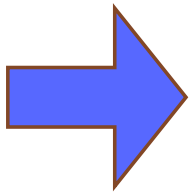
Kapitel 2: Algorithmen für CTL und LTL

Für eine gegebene Kripke-Struktur $M = (S, R, L)$ 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 $L(s)$ für alle $s \in S$ schrittweise zu $label(s)$, die Menge der Teilformeln von f, die in s wahr sind.

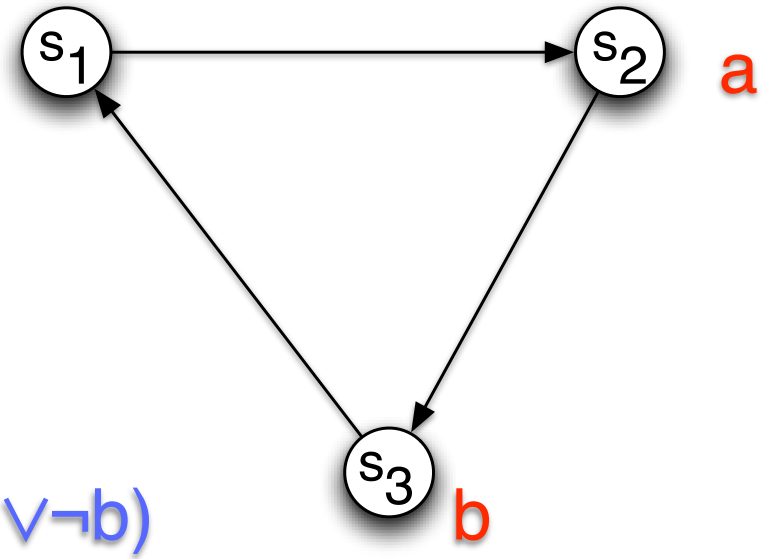
in Schritt i: alle Teilformeln mit $i-1$ geschachtelten CTL-Operatoren sind behandelt.

also: Rekursion über Schachtelungstiefe mit 6 Fällen:

Welche Basisformeln von CTL wählen?

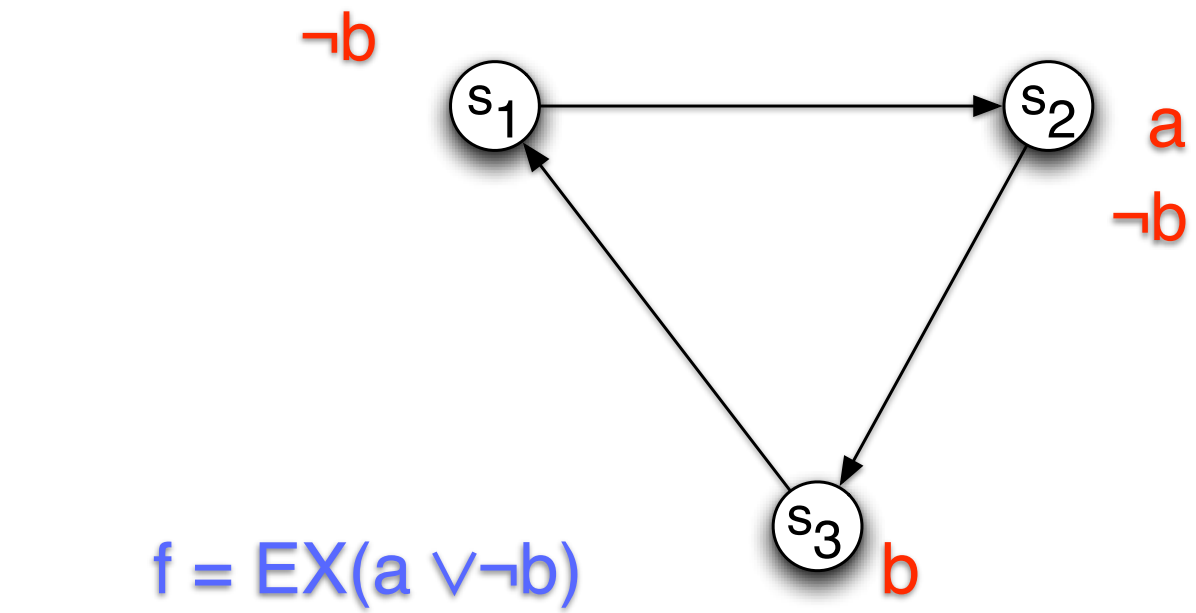
Clarke	Bérard et al	Huth & Ryan	Gopalakrishnan
EX	EX	EX	EX, AX
EU	EU	EU	AU
EG	AU	AF	AF
		EG	

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



f = EX(a ∨¬b)

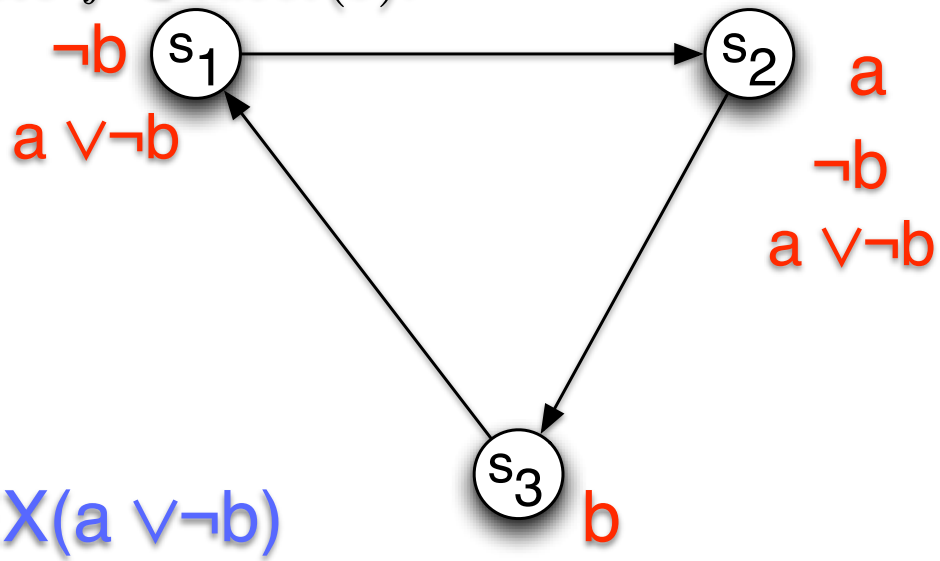
1. f atomar:
   für Zustände s mit $f \in L(s)$ setzte $f \in label(s)$.
2. $f = \neg f_1$:
   für Zustände s mit $f_1 \notin label(s)$ setzte $\neg f_1 \in label(s)$.



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

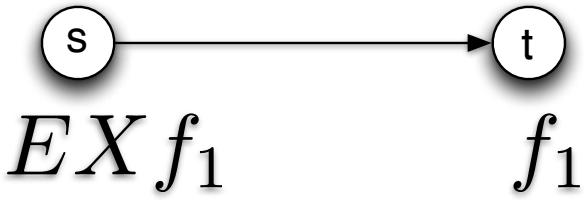
2. $f = \neg f_1$:
   für Zustände s mit $f_1 \notin label(s)$ setzte $\neg f_1 \in label(s)$.

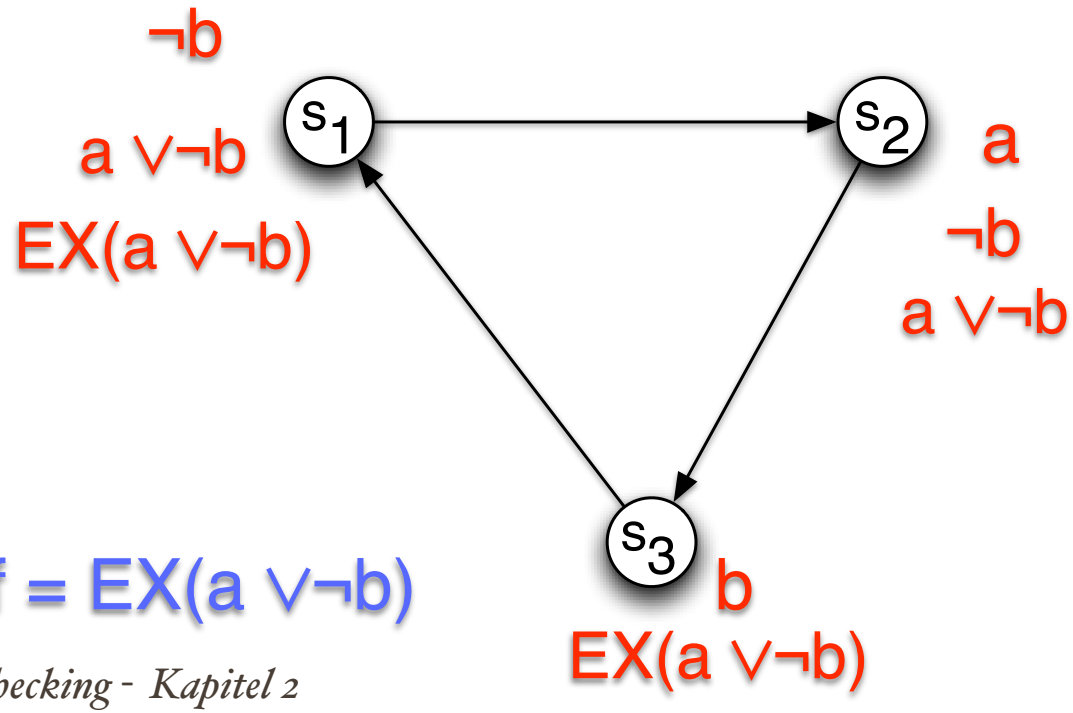
3. $f = f_1 \vee f_2$:
   für Zustände s mit $f_1 \in label(s)$ oder $f_2 \in label(s)$ setzte $f \in label(s)$.



f = EX(a ∨¬b)

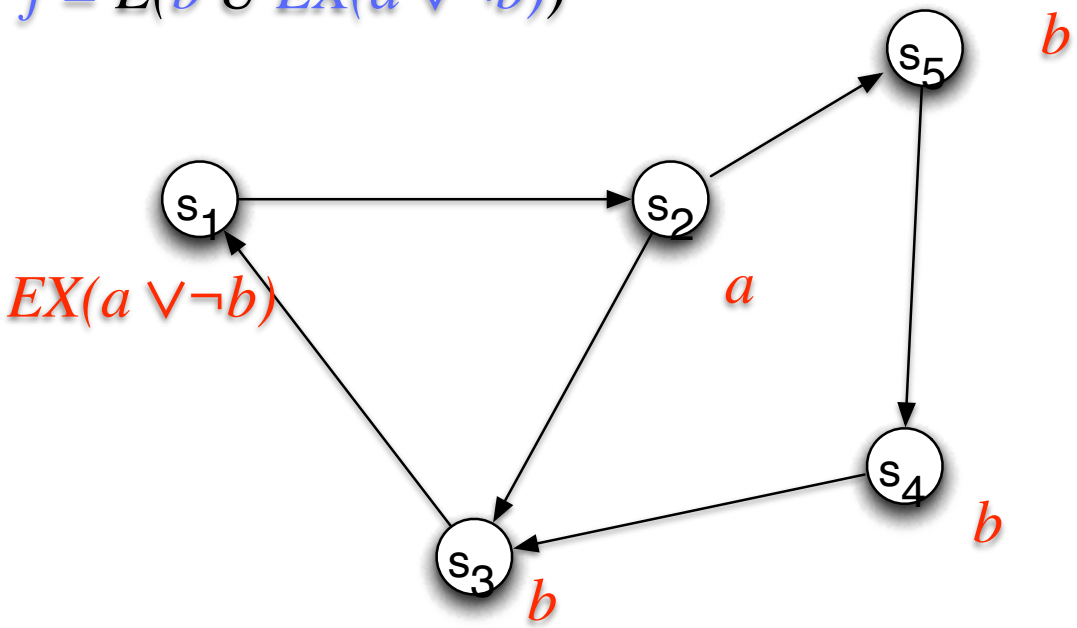
4. $f = EX\, f_1$:
für Zustände s mit $R(s,t)$ und $f_1 \in label(t)$ setzte $f \in label(s)$.





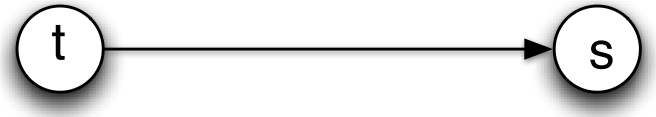
5. $f = E[f_1 U f_2]$:

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



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 $f \in label(t)$;



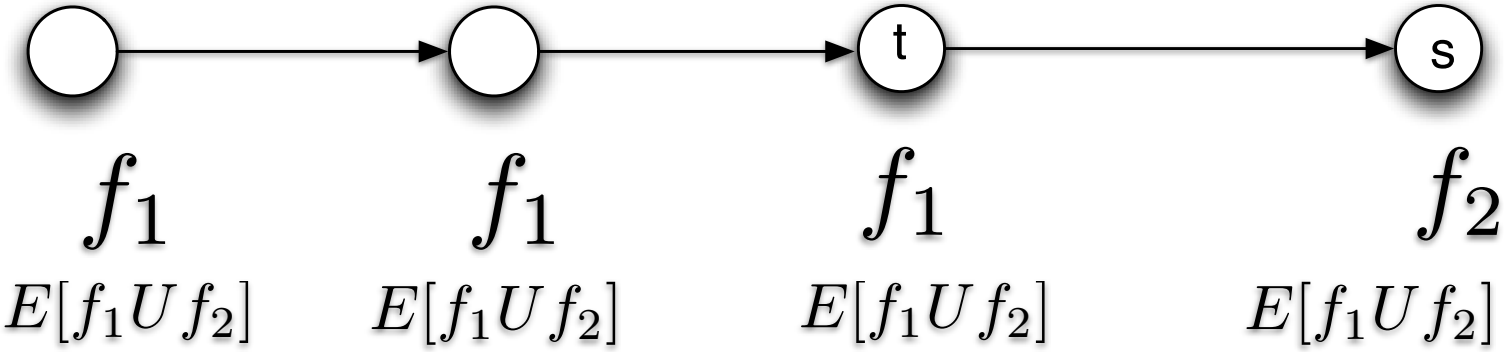
f_1 $\qquad$ f_2

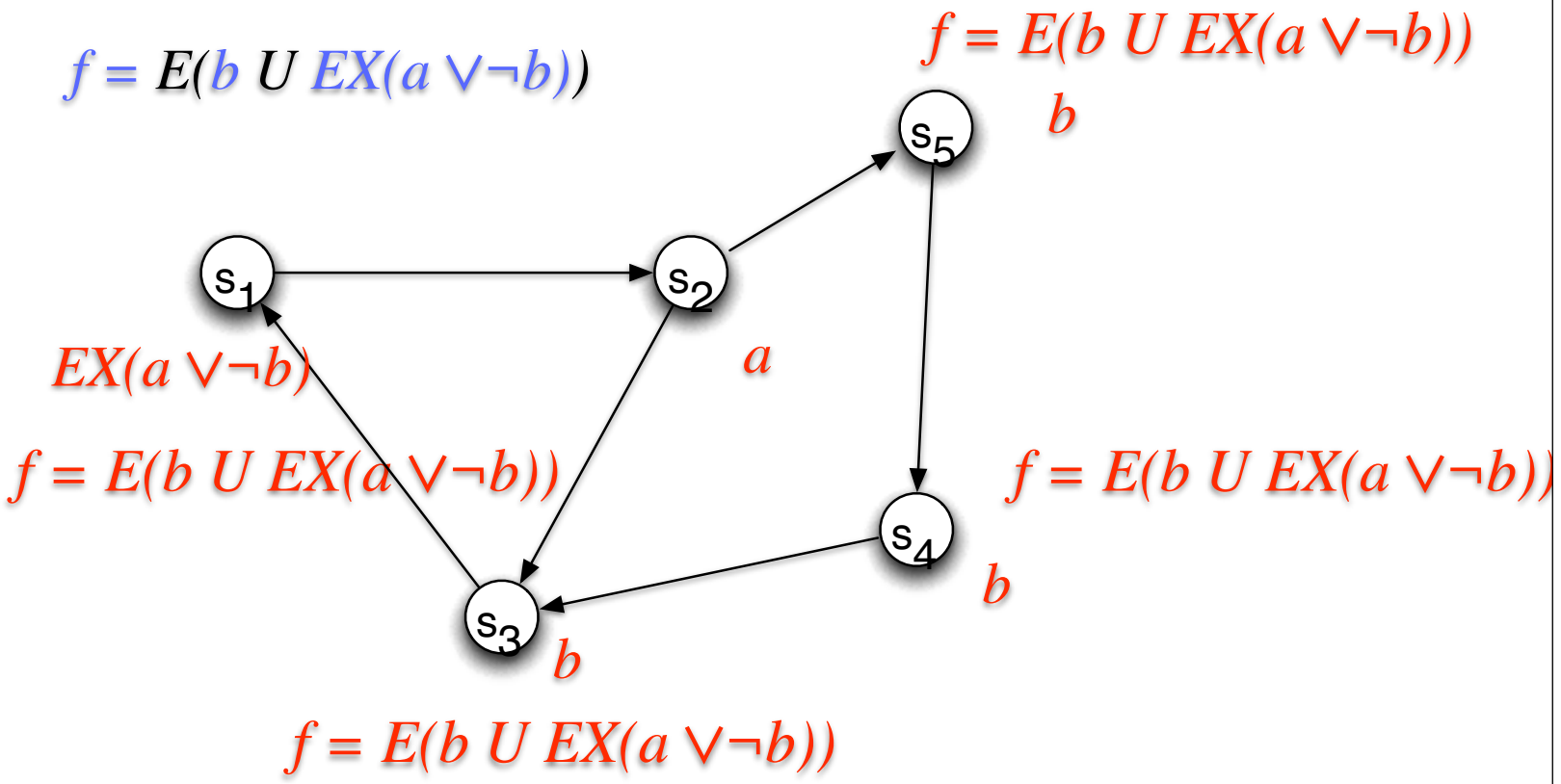
$E[f_1 U f_2]$ $\qquad$ $E[f_1 U f_2]$

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 ∨¬b))
f = E(b U EX(a ∨¬b))
b
s5
s1
s2
a
EX(a ∨¬b)
f = E(b U EX(a ∨¬b))
f = E(b U EX(a ∨¬b))
s4
b
s3
b
f = E(b U EX(a ∨¬b))

Algorithmus 5.5 Auszeichnen mit $E(f_1 U f_2)$

```
PROCEDURE CheckEU(f1, f2)
    T := {s | f2 ∈ label(s)};
    FOR ALL s ∈ T DO label(s) := label(s) ∪ {E[f1 U f2]};
    WHILE T ≠ ∅ DO
        CHOOSE s ∈ T;
        T := T\{s};
        FOR ALL t SUCH THAT R(t, s) DO
            IF E[f1 U f2] ∉ label(t) AND f1 ∈ label(t) THEN
                label(t) := label(t) ∪ {E[f1 U f2]};
                T := T ∪ {t};
            END IF ;
        END FOR ALL ;
    END WHILE ;
END PROCEDURE ;
```

EG f ??

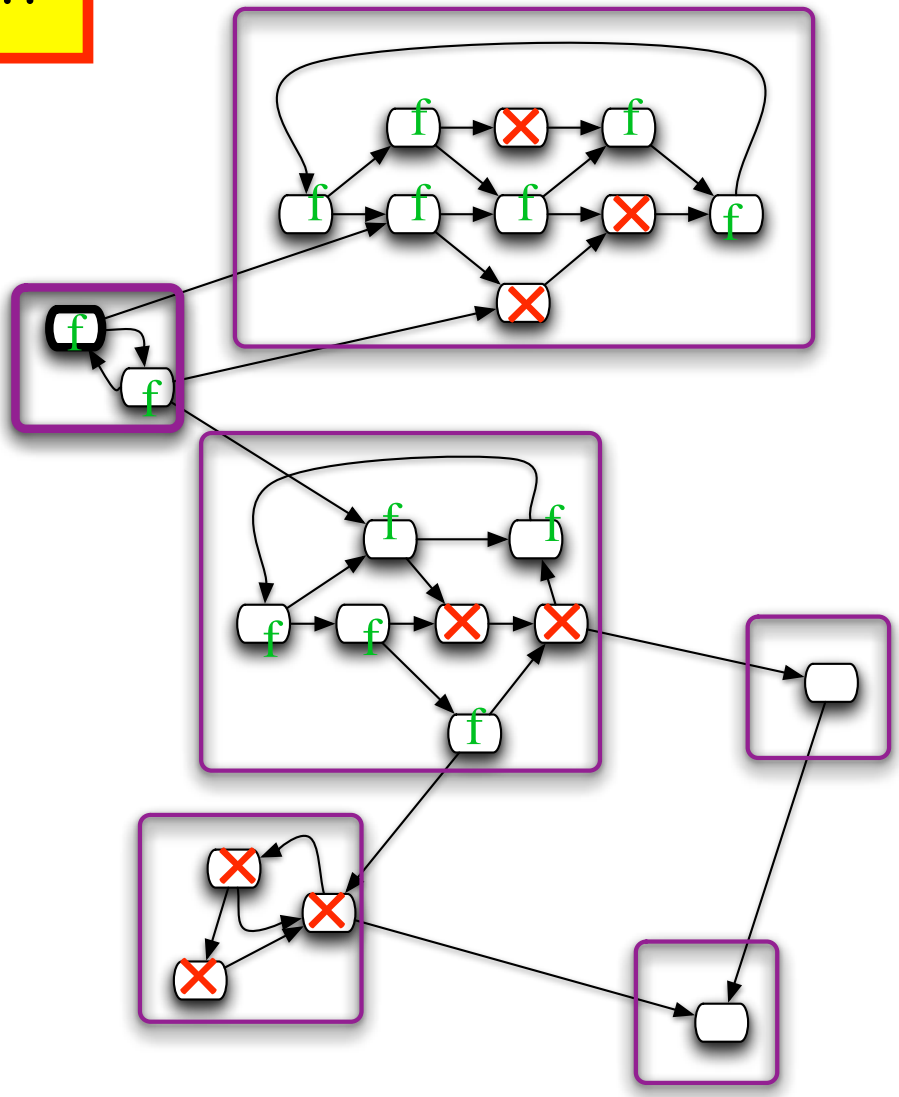
Definition 5.37 *Sei* $G = (K, R)$ *ein gerichteter Graph, d.h.:* $R \subseteq K \times K$*:*

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.:* $\neg\exists k \in K \backslash A \,.\, \forall a \in A : kR^*a \wedge aR^*k$.

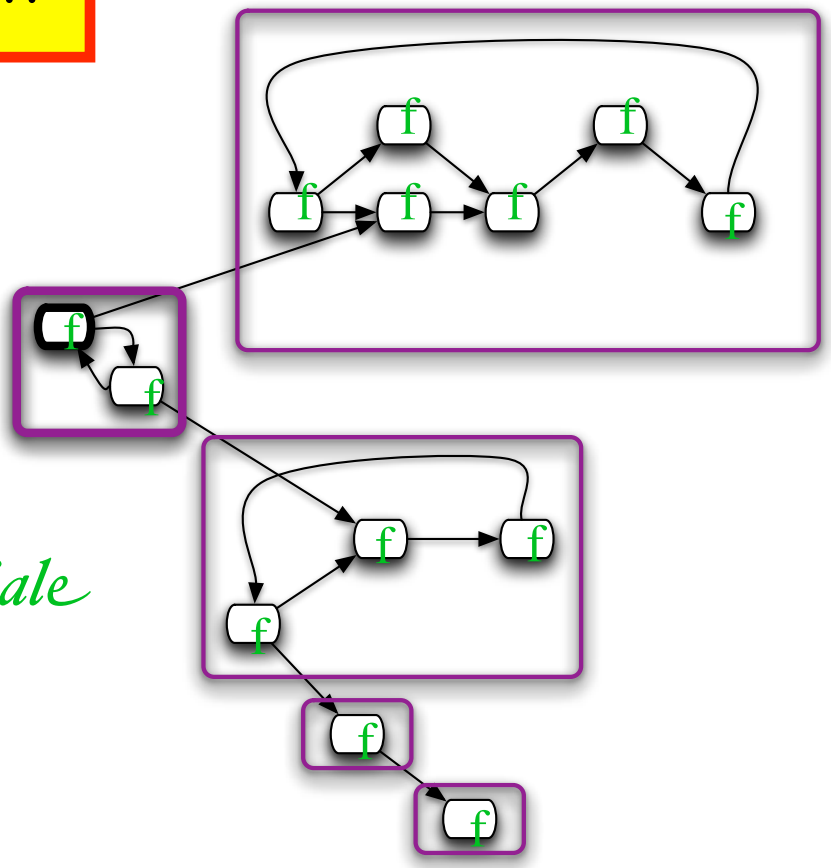
c) Sie heißt nichttriviale Zusammenhangskomponente, *falls:* $|A| > 1$ *oder* $\exists a \in A \,.\, aR^+a$.

EG f ??



SZK

EG f ??



nichttriviale SZK

triviale SZK

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 5.38 $M, s \models EG f_1$ *gdw.*

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

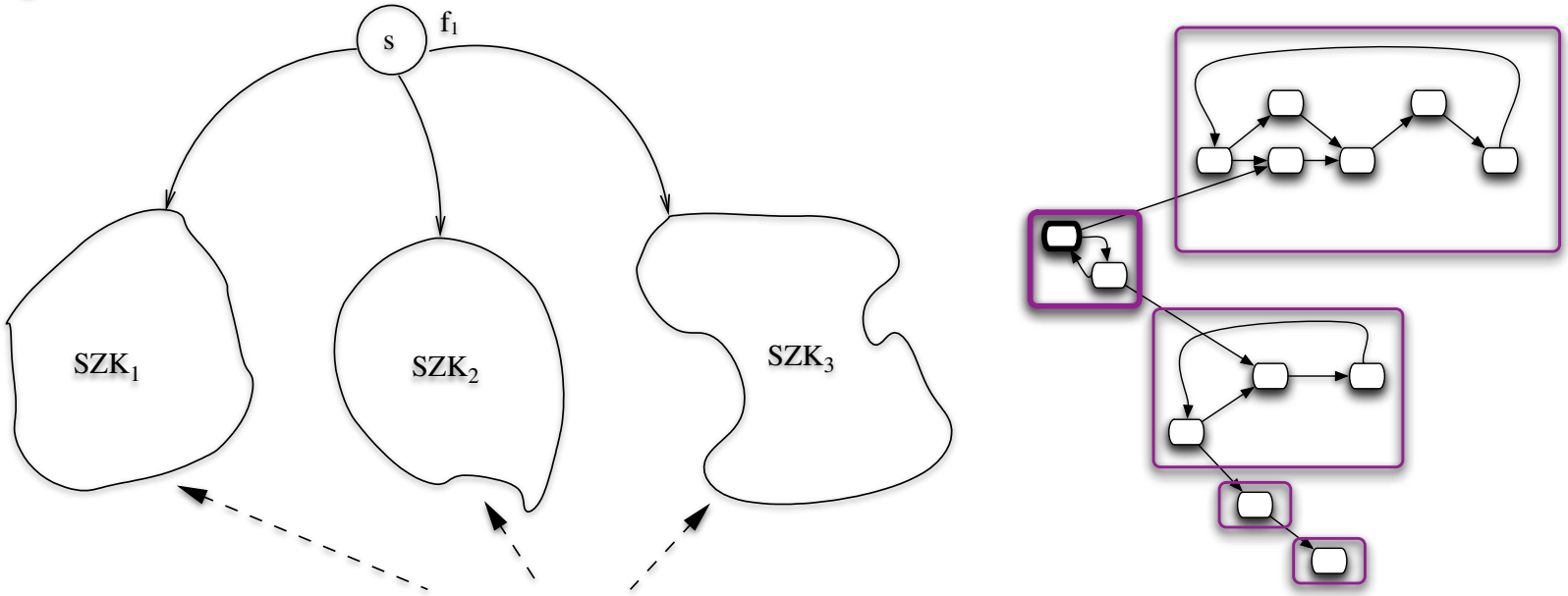


Abbildung 5.17: Strenge Zusammenhangskomponenten mit f_1

Daraus Algorithmus zur Entscheidung von EGf_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 5.6 Auszeichnen mit EGf_1

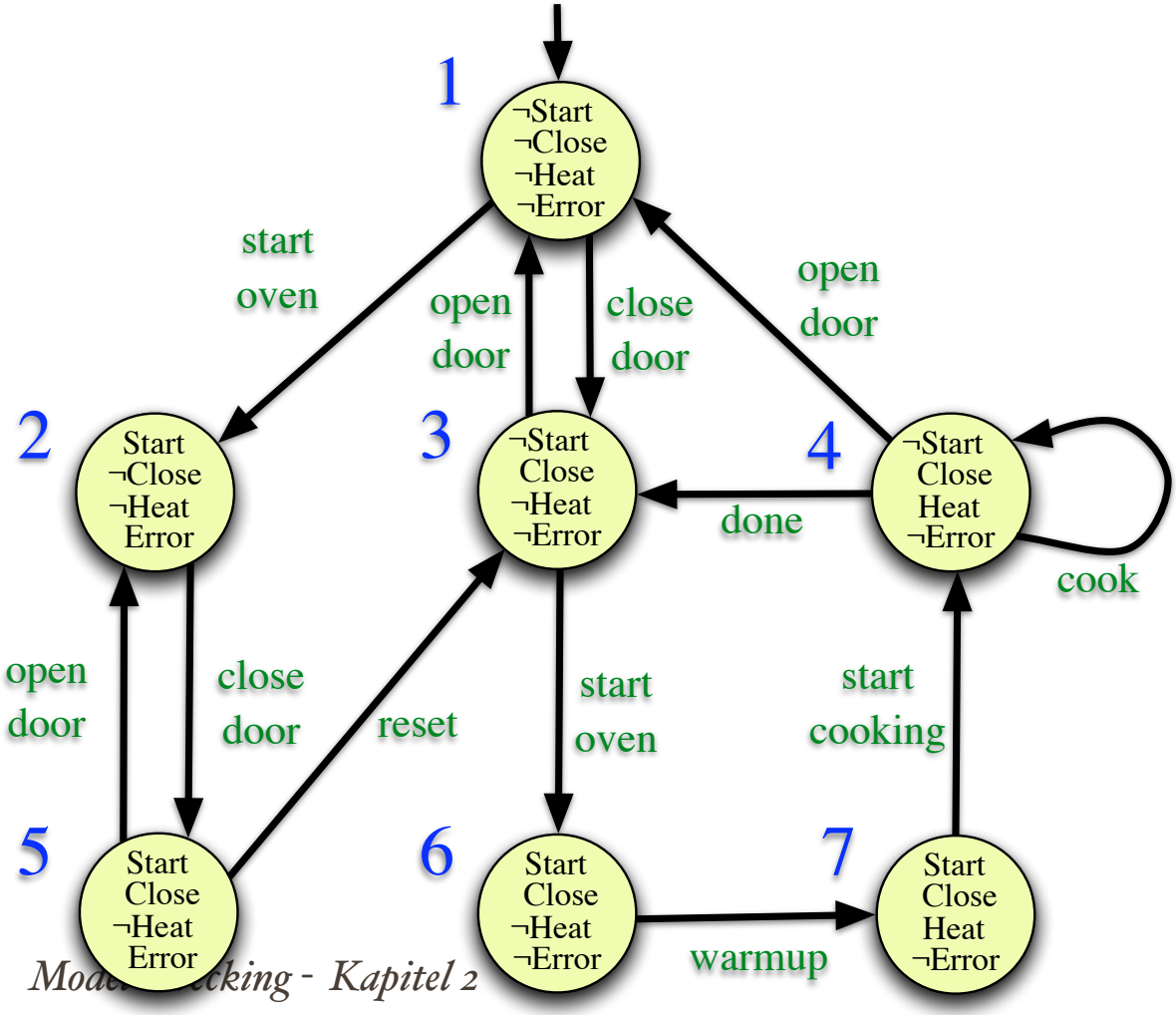
```
PROCEDURE CheckEGf1
    S' := {s | f1 ∈ label(s)};
    SCC := {C | C a nontrivial SCC of S'};
    T := ⋃_{C∈SCC} {s | s ∈ C};
    FOR ALL s ∈ T DO label(s) := label(s) ∪ {EGf1};
    WHILE T ≠ ∅ DO
        CHOOSE s ∈ T;
        T := T\{s};
        FOR ALL t SUCH THAT t ∈ S' AND R(t, s) DO
       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

Satz 5.39 *Es gibt einen Algorithmus, der für eine Struktur* $M = (S, R, L)$ *und eine Formel* $f \in CTL$ *in* $O(|f| \cdot (|S| + |R|))$ *Zeitkomplexität entscheidet, ob* f *für* M *gilt.*

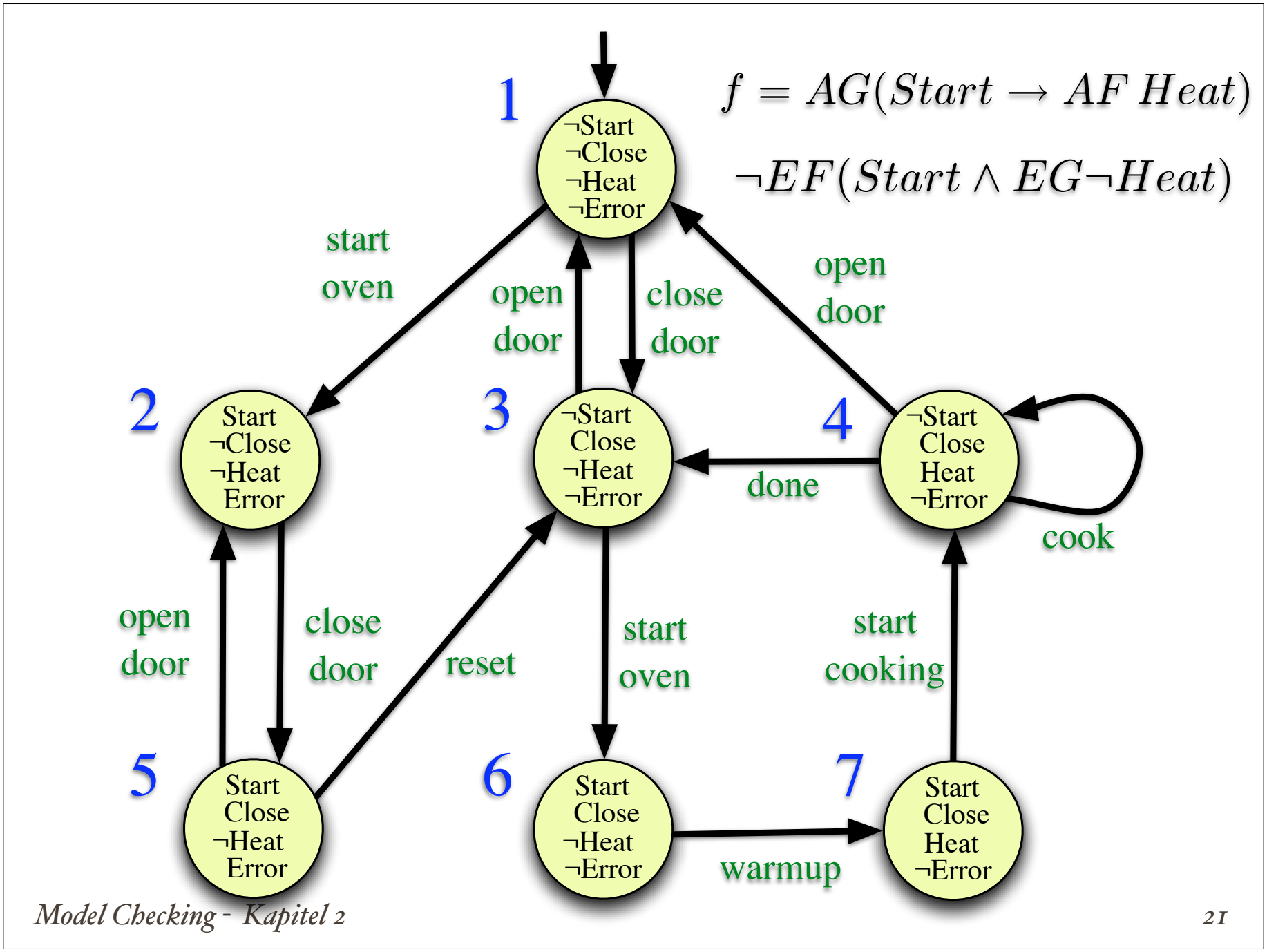
Beweis:
Wende obiges Verfahren auf die Atome von f an und fahre induktiv fort mit den Teilformeln von f, aufsteigend mit deren Schachtelung. □

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

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

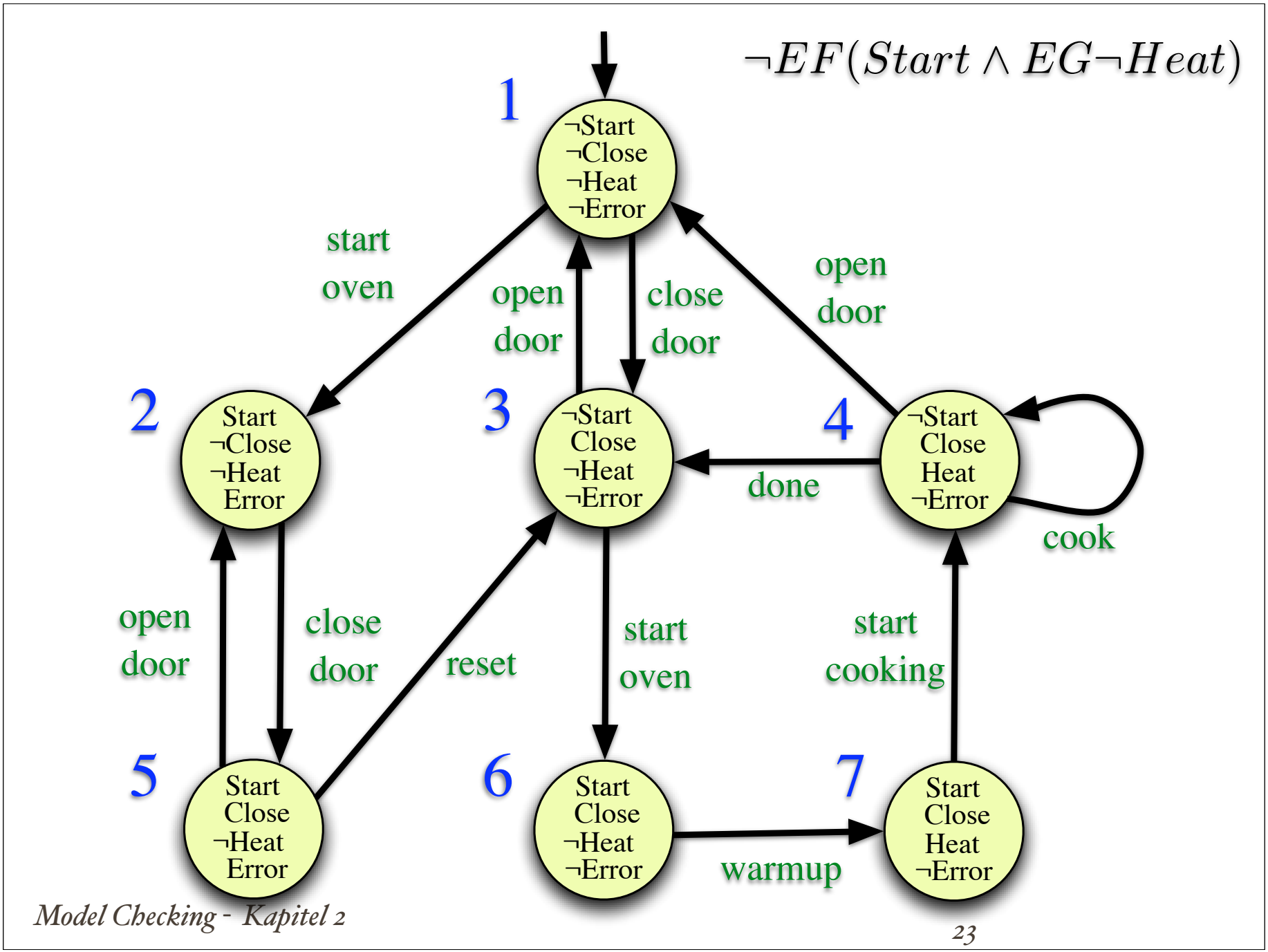


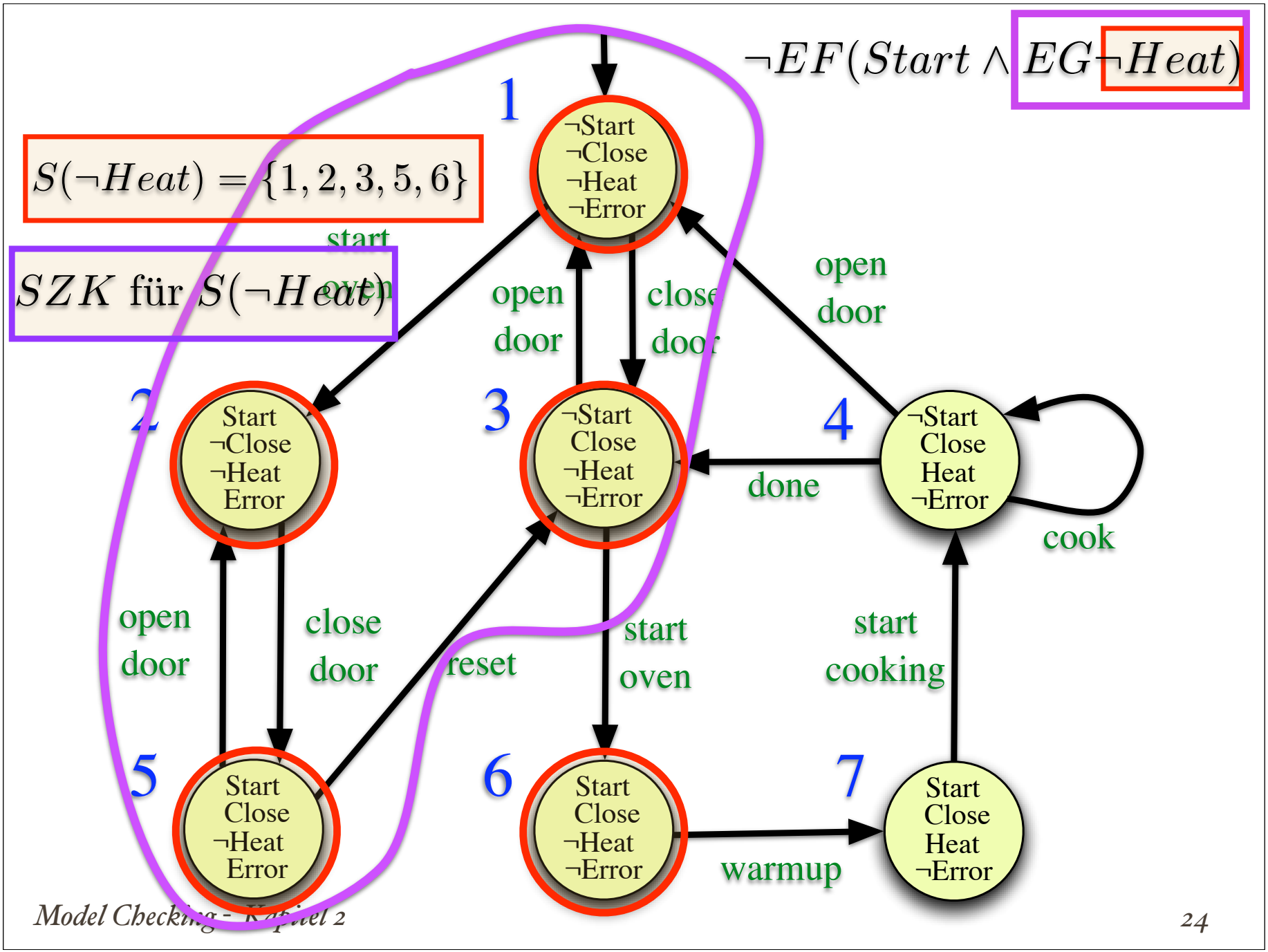
Beispiel: Mikrowellenofen

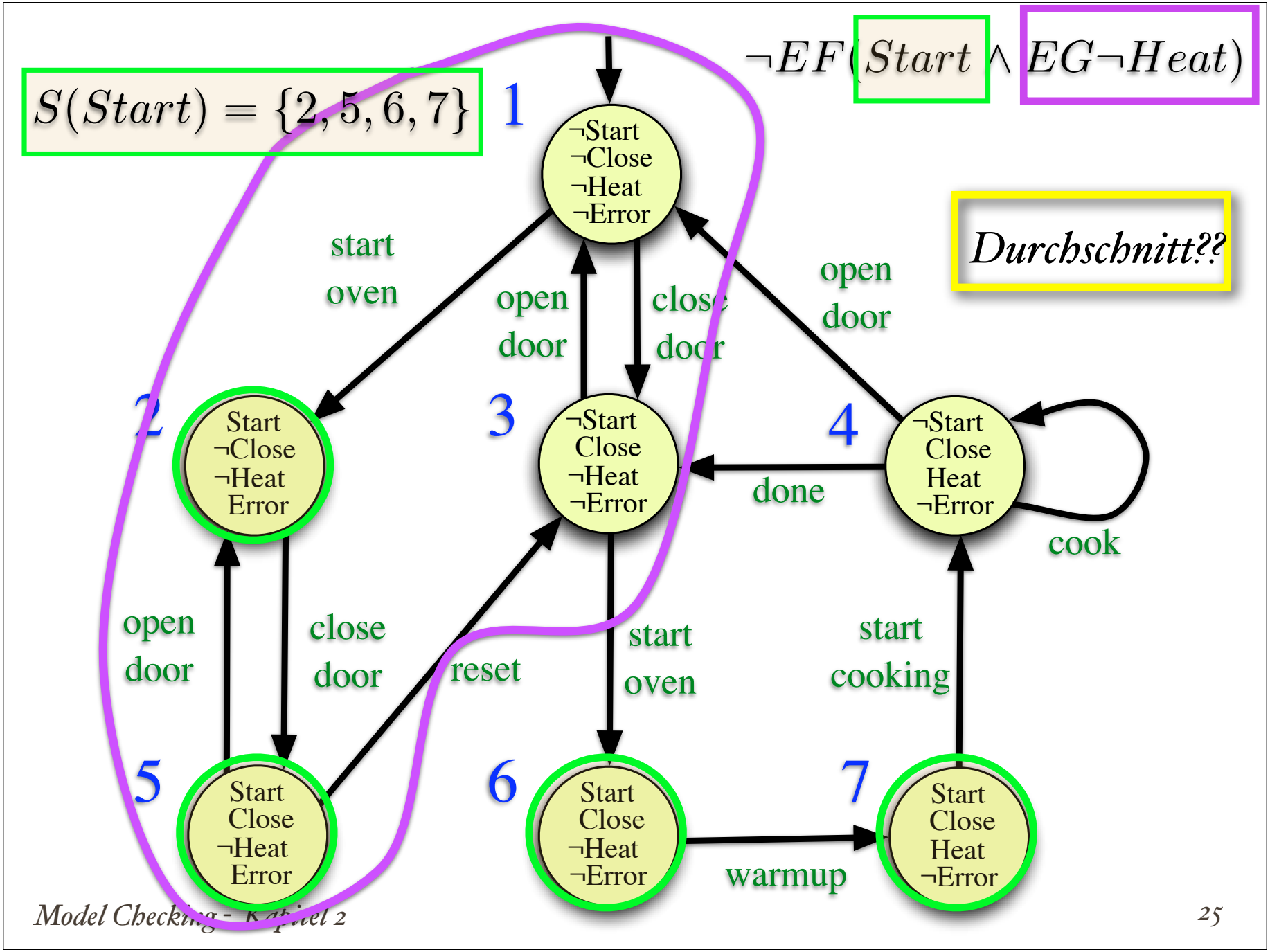

$f = AG(Start \rightarrow AF\ Heat)$
$\neg EF(Start \wedge EG\neg Heat)$
1
¬Start
¬Close
¬Heat
¬Error
start
oven
open
door
close
door
open
door
2
Start
¬Close
¬Heat
Error
3
¬Start
Close
¬Heat
¬Error
4
¬Start
Close
Heat
¬Error
done
cook
open
door
close
door
reset
start
oven
start
cooking
5
Start
Close
¬Heat
Error
6
Start
Close
¬Heat
¬Error
7
Start
Close
Heat
¬Error
warmup

Umschreibung von $f := AG(\underbrace{Start \rightarrow AF\, Heat}_{f_1})$

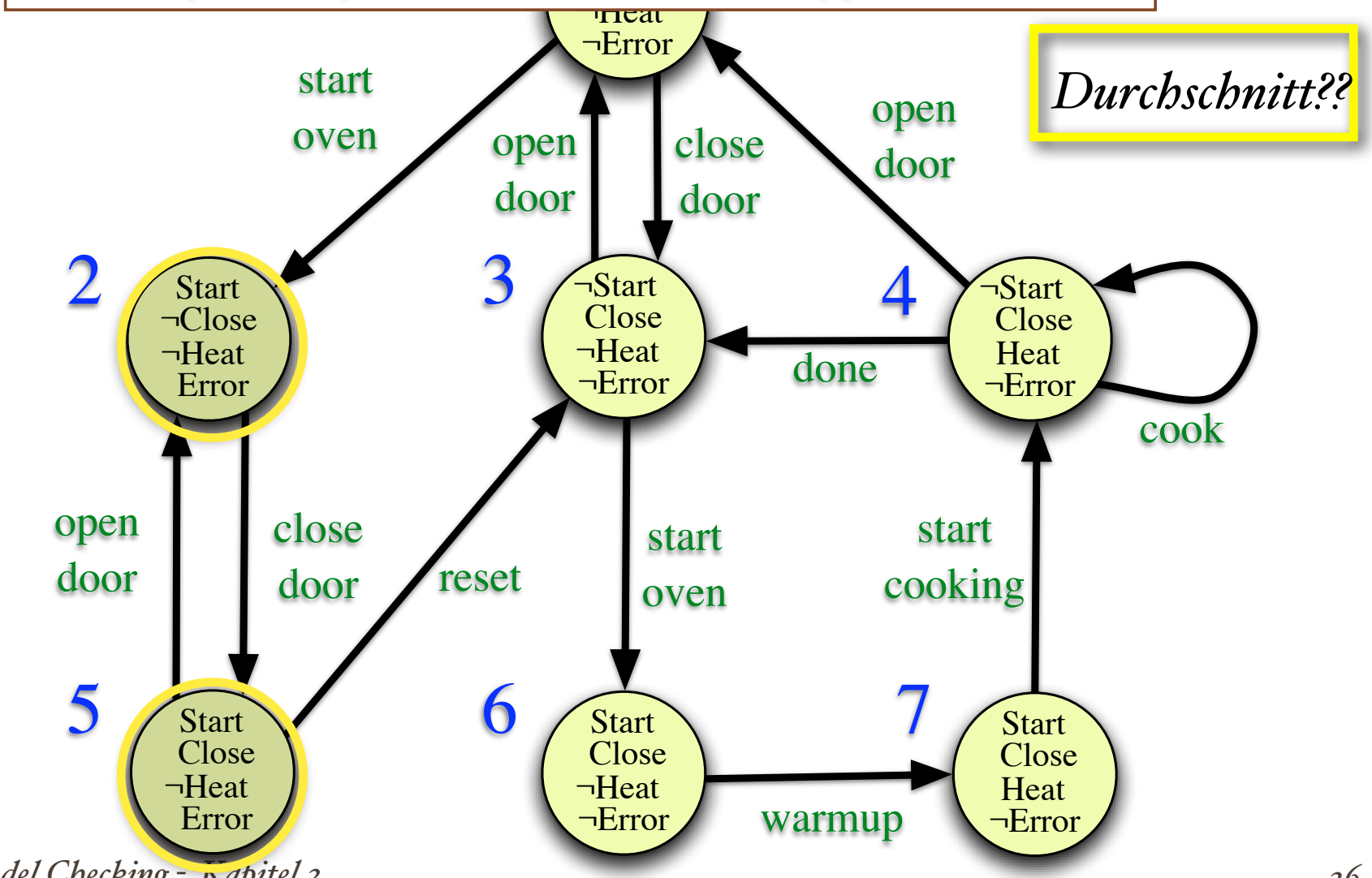
$$
\begin{aligned}
AG f_1 &= \neg EF(\neg f_1) \\
&= \neg EF(\neg(\neg Start \vee AF\, Heat)) \\
&= \neg EF(Start \wedge \neg AF\, Heat) && (AF f = \neg EG(\neg f)) \\
&= \neg EF(Start \wedge EG \neg Heat) && (EF f \equiv E[True\, U\, f])
\end{aligned}
$$


$\neg EF(Start \wedge EG\neg Heat)$
1
¬Start
¬Close
¬Heat
¬Error
start
oven
open
door
close
door
open
door
2
Start
¬Close
¬Heat
Error
3
¬Start
Close
¬Heat
¬Error
4
¬Start
Close
Heat
¬Error
done
cook
open
door
close
door
reset
start
oven
start
cooking
5
Start
Close
¬Heat
Error
6
Start
Close
¬Heat
¬Error
7
Start
Close
Heat
¬Error
warmup


$\neg EF(Start \wedge EG\neg Heat)$
$S(\neg Heat) = \{1, 2, 3, 5, 6\}$
SZK für $S(\neg Heat)$
1
¬Start
¬Close
¬Heat
¬Error
2
Start
¬Close
¬Heat
Error
3
¬Start
Close
¬Heat
¬Error
4
¬Start
Close
Heat
¬Error
5
Start
Close
¬Heat
Error
6
Start
Close
¬Heat
¬Error
7
Start
Close
Heat
¬Error
start oven
open door
close door
open door
open door
close door
done
cook
reset
start oven
start cooking
warmup


$S(Start) = \{2, 5, 6, 7\}$
$\neg EF(Start \wedge EG\neg Heat)$
Durchschnitt??
1
¬Start
¬Close
¬Heat
¬Error
start
oven
open
door
close
door
open
door
2
Start
¬Close
¬Heat
Error
3
¬Start
Close
¬Heat
¬Error
4
¬Start
Close
Heat
¬Error
done
cook
open
door
close
door
reset
start
oven
start
cooking
5
Start
Close
¬Heat
Error
6
Start
Close
¬Heat
¬Error
7
Start
Close
Heat
¬Error
warmup

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



Was wurde bewiesen?

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

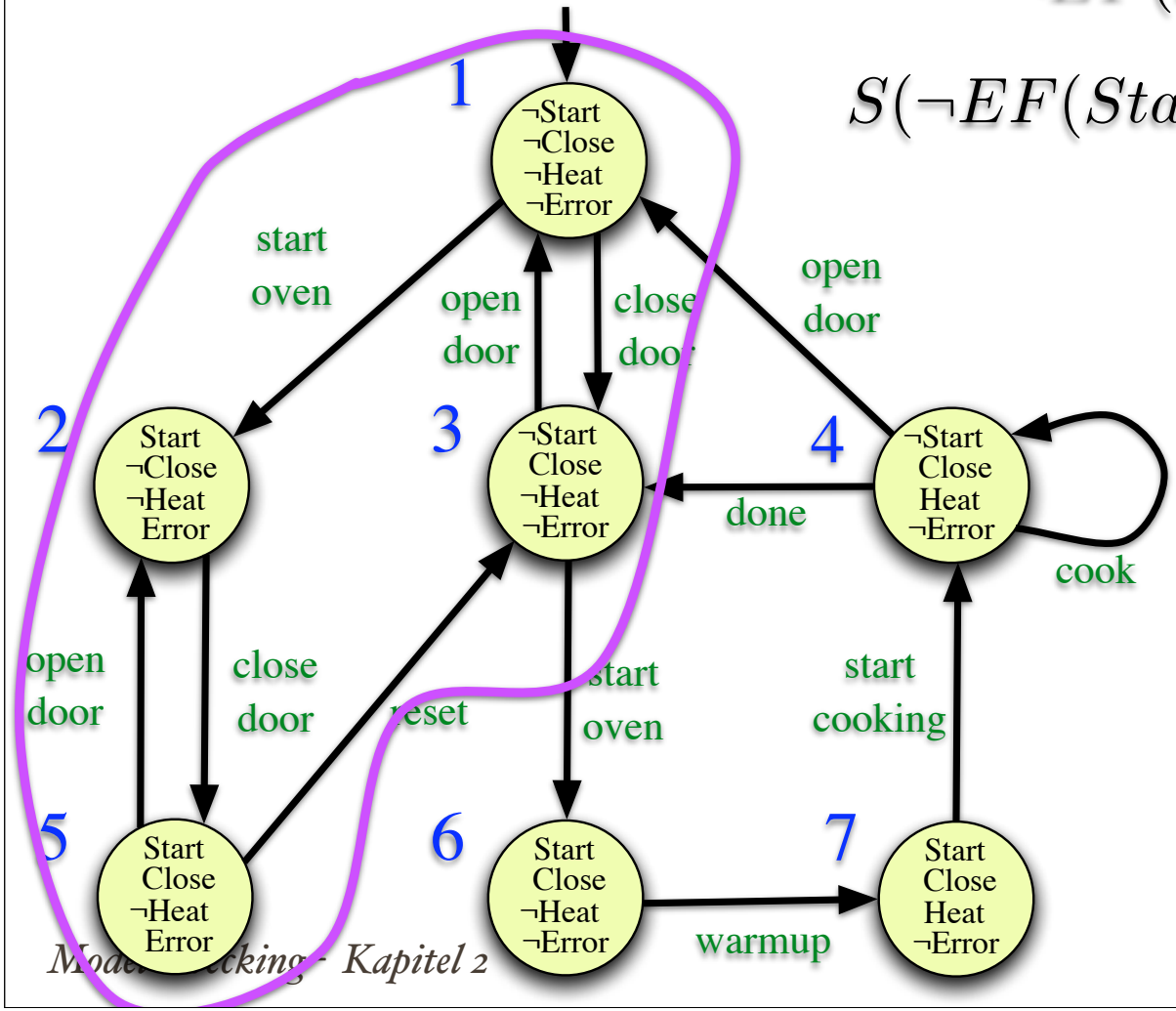
Es gilt immer: nach einem Zustand mit „Start" wird später ein Zustand mit „Heat" erreicht. *gilt nicht!*

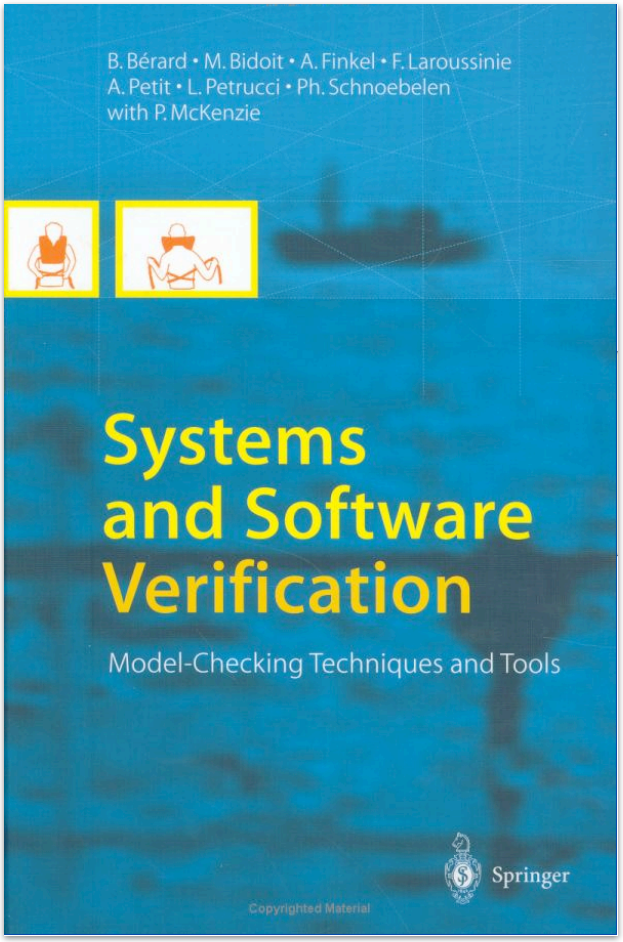
$$\neg EF(Start \wedge EG\neg Heat)$$

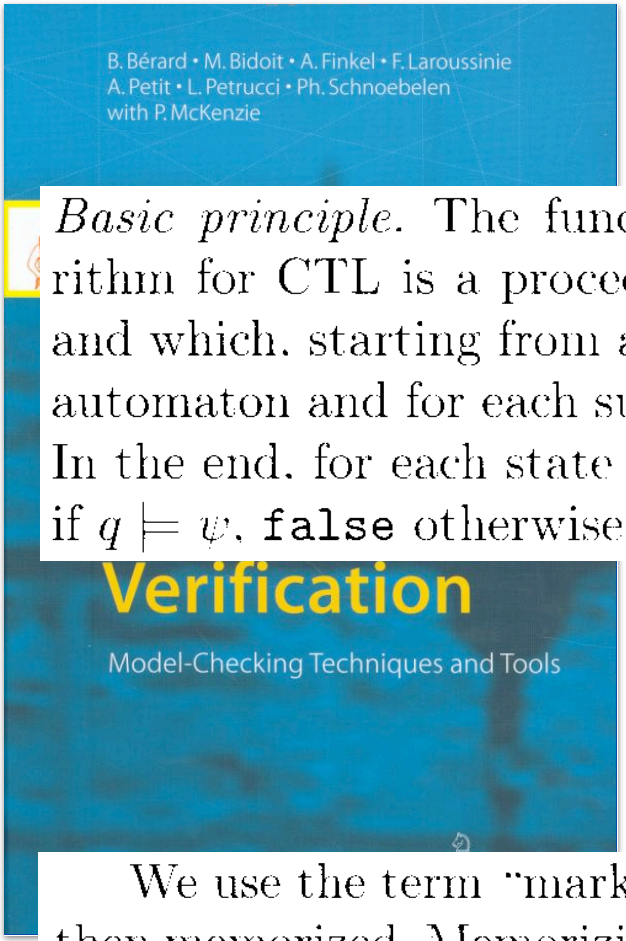
$$S(\neg EF(Start \wedge EG\neg Heat)) = \emptyset$$

Gegenbeispiel:

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



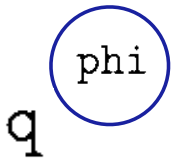

B. Bérard • M. Bidoit • A. Finkel • F. Laroussinie
A. Petit • L. Petrucci • Ph. Schnoebelen
with P. McKenzie
Systems and Software Verification
Model-Checking Techniques and Tools
Springer
Copyrighted Material



Basic principle. The fundamental component of the model checking algorithm for CTL is a procedure `marking` which operates on an automaton $\mathcal{A}$ and which, starting from a CTL formula ϕ, will mark, for each state q of the automaton and for each sub-formula ψ of ϕ, whether ψ is satisfied in state q. In the end, for each state and each sub-formula, `q.psi` holds the value `true` if $q \models \psi$, `false` otherwise.

We use the term "marking" to mean that the value of `q.psi` is computed then memorized. Memorizing is important because the marking of `q.phi` uses the values of `q'.psi` for sub-formulas `psi` of `phi` and for states `q'` reachable from `q`. When the marking for `phi` is completed, it is easy to say whether $\mathcal{A} \models \phi$ by looking up the value of `q0.phi` for the initial state q_0 of $\mathcal{A}$. Here is the crux of the algorithm:

```
procedure marking(phi)

  case 1: phi = P
    for all q in Q, if P in l(q) then do q.phi := true,
                                    else do q.phi := false.

  case 2: phi = not psi
    do marking(psi);
    for all q in Q, do q.phi := not(q.psi).

  case 3: phi = psi1 /\ psi2
    do marking(psi1); marking(psi2);
    for all q in Q, do q.phi := and(q.psi1, q.psi2).

  case 4: phi = EX psi
    do marking(psi);
    for all q in Q, do q.phi := false;      /* initialisation */
    for all (q,q') in T, if q'.psi = true then do q.phi := true.

  case 5: phi = E psi1 U psi2
    do marking(psi1); marking(psi2);
    for all q in Q,
      q.phi := false; q.seenbefore := false;/* initialisation */
    L := {};                        /* L: states to be processed */
    for all q in Q, if q.psi2 = true then do L := L + { q };
    while L nonempty {
      draw q from L;                              /* must mark q */
      L := L - { q };
      q.phi := true;
      for all (q',q) in T {       /* q' is a predecessor of q */
        if q'.seenbefore = false then do {
          q'.seenbefore := true;
          if q'.psi1 = true then do L := L + { q' };
        }
      }
    }

  case 6: phi = A psi1 U psi2
    /* See further */
```



```
case 6: phi = A psi1 U psi2
  do marking(psi1); marking(psi2);
  L := {}                                  /* L: states to be processed */
  for all q in Q,
    q.nb := degree(q); q.phi := false;          /* initialisation */
  for all q in Q, if q.psi2 = true then do L := L + { q };
  while L nonempty {
    draw q from L;                                  /* must mark q */
    L := L - { q };
    q.phi := true;
    for all (q',q) in T {              /* q' is a predecessor of q */
      q'.nb := q'.nb - 1;                                /* decrement */
      if (q'.nb = 0) and (q'.psi1 = true) and (q'.phi = false)
        then do L := L + { q' };
    }
  }
```

The marking algorithm for ϕ of the form $A\psi_1 U\psi_2$ (case 6) is a bit more complicated (see below). It relies on the observation that a state q satisfies $A\psi_1 U\psi_2$ if and only if either (a) q satisfies ψ_2. or (b.1) q satisfies ψ_1. (b.2) q has at least one successor state. and (b.3) all its successors satisfy $A\psi_1 U\psi_2$.

The marking algorithm for ϕ of the form $\mathsf{A}\psi_1\mathsf{U}\psi_2$ (case 6) is a bit more complicated (see below). It relies on the observation that a state q satisfies $\mathsf{A}\psi_1\mathsf{U}\psi_2$ if and only if either (a) q satisfies ψ_2. or (b.1) q satisfies ψ_1. (b.2) q has at least one successor state. and (b.3) all its successors satisfy $\mathsf{A}\psi_1\mathsf{U}\psi_2$.

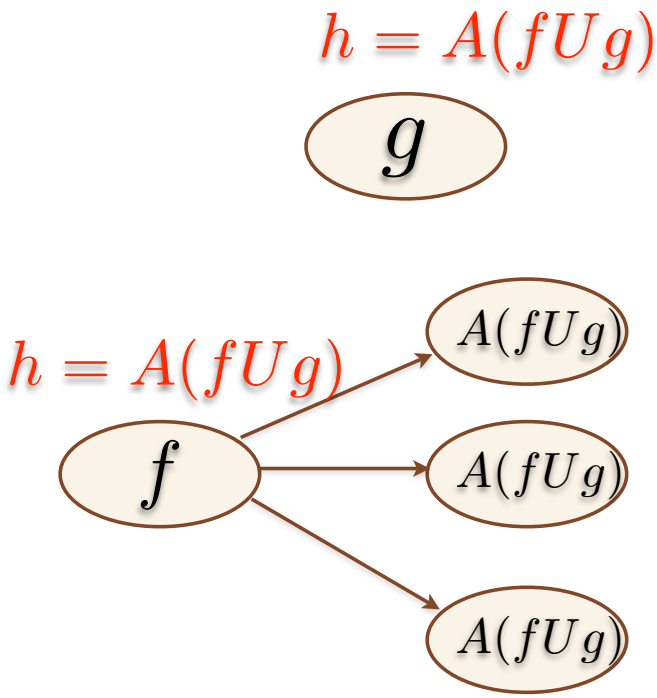
$$q \models A(fUg) :\Leftrightarrow$$

a) $q \models g \ \vee$

b_1) $q \models f \ \wedge$

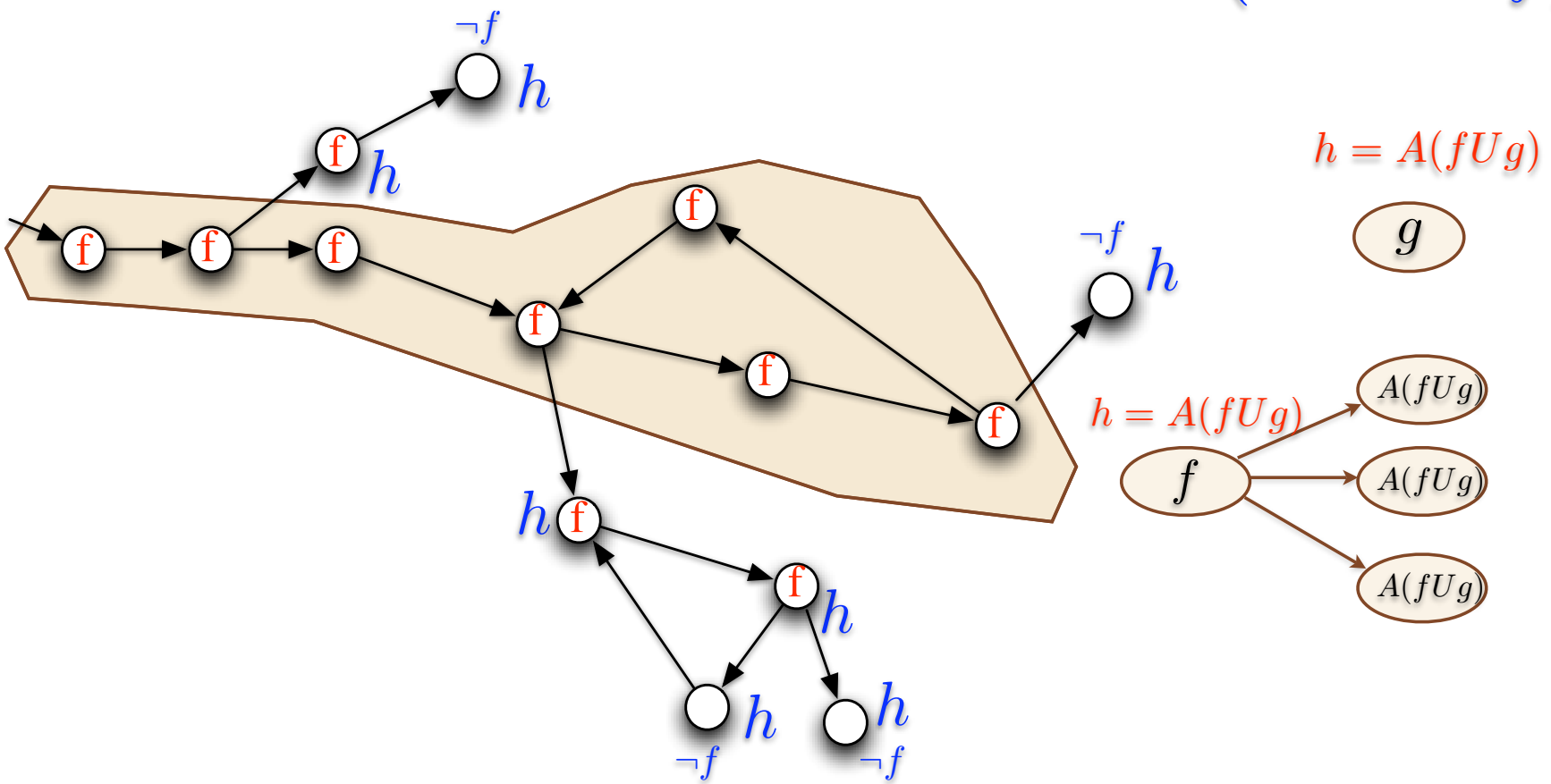
b_2) $\exists q' : (q, q') \in R \ \wedge$

b_3) $\forall q' : (q, q') \in R \Rightarrow q' \models A(fUg)$



$$EGf \Leftrightarrow EG\neg\neg f$$

$$h = A(true\ U \neg f)$$



Huth & Ryan

EX

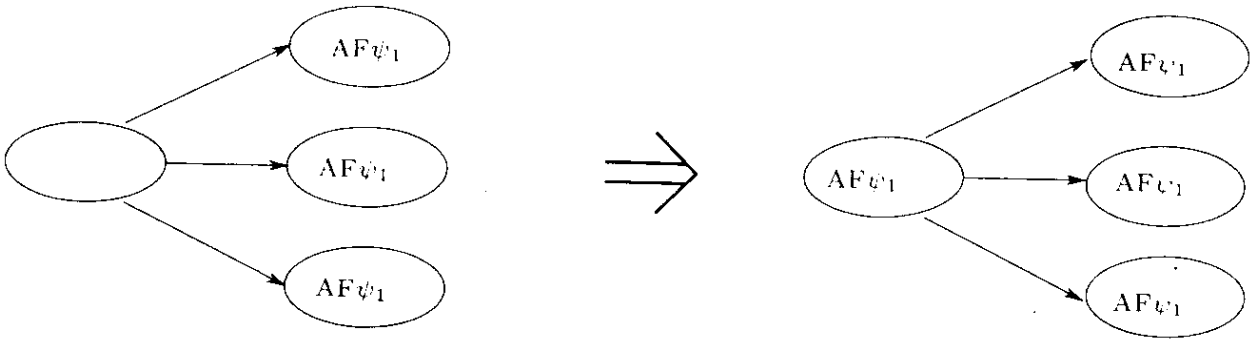
EU

AF

EG

- $\mathrm{AF}\,\psi_1$:
  - If any state s is labelled with ψ_1, label it with $\mathrm{AF}\,\psi_1$.
  - Repeat: label any state with $\mathrm{AF}\,\psi_1$ if all successor states are labelled with $\mathrm{AF}\,\psi_1$, until there is no change. This step is illustrated in Figure 3.24.

Repeat...



... until no change.

Huth & Ryan

EX

EU

AF

EG

Handling EG directly Instead of using a minimal adequate set of connectives, it would have been possible to write similar routines for the other connectives. Indeed, this would probably be more efficient. The connectives AG and EG require a slightly different approach from that for the others, however. Here is the algorithm to deal with EG ψ_1 *directly*:

- EG ψ_1:
  - Label *all* the states with EG ψ_1.
  - If any state s is *not* labelled with ψ_1, *delete* the label EG ψ_1.
  - Repeat: *delete* the label EG ψ_1 from any state if *none* of its successors is labelled with EG ψ_1: until there is no change.

Here, we label all the states with the subformula EG ψ_1 and then whittle down this labelled set, instead of building it up from nothing as we did in the case for EU. Actually, there is no real difference between this procedure for EG ψ and what you would do if you translated it into $\neg$AF $\neg\psi$ as far as the final result is concerned.

- EG ψ_1:
  - Label *all* the states with EG f
  - If any state s is *not* labelled with f , *delete* the label EG f .
  - Repeat: *delete* the label EG f from any state if *none* of its successors is labelled with EG f : until there is no change.

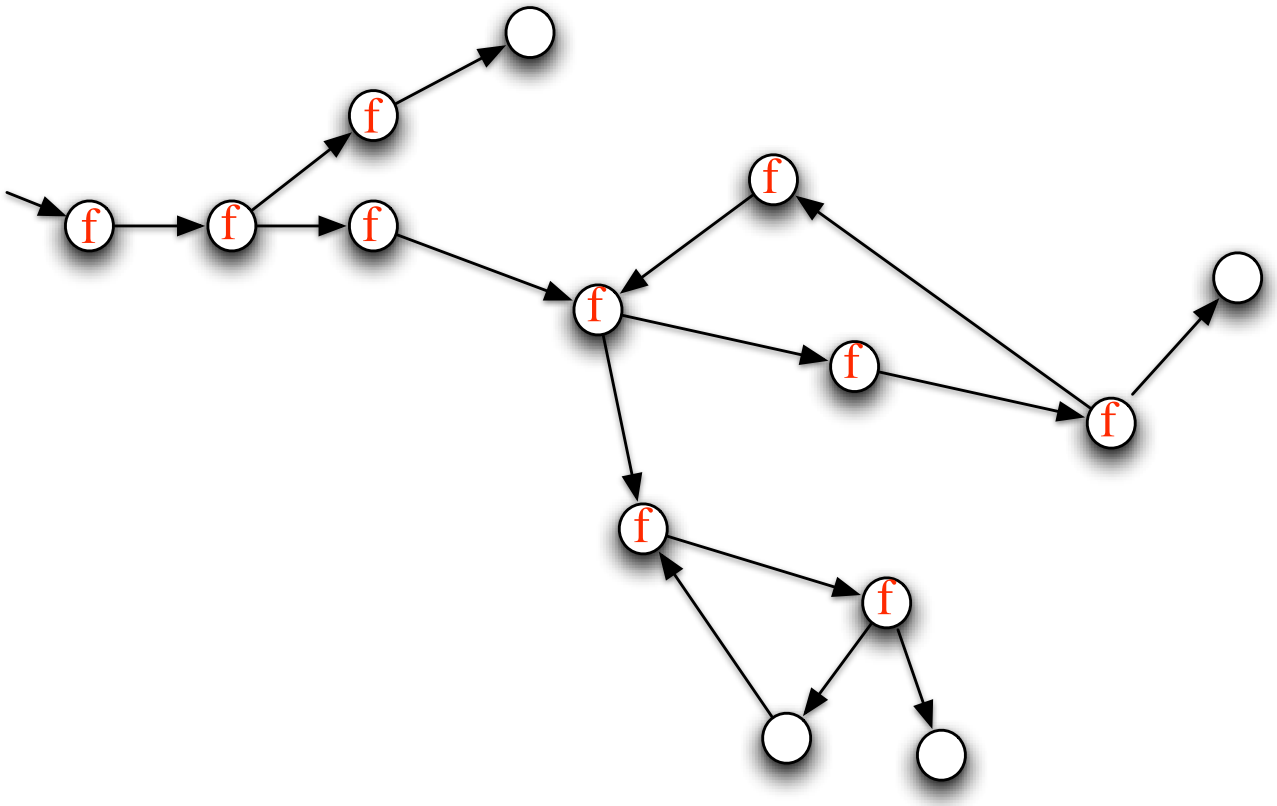
Huth & Ryan

EX

EU

AF

EG



- EG ψ_1:
  - Label *all* the states with EG f
  - If any state s is *not* labelled with f , *delete* the label EG f .
  - Repeat: *delete* the label EG f from any state if *none* of its successors is labelled with EG f : until there is no change.

Huth & Ryan

EX

EU

AF

EG