Computer aided verification

Lecture 4: Model checking for LTL

Algorithm

(i) $M \mapsto \mathcal{A}_M$

(ii) $\neg\phi \mapsto \mathcal{A}_{\neg\phi}$ (not $\phi \mapsto \mathcal{A}_\phi \mapsto \bar{\mathcal{A}}_\phi$)

(iii) $L_\omega(\mathcal{A}_M) \cap L_\omega(\mathcal{A}_{\neg\phi}) = \emptyset$? (not $L_\omega(\mathcal{A}_M) \subseteq L_\omega(\mathcal{A}_\phi)$)

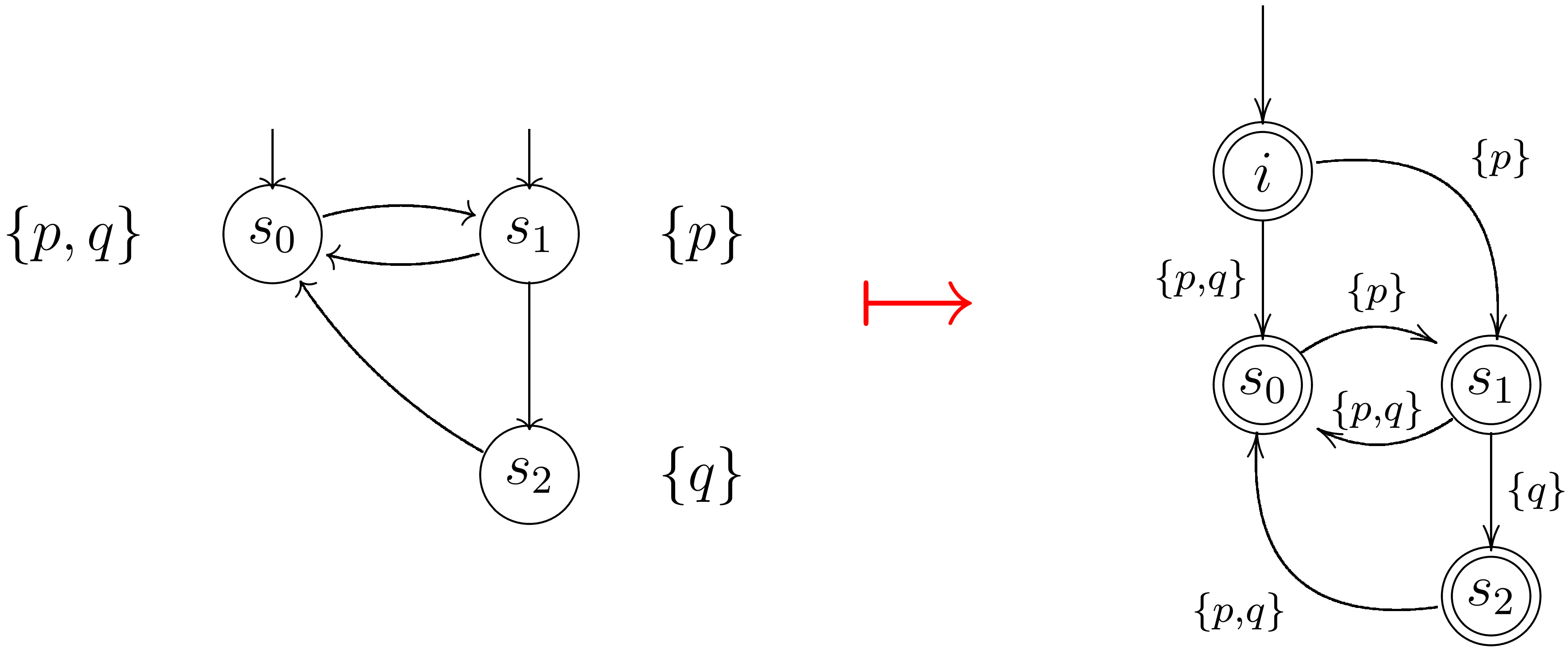
$L_\omega(\mathcal{A}_M \times \mathcal{A}_{\neg\phi}) = \emptyset$?

yes $\rightarrow M \models \phi$

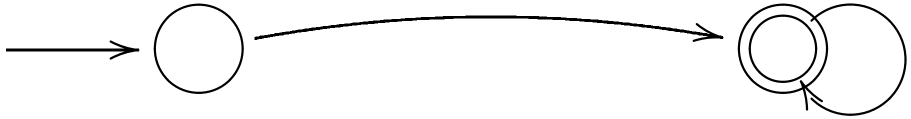
no $\rightarrow \neg(M \models \phi)$, counterexample = a path in M

$$\text{(i)} \quad M \longmapsto \mathcal{A}_M$$

$M \mapsto \mathcal{A}_M$

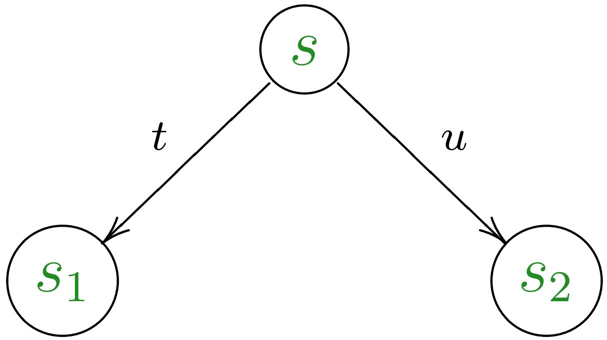


(iii) $L_\omega(\mathcal{A}) \neq \emptyset$?



(1) On the fly verification

for each **successsor** s_i **of** s **do ...**



...

Double DFS

procedure $dfs1(q)$
  local q'**;**
  $hash(q)$**;**
  for all successors q' of q **do**
    if q' not in the hash table **then**
      $dfs1(q')$**;**
  if $accept(q)$ **then** $dfs2(q)$**;**
end procedure

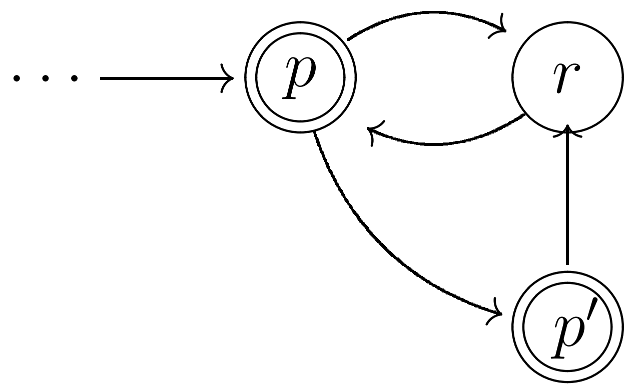
procedure $dfs2(q)$**;**
  local q'**;**
  $flag(q)$**;**
  for all successors q' of q **do**
    if q' on $dfs1$ stack **then**
      terminate($True$**)**
    else if q' not flagged **then**
      $dfs2(q')$**;**
end procedure

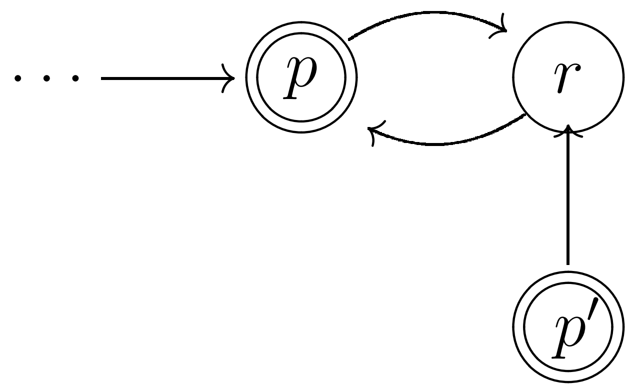
procedure $emptiness$
  for all $q_o \in Q^0$ **do** $dfs1(q_0)$**;**
  terminate($False$**);**
end procedure

Assume an acceping state p with a cycle not detected by $dfs2(p)$. Let p – the first such state.

Let r – the first flagged state inspected by $dfs2(p)$ that is on a p-cycle.

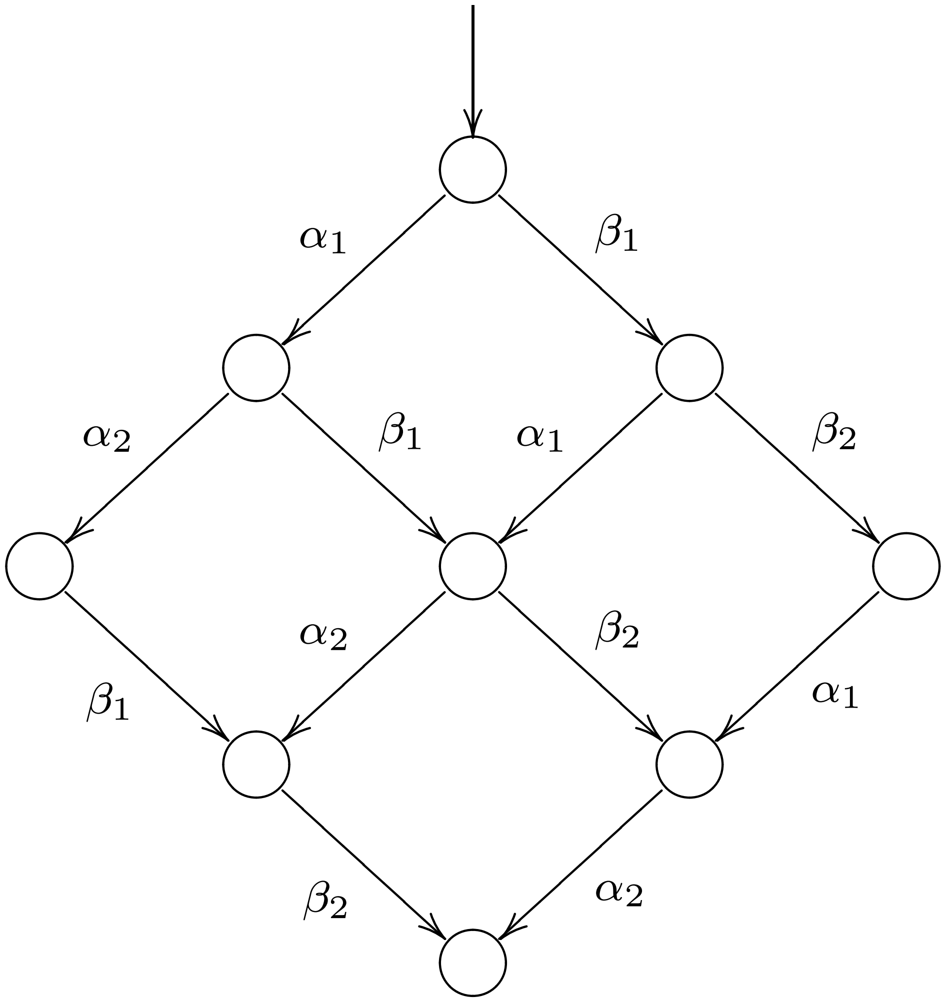
Let p' – the accepting state such that r visited by $dfs2(p')$.





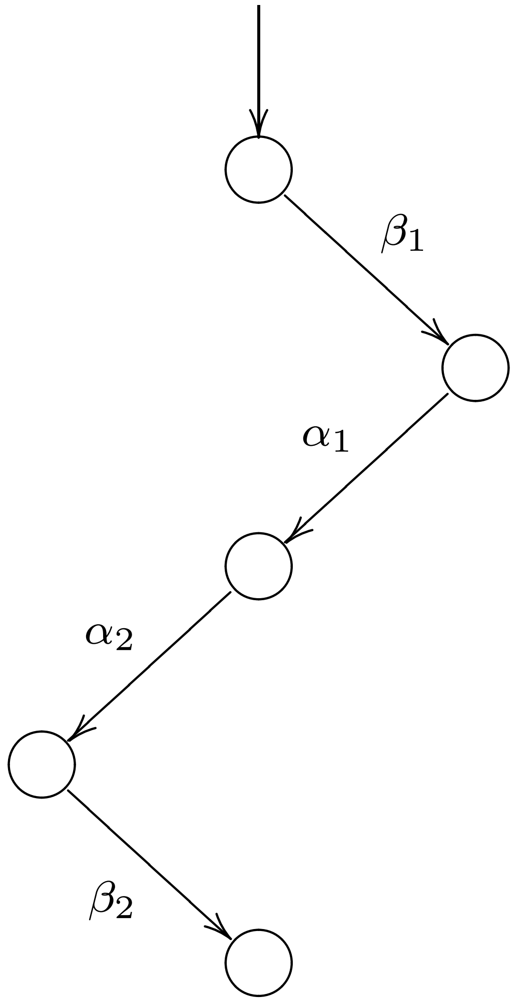
Partial-order reductions

Motivation

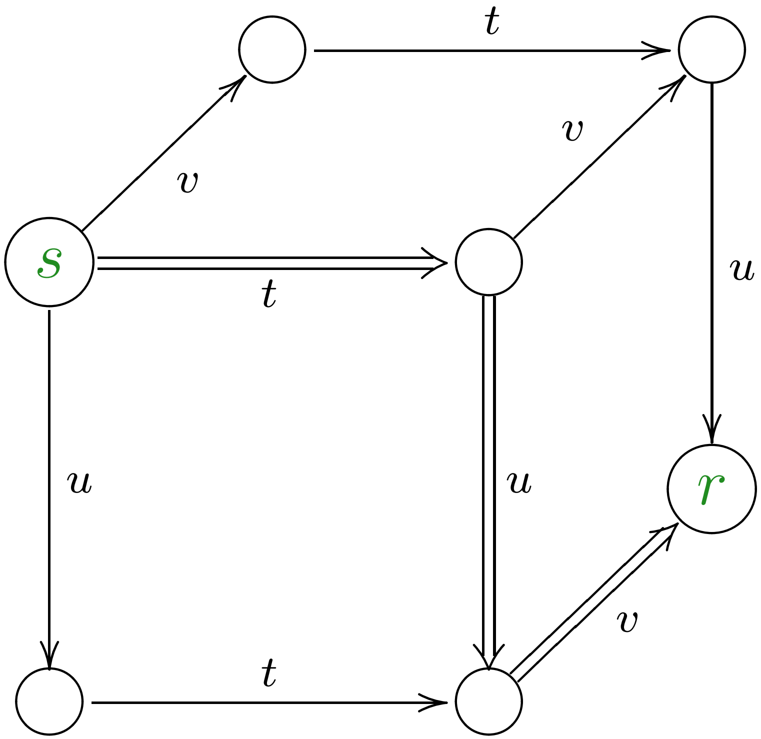


α_1
β_1
α_2
β_1
α_1
β_2
α_2
β_2
β_1
α_1
β_2
α_2

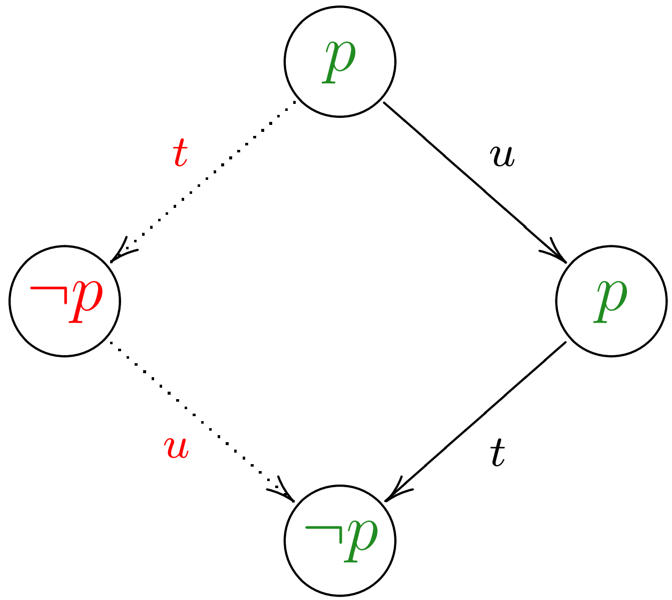
Motivation



Motivation



$\mathsf{F}\,\neg p$



t, u independent

Model

Def.: $M = \langle S, S_{\text{init}}, T, L \rangle$ $\qquad$ T – operations (transitions)

for $\alpha \in T$: $\quad \mathrm{en}_\alpha \subseteq S$, $\quad \alpha : \mathrm{en}_\alpha \to S$ $\qquad$ (determinism)

path: $\quad \Pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \ldots$ $\qquad$ $s_0 = s_{\text{init}}$

$\alpha_i(s_i) = s_{i+1}$

$\mathrm{en}_s := \{\alpha \mid s \in \mathrm{en}_\alpha\}$ $\qquad$ $(\alpha \in \mathrm{en}_s \iff s \in \mathrm{en}_\alpha)$

Idea: $\mathrm{ample}_s \subseteq \mathrm{en}_s$ instead of en_s in double DFS ?

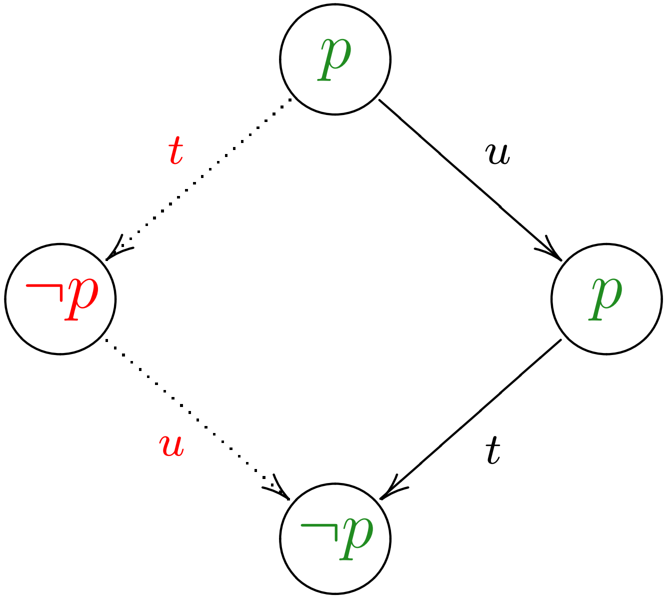
Idea: $\mathrm{ample}_s \subseteq \mathrm{en}_s$ instead of en_s in double DFS ?

This makes sense, when:

- the result of verification is the same (correctness)
- significantly less states visited
- time overhead reasonable (effectivity)

Problems?

When may we ignore t ?



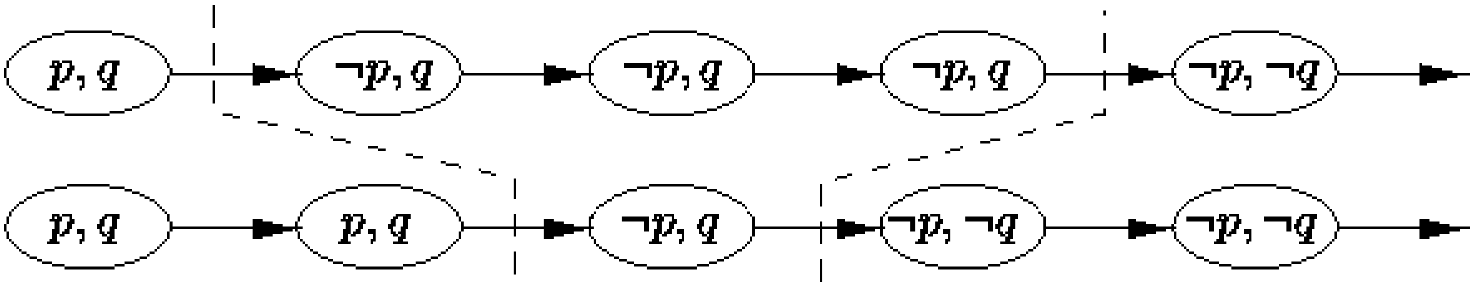
Problem 1: Property may depend on state $\neg p$.

Problem 2: $\neg p$ –successors unreachable otherwise.

Def.: $\Pi = s_0 \to s_1 \to s_2 \to \ldots$ i $\Pi' = s'_0 \to s'_1 \to s'_2 \to \ldots$ are stuttering equivalent, $\Pi \equiv \Pi'$, if sequences

$$L(s_0), L(s_1), L(s_2), \ldots \qquad L(s'_0), L(s'_1), L(s'_2), \ldots$$

become identical after grouping is done:



Def.: $M \equiv M'$ if and only if

– $\forall \Pi \; w \; M \quad \exists \Pi' \; w \; M' \quad \Pi \equiv \Pi'$

– $\forall \Pi' \; w \; M' \quad \exists \Pi \; w \; M \quad \Pi \equiv \Pi'$

LTL$_{-X}$ = LTL without X

Thm: If $\phi \in$ LTL$_{-X}$ and $\Pi \equiv \Pi'$, then $\Pi \models \phi \iff \Pi' \models \phi$

Thm: If $\phi \in$ LTL$_{-X}$ and $M \equiv M'$, then $M \models \phi \iff M' \models \phi$

Thm: LTL$_{-X}$ $=$ FO$_{\equiv}$

$$M \xmapsto{\text{partial-order reduction}} M'$$

$$M \equiv M'$$

Sufficient condition for correctness

(C0) $\mathrm{ample}_s = \emptyset \iff \mathrm{en}_s = \emptyset$

(C1) $\ldots$

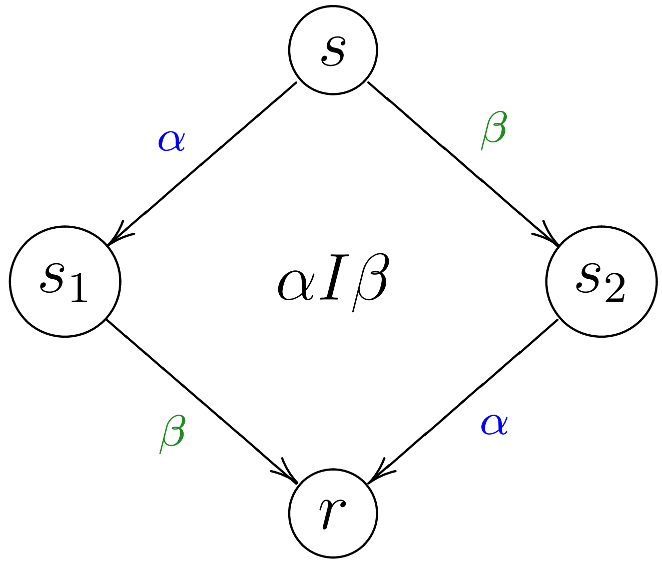
(C2) $\ldots$

(C3) $\ldots$

Def.: α is invisible if $L(s) = L(\alpha(s))$, $\forall\ s \in \mathrm{en}_\alpha$.

Przykład: If α invisible, then

$$ss_1r \equiv ss_2r$$



Sufficient condition for correctness

(C0) $\mathrm{ample}_s = \emptyset \iff \mathrm{en}_s = \emptyset$

(C1) if $\mathrm{ample}_s \neq \mathrm{en}_s$ then every $\alpha \in \mathrm{ample}_s$ is invisible

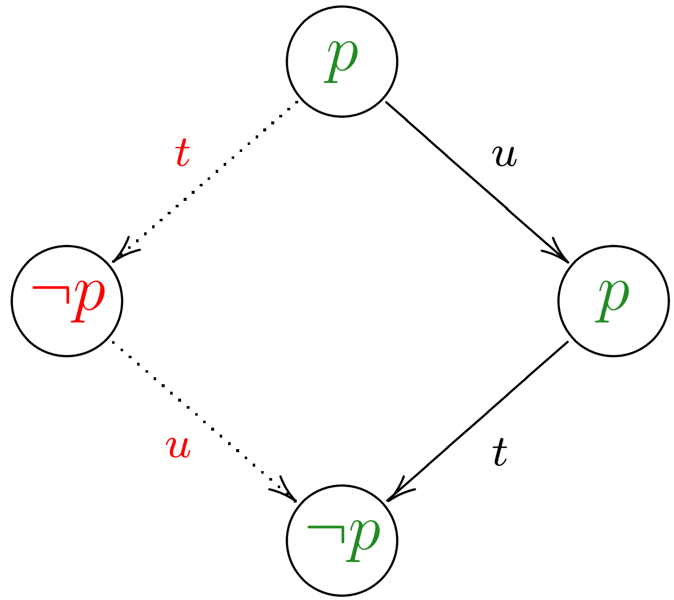
(C2) ...

(C3) ...

Idea: Instead of doing sth now, do it in future!

Problems?

Problem 1: Property may depend on state $\neg p$.

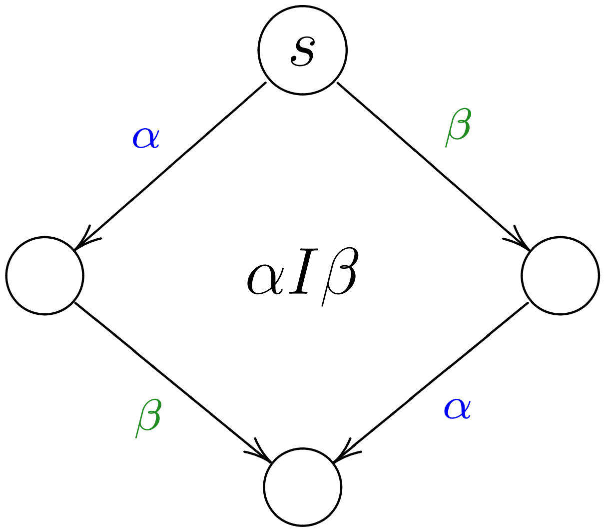


Solved due to **(C1)** !

(C1) if $\mathrm{ample}_s \neq \mathrm{en}_s$, then every $\alpha \in \mathrm{ample}_s$ is invisible

Def.: Relation of independence $I \subseteq T \times T$:

- irreflexive and symmetric
- if $\alpha I \beta$, $\alpha \in \mathrm{en}_s$, $\beta \in \mathrm{en}_s$, then $(s \in \mathrm{en}_\alpha \cap \mathrm{en}_\beta)$
  - $\beta(s) \in \mathrm{en}_\alpha$, $\alpha(s) \in \mathrm{en}_b$
  - $\beta(\alpha(s)) = \alpha(\beta(s))$



$D = T \times T \ \setminus \ I$ (dependency)

Example: Independent may be:

- 2 instructions of different processes operating on local variables
- 2 instructions of different processes that increment the same global variable
- 2 instructions of different processes writing to/reading from different buffers

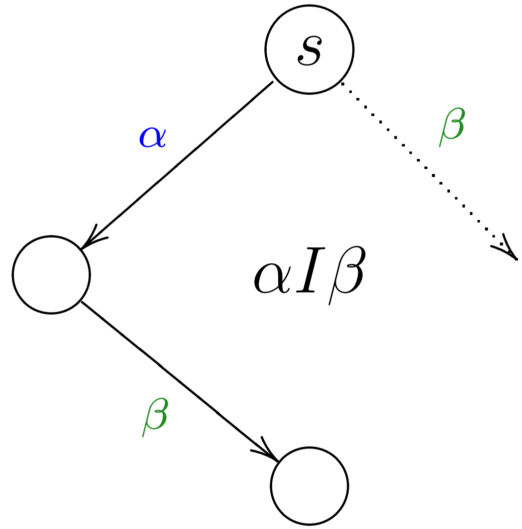
Example: Independent may be:

- 2 instructions of different processes operating on local variables
- 2 instructions of different processes that increment the same global variable
- 2 instructions of different processes writing to/reading from different buffers

- 2 instructions of the same process ?

Question: Let $\alpha I \beta$. Is it possible that

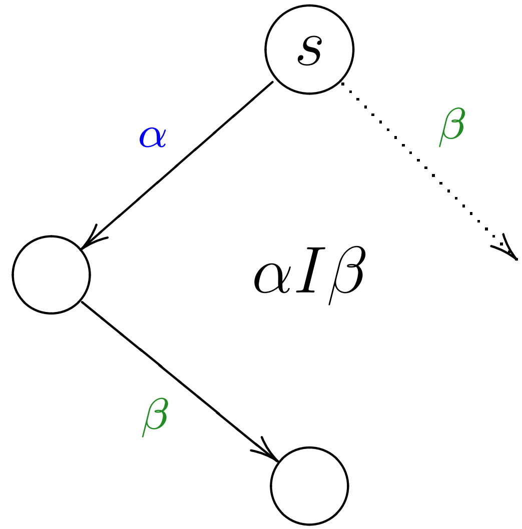
$$s \in \mathrm{en}_\alpha \setminus \mathrm{en}_\beta \qquad \alpha(s) \in \mathrm{en}_\beta \;?$$



Independence

Question: Let $\alpha I \beta$. Is it possible that

$$s \in \mathrm{en}_\alpha \setminus \mathrm{en}_\beta \qquad \alpha(s) \in \mathrm{en}_\beta\ ?$$



Yes! E.g. asynchronous reading and writing from/to the same buffer by two different processes.

Sufficient condition for correctness

(C0) $\text{ample}_s = \emptyset \iff \text{en}_s = \emptyset$

(C1) if $\text{ample}_s \neq \text{en}_s$ then every $\alpha \in \text{ample}_s$ is invisible

(C2) ? $\left(\text{en}_s \setminus \text{ample}_s\right) \; I \; \text{ample}_s$

(C3) . . .

Idea: Instead of doing sth now, do it in future!

(C2) a transition dependent on some transition from ample_s

can not be executed

before some transition from ample_s is executed

(C2) a transition dependent on some transition from ample_s

can not be executed

before some transition from ample_s is executed

(C2) for every path Π starting in s:

if $\alpha \in \mathrm{ample}_s,\ \beta \notin \mathrm{ample}_s,\ \alpha D \beta$

then β can not be executed in Π

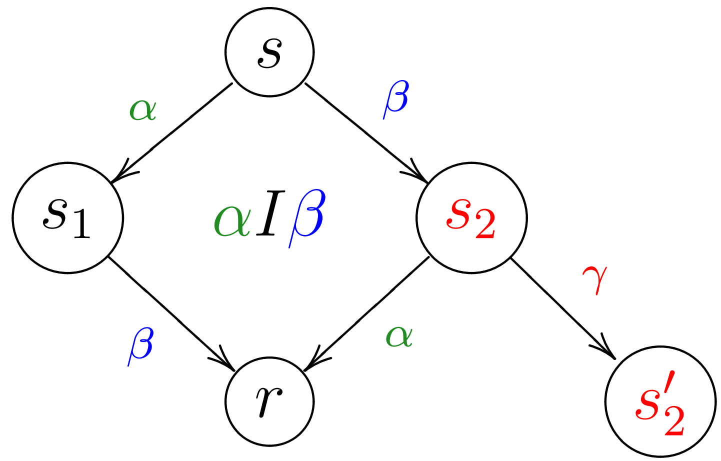
before some transition from ample_s is executed

Lemma: **(C2)** implies $\left(\mathrm{en}_s \setminus \mathrm{ample}_s\right)\ I\ \mathrm{ample}_s$.

Proof: Let $\beta \in \mathrm{en}_s \setminus \mathrm{ample}_s$, $\alpha \in \mathrm{ample}_s$, $\alpha D \beta$.

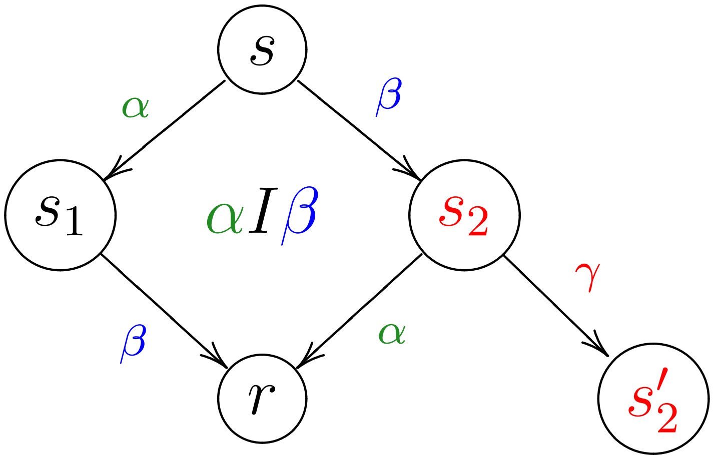
$s \xrightarrow{\beta} \beta(s) \rightarrow \ldots$ contradiction with **(C2)** .

Problem 2: s_2–successors unreachable otherwise.



e.g., let $\alpha \in \mathrm{ample}_s$, $\beta \notin \mathrm{ample}_s$

Problem 2: s_2 –successors unreachable otherwise.

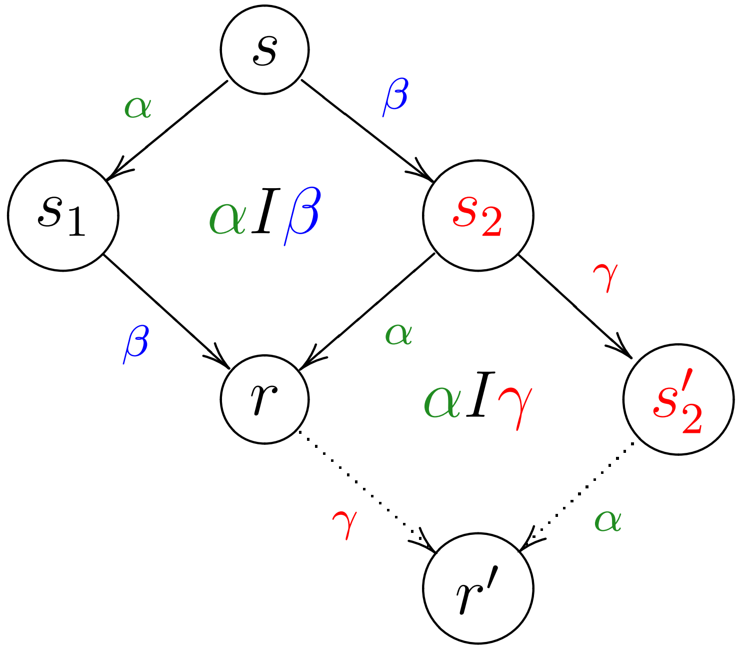


e.g., let $\alpha \in \mathrm{ample}_s, \beta \notin \mathrm{ample}_s$

by **(C2)** applied to $\beta\gamma\ldots,$ we deduce $\gamma I \alpha$

Problems?

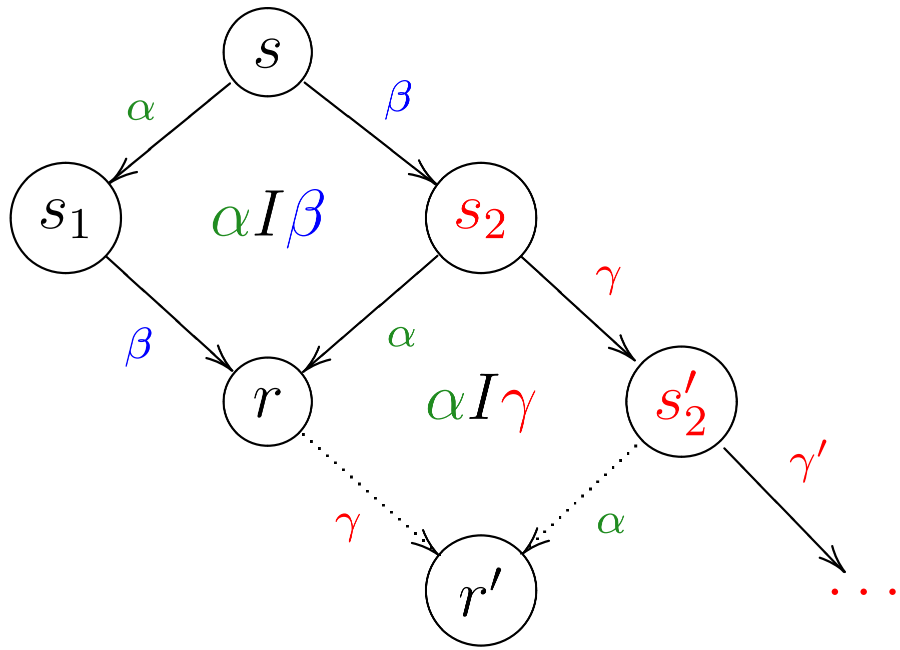
Problem 2: s_2–successors unreachable otherwise.



α invisible, thus $ss_1rr' \equiv ss_2s'_2$

Problemy?

Problem 2$^\infty$: s_2–path unreachable otherwise.



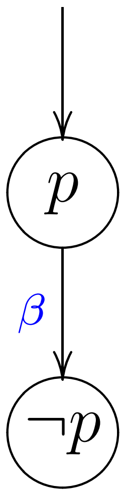
by **(C2)** we deduce $\gamma I \alpha$, $\gamma' I \alpha$, ...

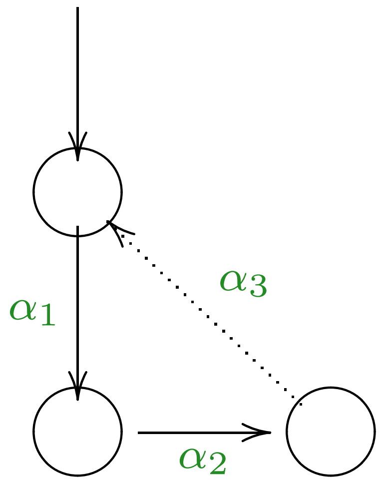
α invisible, thus $s s_1 r r' \ldots \equiv s s_2 s_2' \ldots$

Are **(C0)** – **(C2)** sufficient?

Are **(C0)** – **(C2)** sufficient?

No!





(C3) we forbid cycles C such that $\exists \beta \ \ \forall s \in C \ \ \beta \in \mathrm{en}_s \setminus \mathrm{ample}_s$

Sufficient condition for correctness

(C0) $\mathrm{ample}_s = \emptyset \iff \mathrm{en}_s = \emptyset$

(C1) if $\mathrm{ample}_s \neq \mathrm{en}_s$ then every $\alpha \in \mathrm{ample}_s$ is invisible

(C2) for every path Π starting in s:

if $\alpha \in \mathrm{ample}_s,\ \beta \notin \mathrm{ample}_s,\ \alpha D \beta$

then β can not be executed in Π

before some transition from ample_s is executed

(C3) we forbid cycles C such that $\exists \beta\ \ \forall s \in C\ \ \beta \in \mathrm{en}_s \setminus \mathrm{ample}_s$

How to implement this?

Sufficient condition for correctness

(C1) easy

(C2) hard, implemented in an approximate manner

- an over-approximation of D is computed
- condition **(C2)** is monotonic
- static analysis only

(C3) replaced by an easier but stronger:

(C3') if $\mathrm{ample}_s \neq \mathrm{en}_s$ then $\forall \alpha \in \mathrm{ample}_s \;\; \alpha(s) \notin$ stack

Implementation decision:

$\mathrm{ample}_s =$ all transitions of some process i enabled in s

Implementation decision:

$\mathrm{ample}_s =$ all transitions of some process i enabled in s

whenever

- they are independent from all operations of all other processes
- no operation of any other process may enable any other operation of process i

β enabling α (over-approximation)

- if β modifies pc so that α may be executed
- if Promela enabling condition for α depends on global variables, then any β that modifies these variables
- if α is reading from/writing to a buffer then any β that reads from/writes to this buffer

$\alpha D\beta$ **(over-approximation)**

– α i β refer to the same global variable

and at least one of them modifies the variable (over-appr.)

– α i β belong to the same process; synchronous communi-

-cation is understood as belonging to both processes

– α i β write to/read from the same buffer

However reading from and writing to the same buffer is independent!

What remains independent?

Example:

Operations independent from all operations of other processes:

- operating on local variables
- reading from a buffer with xr flag set
- writing to a buffer with xs flag set
- test `nempty(q)` if xr flag is set for q
- test `nfull(q)` if xs flag is set for q

P.-o. reductions and on the fly verification

– in both DFS's the set ample_s should be the same

– condition **(C3')** is applied to $M \times \mathcal{A}_{\neg\phi}$ instead of M.