in \mathcal{P}(AP)^\omega$ gilt. Man schreibt $M, \pi \models f$.

Definition 1.33 *Sei $M := (S, S_0, R, E_S)$ eine Kripke-Struktur und $\pi = s_0 s_1 s_2 \cdots \in S^\omega$ eine unendliche Folge von Zuständen von M. Dann sei $M, \pi \models f :\Leftrightarrow E_S(\pi) \models f$.*

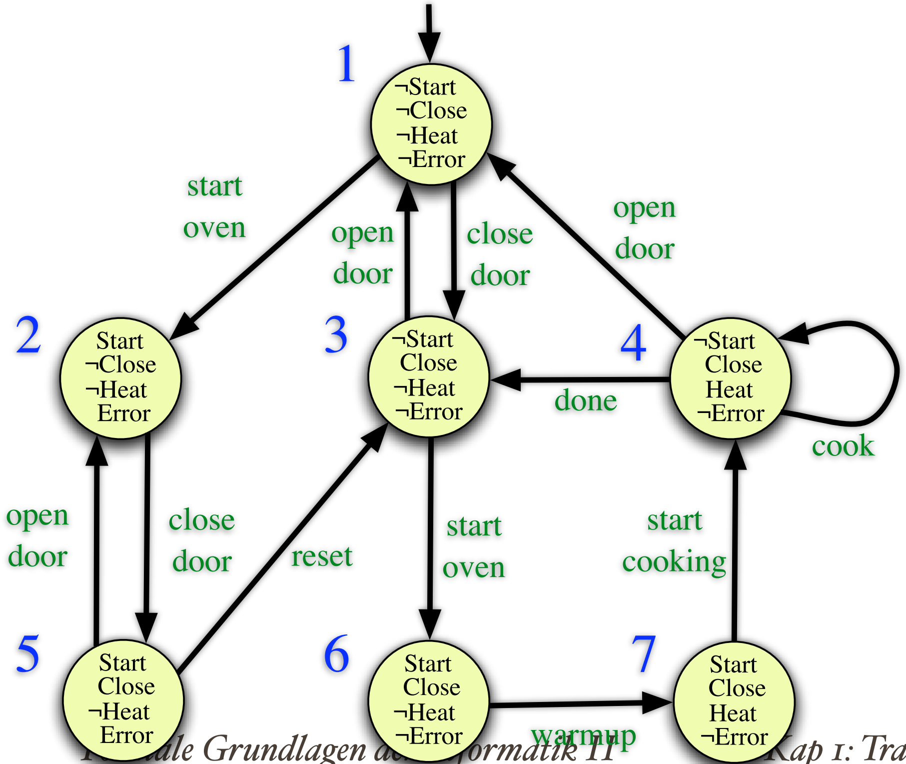


Abb. 1.16
Seite 36

Definition 1.35 *Eine LTL-Formel f* gilt *(ist* gültig*) im Zustand $s \in S$ einer Kripke-Struktur $M := (S, S_0, R, E_S)$ falls $M, \pi \models f$ für alle Pfade $\pi = s_0 s_1 s_2 \cdots \in S^\omega$ gilt, die in s beginnen, d.h. $s_0 = s$ (in Zeichen $M, s \models f$). Sie gilt in M, falls sie in allen Anfangszuständen gilt, also: $M \models f :\Leftrightarrow \forall s \in S_0 : M, s \models f$.*

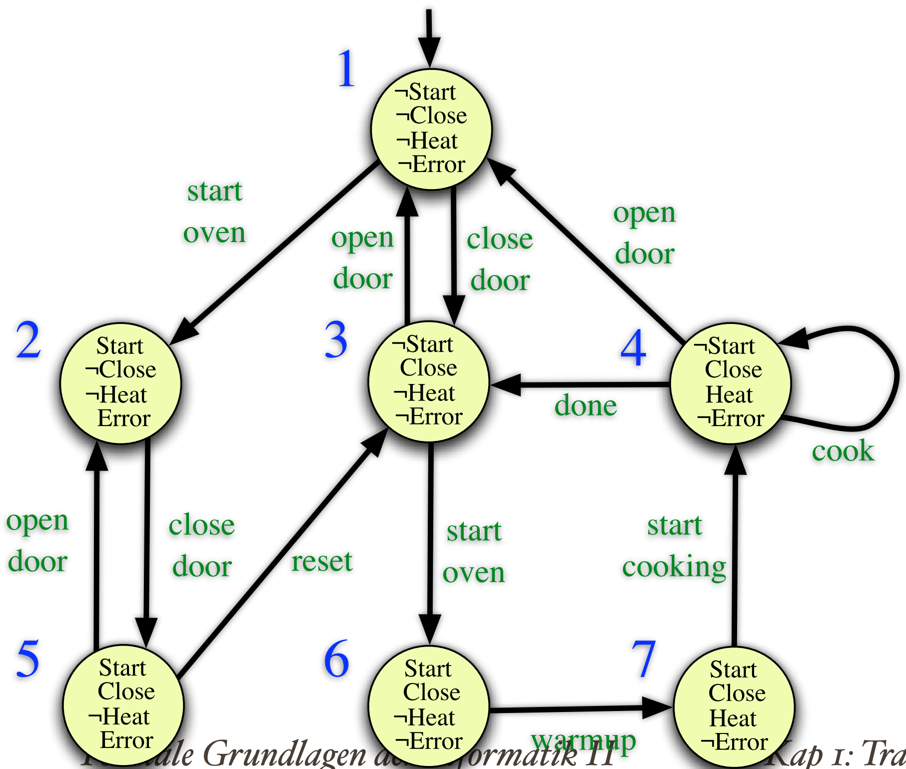


Abb. 1.16
Seite 36

[] (~ heat U close)

$G(\neg heat\ U\ close)$

gilt!

Für alle unendlichen Abläufe gilt:
irgend wann gilt $close$
und bis dahin immer $\neg heat$ *!*

$\neg heat\ \neg heat\ \neg heat \cdots \neg heat\ \ close \cdots$

Kripke-Struktur
Mikrowellenofen

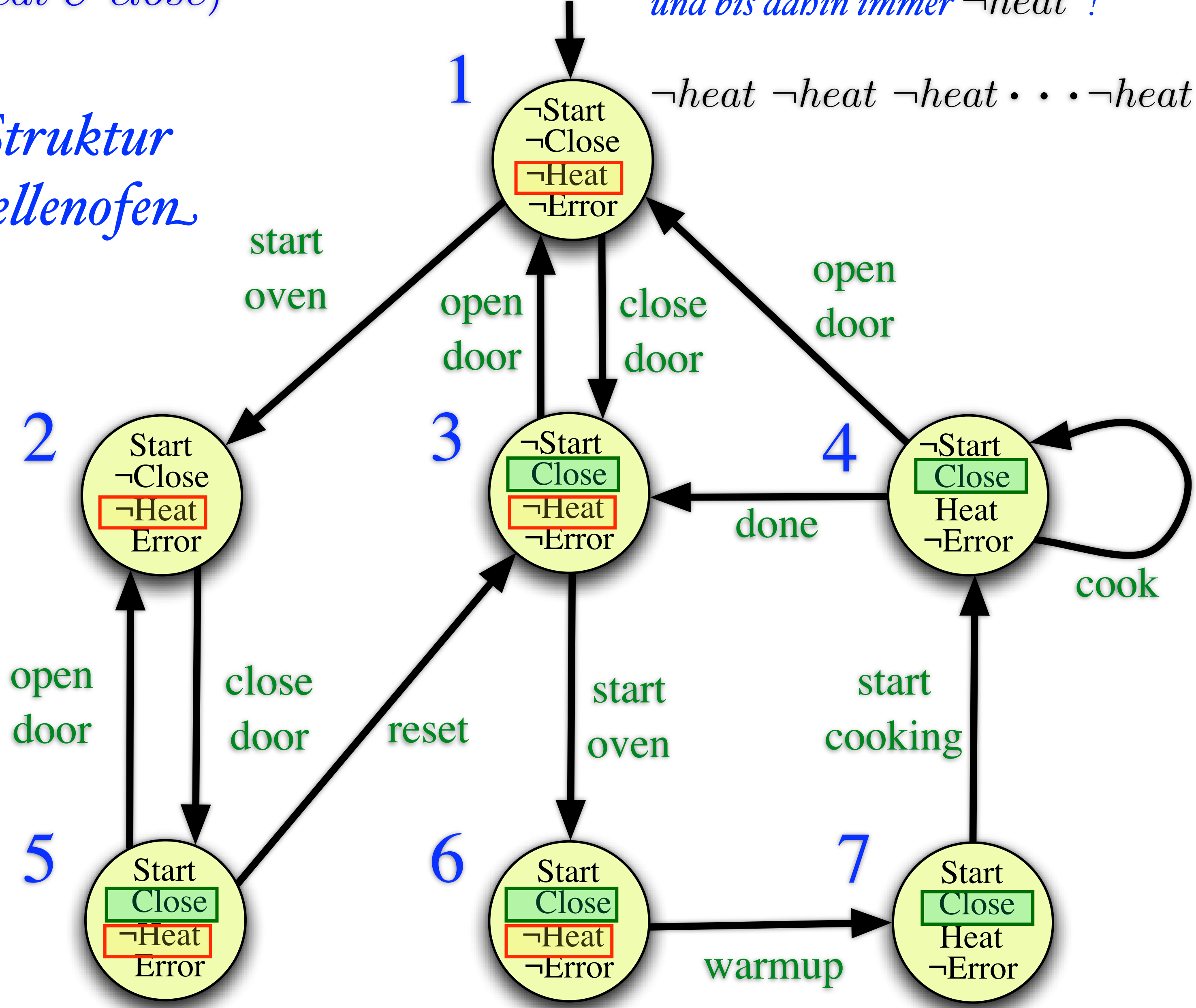


Abb. 1.16
Seite 36

[]<> (start Λ ~ error -> <> heat))   *gilt!*

Kripke-Struktur
Mikrowellenofen

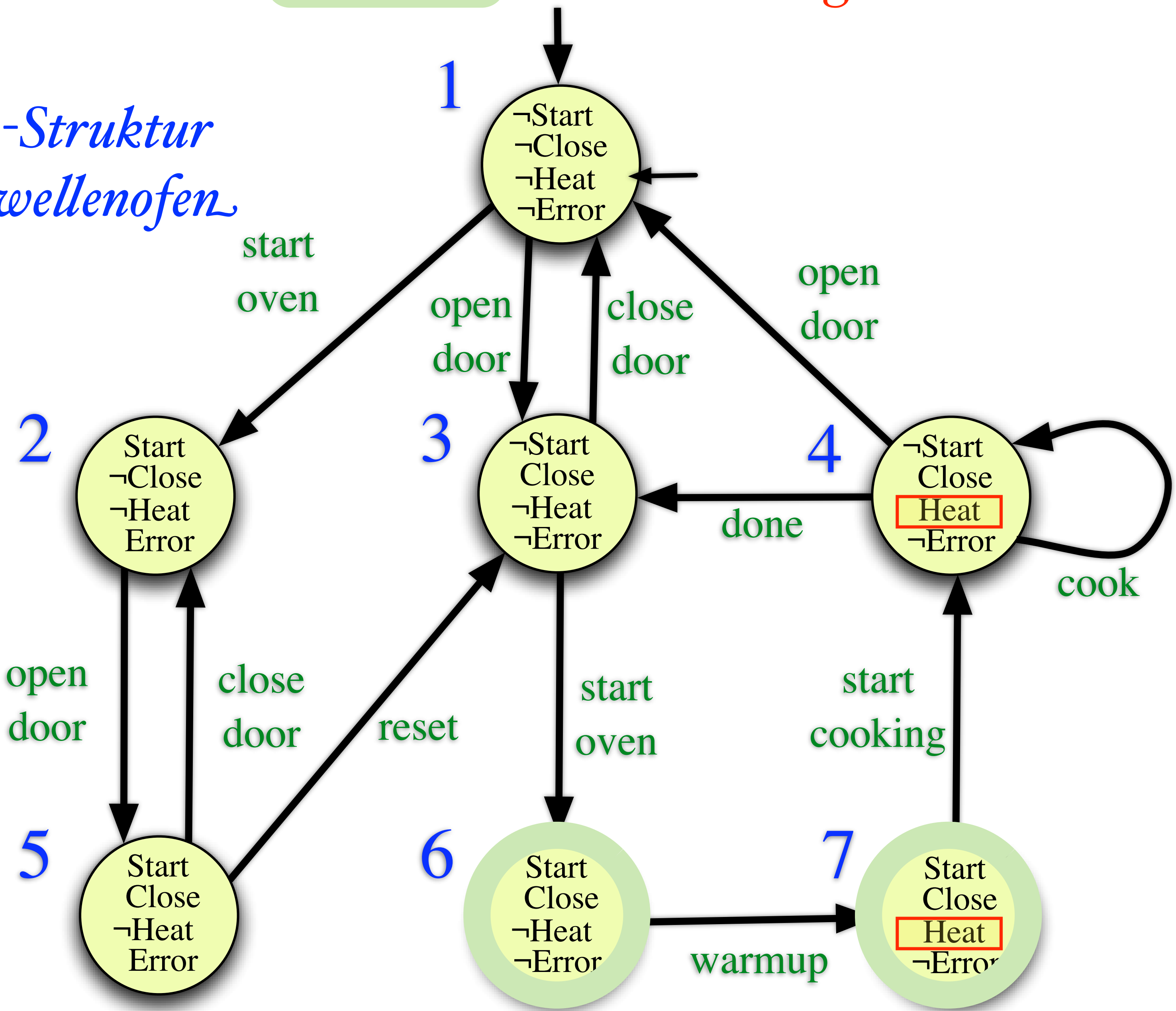


Abb. 1.16
Seite 36

[] (start -> <> heat) counterexample({s1,'start_oven}, {s2,'close_door} {s5,'reset} {s3,'open_door} {s1,'start_oven})

[]<> (start -> <> heat)) counterexample({s1,'start_oven}, {s2,'close_door} {s5,'open_door})

Kripke-Struktur
Mikrowellenofen

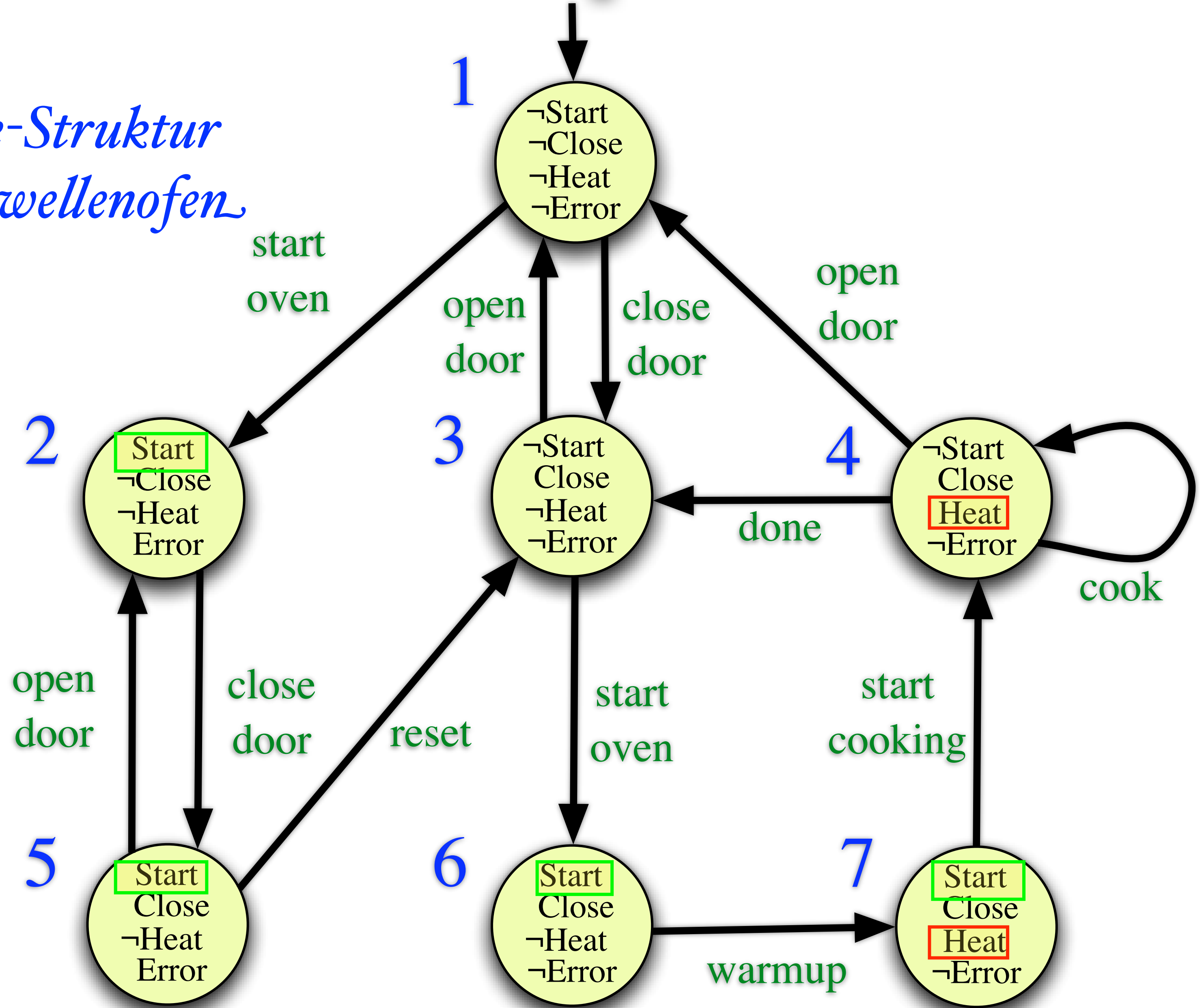


Abb. 1.16
Seite 36

[] (start -> <> heat)) **counterexample({s1,'start_oven}, {s2,'close_door} {s5,'reset} {s3,'open_door} {s1,'start_oven})**

[]<> (start -> <> heat)) **counterexample({s1,'start_oven}, {s2,'close_door} {s5,'open_door})**

<> (heat ∧ start) -> []<> heat) **counterexample({s1,'start_oven} {s2,'close_door} {s5,'reset} {s3,'start_oven} {s6,'warmup} {s7,'start_cooking} {s4,'open_door}, {s1,'start_oven} {s2,'close_door} {s5,'reset} {s3,'open_door})**

Kripke-Struktur
Mikrowellenofen

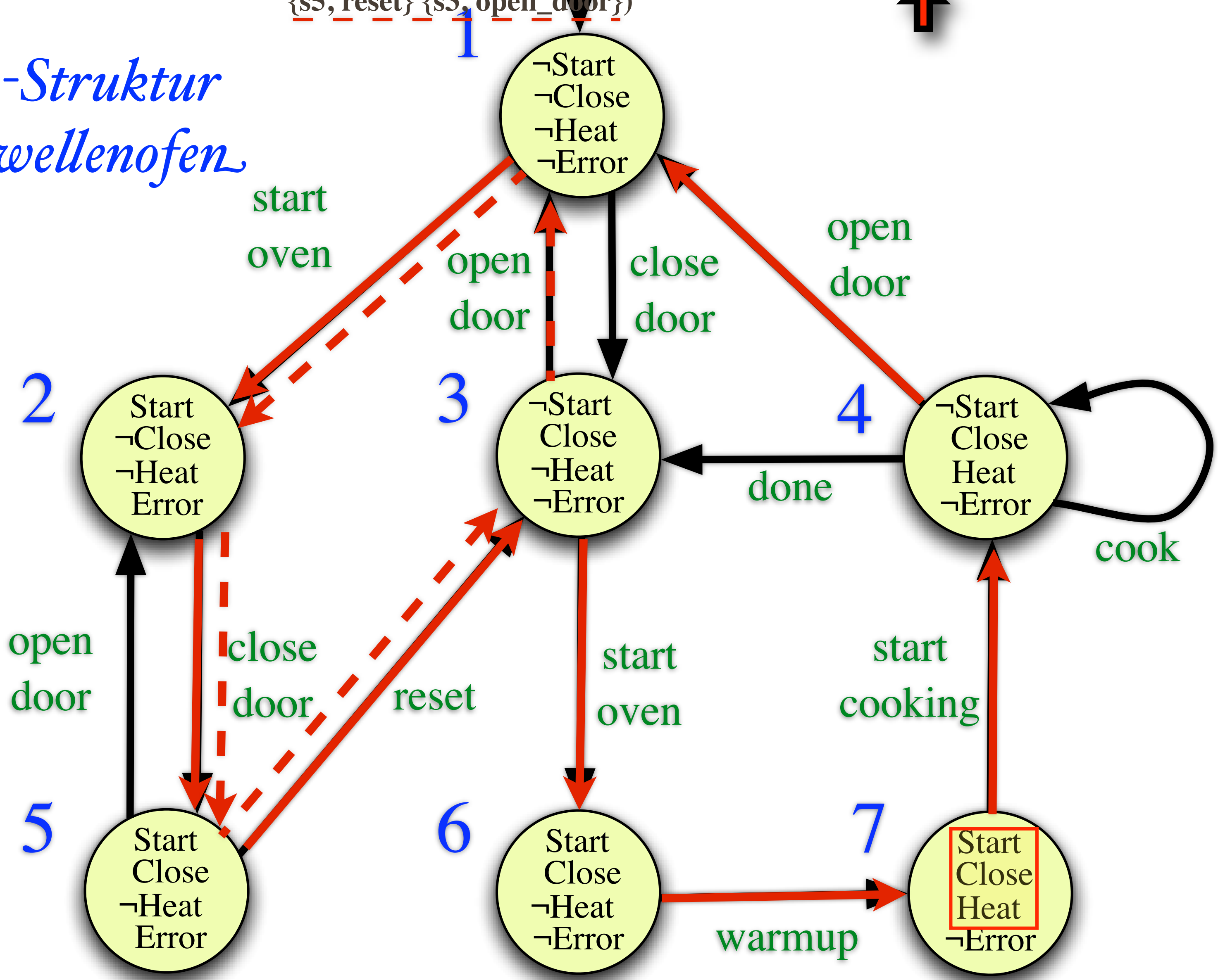
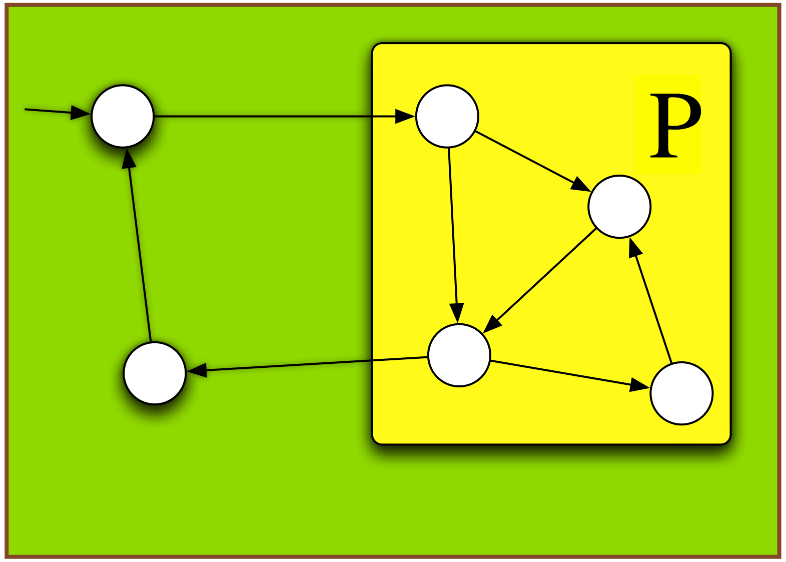


Abb. 1.16
Seite 36

Satz 1.36 *Zu jeder LTL-Formel f kann eine Kripke-Struktur (ein Büchi-Automat) M konstruiert werden, die genau die für f gültigen Folgen akzeptiert: $L^\omega(M) = L^\omega(f)$.*

Die Zeit- und Platz-Komplexität dieses Algorithmus ist $2^{\mathcal{O}(|f|)}$

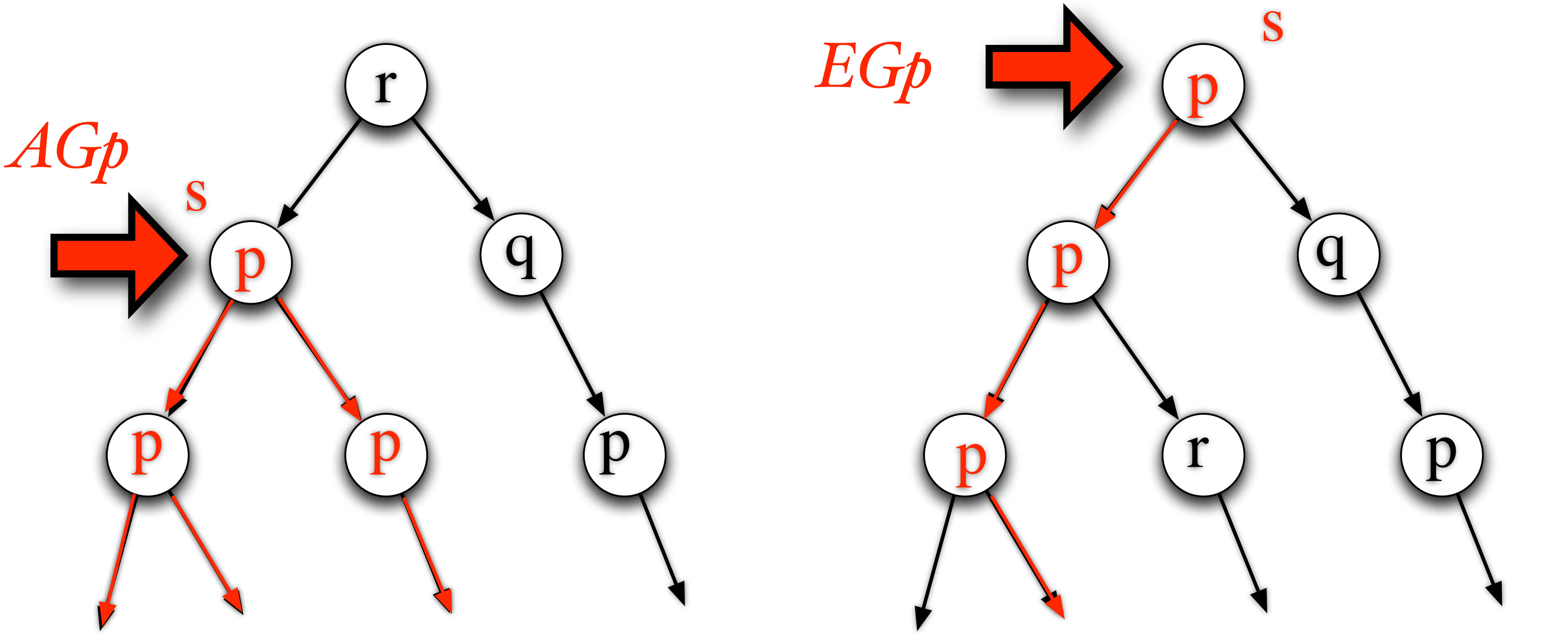


1.5.2 Syntax und Semantik von CTL- und CTL*-Formeln

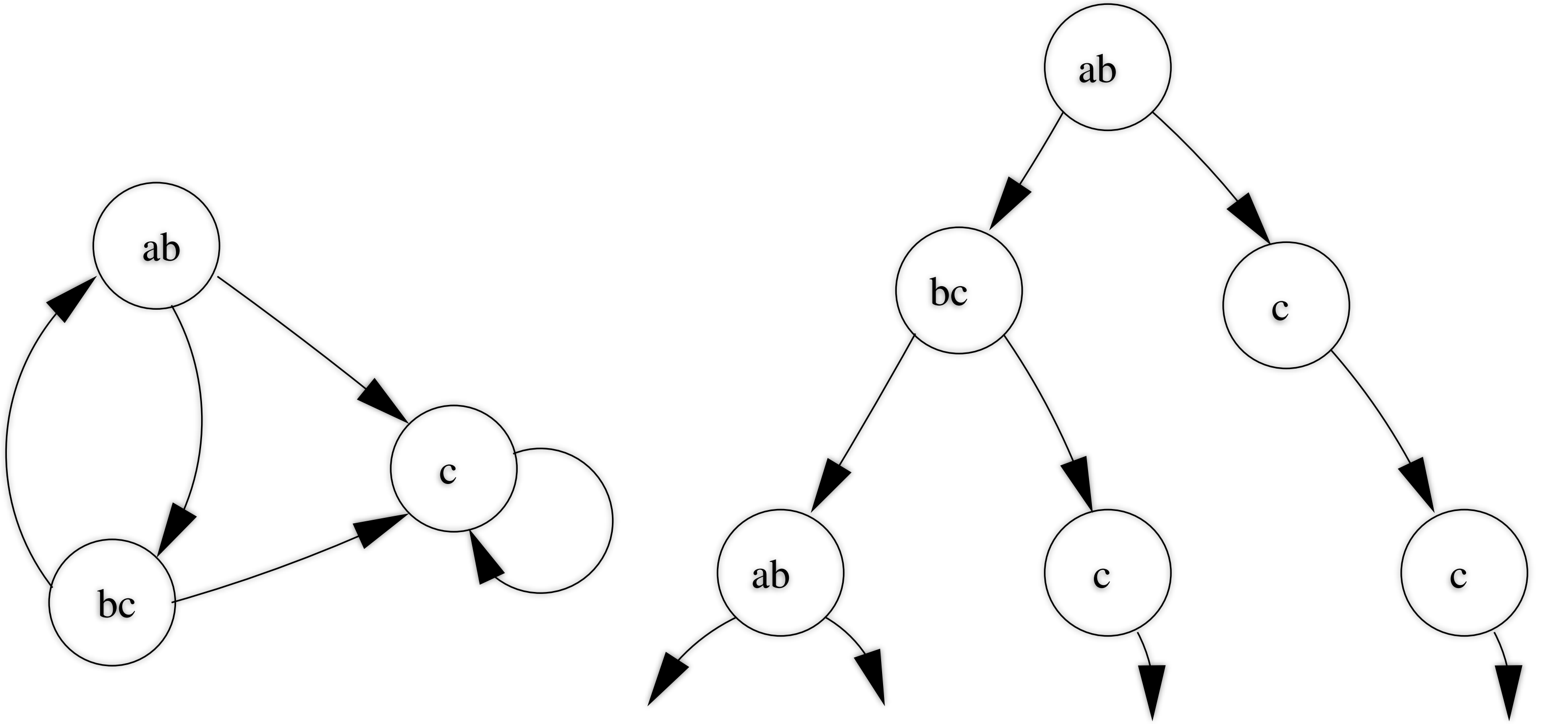
CTL besitzt zusätzlich *Zustandsformeln* mit Quantoren, die sich auf alle von einem Zustand ausgehenden Pfade beziehen.

Ap „für alle Pfade, die von einem gegeben Zustand s ausgehen, gilt die Formel p“,

Ep „es gibt einen Pfad, der von einem gegeben Zustand s ausgeht und für den die Formel p gilt“.



"Abwickeln" der Kripke-Struktur



In CTL müssen vor den Pfad-Quantoren X, F, G, U immer Zustands-Quantoren A oder E stehen. Es gibt damit 8 Kombinationen:

- AXg und EXg,
- AFg und EFg,
- AGg und EGg,
- $A[g_1 U g_2]$ und $E[g_1 U g_2]$,

Definition 1.38 *Es sei AP eine Menge von aussagenlogischen Atomen. Dann wird die Syntax von* CTL-Zustands-Formeln *wie folgt durch Backus-Naur-Form definiert, wobei $p \in AP$ und f eine* CTL-Pfad-Formel *ist:*

$$g ::= \textbf{true} | \textbf{false} | p | (\neg g) | (g \wedge g) | (g \vee g) | (Ef) | (Af)$$

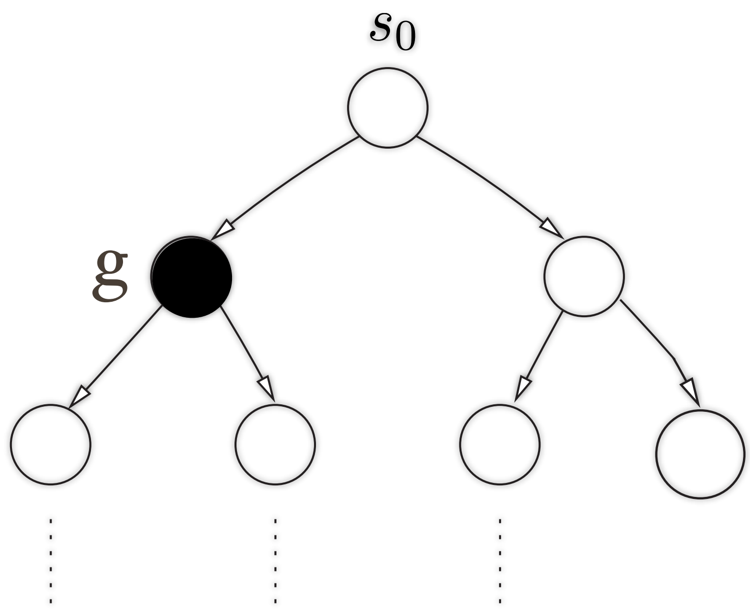
Eine CTL-Pfad-Formel *wird durch*

$$f ::= (Xg) | (Fg) | (Gg) | (g_1 U g_2)$$

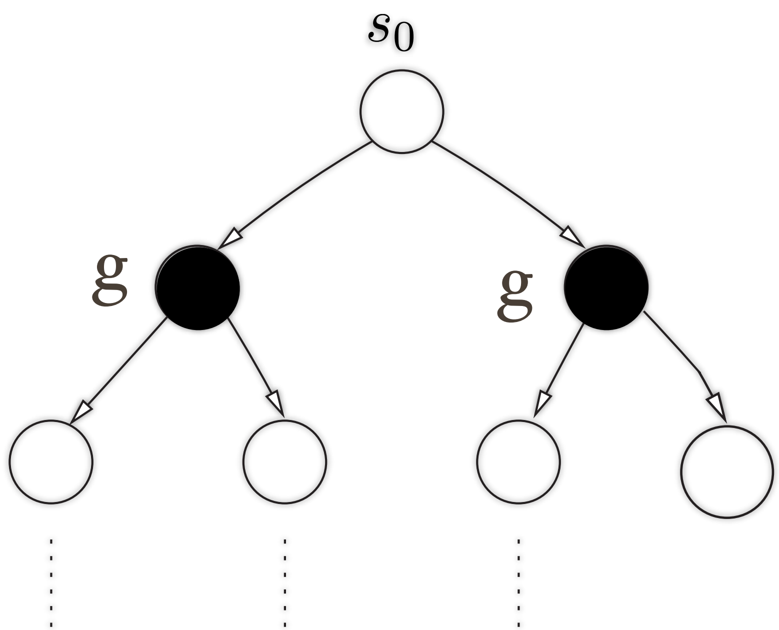
definiert. Dabei sind g, g_1, g_2 CTL-Zustands-Formeln.

- AXg und EXg,
- AFg und EFg,
- AGg und EGg,
- $A[g_1 U g_2]$ und $E[g_1 U g_2]$,

Seite 47



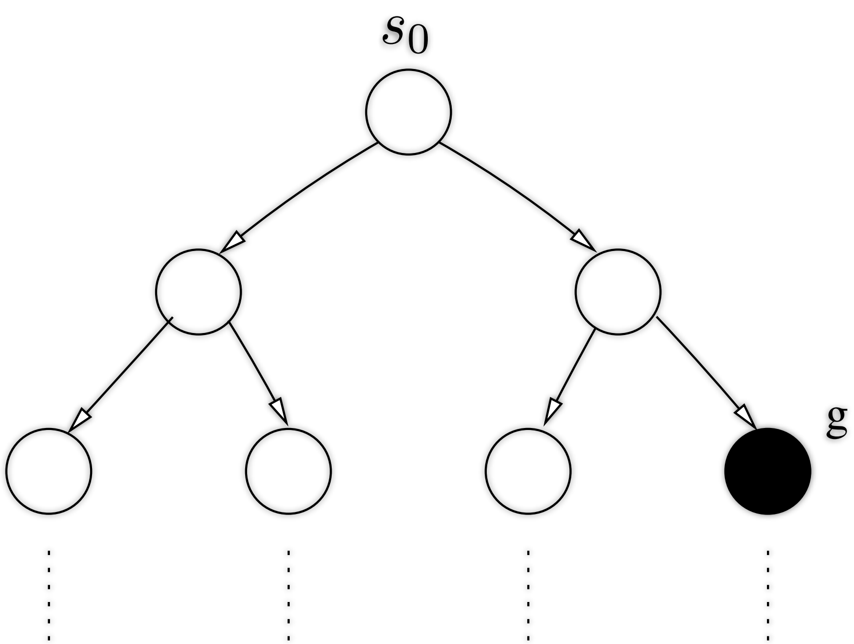
$M, s_0 \models \mathbf{EX}g$



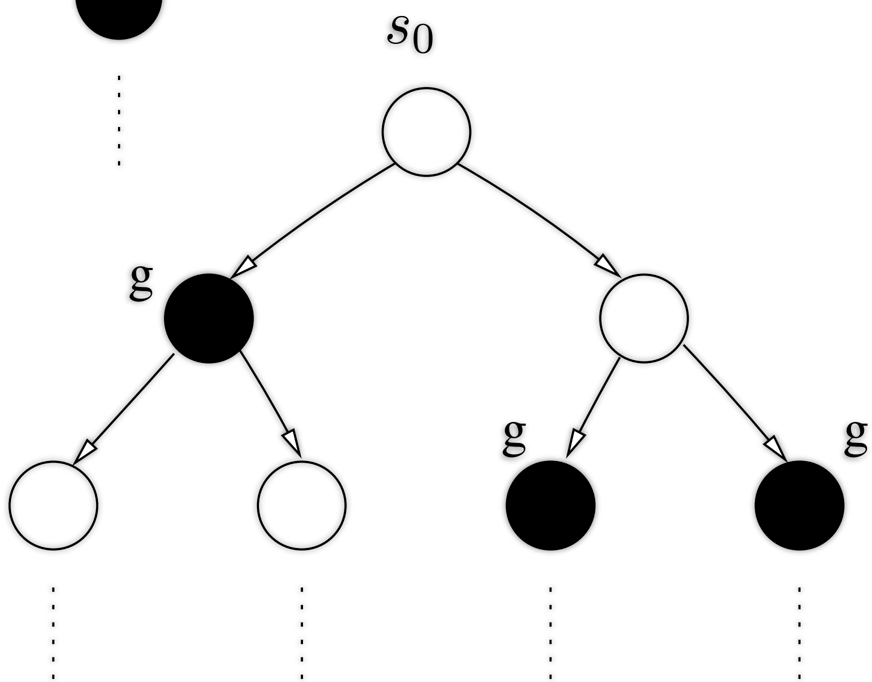
$M, s_0 \models \mathbf{AX}g$

- AXg und EXg,
- AFg und EFg,
- AGg und EGg,
- $A[g_1Ug_2]$ und $E[g_1Ug_2]$,

Seite 47



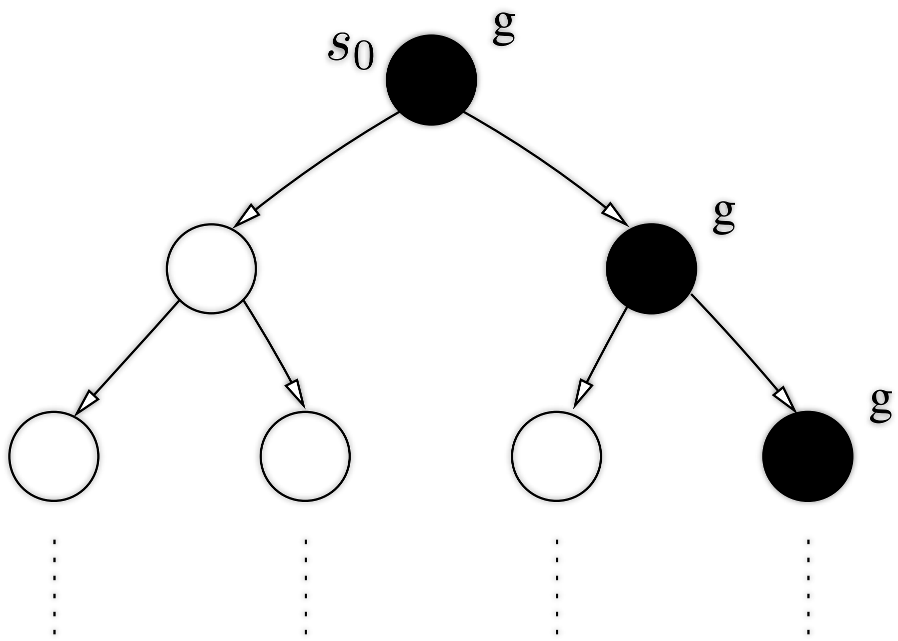
(a) $M, s_0 \models \mathbf{EF}g$



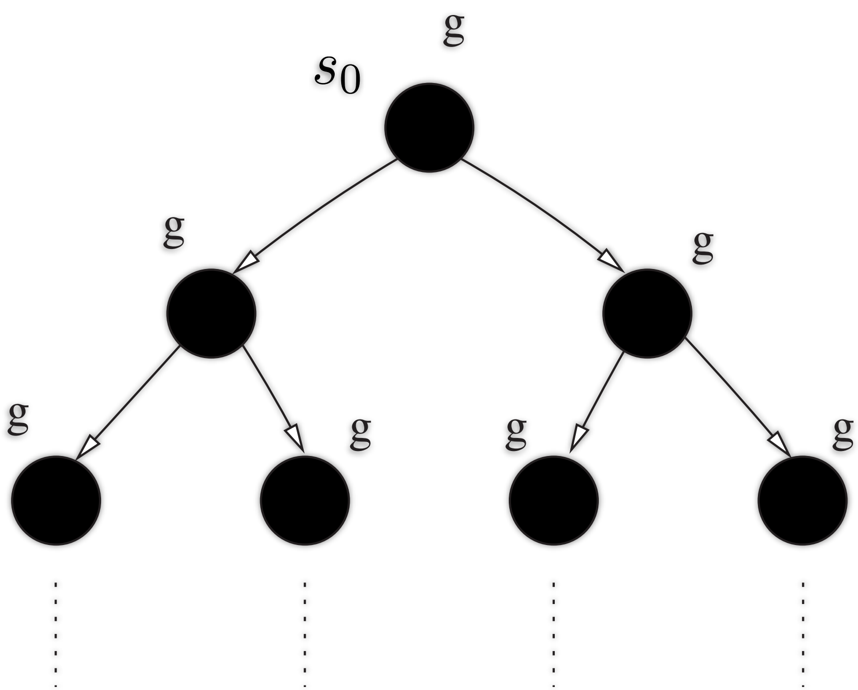
(b) $M, s_0 \models \mathbf{AF}g$

- AXg und EXg,
- AFg und EFg,
- AGg und EGg,
- $A[g_1 U g_2]$ und $E[g_1 U g_2]$,

Seite 47



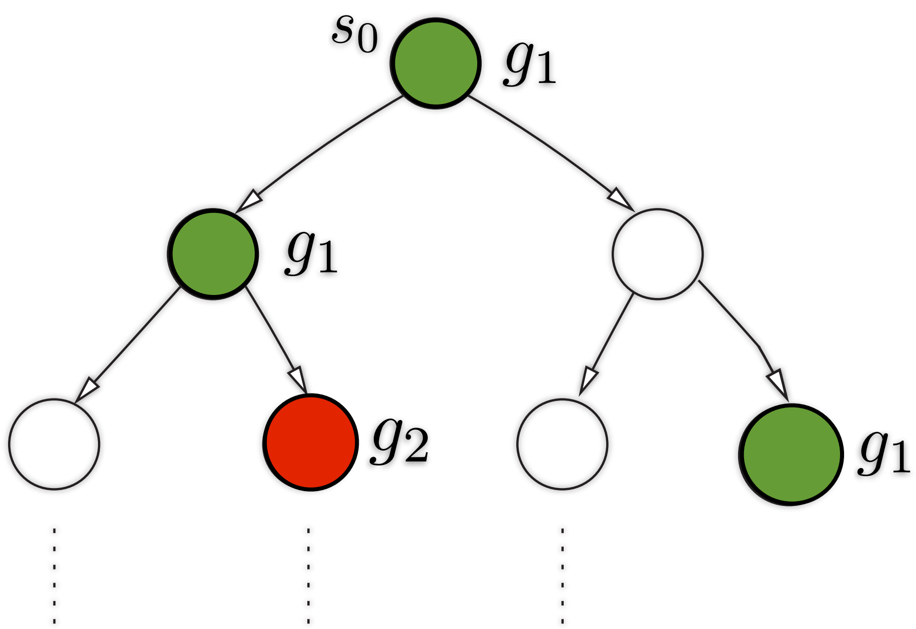
(c) $M, s_0 \models \mathbf{EG}g$



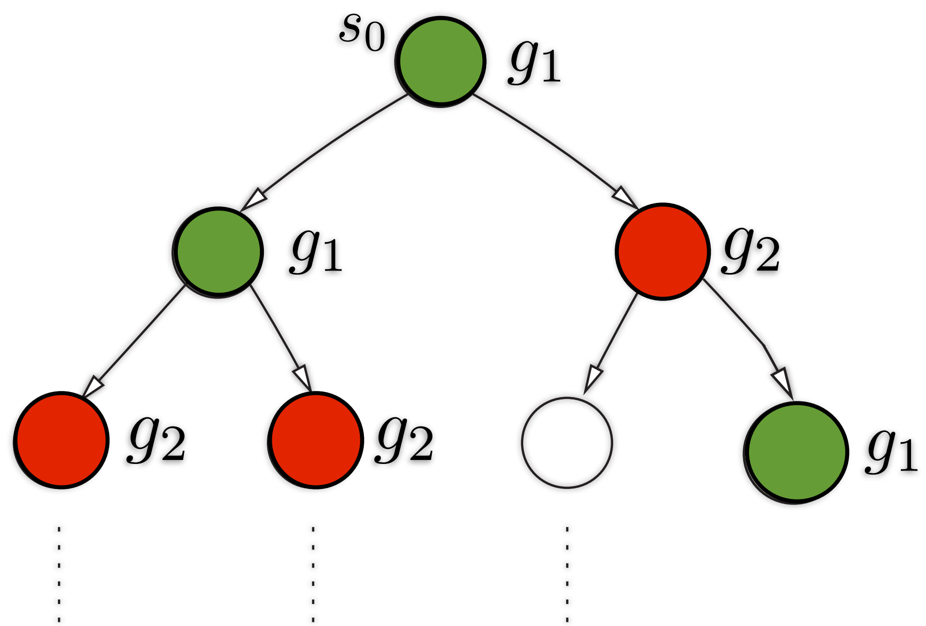
(d) $M, s_0 \models \mathbf{AG}g$

- AXg und EXg,
- AFg und EFg,
- AGg und EGg,
- $A[g_1 U g_2]$ und $E[g_1 U g_2]$,

Seite 47



$M, s_0 \models \mathbf{E}[g_1 \mathbf{U} g_2]$



$M, s_0 \models \mathbf{A}[g_1 \mathbf{U} g_2]$

- AXg und EXg,
- AFg und EFg,
- AGg und EGg,
- $A[g_1 U g_2]$ und $E[g_1 U g_2]$,

Diese können alle mittels EXg, EGg, $E[g_1 U g_2]$ ausgedrückt werden:

Satz 1.40 *Es gelten die folgenden Äquivalenzen:*

- $AXg \Leftrightarrow \neg EX(\neg g)$,
- $EFg \Leftrightarrow E(\mathbf{true}\, U g)$,
- $AGg \Leftrightarrow \neg EF(\neg g)$,
- $AFg \Leftrightarrow \neg EG(\neg g)$,
- $A[g_1 U g_2] \Leftrightarrow \neg E[\neg g_2 U(\neg g_1 \wedge \neg g_2)] \wedge \neg EG \neg g_2$

Beispiele:

- $EF(Start \wedge \neg Ready)$
  Es ist möglich in einen Zustand zu kommen, in dem „*Start*" aber nicht „*Ready*" gilt.

- $AG(Req \rightarrow AF\ Ack)$
  Immer wenn ein Request *Req* erfolgt, dann wird er später einmal mit *Ack* bestätigt.

- $AG(AF\ DeviceEnabled)$
  Die Aussage „*DeviceEnabled*" gilt unendlich oft auf jedem Pfad.

- $AG(EF\ Restart)$
  Von jedem Zustand aus ist es möglich, einen Zustand mit „*Restart*" zu erreichen.

Vergleich

LTL

linear time logic

Syntax:

(A) f

nur Pfad-Quantoren
X, F, G, U

CTL

computation tree logic
branching time logic

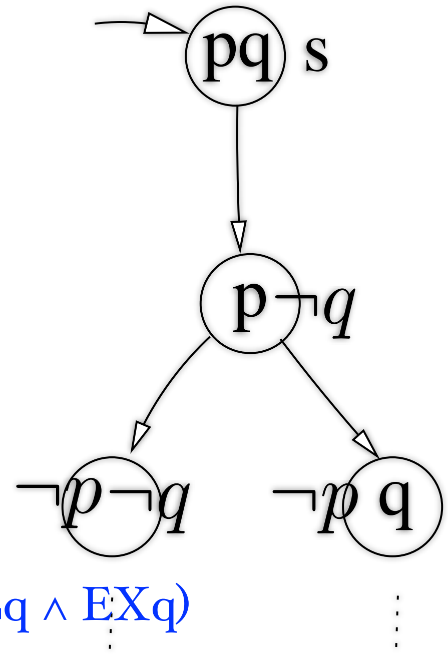
Syntax:

- AXf und EXf,
- AFf und EFf,
- AGf und EGf,
- $A[fUg]$ und $E[fUg]$,

Aufgabe 1.41 Gegeben sind die folgenden Kripke-Strukturen M_1 und M_2:

LTL:
A (pq ∧ X(p ∧ (X¬q ∨ Xq))

pq, p¬q, ¬p¬q, ...
pq, p¬q, ¬p q, ...

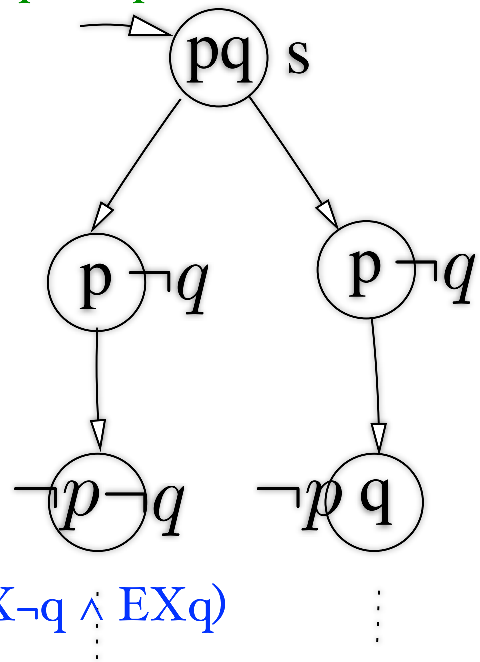


CTL:
pq ∧ AX(p ∧ EX¬q ∧ EXq)

(a) M_1

LTL:
A (pq ∧ X(p ∧ (X¬q ∨ Xq))

pq, p¬q, ¬p¬q, ...
pq, p¬q, ¬p q, ...



CTL:
pq ∧ AX(p ∧ EX¬q ∧ EXq)

gilt hier nicht!

(b) M_2

a) Gibt es Formeln in LTL, die die Strukturen unterscheiden, d.h. nur in einem Modell gelten?

b) Das gleiche für CTL.

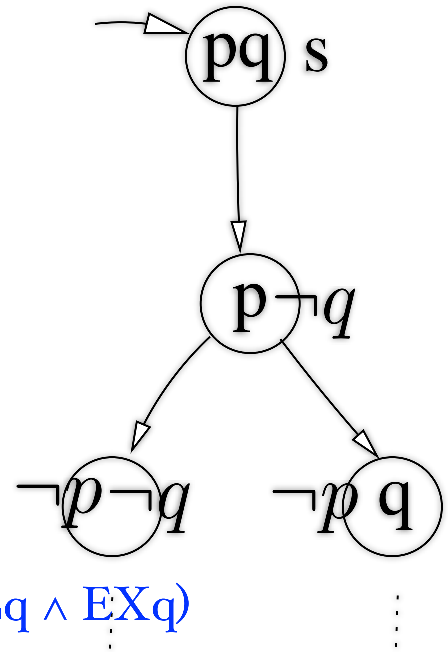
Aufgabe 1.41 Gegeben sind die folgenden Kripke-Strukturen M_1 und M_2:

LTL:

A (pq ∧ X(p ∧ (X¬q ∨ Xq))

pq, p¬q, ¬p¬q, ...

pq, p¬q, ¬p q, ...



CTL:

pq ∧ AX(p ∧ EX¬q ∧ EXq)

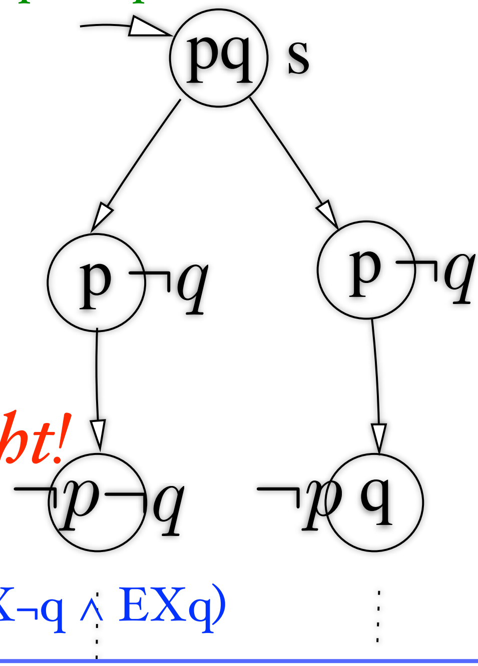
(a) M_1

LTL:

A (pq ∧ X(p ∧ (X¬q ∨ Xq))

pq, p¬q, ¬p¬q, ...

pq, p¬q, ¬p q, ...



gilt hier nicht!

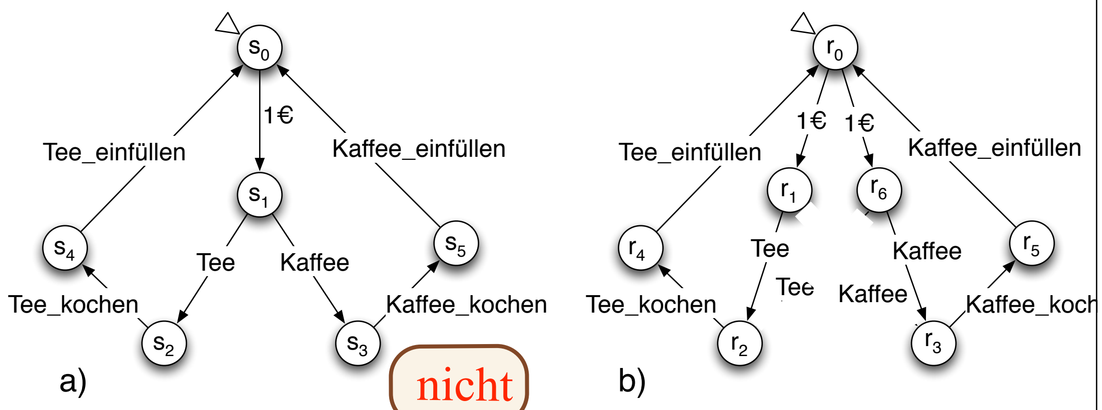
CTL:

pq ∧ AX(p ∧ EX¬q ∧ EXq)

schon mal gesehen?

LTL: folgenäquivalent

CTL: bisimilar



nicht

Abbildung 1.5: Zwei bisimilare Transitionssysteme

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

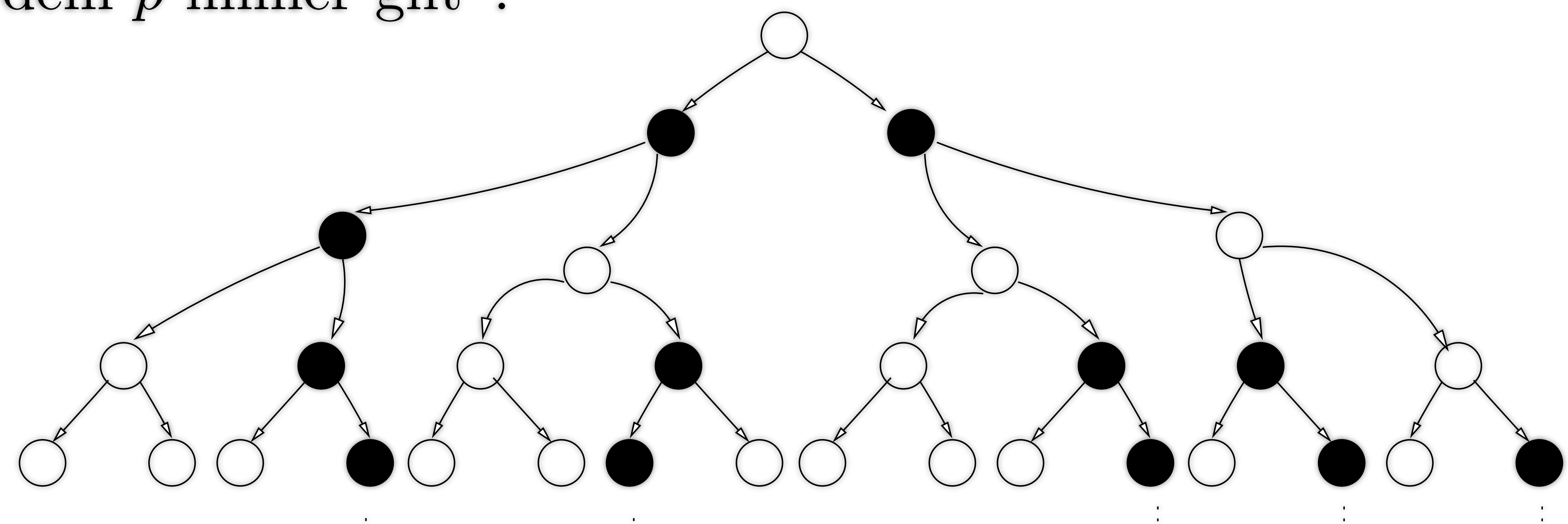
$$FGp$$

$$A(FGp)$$

$$AFAGp$$

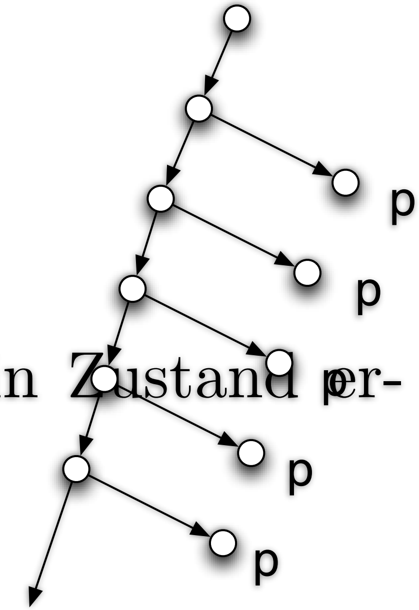
$$AFEGp$$

ist! Sie bedeutet: „auf jedem Pfad gibt es einen Zustand, ab dem p immer gilt".



Es gibt keine LTL-Formel, die äquivalent zur CTL-Formel:

$$AG(EFp)$$



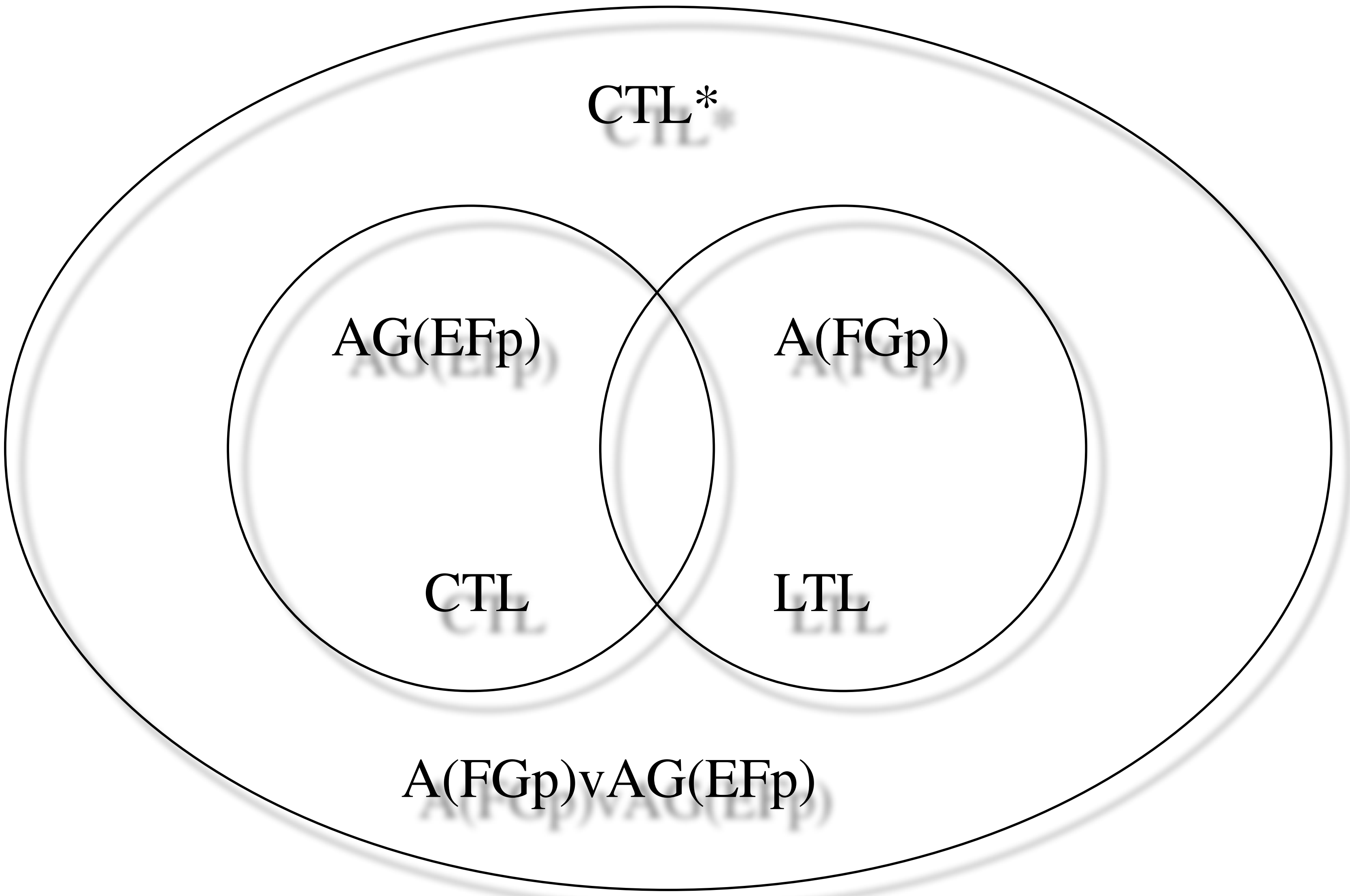
ist! Sie bedeutet: „Von jedem Zustand ist ein Zustand erreichbar, in dem p gilt."

Ist das äquivalent zu folgender Aussage?

„Alle Pfade enthalten unendlich viele Zustände, in denen p gilt."

$$\boxed{AGF\,p}$$

Es gibt eine Formel, z.B. $A(FGp) \vee AG(EFp)$, in CTL^*, die weder in CTL noch in LTL ausdrückbar ist. Also:



1.5.3 Faire Kripke-Struktur

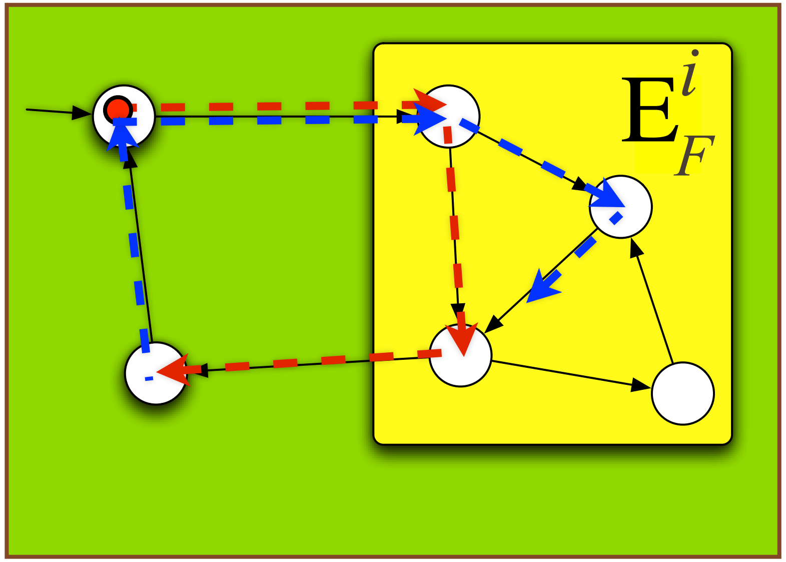
Hier zwei Beispiele:

- „eine Alternative einer sich ständig wiederholenden Alternativ wird irgendwann einmal auch gewählt" z.B. Hardware Arbiter
- „ein gestörter Kanal übermittelt immer wieder einmal eine Nachricht korrekt" z.B Alternierbitprotokoll

ausdrückbar in CTL^*, aber *nicht* in CTL. CTL ist erwünscht wegen besserer Komplexitäts-Eigenschaften bei der Analyse.
Ausweg: „faire Semantik"
Fairness-Eigenschaften oft durch Zustandsmengen ausdrückbar:

Definition 1.42 *(faire Kripke-Struktur)*

$$M := (S, S_0, R, E_S, E_F^1, \cdots, E_F^k)$$



Für $\pi = s_0 s_1 s_2 \cdots \in S^\omega$ sei

$$inf(\pi) = \{s | s = s_i \text{ für unendlich viele } i \in \mathbb{N}_0\}$$

infinite(π)

π heißt *fair* falls für jede Menge $P \in F$ gilt:

$$inf(\pi) \cap P \neq \emptyset$$

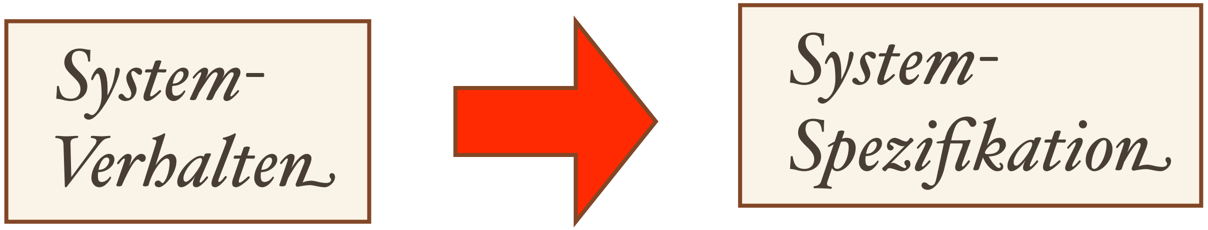
Beispiel 1.43

$$E_F^i = \{s | s \text{ erfüllt } \neg send_i \vee receive_i \text{ für Kanal } i\}$$

Ein fairer Pfad impliziert: in jedem Kanal wird unendlich oft empfangen, falls gesendet wird.

Model Checking

Verifikation eines Systems



LTL-Model Checking

Für eine gegebene Kripke-Struktur $M := (S, S_0, R, E_S)$ und eine gegebene temporal-logische Formel f ist zu berechnen:

$$\{s \in S \mid M, s \models f\}$$

M ist hier als Graph explizit gegeben.

Algorithmus für LTL-Formeln f.

Model-Checking

Verifikation eines Systems

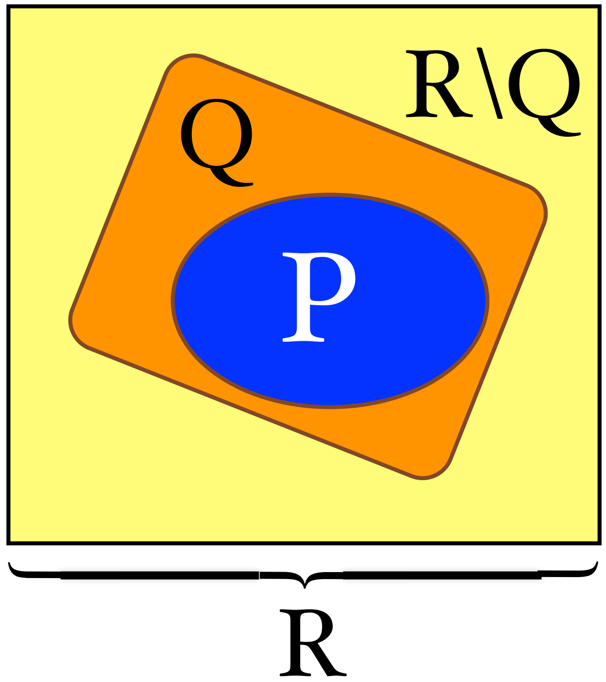
System-Verhalten $\Rightarrow$ *System-Spezifikation*

$$\cancel{L(TS_{sys}) \subseteq L(TS_{spec})}$$

$$L^{\omega}(TS_{sys}) \subseteq L^{\omega}(TS_{spec}).$$

(temporal-)logische Formel f_{spec}

$$L(TS_{sys}) \subseteq L(TS_{spec})$$



$$P \subseteq Q \Leftrightarrow P \cap (R \backslash Q) = \emptyset$$

$$\cancel{L(TS_{sys}) \cap (A^* \backslash L(TS_{spec})) = \emptyset}$$

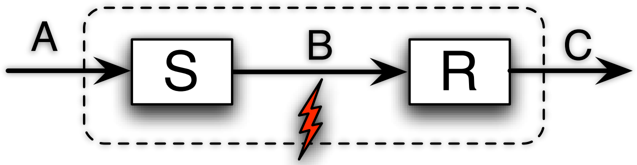
$$L^\omega(TS_{sys}) \cap \underbrace{(A^\omega \backslash \underbrace{L^\omega(TS_{spec})}_{f_{spec}})}_{\neg f_{spec}} = \emptyset$$

Satz 1.37 *Zu jeder LTL-Formel f kann eine Kripke-Struktur (ein Büchi-Automat) M konstruiert werden, die genau die für f gültigen Folgen akzeptiert: $L^\omega(M) = L^\omega(f)$.*

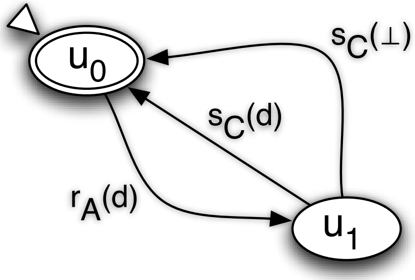
$$f \quad \rightarrow \quad M$$

$$\neg f_{spec} \rightarrow TS_{\neg spec}$$

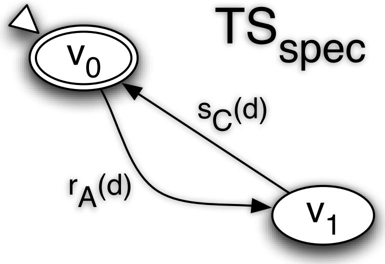
Die Zeit- und Platz-Komplexität dieses Algorithmus ist $2^{\mathcal{O}(|f|)}$



$TS_{sys} = (S \otimes R)_{extern}$

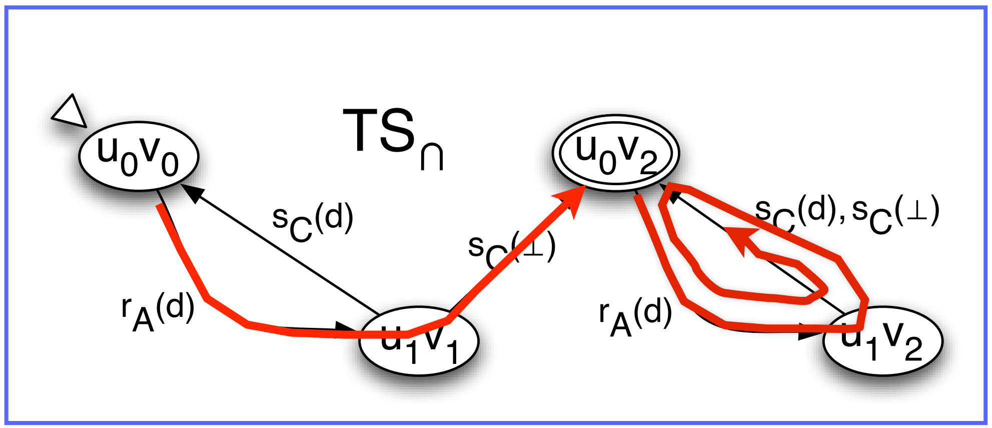


$$\neg\ f = \neg(r_A(d) \wedge [G(r_A(d) \Leftrightarrow X s_C(d))])$$



$$L^\omega(TS_{sys}) \cap L^\omega(\neg f) = \emptyset$$

$$L^\omega(TS_{sys}) \cap L^\omega(TS_{\neg spec}) = \emptyset$$



$$L^\omega(TS_\cap) \neq \emptyset$$

1.5.4 CTL-Model-Checking

Für eine gegebene Kripke-Struktur $M := (S, S_0, R, E_S)$ und eine gegebene temporal-logische Formel f ist zu berechnen:

$$\{s \in S \mid M, s \models f\}$$

M ist hier als Graph explizit gegeben.

Algorithmus für CTL-Formeln f.

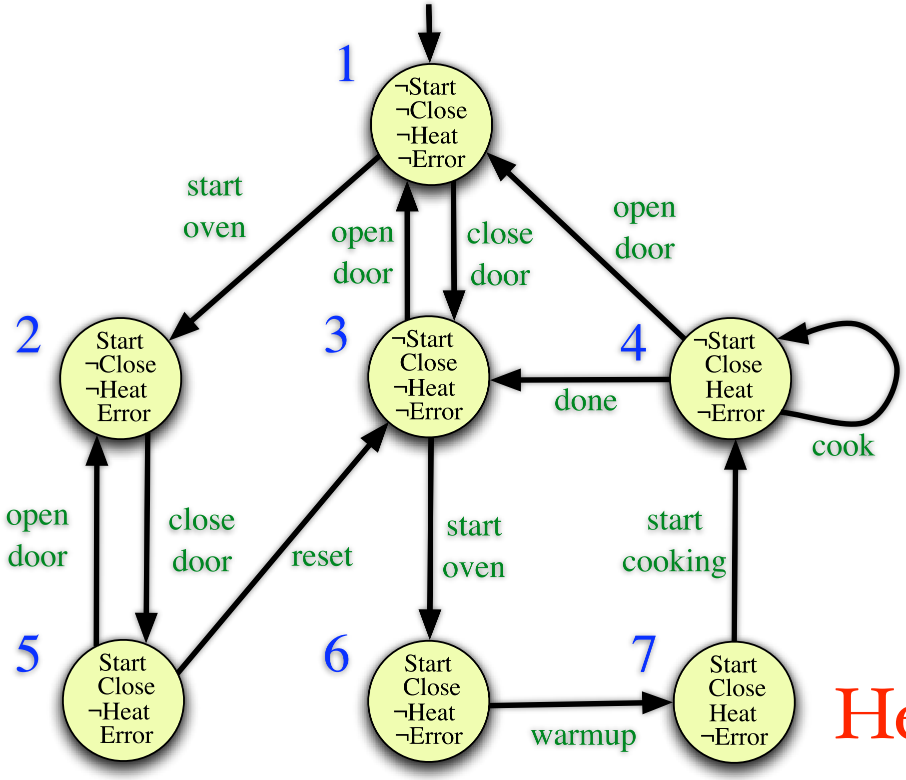
Algorithmus für CTL-Formeln f.

Erweitere $E_S(s)$ für alle $s \in S$ schrittweise zu $label(s)$. Das wird dann die Menge der Teilformeln von f, die in s wahr sind.

Rekursion über die Schachtelungstiefe von f

6 zu entwickelnde Prozeduren

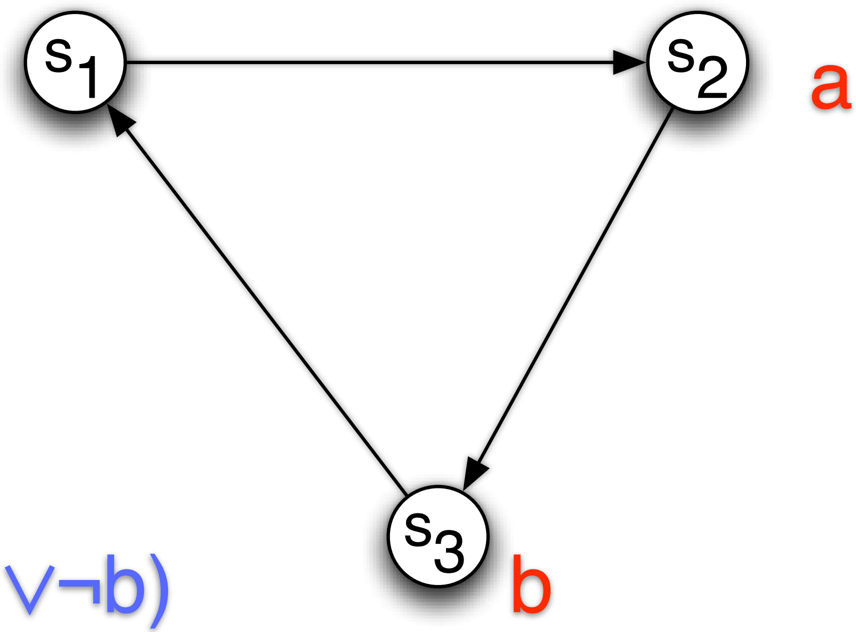
1. $f \in AP$ ist atomar
2. $f = \neg f_1$
3. $f = f_1 \vee f_2$
4. $f = EX\, f_1$
5. $f = E[f_1 U f_2]$
6. $f = EG\, f_1$



$$f = \neg\text{Start} \Rightarrow F(\boxed{\text{Heat} \wedge \neg\text{Error}})$$

Heat ∧ ¬Error

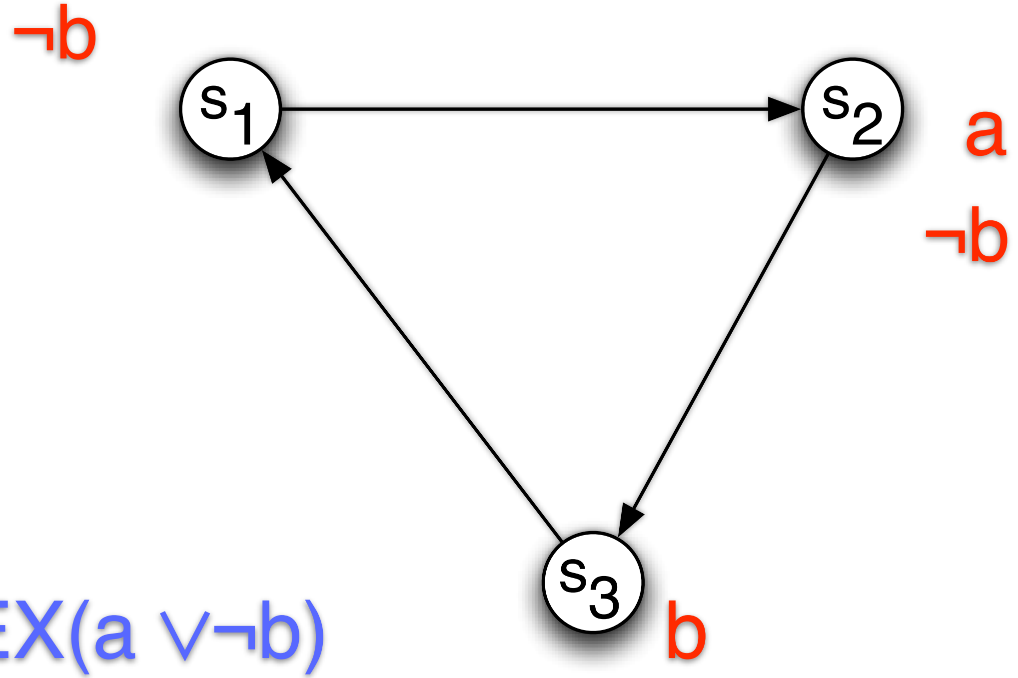
1. f atomar:
   für Zustände s mit $f \in L(s)$ setzte $f \in label(s)$.



f = EX(a ∨¬b)

1. f atomar:
   für Zustände s mit $f \in L(s)$ setzte $f \in label(s)$.

2. $f = \neg f_1$:
   für Zustände s mit $f_1 \notin label(s)$ setzte $\neg f_1 \in label(s)$.

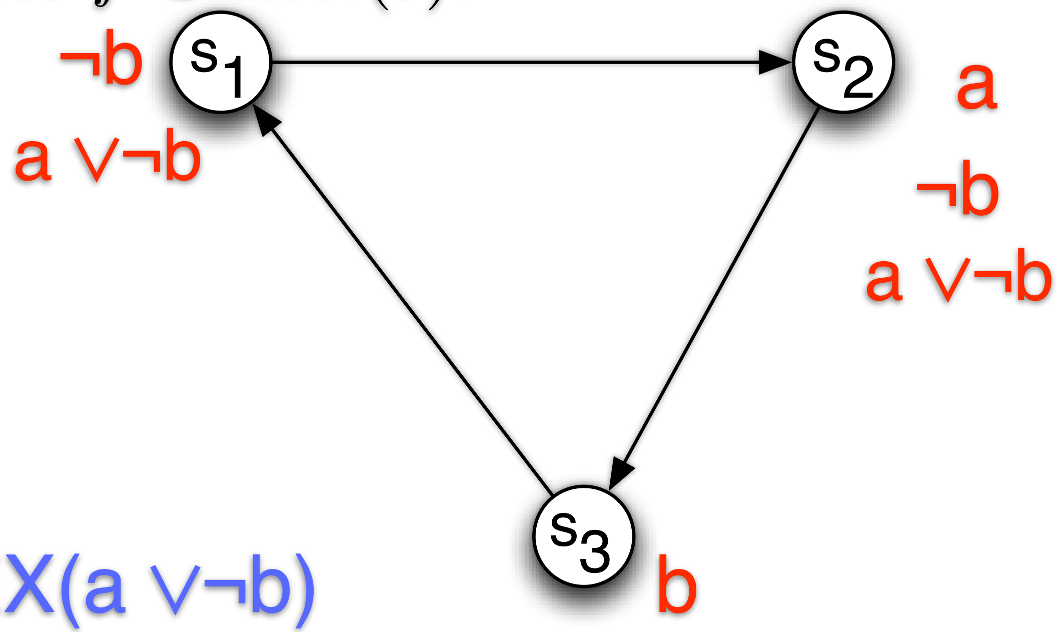


$f = EX(a \vee \neg b)$

1. f atomar:
   für Zustände s mit $f \in L(s)$ setzte $f \in label(s)$.

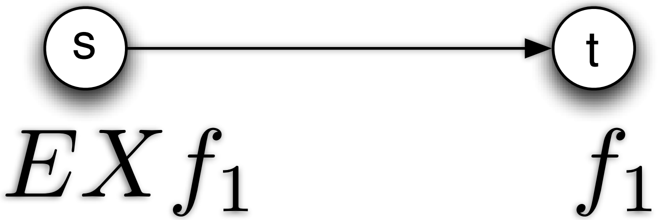
2. $f = \neg f_1$:
   für Zustände s mit $f_1 \notin label(s)$ setzte $\neg f_1 \in label(s)$.

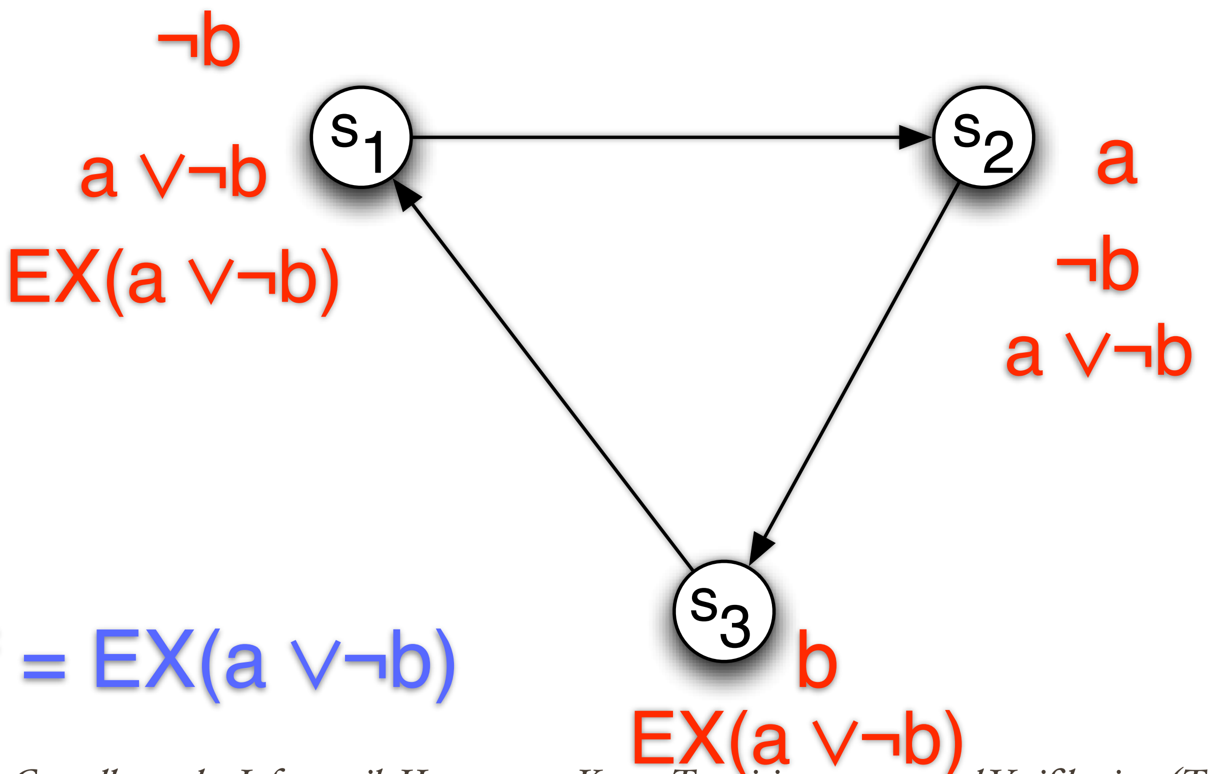
3. $f = f_1 \vee f_2$:
   für Zustände s mit $f_1 \in label(s)$ oder $f_2 \in label(s)$ setzte $f \in label(s)$.



f = EX(a ∨¬b)

4. $f = EX f_1$:
für Zustände s mit $R(s,t)$ und $f_1 \in label(t)$ setzte $f \in label(s)$.

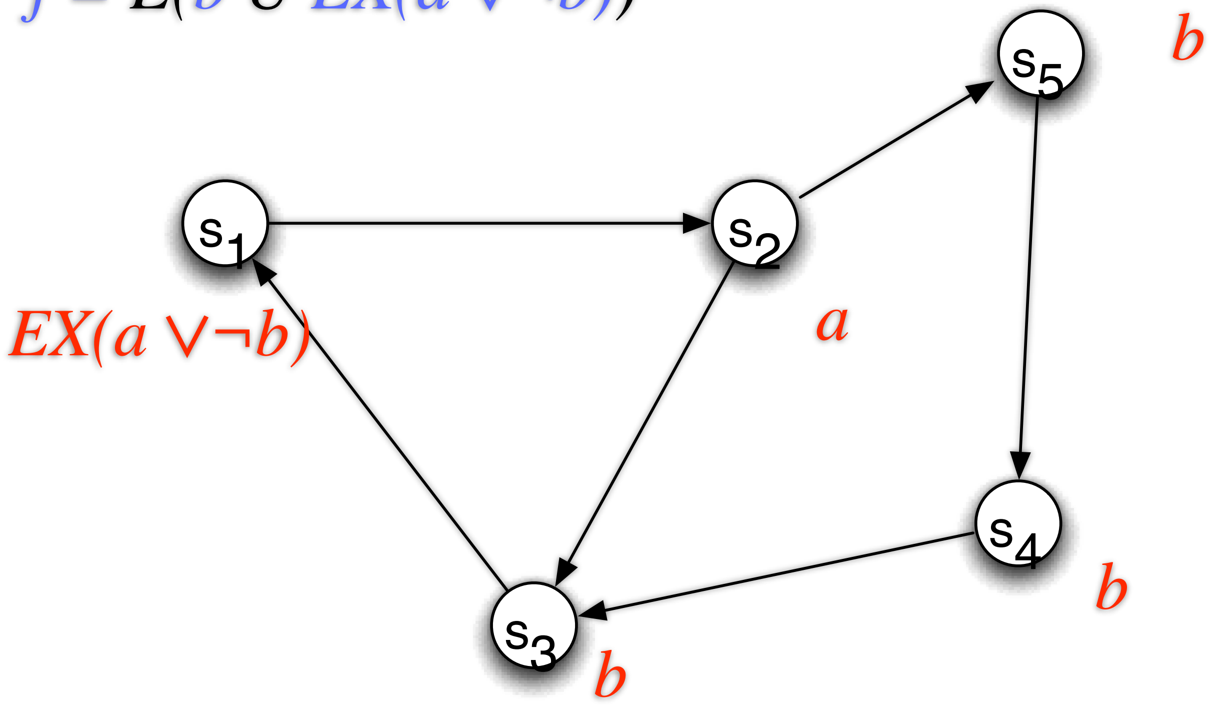




f = EX(a ∨¬b)

5. $f = E[f_1 U f_2]$:

$f = E(b\ U\ EX(a \vee \neg b))$



5. $f = E[f_1 U f_2]$:

für Zustände s mit $f_2 \in label(s)$ setzte $f \in label(s)$;
für Zustände t mit $R(t, s)$ und $f_1 \in label(s)$ setze $f \in label(t)$;

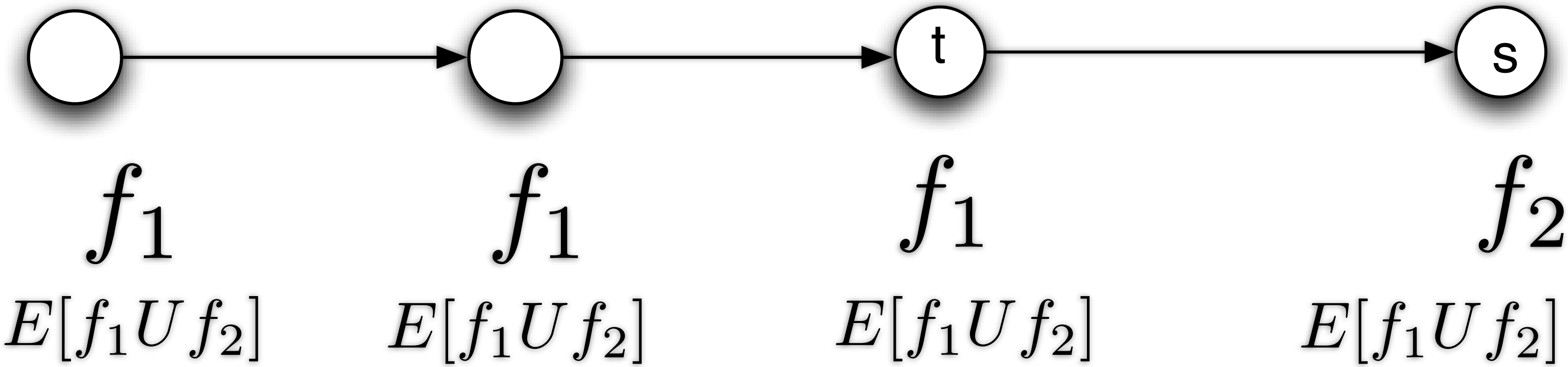
f_1 $\qquad\qquad$ f_2

$E[f_1 U f_2]$ $\qquad$ $E[f_1 U f_2]$

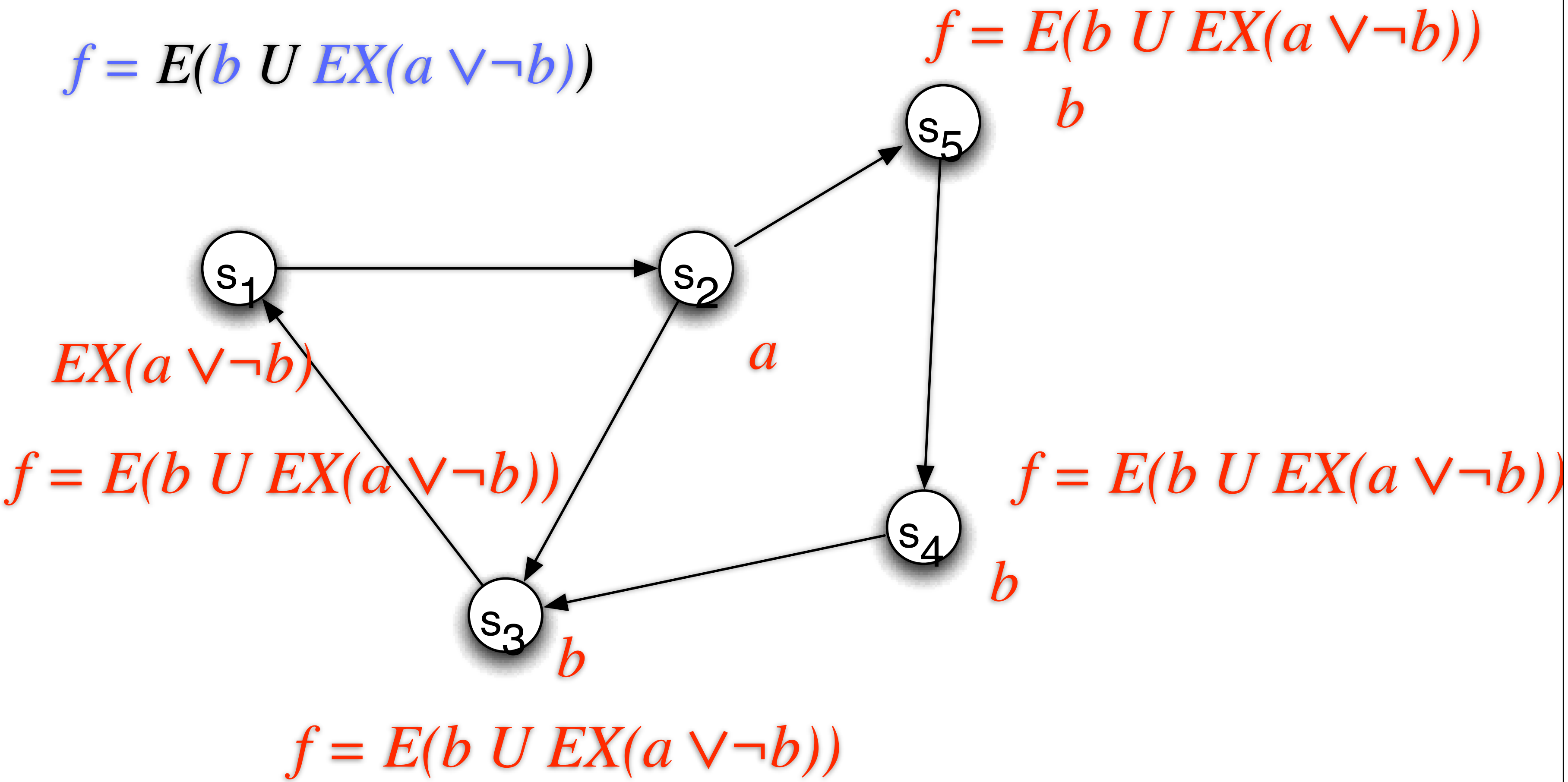
5. $f = E[f_1 U f_2]$:

für Zustände s mit $f_2 \in label(s)$ setzte $f \in label(s)$;
für Zustände t mit $R(t, s)$ und $f_1 \in label(s)$ setze $f \in label(t)$;

fahre schrittweise in Gegenrichtung der Transitionen fort und setze $f \in label(s)$, falls es einen Pfad von s zu einem s' mit $f_2 \in label(s')$ gibt, auf dem für alle Zustände t davor $f_1 \in label(t)$ gilt. Siehe Algorithmus 5.5.

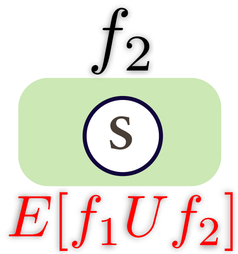


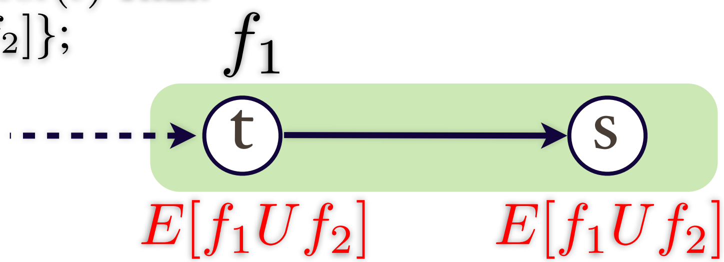
$f = E(b \; U \; EX(a \vee \neg b))$



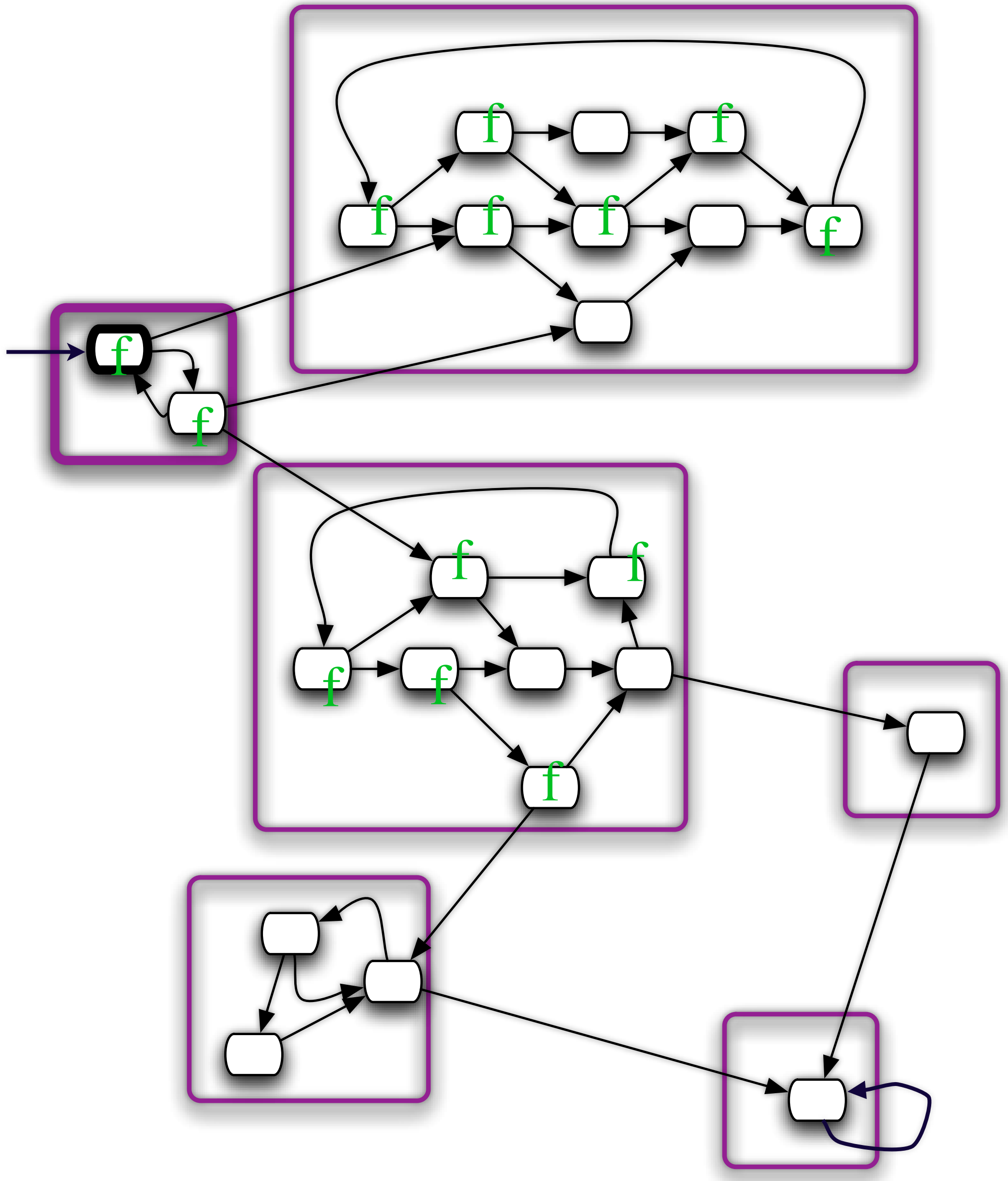
Algorithmus 1.6 Auszeichnen mit $E(f_1 U f_2)$

PROCEDURE $CheckEU(f_1, f_2)$
    $T := \{s \mid f_2 \in label(s)\}$;
    FOR ALL $s \in T$ DO $label(s) := label(s) \cup \{E[f_1 U f_2]\}$;
    WHILE $T \neq \emptyset$ DO
        CHOOSE $s \in T$;
        $T := T \backslash \{s\}$;
        FOR ALL t SUCH THAT $R(t, s)$ DO
            IF $E[f_1 U f_2] \notin label(t)$ AND $f_1 \in label(t)$ THEN
                $label(t) := label(t) \cup \{E[f_1 U f_2]\}$;
                $T := T \cup \{t\}$;
            END IF ;
        END FOR ALL ;
    END WHILE ;
END PROCEDURE ;

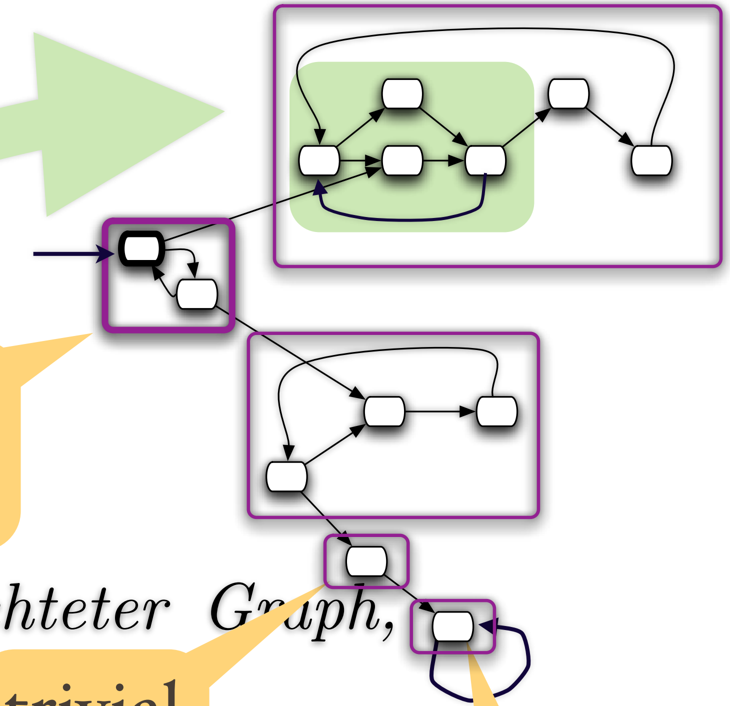




EG f ??



EG f ??



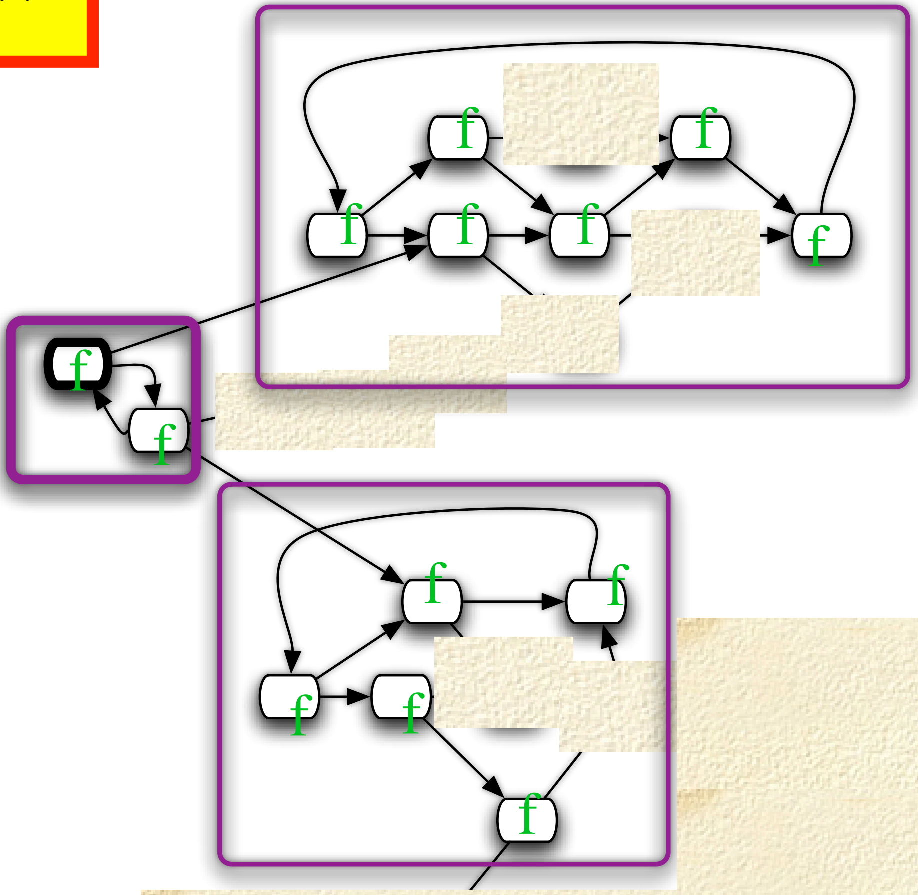
Definition 1.44 *Sei $G = (K, R)$ ein gerichteter Graph, d.h.: $R \subseteq K \times K$:*

a) *$A \subseteq K$ heißt* Zusammenhangskomponente, *falls: $\forall a, a' \in A : aR^*a'$.*

b) *Sie heißt* strenge Zusammenhangskomponente (SZK) *(strongly connected component: SZK), falls sie maximal ist, d.h.: $\neg \exists k \in K \setminus A \,.\, \forall a \in A : kR^*a \wedge aR^*k$.*

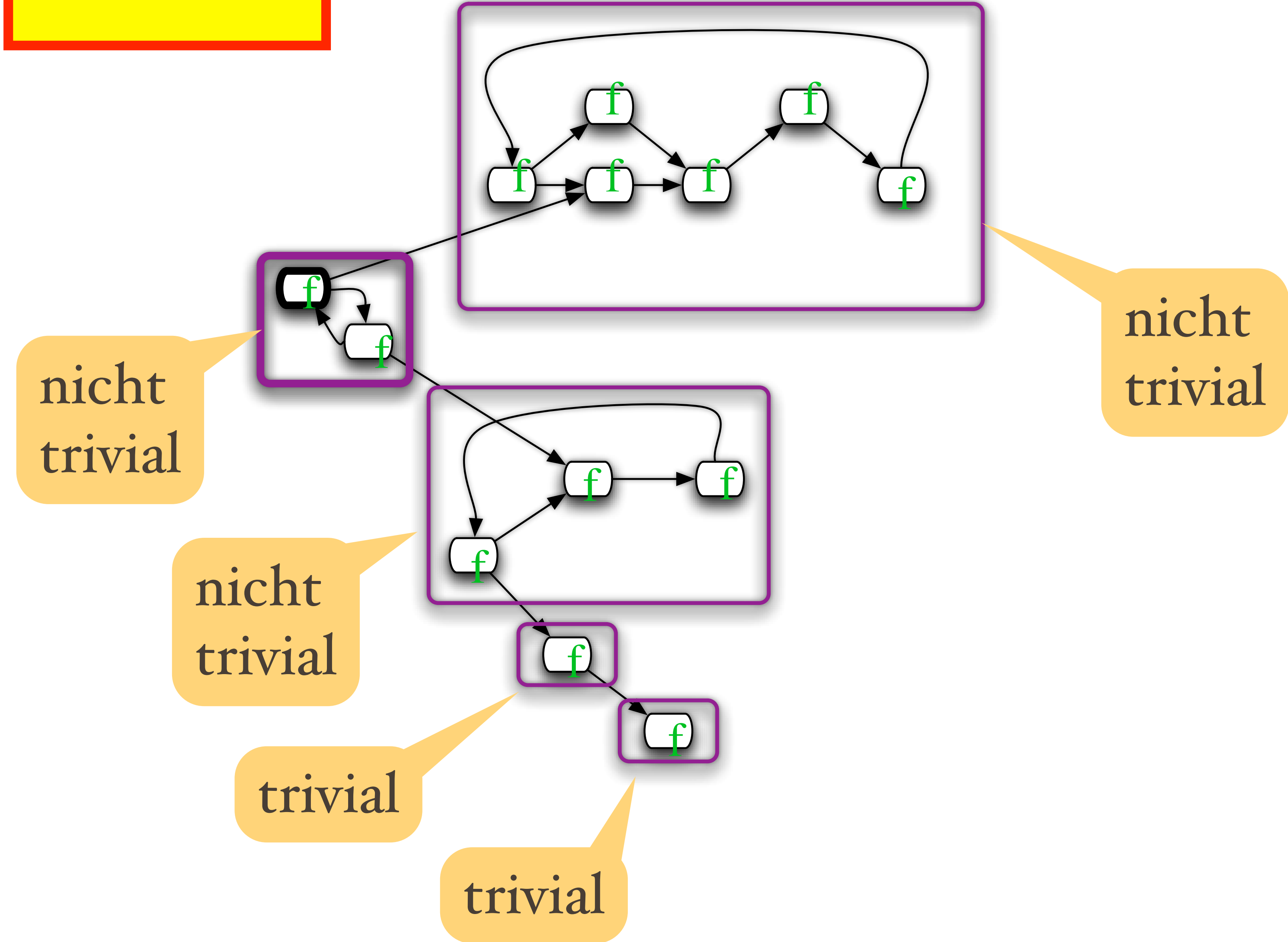
c) *Sie heißt* nichttriviale Zusammenhangskomponente, *falls: $|A| > 1$ oder $\exists a \in A \,.\, aR^+a$.*

EG f ??



SZK

EG f ??



f
f
f
f
f
f
f
f
f
f
f
f
f
f
nicht trivial
nicht trivial
nicht trivial
trivial
trivial

Nun betrachten wir wieder die Formel: $f = EGf_1$:
Aus $M = (S, R, L)$ konstruiere $M' = (S', R', L')$ mit:

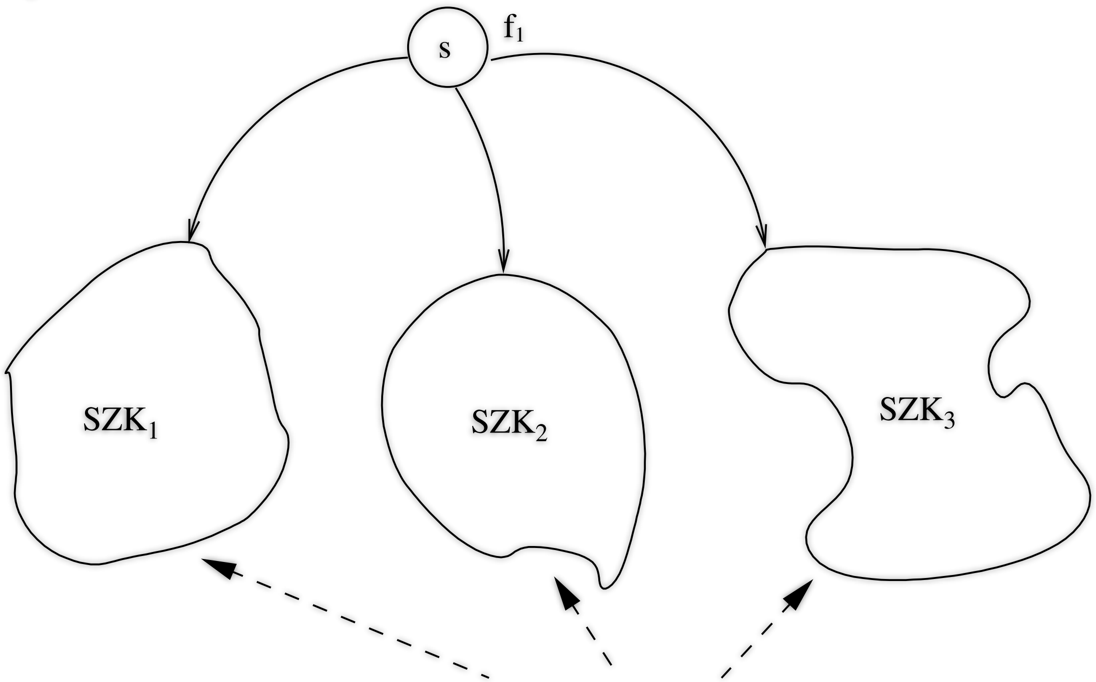
$$\begin{aligned} S' &= \{s \in S \mid M, s \models f_1\} \\ R' &= R_{|S' \times S'} \\ L' &= L_{|S'} \end{aligned}$$

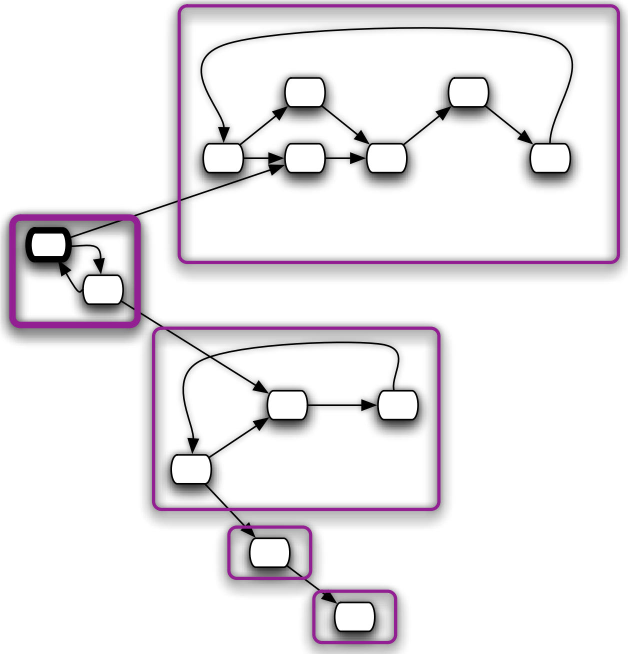
d.h. die „Einschränkung" von M, in der f_1 gilt.

Lemma 1.45 $M, s \models EGf_1 \;\; gdw.$

1. $s \in S'$

2. *Es gibt einen Pfad in M', der von s zu einer nicht-trivialen starken Zusamenhangskomponente in (S', R') führt.*





Daraus Algorithmus zur Entscheidung von $EG f_1$:

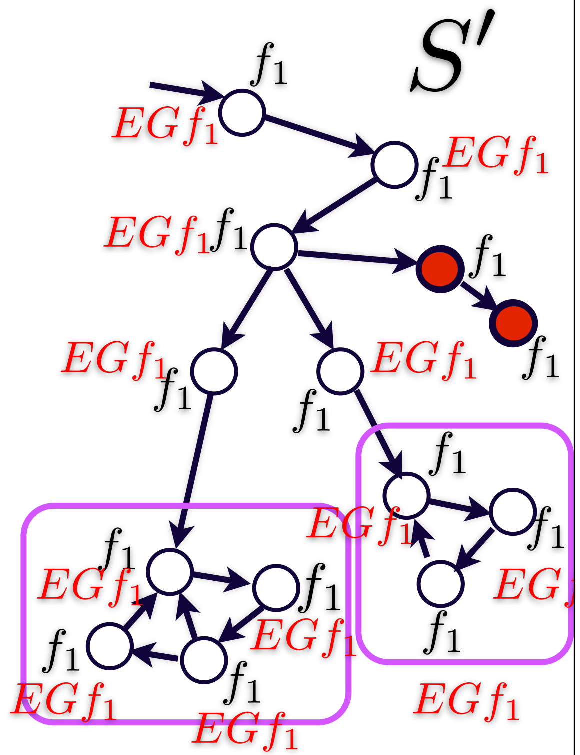
1. Konstruiere $M' = (S', R', L')$
2. Konstruiere alle SZK von M'. (Algorithmus von Tarjan mit $O(|S'| + |R'|)$ Zeitkomplexität).
3. Finde Zustände in nichttrivialen SZK.
4. Suche von diesen rückwärts alle Zustände die dorthin führen.

   insgesamt: $O(|S| + |R|)$. Siehe Algorithmus 5.6.

Algorithmus 1.7 Auszeichnen mit EGf_1

Seite 178

```
PROCEDURE CheckEGf1
    S' := {s | f1 ∈ label(s)};
    SCC := {C | C a nontrivial SCC of S'};
    T := ⋃_{C∈SCC} {s | s ∈ C};
    FOR ALL s ∈ T DO label(s) := label(s) ∪ {EGf1};
    WHILE T ≠ ∅ DO
        CHOOSE s ∈ T;
        T := T\{s};
        FOR ALL t SUCH THAT t ∈ S' AND R(t, s) DO
            IF EGf1 ∉ label(t) THEN
                label(t) := label(t) ∪ {EGf1};
                T := T ∪ {t};
            END IF ;
        END FOR ALL ;
    END WHILE ;
END PROCEDURE ;
```



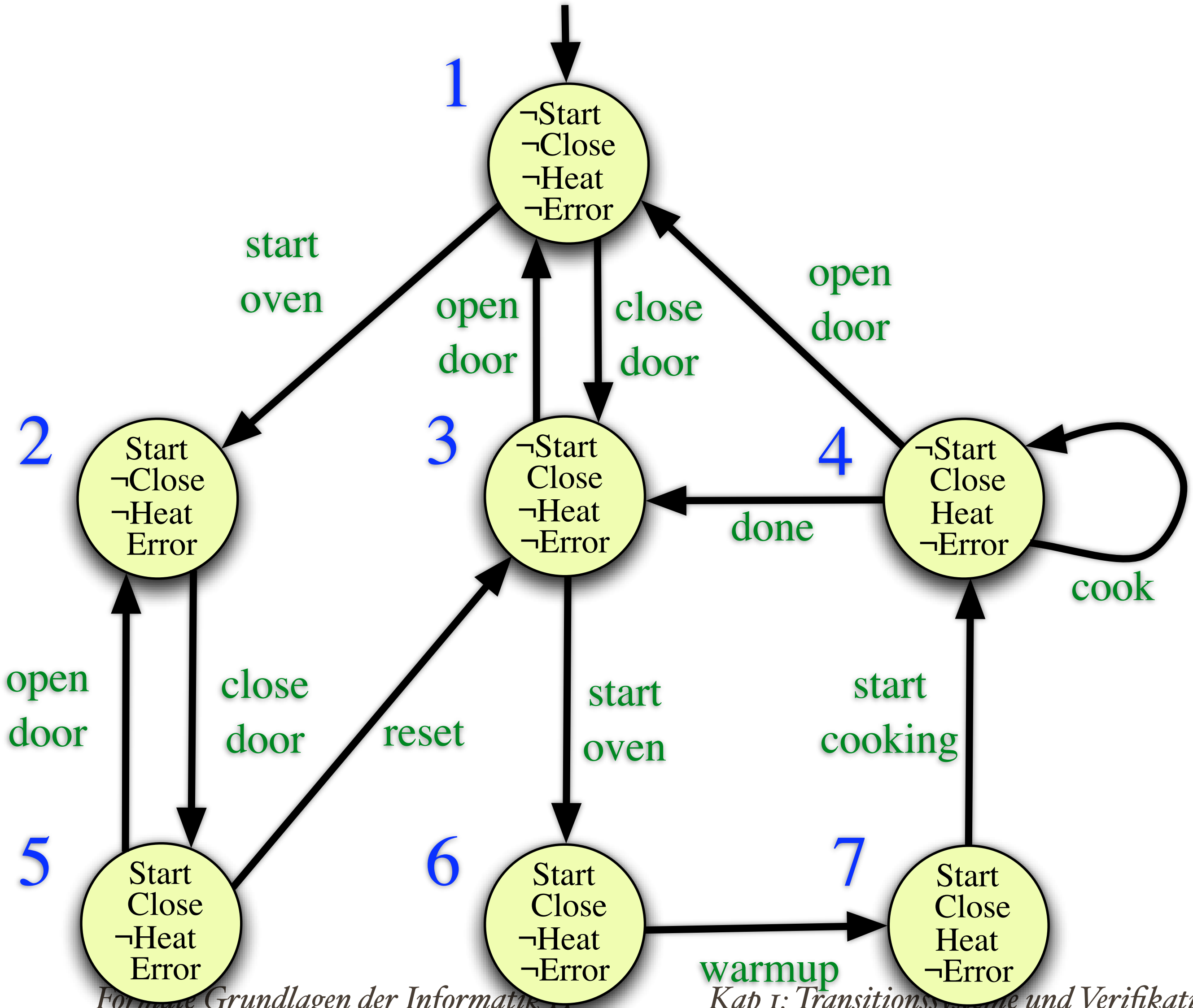
Satz 1.46 *Es gibt einen Algorithmus, der für eine Kripke-Struktur* $M := (S, S_0, R, E_S)$ *und eine CTL-Formel* f *in* $\mathcal{O}(|f| \cdot (|S| + |R|))$ *Zeitkomplexität entscheidet, ob* f *für* M *gilt, d.h. ob* $M \models f$.

Beweis:
Wende obiges Verfahren auf die Atome von f an und fahre induktiv fort mit den Teilformeln von f, aufsteigend mit deren Schachtelung. Die Schachtelungstiefe ist durch $\mathcal{O}(|f|)$ begrenzt. Auf jeder Ebene gibt es maximal $\mathcal{O}(|f| \cdot (|S| + |R|))$ Operationen. $\square$

$$f = AG(Start \rightarrow AF\, Heat)$$

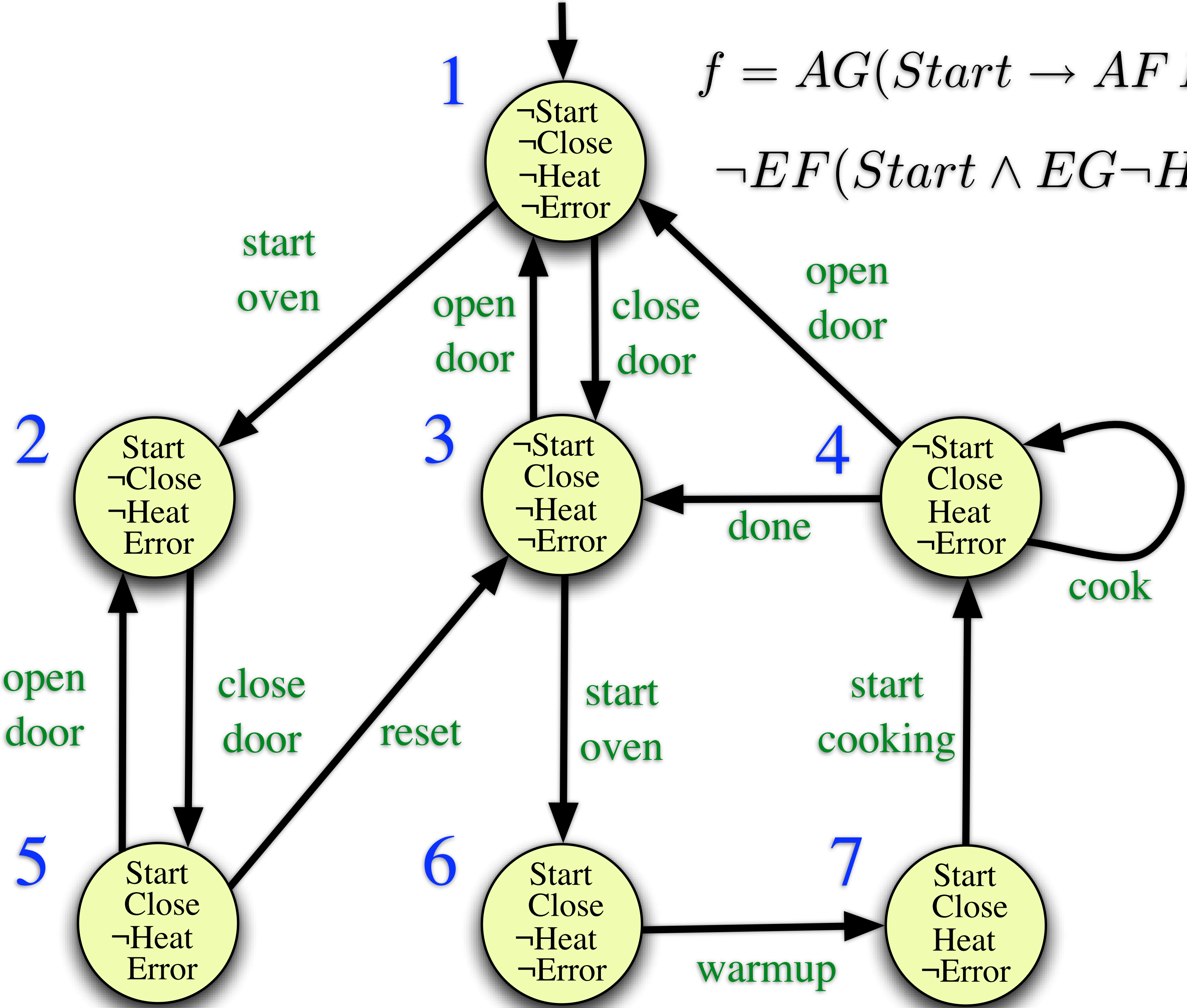
Es gilt immer: nach einem Zustand mit „Start” wird später ein Zustand mit „Heat” erreicht.

Beispiel: Mikrowellenofen



$$f = AG(Start \rightarrow AF\,Heat)$$

$$\neg EF(Start \wedge EG \neg Heat)$$



$$f_1$$

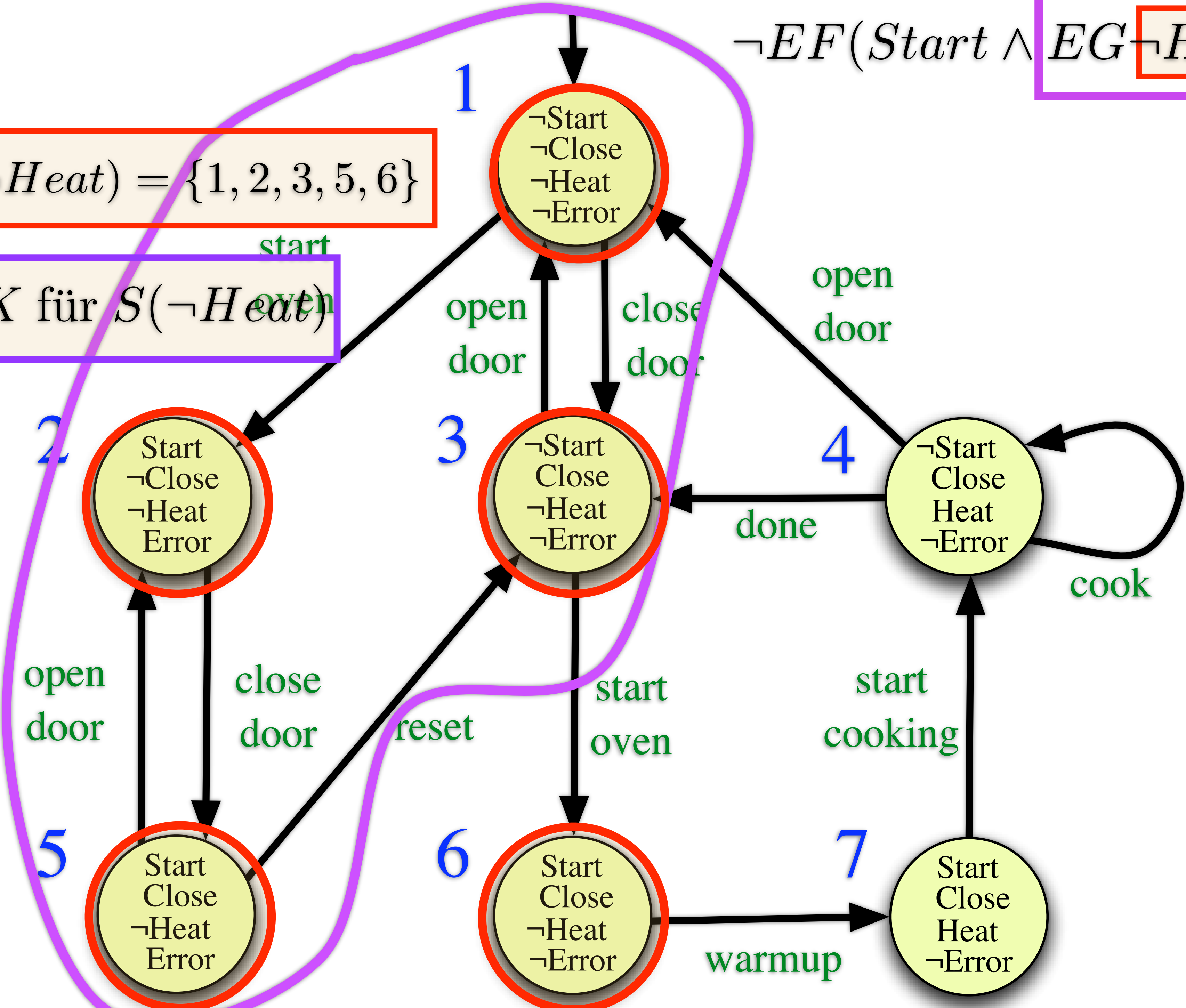
Umschreibung von $f := AG(\boxed{Start \rightarrow AF\,Heat})$

$$\begin{aligned}
AG f_1 &= \neg EF(\neg f_1) \\
&= \neg EF(\neg(\neg Start \vee AF\,Heat)) \\
&= \neg EF(Start \wedge \neg AF\,Heat) && (AF f = \neg EG(\neg f)) \\
&= \neg EF(Start \wedge EG \neg Heat) && (EF f \equiv E[True\, U\, f])
\end{aligned}$$

$$\neg EF(Start \wedge \boxed{EG\boxed{\neg Heat}})$$

$$\boxed{S(\neg Heat) = \{1, 2, 3, 5, 6\}}$$

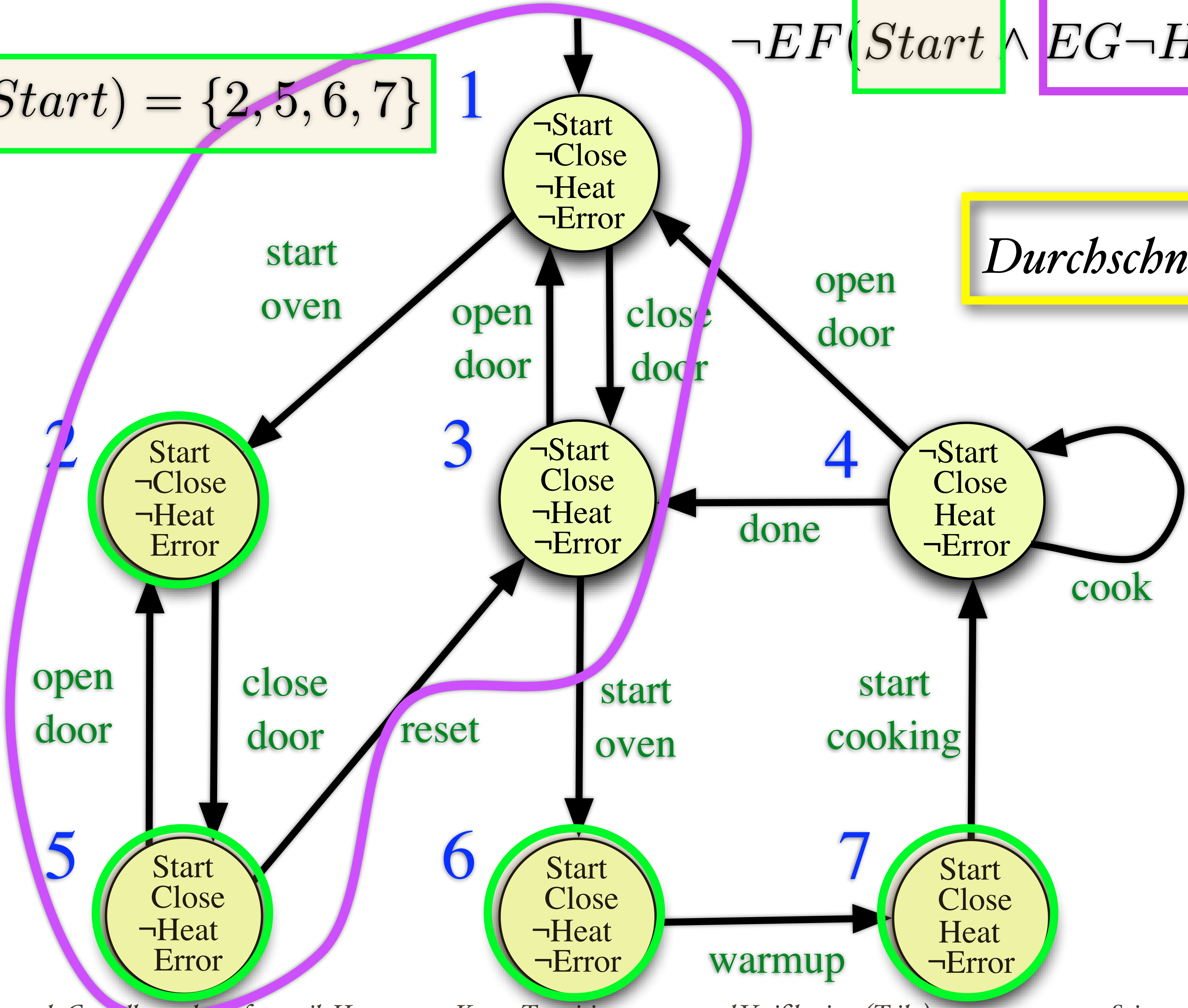
$\boxed{SZK \text{ für } S(\neg Heat)}$



$\boxed{S(Start) = \{2, 5, 6, 7\}}$

$\neg EF(\boxed{Start} \wedge \boxed{EG\neg Heat})$

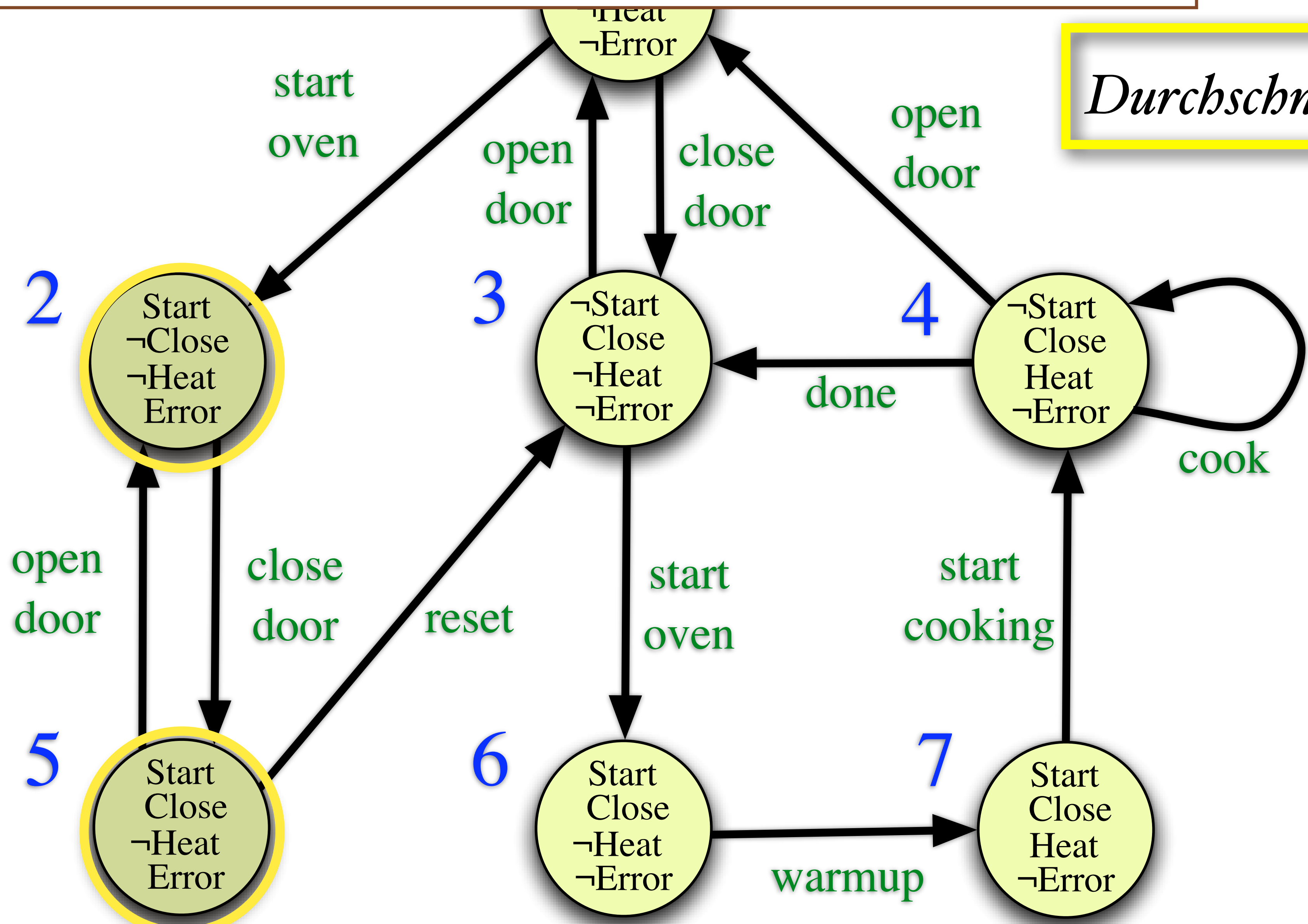
Durchschnitt??



$$(\boxed{Start \wedge EG\neg Heat})) :$$

$$S(\neg EF(Start \wedge EG\neg Heat)) = \emptyset$$

Durchschnitt??



Was wurde bewiesen?

$f = AG(Start \rightarrow AF\ Heat)$

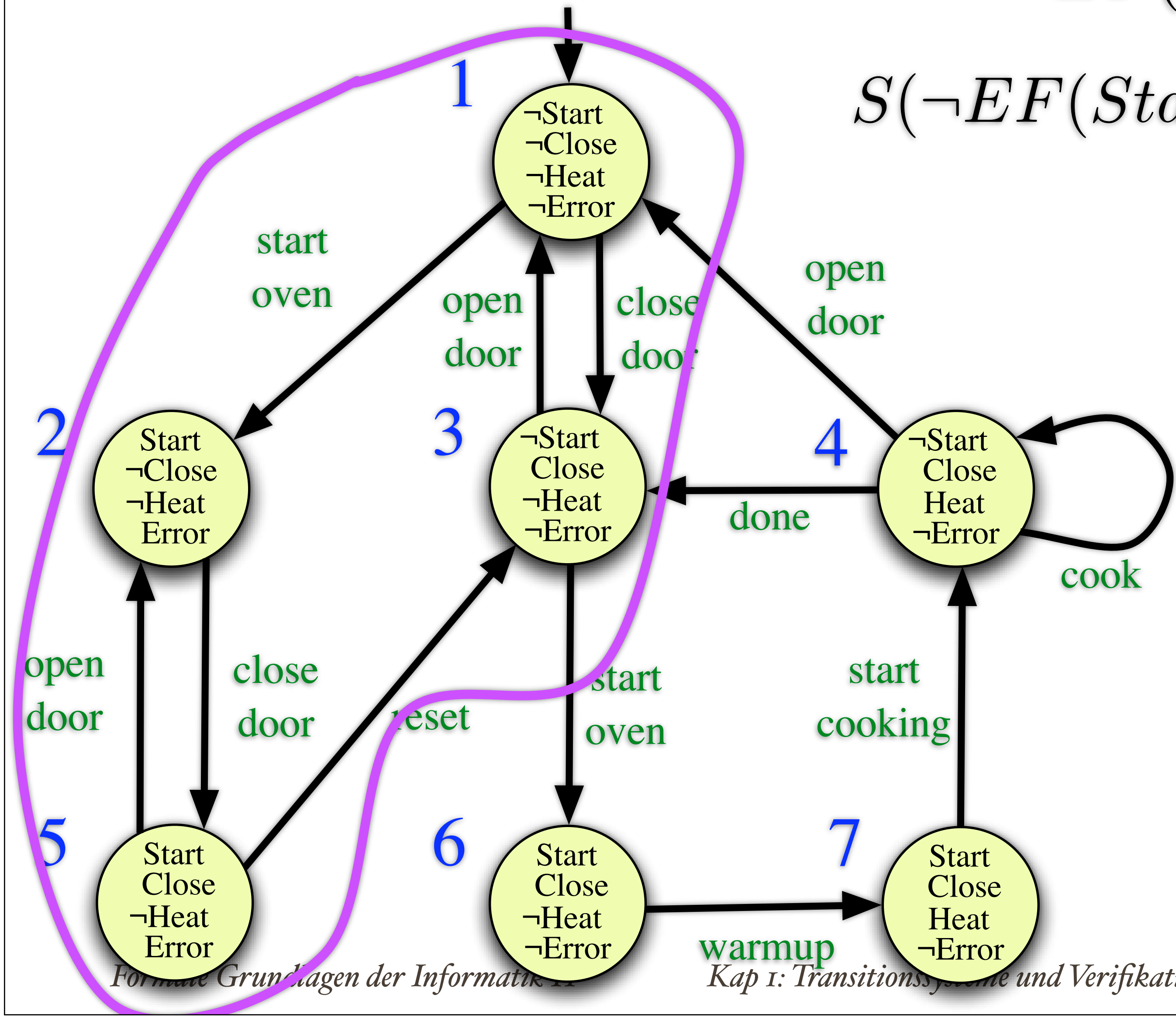
Es gilt immer: nach einem Zustand mit „Start” wird später ein Zustand mit „Heat” erreicht. gilt nicht!

$\neg EF(Start \wedge EG \neg Heat)$

$S(\neg EF(Start \wedge EG \neg Heat)) = \emptyset$

Gegenbeispiel:

$1, 3, 1, (2,5)^{\omega}$



CTL-Model-Cecking mit Fairness

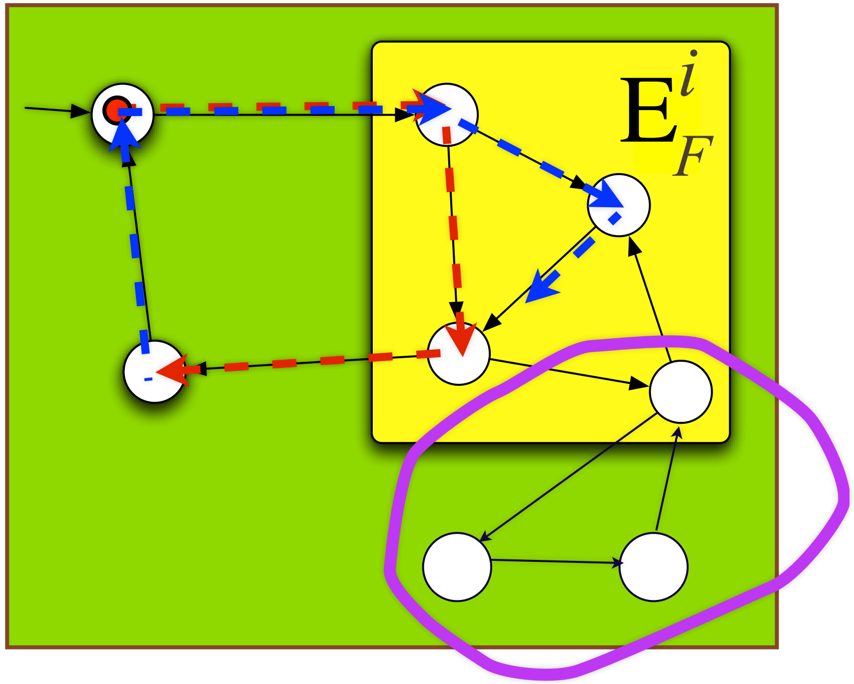
Definition 1.48 *Sei* $M := (S, S_0, R, E_S, E_F^1, \cdots, E_F^k)$ *eine faire Kripke-Struktur. Eine starke Zusammenangskomponente* $C \subseteq S$ *heißt* $\boxed{\textit{fair}}$*, falls* $\forall i \in \{1, \cdots, k\} : E_F^i \cap C \neq \emptyset$.

Ferner sei $M' := (S', S_0', R', E_S', F_1, \cdots, F_k)$ *mit:*

$$\begin{aligned} S' &= \{s \in S \mid M, s \models_F f_1\} \\ R' &= R_{|S' \times S'} \\ E_S' &= E_{S|S'} \\ F_i &= E_F^i \cap C \end{aligned}$$

($\models_F$ *siehe Seite 255*)

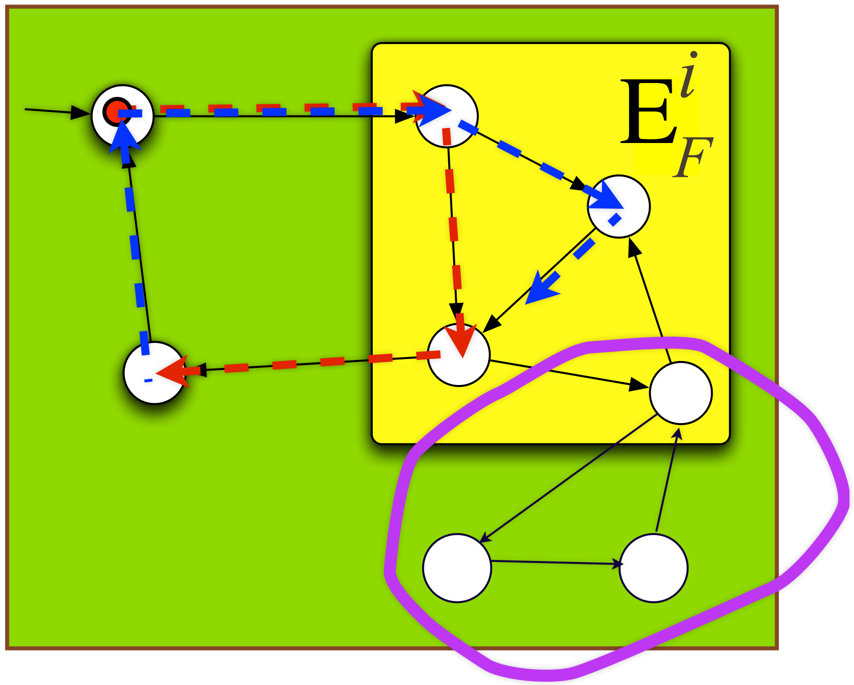
$EG f$

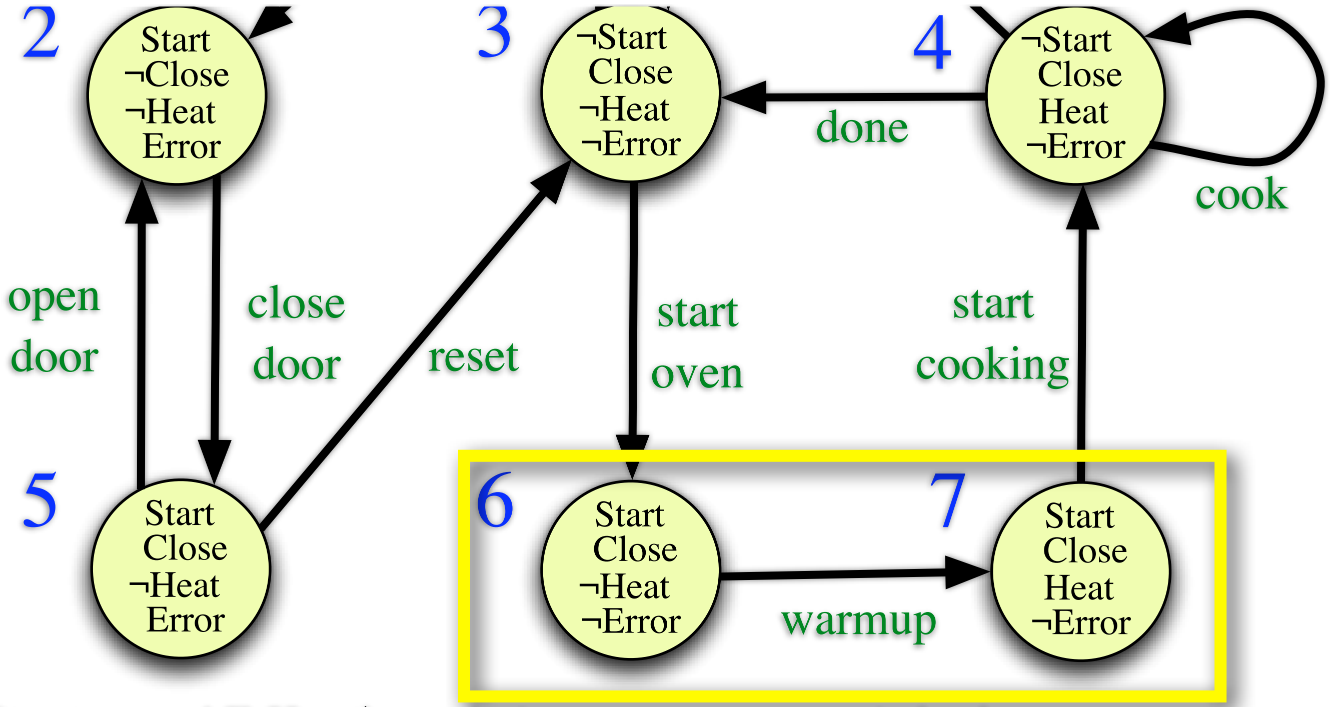


CTL-Model-Cecking mit Fairness

Lemma 1.49 *Es gilt genau dann $M, s \models_F EGf_1$, wenn (1.) $s \in S'$ und (2.) es einen Pfad in M' gibt, der von s zu einer fairen nichttrivialen starken Zusamenhangskomponente in (S', R') führt.*

$$EGf$$





Beispiel 1.51

Prüfe $f = AG(Start \to AF\,Heat)$, wobei vorausgesetzt wird, dass die Benutzer den Ofen immer korrekt bedienen. Dabei interpretieren wir „immer korrekt bedienen“ als „unendlich oft gilt $Start \wedge Close \wedge \neg Error$“.

„immer”	$\hat{=}$	„unendlich oft”
„korrekt bedienen”	$\hat{=}$	„unendlich oft gilt $Start \wedge Close \wedge \neg Error$”

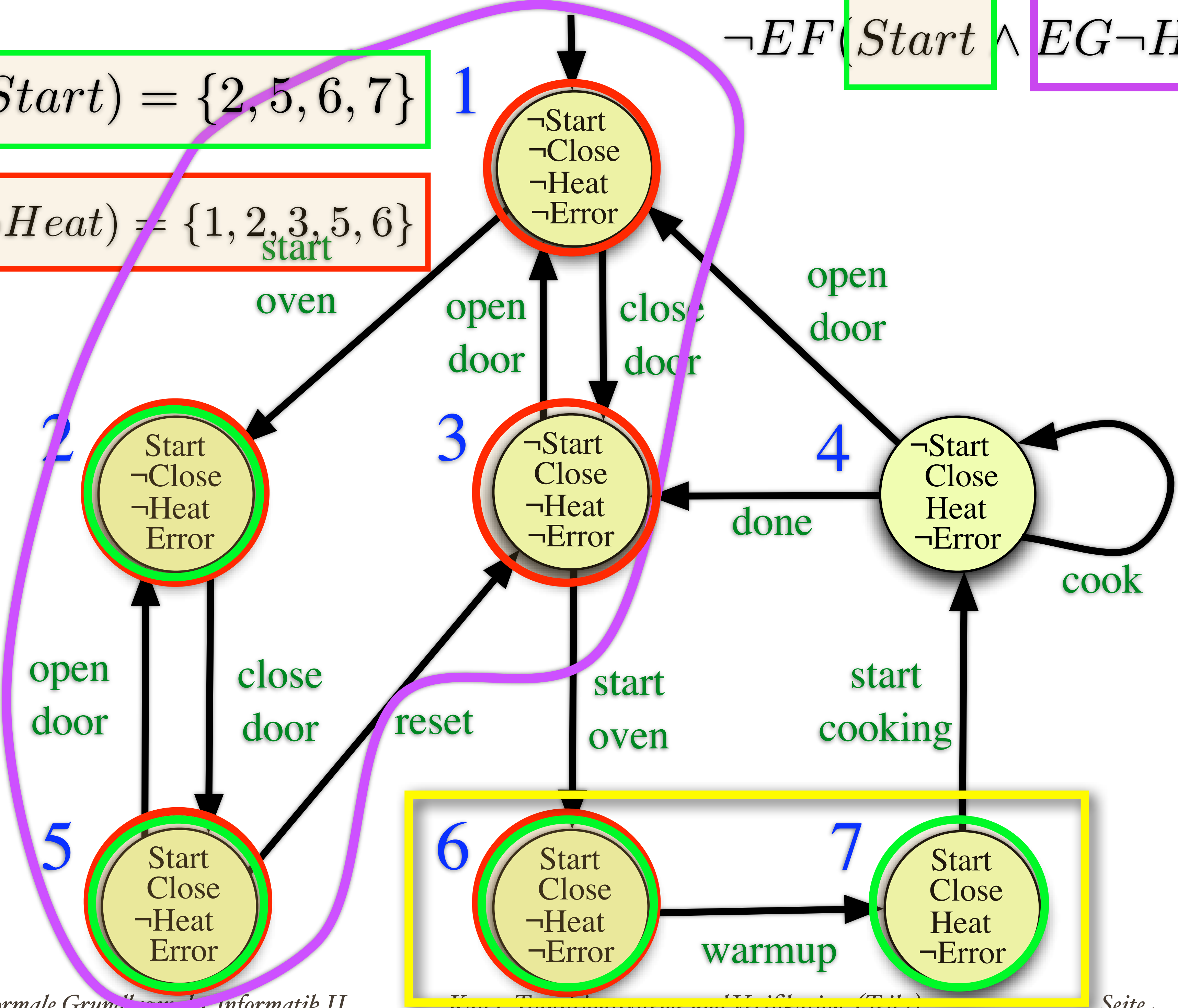
Daher setzten wir $M := (S, S_0, R, E_S, E_F^1)$ (also $k = 1$) mit

$$E_F^1 = \{s \mid s \models Start \wedge \; Close \wedge \;\; Error)\} = \{6, 7\}.$$

$\neg EF(\boxed{Start} \wedge \boxed{EG\neg Heat})$

$\boxed{S(Start) = \{2, 5, 6, 7\}}$

$\boxed{S(\neg Heat) = \{1, 2, 3, 5, 6\}}$



Mit $S(Start)$, $S(\neg Heat)$ wie oben, erhält man:

$$\{1, 2, 3, 5\}$$

nicht fair, da disjunkt zu P. Also:

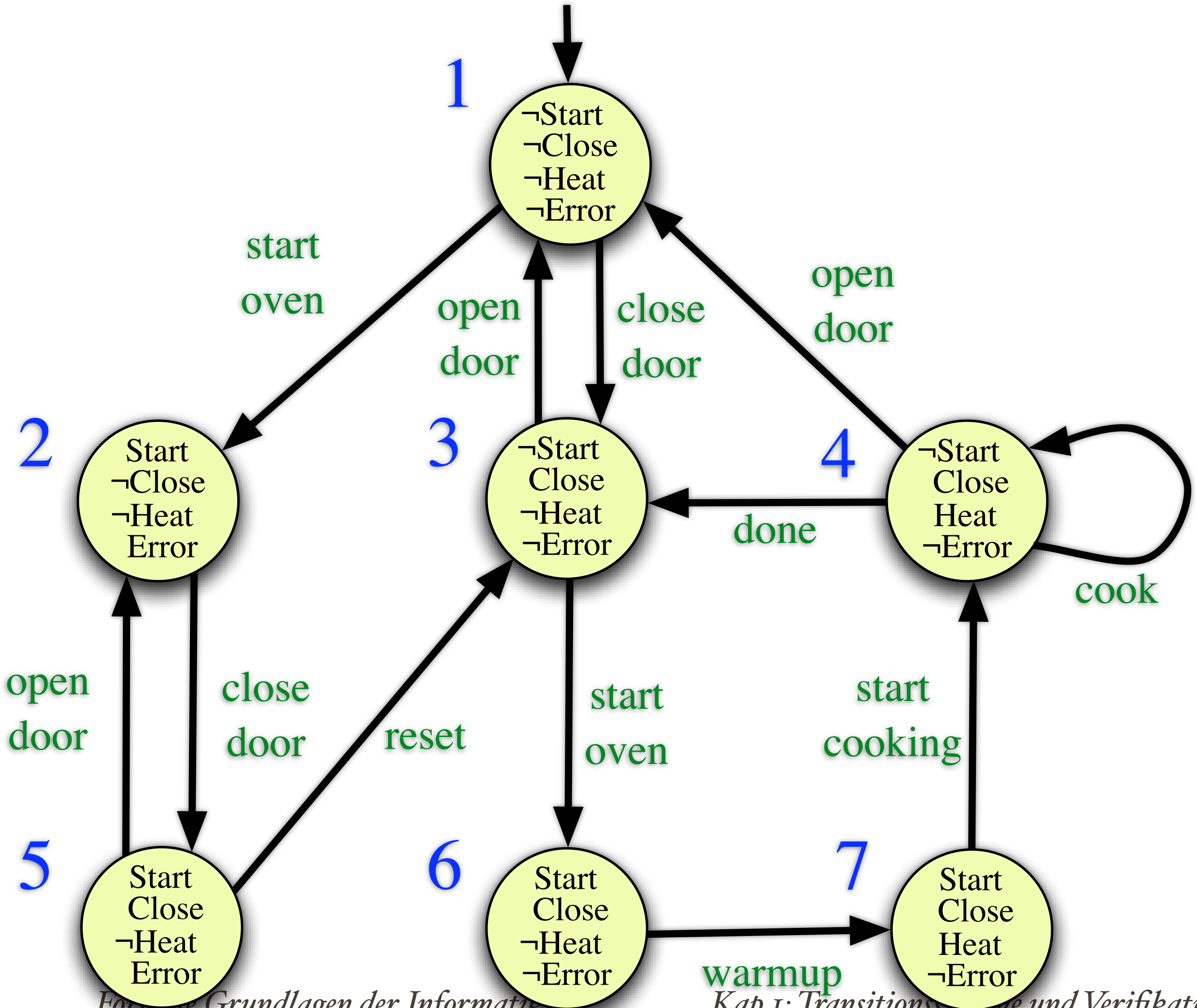
$$\begin{aligned} S(EG\neg Heat) &= \emptyset \\ S(EF(Start \wedge EG\neg Heat)) &= \emptyset \\ S(\neg EF(Start \wedge EG\neg Heat)) &= \{1, \cdots, 7\} \end{aligned}$$

Die Spezifikation ist in der fairen Semantik erfüllt, da die Formel f im Anfangszustand gilt.

Prüfe $f = AG(Start \rightarrow AF\, Heat)$, wobei vorausgesetzt wird, dass die Benutzer den Ofen immer korrekt bedienen.

Was wurde bewiesen?

„immer"	$\triangleq$	„unendlich oft"
„korrekt bedienen"	$\triangleq$	„unendlich oft gilt $Start \wedge Close \wedge \neg Error$"



gilt !

Gegenbeispiel:
keines

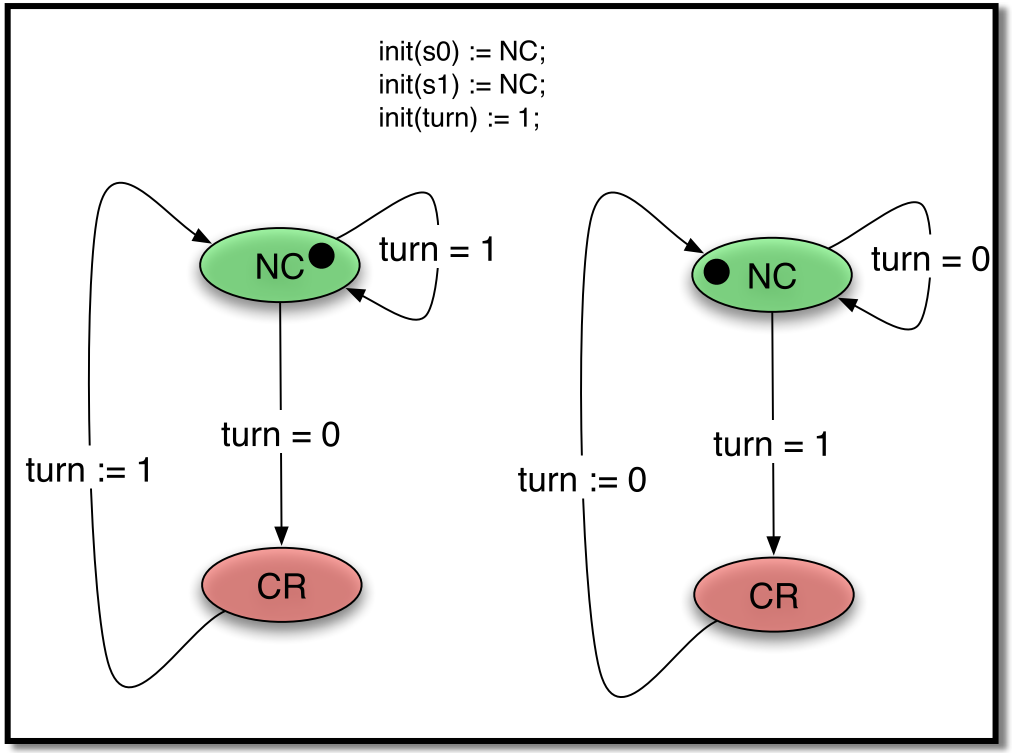
Model Checker NuSMV („Symbolic Model Checker“)

ein Programm zum wechselseitigen Ausschluss:

```
MODULE main
VAR s0: {NC, CR}; s1: {NC, CR}; turn:
boolean;

ASSIGN
init(s0) := NC;
init(s1) := NC;
init(turn) := 1;

next(s0) :=
case
   (s0 = NC) & (turn = 1) : NC;
   (s0 = NC) & (turn = 0) : CR;
   (s0 = CR) : NC;
   1: s0;
esac;

next(s1) :=
case
   (s1 = NC) & (turn = 0) : NC;
   (s1 = NC) & (turn = 1) : CR;
   (s1 = CR) : NC;
   1: s1;
esac;

next(turn) :=
case
    s0 = CR: 1;
    s1 = CR: 0;
   1: turn;
```

critical

non critical

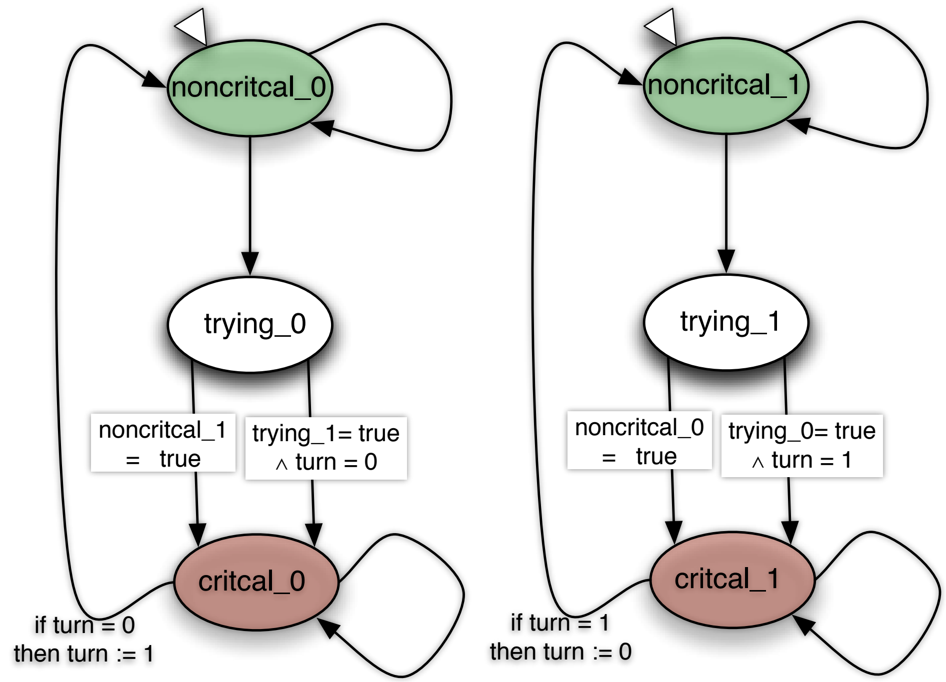


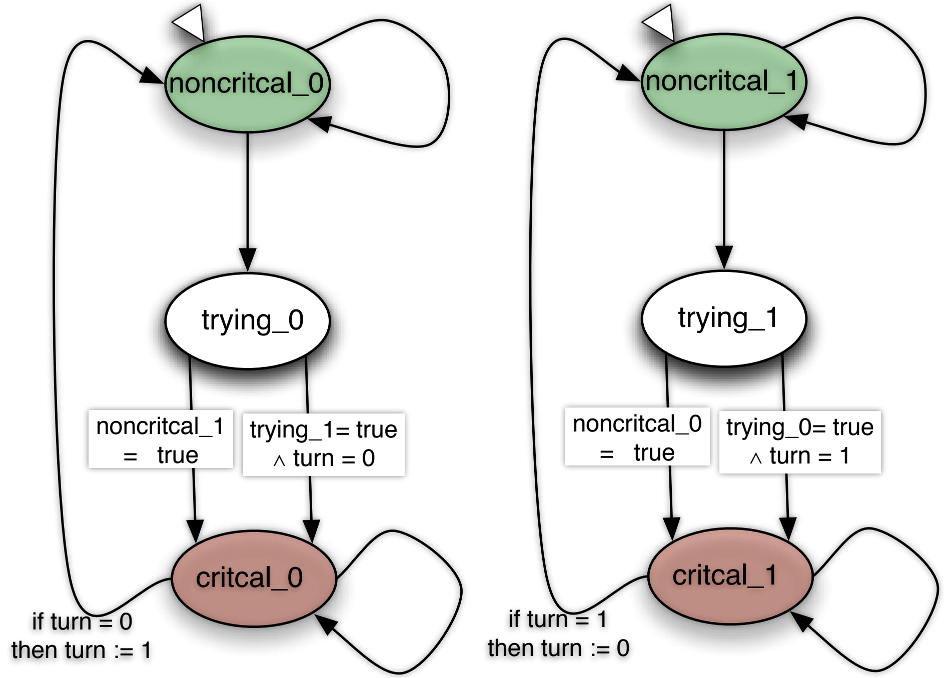
SPEC AG((s0 = NC) -> AF(s0 = CR))

SPEC AG(!(s0 = CR & s1 = CR))

der NuSMV-Code dazu:

```
MODULE main

VAR

s0: {noncritical, trying, critical};

s1: {noncritical, trying, critical};

turn: boolean;

pr0: process prc(s0, s1, turn, 0);

pr1: process prc(s1, s0, turn, 1);

ASSIGN

init(turn) := 0;

 FAIRNESS !(s0 = critical)

FAIRNESS  !(s1 = critical)

SPEC   AG(!((s0 = critical) & (s1 = critical)))

SPEC   AG((s0 = trying) -> AF (s0 = critical))

--  SPEC   AG((s1 = trying) -> AF (s1 = critical))
```





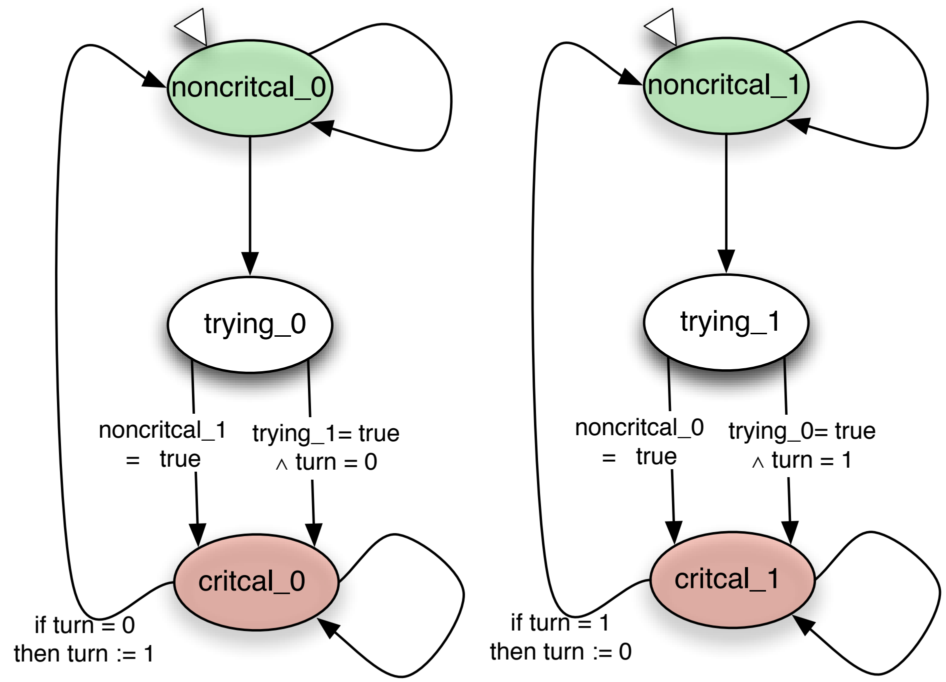

```
MODULE prc(state0, state1, turn, turn0)

ASSIGN

init(state0) := noncritical;

next(state0) :=
case
   (state0 = noncritical) : {trying,noncritical};
   (state0 = trying) & (state1 = noncritical): critical;
   (state0 = trying) & (state1 = trying) & (turn = turn0):  critical;
   (state0 = critical) : {critical,noncritical};
   1: state0;
esac;
```

- 1/2 -

```
SPEC   AG(!((s0 = critical) & (s1 = critical)))

SPEC   AG((s0 = trying) -> AF (s0 = critical))

--  SPEC    AG((s1 = trying) -> AF (s1 = critical))

SPEC    AG((s0 = critical) -> A[(s0 = critical) U (!(s0 = critical) & A[!(s0 =
critical) U (s1 = critical)])])

-- SPEC     AG((s1 = critical) -> A[(s1 = critical) U (!(s1 = critical) & A[!(s1 =
critical) U (s0 = critical)])])

MODULE prc(state0, state1, turn, turn0)

ASSIGN

init(state0) := noncritical;

next(state0) :=
case
   (state0 = noncritical) : {trying,noncritical};
   (state0 = trying) & (state1 = noncritical): critical;
   (state0 = trying) & (state1 = trying) & (turn = turn0):  critical;
   (state0 = critical) : {critical,noncritical};
   1: state0;
esac;
```

Model Checker NuSMV („Symbolic Model Checker“)

```
3mutex2.smv
MODULE main

VAR

s0: {noncritical, trying, critical};

s1: {noncritical, trying, critical};

turn: boolean;

pr0: process prc(s0, s1, turn, 0);

pr1: process prc(s1, s0, turn, 1);

ASSIGN

init(turn) := 0;
```

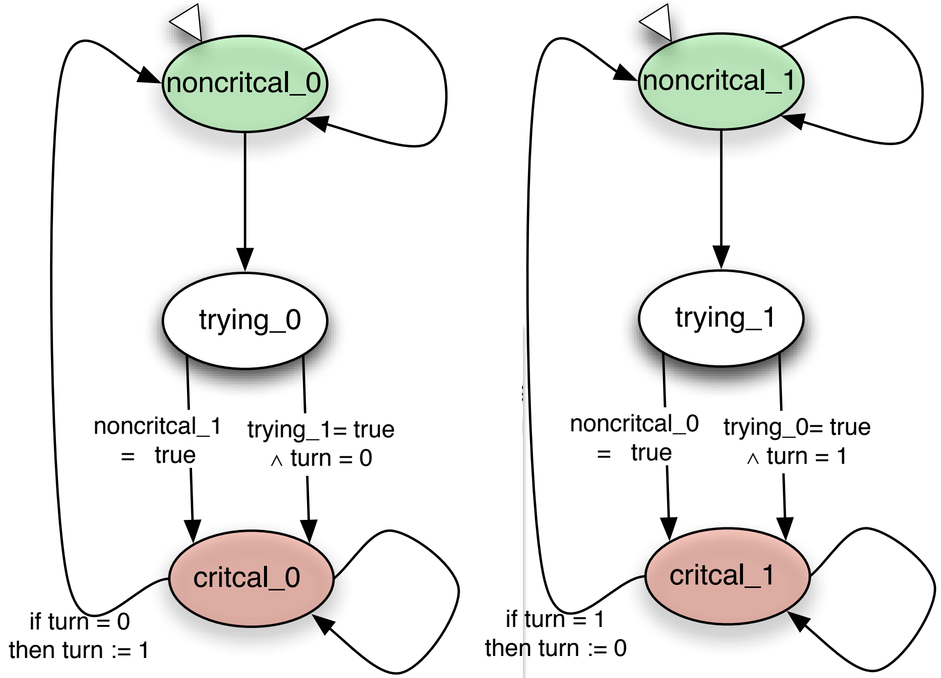


```
pr0: process prc(s0, s1, turn, 0);

pr1: process prc(s1, s0, turn, 1);
```

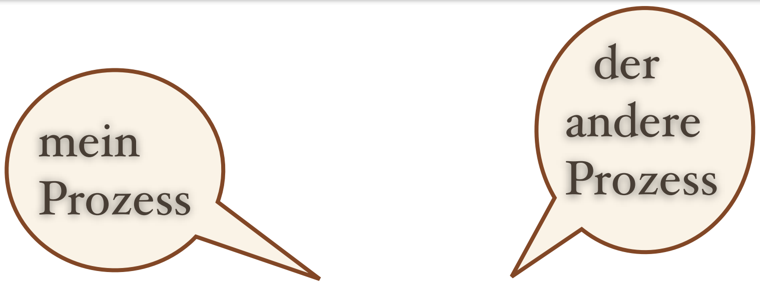
mein Prozess

der andere Prozess

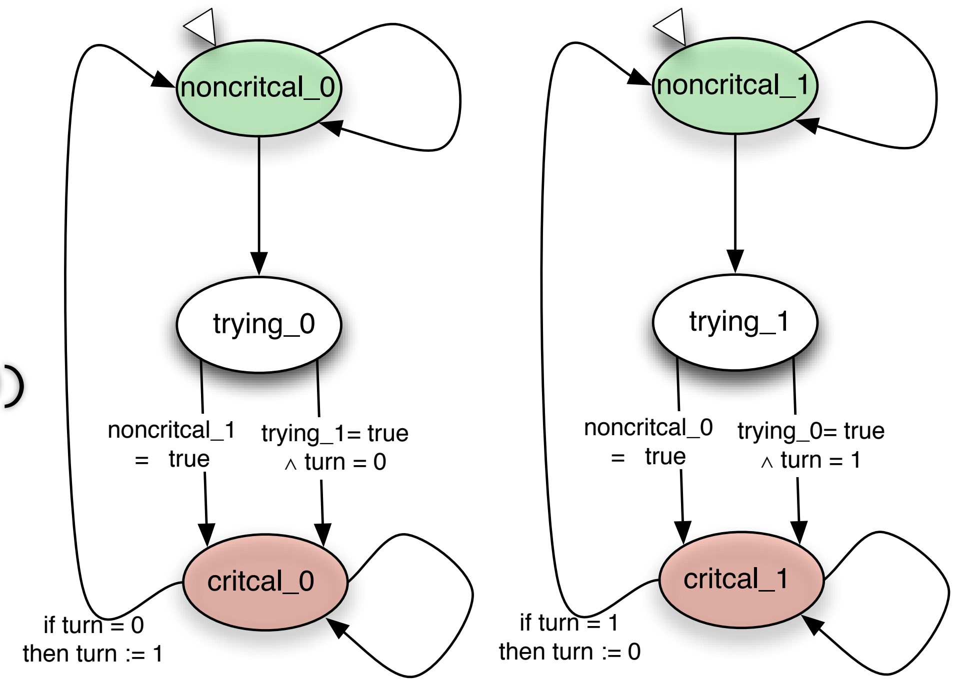
```
MODULE prc(state0, state1, turn, turn0)

ASSIGN

init(state0) := noncritical;

next(state0) :=
case
   (state0 = noncritical) : {trying,noncritical};
   (state0 = trying) & (state1 = noncritical): critical;
   (state0 = trying) & (state1 = trying) & (turn = turn0):  critical;
   (state0 = critical) : {critical,noncritical};
   1: state0;
esac;
```



```
pr0: process prc(s0, s1, turn, 0);

pr1: process prc(s1, s0, turn, 1);
```



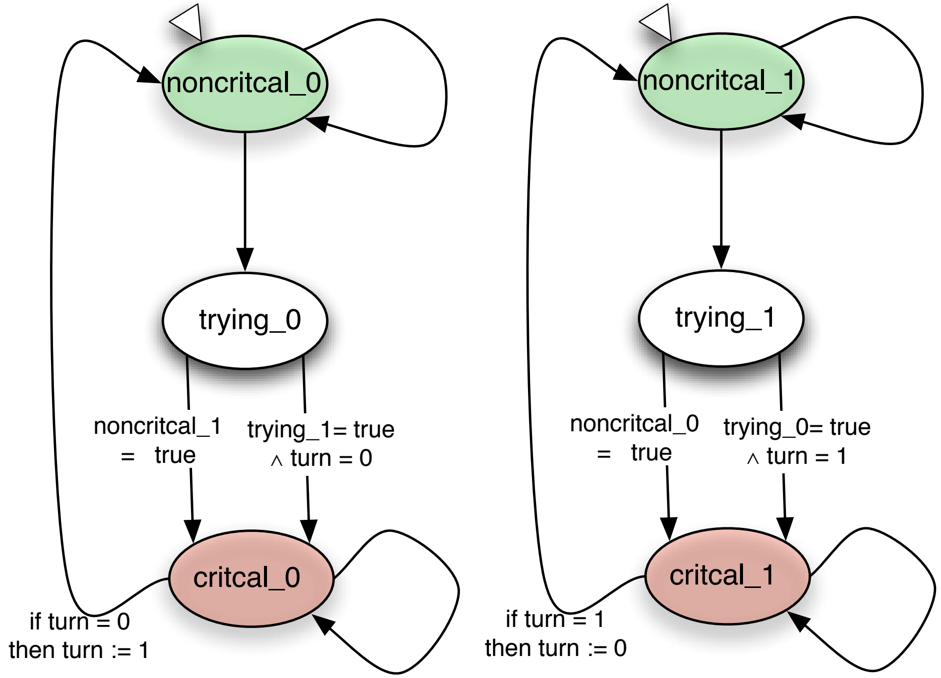

```
MODULE prc(state0, state1, turn, turn0)
```




```
next(turn) :=
case
   turn = turn0 & state0 = critical: !turn;
   1: turn;
esac;

FAIRNESS
running
```

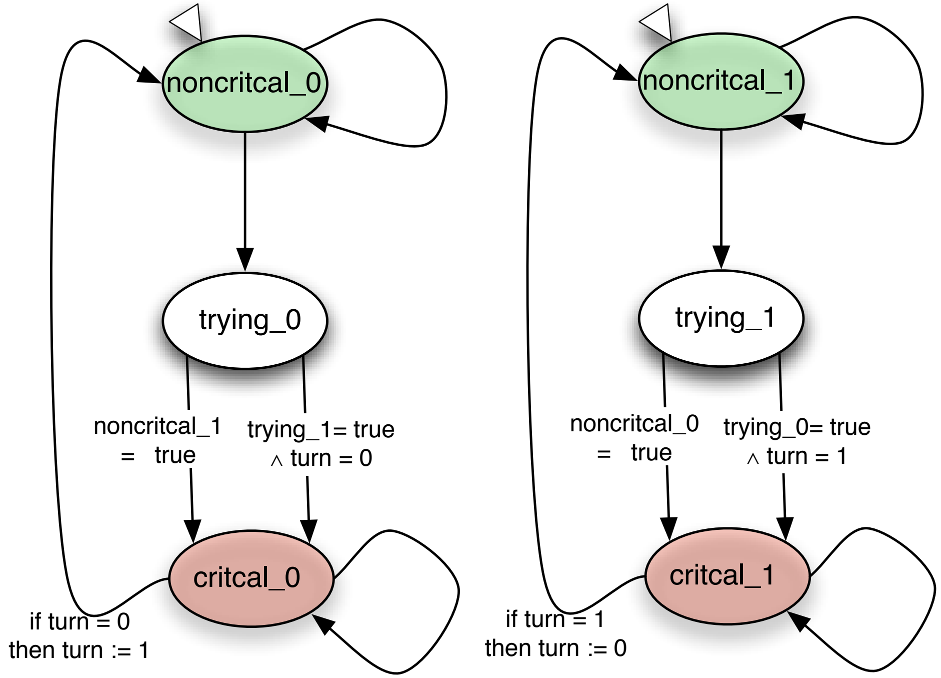
```
pr0: process prc(s0, s1, turn, 0);

pr1: process prc(s1, s0, turn, 1);
```

```
FAIRNESS !(s0 = critical)

FAIRNESS  !(s1 = critical)
```




```
SPEC

AG !((s0 = critical) & (s1 = critical))
```

```
pr0: process prc(s0, s1, turn, 0);

pr1: process prc(s1, s0, turn, 1);
```




```
FAIRNESS !(s0 = critical)

FAIRNESS  !(s1 = critical)

SPEC

AG((s0 = trying) -> AF (s0 = critical))

SPEC

AG((s1 = trying) -> AF (s1 = critical))
```

```
-- specification AG (!(s0 = critical & s1 = critical)) is true
```