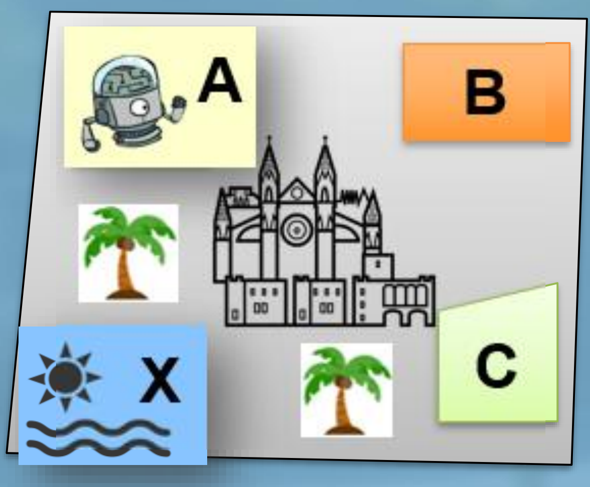
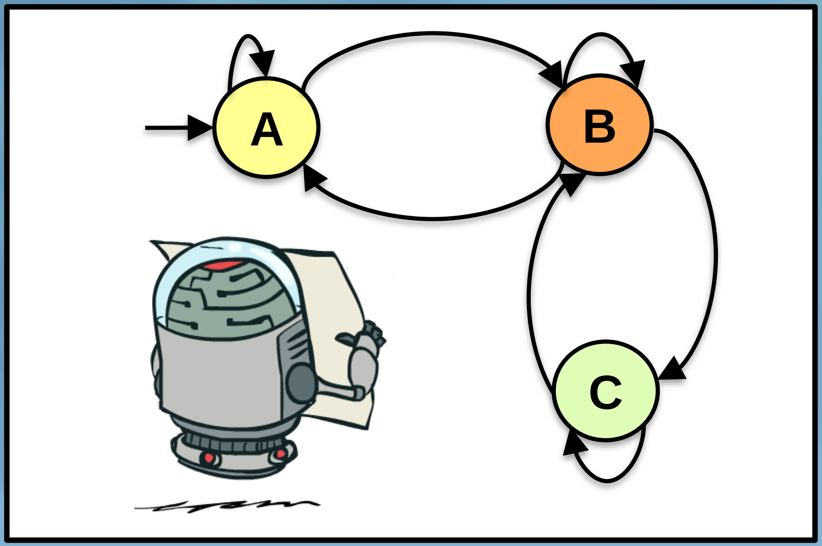
Graz University of Technology
Institute for Applied Information
Processing and Communications

CTL Model Checking

Bettina Könighofer

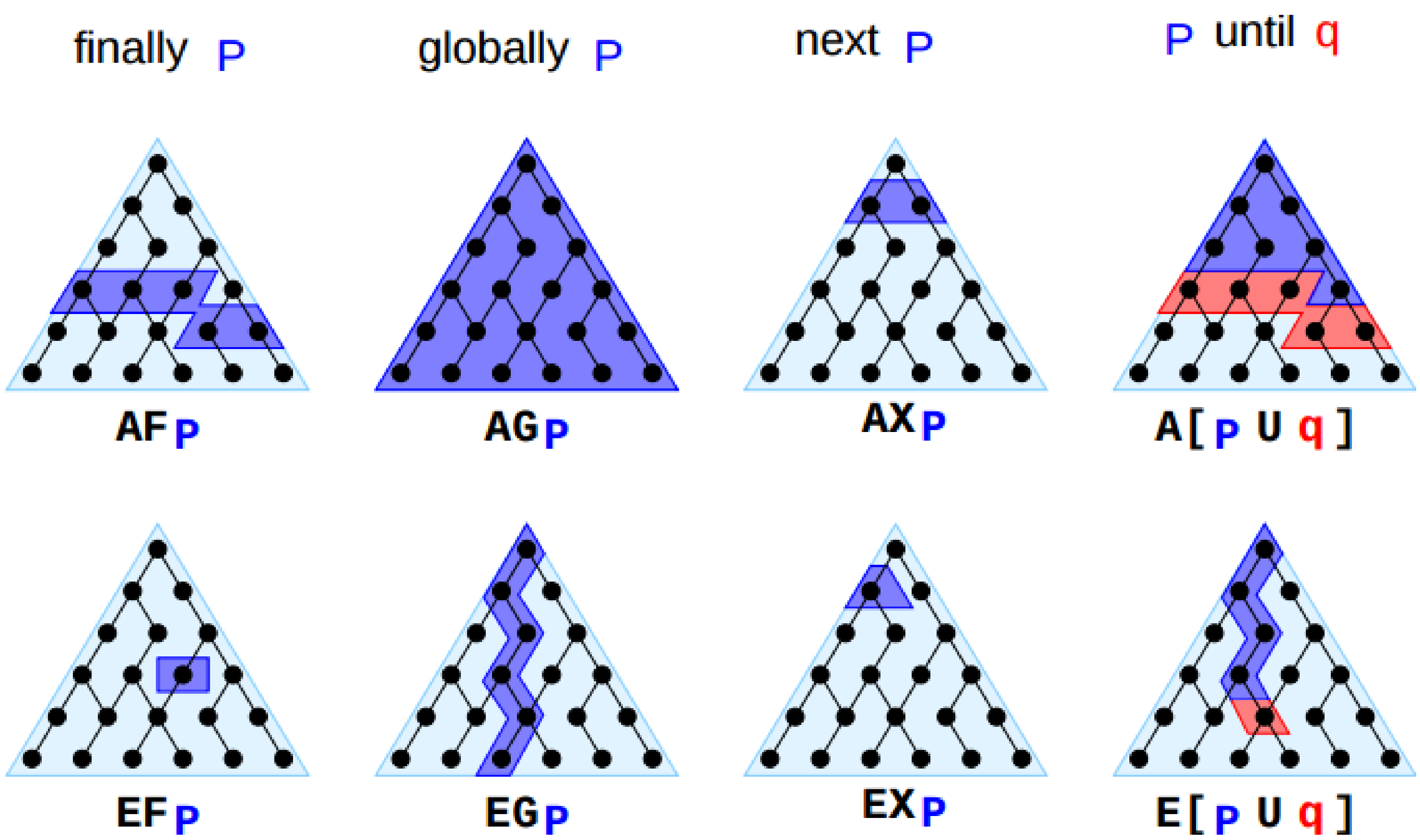




Model Checking SS21

May 5th 2021

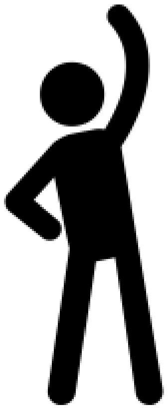
Warm-Up CTL



SCOS Secure & Correct Systems

Warm-Up CTL

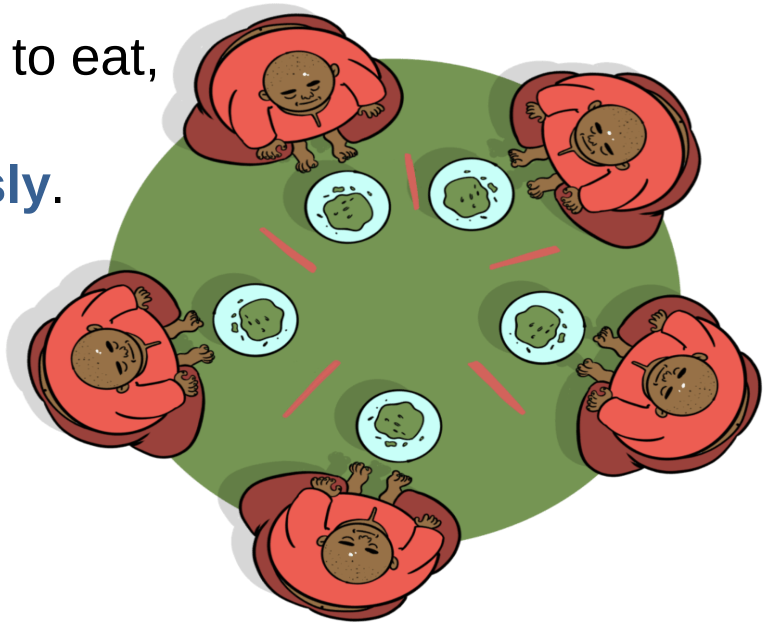
The Dining-Philosophers Verification-Problem



There are n philosophers sitting at a round table.

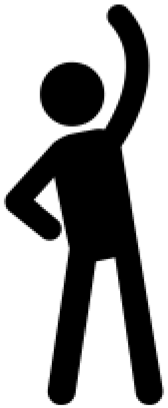
There is one chopstick between each pair of adjacent philosophers.

Each philosopher needs two chopsticks to eat,
Therefore, adjacent
philosophers **cannot eat simultaneously**.



Warm-Up CTL

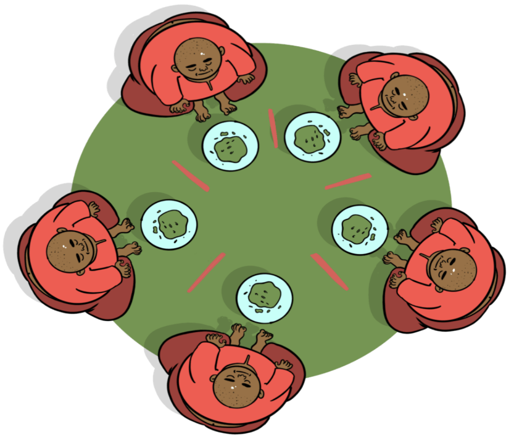
The Dining-Philosophers Verification-Problem



Variables:

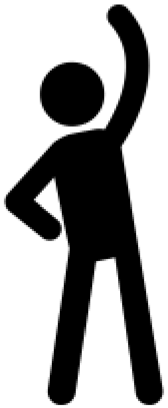
- h_i … philosopher i is hungry
- e_i … philosopher i is eating

- Translate into CTL:
  - "Every hungry philosopher eats eventually"



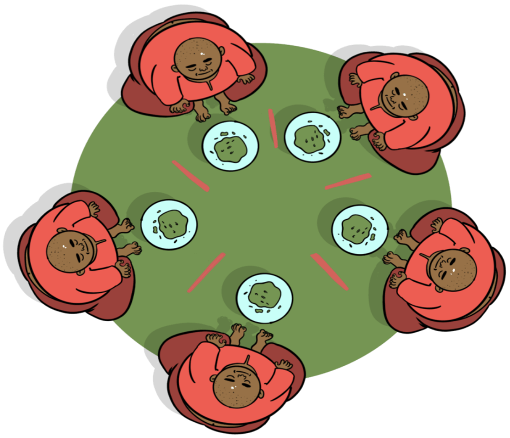
Warm-Up CTL

The Dining-Philosophers Verification-Problem



Variables:

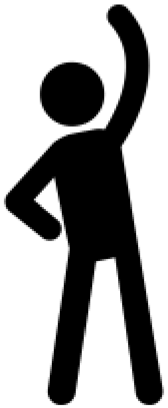
- h_i … philosopher i is hungry
- e_i … philosopher i is eating



- Translate into CTL:
  - "Every hungry philosopher eats eventually"
  - $AG\ (h_1 \ \rightarrow AF\ e_1) \wedge$
  - $AG\ (h_2 \ \rightarrow AF\ e_2) \wedge \cdots .$

Warm-Up CTL

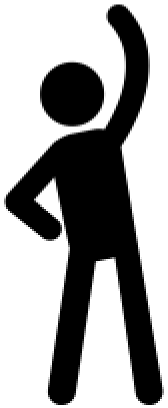
The Dining-Philosophers Verification-Problem



- Translate into CTL:
  - "An eating philosopher eventually loses her appetite".
  - "An eating philosopher that is still hungry will continue to eat"
  - "An eating philosopher prevents her neighbours from eating"
  - "There exists a scenario in which philosopher 2 starves"

Warm-Up CTL

The Dining-Philosophers Verification-Problem

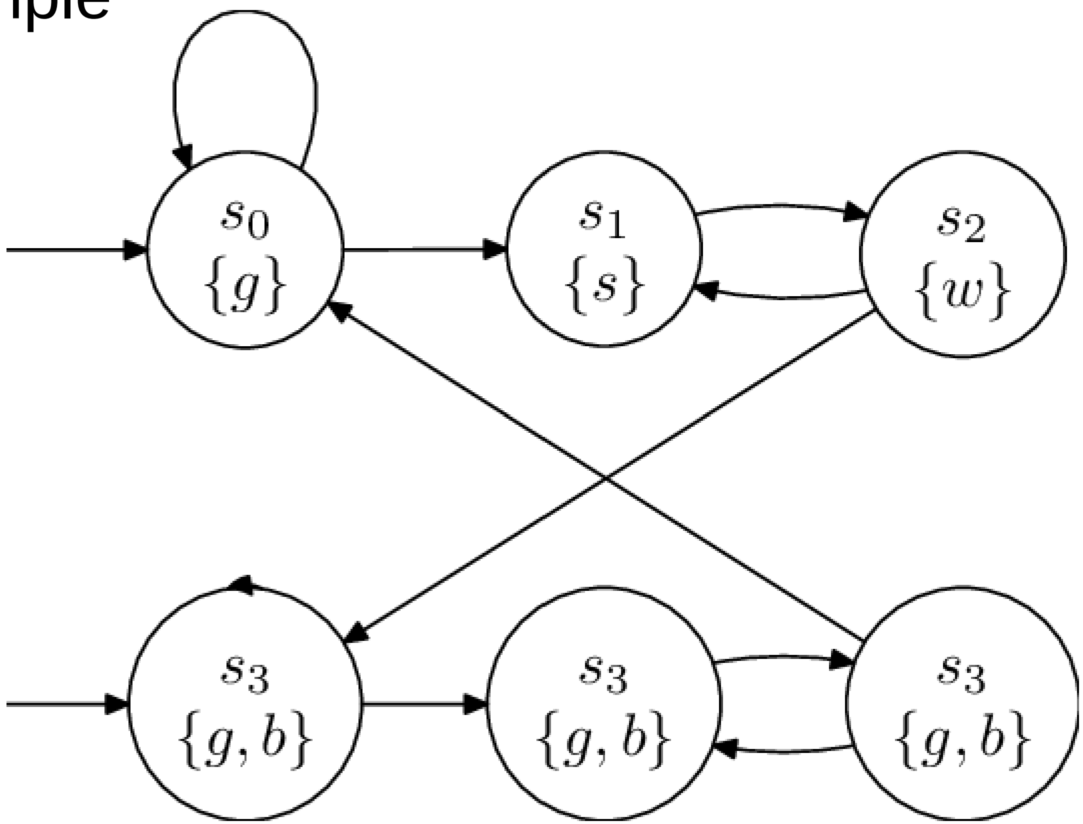


Translate into CTL:

- "An eating philosopher eventually loses her appetite".
  - $AG(e_i \rightarrow AF \neg h_i)$
- "An eating philosopher that is still hungry will continue to eat"
  - $AG(e_i \wedge h_i \rightarrow AXe_i)$
- "An eating philosopher prevents her neighbours from eating"
  - $AG(e_i \rightarrow (\neg e_{i-1} \wedge \neg e_{i+1})$
- "There exists a scenario in which philosopher 2 starves"
  - $\mathrm{E}G(h_i \wedge \neg e_i)$

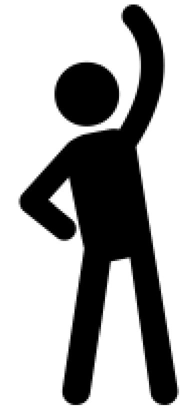
Warm-Up Kripke Structure

Example



Warm-Up Kripke Structure

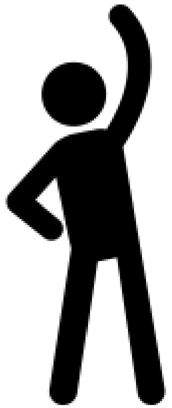
Mutual Exclusion



- 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

Warm-Up Kripke Structure

Mutual Exclusion



- Each process runs the following program:

P_i :: while (true) {

Atomic action →

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; }

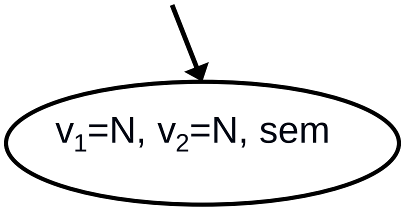
}

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

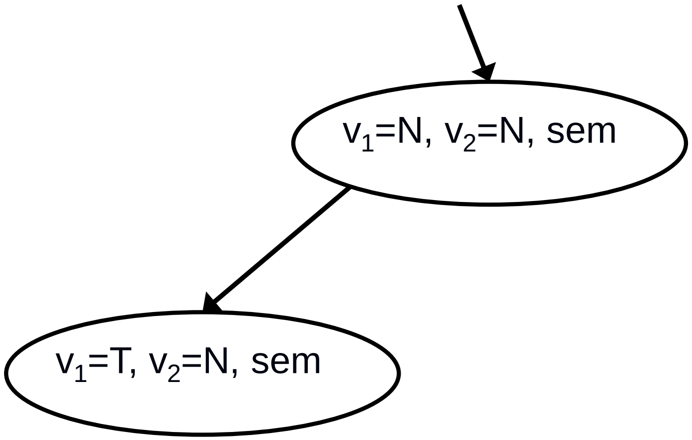
ToDo: Draw the **Kipke Structure** that represents the interleaving execution

SCOS Secure & Correct Systems

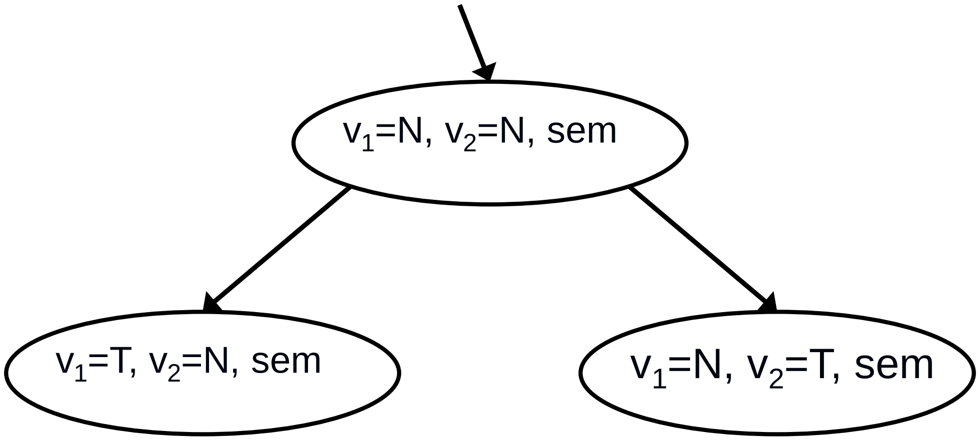
Warm-Up Example: Mutual Exclusion



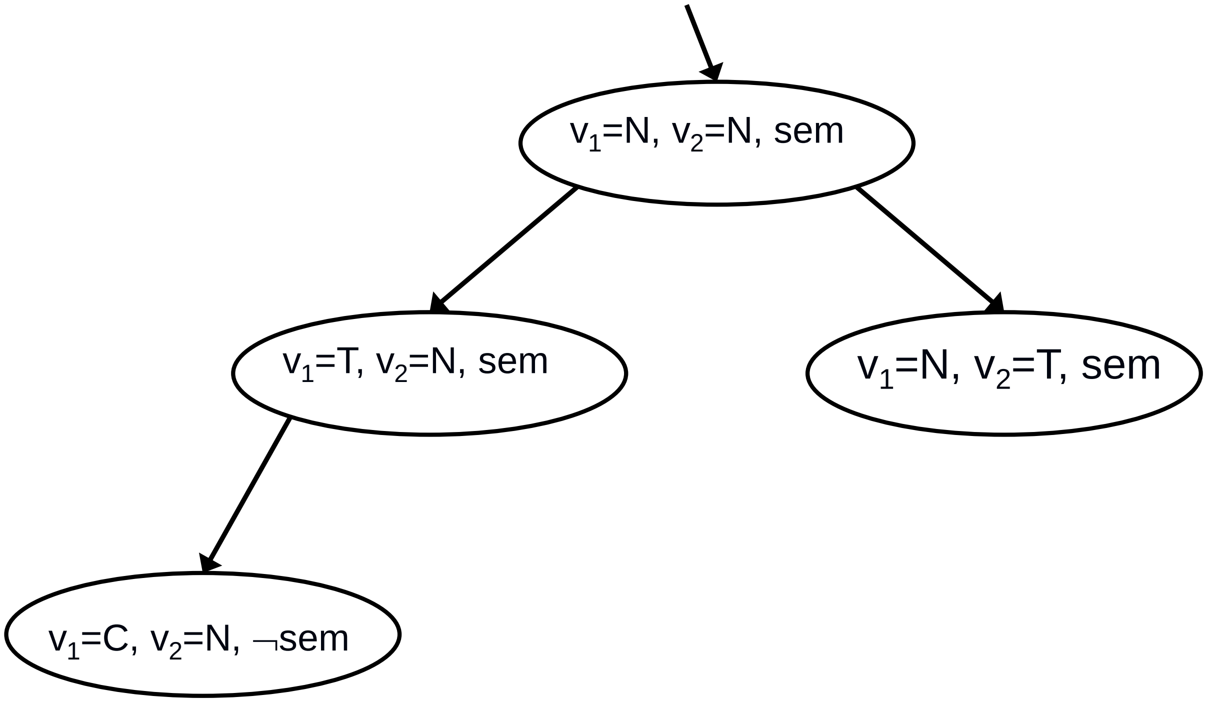
Warm-Up Example: Mutual Exclusion



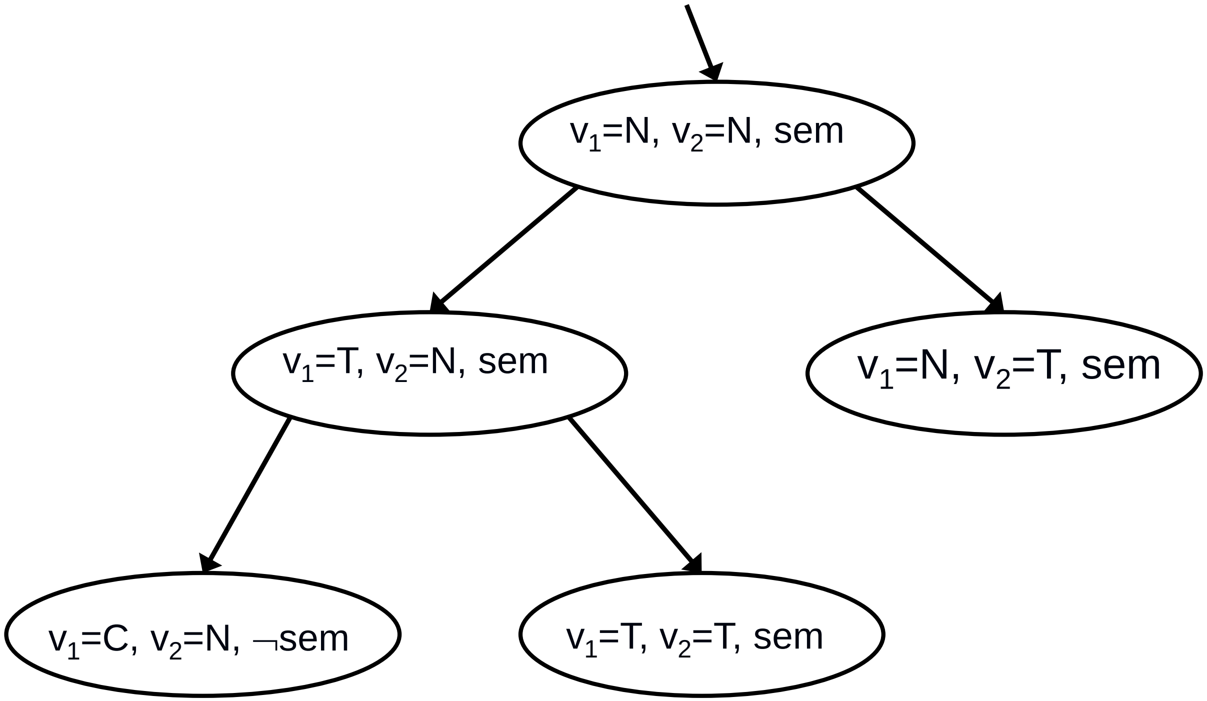
Warm-Up Example: Mutual Exclusion



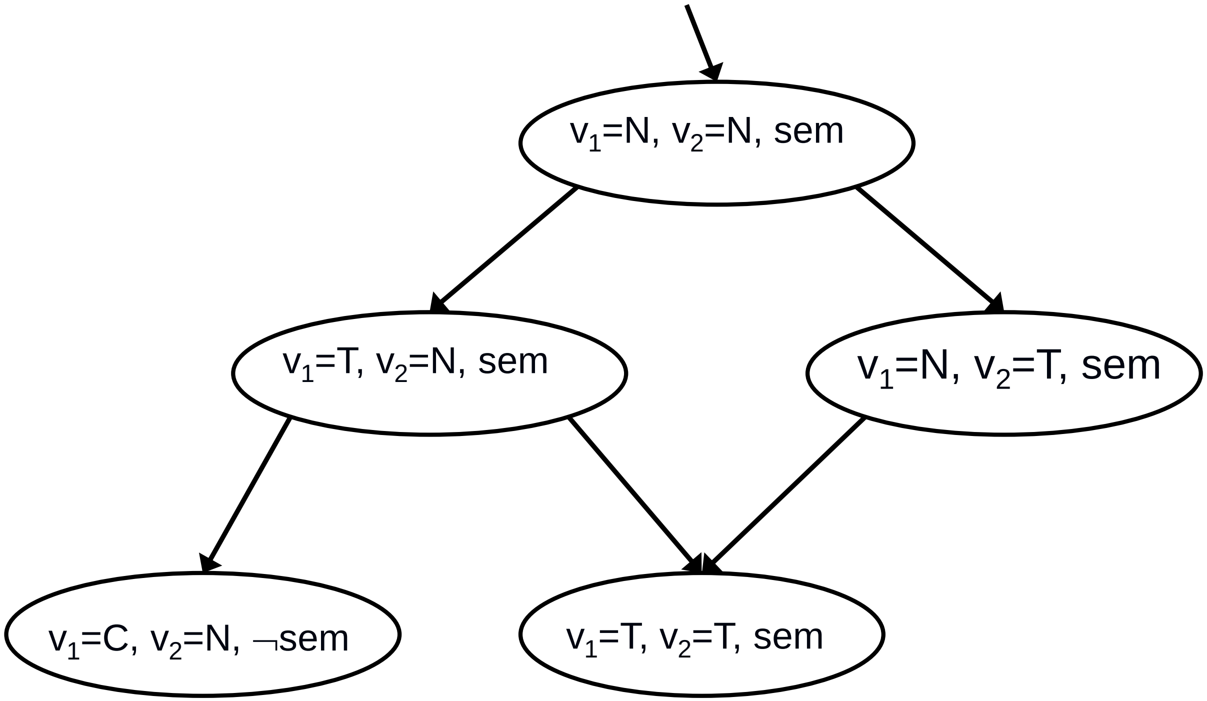
Warm-Up Example: Mutual Exclusion



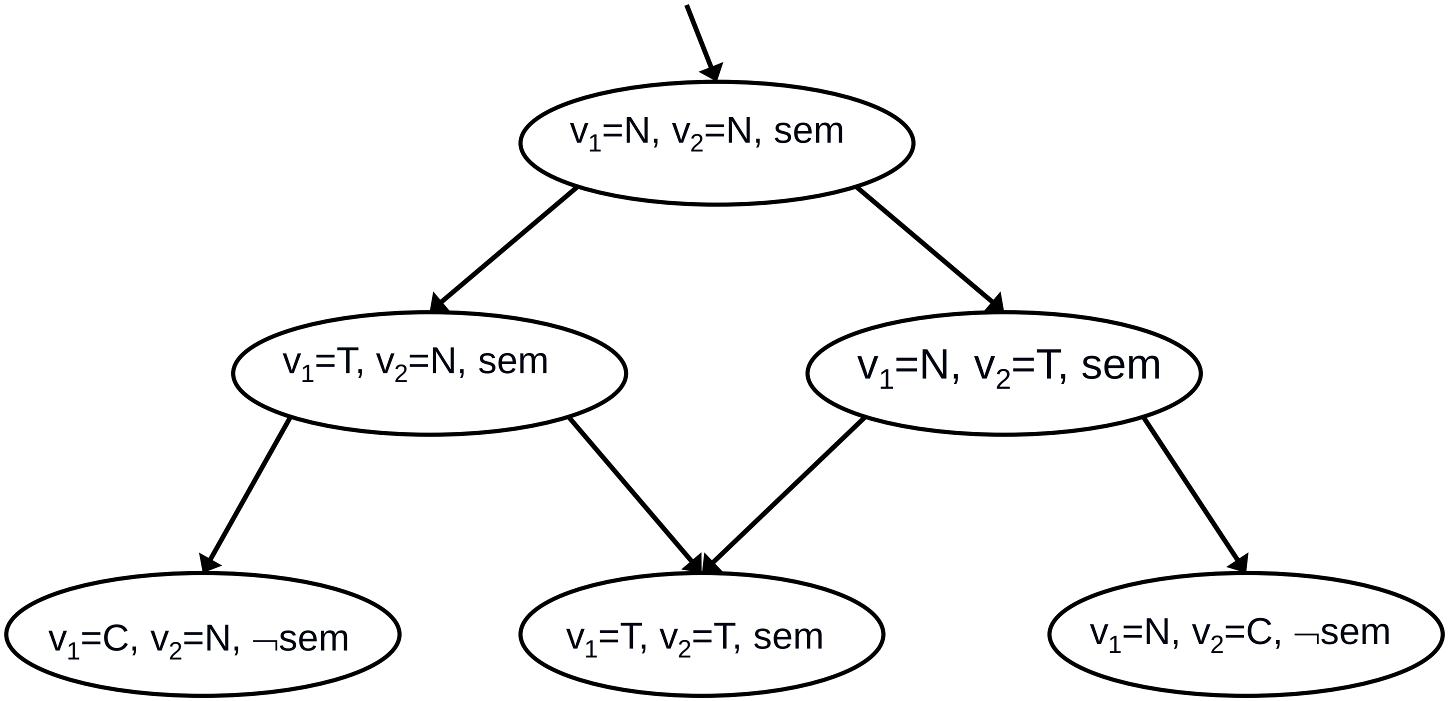
Warm-Up Example: Mutual Exclusion



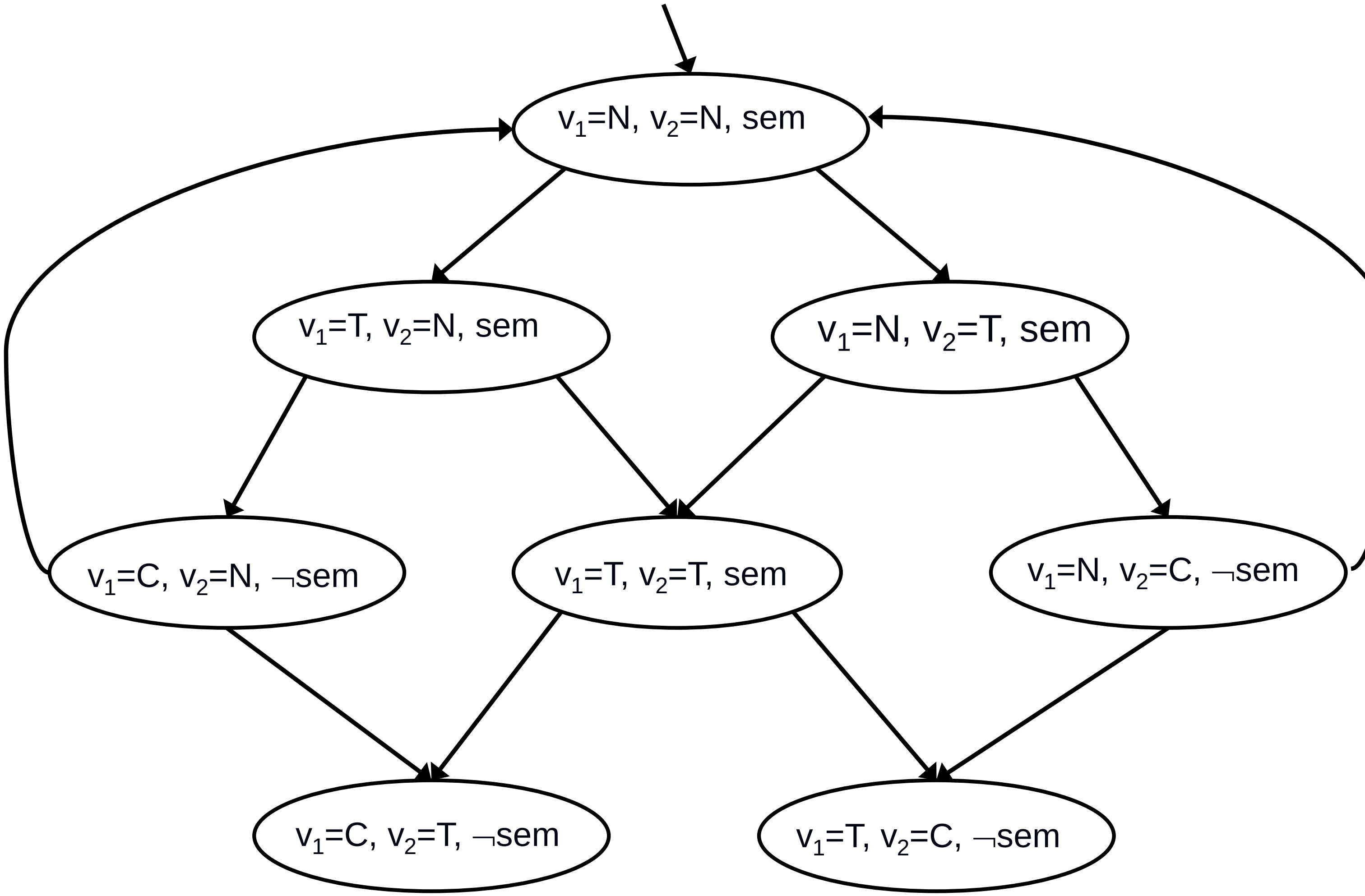
Warm-Up Example: Mutual Exclusion



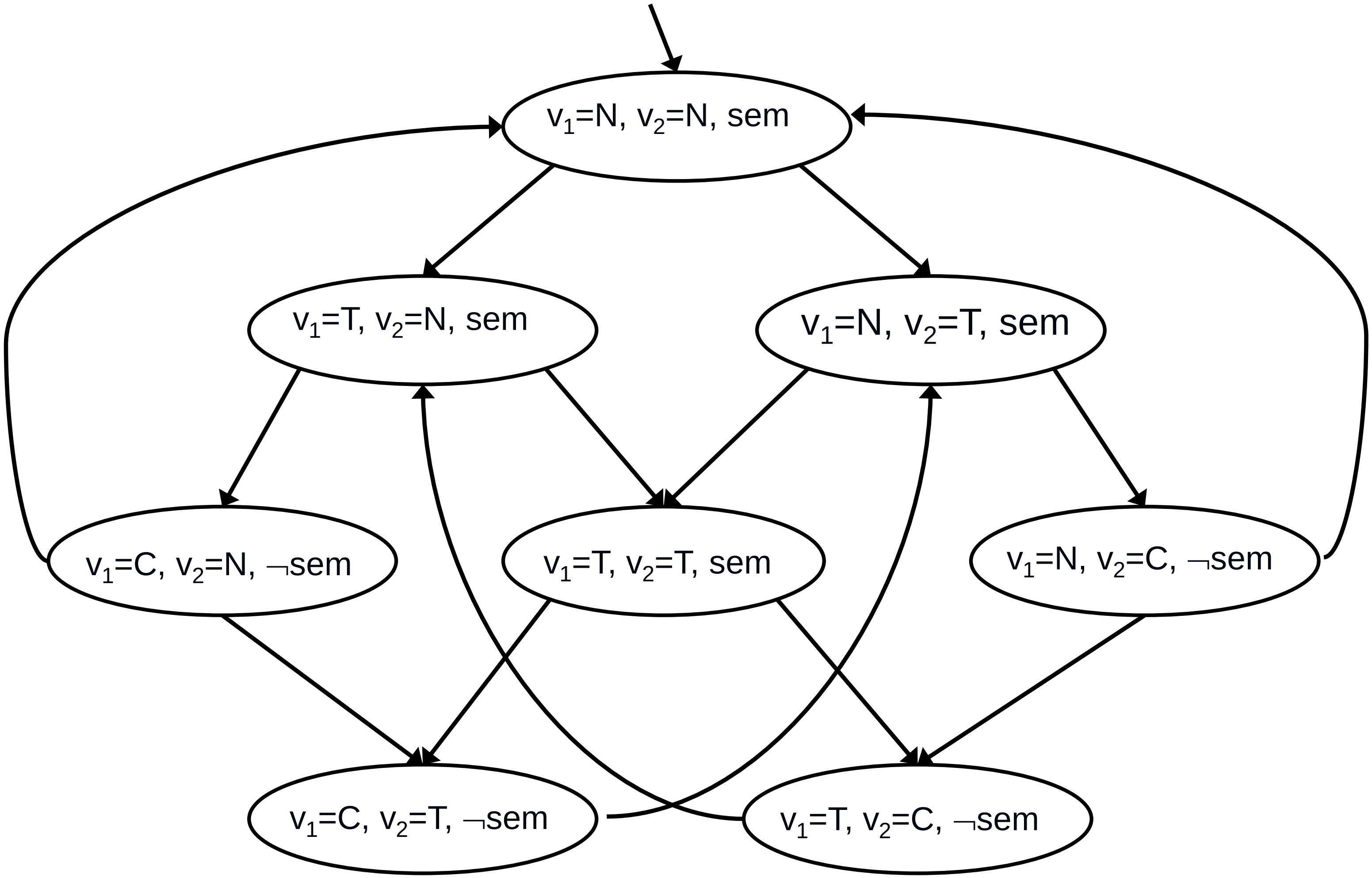
Warm-Up Example: Mutual Exclusion



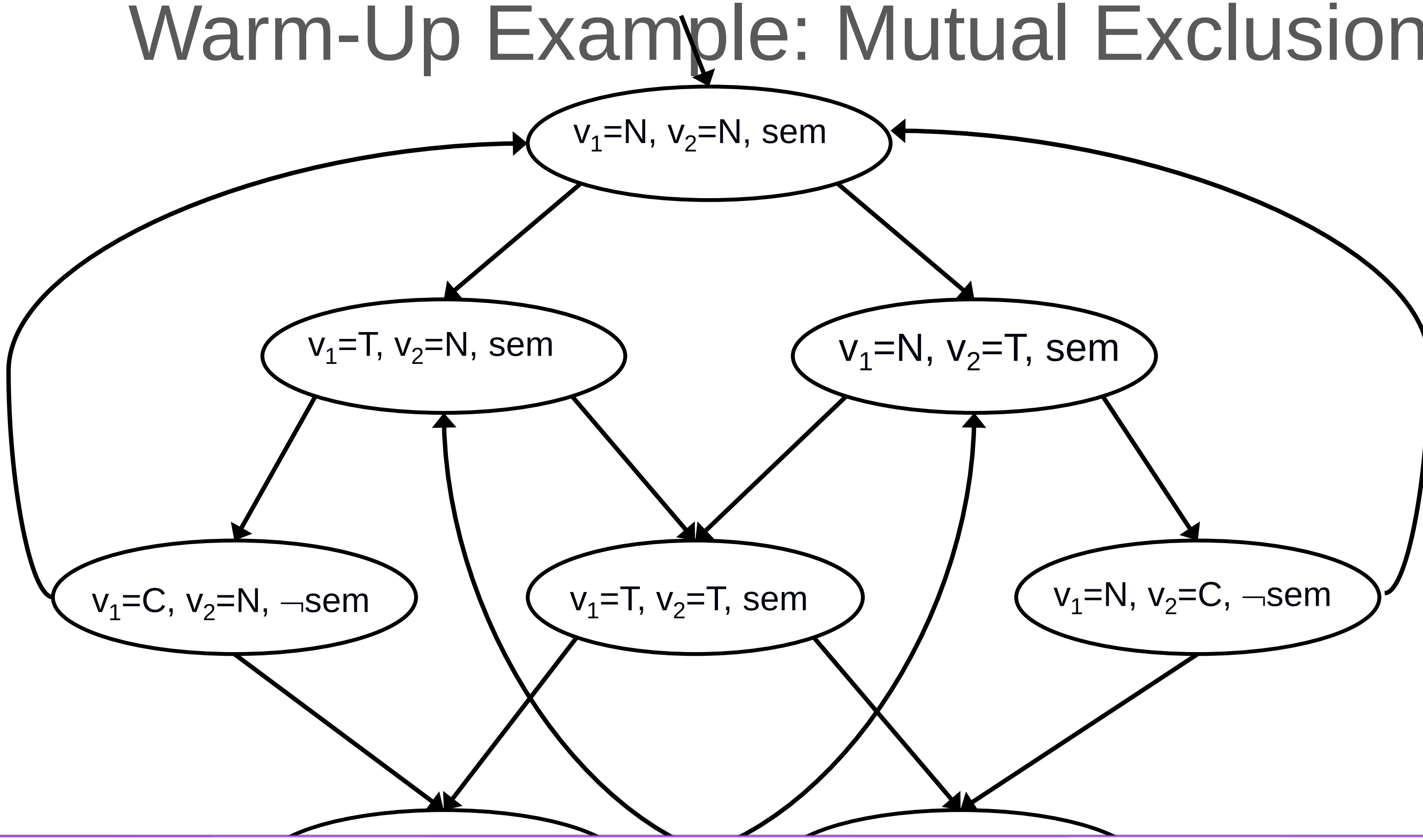
Warm-Up Example: Mutual Exclusion



Warm-Up Example: Mutual Exclusion

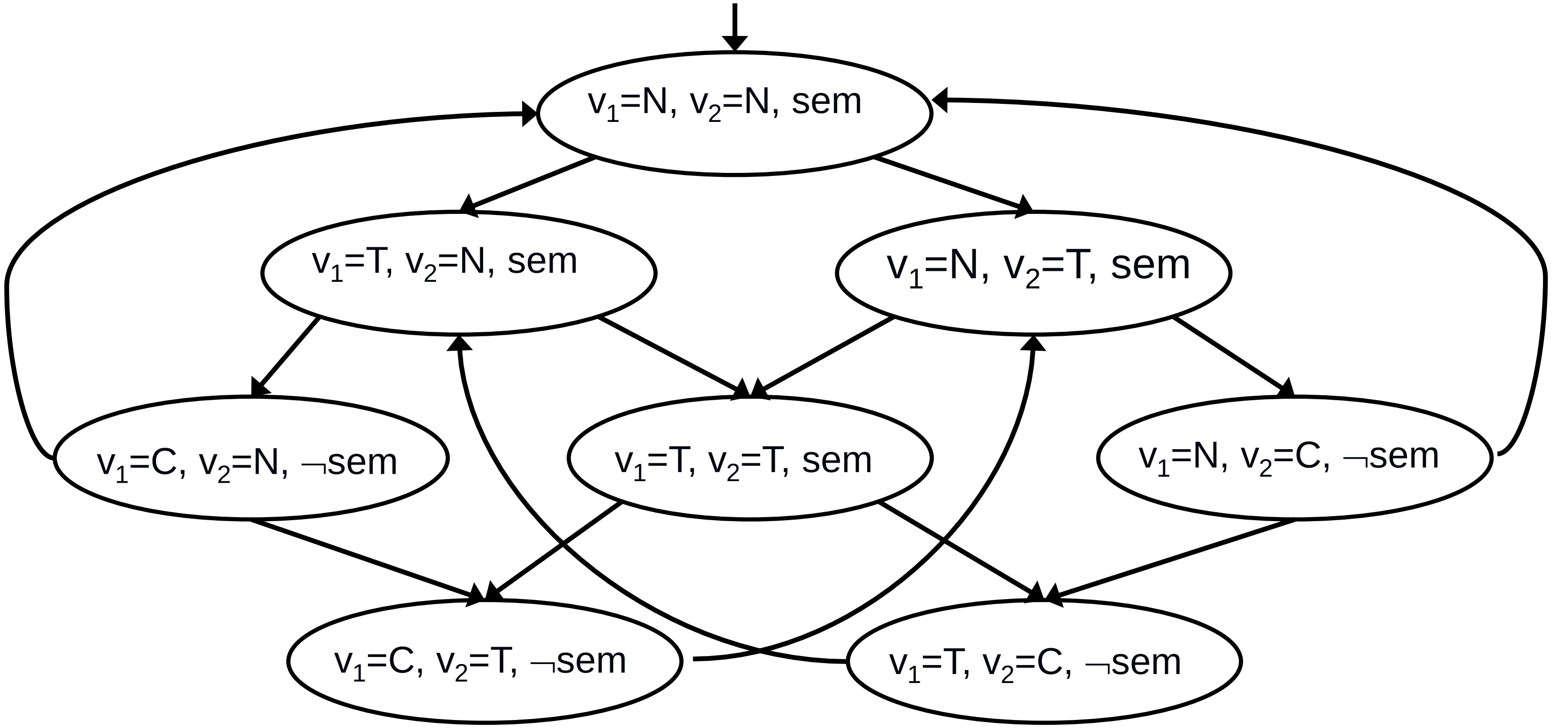


Warm-Up Example: Mutual Exclusion

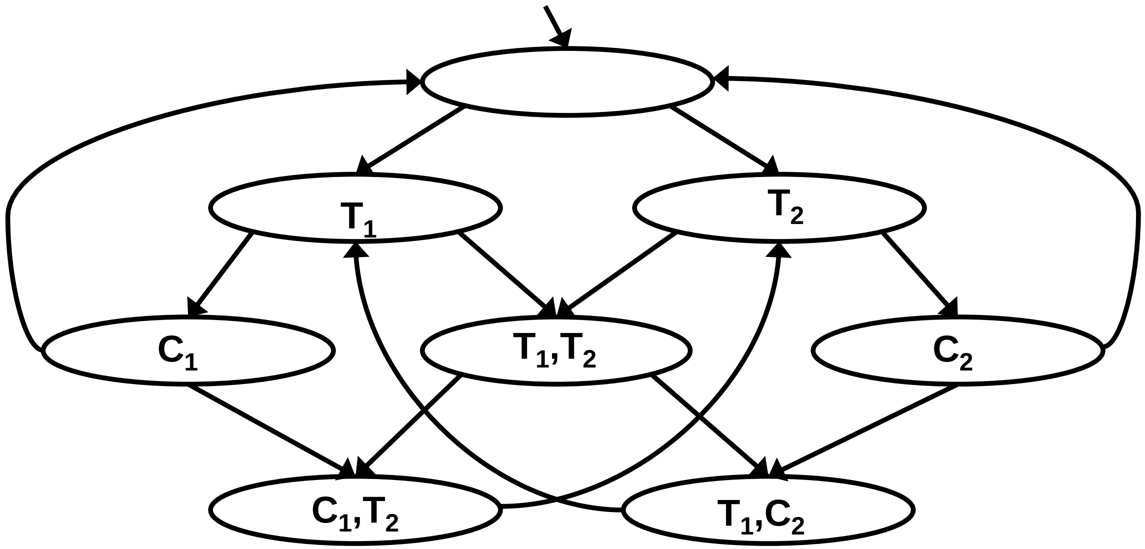


- **Today – Check Properties on Kripke Structures:** E.g.: Is there an execution trace s.t. P1 and P2 are both in the critical section?

Warm-Up Example: Mutual Exclusion

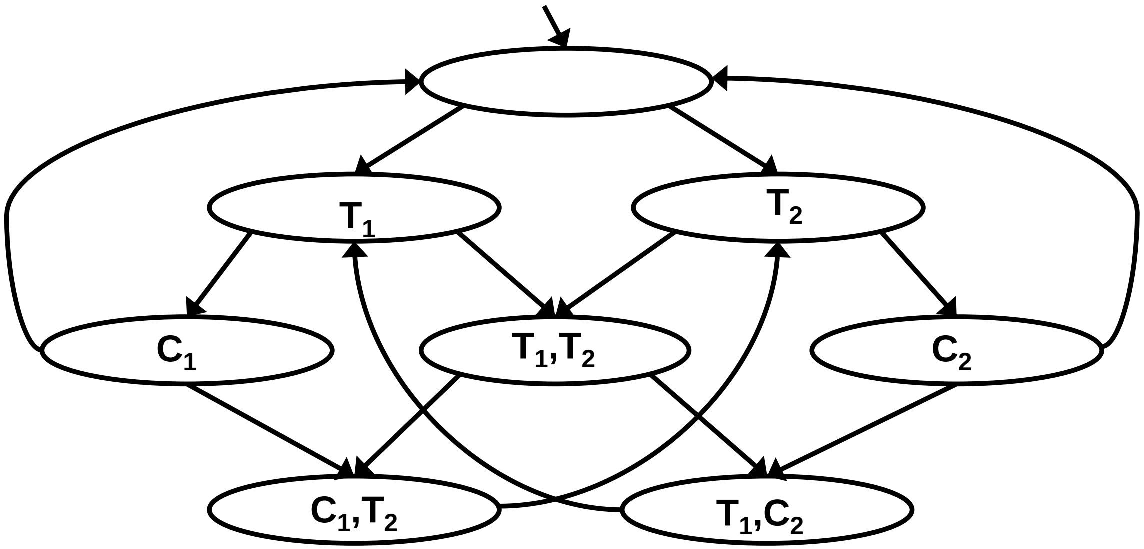


Warm-Up Example: Mutual Exclusion



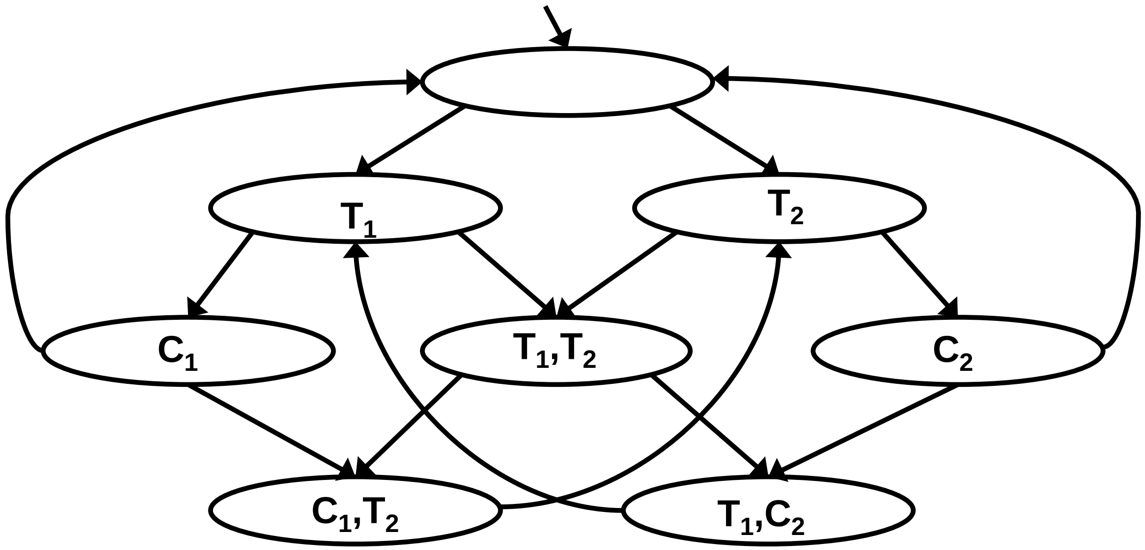
- We define atomic propositions: $AP=\{C_1,C_2,T_1,T_2\}$
- A state is labeled with T_i if $v_i=T$
- A state is labeled with C_i if $v_i=C$

Warm-Up Example: Mutual Exclusion



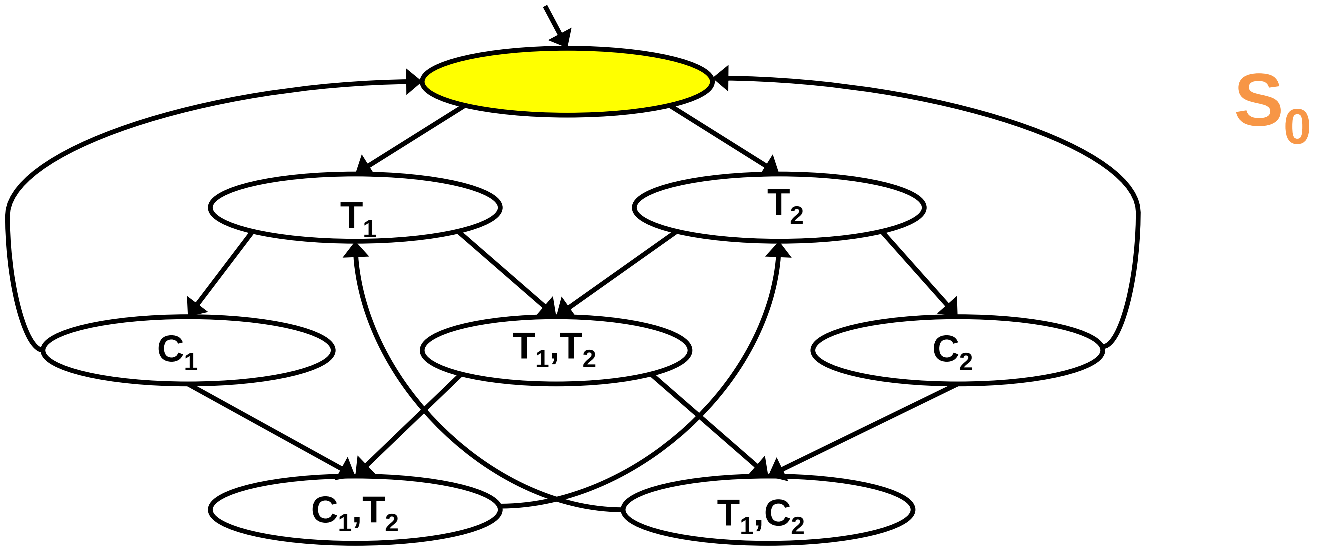
- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(C_1 \wedge C_2)$
  - Compute $[\![f]\!]_M = \{\, s \in S \mid M,s \models f \,\}$ and check $S_0 \subseteq [\![f]\!]_M$

Warm-Up Example: Mutual Exclusion



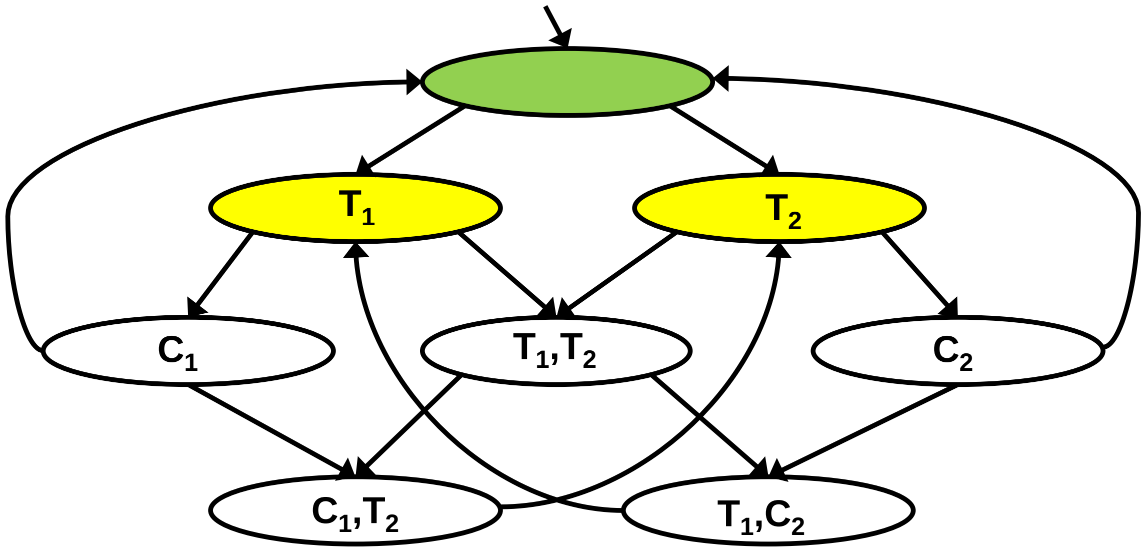
- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(C_1 \wedge C_2)$
- Yes, if $\neg(C_1 \wedge C_2)$ holds in all reachable states
- $S_i \equiv$ reachable states from an initial state after **i** steps

Warm-Up Example: Mutual Exclusion



- Does it hold that M ⊨ f?
  - Property 1: f := **AG**$\neg(C_1 \wedge C_2)$
- Yes, if $\neg(C_1 \wedge C_2)$ holds in all reachable states
- S_i ≡ reachable states from an initial state after **i** steps

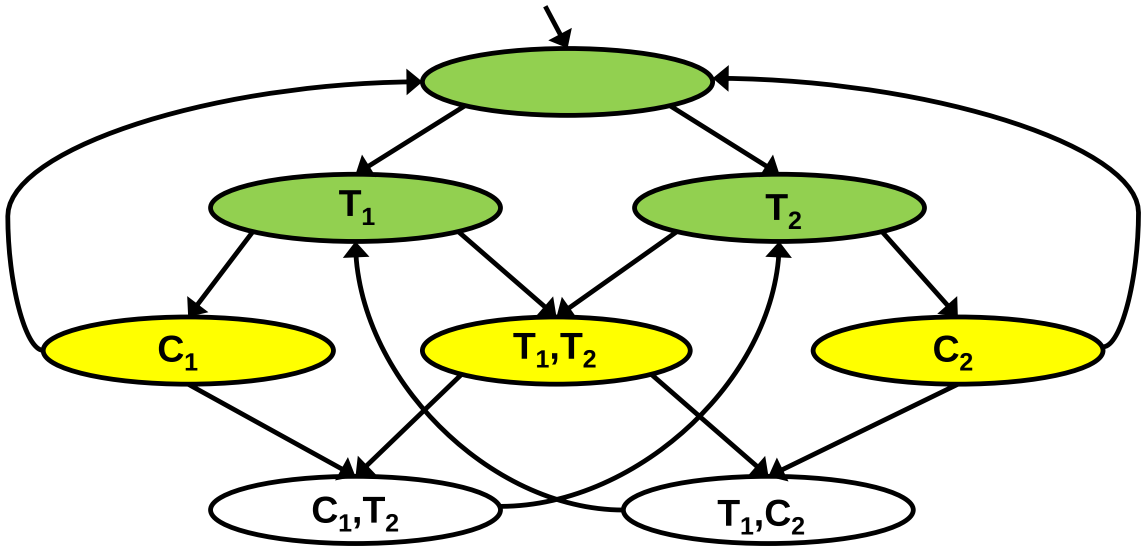
Warm-Up Example: Mutual Exclusion



S_1

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

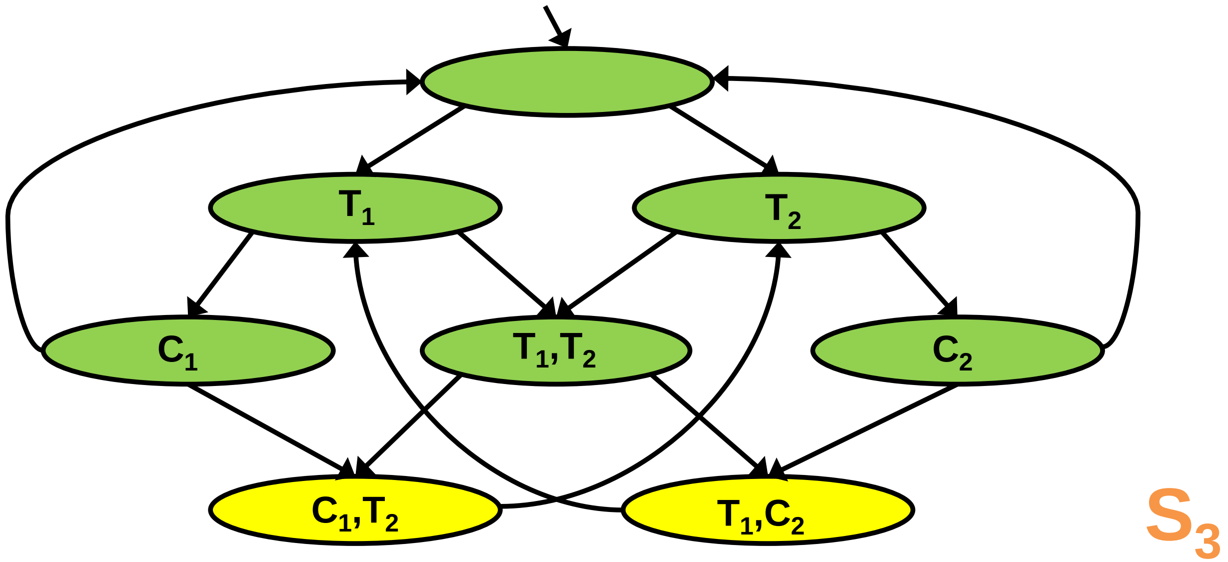
Warm-Up Example: Mutual Exclusion



S_2

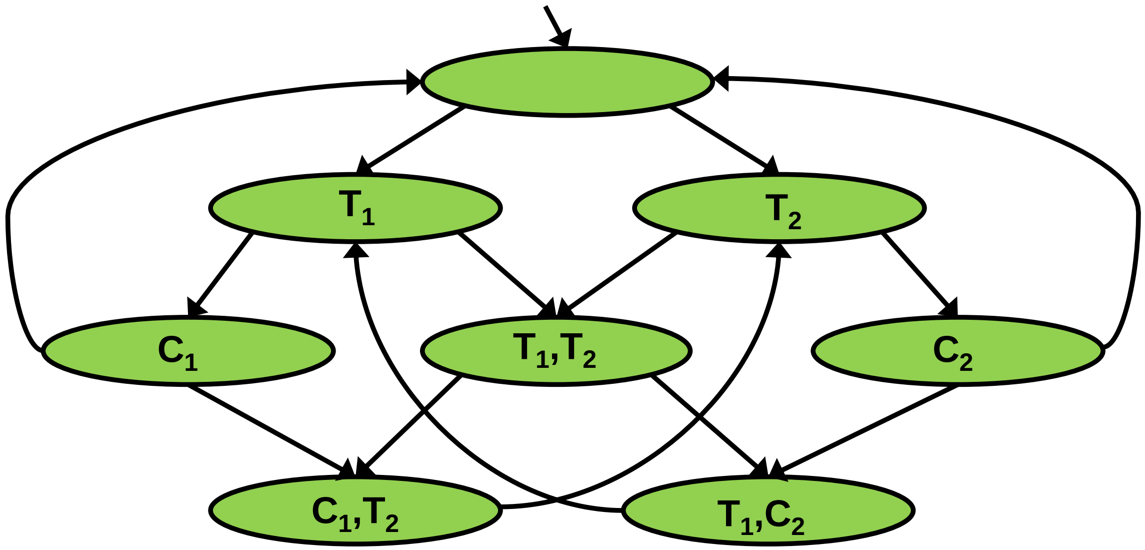
- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(C_1 \wedge C_2)$
- Yes, if $\neg(C_1 \wedge C_2)$ holds in all reachable states
- $S_i \equiv$ reachable states from an initial state after **i** steps

Warm-Up 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

Warm-Up Example: Mutual Exclusion

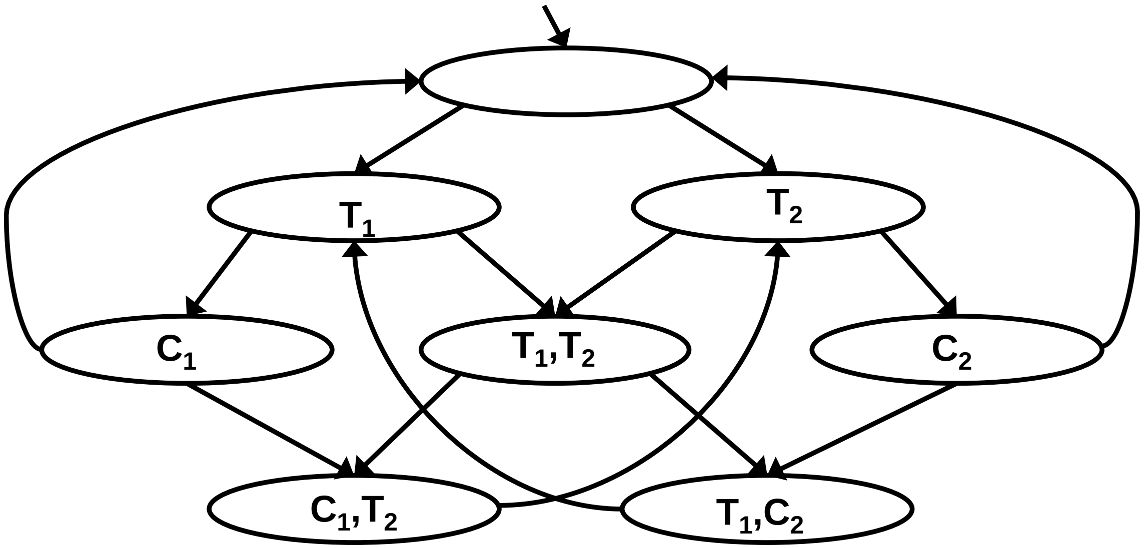


- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(C_1 \wedge C_2)$

$$M \models AG\ \neg\ (C_1 \wedge C_2)$$

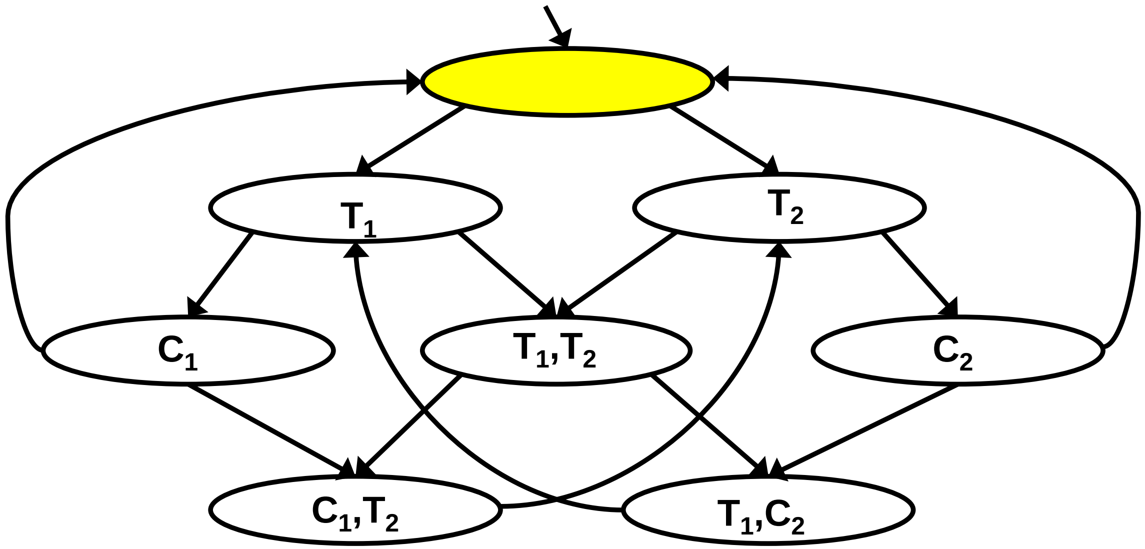
SCOS Secure & Correct Systems

Warm-Up Example: Mutual Exclusion



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

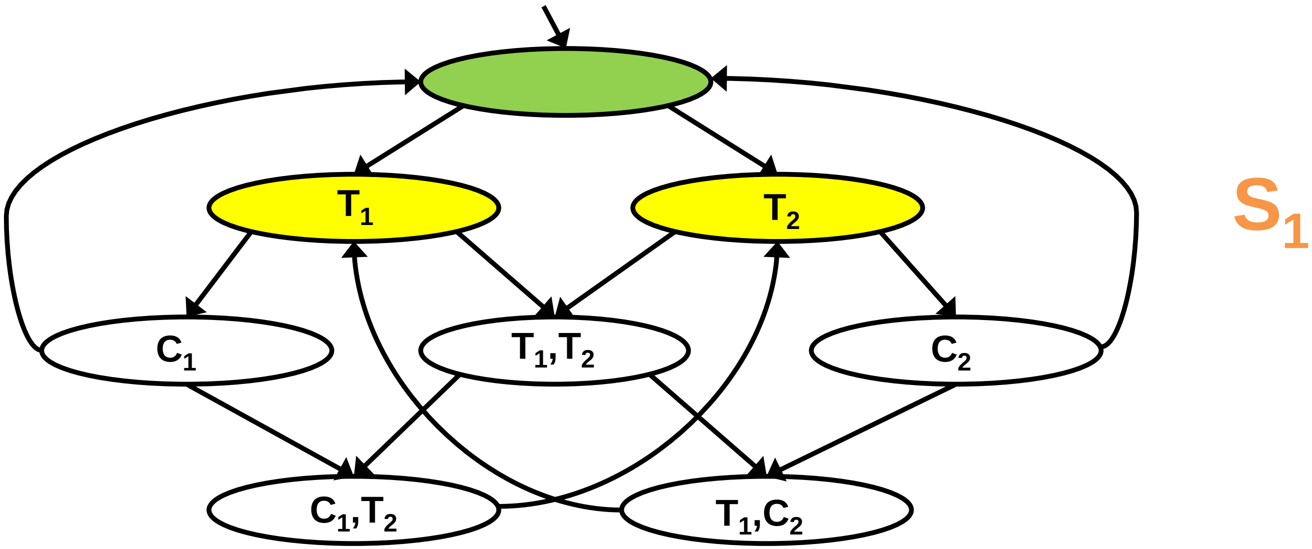
Warm-Up Example: Mutual Exclusion



S_0

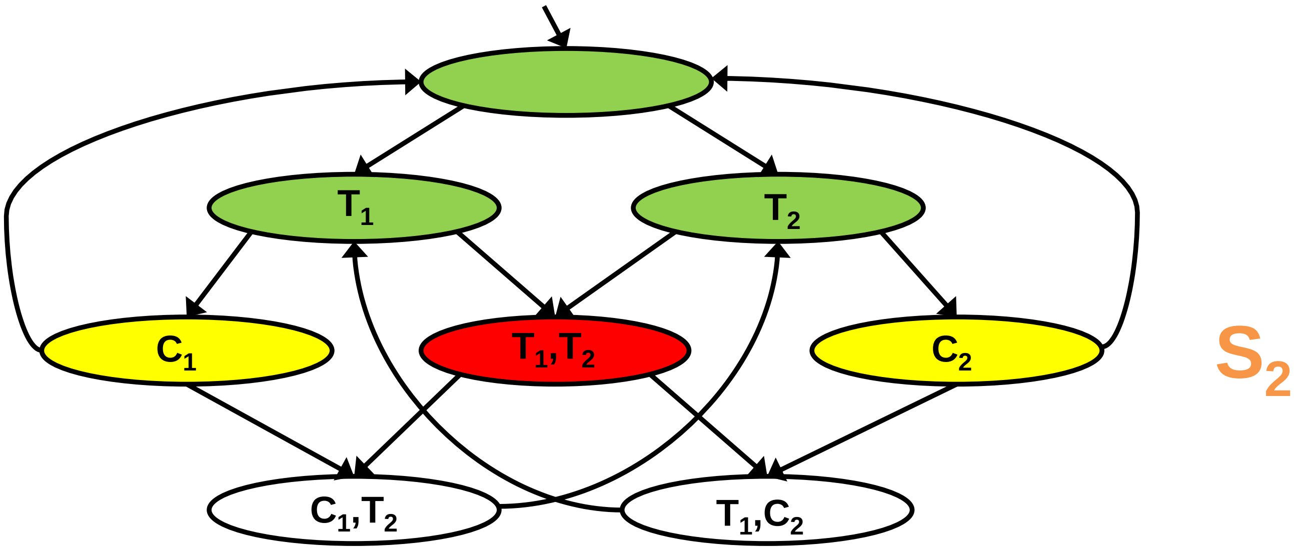
- Does it hold that $M \models f$?
  - Property 2: $f := \mathbf{AG}\neg(T_1 \wedge T_2)$
- $S_i \equiv$ reachable states from an initial state after **i** steps

Warm-Up Example: Mutual Exclusion



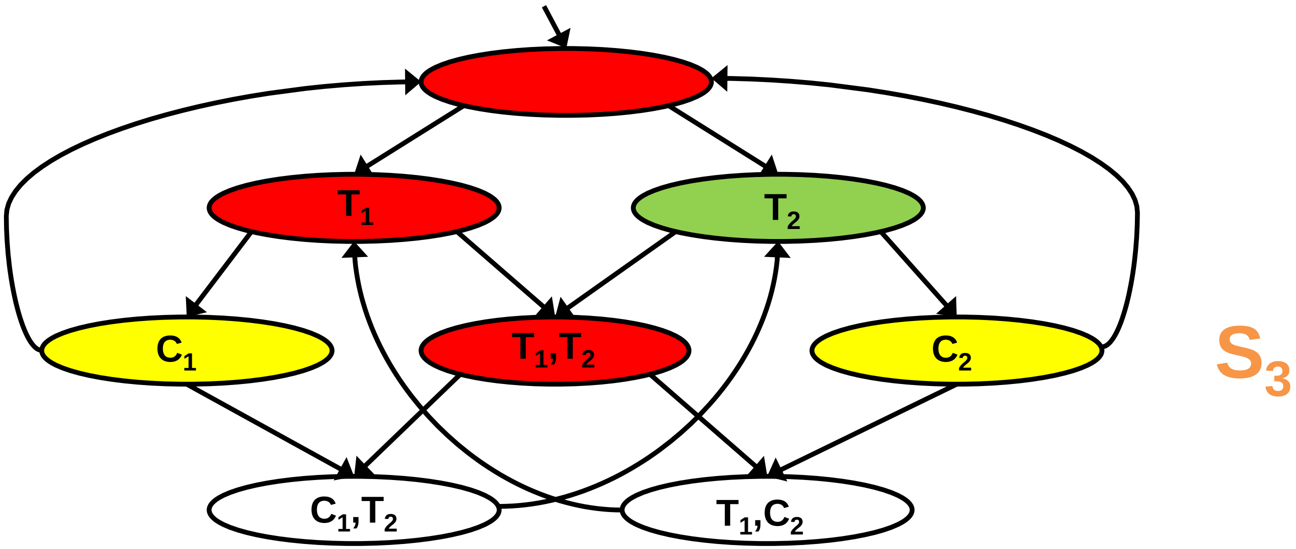
- Does it hold that $M \models f$?
  - Property 2: $f := \mathbf{AG}\neg(T_1 \wedge T_2)$
- $S_i \equiv$ reachable states from an initial state after **i** steps

Warm-Up Example: Mutual Exclusion



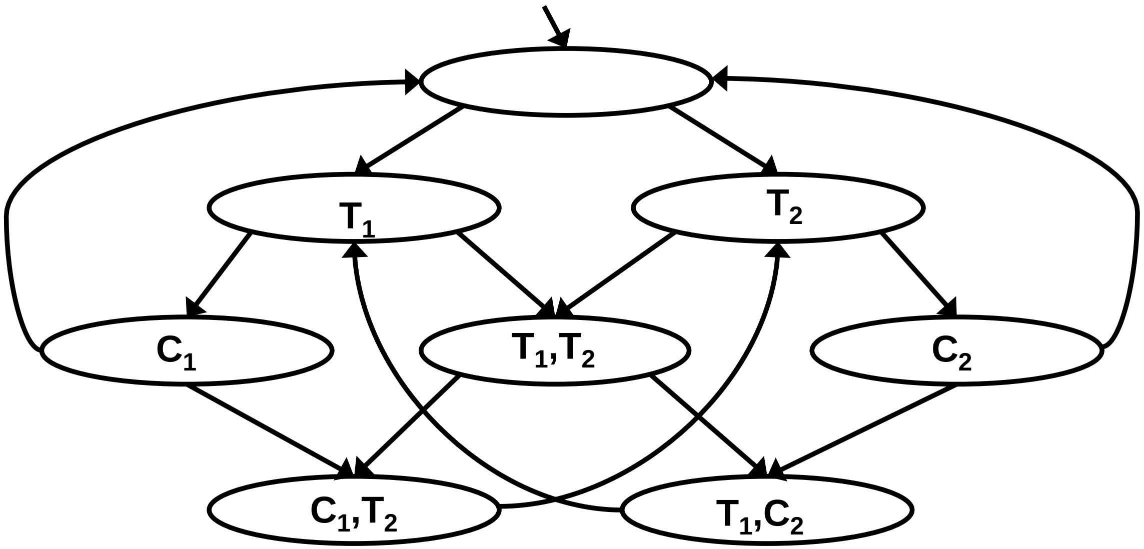
- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(T_1 \wedge T_2)$ ✗ $M \not\models AG \neg (T_1 \wedge T_2)$

Warm-Up Example: Mutual Exclusion



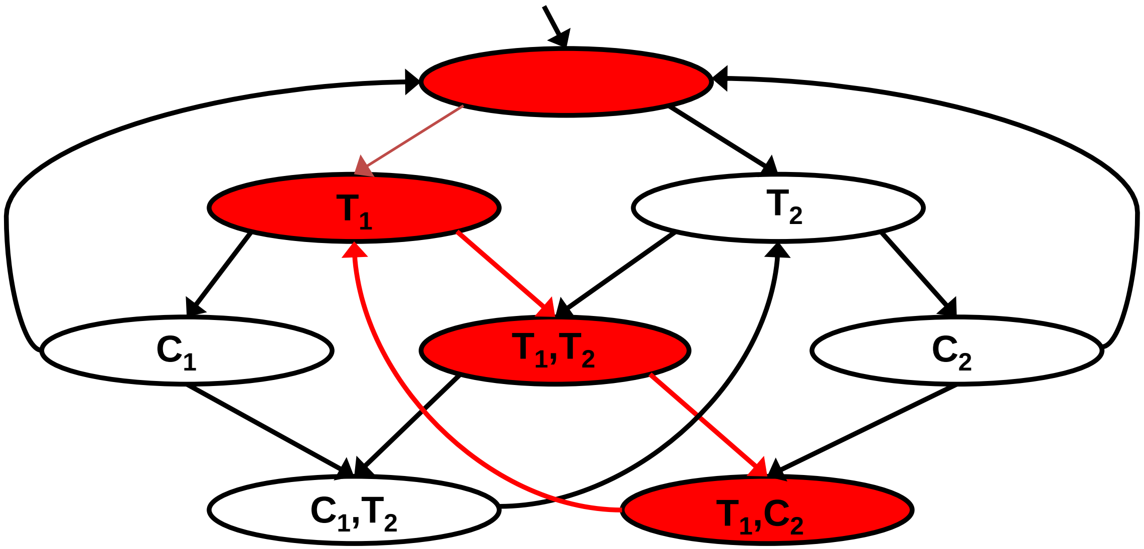
- Does it hold that $M \models f$?
  - Property 1: $f := \mathbf{AG}\neg(T_1 \wedge T_2)$ ✗ $M \not\models AG \neg (T_1 \wedge T_2)$
- Model checker returns a **counterexample**

Warm-Up 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

Warm-Up Example: Mutual Exclusion

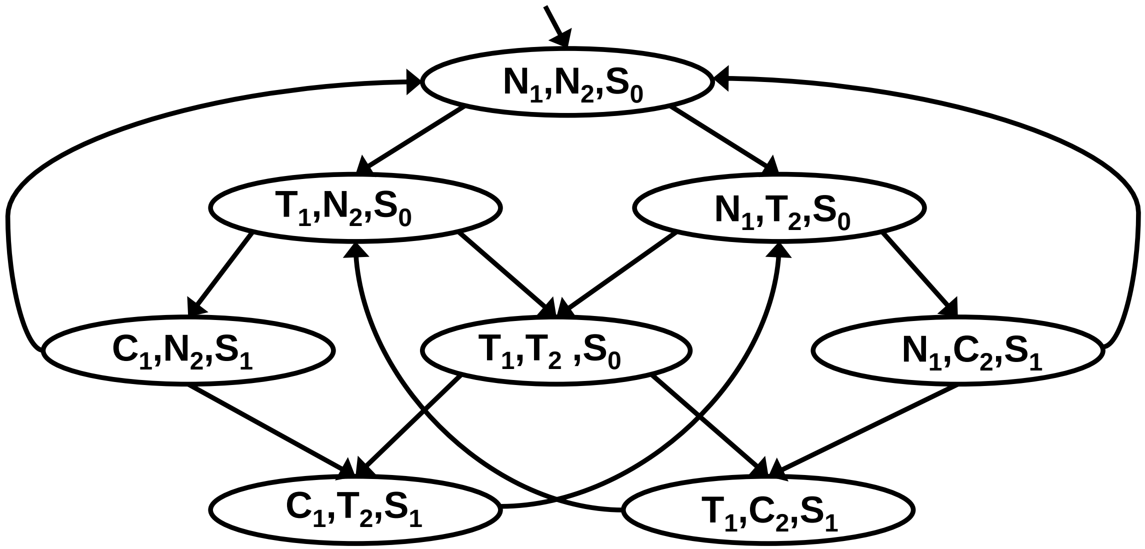


- Does it hold that M ⊨ f?
  - Property 3: f := **AG** (($T_1 \rightarrow$ **F** C_1) ∧ ($T_2 \rightarrow$ **F** C_2))
- In case M ⊭ f, compute a counterexample

✗ M ⊭ AG (($T_1 \rightarrow$ F C_1) ∧ ($T_2 \rightarrow$ F C_2))

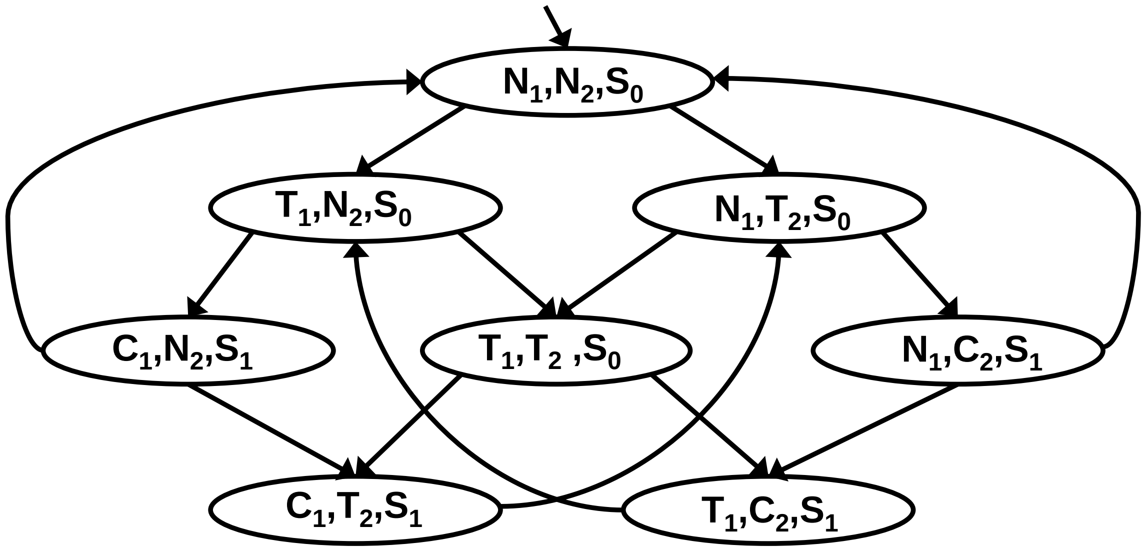
SCOS Secure & Correct Systems

Warm-Up Example: Mutual Exclusion



- Does it hold that M ⊨ 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 ⊭ f, compute a counterexample

Warm-Up Example: Mutual Exclusion



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

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

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 $M \models f$

SCOS
Secure & Correct Systems

CTL Model Checking $M \models f$

The goal 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

- Example: Sub-Formulas for checking
  AG(request $\rightarrow$ **AF** grant**)**

CTL Model Checking $M \models f$

The goal 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

- Example: Sub-Formulas for checking
  AG(request → **AF** grant**)**
  - Ceck grant, request
  - Then check **AF** grant
  - Next check request → **AF** grant
  - Finally check **AG(** request → **AF** grant**)**

CTL Model Checking $M \models f$

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

CTL Model Checking $M \models f$

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

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

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

- $M \models f$ if and only if $f \in label(s)$ for all initial states

For what types of sub-formulas to we need an MC algorithm?

- 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 sub-formulas that have already been checked
  - added to label(s), when needed

- Procedures for labeling states satisfying $\neg f_1$:
  - $\neg f_1$ add to label(s) if and only if $f_1 \notin label(s)$

- Give the procedure for labeling states satisfying $f_1 \vee f_2$

Model Checking ¬, ∨- Formulas

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

- Procedures for labeling states satisfying $\neg f_1$:
  - add $\neg f_1$ to label(s) if and only if $f_1 \notin label(s)$

- Give the procedure for labeling states satisfying $f_1 \vee f_2$
  - add $f_1 \vee f_2$ 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) };
```

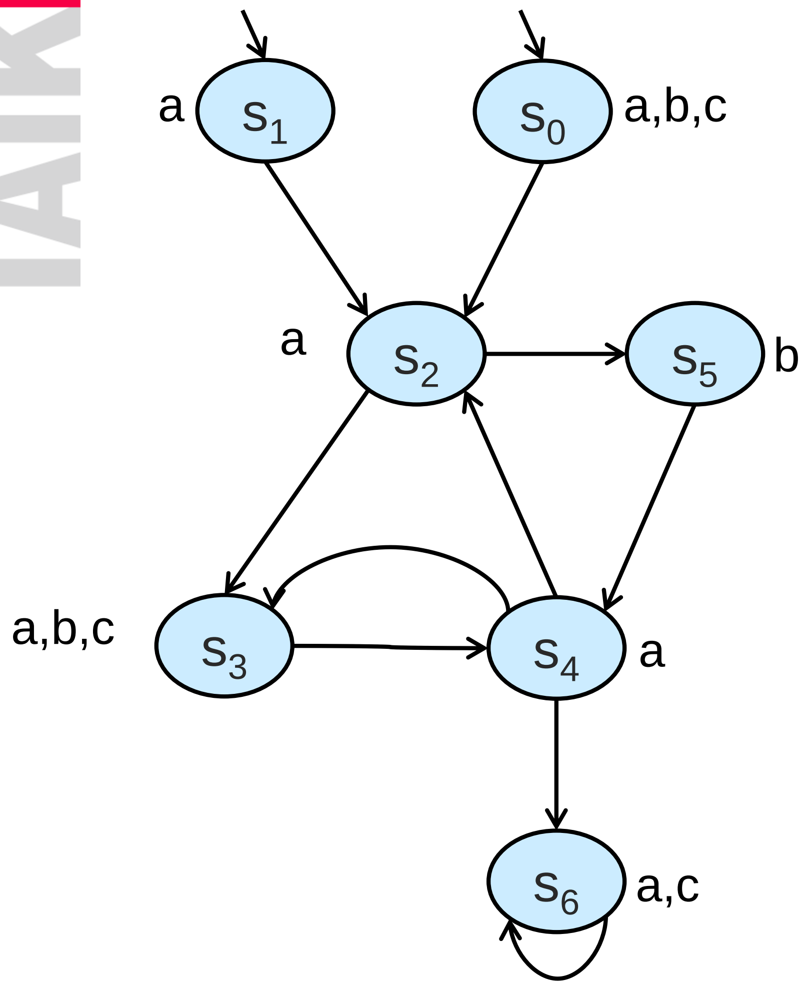
Model Checking $g = E(f_1 U f_2)$

- 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?

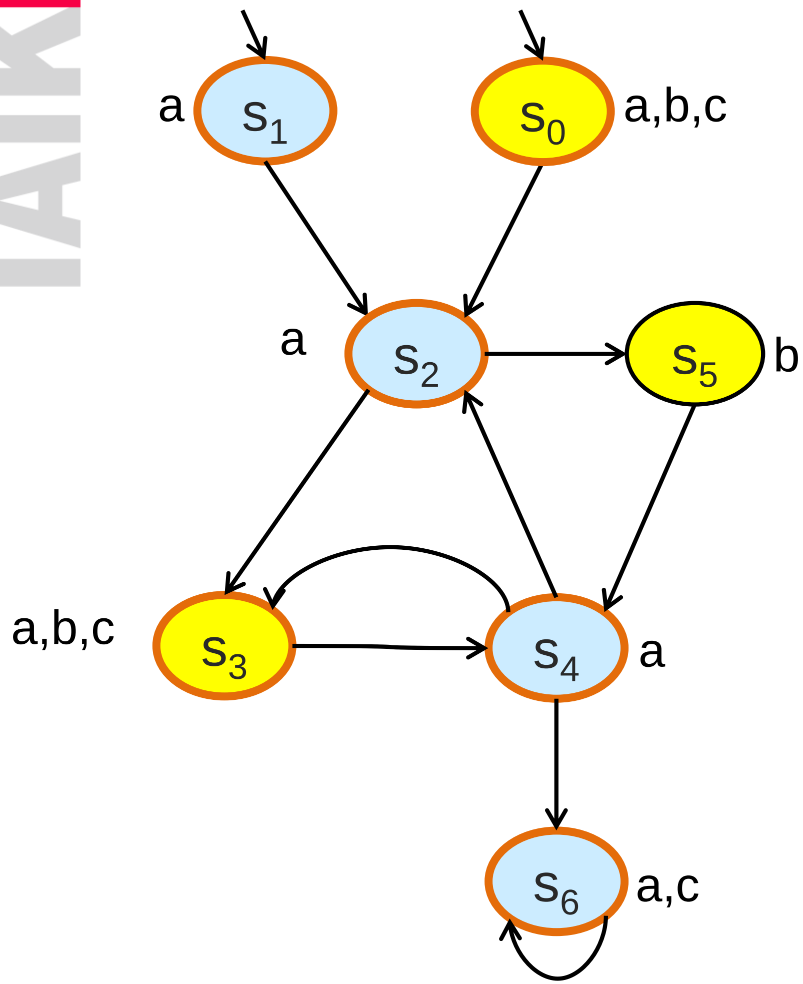
- $f := E(aUb)$

procedure CheckEU (f_1,f_2)
  T := { t | $f_2 \in$ 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 $f_1 \in$ label(s) then
        label(s) : = label(s) ∪ {E(f_1 U f_2) };
        T : = T ∪ {s}

Example: Model Checking U Formulas



Does it hold that M ⊨ f?

- $f := E(aUb)$

procedure CheckEU (f_1,f_2)

$T := \{ t \mid f_2 \in label(t) \}$

for all $t \in T$ **do**

  $label(t) := label(t) \cup \{ E(f_1\ U\ f_2) \}$

while $T \neq \emptyset$ **do**

  choose $t \in T$; $T := T \setminus \{t\}$;

  for all s **such that** R(s,t) **do**

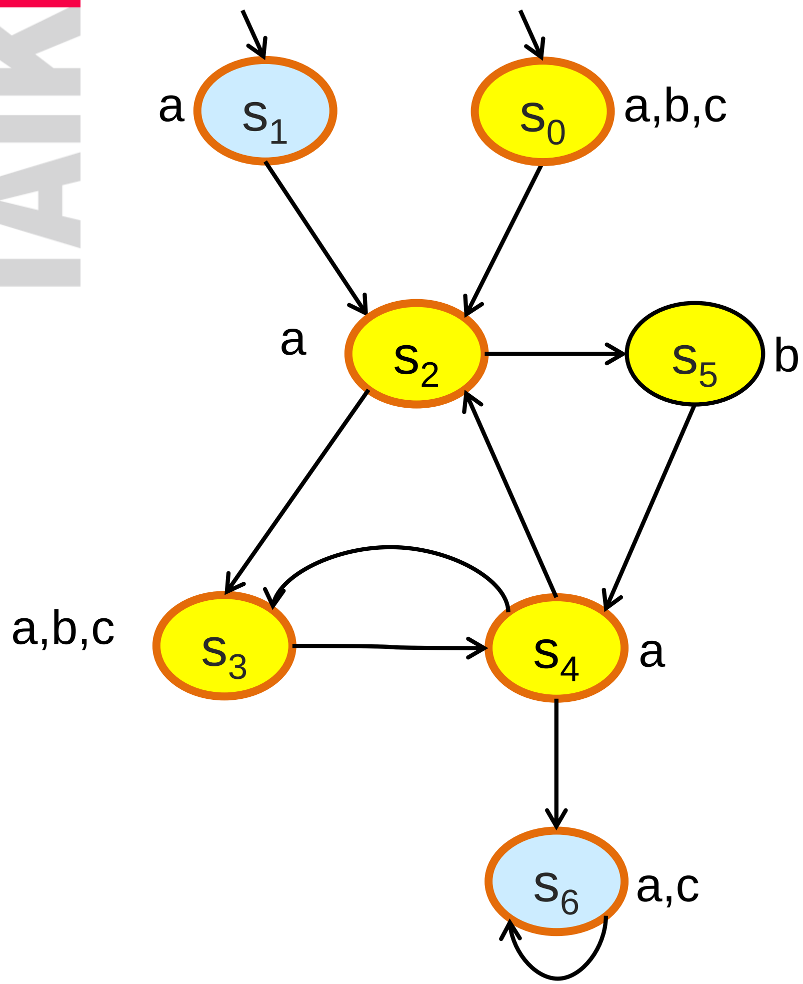
    if $E(f_1\ U\ f_2) \notin label(s)$ **and** $f_1 \in label(s)$ **then**

      $label(s) := label(s) \cup \{E(f_1\ U\ f_2)\}$;

      $T := T \cup \{s\}$

SCOS Secure & Correct Systems

Example: Model Checking U Formulas

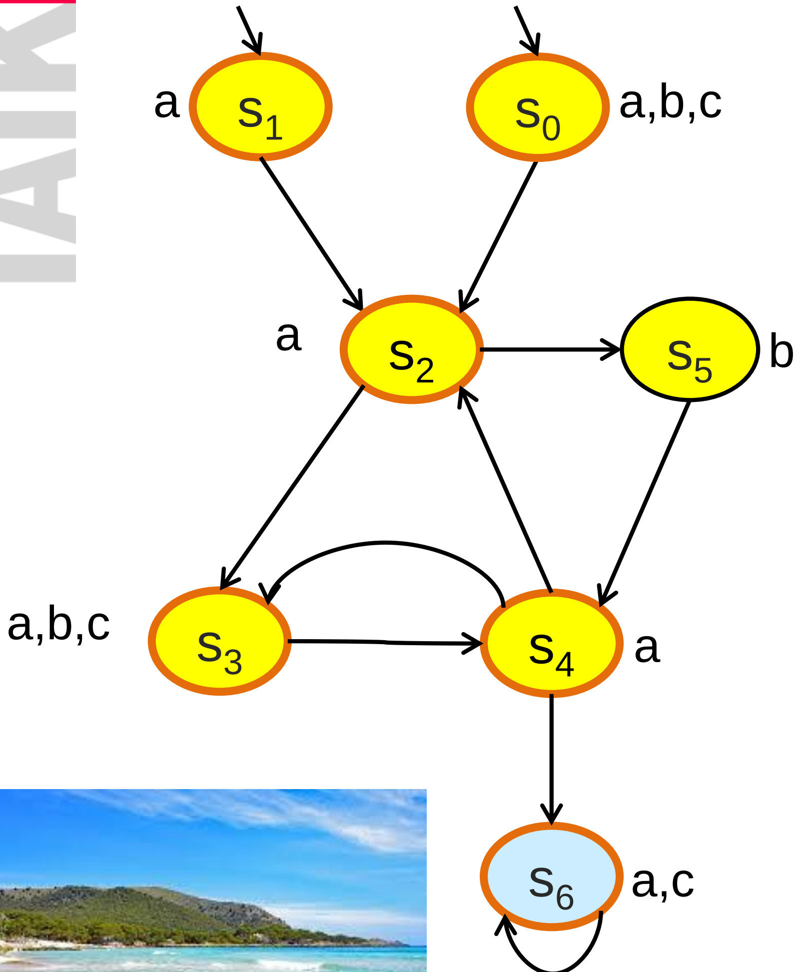


Does it hold that M ⊨ f?

- $f := E(aUb)$

```
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 (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}
```


SCOS Secure & Correct Systems

Model Checking $g = EGf_1$

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)

Model Checking $g = EG f_1$

- 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 $g = EGf_1$

- 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. s has a $path$ in M' to some state t in a MSCC of M'

SCOS Secure & Correct Systems

Model Checking $g = EG f_1$

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

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

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 $g = EG f_1$

Model Checking Complexity

Steps per Subformula

- MC Atomic Propositions
  - O(|S|) steps
- MC $\neg$, $\vee$ formulas
  -
- 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 ¬, ∨ formulas
  - O(|S|) steps
- 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 $\neg$, $\vee$ 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 $g = EG f_1$

SCOS Secure & Correct Systems

Model Checking Complexity

Steps per Subformula

- MC Atomic Propositions
  - O(|S|) steps
- MC ¬, ∨ 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 $g = EG f_1$

SCOS
Secure & Correct Systems

Model Checking Complexity

Steps per Subformula

- MC $g = EG f_1$
  - 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|)

SCOS Secure & Correct Systems

Model Checking Complexity

- Each subformula
  - O(|S|+ |R|) = O(|M|)

What is the total complexity for checking f?

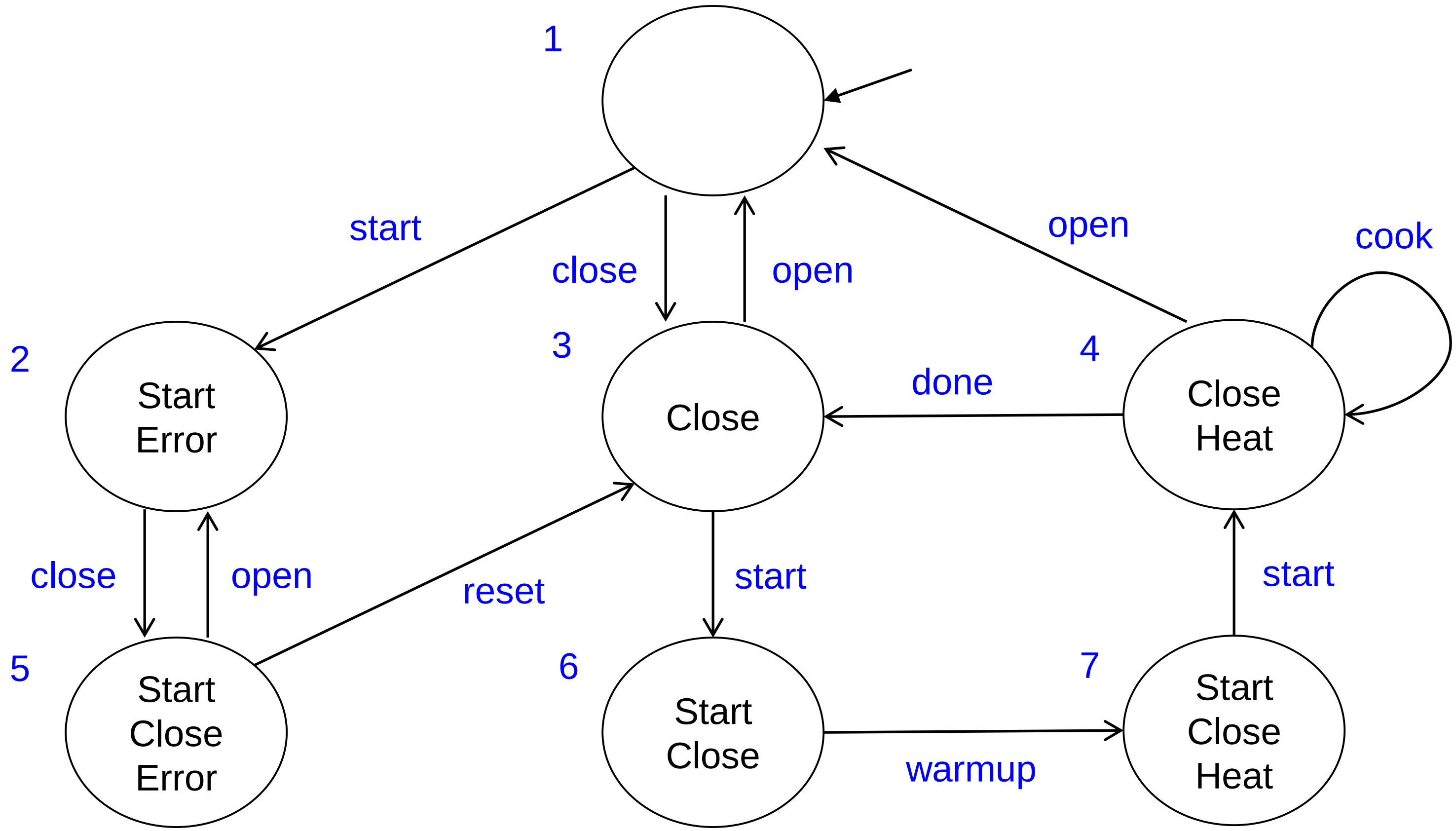
Model Checking Complexity

- 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

ToDo

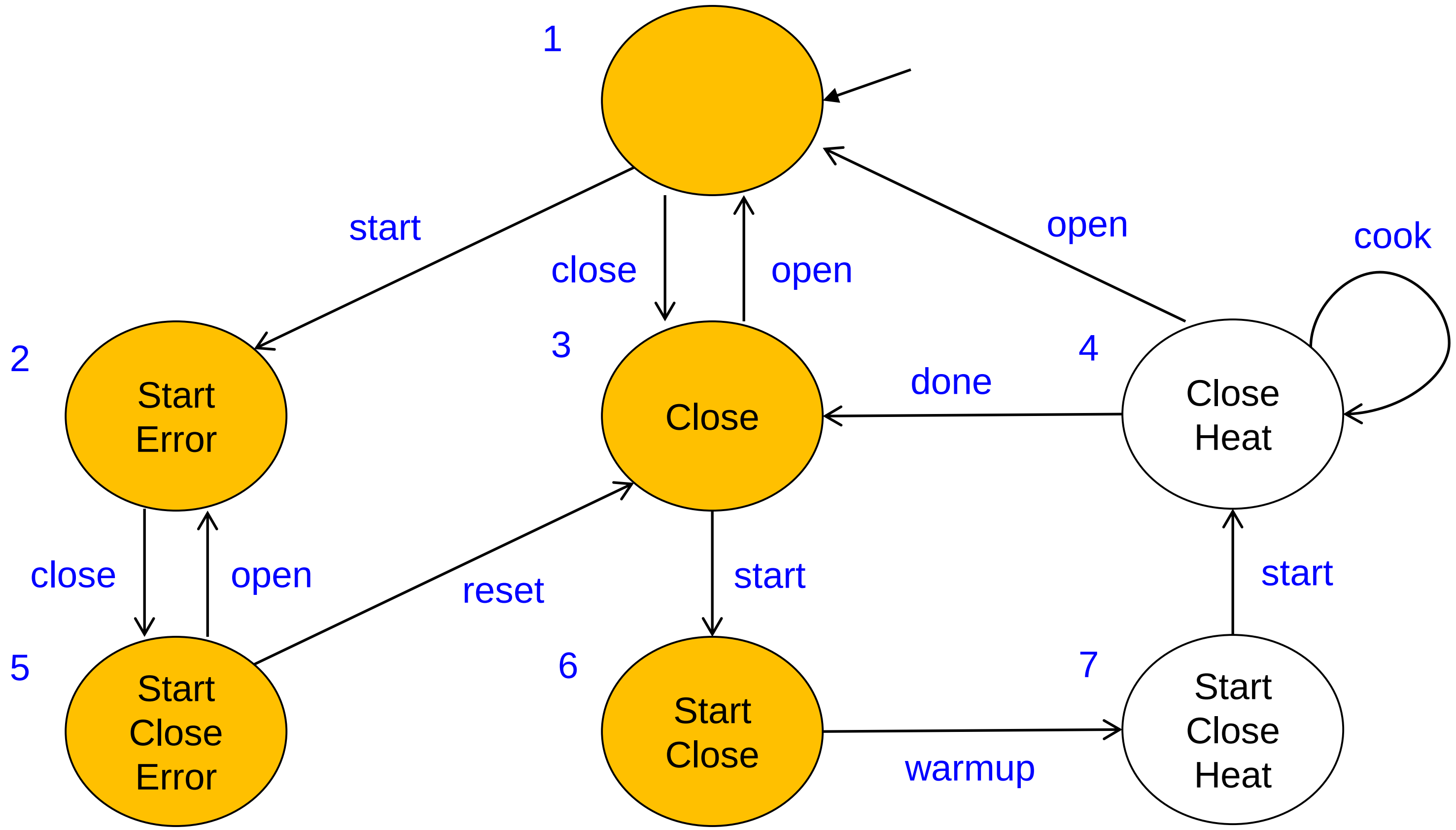
- Use the proposed algorithm to compute if M ⊨ f?
  - f := ¬E (true U (Start ∧ EG ¬Heat))



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

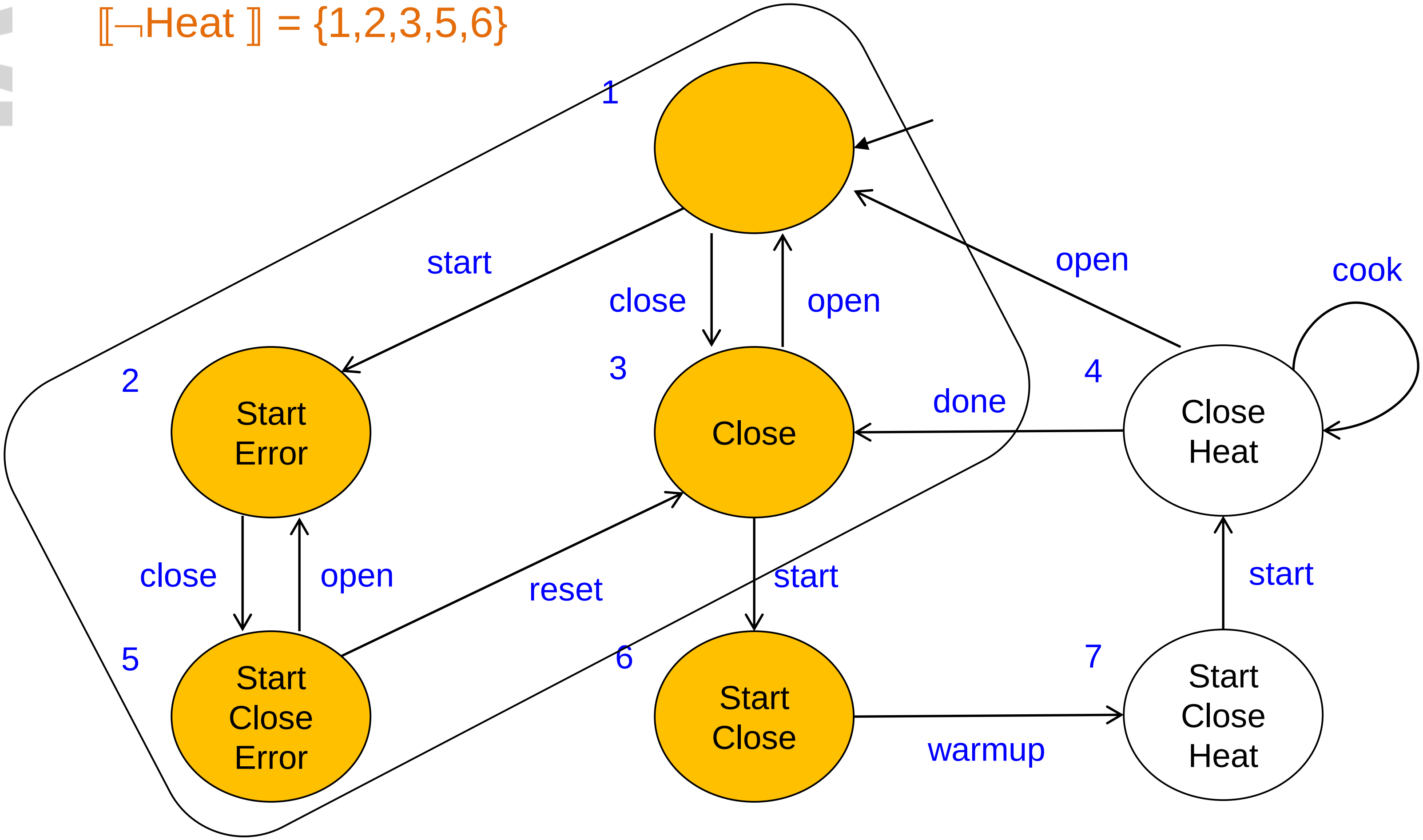
⟦start⟧ = {2,5,6,7}

⟦¬Heat ⟧ = {1,2,3,5,6}



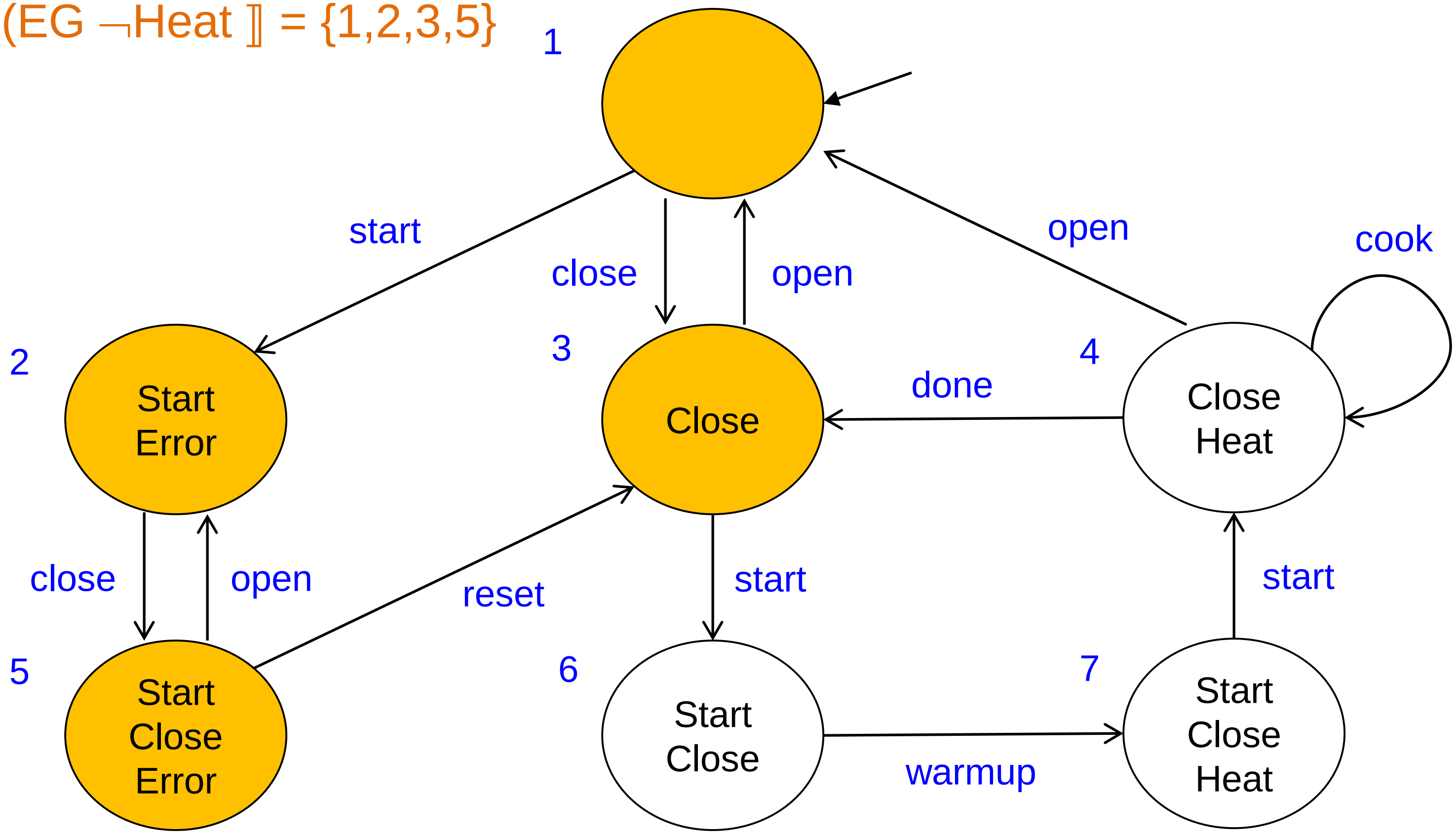
$$f := \neg E\ (true\ U\ (Start \wedge EG\ \neg Heat))$$

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



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

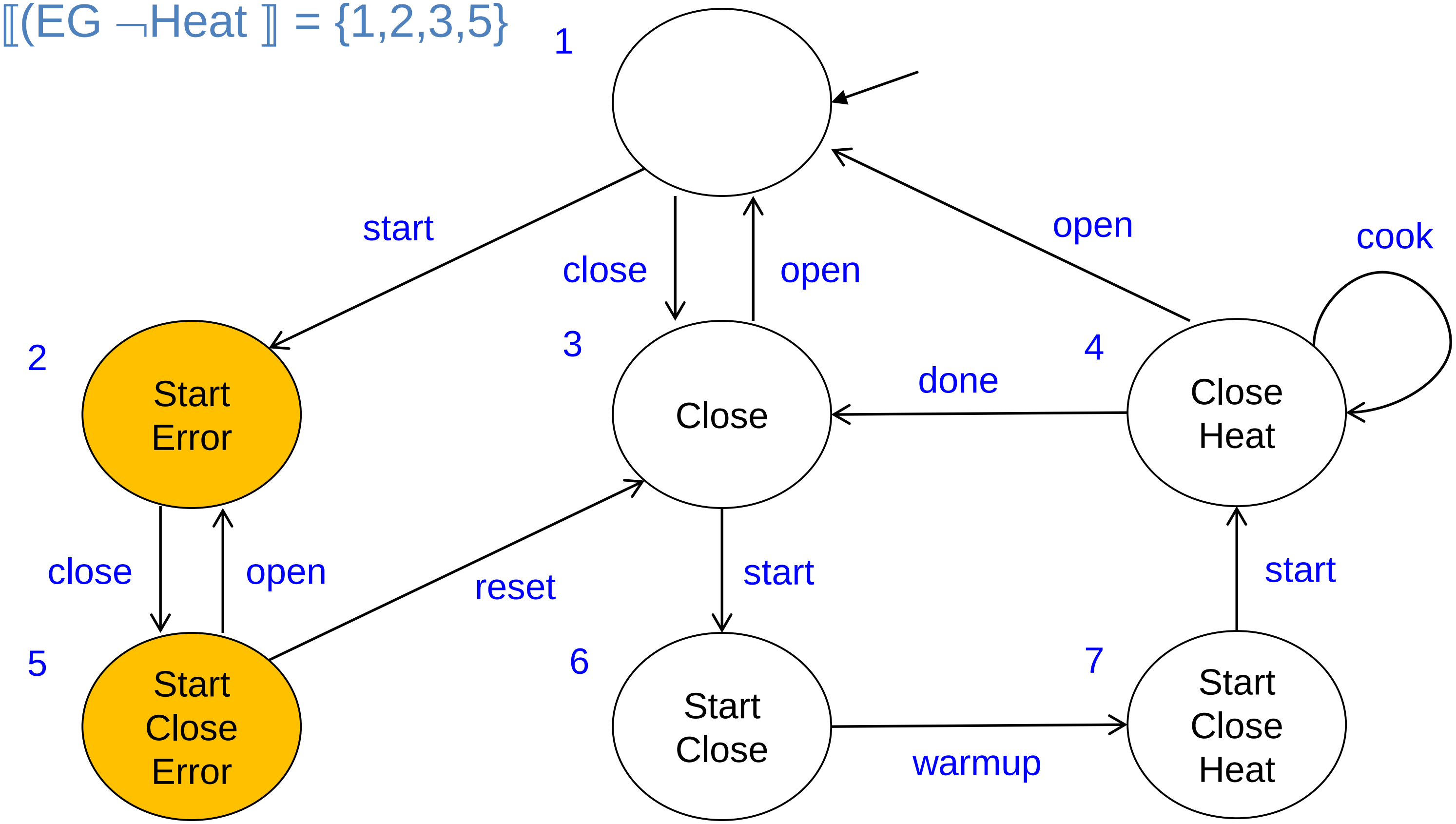
⟦start⟧ = {2,5,6,7}
⟦¬Heat⟧ = {1,2,3,5,6}
⟦(EG ¬Heat⟧ = {1,2,3,5}



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

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

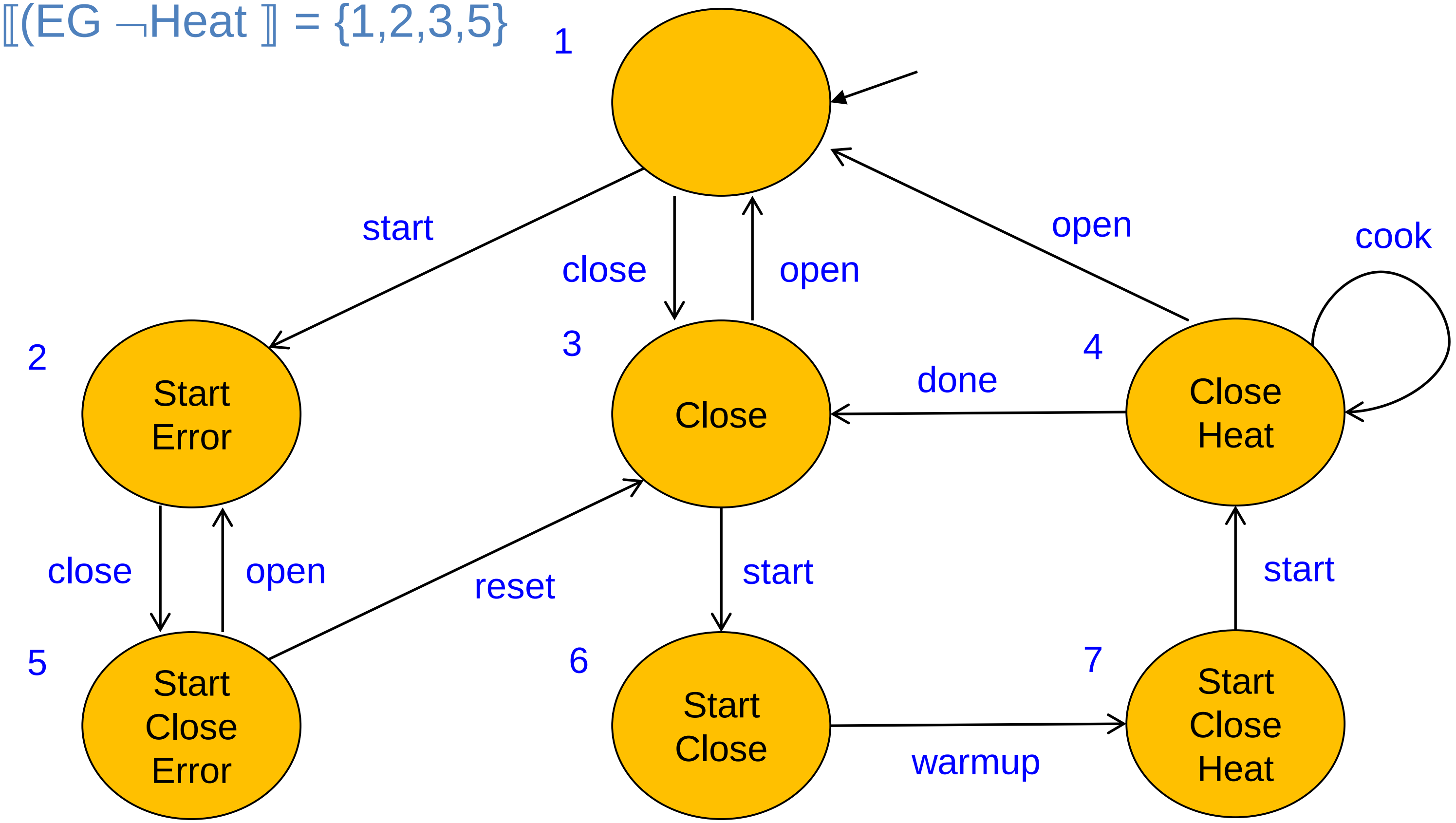
⟦ Start ∧ EG ¬Heat ⟧ = {2, 5}



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

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

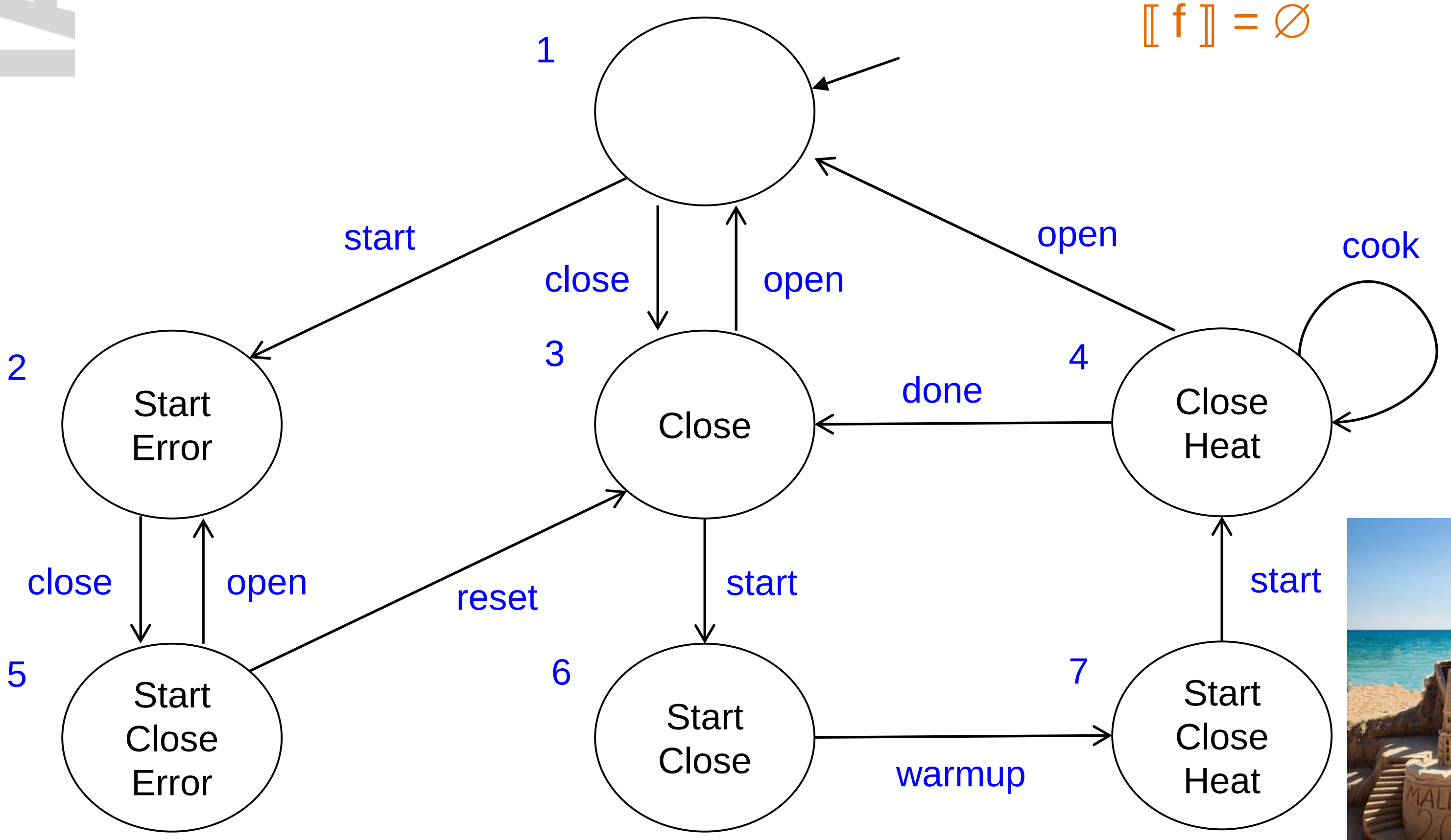
⟦ Start ∧ EG ¬Heat ⟧ = {2, 5}
⟦ EU ⟧ = {1,2,3,4,5,6,7}



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

⟦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⟧ = ∅



Secure & Correct Systems


THANK YOU!