The Algorithmic Verification of Reactive Systems

Lecture 2:

Modelling Concurrent and Reactive Systems

Motivation

- A first and fundamental question is: *what do we verify?*
- The answer must be "programs"! But, in which language?
- To avoid this language issue, we introduce a *modelling language* with the following characteristics.
  - It is sufficiently simple to be easily manipulated by verification algorithms.
  - It can model concurrency and the most common process communication and synchronization mechanisms.
  - Usual programming languages can be compiled (translated) into the modelling language.
- Verification algorithms will all be given for the modelling language.
- Any language for which a compiler to the modelling language exists can then be handled.

A modelling Language: Formal Concurrent Systems

The main characteristics of this language are the following.

- Only systems with a fixed number of *processes* are considered.
- Each process has a finite number of *control states*.
- The processes have access to a *shared memory* represented by a fixed number of variables.
- The actions of the system are described by *transitions* that specify
  - A change of control state for one (simple transition) or several (joint transition) processes.
  - A *condition* (guard) on the variable values. This condition must be true for the transition to be enabled.
  - A set of *simultaneous assignments* that describe a modification of the memory content.

Definition A Formal Concurrent System (FCS) is defined by the following items.

- A finite set $\mathcal{P}$ of processes.
- For each process $p_i \in \mathcal{P}$, a finite set $\ell(p_i)$ of control states (or *locations*, as well as an *initial location* $\ell_0^{p_i} \in \ell(p_i)$. The sets of locations of distinct processes must be disjoint, i.e. for $i \neq j$, $\ell(p_i) \cap \ell(p_j) = \emptyset$.
- A *memory* $\mathcal{M}$ consisting of a finite set of *variables* and a function $\mathcal{I}$ assigning an initial value to each variable.
- A finite set $\mathcal{T}$ of transitions.

Each transition $t \in \mathcal{T}$ is defined by

- A function $n(t)$ giving a name (an identifier) to the transition.
- The set $p(t) \subseteq \mathcal{P}$ of processes that are *active* for the transition.
- A source location $\ell_s(t, p_i) \in \ell(p_i)$ for each process $p_i \in p(t)$.
- A destination location $\ell_d(t, p_i) \in \ell(p_i)$ for each process $p_i \in p(t)$.
- A Boolean condition $C(t)$ built from predicates applied to the variables of $\mathcal{M}$.
- A set of assignments $A(t)$ operating on variables of $\mathcal{M}$.

Variables, Conditions and Assignments

Just as in a programming language, a variable represents a memory location. What are the allowed types for the variables?

This question is not crucial. In practice, we can just adopt the type system of an existing programming language. In examples, only common simple types will be used (integer, boolean, array, ...).

Similarly, the exact form of Boolean conditions.will not be specified. They can be any Boolean combination of computable predicates operating on the types of variables being used.

Finally, a simultaneous assignment is of the following form

$$(var_1, \ldots, var_n) := (exp_1, \ldots, exp_n).$$

Each expression exp_i must have the type of the corresponding variable var_i. The evaluation rule is that all expressions are first evaluated and that thereafter all assignments are performed.

Notation conventions

- Processes of $\mathcal{P}$ are denoted by $p_1, \ldots, p_n$ (or $p, q, \ldots$ if there are few of them).

- The locations of a process p are denoted by $\ell_0^p, \ldots, \ell_{k_p}^p$. By convention, the initial location of a process p is ℓ_0^p.

- The memory variables are denoted by $x_1, \ldots, x_m$ (alternatively by $x, y, \ldots$ or by any other set of identifiers).

- A transition t will be denoted by

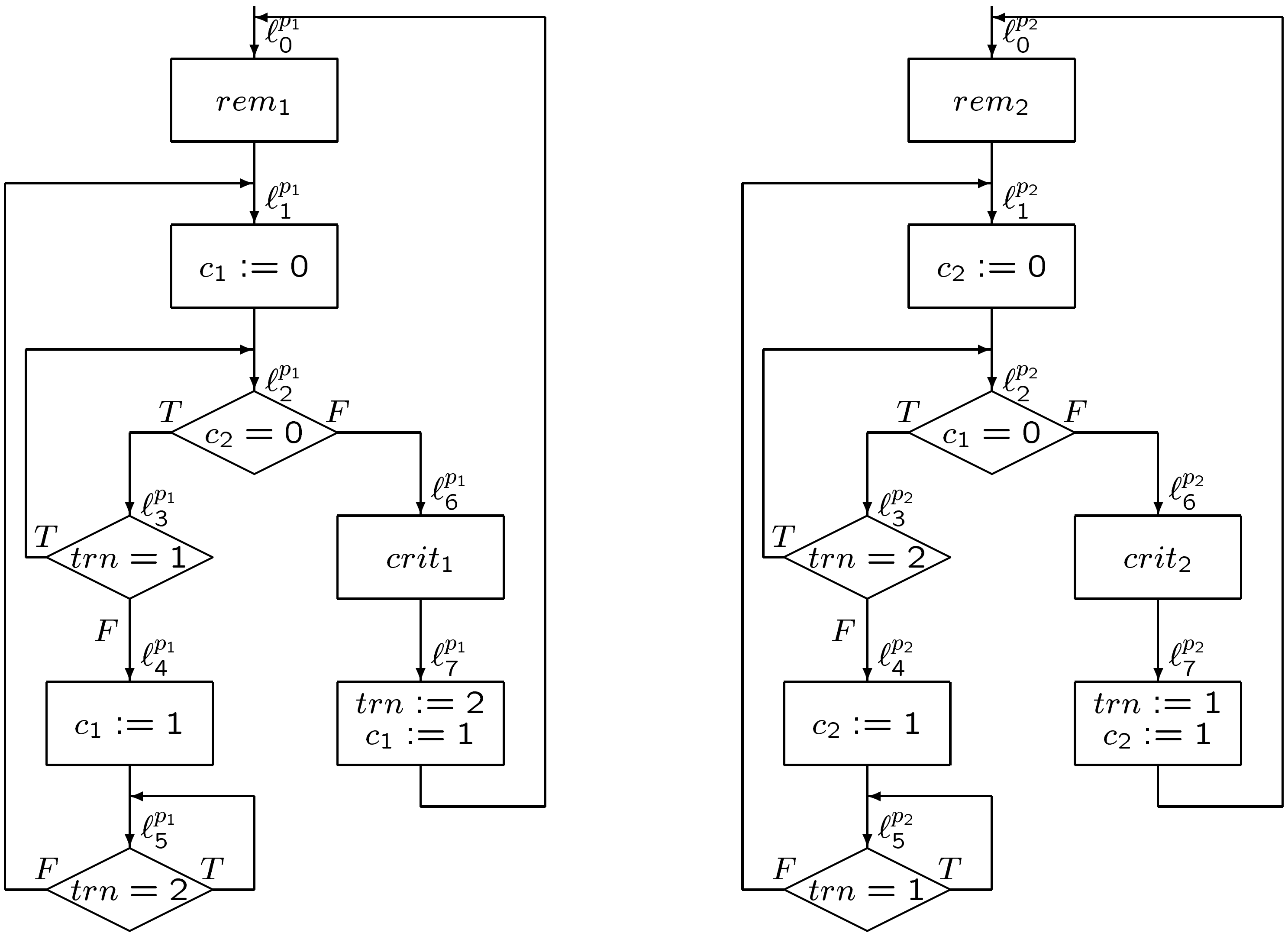
  $$name, active : (sources, C \rightarrow A, destinations)$$

  where $name$ is the name of the transition, $active$ is the list of processes that are active for the transition, $sources$ is the list of source locations for the transition, and $destinations$ is the list of destination locations of the transition.

An Example: Dekker's Algorithm

Initializations:

$c_1 := 1; c_2 := 1; trn := 1$



Dekker's Algorithm as a FCS

- The processes : $\mathcal{P} = \{p_1, p_2\}$,

  $\ell(p_1) = \{\ell_0^{p_1}, \ell_1^{p_1}, \ell_2^{p_1}, \ell_3^{p_1}, \ell_4^{p_1}, \ell_5^{p_1}, \ell_6^{p_1}, \ell_7^{p_1}\}$

  $\ell(p_2) = \{\ell_0^{p_2}, \ell_1^{p_2}, \ell_2^{p_2}, \ell_3^{p_2}, \ell_4^{p_2}, \ell_5^{p_2}, \ell_6^{p_2}, \ell_7^{p_2}\}$

- The memory : $\mathcal{M} = \{c_1, c_2, T\}$

  $\mathcal{I}(c_1) = 1$, $\mathcal{I}(c_2) = 1$, $\mathcal{I}(T) = 1$

- The transitions :

  rem_1, p_1 : $(\ell_0^{p_1}, \mathbf{true} \to () := (), \ell_1^{p_1})$

  $t_{1,2}, p_1$: $(\ell_1^{p_1}, \mathbf{true} \to (c_1) := (0), \ell_2^{p_1})$

  $t_{1,3}, p_1$: $(\ell_2^{p_1}, c_2 = 1 \to () := (), \ell_6^{p_1})$

  $crit_1, p_1$: $(\ell_6^{p_1}, \mathbf{true} \to () := (), \ell_7^{p_1})$

  $t_{1,5}, p_1$: $(\ell_7^{p_1}, \mathbf{true} \to (T, c_1) := (2, 1), \ell_0^{p_1})$

  $t_{1,6}, p_1$: $(\ell_2^{p_1}, c_2 = 0 \to () := (), \ell_3^{p_1})$

  $t_{1,7}, p_1$: $(\ell_3^{p_1}, T = 1 \to () := (), \ell_2^{p_1})$

  $t_{1,8}, p_1$: $(\ell_3^{p_1}, T = 2 \to () := (), \ell_4^{p_1})$

  $t_{1,9}, p_1$: $(\ell_4^{p_1}, \mathbf{true} \to (c_1) := (1), \ell_5^{p_1})$

  $t_{1,10}, p_1$: $(\ell_5^{p_1}, T = 2 \to () := (), \ell_5^{p_1})$

  $t_{1,11}, p_1$: $(\ell_5^{p_1}, T = 1 \to () := (), \ell_1^{p_1})$

$$
\begin{aligned}
rem_2, p_2 :&\quad (\ell_0^{p_2}, \mathbf{true} \rightarrow () := (), \ell_1^{p_2}) \\
t_{2,2}, p_2 :&\quad (\ell_1^{p_2}, \mathbf{true} \rightarrow (c_2) := (0), \ell_2^{p_2}) \\
t_{2,3}, p_2 :&\quad (\ell_2^{p_2}, c_1 = 1 \rightarrow () := (), \ell_6^{p_2}) \\
crit_2, p_2 :&\quad (\ell_6^{p_2}, \mathbf{true} \rightarrow () := (), \ell_7^{p_2}) \\
t_{2,5}, p_2 :&\quad (\ell_7^{p_2}, \mathbf{true} \rightarrow (T, c_2) := (1, 1), \ell_0^{p_2}) \\
t_{2,6}, p_2 :&\quad (\ell_2^{p_2}, c_1 = 0 \rightarrow () := (), \ell_3^{p_2}) \\
t_{2,7}, p_2 :&\quad (\ell_3^{p_2}, T = 2 \rightarrow () := (), \ell_2^{p_2}) \\
t_{2,8}, p_2 :&\quad (\ell_3^{p_2}, T = 1 \rightarrow () := (), \ell_4^{p_2}) \\
t_{2,9}, p_2 :&\quad (\ell_4^{p_2}, \mathbf{true} \rightarrow (c_2) := (1), \ell_5^{p_2}) \\
t_{2,10}, p_2 :&\quad (\ell_5^{p_2}, T = 1 \rightarrow () := (), \ell_5^{p_2}) \\
t_{2,11}, p_2 :&\quad (\ell_5^{p_2}, T = 2 \rightarrow () := (), \ell_1^{p_2})
\end{aligned}
$$

A Semantics for Formal Concurrent Systems

An FCS is a program expressed in a very simple language. Our goal is to analyze the possible behaviors of this program.

For doing this we need to define the rules that tell us what these possible behaviors are, i.e. to give a *semantics* to FCSs.

Note: To give a *semantics* to a language or formalism is to give a way of obtaining the meaning of a system described in this language.

However, this meaning has to itself be described in some language. A semantics can thus be viewed as a translation from one language to another. For this translation to be useful, the target language has to be simpler than the source language or, at least, it should make it easier to obtain information about the system one started with.

The semantics we will use is a translation to *transition systems* (or automata), which will enable us to directly analyze the behaviors of the system.

Transition Systems

A transition system is defined by the following elements:

- A set of labels or *alphabet* Σ, i.e. a finite set of distinct symbols;
- A set of *states* S, this set can be finite of infinite;
- An initial state $s_0 \in S$;
- A set $T \subseteq S \times \Sigma \times S$ of transitions.

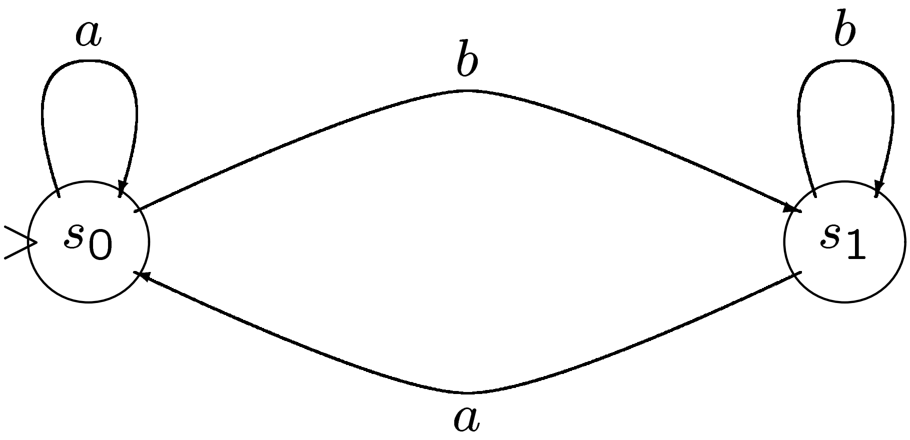
Each transition is thus a triple (s_1, σ, s_2) containing

- a source state s_1,
- a label σ et
- a destination state s_2.

Representation and Example

If it has a finite number of states, a transition system can be represented by a graph. Each state is represented by a node and each transition by a labeled edge.

Example



The Behaviors of a Transition System

A transition system can itself be given a semantics which is a set of *behaviors*. An infinite behavior σ of a transition system $(\Sigma, S.s_0, T)$, is a pair of functions $\sigma_S : \mathrm{N} \rightarrow S$ and $\sigma_T : \mathrm{N} \rightarrow T$, often written as

$$\sigma_S(0) \stackrel{\sigma_T(0)}{\rightarrow} \sigma_S(1) \stackrel{\sigma_T(1)}{\rightarrow} \sigma_S(2) \stackrel{\sigma_T(2)}{\rightarrow} \ldots$$

such that

- $\sigma_S(0) = s_0$ and
- $\forall i \geq 0 \ (\sigma_S(i), \sigma_T(i), \sigma_S(i+1)) \in T$.

Finite behaviors can be similarly defined.

The Transition System corresponding to a Formal Concurrent System

Let $FCS = \{\mathcal{P}, \mathcal{M}, \mathcal{T}\}$ be a formal concurrent system. The corresponding transition system $TS = \{\Sigma, S, s_0, T\}$ is defined as follows.

- The alphabet Σ is $\{n(t) \mid t \in \mathcal{T}\}$.
- If $\mathcal{P} = \{p_1, \ldots, p_n\}$, $\mathcal{M} = \{x_1, \ldots, x_m\}$ and if the set of possible values of a variable x_i is V_i, then

  $$S = \ell(p_1) \times \cdots \times \ell(p_n) \times V_1 \times \cdots \times V_m.$$

  In other words, each state of S is a tuple consisting of a control state for each process and a value for each variable of the memory. For a state $s \in S$, the location of p_i in s is denoted by $s(p_i)$ and the value of the variable x_i in s by $s(x_i)$. That is,

  $$s = (s(p_1), \ldots, s(p_n), s(x_1), \ldots, s(x_m))$$

- The initial state is given by

$$s_0 = (\ell_0^{p_1}, \ldots, \ell_0^{p_n}, \mathcal{I}(x_1), \ldots, \mathcal{I}(x_m))$$

where $\mathcal{I}$ is the function giving the initial values of the memory.

- For each transition $t \in \mathcal{T}$ and for each state $s \in S$, if

  - for each process p_i that is active for t, $s(p_i) = \ell_s(t, p_i)$ and if
  - the condition $C(t)$ is satisfied for the values $(s(x_1), \ldots, s(x_m))$ of the variables in state s

  Then, the transition t is said to be enabled in s and TS contains a transition labeled by $n(t)$ from the state s to the state s' defined as follows.

  - For each process p_i that is not active for t, $s'(p_i) = s(p_i)$,
  - For each process p_i active for t, $s'(p_i) = \ell_d(t, p_i)$,
  - For each variable x_i that is not modified by $A(t)$, $s'(x_i) = s(x_i)$, and
  - For each variable x_i to which $A(t)$ assigns an expression exp_i, $s'(x_i) = [exp_i]_s$ where $[exp_i]_s$ is the value of the of the expression exp_i in which the variables have been replaced by their value in s.

  Note that for a given state s and transition $t \in \mathcal{T}$ the state s' is unique.

Extending FCSs with Fairness

- The semantics of concurrent systems we have just given is an interleaving semantics, i.e, the possible behaviors of the transition system are obtained by interleaving the behaviors of the various processes.

- This semantics allows absolutely all possible interleavings, including some that practical intuition would rule out, e.g. interleavings in which some process is deliberately ignored even though it could progress.

- In this case, the practical intuition is correct and one needs to correct our model in order to rule out some behaviors that are in some sense "unfair". This is the purpose of *fairness assumptions*.

- Without fairness assumptions, it would be impossible to prove that a concurrent system satisfies any property that requires the progress of several processes.

- Fairness assumptions are qualitative restrictions on behaviors. They impose that each process eventually progresses, without any quantitative restriction. They thus can only be defined on infinite behaviors.
- A fairness assumption can be viewed as an abstraction of particular process scheduling policies.
- A single fairness assumption models a whole range of scheduling policies. If it is used to verify a property, it guarantees that any implementation that satisfies the fairness assumption will make the property true.
- There is a whole range of possible fairness assumptions. One of the least restrictive is *weak fairness*

Weak Fairness

We will define weak fairness at the level of processes (an alternative is to define it at the level of transitions).

It is convenient to first describe the negation of weak fairness.

> A behavior is not weakly fair, if for some process p and from some point on, there is always an enabled transition in which p is active and no transition in which p is active is taken.

A positive formulation of weak fairness is the following.

> A behavior is weakly fair if, for every process p, it is the case that infinitely often either no transition in which p is active is enabled, or a transition in which p is active is executed.

Weak fairness basically says that each process should infinitely often be given a chance to progress. If when given this chance it cannot progress, nothing needs to be guaranteed.

Modelling Fairness assumptions: Observer Processes

An observer process is a process that "observes" but does not modify the behavior of a Formal Concurrent System. It monitors the behavior of the FCS and can be used to check some properties, e.g. whether behaviors are fair or not.

An observer process is similar to a process, except that it has a different type of transitions: they don't contain an assignment and their conditions are more general.

An observer process ob for a formal concurrent system $FCS = \{\mathcal{P}, \mathcal{M}, \mathcal{T}\}$ is defined by the following items:

- A finite set $\ell(ob)$ of *locations* (control states) as well as an initial location $\ell_0^{ob} \in \ell(ob)$.
- A finite set $\mathcal{T}_{ob}$ of transitions.

A transition $\tau \in \mathcal{T}_{ob}$ of an observer process is defined by the following items.

- A source location $\ell_s(\tau) \in \ell(ob)$.
- A destination location $\ell_d(\tau) \in \ell(ob)$.
- A Boolean condition $C(\tau)$ expressed in terms of
  - a location variable loc_{p_i} for each process $p_i \in \mathcal{P}$,
  - the variables $x_1, \ldots, x_m$ de $\mathcal{M}$,
  - a variable tr whose value is en element of the set of transitions $\mathcal{T}$ of FCS.

  We will write $C(\tau)[loc_{p_1}, \ldots loc_{p_n}, x_1, \ldots x_m, tr]$ when we wish to make explicit the variables used in $C(\tau)$.

The Conditions of the Transitions of Observer Processes

These conditions are Boolean formulas built from atomic formulas of the following form (non exhaustive list).

- Standard predicates applied to the variables of $\mathcal{M}$, for example $x_1 \leq x_2$.
- Equalities and inequalities between a location variable relative to a process and a location of that process, for example $loc_{p_1} = \ell_1^{p_1}$.
- Comparisons between the elements defining the transition tr and the elements used in defining the FCS, for example $n(tr) = t_{1,1}$, $p_i \in p(tr)$, or $\ell_s(tr, p_i) = \ell_j^{p_i}$.

An example

In the case of Dekker's algorithm, an observer process ob whose role is to detect if the transition $crit_1$ can be immediately followed by the transition $crit_2$, can be defined as follows.

- The locations:$\ell(ob) = \{\ell_0^{ob}, \ell_1^{ob}, \ell_2^{ob}\}$

- The transitions (written as $(\ell_s, condition, \ell_d)$):

  $(\ell_0^{ob}, n(tr) = crit_1, \ell_1^{ob})$

  $(\ell_1^{ob}, n(tr) = crit_2, \ell_2^{ob})$

  $(\ell_1^{ob}, n(tr) \neq crit_2, \ell_0^{ob})$

The problem of checking whether $crit_1$ can be immediately followed by $crit_2$ (absence of mutual exclusion) then amounts to checking whether the process ob can reach the location ℓ_2^{ob}.

Composing an Observer Process and a Formal Concurrent System

When building the transition system corresponding to a FCS, an observer process should not be handled in the same way as the other processes. Indeed, its transitions should not be interleaved with the transitions of the other processes, but synchronized with them.

Each time a FCS executes a transition, the observer should also execute a transition. If the observer has no enabled transition, it executes a default transition that keeps it in the same state.

Let $FCS = \{\mathcal{P}, \mathcal{M}, \mathcal{T}\}$ be a formal concurrent system and $ob = \{\ell(ob), \mathcal{T}_{ob}\}$ be an observer process. The corresponding transition system is $TS = \{\Sigma, S, s_0, T\}$, which is defined as follows.

- The alphabet Σ is $\{n(t) \mid t \in \mathcal{T}\}$.
- If $\mathcal{P} = \{p_1, \ldots, p_n\}$, $\mathcal{M} = \{x_1, \ldots, x_m\}$ and if the set of possible values for a variable x_i is V_i, then

  $S = \ell(p_1) \times \cdots \times \ell(p_n) \times \ell(ob) \times V_1 \times \cdots \times V_m.$

  In other words, each state of S is a tuple consisting of a control state for each process and for the observer process, as well as a value for each variable of the memory.
- The initial state is given by

$$s_0 = (\ell_0^{p_1}, \ldots, \ell_0^{p_n}, \ell_0^{ob}, \mathcal{I}(x_1), \ldots, \mathcal{I}(x_m))$$

- For each transition $t \in \mathcal{T}$ and for each state $s \in S$, if
  - for each process p_i that is active for t, $s(p_i) = \ell_s(t, p_i)$ and if
  - the condition $C(t)$ is satisfied for the values $(s(x_1), \ldots, s(x_m))$ of the variables in state s

  Then, the transition t is said to be enabled in s and TS contains a transition defined as follows.

For each transition τ of the observer process such that

$$\ell_s(\tau) = s(ob) \quad \text{and} \quad C(\tau)[s,t] = \mathbf{true}$$

(and for the default transition

$$\tau = (s(ob), \mathbf{true}, s(ob))$$

if no such transition exists), TS contains a transition labeled by $n(t)$ from the state s to the state s' defined by

- For each process p_i that is not active for t, $s'(p_i) = s(p_i)$,
- For each process p_i active for t, $s'(p_i) = \ell_d(t, p_i)$,
- For the observer process ob, $s'(ob) = \ell_d(\tau)$,
- For each variable x_i that is not modified by $A(t)$, $s'(x_i) = s(x_i)$, and
- For each variable x_i to which $A(t)$ assigns an expression exp_i, $s'(x_i) = [exp_i]_s$ where $[exp_i]_s$ is the value of the expression exp_i in which the variables have been replaced by their value in s.

For a given state s and a given transition $t \in \mathcal{T}$, the state s' is not necessarily unique The possibility of nondeterminism comes from the observer process.

Modelling Fairness Assumptions with Observers

A fairness assumption is a restriction on the infinite behaviors of a FCS. It can be modelled by an observer process that distinguishes between fair and unfair infinite behaviors.

However, this requires extending our notion of observer process (and hence of transition system) with a notion of acceptance condition.

A state s appears infinitely often in an infinite behavior σ of a transition system, if there exists an infinite set of points $i \in N$ such that $\sigma_S(i) = s$.

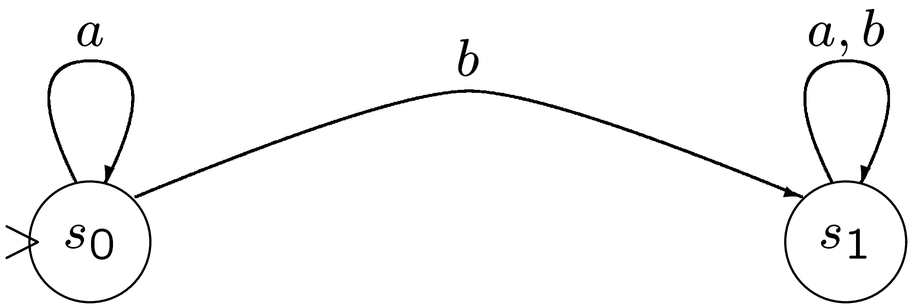
For a behavior σ, $inf(\sigma)$ represents the set of states of S that appear infinitely often in σ.

A generalized Büchi acceptance condition for the infinite behaviors of a transition system is a set $\mathbf{F} = \{F_1, \ldots F_k\}$ of subsets of the set of states S.

An infinite behavior σ of a transition system satisfies an acceptance condition $\mathbf{F} = \{F_1, \ldots F_k\}$ (is accepted) iff

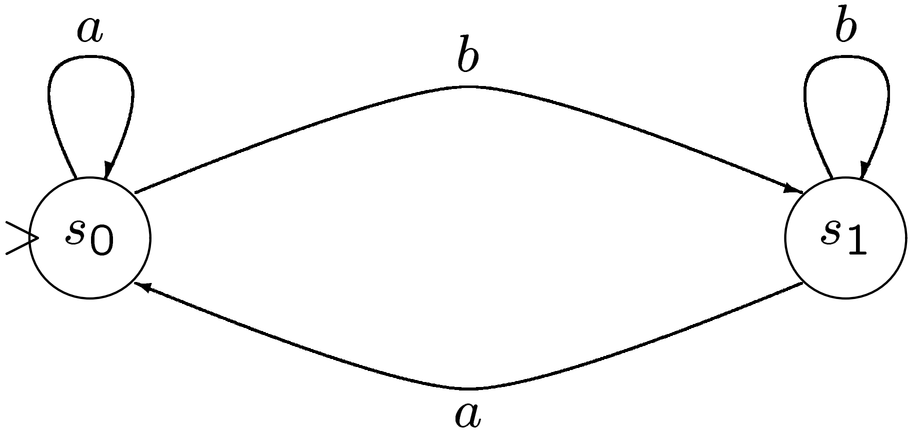
$$\forall F_i \in \mathbf{F} \text{ on a } \; inf(\sigma) \cap F_i \neq \emptyset$$

Examples :



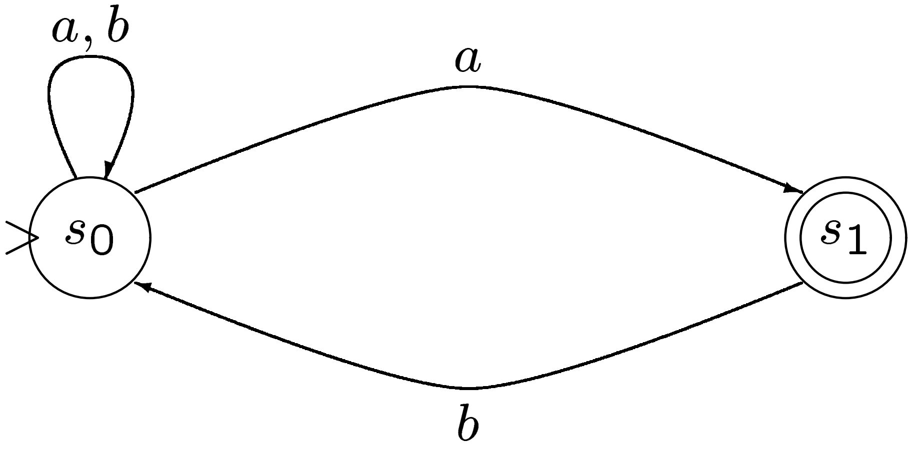
$$\mathbf{F} = \{\{s_1\}\}$$

The accepted behaviors are those in which b is executed at least once.



$$\mathbf{F} = \{\{s_0\}, \{s_1\}\}$$

The accepted behaviors are those in which the transitions a and b are both executed infinitely often.



$$\mathbf{F} = \{\{s_1\}\}$$

The accepted behaviors are those in which the pair of transitions ab is executed infinitely often.

An Observer for Weak Fairness

To impose weak fairness, we use a set of observer processes (one for each process of the FCS). To describe these observers, we use two predicates that can be evaluated a state s of the global transition system and a transition t of the FCS.

- $e(p)[s,t]$ is true if the process p is active for at least one transition that is enabled in s.
- $a(p)[s,t]$ is true if p is active for the transition t.

Weak fairness is then modelled by adding the following observer for each process of the FCS.

- The locations:$\ell(\textit{ob-p}) = \{\ell_0^{\textit{ob-p}}, \ell_1^{\textit{ob-p}}\}$

- The transitions:

  $(\ell_0^{\textit{ob-p}}, e(p) \wedge \neg a(p), \ell_1^{\textit{ob-p}})$
  $(\ell_1^{\textit{ob-p}}, \neg e(p) \vee a(p), \ell_0^{\textit{ob-p}})$

- The acceptance condition: $\mathbf{F}^{\textit{ob-p}} = \{\{\ell_0^{\textit{ob-p}}\}\}$

Note. The acceptance condition of an observer process yields an acceptance condition for the global transition system as follows.

For each set $F_i \in \mathbf{F}^{\textit{ob-p}}$, the acceptance condition $\mathbf{F}$ of the transition system contains the set

$$F_i^{\textit{ob-p}} = \{s \in S \mid s(\textit{ob-p}) \in F_i\}$$

Semaphores and Strong Fairness

A semaphore is an integer variable on which just two atomic operations are possible:

$$\begin{array}{lll} wait(s) & : & \textbf{if } s > \mathit{0} \textbf{ then } \; s := s - \mathit{1} \\ & & \qquad\qquad \textbf{else } \;\; \text{``}wait\text{''} \end{array}$$

$$signal(s) \quad : \quad s := s + \mathit{1}$$

To model a semaphore in a FCS, one can proceed as follows.

- A shared variable s is used
- A $wait(s)$ operation of a process p is modelled by a transition t of the form

$$t, p \ : \ (\ell_{ow}, s > 0 \rightarrow (s) := (s - 1), \ell_{dw}))$$

- A $signal(s)$ operation of a process p is modelled by a transition t of the form

$$t, p \ : \ (\ell_{os}, \mathbf{true} \rightarrow (s) := (s + 1), \ell_{ds})$$

Modelling Fairness for Semaphores

Even assuming weak fairness, the semaphore model we have just given does not accurately represent the usual fairness aspects of the behavior of semaphores. There are two ways to solve this problem.

The first is to model part of the process scheduling policy.

- Shared data representing the process table and the semaphore wait queue is introduced.
- The $wait(s)$ operation is then modelled by the following transitions.

$$
\begin{array}{lcl}
t_{1w}, p & : & (\ell_{1w}, s > 0 \to (s) := (s-1), \ell_{2w}) \\
t_{2w}, p & : & (\ell_{1w}, s \leq 0 \to (status[p], list_s) := \\
 & & \qquad (W, list_s \circ p), \ell_{3w}) \\
t_{3w}, p & : & (\ell_{3w}, status[p] = R \to () := (), \ell_{2w})
\end{array}
$$

- And $signal(s)$ is modelled by

$$
\begin{array}{lcl}
t_{1s}, p & : & (\ell_{1s}, list_s = \emptyset \to (s) := (s+1), \ell_{2s}) \\
t_{2s}, p & : & (\ell_{1s}, list_s \neq \emptyset \to \\
 & & \qquad (status[head(list_s)], list_s) := \\
 & & \qquad\quad (R, tail(list_s)), \ell_{2s})
\end{array}
$$

- The actual process scheduling is then represented by the interleaving of the transitions included in the model.

This model is precise, but is only valid for a particular implementation of semaphores and substantially increases the size of the state space of the system being studied.

An alternative is to use a fairness assumption stronger than weak fairness. This has several advantages.

- The model stays very simple and the size of its state space does not increase.
- The model does not represent a specific implementation of the semaphore, but all the implementations that do satisfy the fairness hypothesis.
- A property verified on the model will thus be true of any implementation that satisfies the fairness assumption.

Strong Fairness

We define strong fairness at the process level. An alternative is to do so at the transition level. It is convenient to first define the negation of this notion.

> A behavior is not strongly fair if, for some process p, there is infinitely often an enabled transition in which p is active, but from some point on no transition in which p is active is executed.

An equivalent positive formulation is the following.

> A behavior is strongly fair is, for all processes p, either from some point on no transition in which p is active is enabled, or a transition in which p is active is executed infinitely often.

With strong fairness, the simple direct model of semaphores is sufficient for proving most properties of systems involving semaphores.

Observers for Strong Fairness

To model strong fairness, one can add the following observer for each process p of the FCS.

- The locations:
  $\ell(\textit{ob-p}) = \{\ell_0^{\textit{ob-p}}, \ell_1^{\textit{ob-p}}, \ell_2^{\textit{ob-p}}, \ell_3^{\textit{ob-p}}\}$

- The transitions:

  $(\ell_0^{\textit{ob-p}}, \neg e(p), \ell_0^{\textit{ob-p}})$
  $(\ell_0^{\textit{ob-p}}, a(p), \ell_1^{\textit{ob-p}})$
  $(\ell_1^{\textit{ob-p}}, \mathbf{true}, \ell_0^{\textit{ob-p}})$
  $(\ell_0^{\textit{ob-p}}, \neg e(p), \ell_2^{\textit{ob-p}})$
  $(\ell_2^{\textit{ob-p}}, e(p), \ell_3^{\textit{ob-p}})$

- The acceptance condition:
  $\mathbf{F}^{\textit{ob-p}} = \{\{\ell_1^{\textit{ob-p}}, \ell_2^{\textit{ob-p}}\}\}$

Note the these observers are nondeterministic.

Conclusions

- We need to know more about infinite behaviors and transition system (automata) accepting them.

- This is the topic of next week's lecture.