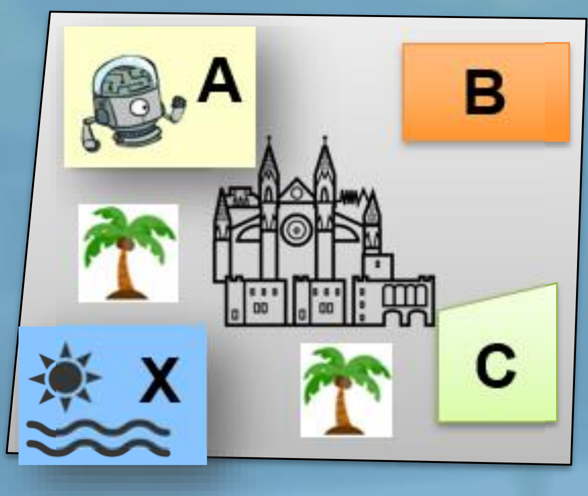
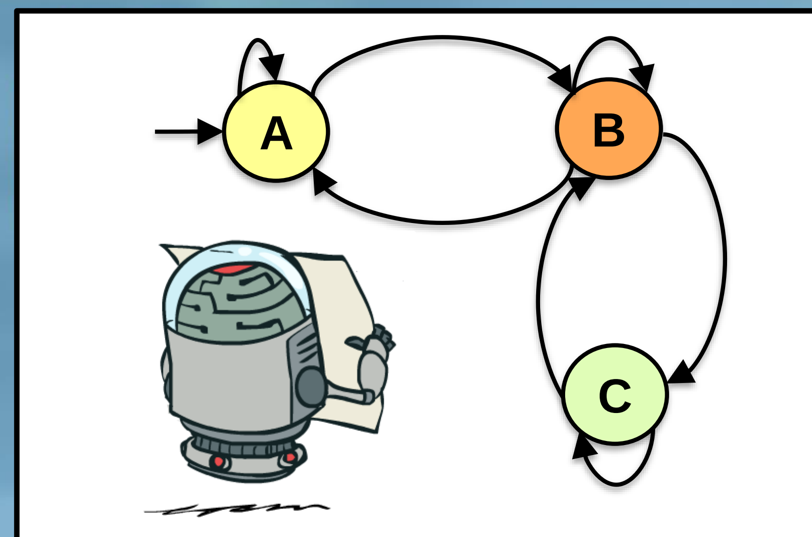
**Graz University of Technology**
Institute for Applied Information Processing and Communications

# CTL Model Checking

Bettina Könighofer



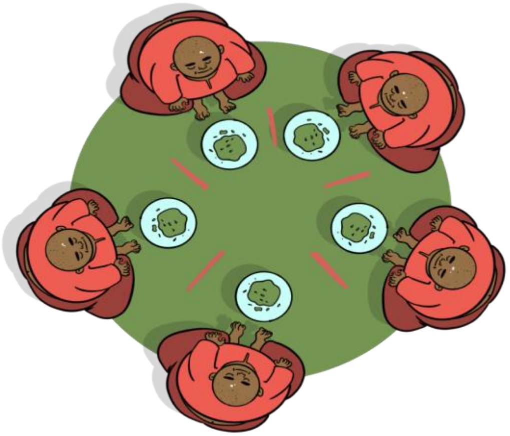

Model Checking SS21

May 5th 2021

# Homework Nr 6

## *The Dining-Philosophers Verification-Problem*

We consider a variant of the dining philosophers problem.
There are $n$ philosophers sitting at a round table.
There is one chopstick between each pair of adjacent philosophers.
Because each philosopher needs two chopsticks to eat, adjacent philosophers cannot eat simultaneously. We are interested in schedulers that use input variables $h_i$ signifying that philosopher $i$ is **hungry** and output variables $e_i$ signifying that philosopher $i$ is **eating**.



# Solutions Homework

## *The Dining-Philosophers Verification-Problem*

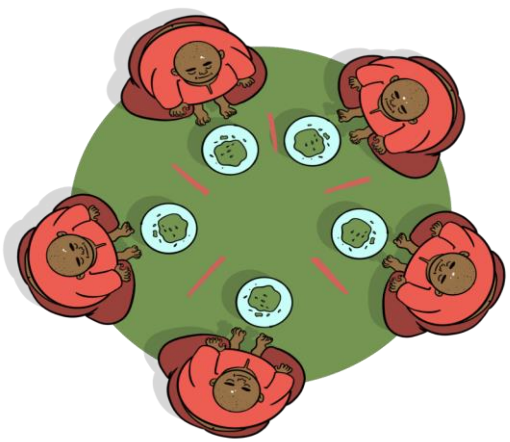
***[4 Points] Formulate the following requirements in LTL.***
Guarantee 1: An eating philosopher prevents her neighbours from eating.
Guarantee 2: An eating philosopher eats until she is no longer hungry.
Guarantee 3: Every hungry philosopher eats eventually.
Assumption: An eating philosopher eventually loses her appetite.

$$G_{1i} = AG(e_i \rightarrow (\neg e_{(i-1) \bmod n} \wedge e_{(i+1) \bmod n}))$$

$$G_{2i} = AG((h_i \wedge e_i) \rightarrow X\, e_i)$$

$$G_{3i} = A(h_i \rightarrow F\, e_i)$$

$$A_{1_i} = A(e_i \rightarrow F \neg h_i)$$



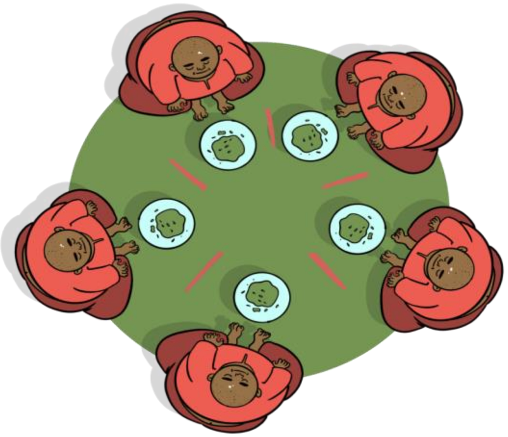
$$\bigwedge_{i=1}^{n} (A_{1_i}) \rightarrow \bigwedge_{i=1}^{n} (G_{1i} \wedge G_{2i} \wedge G_{3i})$$

# Solutions Homework

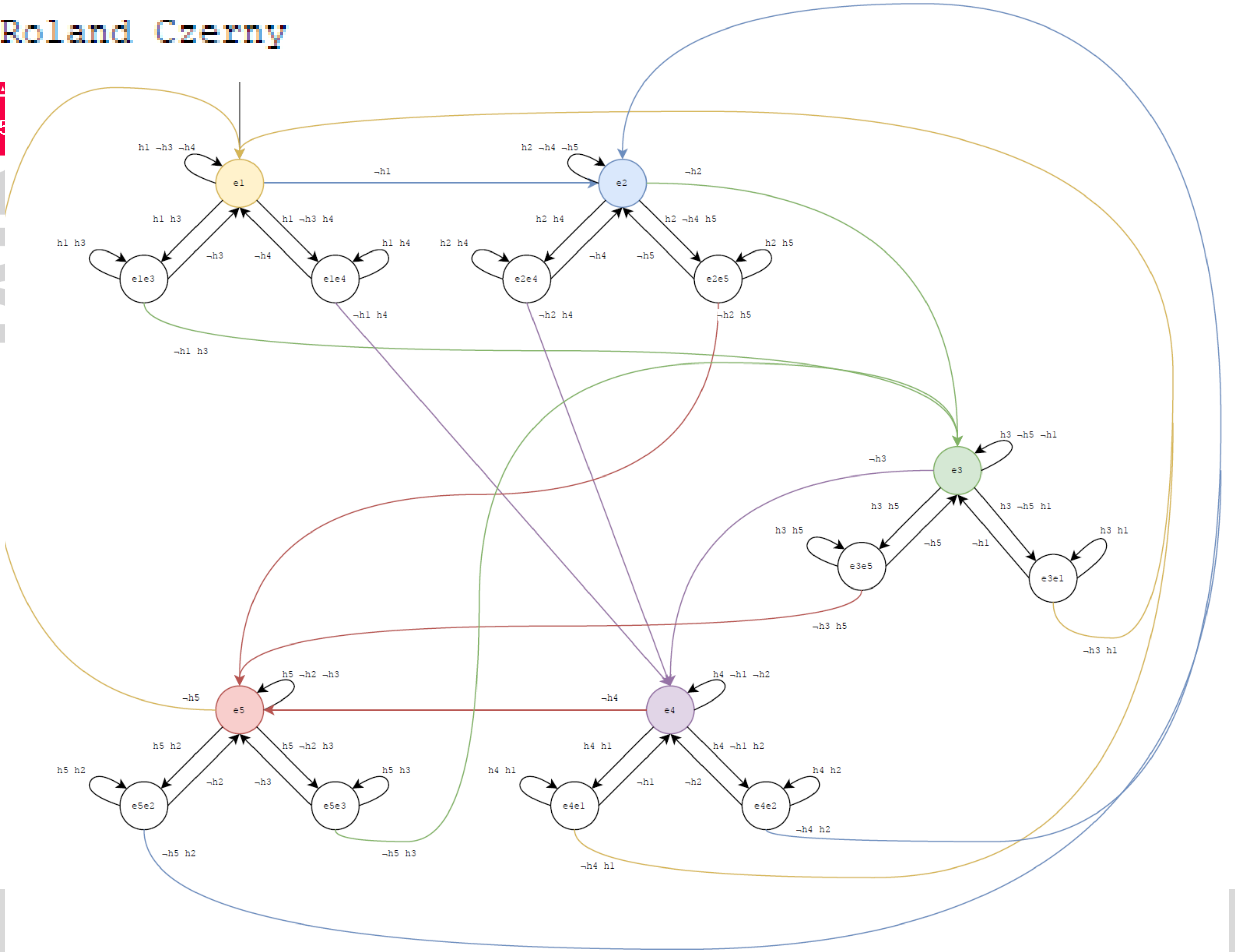
## *The Dining-Philosophers Verification-Problem*

***[6 Points]*** **Your Task: Design a system as Moore machine or Mealy machine for 5 dining philosophers that is**

- **Correct**, i.e., it satisfies the specification
- **and Robust** in the sense that if one philosopher is hungry forever, she eats forever and the only two other philosophers starve**.**



# Roland Czerny



# CTL Model Checking

# The Model Checking Problem

- Given a Kripke structure M and a CTL formula f
- Model Checking Problem:
  - $M \models f$, i.e., M is a model for f
- Alternative Definition
  - Compute $[\![f]\!]_M = \{ s \in S \mid M,s \models f \}$, i.e., all states satisfying f
  - Check $S_0 \subseteq [\![f]\!]_M$ to conclude that $M \models f$

# Illustrative Example: Mutual Exclusion

- Two processes with a joint Boolean signal sem
- Each process $P_i$ has a variable $v_i$ describing its state:
  - $v_i = N$ Non-critical
  - $v_i = T$ Trying
  - $v_i = C$ Critical

# Illustrative Example: Mutual Exclusion

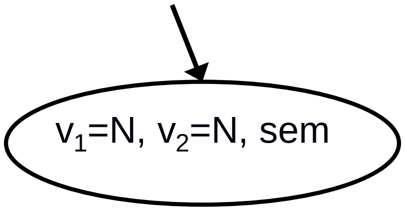
- Each process runs the following program:

```
P_i ::  while (true) {
            if (v_i == N)  v_i = T;
            else if (v_i == T && sem)  { v_i = C; sem = 0; }
            else if (v_i == C)  {v_i = N; sem = 1; }
        }
```

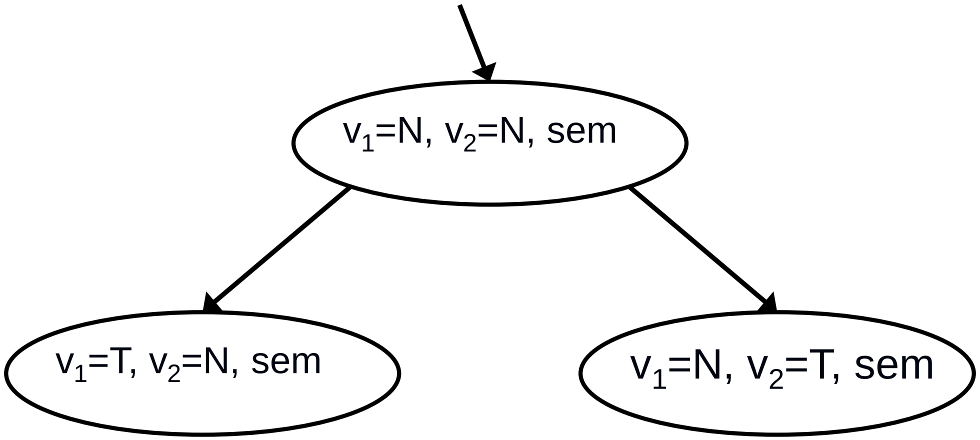
Atomic action (→ each of the three if/else-if branches)

- The full program is: $P_1||P_2$
- Initial state: $(v_1=N, v_2=N, sem)$
- The execution is interleaving

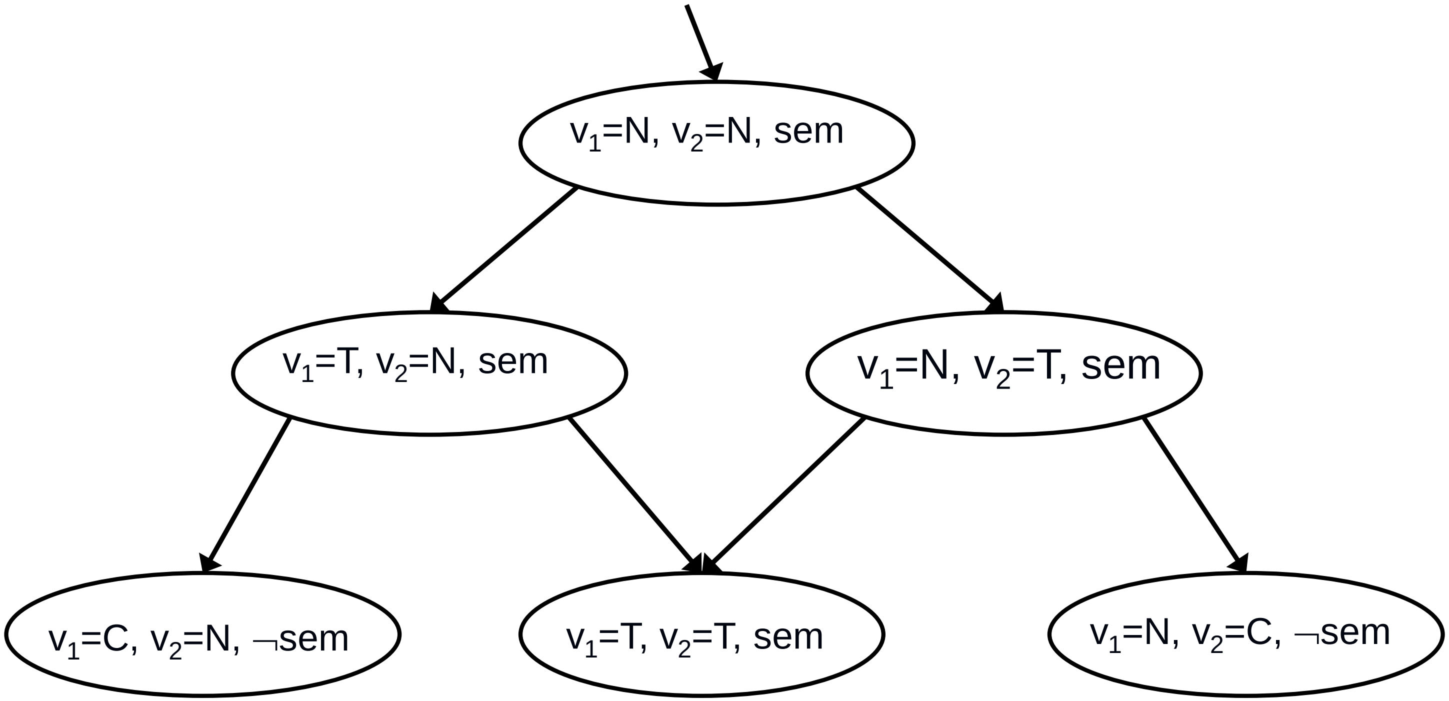
# Illustrative Example: Mutual Exclusion



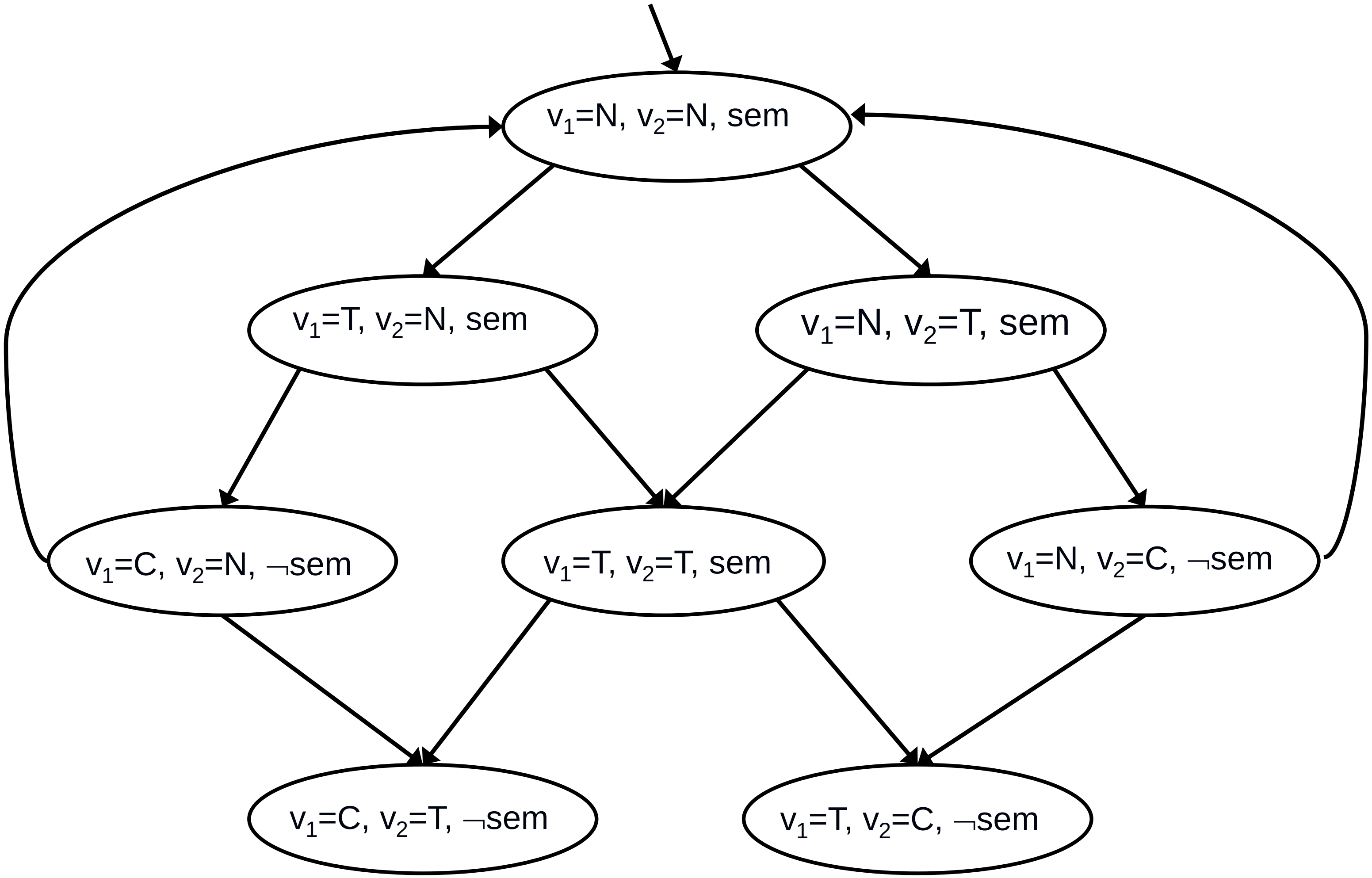
# Illustrative Example: Mutual Exclusion



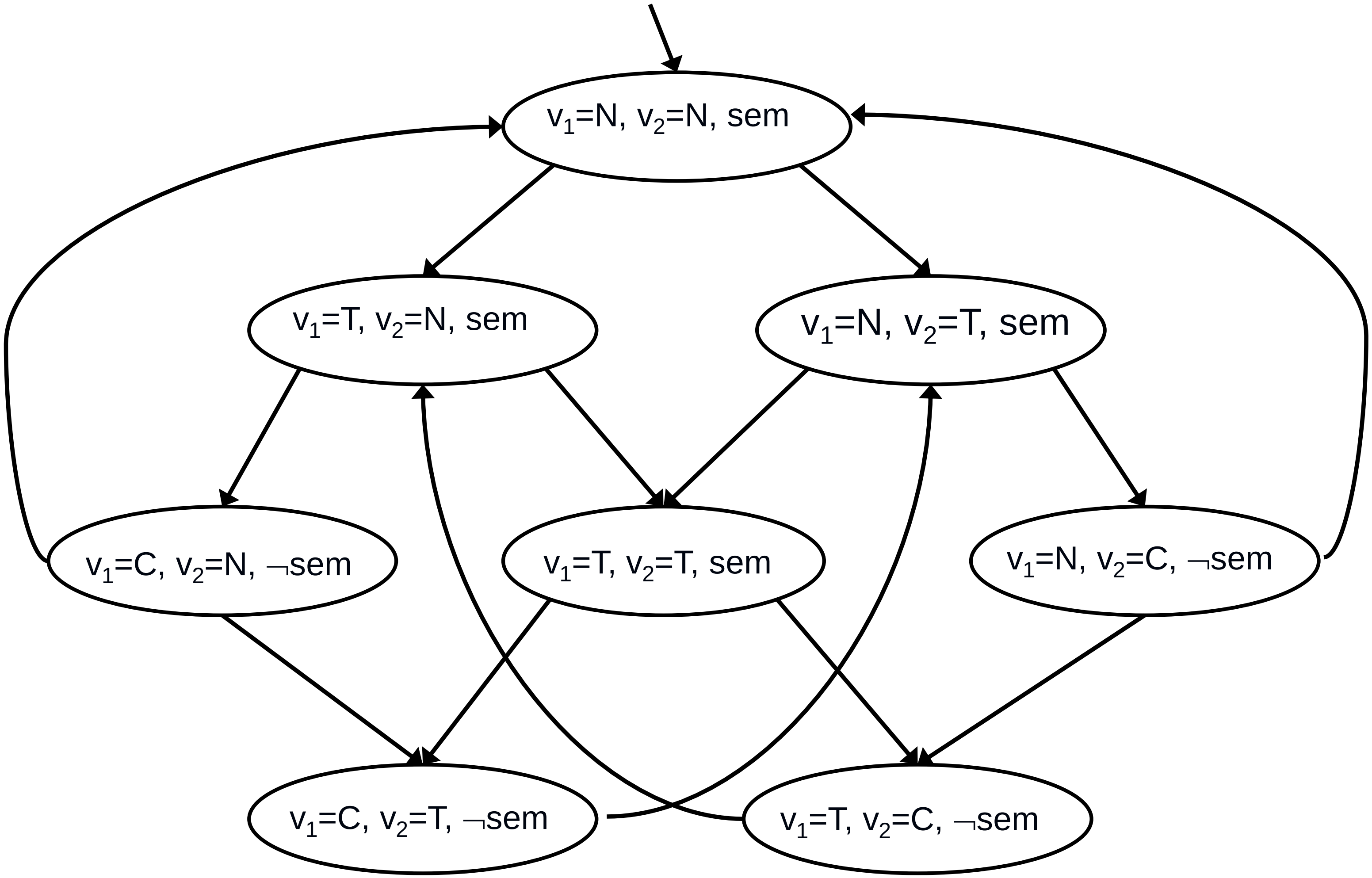
# Illustrative Example: Mutual Exclusion



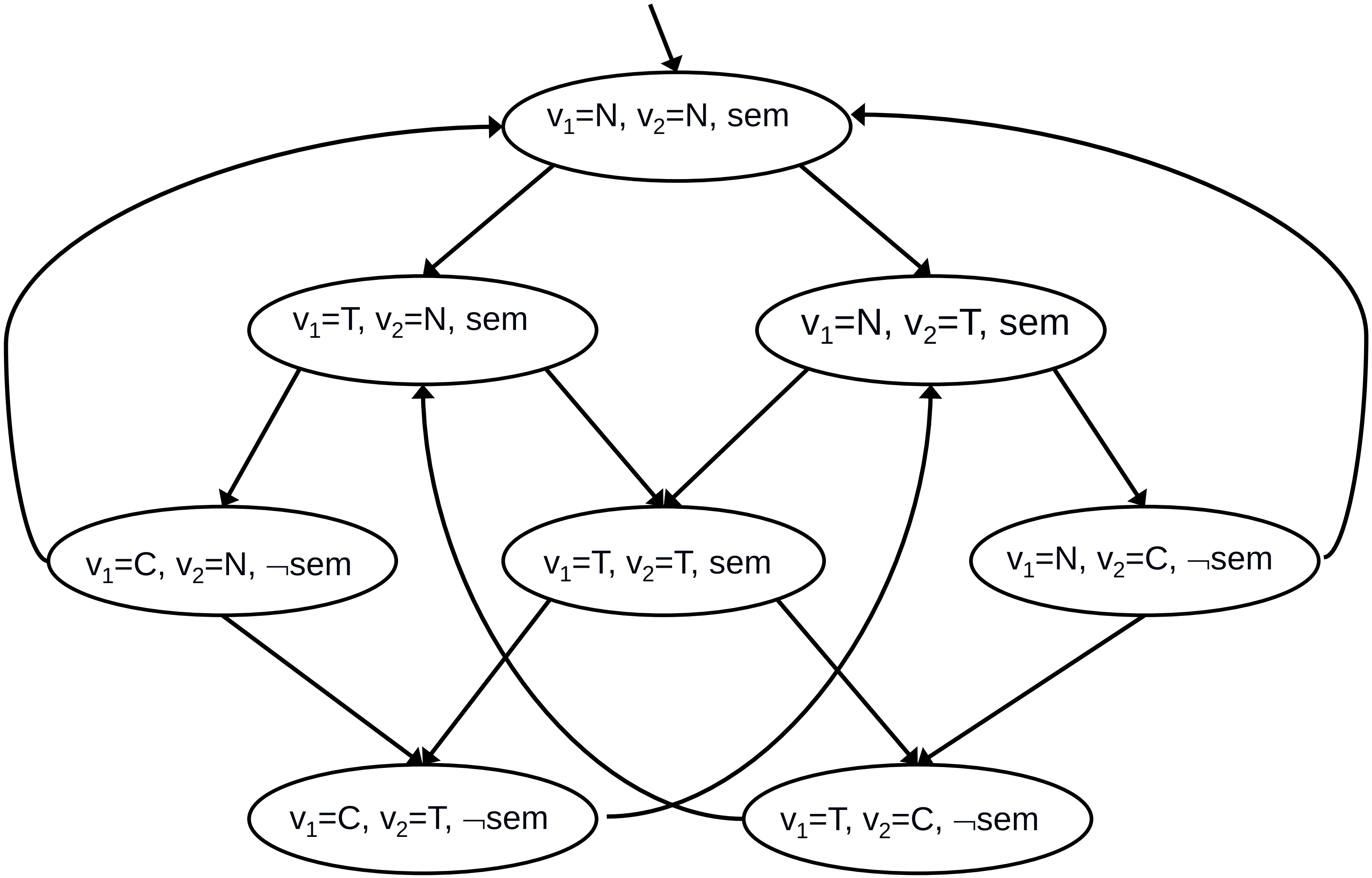
# Illustrative Example: Mutual Exclusion



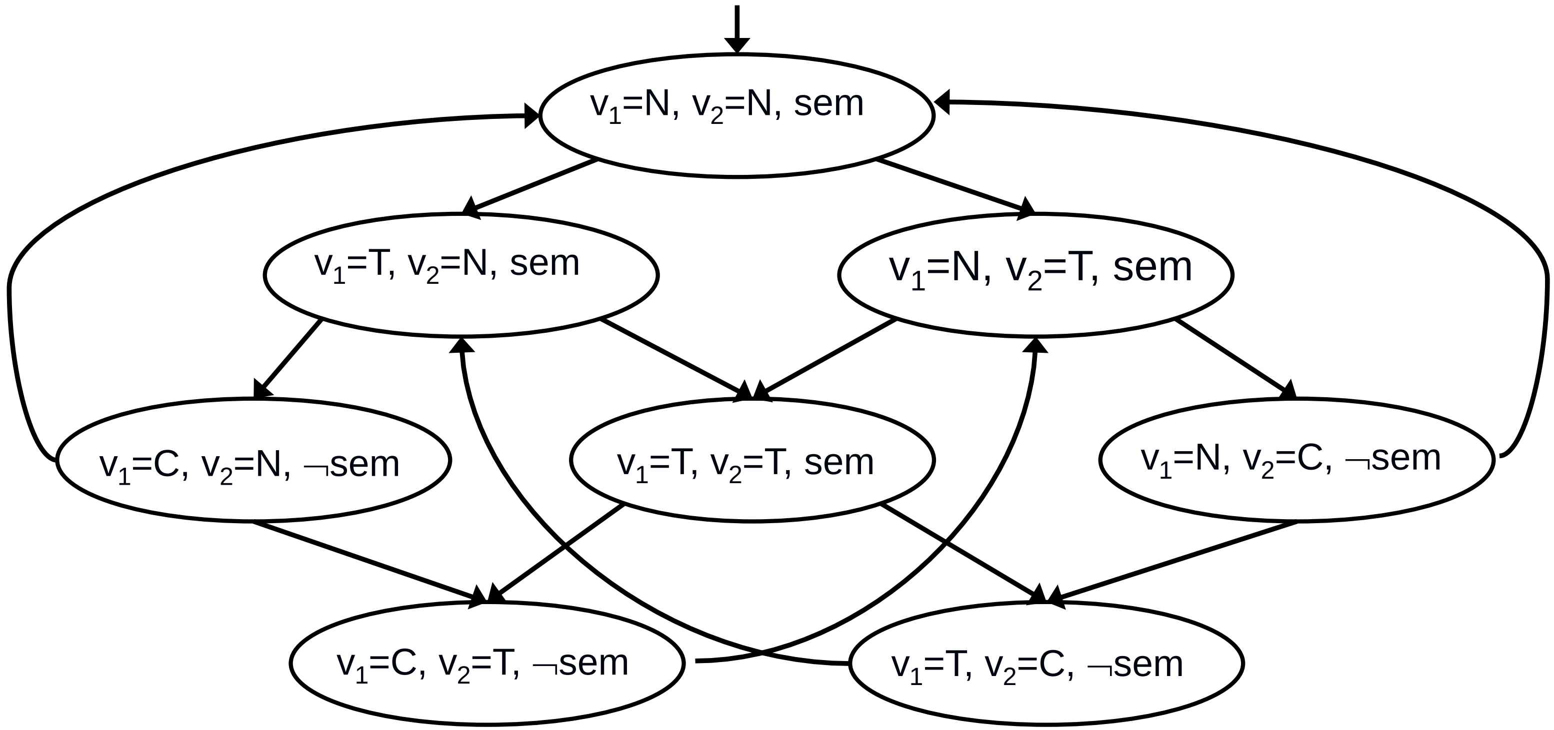
# Illustrative Example: Mutual Exclusion



# Illustrative Example: Mutual Exclusion

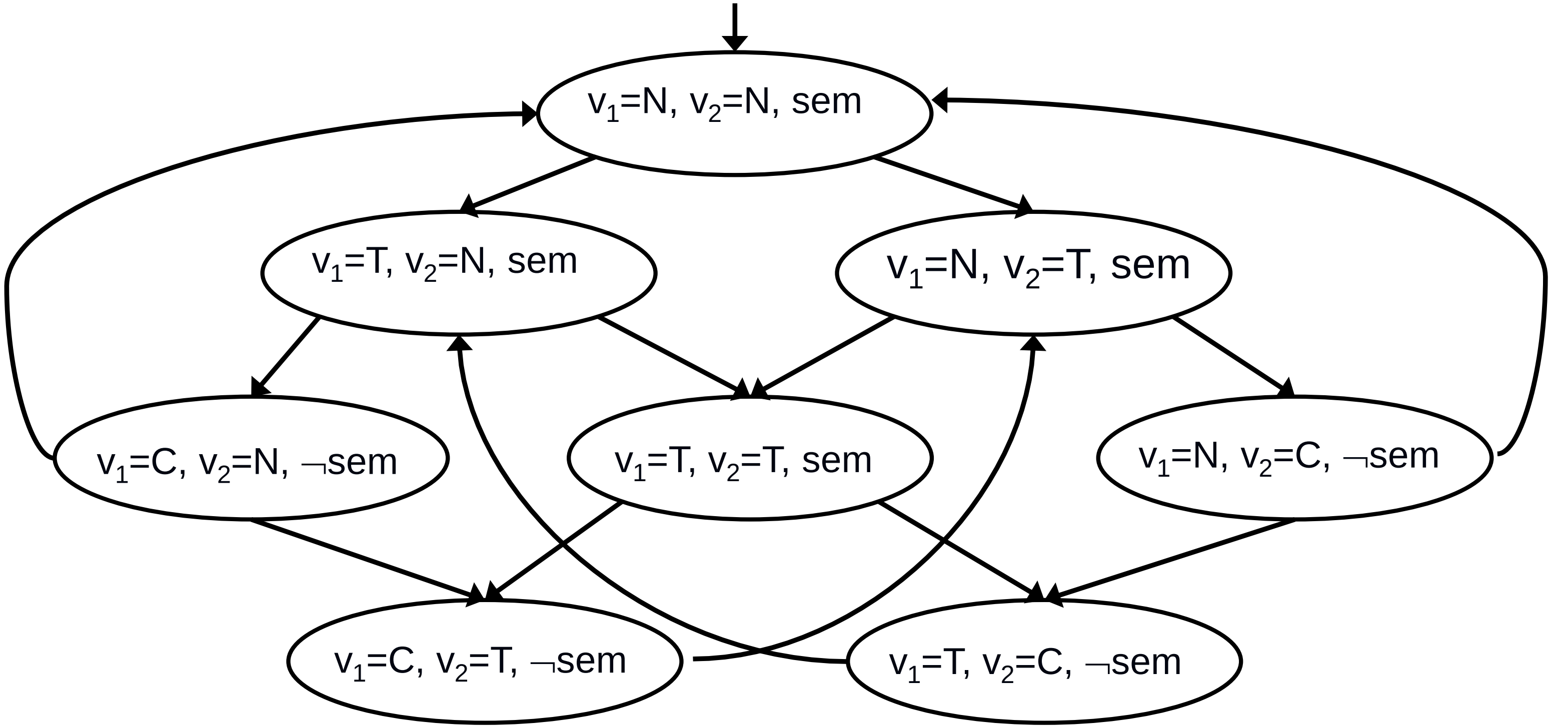


# Illustrative Example: Mutual Exclusion



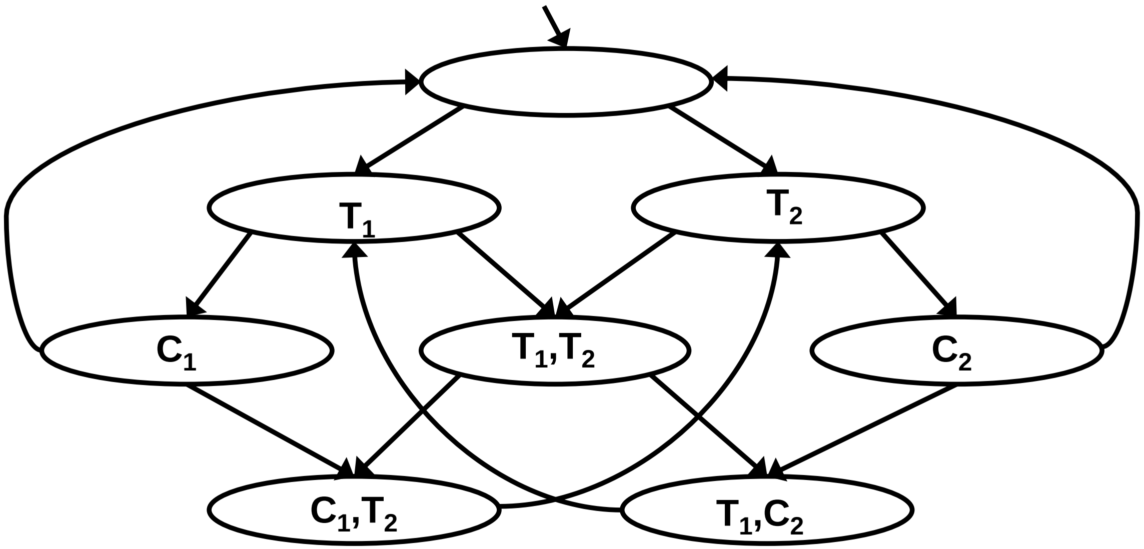

- We define atomic propositions: $AP=\{C_1,C_2,T_1,T_2)$
- A state is labeled with $T_i$ if $v_i$=T
- A state is labeled with $C_i$ if $v_i$=C

# Illustrative Example: Mutual Exclusion



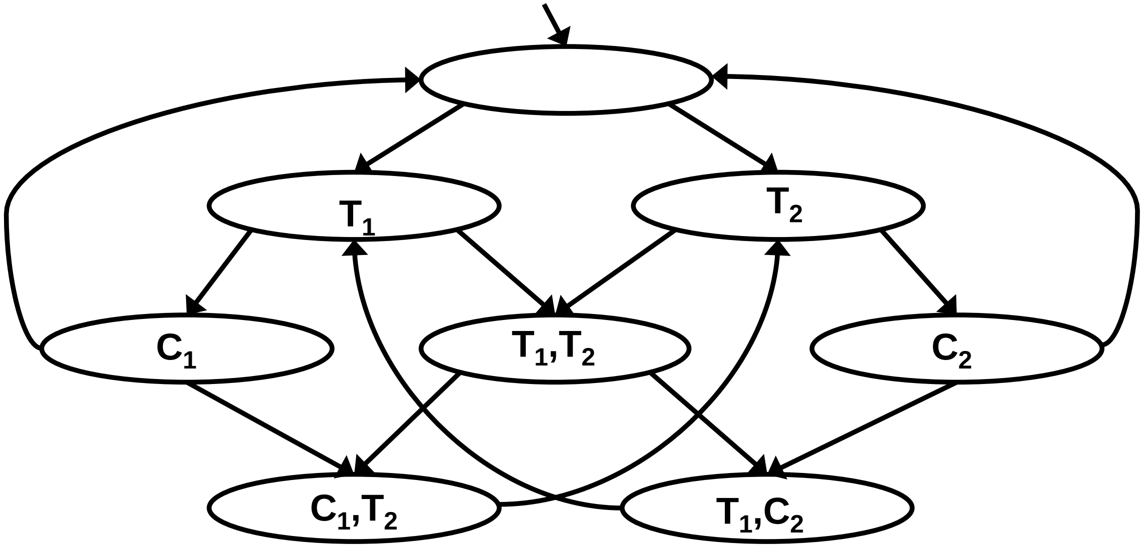

- We define atomic propositions: $AP=\{C_1,C_2,T_1,T_2)$
- A state is labeled with $T_i$ if $v_i=T$
- A state is labeled with $C_i$ if $v_i=C$

# Illustrative Example: Mutual Exclusion



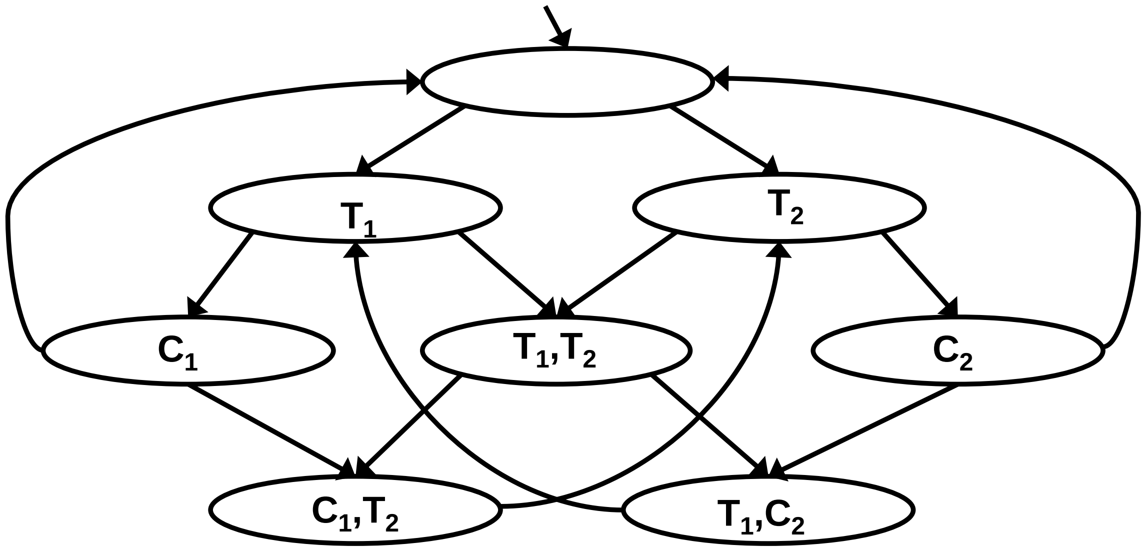

- We define atomic propositions: $AP=\{C_1,C_2,T_1,T_2)$
- A state is labeled with $T_i$ if $v_i=T$
- A state is labeled with $C_i$ if $v_i=C$

# Illustrative Example: Mutual Exclusion



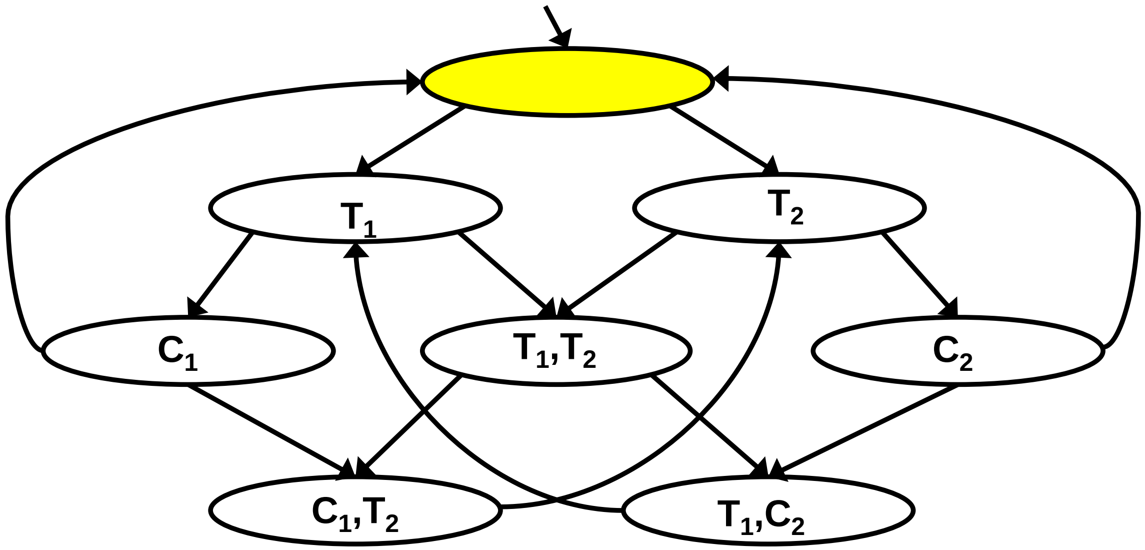

- Does it hold that M ⊨ f?
  - Property 1: f := **AG**¬($C_1 \wedge C_2$)
  - Compute $[\![f]\!]_M = \{ s \in S \mid M,s \models f \}$ and check $S_0 \subseteq [\![f]\!]_M$

# Illustrative Example: Mutual Exclusion




- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(C_1 \wedge C_2)$
- $S_i \equiv$ reachable states from an initial state after **i** steps

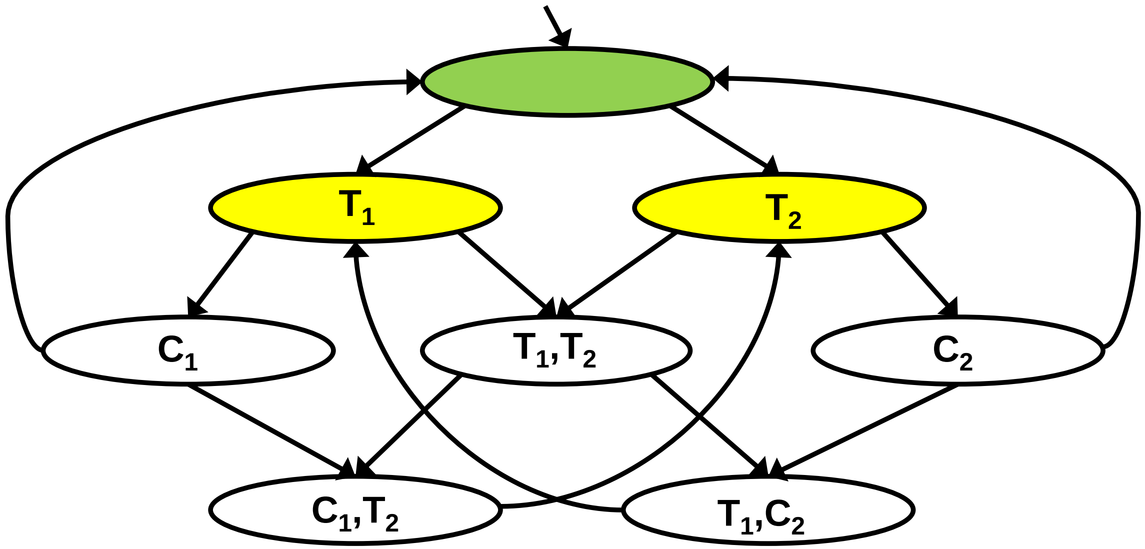
# Illustrative Example: Mutual Exclusion




$S_0$

- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(C_1 \wedge C_2)$
- $S_i \equiv$ reachable states from an initial state after **i** steps

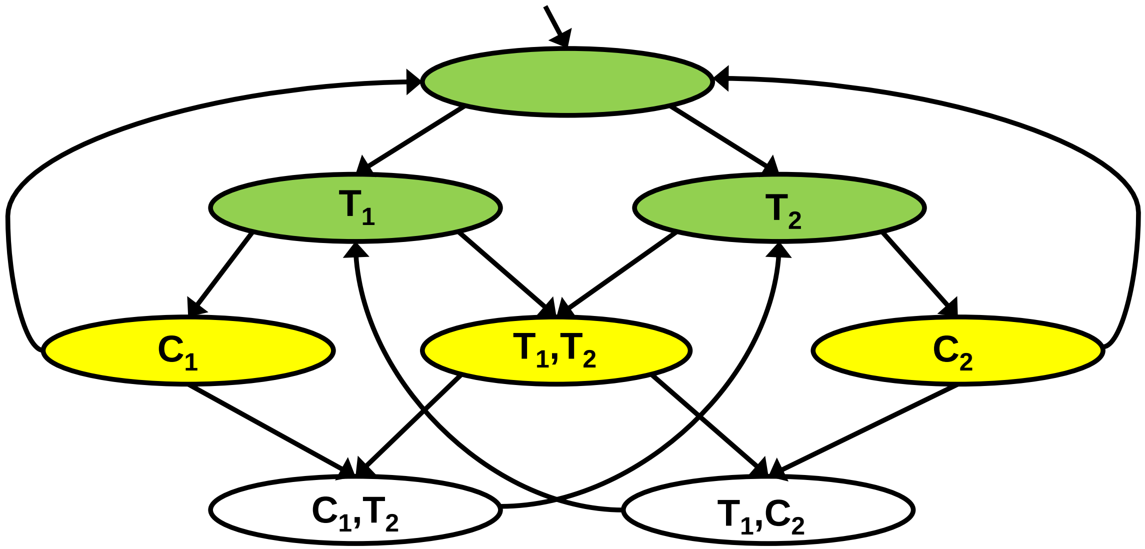
# Illustrative Example: Mutual Exclusion




$S_1$

- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(C_1 \wedge C_2)$
- $S_i \equiv$ reachable states from an initial state after **i** steps

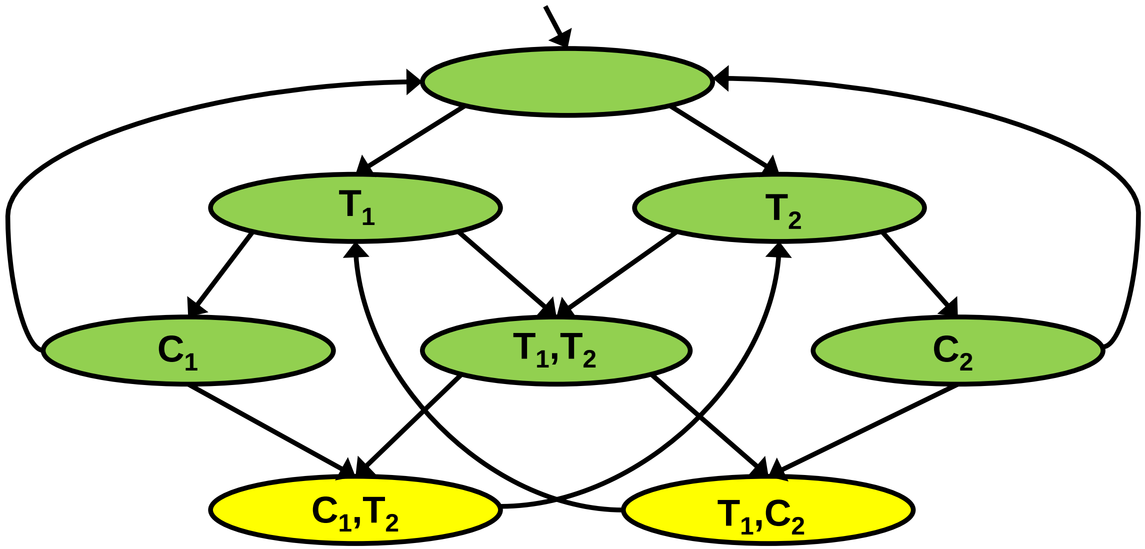
# Illustrative Example: Mutual Exclusion




$S_2$

- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(C_1 \wedge C_2)$
- $S_i \equiv$ reachable states from an initial state after **i** steps

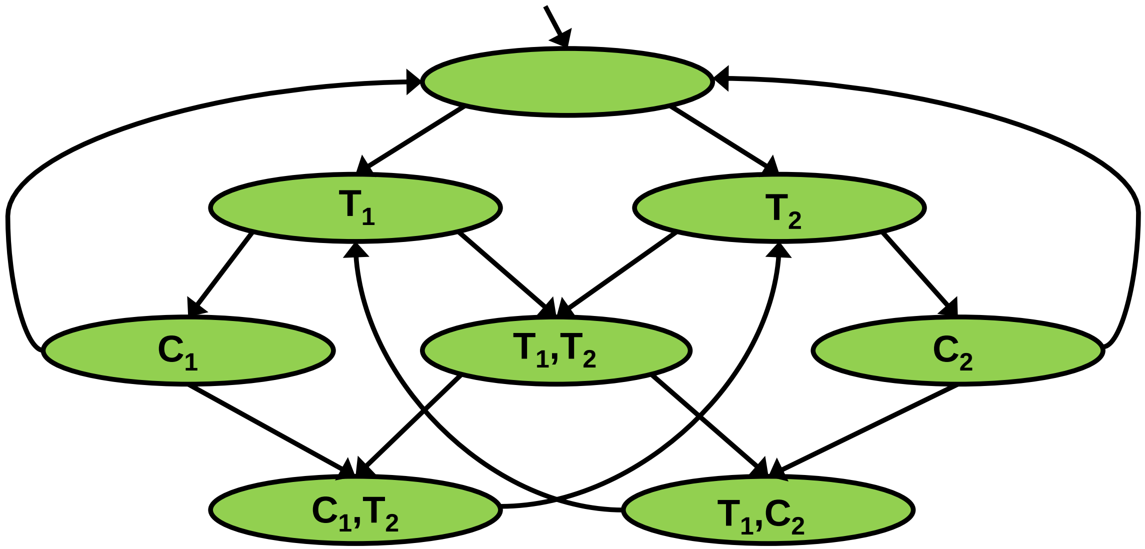
# Illustrative Example: Mutual Exclusion




$S_3$

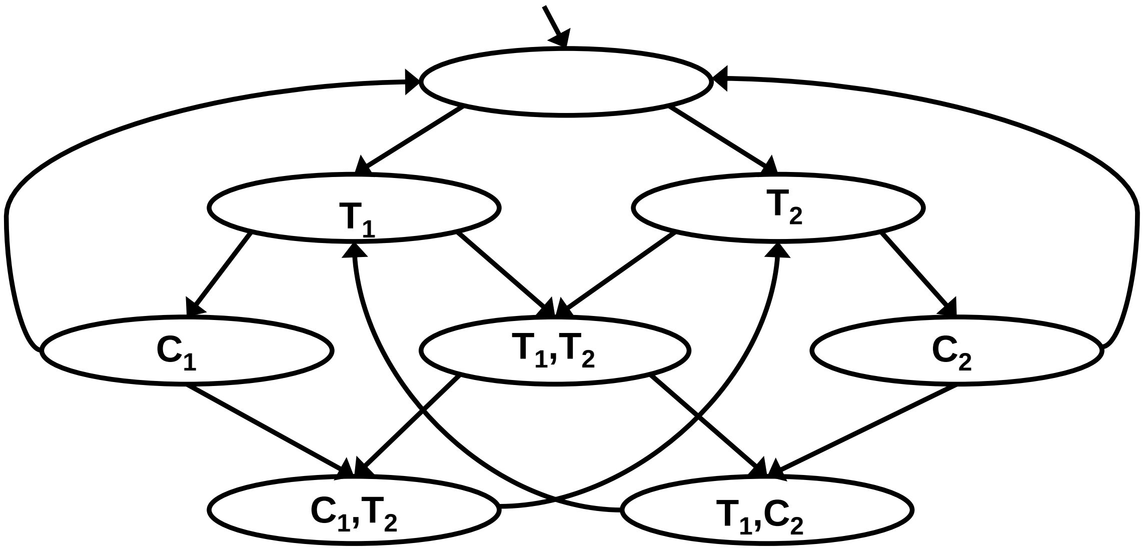
- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(C_1 \wedge C_2)$
- $S_i \equiv$ reachable states from an initial state after **i** steps

# Illustrative Example: Mutual Exclusion




- Does it hold that M ⊨ f?
  - Property 1: f := **AG**$\neg(C_1 \wedge C_2)$ ✓ $M \models AG\ \neg\ (C_1 \wedge C_2)$

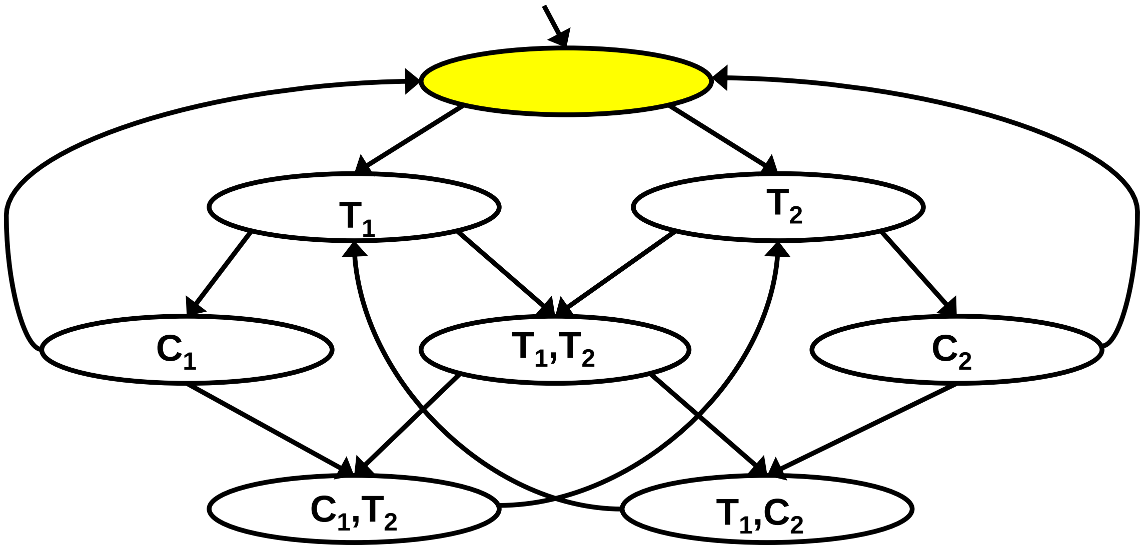
# Illustrative Example: Mutual Exclusion





- Does it hold that $M \models f$?
  - Property 2: $f := \mathbf{AG}\neg(T_1 \wedge T_2)$

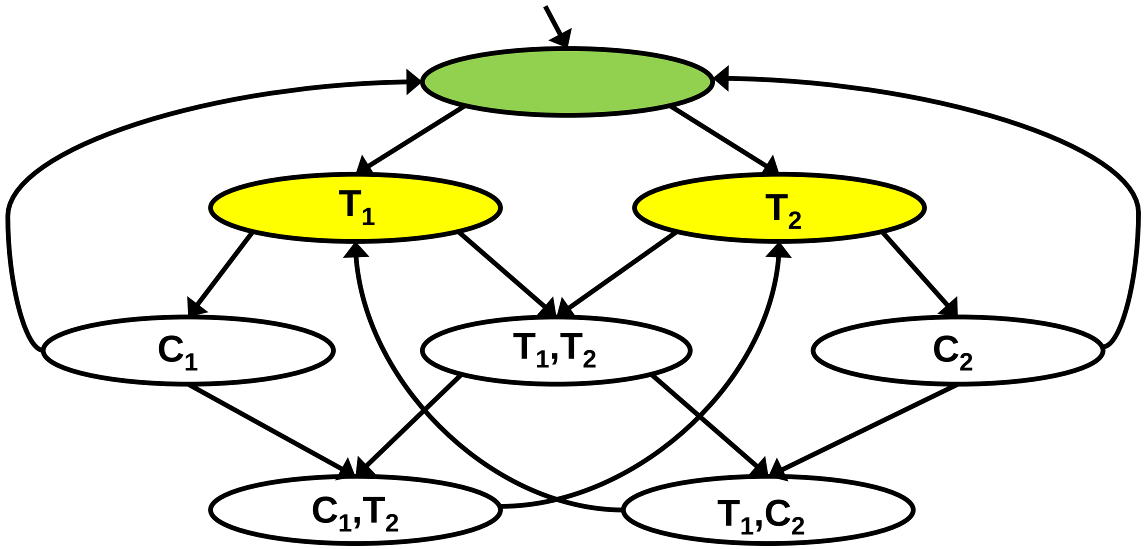
# Illustrative Example: Mutual Exclusion




$S_0$

- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(T_1 \wedge T_2)$
- $S_i \equiv$ reachable states from an initial state after **i** steps

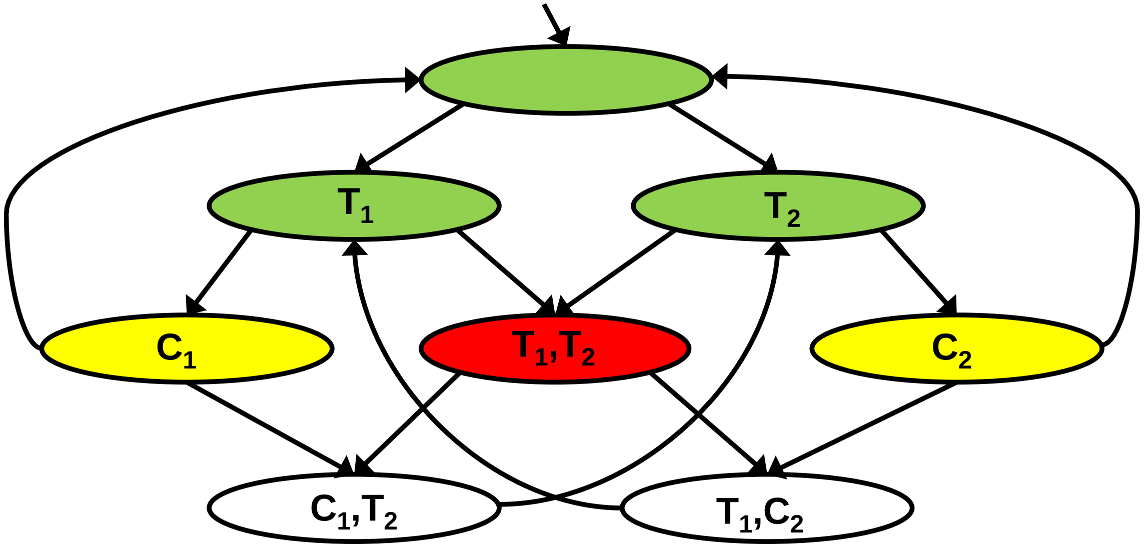
# Illustrative Example: Mutual Exclusion




$S_1$

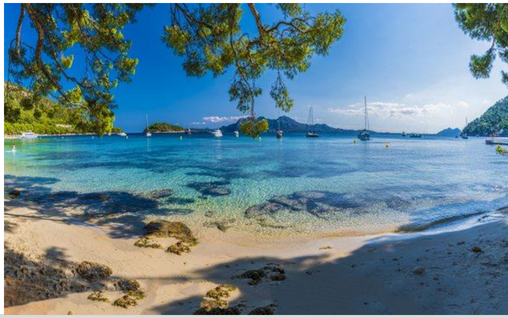
- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(T_1 \wedge T_2)$
- $S_i \equiv$ reachable states from an initial state after **i** steps

# Illustrative Example: Mutual Exclusion

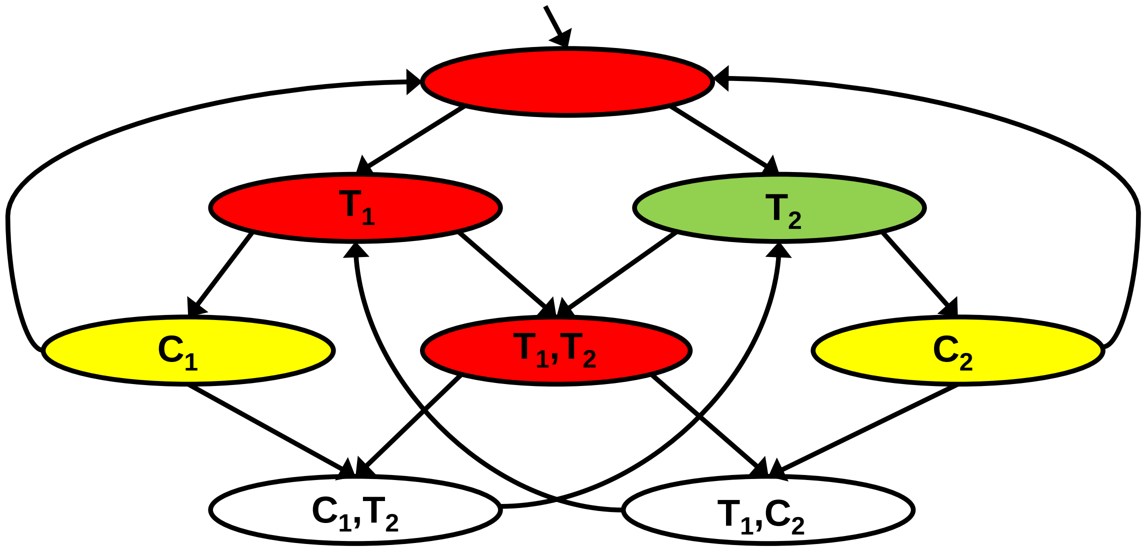



$S_3$

- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(T_1 \wedge T_2)$ ✗ $M \not\models AG \neg (T_1 \wedge T_2)$



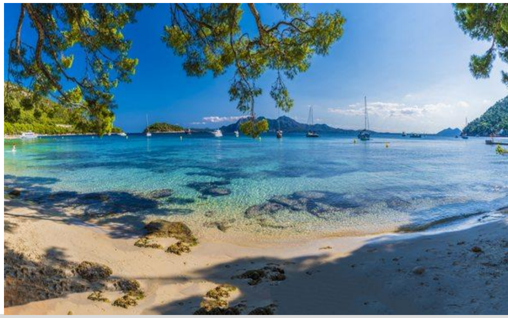
# Illustrative Example: Mutual Exclusion



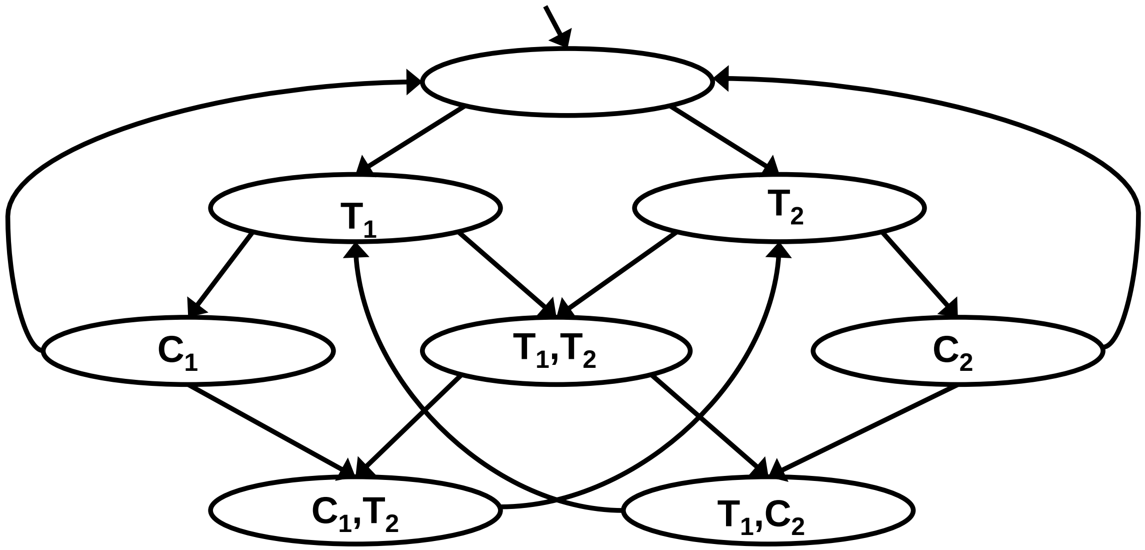

$S_3$

- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(T_1 \wedge T_2)$ ✗ $M \not\models AG \neg (T_1 \wedge T_2)$

- Model checker returns a **counterexample**

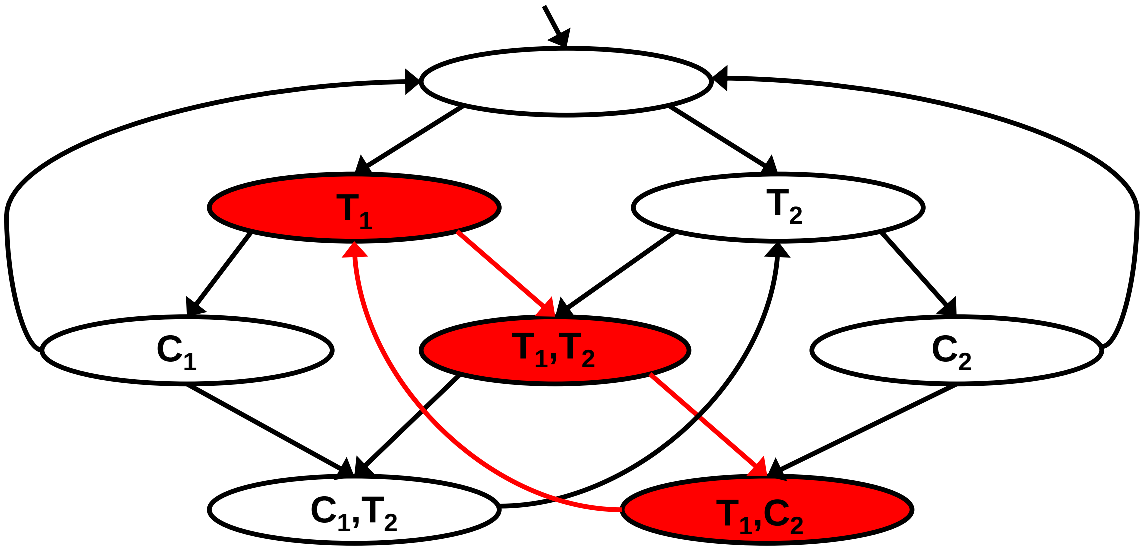


# Illustrative Example: Mutual Exclusion




- Does it hold that $M \models f$?
  - Property 3: $f := \mathbf{AG}\ ((T_1 \rightarrow \mathbf{F}\ C_1) \wedge (T_2 \rightarrow \mathbf{F}\ C_2))$
- In case $M \not\models f$, compute a counterexample

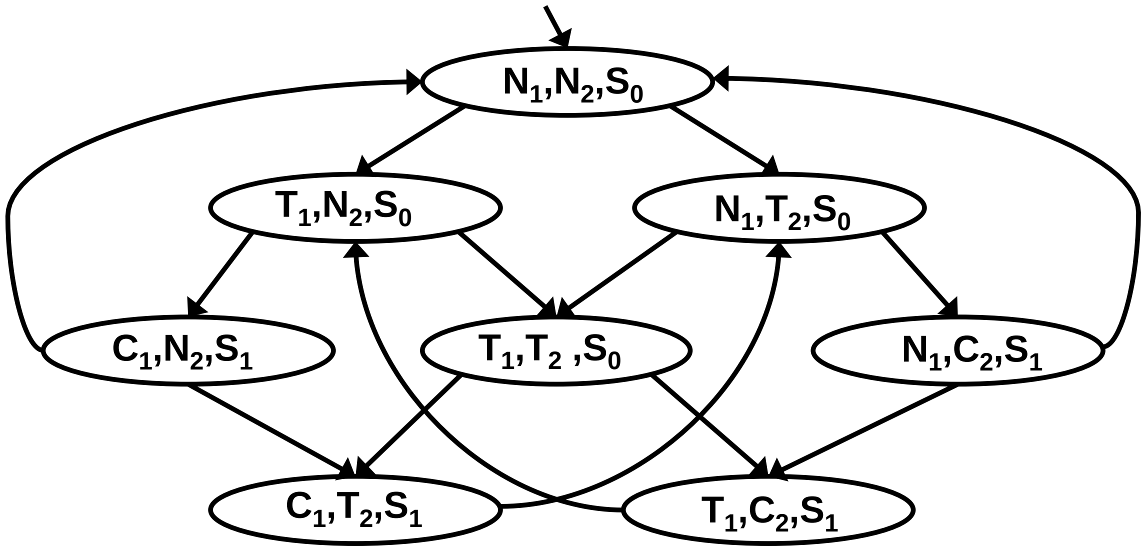
# Illustrative Example: Mutual Exclusion




- Does it hold that $M \models f$?
  - Property 3: $f := \mathbf{AG}\ ((T_1 \rightarrow \mathbf{F}\ C_1) \wedge (T_2 \rightarrow \mathbf{F}\ C_2))$
- In case $M \not\models f$, compute a counterexample

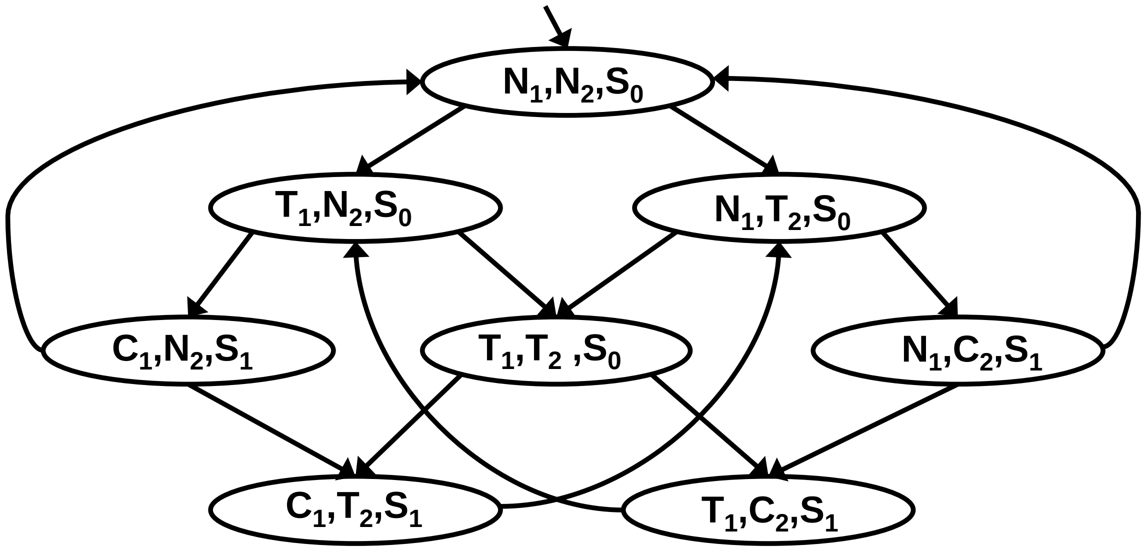
✗ $M \not\models AG\ ((T_1 \rightarrow F\ C_1) \wedge (T_2 \rightarrow F\ C_2))$

# Illustrative Example: Mutual Exclusion




- Does it hold that $M \models f$?
  - Property 4: $f := \mathbf{AG}\ \mathbf{EF}\ (N_1 \wedge N_2 \wedge S_0)$
- How would you express property 4 in natural language?
- In case $M \not\models f$, compute a counterexample

# Illustrative Example: Mutual Exclusion




- Does it hold that $M \models f$? ✓
  - Property 4: $f := \mathbf{AG}\ \mathbf{EF}\ (N_1 \wedge N_2 \wedge S_0)$
- ***No matter where you are***
  ***there is always a way***
  ***to get to the initial state (restart)***

# Explicit Model Checking for CTL

# Explicit Model Checking for CTL

- Explicit MC uses Kripke structure M as a graph:
  (S, R) with labeling L

- Use graph traversal algorithms (e.g., Depth First Search (DFS) or Breadth First Search (BFS)) to traverse states and paths of M

# CTL Model Checking

Receives:

- A Kripke structure M, modeling a system
- A CTL formula f, describing a property

- Determines whether $M \models f$

- Alternatively, it returns $[\![f]\!] = \{ s \in S \mid M,s \models f \}$
  - M is omitted from $[\![f]\!]_M$ when clear from the context

# CTL Model Checking $M \models f$

The goal of MC is to compute $[\![g]\!]_M$
for every subformula g of f, including $[\![f]\!]_M$

# CTL Model Checking $M \models f$

The goal of MC is to compute $[\![g]\!]_M$
for every subformula g of f, including $[\![f]\!]_M$

- Work iteratively on subformulas of **f**
  - from **simpler** to **complex** subformulas

# CTL Model Checking $M \models f$

The goal of MC is to compute $[\![g]\!]_M$
for every subformula g of f, including $[\![f]\!]_M$

- Work iteratively on subformulas of **f**
  - from **simpler** to **complex** subformulas

- For checking **AG(** request $\rightarrow$ **AF** grant**)**
  - Check grant, request
  - Then check **AF** grant
  - Next check request $\rightarrow$ **AF** grant
  - Finally check **AG(** request $\rightarrow$ **AF** grant**)**

# CTL Model Checking $M \models f$

- For each s, computes label(s), which is the set of subformulas of f that are true in s

# CTL Model Checking $M \models f$

- For each s, computes label(s), which is the set of subformulas of f that are true in s

- We check subformula g of f only after all subformulas of g have already been checked

# CTL Model Checking $M \models f$

- For each s, computes label(s), which is the set of subformulas of f that are true in s
- We check subformula g of f only after all subformulas of g have already been checked
- For subformula g, the algorithm adds g to label(s) for every state s that satisfies g

# CTL Model Checking $M \models f$

- For each s, computes label(s), which is the set of subformulas of f that are true in s

- We check subformula g of f only after all subformulas of g have already been checked

- For subformula g, the algorithm adds g to label(s) for every state s that satisfies g

- When we finish checking g, the following holds:
  - $g \in label(s) \Leftrightarrow M,s \models g$

# CTL Model Checking $M \models f$

- For each s, computes label(s), which is the set of subformulas of f that are true in s

- $M \models f$ if and only if $f \in label(s)$ for all initial states s of M
  - $M \models f$ if and only if $S_0 \subseteq [\![f]\!]_M$

# Minimal set of operators for CTL

- All CTL formulas can be transformed to use only the operators:
  - $\neg$, $\vee$, **EX**, **EU**, **EG**

- MC algorithm needs to handle AP and $\neg$, $\vee$, EX, EU, EG

# Model Checking Atomic Propositions

- Procedure for labeling the states satisfying $p \in AP$:

$$\underbrace{p \in label(s)}_{\text{Held by alg}} \Leftrightarrow \underbrace{p \in L(s)}_{\text{Defined by M}}$$

# Model Checking ¬, ∨- Formulas

- Let $f_1$ and $f_2$ be subformulas that have already been checked
  - added to label(s), when needed


Give the procedures for labeling the states satisfying $\neg f_1$ and $f_1 \vee f_2$

# Model Checking ¬, ∨- Formulas

- Let $f_1$ and $f_2$ be subformulas that have already been checked
  - added to label(s), when needed
- Give the procedures for labeling the states satisfying $\neg f_1$ and $f_1 \vee f_2$
  - $\neg f_1$ add to label(s) if and only if $f_1 \notin label(s)$
  - $f_1 \vee f_2$ add to label(s) if and only if $f_1 \in labels(s)$ **or** $f_2 \in label(s)$

# Model Checking $g = EX\,f_1$

Give the procedures for labeling states satisfying $EX f_1$

# Model Checking $g = EX\, f_1$

- Give the procedures for labeling states satisfying $EX\, f_1$
  - Add g to label(s) if and only if s has a successor t such that $f_1 \in$ label(t)

```
procedure CheckEX (f1)
    T := { t | f1 ∈ label(t) }
    while T ≠ ∅  do
        choose t ∈T;  T := T \ {t};
        for all s  such that  R(s,t) do
            if EX f1 ∉ label(s) then
                label(s) : = label(s) ∪ { EX f1};
```

# Model Checking $g = E(f_1 U f_2)$

Procedures for labeling states satisfying $E(f_1 U f_2)$

- Think how you can rewrite the procedure CheckEX

```
procedure CheckEX (f1)
  T := { t | f1 ∈ label(t) }

while T ≠ ∅  do
    choose t ∈T;  T := T \ {t};
    for all s  such that  R(s,t) do
        if EX f1 ∉ label(s) then
           label(s) : = label(s) ∪ { EX f1};
```

```
procedure CheckEU (f1,f2)
  T :=

  for all t∈T do
     label(t) :=

  while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
```

# Model Checking $g = E(f_1 U f_2)$

- Procedures for labeling states satisfying $E(f_1 U f_2)$
  - Rewriting the procedure CheckEX

```
procedure CheckEX (f1)
  T := { t | f1 ∈ label(t) }

while T ≠ ∅  do
    choose t ∈T;  T := T \ {t};
    for all s  such that  R(s,t) do
        if EX f1 ∉ label(s) then
            label(s) : = label(s) ∪ { EX f1};
```

```
procedure CheckEU (f1,f2)
  T := { t | f2 ∈ label(t) }

  for all t∈T do
     label(t) :=

  while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
```



SCOS Secure & Correct Systems

# Model Checking $g = E(f_1 U f_2)$

- Procedures for labeling states satisfying $E(f_1 U f_2)$
  - Rewriting the procedure CheckEX

```
procedure CheckEX (f1)
  T := { t | f1 ∈ label(t) }

while T ≠ ∅  do
    choose t ∈T;  T := T \ {t};
    for all s  such that  R(s,t) do
        if EX f1 ∉ label(s) then
            label(s) : = label(s) ∪ { EX f1};
```

```
procedure CheckEU (f1,f2)
  T := { t | f2 ∈ label(t) }

  for all t∈T do
     label(t) := label(t) ∪ { E(f1 U f2) }

  while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
```

# Model Checking $g = E(f_1 U f_2)$

- Procedures for labeling states satisfying $E(f_1 U f_2)$
  - Rewriting the procedure CheckEX

```
procedure CheckEX (f1)
  T := { t | f1 ∈ label(t) }

while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
          if EX f1 ∉ label(s) then
               label(s) : = label(s) ∪ { EX f1};
```

```
procedure CheckEU (f1,f2)
  T := { t | f2 ∈ label(t) }

  for all t∈T do
     label(t) := label(t) ∪ { E(f1 U f2) }

  while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
          if E(f1 U f2) ∉ label(s) and f1 ∈ label(s) then
               label(s) : = label(s) ∪ {E(f1 U f2) };
```

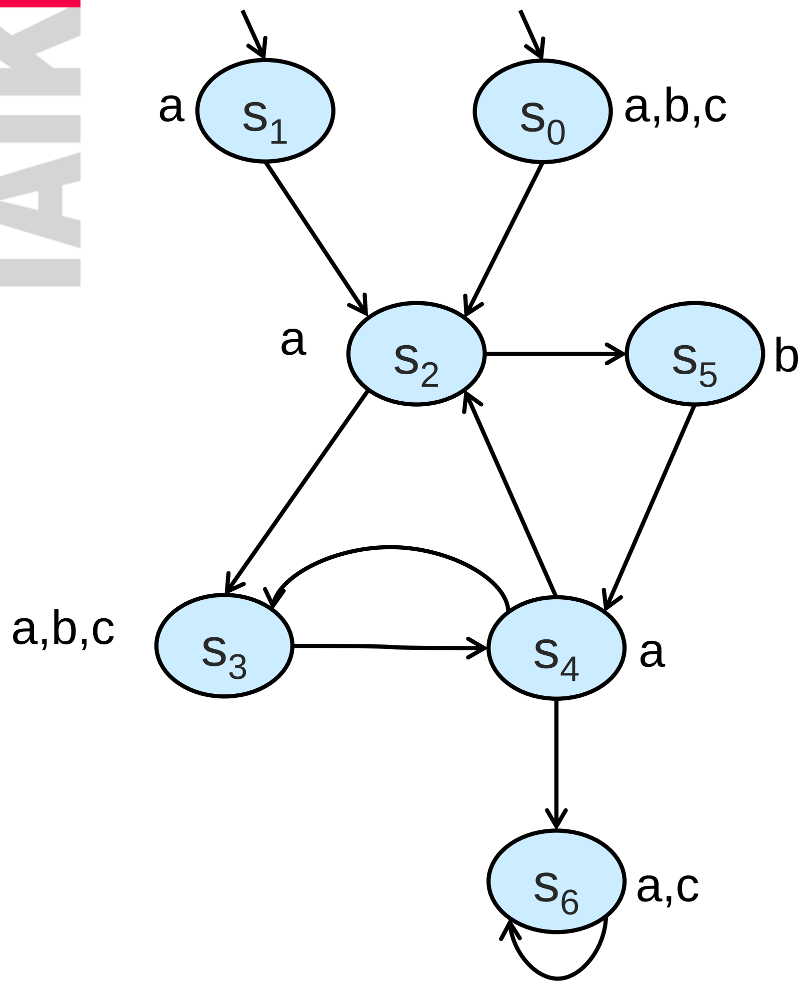
# Model Checking $g = E(f_1 U f_2)$

- Procedures for labeling states satisfying $E(f_1 U f_2)$
  - Rewriting the procedure CheckEX

```
procedure CheckEX (f1)
  T := { t | f1 ∈ label(t) }



while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
          if EX f1 ∉ label(s) then
               label(s) : = label(s) ∪ { EX f1};
```

```
procedure CheckEU (f1,f2)
  T := { t | f2 ∈ label(t) }

  for all t∈T do
     label(t) := label(t) ∪ { E(f1 U f2) }

  while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
          if E(f1 U f2) ∉ label(s) and f1 ∈ label(s) then
               label(s) : = label(s) ∪ {E(f1 U f2) };
               T : = T ∪ {s}
```

# Example: Model Checking *U* Formulas

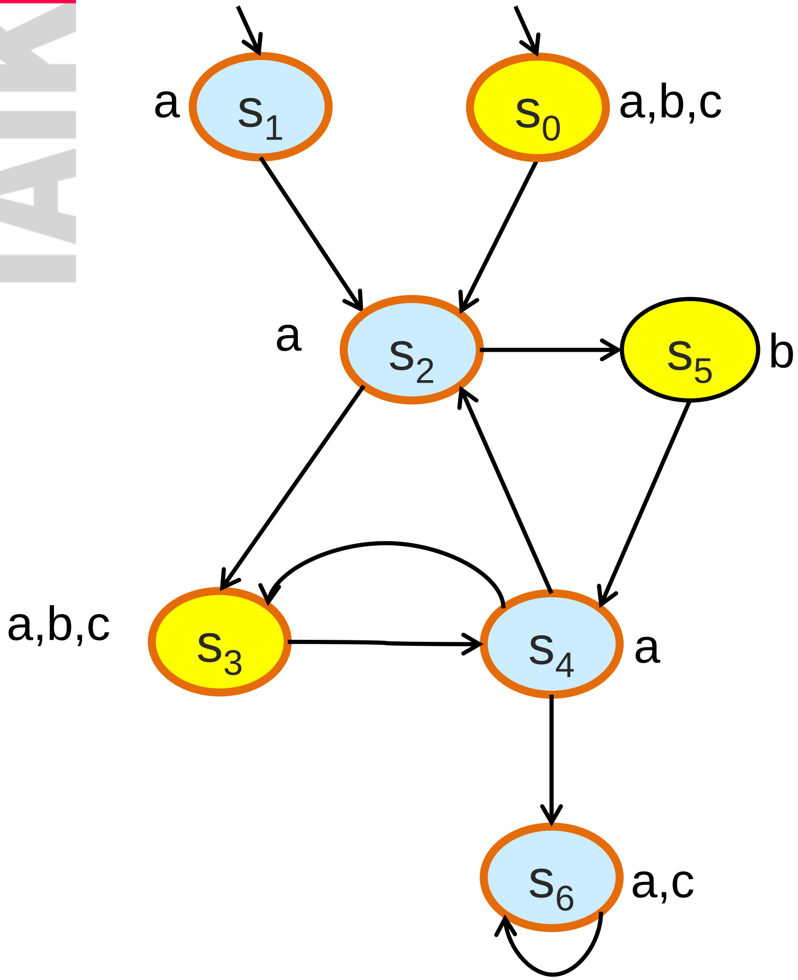



ToDo Does it hold that M ⊨ f?

- $f := E(aUb)$

```
procedure CheckEU (f1,f2)
  T := { t | f2 ∈ label(t) }

  for all t∈T do
     label(t) := label(t) ∪ { E(f1 U f2) }

  while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
        if E(f1 U f2) ∉ label(s) and f1 ∈ label(s) then
           label(s) : = label(s) ∪ {E(f1 U f2) };
           T : = T ∪ {s}
```

# Example: Model Checking $U$ Formulas

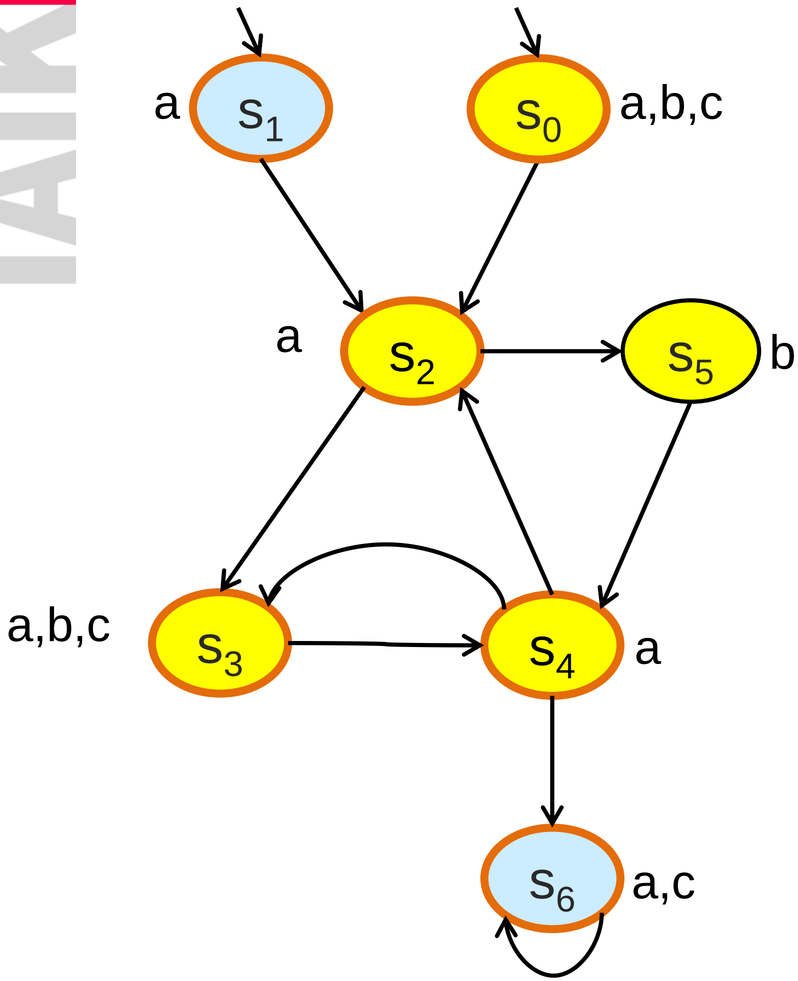



Does it hold that M ⊨ f?

- $f := E(aUb)$

```
procedure CheckEU (f1,f2)
  T := { t | f2 ∈ label(t) }

  for all t∈T do
     label(t) := label(t) ∪ { E(f1 U f2) }

  while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
        if E(f1 U f2) ∉ label(s) and f1 ∈ label(s) then
           label(s) : = label(s) ∪ {E(f1 U f2) };
           T : = T ∪ {s}
```

# Example: Model Checking $U$ Formulas

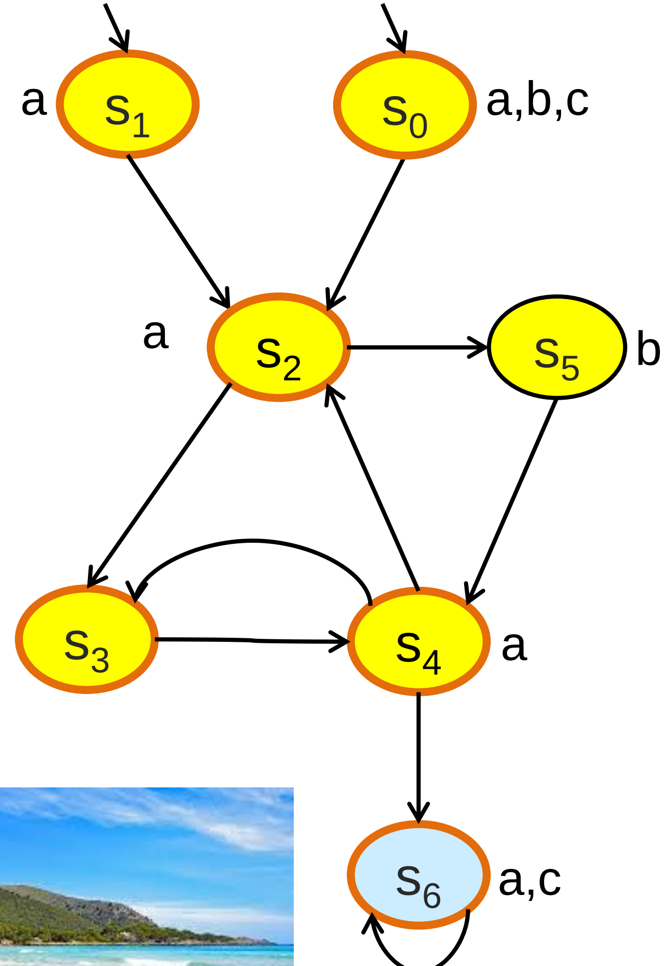



Does it hold that M ⊨ f?

- $f := E(aUb)$

```
procedure CheckEU (f1,f2)
  T := { t | f2 ∈ label(t) }

  for all t∈T do
     label(t) := label(t) ∪ { E(f1 U f2) }

  while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
        if E(f1 U f2) ∉ label(s) and f1 ∈ label(s) then
           label(s) : = label(s) ∪ {E(f1 U f2) };
           T : = T ∪ {s}
```

# Example: Model Checking *U* Formulas




Does it hold that M ⊨ f?

- $f := E(aUb)$

✓ M ⊨ E(aUb)

[[E(aUb)]] = {0,3,5,4}

```
procedure CheckEU (f_1,f_2)
  T := { t | f_2 ∈ label(t) }

  for all t∈T do
     label(t) := label(t) ∪ { E(f_1 U f_2) }

  while T ≠ ∅  do
     choose t ∈T;  T := T \ {t};
     for all s  such that  R(s,t) do
        if E(f_1 U f_2) ∉ label(s) and f_1 ∈ label(s) then
           label(s) : = label(s) ∪ {E(f_1 U f_2) };
           T : = T ∪ {s}
```

# Model Checking $g = EG f_1$

Observation:

s ⊨ **EG** $f_1$

iff

There is a path π, starting at s, such that π ⊨ **G** $f_1$

# Model Checking $g = EG f_1$

Observation:

s ⊨ **EG** $f_1$

iff

There is a path π, starting at s, such that π ⊨ **G** $f_1$

iff

There is a path from s to a strongly connected component, where all states satisfy $f_1$

# Model Checking $g = EGf_1$

- A Strongly Connected Component (SCC) in a graph is a subgraph C such that every node in C is reachable from any other node in C via nodes in C
- An SCC  C is maximal (MSCC) if it is not contained in any other SCC in the graph
  - Possible to find all MSCC in linear time O(|S|+|R|) (Tarjan)
- C is nontrivial if it contains at least one edge. Otherwise, it is trivial

# Model Checking $g = EGf_1$

- Reduced structure for M and $f_1$:
  - Remove from M all states such that $f_1 \notin$ label(s)

- Resulting model: $M' = (S', R', L')$
  - $S' = \{ s \mid M, s \models f_1 \}$
  - $R' = ( S' \times S' ) \cap R$
  - $L'(s') = L(s')$ for every $s' \in S'$

# Model Checking $g = EG f_1$

- Reduced structure for M and $f_1$:
  - Remove from M all states such that $f_1 \notin$ label(s)

- Resulting model: $M' = (S', R', L')$
  - $S' = \{ s \mid M, s \models f_1 \}$
  - $R' = ( S' \times S' ) \cap R$
  - $L'(s') = L(s')$ for every $s' \in S'$

- Theorem: $M,s \models EG\ f_1$ iff

1. $s \in S'$ and
2. There is *a path* in $M'$ from $s$ to some state $t$ in a nontrivial MSCC of $M'$

# Model Checking $g = EG f_1$

**procedure** CheckEG ($f_1$)
S′ := {s | $f_1$ ∈ label(s) }
MSCC := { C | C is a nontrivial MSCC of M′ }
T := $\cup_{C \in MSCC}$ { s | s ∈ C}

**for all** t∈T **do**
label(t) := label(t) ∪ { EG $f_1$}

# Model Checking $g = EG f_1$

```
procedure CheckEG (f1)
  S′ := {s | f1 ∈ label(s) }
  MSCC := { C | C is a nontrivial MSCC of M′ }
  T := ∪C ∈MSCC { s | s ∈ C}

  for all t∈T do
      label(t) := label(t) ∪ { EG f1}

  while T ≠ ∅  do
      choose t ∈T;  T := T \ {t};
      for all s ∈S'  such that  R'(s,t) do
          if EG f1 ∉ label(s) then
              label(s) : = label(s) ∪ {EG f1};
              T : = T ∪ {s}
```


# Model Checking Complexity

## Steps per Subformula

- MC Atomic Propositions
- MC ¬, ∨ formulas
- MC g = EX $f_1$
- MC $g = E(f_1 U f_2)$
- MC $g = EG f_1$

ToDo

# Model Checking Complexity

## Steps per Subformula

- MC Atomic Propositions
  - O(|S|) steps
- MC ¬, ∨ formulas
  - 
- MC g = EX $f_1$
  - 
- MC $g = E(f_1 U f_2)$
  - 
- MC $g = EG f_1$



SCOS
Secure & Correct Systems

# Model Checking Complexity

## Steps per Subformula

- MC Atomic Propositions
  - O(|S|) steps
- MC $\neg$, $\vee$ formulas
  - O(|S|) steps
- MC g = EX $f_1$
  -
- MC $g = E(f_1 U f_2)$
  -
- MC $g = EG f_1$

# Model Checking Complexity


## Steps per Subformula

- MC Atomic Propositions
  - O(|S|) steps
- MC $\neg$, $\vee$ formulas
  - O(|S|) steps
- MC g = EX $f_1$
  - Add g to label(s) iff s has a successor t such that $f_1 \in$ label(t)
  - O(|S| + |R|)
- MC $g = E(f_1 U f_2)$
  - 
- MC $g = EG f_1$

# Model Checking Complexity

## Steps per Subformula

- MC Atomic Propositions
  - O(|S|) steps
- MC $\neg$, $\vee$ formulas
  - O(|S|) steps
- MC g = EX $f_1$
  - Add g to label(s) iff s has a successor t such that $f_1 \in$ label(t)
  - O(|S| + |R|)
- MC $g = E(f_1 U f_2)$
  - O(|S| + |R|)
- MC $g = EG f_1$

# Model Checking Complexity

## Steps per Subformula

- MC $g = EGf_1$
  - Computing M′ : O (|S| + |R|)
  - Computing MSCCs using Tarjan’s algorithm: O (|S′| + |R′|)
  - Labeling all states in MSCCs: O (|S′| )
  - Backward traversal: O (|S′| + |R′|)
  - => Overall: O (|S| + |R|)

# Model Checking Complexity

## Steps per Subformula

- MC Atomic Propositions
  - O(|S|) steps
- MC $\neg$, $\vee$ formulas
  - O(|S|) steps
- MC g = EX $f_1$
  - Add g to label(s) iff s has a successor t such that $f_1 \in$ label(t)
  - O(|S| + |R|)
- MC $g = E(f_1 U f_2)$
  - O(|S| + |R|)
- MC $g = EG f_1$
  - O(|S| + |R|)

# Model Checking Complexity

- Each subformula
  - $O(|S|+|R|) = O(|M|)$
- What is the total complexity for checking f?

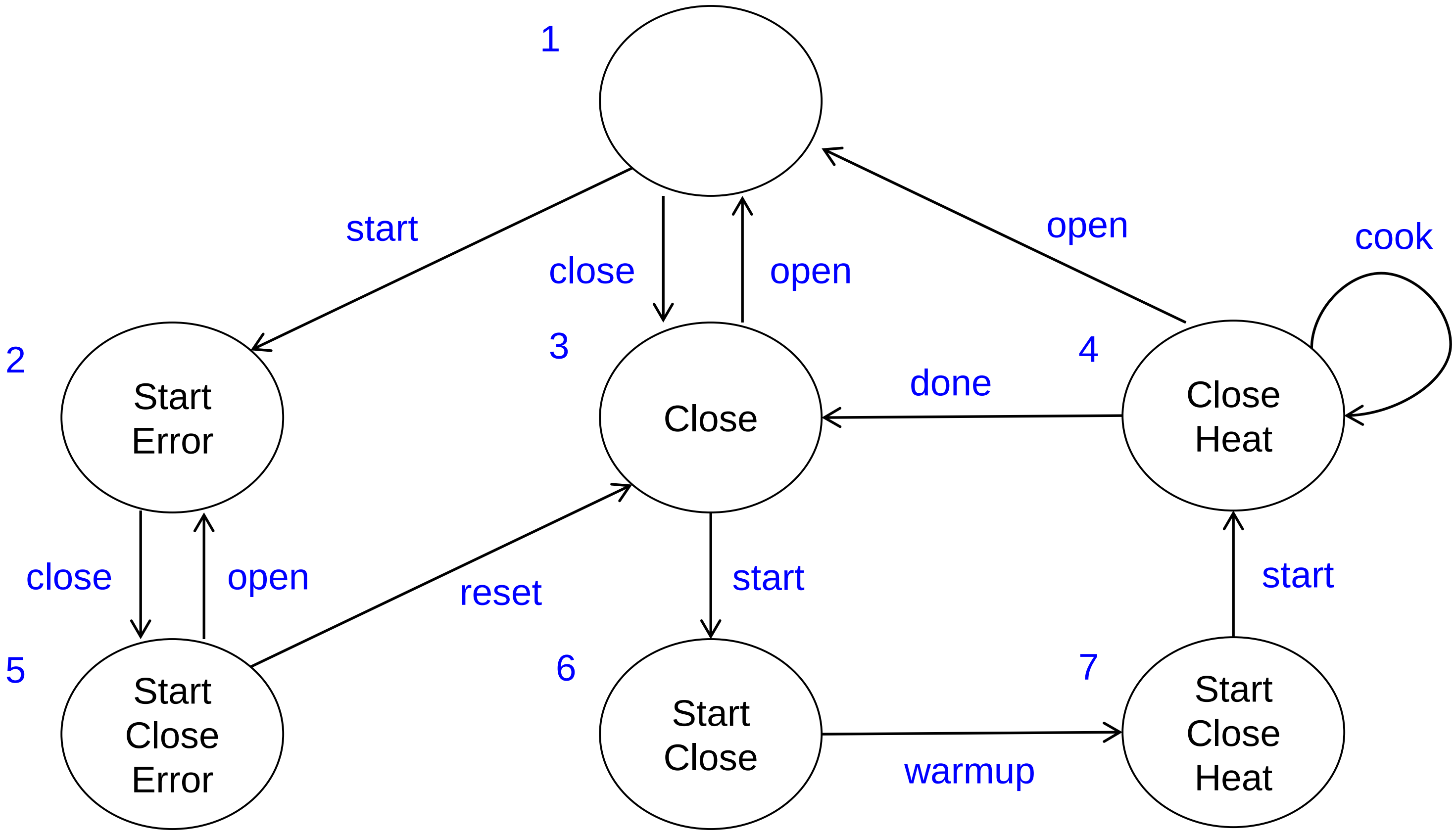
# Model Checking Complexity

- Each subformula
  - $O(|S|+ |R|) = O(|M|)$
- Number of subformulas in f:
  - $O(|f|)$
- Total
  - $O(|M| \times |f|)$

- For comparison
  - Complexity of MC for LTL and CTL* is $O( |M| \times 2^{|f|} )$

# Microwave Example

- Use the proposed algorithm to compute if $M \models f$?
  - $f := AG\,(Start \rightarrow AF\ Heat)$



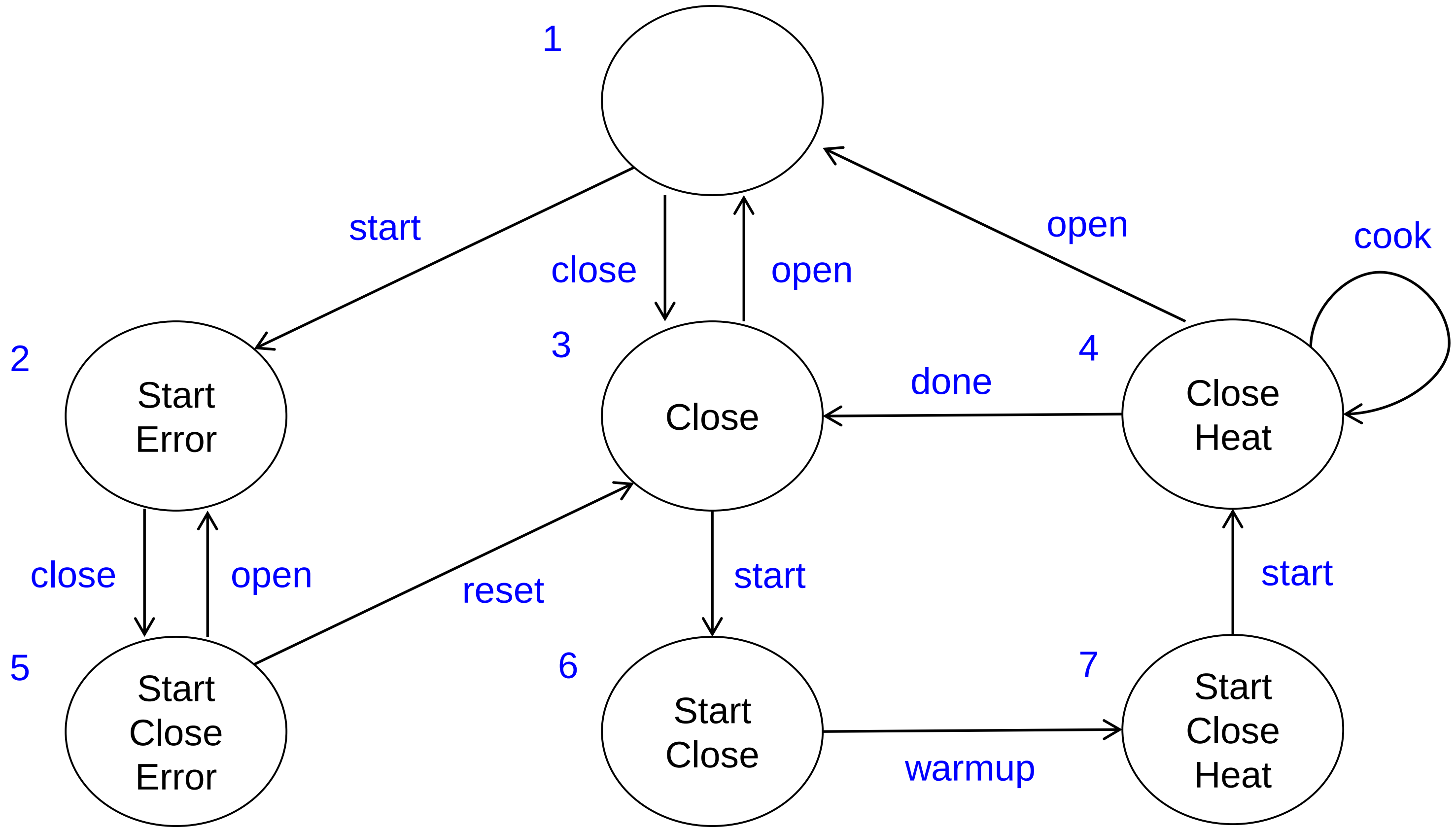
# Microwave Example

- Step 1: Rewrite the formula

  - $AG\ (Start \rightarrow AF\ Heat) \equiv$
  - $\neg EF\ (Start \wedge EG\ \neg Heat) \equiv$
  - $\neg E\ (true\ U\ (Start \wedge EG\ \neg Heat))$

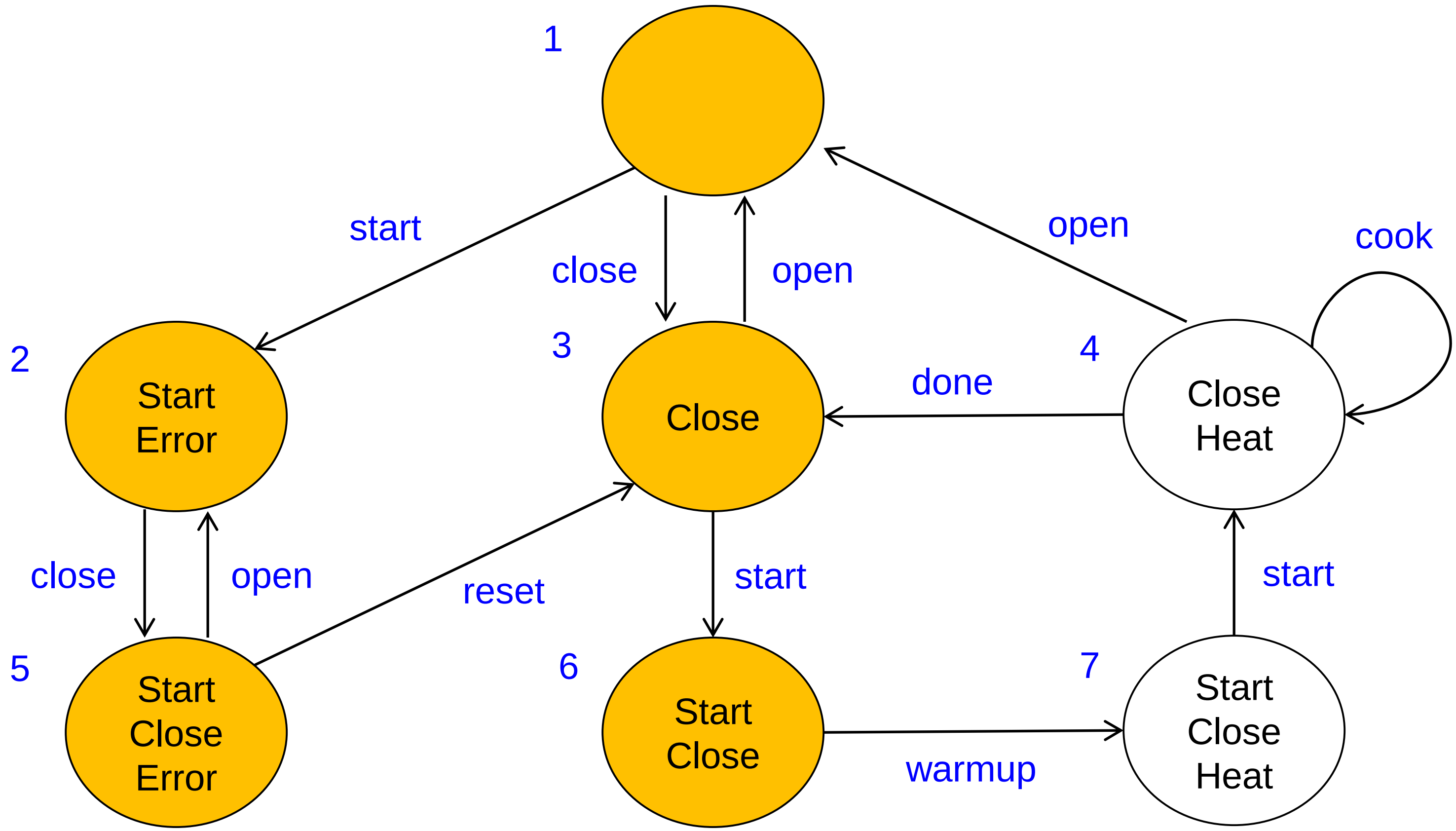
# Microwave Example

- Use the proposed algorithm to compute if $M \models f$?
  - $f := \neg E\,(\text{true}\ U\ (\text{Start} \wedge EG\ \neg \text{Heat}))$



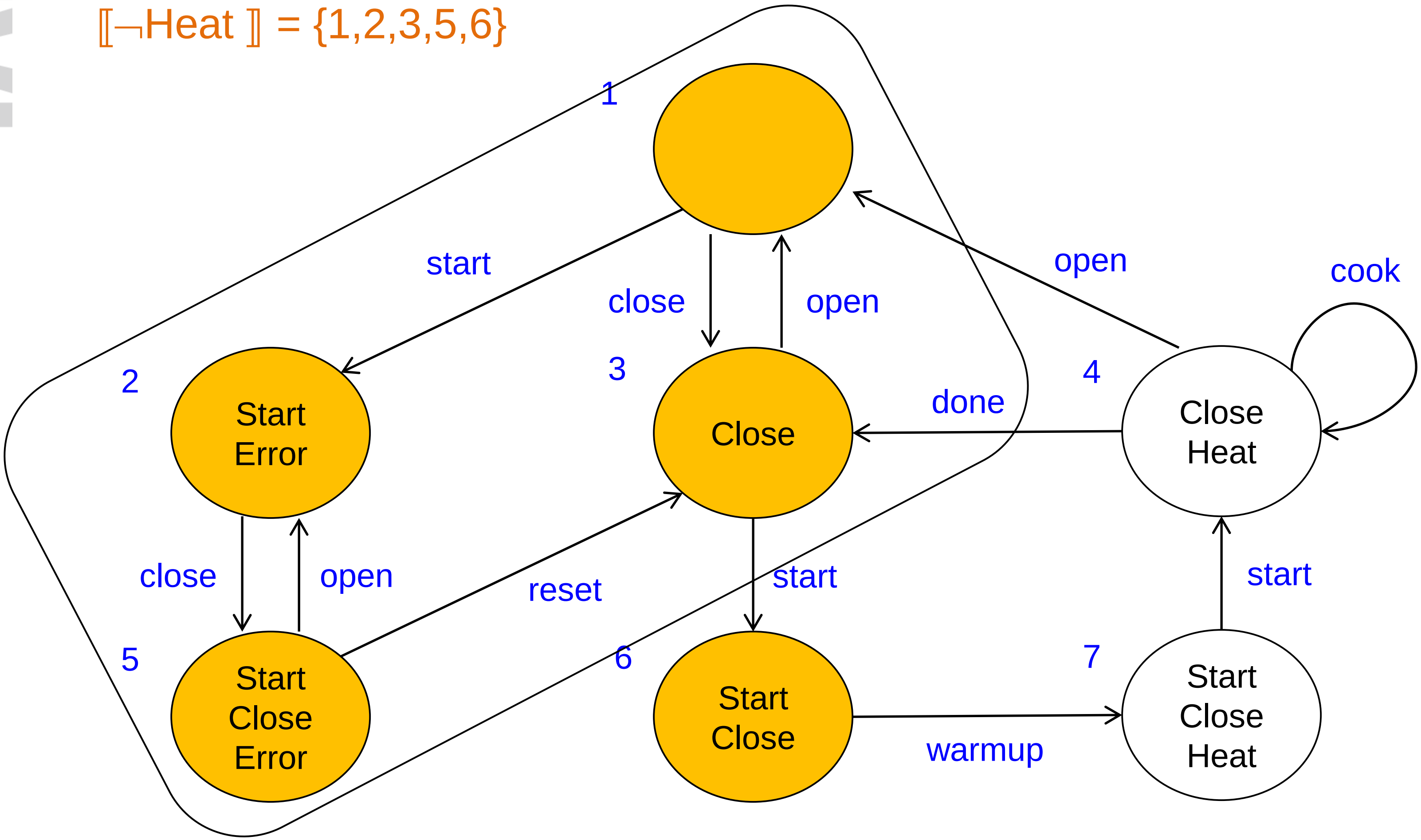
$$f := \neg E\ (true\ U\ (Start \wedge EG\ \neg Heat))$$

⟦start⟧ = {2,5,6,7}
⟦¬Heat ⟧ = {1,2,3,5,6}



# $f := \neg E\ (true\ U\ (Start \wedge EG\ \neg Heat))$

⟦start⟧ = {2,5,6,7}
⟦¬Heat ⟧ = {1,2,3,5,6}

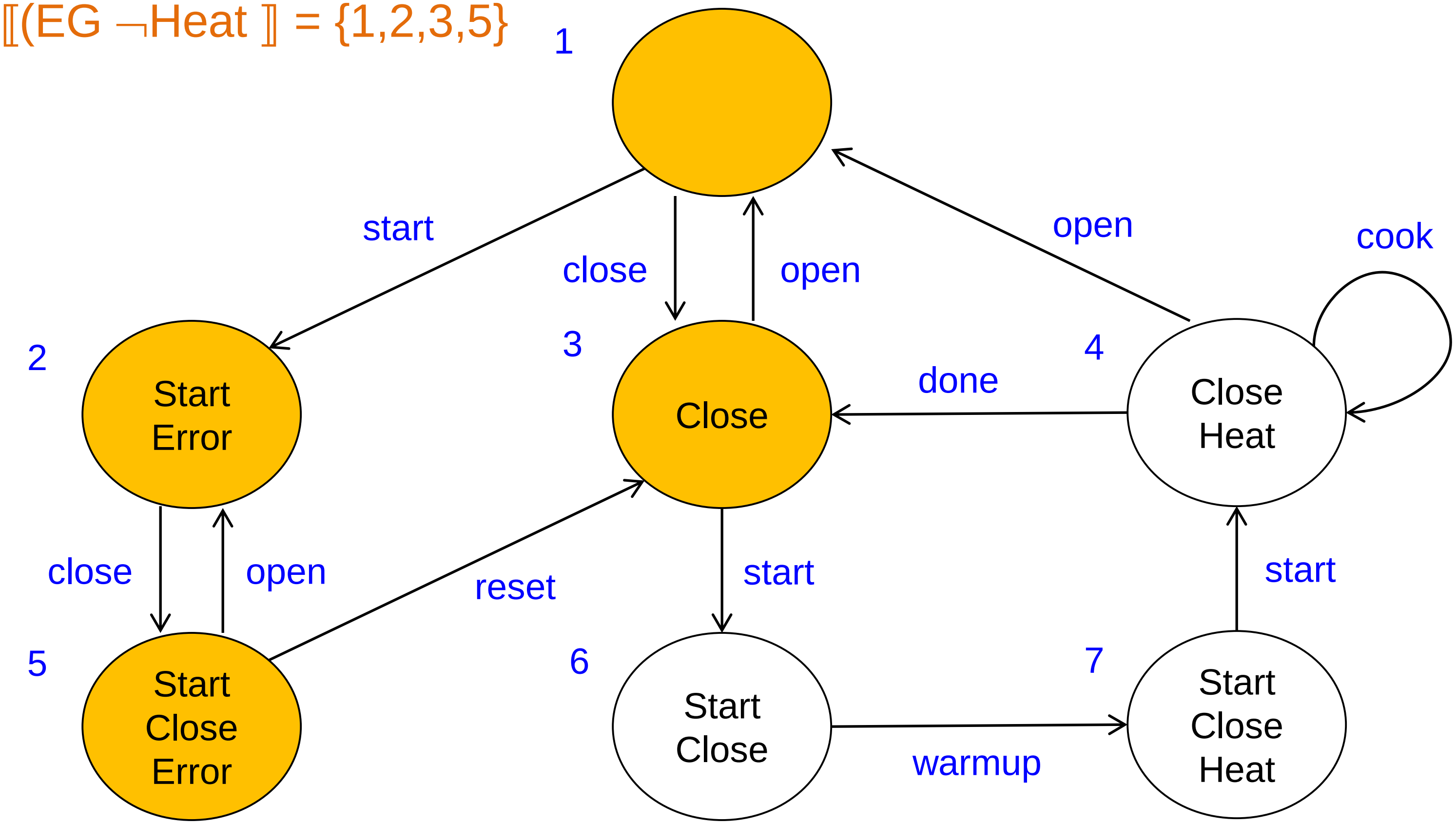




SCOS
Secure & Correct Systems

$$f := \neg E\ (true\ U\ (Start \wedge EG\ \neg Heat))$$

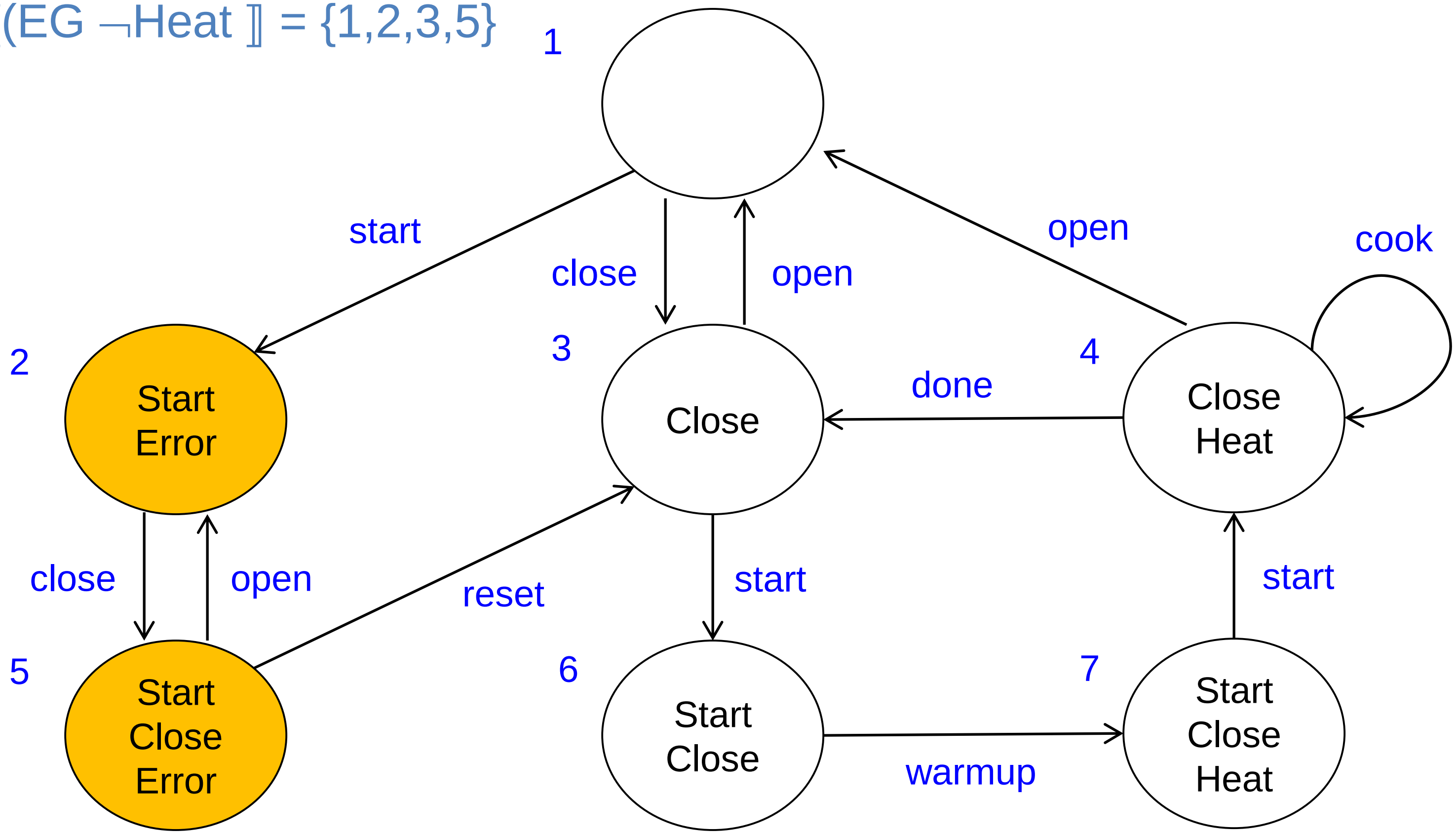
⟦start⟧ = {2,5,6,7}
⟦¬Heat ⟧ = {1,2,3,5,6}
⟦(EG ¬Heat ⟧ = {1,2,3,5}



# $f := \neg E\ (true\ U\ (Start \wedge EG\ \neg Heat))$

⟦start⟧ = {2,5,6,7}
⟦¬Heat ⟧ = {1,2,3,5,6}
⟦(EG ¬Heat ⟧ = {1,2,3,5}

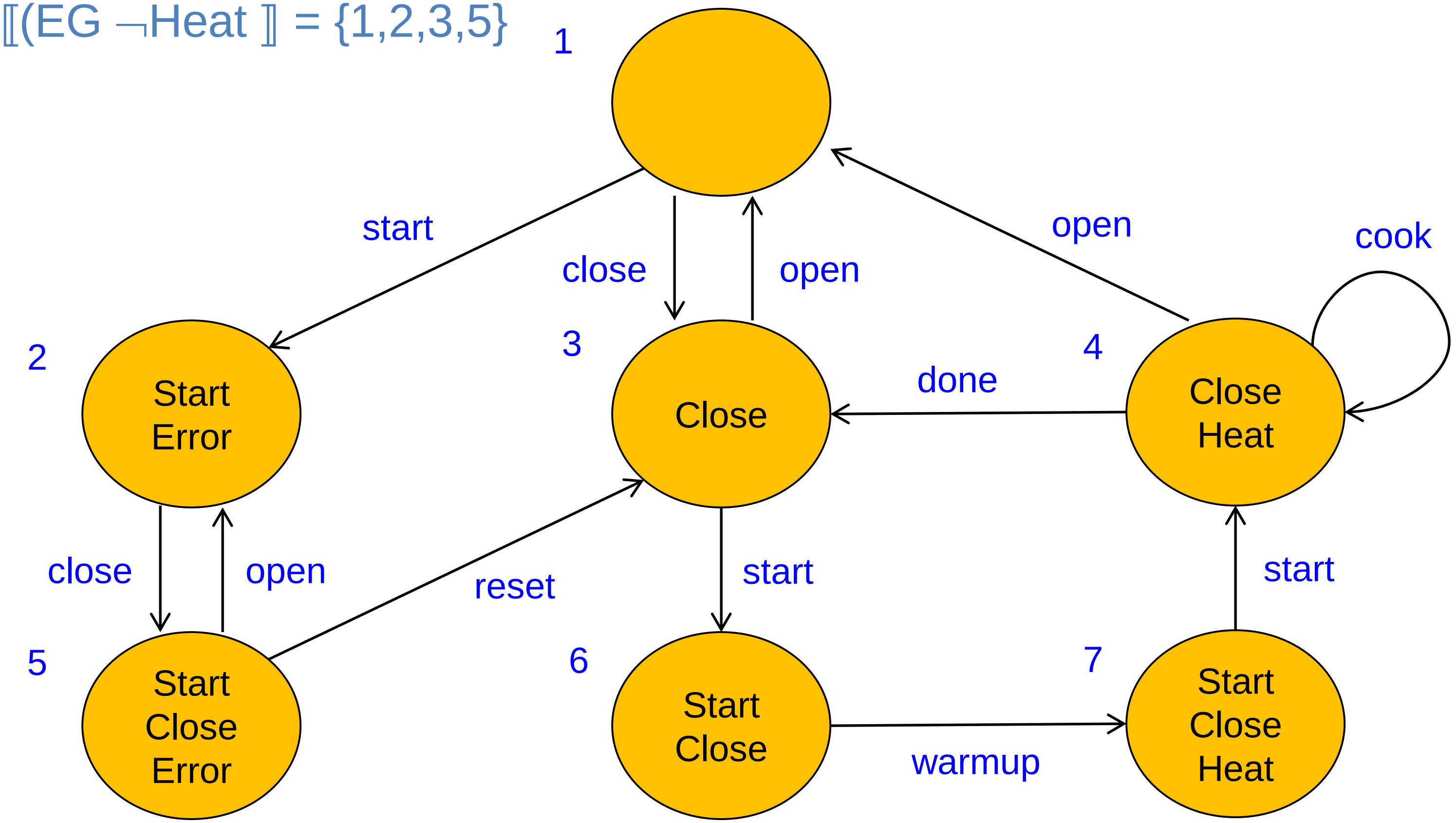
⟦ Start ∧ EG ¬Heat ⟧ = {2, 5}



# $f := \neg E\ (true\ U\ (Start \wedge EG\ \neg Heat))$

⟦start⟧ = {2,5,6,7}
⟦¬Heat ⟧ = {1,2,3,5,6}
⟦(EG ¬Heat ⟧ = {1,2,3,5}

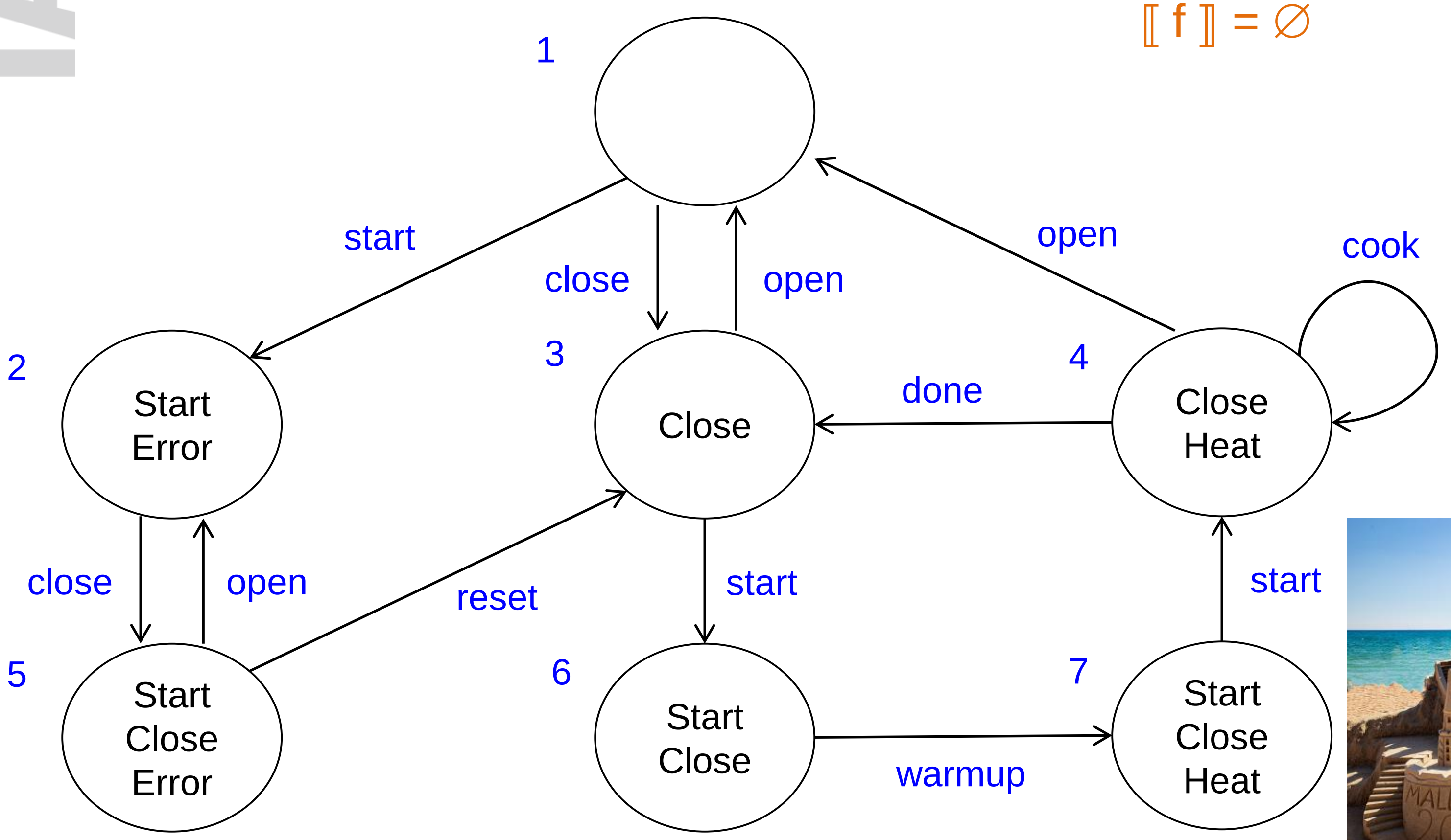
⟦ Start ∧ EG ¬Heat ⟧ = {2, 5}
⟦ EU ⟧ = {1,2,3,4,5,6,7}



$$f := \neg E\ (true\ U\ (Start \wedge EG\ \neg Heat))$$

⟦start⟧ = {2,5,6,7}
⟦¬Heat ⟧ = {1,2,3,5,6}
⟦(EG ¬Heat ⟧ = {1,2,3,5}

⟦ Start ∧ EG ¬Heat ⟧ = {2, 5}
⟦ EU ⟧ = {1,2,3,4,5,6,7}
⟦ f ⟧ = ∅





THANK YOU!