

Mutual Exclusion I

- Setting: Concurrent processes with shared resource (file, database entry, ...).
- Requirement: No (e.g., write-)access at the same time.
- Every process has *critical section* (for access on shared resource).
- Requirement: **Only one process in its critical section at a time.**
- Solution approach: Find (define) *protocol* in order to determine when every process is allowed to enter its *critical section*.

Mutual Exclusion II

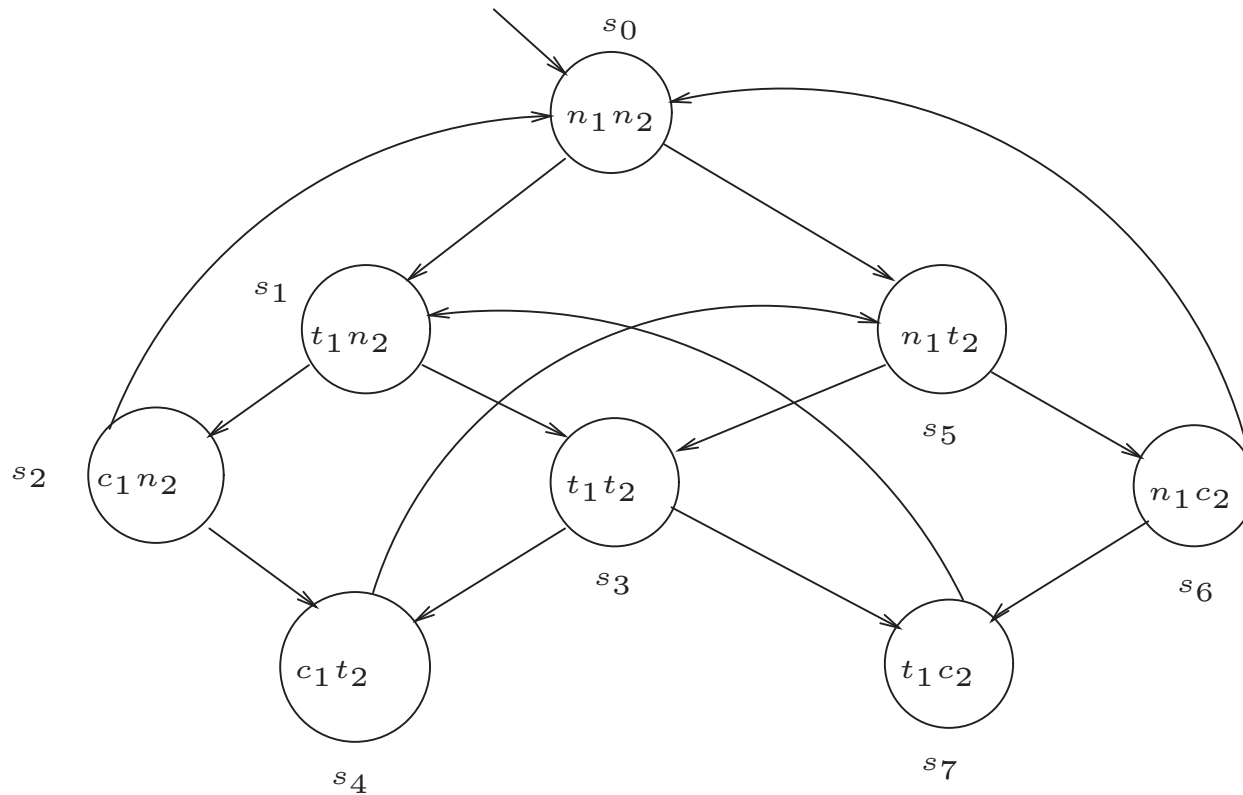
Task regarding protocol: Verify intended properties, e.g.:

- *Safety*: Only one process in its critical section at a time.
- *Liveness*: Whenever a process wants to enter its critical section, it will eventually be permitted to do so.
- *Non-blocking*: A process can always request to enter its critical section.
- *No strict sequencing*: Processes need not enter their critical section in strict sequence.

Mutual Exclusion III

Modelling: 1st Attempt

- 2 processes (1,2) with the following transitions: $n \rightarrow t \rightarrow c \rightarrow n \rightarrow \dots$
- States: In non-critical state (n), trying to enter critical state (t), in its critical state (c)



Mutual Exclusion IV

Modelling (1st Attempt): Properties (symmetric in 1,2)

- Safety: $\phi_1 \stackrel{\text{def}}{=} AG \neg(c_1 \wedge c_2)$ satisfied.
- Liveness: $\phi_2 \stackrel{\text{def}}{=} AG (t_1 \rightarrow AF c_1)$ **not satisfied**
from s_0 : t_1 in s_1 true, but not $AF c_1$ (consider path $s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow s_1 \dots$).
- Non-blocking: $\phi_3 \stackrel{\text{def}}{=} AG (n_1 \rightarrow EX t_1)$ satisfied.
- No strict sequencing: $\phi_4 \stackrel{\text{def}}{=} EF (c_1 \wedge E[c_1 U (\neg c_1 \wedge E[\neg c_2 U c_1])])$ satisfied
(choose e.g. path $s_5 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5 \dots$).

Note:

- Here: Specification of the last 3 properties from the point of view of process 1.
- Specification jointly for both processes: ???

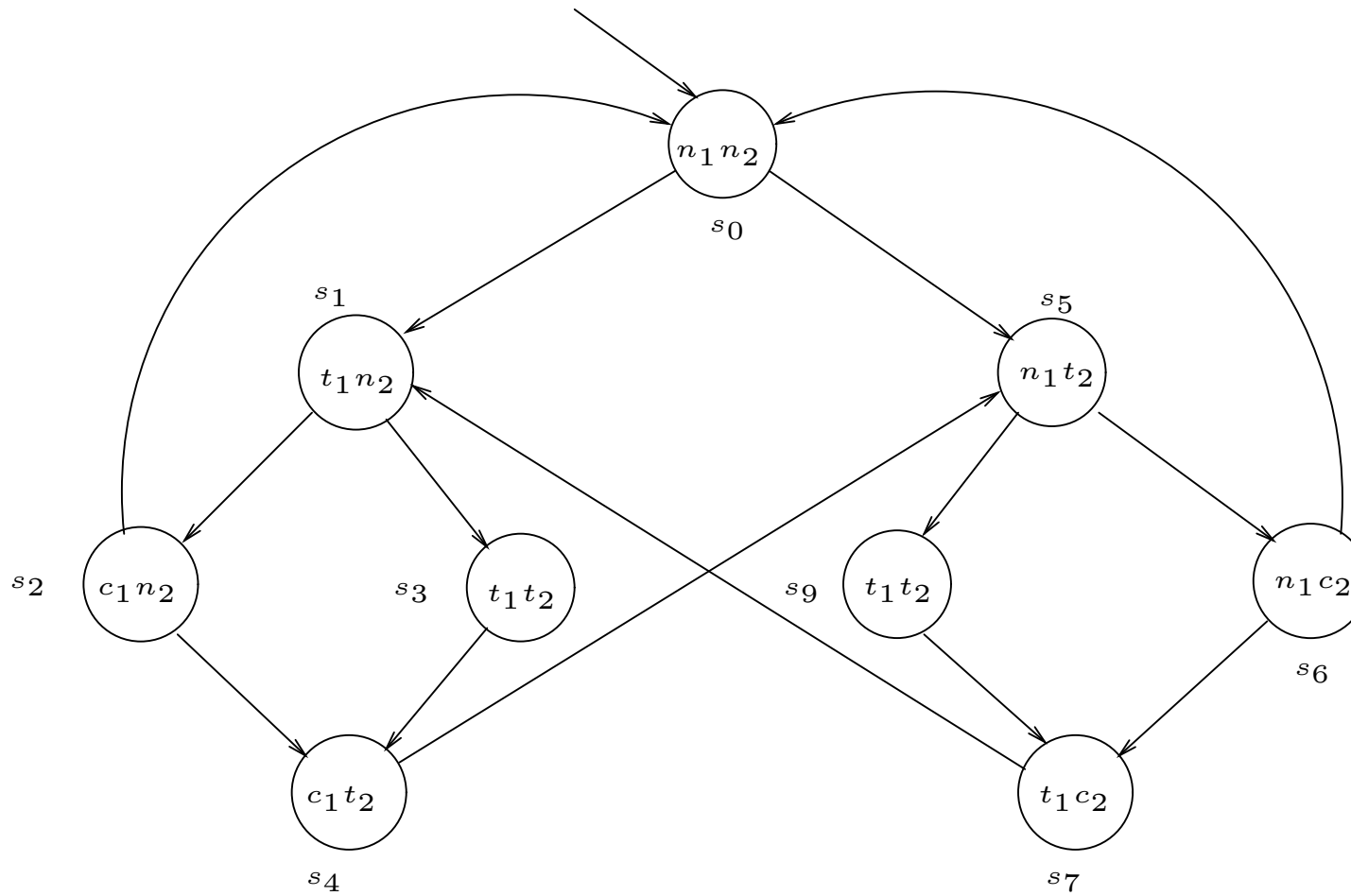
Mutual Exclusion V

Problem with 1st modelling:

- Non-determinism enables repeated preference of the same process.
- Problem technically: s_3 does not distinguish situation according to “history” (which process was first in state **trying**?).
- Solution idea: Split s_3 into two states, according to the current “history”.
- Modelling – 2nd attempt (see figure): Satisfies all 4 properties!
- Possibilities for extensions / improvements: E.g. allow longer time in critical state (for instance via a cycle in s_1)
→ Liveness violated → Solution: *fairness constraints*.

Mutual Exclusion VI

Modelling: 2nd Attempt:



Model Checking: Problems

- Realistic transition systems may easily have millions of states, hence **efficiency very important**.
- Unfolding into infinite graph infeasible in practice.
- Algorithms must work on finite data structure.
- Questions:
 - (a) Given: \mathcal{M}, s_0, ϕ . Question: $\mathcal{M}, s_0 \models? \phi$ – Yes/No (Why?)
 - (b) Given: \mathcal{M}, ϕ . Question: $\mathcal{M}, s \models \phi$ for which s ?
- Algorithm for (b) (with solution set L) yields solution also for (a) (test additionally whether $s_0 \in L$?)
- Pre-processing for algorithm gives: $\phi \longrightarrow \phi'$, ϕ' only contains $\perp, \neg, \wedge, AF, EU, EX$

Labelling Algorithm I

Input: CTL-model $\mathcal{M} = (S, \rightarrow, L)$, CTL-formula ϕ

Output: Set of all states of \mathcal{M} , in which ϕ holds.

Working mode (principle):

- Start with $\phi \stackrel{\text{def}}{=} \text{TRANSLATE}(\phi)$
(version after pre-processing).
- Label states of \mathcal{M} with all sub-formulae of ϕ , that hold in the respective state.
- Start with the smallest sub-formulae of ϕ und work outwards towards ϕ .
- Induction step: Let ψ be a sub-formula of ϕ and let all states, which are direct sub-formulae of ψ , be already labelled.

Then label ψ by case distinction:

Labelling Algorithm II

Labelling w.r.t. ψ according to case distinction:

- \perp : No state is labelled with \perp .
- p : Label s with p if $p \in L(s)$.
- $\psi_1 \wedge \psi_2$: Label s with $\psi_1 \wedge \psi_2$, if s already marked with ψ_1 and ψ_2 .
- $\neg\psi_1$: Label s with $\neg\psi_1$, if not marked with ψ_1 .

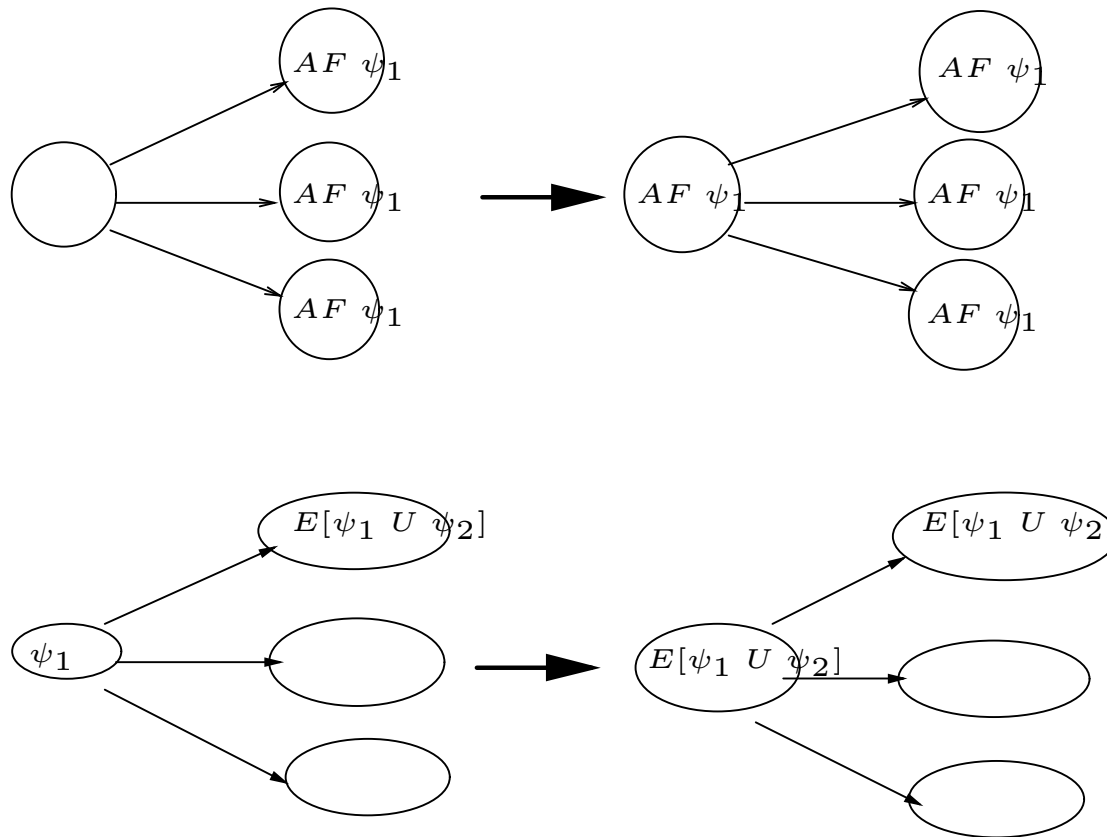
Labelling Algorithm III

- $AF \psi_1$:
 - Label s , if already marked with ψ_1 , with $AF \psi_1$.
 - REPEAT mark s with $AF \psi_1$ if all (direct) successor states of s are already marked with $AF \psi_1$ UNTIL “no change”.
- $E[\psi_1 U \psi_2]$:
 - Label s , if already marked with ψ_2 , with $E[\psi_1 U \psi_2]$.
 - REPEAT mark s with $E[\psi_1 U \psi_2]$, if s marked with ψ_1 and at least one (direct) successor of s marked with $E[\psi_1 U \psi_2]$ UNTIL “no change”.
- $EX \psi_1$:

Label s with $EX \psi_1$, if some (direct) successor of s already marked with ψ_1 .

Labelling Algorithm IIIa

Illustration of the cases $AF \psi_1$ and $E[\psi_1 U \psi_2]$:

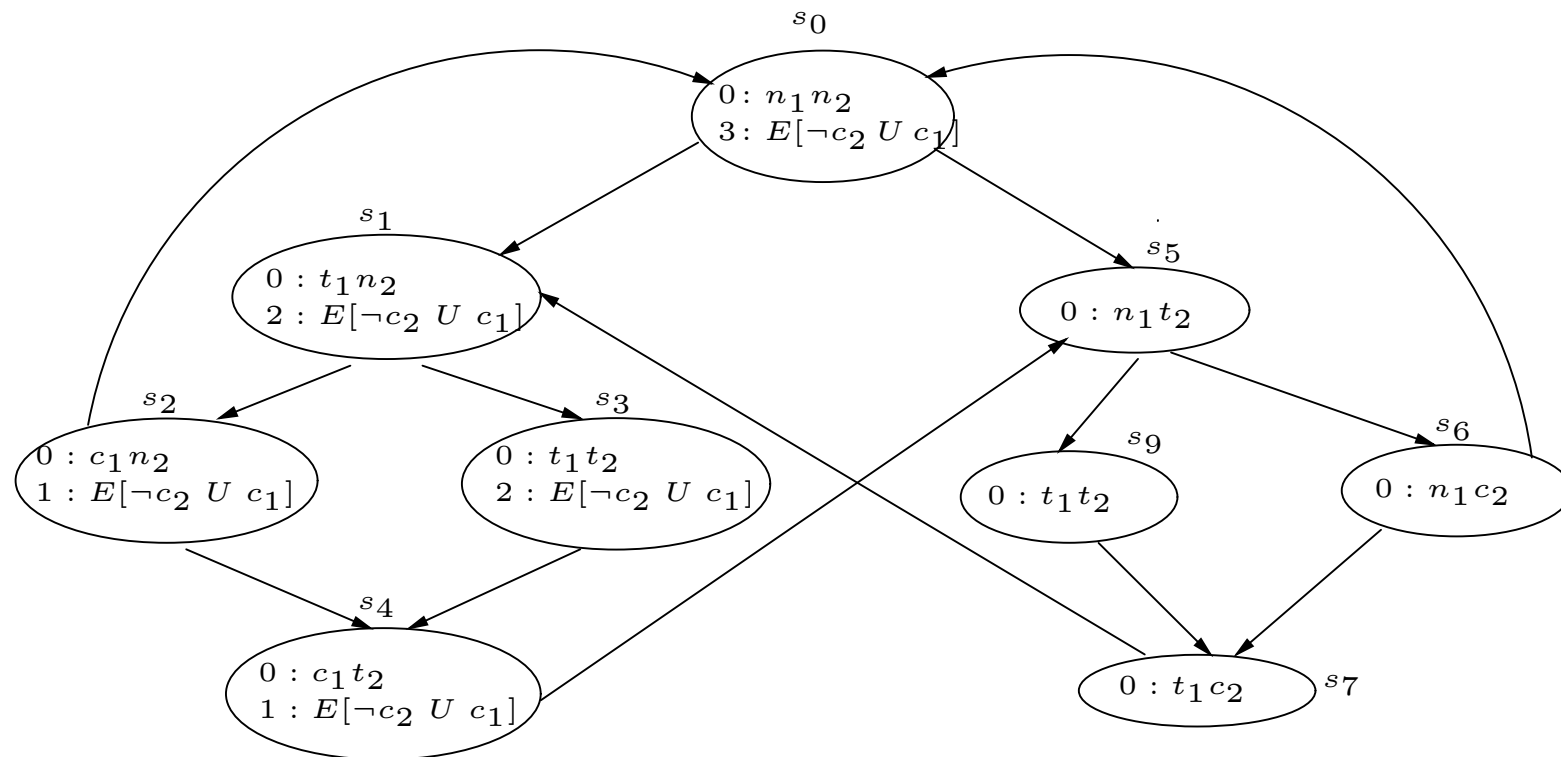


Labelling Algorithm IV

- Final phase: If labelling for all sub-formulae of ϕ done (incl. ϕ), output $\stackrel{\text{def}}{=} \text{all}$ states that are labelled with ϕ .
- Complexity of the algorithm: $O(f \cdot V \cdot (V + E))$, where f number of operators, V number of states, E number of transitions.
- Algorithm linear in the size of the formula und quadratic in the size of the model.
- Variants:
 - Without pre-processing, direct treatment of e.g. EG .
 - Usage of EX , EU , EG als basic operators (instead of EX , EU , AF), with *backwards breadth-first search*: In total more efficient (linear in size of formula and size of model).

Labelling Algorithm IVa: Example

Labelling for Mutual Exclusion (2nd Variant) w.r.t. $E[\neg c_2 U c_1]$:



The “State Explosion” Problem

- Labelling algorithm linear in size of model, **but size of model exponential in number of variables and in number of “parallel system components”**
→ **state explosion problem.**
- Approaches for dealing with (partial solutions of) the state explosion problem:
 - **Efficient data structures: OBDDs** for representing sets of states instead of individual states.
 - **Abstraction** (from irrelevant details in the model).
 - **Partial order reduction** (in asynchronous systems).
 - **Induction** (when there exist many identical or similar system components).
 - **Composition / decomposition** (decompose the problem into several simpler ones).