Ample Set Conditions

Lecture #10 of Advanced Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

November 27, 2006

Outline of partial-order reduction

- During state space generation obtain $\widehat{TS}$
  - a *reduced version* of transition system TS such that $\widehat{TS} \cong TS$
  - $\Rightarrow$ this preserves all stutter sensitive LT properties, such as $\text{LTL}_{\setminus \bigcirc}$
  - at state s select a (small) subset of enabled actions in s
  - different approaches on how to select such set: consider Peled's *ample sets*
- *Static* partial-order reduction
  - obtain a high-level description of $\widehat{TS}$ (without generating TS)
  - $\Rightarrow$ POR is preprocessing phase of model checking
- *Dynamic (or: on-the-fly)* partial-order reduction
  - construct TS during $\text{LTL}_{\setminus \bigcirc}$ model checking
  - if accept cycle is found, there is no need to generate entire $\widehat{TS}$

Independence of actions

Let $\mathit{TS} = (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ be action-deterministic and $\alpha \neq \beta \in \mathit{Act}$

- α and β are *independent* if for any $s \in S$ with $\alpha, \beta \in \mathit{Act}(s)$:

$$\beta \in \mathit{Act}(\alpha(s)) \quad \text{and} \quad \alpha \in \mathit{Act}(\beta(s)) \quad \text{and} \quad \alpha(\beta(s)) \;=\; \beta(\alpha(s))$$

- α and β are *dependent* if α and β are not independent
- For $A \subseteq \mathit{Act}$ and $\beta \in \mathit{Act} \setminus A$:
  - β is independent of A if for any $\alpha \in A$, β is independent of α
  - β depends on A in TS if $\beta \in \mathit{Act} \setminus A$ and α are dependent for some $\alpha \in A$

Permuting independent stutter actions

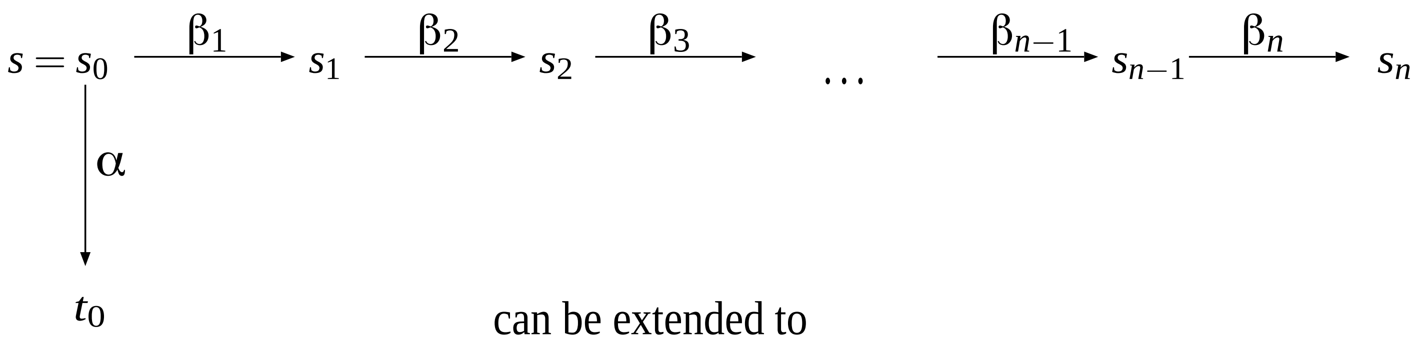
Let *TS* be action-deterministic, s a state in *TS* and:

- ϱ is a finite execution in s with action sequence $\beta_1 \ldots \beta_n \, \alpha$
- ϱ' is a finite execution in s with action sequence $\alpha \, \beta_1 \ldots \beta_n$

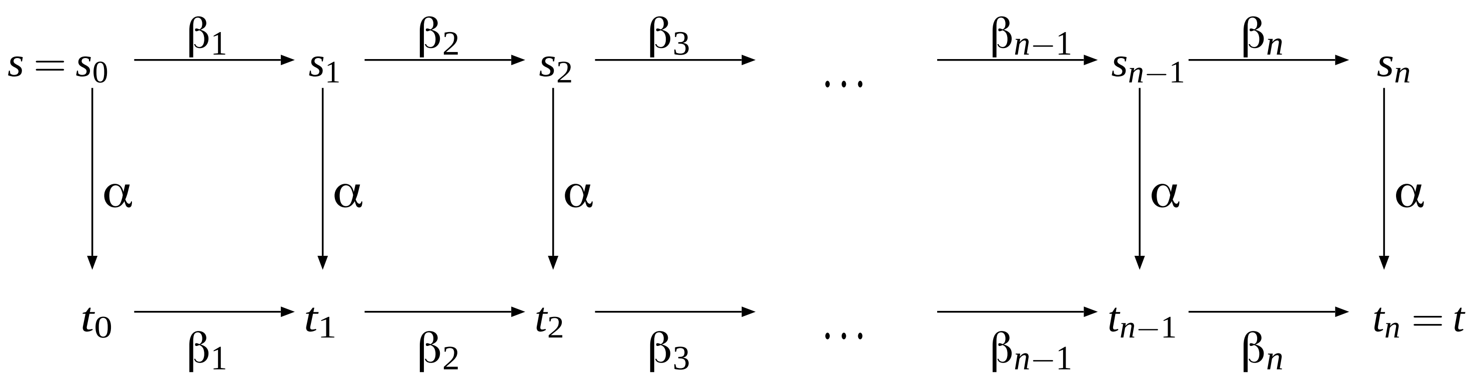
Then:

> if α is a stutter action independent of $\{ \beta_1, \ldots, \beta_n \}$ then $\varrho \cong \varrho'$

Permuting independent actions



can be extended to



Adding an independent stutter action

Let *TS* be action-deterministic, s a state in *TS* and:

- ρ is an infinite execution in s with action sequence $\beta_1 \, \beta_2 \ldots$
- ρ' is an infinite execution in s with action sequence $\alpha \, \beta_1 \, \beta_2 \ldots$

Then:

if α is a stutter action independent of $\{ \beta_1, \beta_2, \ldots \}$ then $\rho \cong \rho'$

The ample-set approach

- Partial-order reduction for LT properties using *ample sets*
  - generate $\widehat{TS}$ from a high-level description of TS (e.g., program graph)
  - . . . without the need for ever generating the entire transition system TS

- $\widehat{TS} = (\widehat{S}, Act, \Longrightarrow, I, AP, L')$ where:
  - $\widehat{S}$ contains the states that are reachable (under $\Longrightarrow$) from some $s_0 \in I$
  - $$\frac{s \xrightarrow{\alpha} s' \;\wedge\; \alpha \in \mathit{ample}(s)}{s \stackrel{\alpha}{\Longrightarrow} s'}$$
  - $L'(s) = L(s)$ for any $s \in \widehat{S}$

- Constraints: correctness ($\cong$), effectivity and efficiency

Transforming executions

Transforming executions (case 1)

$\exists n > 0$ such that $\alpha = \beta_{n+1} \in$ *ample*(s) and $\beta_1, \ldots, \beta_n \notin$ *ample*(s)

Then ρ_0 can be changed into ρ_1:

$$\rho_0 = \underbrace{u \overset{\gamma_1}{\Longrightarrow} \ldots \overset{\gamma_m}{\Longrightarrow}}_{\text{common prefix } \varrho_0} \underbrace{s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha}}_{} \underbrace{t \xrightarrow{\beta_{n+2}} s_{n+2} \xrightarrow{\beta_{n+3}} \ldots}_{}$$

$$\rho_1 = u \overset{\gamma_1}{\Longrightarrow} \ldots \overset{\gamma_m}{\Longrightarrow} s \overset{\alpha}{\Longrightarrow} t_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} t \xrightarrow{\beta_{n+2}} s_{n+2} \xrightarrow{\beta_{n+3}} \ldots$$

common prefix ϱ_0 | stutter-trace equivalent execution fragments | common suffix

m is the minimal index at which some non-ample action is taken

if α is a stutter action we have $\rho_0 \cong \rho_1$

Transforming executions (case 2)

For all $i > 0$, $\beta_i \notin \mathit{ample}(s)$

Then for any $\alpha \in \mathit{ample}(s)$, ρ_0 can be changed into ρ_1

$$\begin{array}{lcl}
\rho_0 & = & \underbrace{u \overset{\gamma_1}{\Longrightarrow} \ldots \overset{\gamma_m}{\Longrightarrow}}_{\text{common prefix } \varrho_0} \; s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_1} s_2 \xrightarrow{\beta_2} \ldots \\
\rho_1 & = & u \overset{\gamma_1}{\Longrightarrow} \ldots \overset{\gamma_m}{\Longrightarrow} \; \underbrace{s \overset{\alpha}{\Longrightarrow} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_1} t_2 \xrightarrow{\beta_2} \ldots}_{\text{stutter-trace equivalent execution fragments}}
\end{array}$$

common prefix ϱ_0 stutter-trace equivalent execution fragments

m is the minimal index at which some non-ample action is taken

if α is a stutter action we have $\rho_0 \cong \rho_1$

Which actions to select?

(A1) **Nonemptiness condition**

Select in any state in $\widehat{TS}$ at least one action.

(A2) **Dependency condition**

For any finite execution in *TS*: an action depending on *ample*(s) can only occur after some action in *ample*(s) has occurred.

(A3) **Stutter condition**

If not all actions in s are selected, then only select stutter actions in s.

(A4) **Cycle condition**

Any action in *ample*(s_i) with s_i on a cycle in $\widehat{TS}$ must be selected in some s_j on that cycle.

(A1) through (A3) apply to states in $\widehat{S}$; (A4) to cycles in $\widehat{TS}$

Example

Nonemptiness condition

(A1)

$\varnothing \neq \textit{ample}(s) \subseteq \textit{Act}(s)$

- If a state has at least one direct successor in *TS*, then it has least at one direct successor in $\widehat{\textit{TS}}$

$\Rightarrow$ As *TS* has no terminal states, $\widehat{\textit{TS}}$ has no terminal states

Dependency condition

(A2) Dependency condition

Let $s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$ be a finite execution in *TS* such that α depends on *ample*(s).

Then: $\beta_i \in$ *ample*(s) for some $0 < i \leqslant n$.

- In every (!) finite execution fragment of *TS*, an action depending on *ample*(s) cannot occur before some action from *ample*(s) occurs first
- (A2) ensures that for any state s with *ample*$(s) \subset$ *Act*(s), any $\alpha \in$ *ample*(s) is independent of *Act*$(s) \setminus$ *ample*(s)

Properties

- (A2) guarantees that any finite execution in *TS* is of the form:

$$\varrho \;=\; s_1 \xrightarrow{\beta_1} s_2 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \quad \text{with} \quad \alpha \in \textit{ample}(s)$$

  and β_i independent of *ample*(s) for $0 < i \leqslant n$.

  - if α is a stutter action: shifting α to the beginning yields an equivalent execution

  $\Rightarrow$ if ϱ is pruned in *TS*, then an execution is obtained by first taking α in s

- (A2) guarantees that any infinite execution in *TS* is of the form:

$$s_1 \xrightarrow{\beta_1} s_2 \xrightarrow{\beta_2} \ldots \quad \text{with } \beta_i \text{ independent of } \textit{ample}(s) \text{ for } 0 < i \leqslant n.$$

  - performing stutter action $\alpha \in$ *ample*(s) in s yields an equivalent execution

Properties

For any $\alpha \in \textit{ample}(s)$ and $s \in \textit{Reach}(\textit{TS})$:

if $\textit{ample}(s)$ satisfies (A2) then α is independent of $\textit{Act}(s) \setminus \textit{ample}(s)$

For finite execution $s = s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n$ in *TS*:

if $\textit{ample}(s)$ satisfies (A2) and $\{\, \beta_1, \ldots, \beta_n \,\} \cap \textit{ample}(s) = \varnothing$, then:

α is independent of $\{\, \beta_1, \ldots, \beta_n \,\}$ and $\alpha \in \textit{Act}(s_i)$ for $0 \leqslant i \leqslant n$

A too simplistic dependency condition (1)

(A2')

If *ample*$(s) \neq$ *Act*(s)

then $\alpha \in$ *ample*(s) is independent of *Act*$(s) \setminus$ *ample*(s).

this is a property of (A2), but in itself too weak: see next example

A too simplistic dependency condition (2)

Stutter condition

(A3)

If *ample*$(s) \neq$ *Act*(s) then any $\alpha \in$ *ample*(s) is a stutter action.

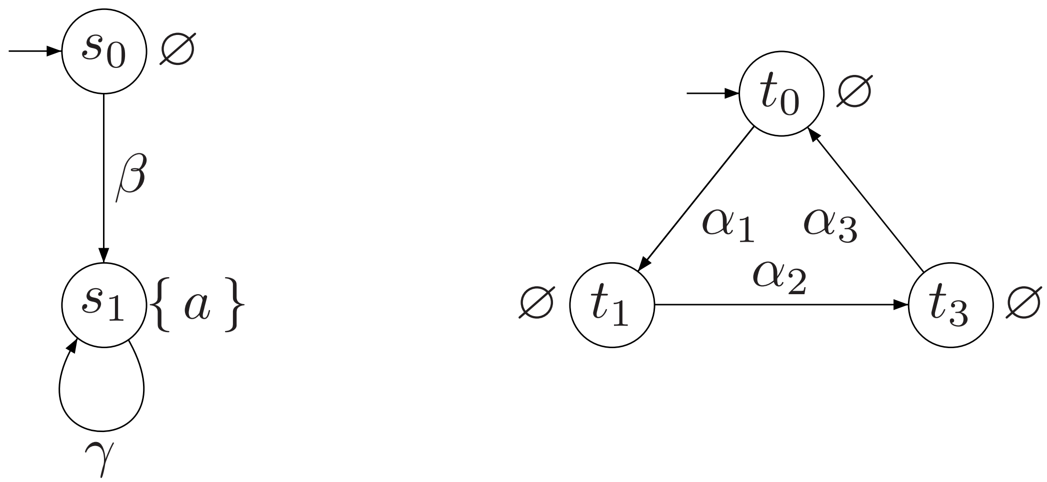
- All ample actions of a non-fully expanded state are stutter actions
- (A3) ensures that:
  - changing $\beta_1 \, , \ldots \, \beta_n \, \alpha$ into $\alpha \, \beta_1 \, \ldots \, \beta_n$, and
  - changing $\beta_1 \, \beta_2 \, \beta_3 \ldots$ into $\alpha \, \beta_1 \, \beta_2 \, \beta_3 \ldots$

  yields stutter-equivalent executions

Correctness of transformation (1)

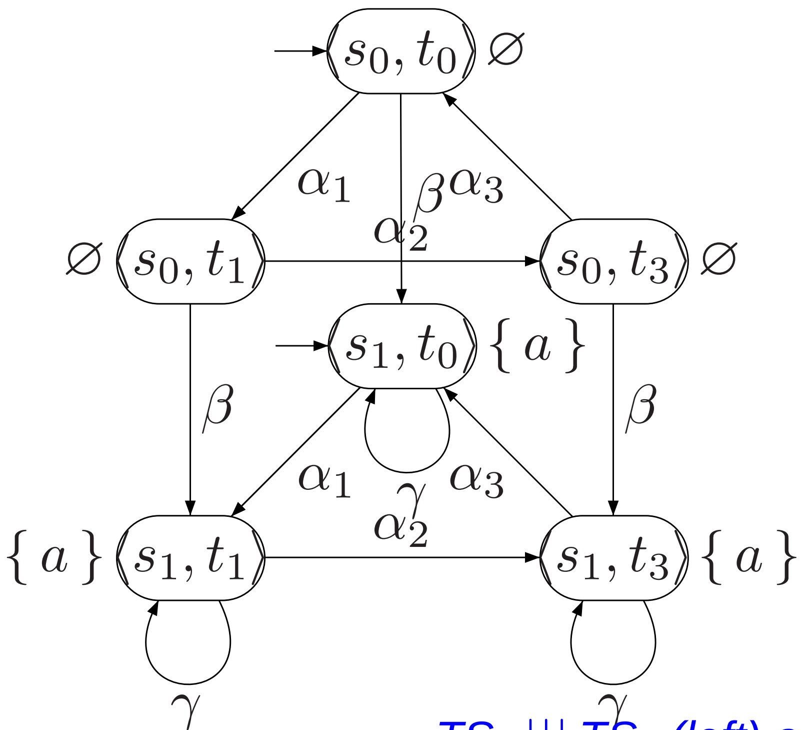
Necessity of cycle condition

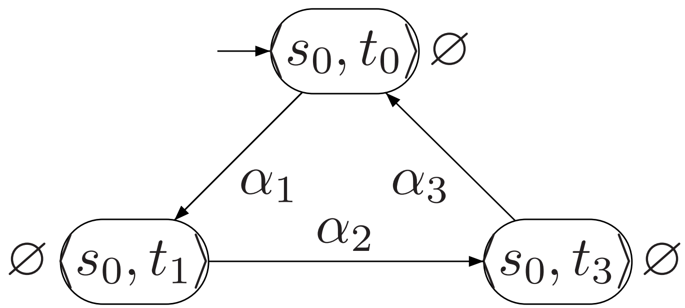
Necessity of cycle condition: example (1)



transition systems TS_1 and TS_2

Necessity of cycle condition: example (1)





$TS_1 \; ||| \; TS_2$ (left) and $\widehat{TS_1 \; ||| \; TS_2}$ (right)

$TS_1 \; ||| \; TS_2 \not\models \Box \neg a$ but $\widehat{TS_1 \; ||| \; TS_2} \models \Box \neg a$

Cycle condition

(A4) Cycle condition

For any cycle $s_0\, s_1\, \ldots\, s_n$ in $\widehat{TS}$ and $\alpha \in \mathit{Act}(s_i)$, for some $0 < i \leqslant n$, there exists $j \in \{1, \ldots, n\}$ such that $\alpha \in \mathit{ample}(s_j)$.

any enabled action in some state on a cycle must be selected in some state on that cycle

Example

Overview of ample-set conditions

(A1) **Nonemptiness condition**

$\varnothing \neq \textit{ample}(s) \subseteq \textit{Act}(s)$

(A2) **Dependency condition**

Let $s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$ be a finite execution fragment in *TS* such that α depends on *ample*(s). Then: $\beta_i \in \textit{ample}(s)$ for some $0 < i \leqslant n$.

(A3) **Stutter condition**

If *ample*$(s) \neq$ *Act*(s) then any $\alpha \in \textit{ample}(s)$ is a stutter action.

(A4) **Cycle condition**

For any cycle $s_0\, s_1\, \ldots\, s_n$ in $\widehat{TS}$ and $\alpha \in \textit{Act}(s_i)$, for some $0 < i \leqslant n$, there exists $j \in \{ 1, \ldots, n \}$ such that $\alpha \in \textit{ample}(s_j)$.

Correctness theorem

For action-deterministic, finite TS without terminal states:

if conditions (A1) through (A4) are satisfied, then $\widehat{TS} \cong TS$.

as $Traces(\widehat{TS}) \subseteq Traces(TS)$, it follows $\widehat{TS} \sqsubseteq TS$

proof sketch of reverse direction in lecture notes

Strong cycle condition

(A4') Strong cycle condition

On any cycle $s_0\, s_1\, \ldots\, s_n$ in $\widehat{TS}$,

there exists $j \in \{ 1, \ldots, n \}$ such that $ample(s_j) = Act(s_j)$.

- (A4') implies the cycle condition (A4)
- (A4') can be checked easily in DFS when backward edge is found

Invariant checking with POR

- Invariant checking
  - on state space generation, check whether each state satisfies prop. formula Φ
  - on finding a refuting state, (reversed) stack content yields counterexample
- Incorporating partial order reduction
  - on encountering a new state, compute ample set satisfying (A1) through (A3)
  - e.g., $\mathit{ample}(s) = \mathit{Act}(P_i)$, enabled actions of a concurrent process
  - enlarge $\mathit{ample}(s)$ on demand using strong cycle condition (A4')
  - mark actions to keep track of which actions have been taking

Depth-first search under POR (1)

Input: finite transition system *TS* and propositional formula Φ
Output: "yes" if *TS* $\models \Box\Phi$", otherwise "no" plus a counterexample

```
set of  states R := ∅;                         (* the set of reachable states *)
stack of  states U := ε;                       (* the empty stack *)
bool b := true;                                (* all states in R satisfy Φ *)
while (I \ R ≠ ∅  ∧  b) do
   let s ∈ I \ R;                              (* choose an arbitrary initial state not in R *)
   visit(s);                                   (* perform a DFS for each unvisited initial state *)
od
if b then
   return("yes")                               (* TS ⊨ "always Φ" *)
else
   return("no", reverse(U))                    (* counterexample arises from the stack content *)
fi
```

```
procedure visit (state s)
  push(s, U); R := R ∪ { s };                                  (* mark s as reachable *)
  compute ample(s) satisfying (A1)–(A3);
  mark(s) := ∅;                                                (* taken actions in s *)
  repeat
    s' := top(U);
    if ample(s') = mark(s') then
      pop(U); b := b ∧ (s' ⊨ Φ);                               (* all ample actions have been taken *)
    else
      let α ∈ mark(s') \ ample(s')
      mark(s') := mark(s') ∪ { α };                            (* mark α as taken *)
      if α(s') ∉ R then
        push(α(s'), U); R := R ∪ { α(s') }                     (* α(s') is a new reachable state *)
        compute ample(α(s')) satisfying (A1)–(A3);
        mark(α(s')) := ∅;
      else
        if s' ∈ U then ample(s) := Act(s);  fi                 (* enlarge ample(s) for (A4) *)
      fi
    fi
  until ((U = ε) ∨ ¬ b)
endproc
```

Example

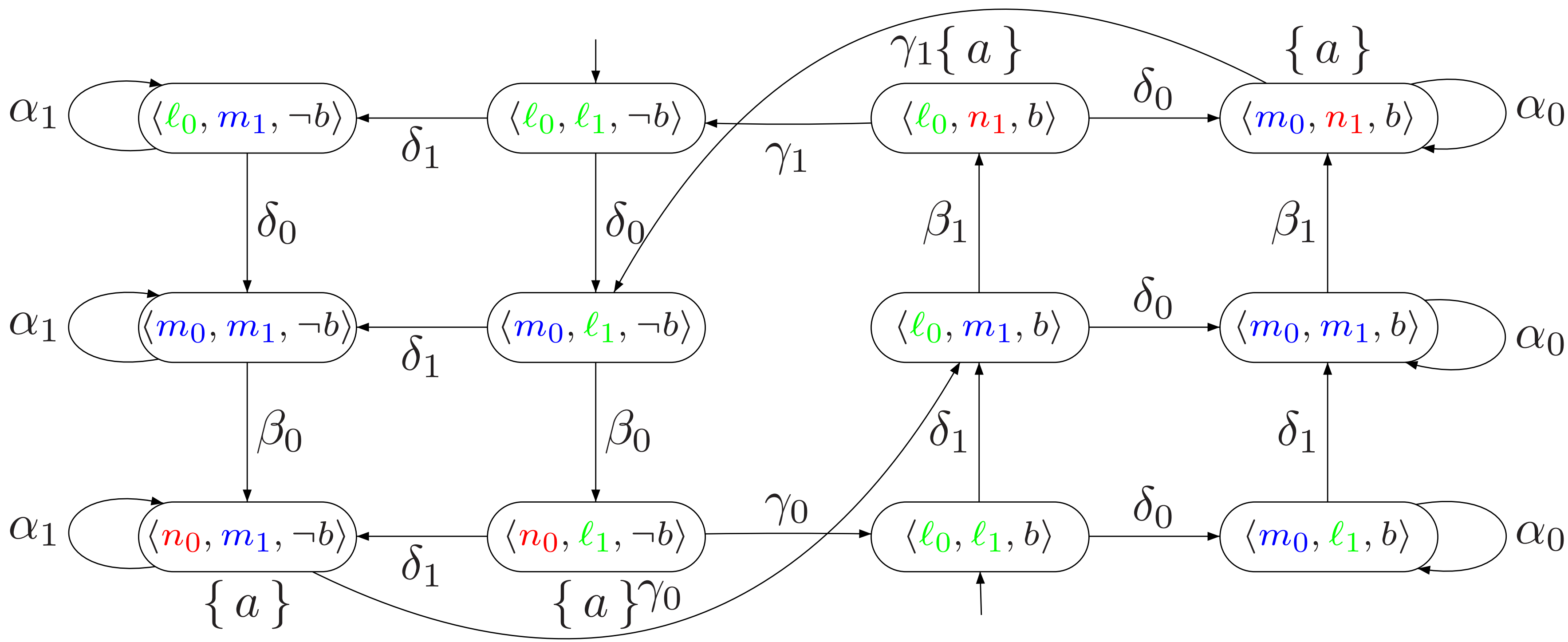
Process 0:

ℓ_0 : **while** true {
skip;
m_0 : $\textbf{wait until } (\neg b)$ {
n_0 : ... critical section ...}
$b :=$ true;
}

Process 1:

ℓ_1 : **while** true {
skip;
m_1 : $\textbf{wait until } (b)$ {
n_1 : ... critical section ...}
$b :=$ false;
}

Transition system



Reduced transition system