The Algorithmic Verification of Reactive Systems

Lecture 9:

Algorithms for Compositional Verification

The Problem

- All verification algorithms we have given so far operate on the description of a complete system.
- Since the size of the complete system is often much too large, *compositional verification* aims at verifying the system component by component, e.g. process by process.
- Basically, compositional verification aims at proceeding as follows. To verify that $P_1 \parallel P_2$ satisfies a property f, find properties f_1 and f_2 such that

$$\begin{array}{l} P_1 \models f_1, \\ P_2 \models f_2, \text{ and} \\ f_1 \wedge f_2 \supset f. \end{array}$$

However, this can be inapplicable for a number of reasons.

- The process P_1 can satisfy f_1 when running alone, but not in the presence of P_2. In fact, one has to verify the $P_1 \models f_1$ whatever the other process does. This is a different verification problem from the one we have considered so far.

  Example: P_1 uses a variable that can be modified by P_2

- The assertion language might not be able to express sufficiently complete properties of the processes for compositional verification to be possible.

  Example: Propositional temporal logic cannot represent all finite-state behaviors.

- The logical conjunction of assertions might not be the adequate semantical representation of concurrent composition.

  Example: Fair/unfair concurrent composition.

A Less Ambitious Objective

Eliminate the problems associated with the assertion language as follows.

To verify that $P_1 \parallel P_2$ satisfies a property f,

- Replace P_1 by a "simpler" "equivalent" process P_1',
- Replace P_2 by a "simpler" "equivalent" process P_2',
- Verify that $P_1' \parallel P_2'$ satisfies the property f.

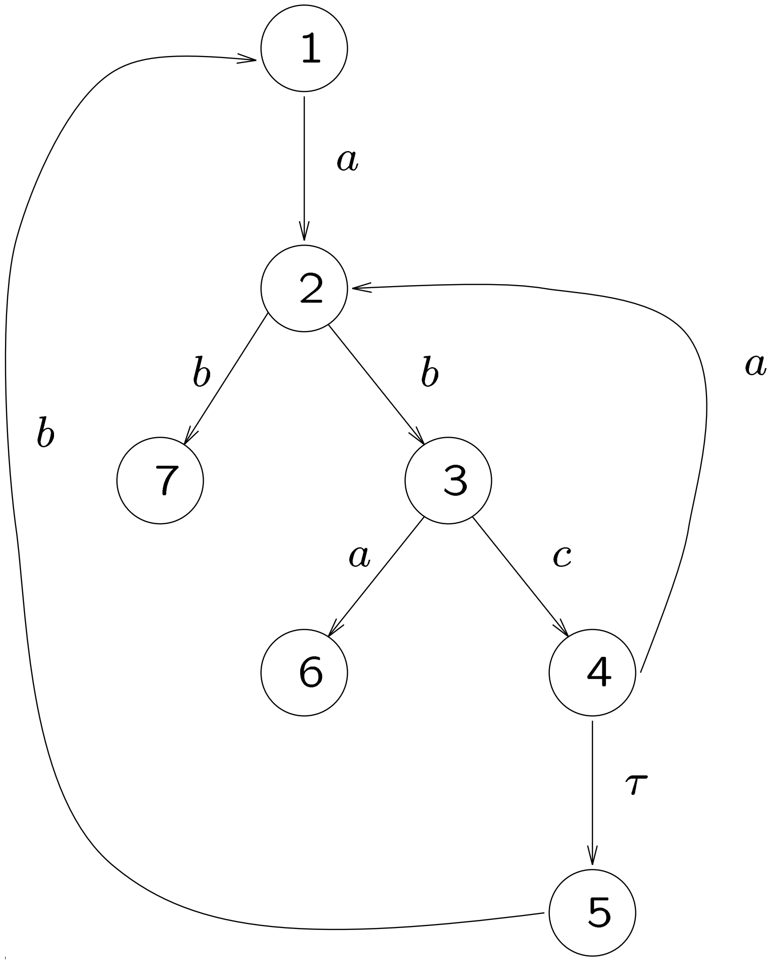
The problem is now to define "equivalent" processes. Note that "equivalent" should be "equivalent in the context of any other process".

A Restricted Class Of Processes

To simplify the problem of defining process equivalence, we consider a restriction of the class of processes used in FCSs.

- The processes do not use variables. Thus the only interaction between processes is through shared transitions (rendez-vous).
- Since shared transitions cannot be defined at the level of a single process, we proceed as follows.
  - Each process is a transition system (S, Σ, s_0, T) with labeled transitions.
  - Among the transition labels, one distinguishes the *internal* label $\tau \notin \Sigma$ which always corresponds to a nonsynchronized transition.
  - Transitions labeled by any other label $a \in \Sigma$ have to be synchronized with a similarly labeled transition of all processes that contain a in their alphabet.

Example



Note : the transition system can be nondeterministic in the sense that several identically labeled transitions can leave from a given state.

Composing Process

To define precisely the parallel composition of processes of the type we have just defined, we show how such a collection of processes can be mapped to an FCS.

Let $P_i = (S_i, \Sigma_i, s_{0i}, T_i)$, $1 \leq i \leq n$ be a set of processes. The corresponding FCS is defined as follows.

- $\mathcal{P} = \{P_1, \ldots, P_n\}$ with $\ell(P_i) = S_i$ and $\ell_0^{P_i} = s_{0i}$.
- $\mathcal{M} = \emptyset$

.../...

- To define $\mathcal{T}$, we consider each possible transition label a.

  For each $a \in \bigcup_i \Sigma_i$, let

  $$active(a) = \{P_i \mid a \in \Sigma_i\}$$

  and let

  $$T(P_i, a) = \{t \in T_i \mid \text{the label of } t \text{ is } a\}$$

  Then, with $k = |active(a)|$, for each

  $$(t_{\alpha_1}, \ldots, t_{\alpha_k}) \in \prod_{P_i \in active(a)} T(P_i, a)$$

  there exists a transition $t \in \mathcal{T}$ such that

  - $n(t) = a$,
  - $p(t) = active(a)$,
  - $\ell_o(t, P_i)$ is the origin of t_{α_i} and
  - $\ell_d(t, P_i)$ is the destination of t_{α_i}.

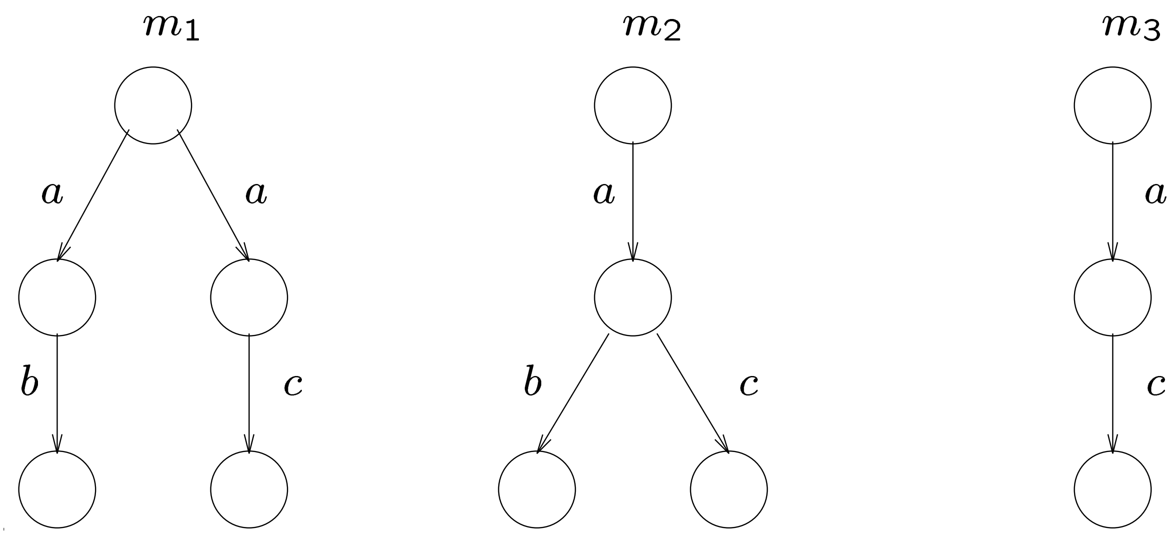
- τ-labeled transitions are simply added unmodified to the FCS.

Equivalence Notions for Processes

Our goal is to define a notion of equivalence such that for two processes to be equivalent, they have to have the same behavior whatever other process (environment) they are combined with.

For instance, defining equivalence as having the same sequences of possible transitions is not sufficient.

Example:



$P_1 \parallel P_3$ has a deadlock that does not appear in $P_2 \parallel P_3$.

A formalization of our requirement is that if $P_1 \equiv P_2$, then for all P $P_1 \parallel P \equiv P_2 \parallel P$, i.e. the equivalence is a congruence.

Bisimulation

Bisimulation is an equivalence notion that requires that, at any time during the execution of the processes, the set of possible transition labels (*actions*) must be identical for both processes.

Consider two processes $P_1 = (S_1, \Sigma_1, s_{01}, T_1)$ and $P_2 = (S_2, \Sigma_2, s_{02}, T_2)$. A bisimulation between P_1 and P_2 is a relation $R \subseteq S_1 \times S_2$ satisfying the three following conditions.

1. $R(s_{01}, s_{02})$.
2. Consider two states s_1 and s_1' of S_1 such that $(s_1, \lambda, s_1') \in T_1$, with $\lambda \in \Sigma_1 \cup \{\tau\}$. For each state $s_2 \in S_2$ such that $R(s_1, s_2)$, there exists a state $s_2' \in S_2$ such that $(s_2, \lambda, s_2') \in T_2$ and $R(s_1', s_2')$.

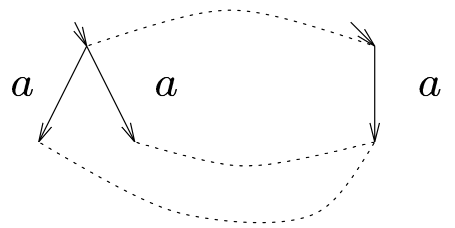
   If P_1 can execute an action, then P_2 can also execute that action

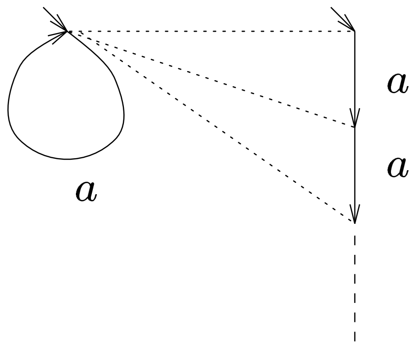
3. Consider two states s_2 et s'_2 of S_2 such that $(s_2, \lambda, s'_2) \in T_2$, with $\lambda \in \mathbf{\Sigma}_1 \cup \{\tau\}$. For each state $s_1 \in S_1$ such that $R(s_1, s_2)$, there exists a state $s'_1 \in S_1$ such that $(s_1, \lambda, s'_1) \in T_1$ and $R(s'_1, s'_2)$.

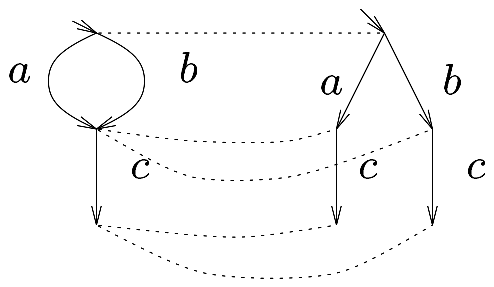
   If P_2 can execute an action, then P_1 can also execute that action

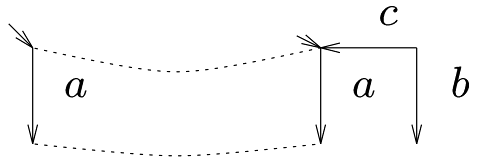
Examples

Equivalent processes.

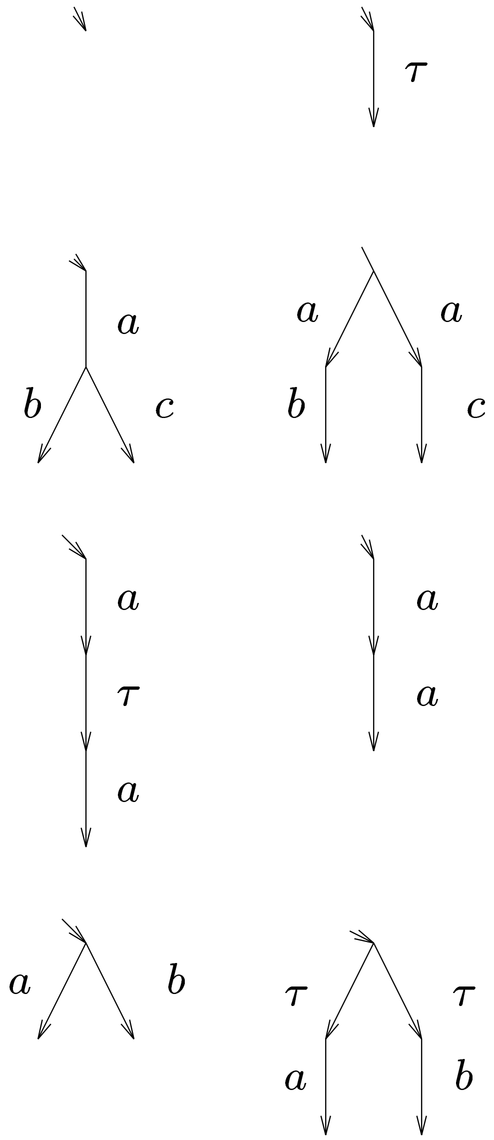








Nonequivalent processes.



Properties of Bisimulation

Bisimulation is strong equivalence that has a number of important properties.

- Bisimulation is a congruence with respect to parallel composition (and also other process composition operators)
- Bisimulation equivalent processes satisfy the same formulas of CTL extended with action names.
- Furthermore, two processes are bisimilar if they satisfy the same (extended) CTL formulas.

On the other hand, bisimulation is sometimes too strong an equivalence. Indeed, it distinguishes processes that only differ by internal actions. This motivates another notion of equivalence that distinguishes internal and other actions : *observational equivalence*.

We first introduce some notation. For a process $P = (S, \Sigma, s_0, T)$ and two states s and s' of S, we will write

$$s \stackrel{a}{\Longrightarrow} s'$$

if there exists $i, j \geq 0$ such that

$$s \underbrace{\stackrel{\tau}{\rightarrow}\stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow}}_{i} \stackrel{a}{\Longrightarrow} \underbrace{\stackrel{\tau}{\rightarrow}\stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow}}_{j} s'$$

And, we will write

$$s \stackrel{\tau}{\Longrightarrow} s'$$

if there exists $i \geq 0$ such that

$$s \underbrace{\stackrel{\tau}{\rightarrow}\stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow}}_{i} s'$$

Observational Equivalence

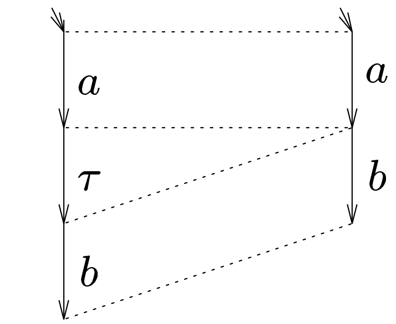
Consider two processes $P_1 = (S_1, \Sigma_1, s_{01}, T_1)$ and $P_2 = (S_2, \Sigma_2, s_{02}, T_2)$. An observational equivalence between P_1 et P_2 is a relation $R \subseteq S_1 \times S_2$ that satisfies the three following conditions.

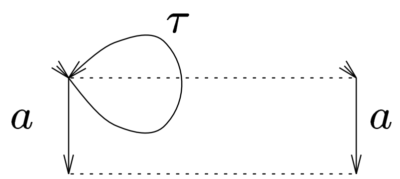
1. $R(s_{01}, s_{02})$.
2. Consider two states s_1 et s_1' of S_1 such that $s_1 \stackrel{\lambda}{\Longrightarrow} s_1'$. For any state $s_2 \in S_2$ such that $R(s_1, s_2)$, there exists a state $s_2' \in S_2$ such that $s_2 \stackrel{\lambda}{\Longrightarrow} s_2'$ and $R(s_1', s_2')$.
3. Consider two states s_2 et s_2' of S_2 such that $s_2 \stackrel{\lambda}{\Longrightarrow} s_2'$. For any state $s_1 \in S_1$ such that $R(s_1, s_2)$, there exists a state $s_1' \in S_1$ such that $s_1 \stackrel{\lambda}{\Longrightarrow} s_1'$ and $R(s_1', s_2')$.

The definition is similar to the one of bisimulation, except that one considers actions surrounded by arbitrary numbers of τ actions.

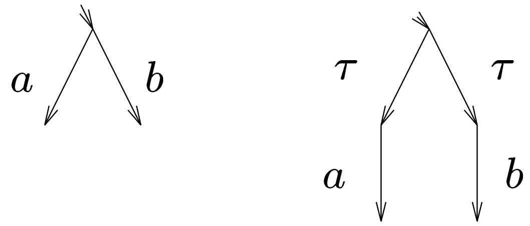
Examples

Observationally equivalent processes.





Observationally nonequivalent processes.



Checking Bisimulation and Observational Equivalence

The following algorithm checks whether two processes $P_1 = (S_1, \Sigma_1, s_{01}, T_1)$ and $P_2 = (S_2, \Sigma_2, s_{02}, T_2)$ are bisimilar. It operates on the product $R = S_1 \times S_2$ of the sets of states of the processes and repeatedly eliminates pairs that cannot be part of a bisimulation relation.

- Initialize R to $S_1 \times S_2$.
- Repeat until stabilization
  - If there exists a pair $(s_1, s_2) \in R$ such that either
    - $(s_1, \lambda, s_1') \in T_1$ with $\lambda \in \Sigma_1 \cup \{\tau\}$ and there is no $s_2' \in S_2$ such that $(s_2, \lambda, s_2') \in T_2$ and $(s_1', s_2') \in R$, or
    - $(s_2, \lambda, s_2') \in T_2$ with $\lambda \in \Sigma_2 \cup \{\tau\}$ and there is no $s_1' \in S_1$ such that $(s_1, \lambda, s_1') \in T_1$ and $(s_1', s_2') \in R$,
  - then $R := R \setminus \{(s_1, s_2)\}$.
- P_1 and P_2 are bisimilar iff $(s_{01}, s_{02}) \in R$

With a careful use of data structures, this algorithm can be implemented in time $O(|P_1| \times |P_2|)$ (Andersen, Vergauwen & Lewi).

There is a more efficient algorithm, $O((|P_1| + |P_2|) \log(|P_1| + |P_2|))$, that works by considering the union of the transition system and reduces the problem to a graph partitioning problem for which an efficient algorithm exists (Paige and Tarjan).

To check observational equivalence, one proceeds similarly, except that one considers the relations $s_1 \stackrel{a}{\Longrightarrow} s_1'$ and $s_2 \stackrel{a}{\Longrightarrow} s_2'$ instead of T_1 and T_2. The complexity of this is higher since it requires computing the τ^* relation.

Implementation Relations

- When replacing a process P_1 by a simpler process P_1' in order to do verification, requiring that the two processes be equivalent is often too restrictive.
- Indeed, P_1' will often be a more abstract and general *specification* process of which P_1 is a more concrete *implementation*.
- What is required is that what the implementation can do be allowed by the specification. The converse is not necessary.
- We thus introduce asymmetrical versions of bisimulation (simulation) and observational equivalence (observational simulation).
- We will write $P_1 \leq_s P_1'$ if P_1 can be simulated by P_1' and $P_1 \leq_{ob} P_1'$ if P_1 can be observationally simulated by P_1'
- The requirement on simulation relations is that they be monotonic with respect to process combination operators: if $P_1 \leq P_1'$, then for all P $P_1 \parallel P \leq P_1' \parallel P$

Simulation

Consider two processes $P_1 = (S_1, \Sigma_1, s_{01}, T_1)$ and $P_2 = (S_2, \Sigma_2, s_{02}, T_2)$. A simulation of P_1 by P_2 ($P_1 \leq_s P_2$) is a relation $R \subseteq S_1 \times S_2$ satisfying the two following conditions.

1. $R(s_{01}, s_{02})$.
2. Consider two states s_1 and s'_1 of S_1 such that $(s_1, \lambda, s'_1) \in T_1$, with $\lambda \in \Sigma_1 \cup \{\tau\}$. For each state $s_2 \in S_2$ such that $R(s_1, s_2)$, there exists a state $s'_2 \in S_2$ such that $(s_2, \lambda, s'_2) \in T_2$ and $R(s'_1, s'_2)$.

   If P_1 can execute an action, then P_2 can also execute that action

Observational simulation ($P_1 \leq_{ob} P_2$) is defined similarly, except that one uses the relations $s_1 \overset{\lambda}{\Longrightarrow} s'_1$ and $s_2 \overset{\lambda}{\Longrightarrow} s'_2$ instead of T_1 and T_2.

The algorithm for checking (observational) simulation is an adaptation of the one for checking (observational) equivalence.

An Application:
Verifying Systems with an Arbitrary Number of Identical Processes

- The problem is to verify a system in which the same process can appear any number of times.

- A typical example is a protocol in which the number of participants is not fixed a priori.

A Simple Approach

Use an induction!

To prove that $\|^n P$ satisfies S:

- Find a property I such that:
  - P satisfies I
  - $I \parallel P$ satisfies I
  - I satisfies S
- I is called a *network invariant*.

The Framework

- Set of processes $\mathbf{P}$

- Relation $\leq$ on $\mathbf{P}$ such that:
  $P_1 \leq P_2$ iff P_1 implements P_2 (for instance, observational simulation)

- *Monotonic* (w.r.t. $\leq$) operations defined on $\mathbf{P}$:

  - Parallel composition, notation $P \parallel_A Q$ for parallel composition in which synchronization is restricted to the actions in A;

  - Renaming (changing the name of an action), notation $P[a \leftarrow b]$ for P in which a is replaced by b;

  - Hiding (changing an action into the internal action τ), notation $P \backslash a$ for P in which a is replaced by τ.

Linear Networks

Consider the system

$$N_n = P_0 \odot \underbrace{P \odot P \odot \ldots \odot P}_{n}$$

where $\odot$ represents some combination of parallel composition, renaming and hiding.

To prove that $\forall n$, $N_n \leq S$

- Find a *network invariant* I
- Verify that:
  - $P_0 \leq I$
  - $I \odot P \leq I$
  - $I \leq S$

Circular Networks

Consider the system

$$C_n = P_\ell \odot_\ell \underbrace{P \odot P \odot \ldots \odot P}_{n}.$$

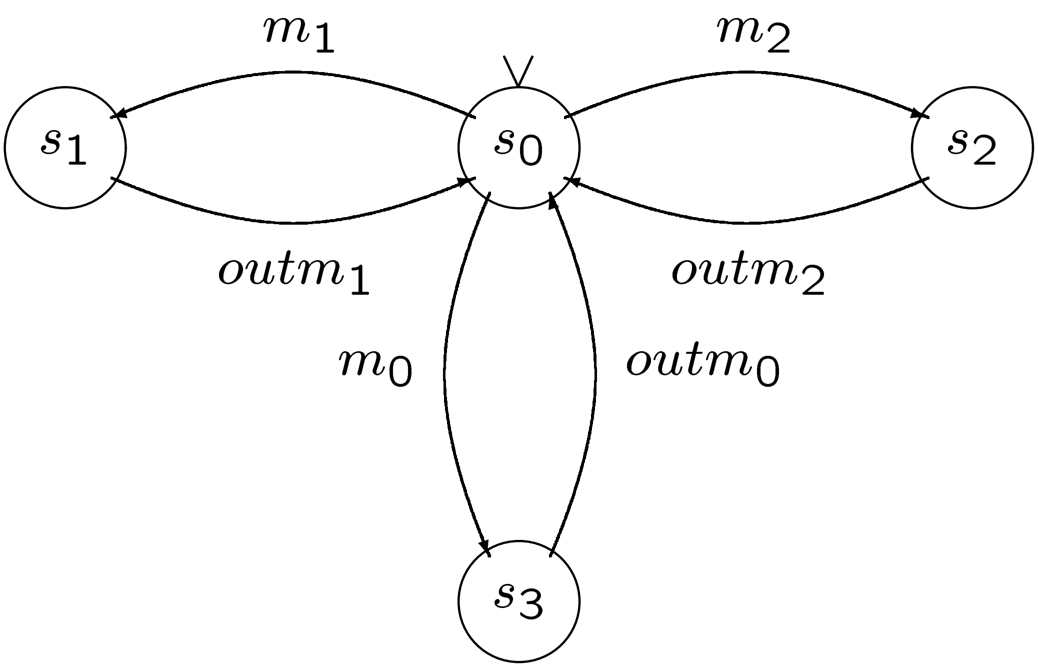
To prove that $\forall n$, $C_n \leq S$

- Find a *network invariant* I
- Verify that:
  - $P \leq I$
  - $I \odot P \leq I$
  - $P_\ell \odot_\ell I \leq S$

First Example: A buffer

$$P_n = \underbrace{P \odot P \odot \ldots \odot\ P}_{n}$$

where P is



and

$$P \odot P = (P[outm_i \leftarrow lk_i] \parallel_{\{lk_i\}} P[[m_i \leftarrow lk_i]) \backslash \{lk_i\}$$

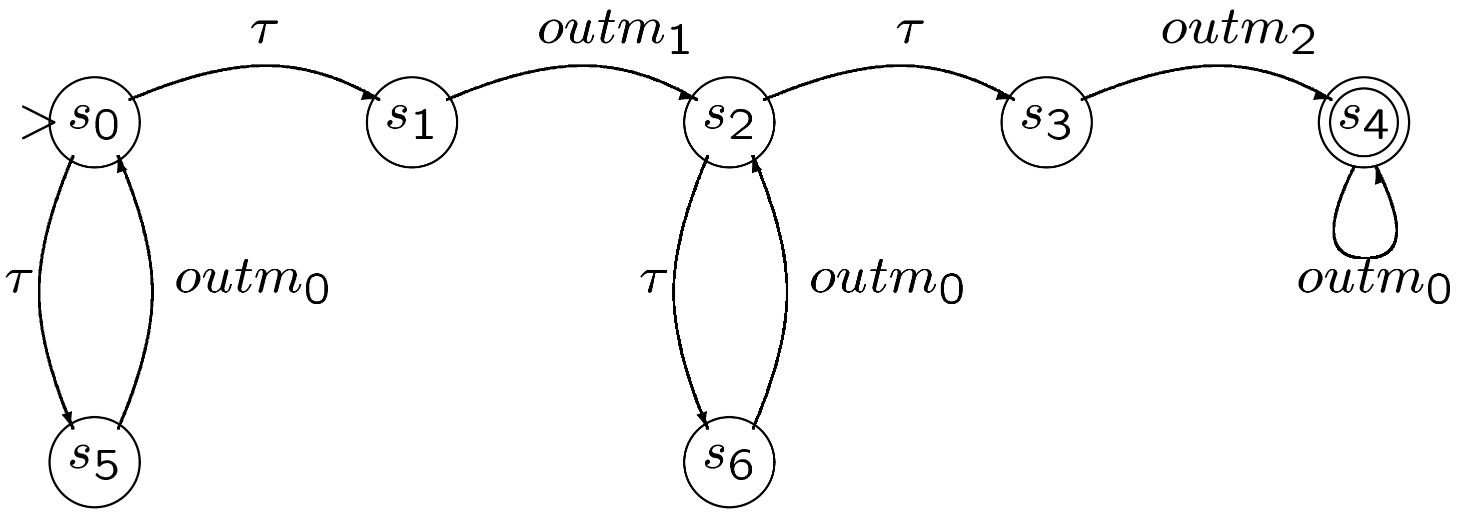
$(i = 0, 1, 2)$

Using the concept of *data independence*, it can be proved that:

P_n behaves like an n-place buffer iff

$$P_0 \odot P_n \leq P_0$$

where P_0 is



The network invariant is $I = P_0$.

The necessary conditions are satisfied:

$$P_0 \leq I$$

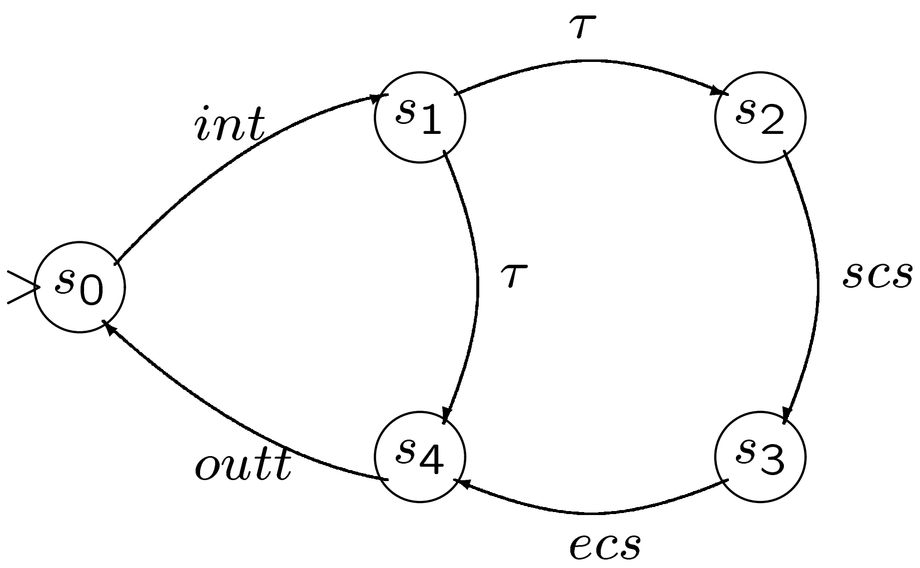
and

$$I \odot P \leq I$$

Second Example: A token-passing mutual exclusion protocol

$$C_n = P_\ell \odot_\ell \underbrace{[P \odot P \odot \ldots \odot P]}_{n}$$

where P and $P\ell$ are

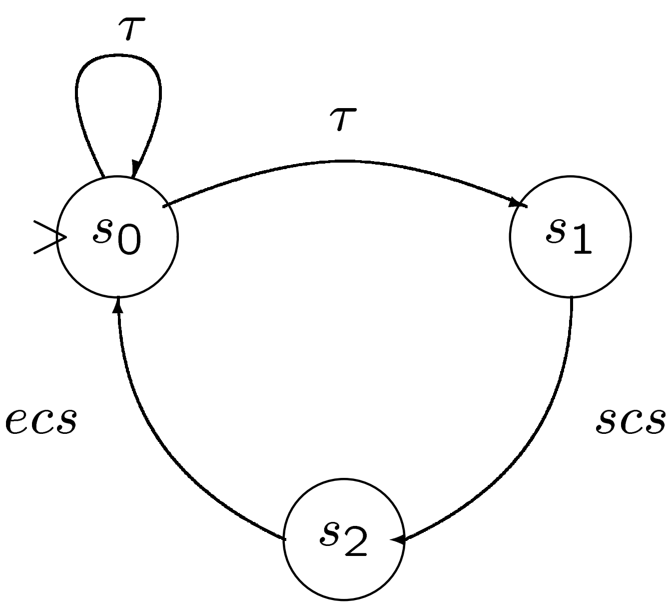


$$P \odot P = [P[outt \leftarrow lk] \parallel_{\{lk\}} P[int \leftarrow lk]] \backslash \{lk\}$$

and

$$\begin{aligned} P_\ell \odot_\ell Q \;\; &= \;\; [P_\ell[outt \leftarrow int, int \leftarrow outt] \\ &\quad\;\; \parallel_{\{int,outt\}} Q] \backslash \{int, outt\} \end{aligned}$$

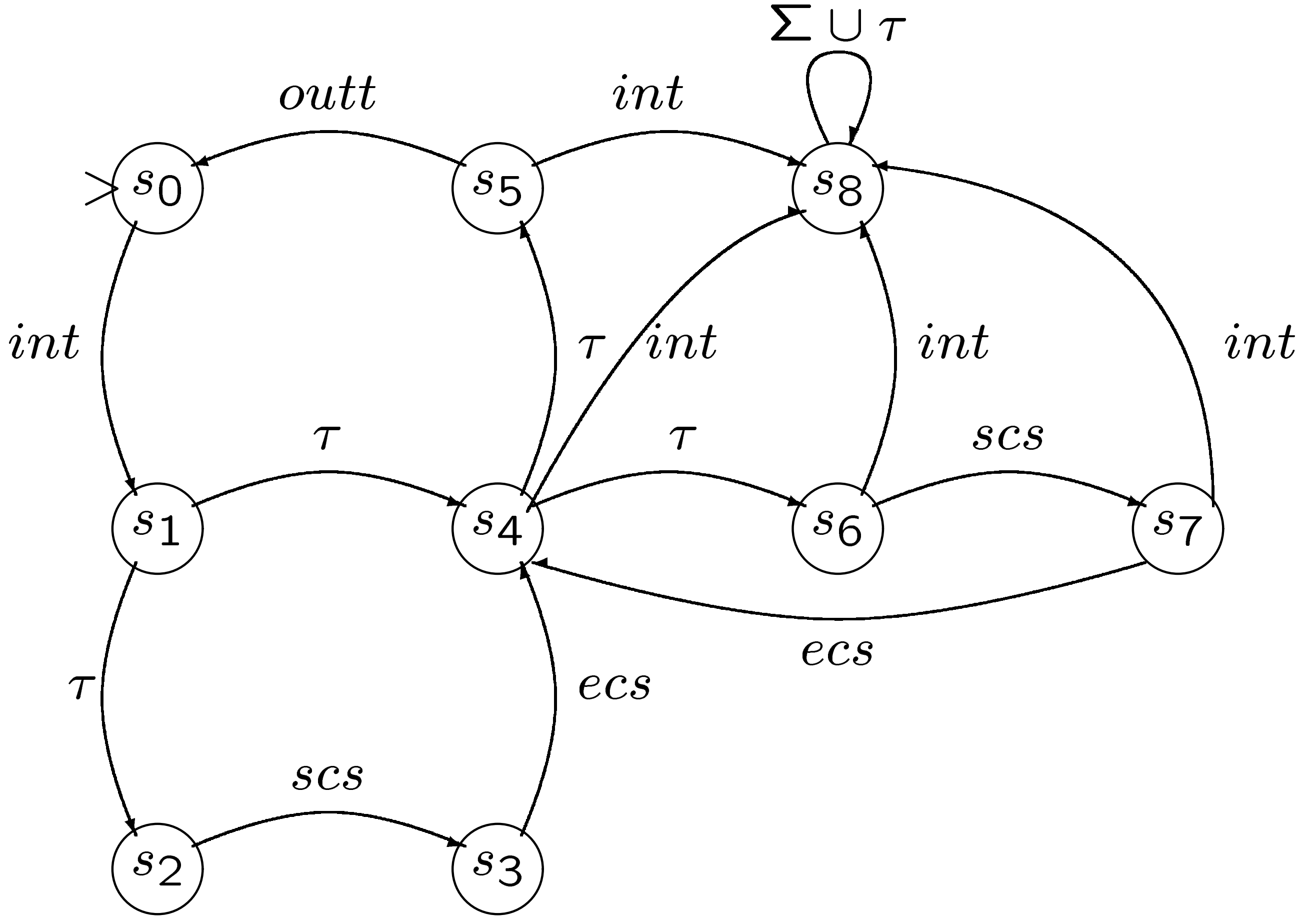
The specification is



The invariant should satisfy

$$P \leq I$$
$$I \odot P \leq I$$
$$P_\ell \odot_\ell I \leq S$$

The invariant is



Conclusions

- Method is *algorithmic* once invariant has been found
- Bad news: a finite-state invariant does not always exist
- The method verifies a whole family of systems, each of which is finite-state but for which there is no global bound on the number of states.
- This moves us towards the topic of next week: verifying systems with infinite state spaces.