(i) \( M \mapsto A_M \)

(ii) \( \neg \phi \mapsto A_{\neg \phi} \)

(iii) \( L_\omega(A_M) \cap L_\omega(A_{\neg \phi}) = \emptyset \)?

\[ L_\omega(A_M \times A_{\neg \phi}) = \emptyset ? \]

yes \( \rightarrow M \models \phi \)

no \( \rightarrow \neg(M \models \phi), \) counterexample = a path in \( M \)
(i) \( M \mapsto A_M \)
(iii) \( L_\omega(A) \neq \emptyset \)?
(1) On the fly verification

for each successor $s_i$ of $s$ do ...
procedure \( \text{dfs1}(q) \)  
local \( q' \);  
\( \text{hash}(q) \);  
for all successors \( q' \) of \( q \) do  
  if \( q' \) not in the hash table then  
    \( \text{dfs1}(q') \);  
  if \( \text{accept}(q) \) then \( \text{dfs2}(q) \);  
end procedure

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

procedure \( \text{emptiness} \)  
for all \( q_0 \in Q^0 \) do \( \text{dfs1}(q_0) \);  
\( \text{terminate}(\text{False}) \);  
end procedure
Proof of correctness

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

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

Let $p'$ be the accepting state such that $r$ visited by $dfs2(p')$.
Partial-order reductions
F $\overline{p}$

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

for \( \alpha \in T \): \( \text{en}_\alpha \subseteq S \), \( \alpha : \text{en}_\alpha \rightarrow S \) (determinism)

path: \( \Pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \ldots \) \( s_0 = S_{\text{init}} \)
\( \alpha_i(s_i) = s_{i+1} \)

\( \text{en}_s := \{ \alpha \mid s \in \text{en}_\alpha \} \) \( (\alpha \in \text{en}_s \iff s \in \text{en}_\alpha) \)

Idea: ample \( s \subseteq \text{en}_s \) instead of \( \text{en}_s \) in double DFS?
Cost-effectivity

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

This makes sense, when:

- the result of verification is the same \textit{(correctness)}
- significantly less states visited
- time overhead reasonable \textit{(effectivity)}
When may we ignore $t$?

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

**Problem 2:** $\neg p$—successors unreachable otherwise.
**Def.:** \( \Pi = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \) i \( \Pi' = s_0' \rightarrow s_1' \rightarrow s_2' \rightarrow \ldots \) are stuttering equivalent, \( \Pi \equiv \Pi' \), if sequences

\[ L(s_0), L(s_1), L(s_2), \ldots \quad L(s_0'), L(s_1'), L(s_2'), \ldots \]

become identical after grouping is done:

**Def.:** \( M \equiv M' \) if and only if

\[ \forall \Pi \; w \; M \; \exists \Pi' \; w \; M' \quad \Pi \equiv \Pi' \]

\[ \forall \Pi' \; w \; M' \; \exists \Pi \; w \; M \quad \Pi \equiv \Pi' \]
\[ \text{LTL}_{-X} = \text{LTL without } X \]

**Thm:** If \( \phi \in \text{LTL}_{-X} \) and \( \Pi \equiv \Pi' \), then
\[
\Pi \models \phi \iff \Pi' \models \phi
\]

**Thm:** If \( \phi \in \text{LTL}_{-X} \) and \( M \equiv M' \), then
\[
M \models \phi \iff M' \models \phi
\]

**Thm:** \( \text{LTL}_{-X} = \text{FO}_{\equiv} \)
Correctness

\[
M \xrightarrow{\text{partial-order reduction}} M'
\]

\[M \equiv M'\]
Sufficient condition for correctness

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

(C1) ...

(C2) ...

(C3) ...
Def.: \( \alpha \) is **invisible** if \( L(s) = L(\alpha(s)) \), \( \forall s \in \text{en}_\alpha \).

Przykład: If \( \alpha \) invisible, then

\[ ss_1 r \equiv ss_2 r \]
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

\((C2)\) \ldots

\((C3)\) \ldots

\textbf{Idea:} Instead of doing sth now, do it in future!
Problem 1: Property may depend on state $\neg p$.

Solved due to (C1)!

(C1) if $\text{ample}_s \neq \text{en}_s$, then every $\alpha \in \text{ample}_s$ is invisible
Def.: Relation of independence $I \subseteq T \times T$:

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

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

$$s \in \text{en}_\alpha \setminus \text{en}_\beta \quad \alpha(s) \in \text{en}_\beta ?$$
**Question:** Let $\alpha I \beta$. Is it possible that 

$$s \in \text{en}_\alpha \setminus \text{en}_\beta \quad \alpha(s) \in \text{en}_\beta ?$$

**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) \( \text{？} \) \( (\text{en}_s \setminus \text{ample}_s) \cap \text{ample}_s \)

(C3) \( \ldots \)

Idea: Instead of doing sth now, do it in future!
(C2) a transition dependent on some transition from $\text{ample}_s$

   can not be executed

   before some transition from $\text{ample}_s$ 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 \(\Pi\) starting in \(s\):

if \(\alpha \in \text{ample}_s, \beta \notin \text{ample}_s, \alpha D \beta\)

then \(\beta\) can not be executed in \(\Pi\)

before some transition from \(\text{ample}_s\) is executed
Lemma: (C2) implies \( (en_s \setminus ample_s) \) I ample\(_s\). 

Proof: Let \( \beta \in en_s \setminus ample_s \), \( \alpha \in ample_s \), \( \alpha D\beta \).

\[
s \xrightarrow{\beta} \beta(s) \rightarrow \ldots \quad \text{contradiction with (C2).}
\]
Problem 2: \( s_2 \) — successors unreachable otherwise.

e.g., let \( \alpha \in \text{ample}_s \), \( \beta \notin \text{ample}_s \)
Problem 2:  \( s_2 \)—successors unreachable otherwise.

e.g., let \( \alpha \in \ample_s, \beta \notin \ample_s \)

by \((\text{C2})\) applied to \( \beta \gamma \ldots \), we deduce \( \gamma I \alpha \)
Problem 2: \( s_2 \) — successors unreachable otherwise.

\[ \alpha \text{ invisible, thus } ss_1rr' \equiv ss_2s_2' \]
Problem 2$^\infty$: $s_2$—path unreachable otherwise.

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

$\alpha$ invisible, thus $ss_1rr'\ldots \equiv ss_2s'_2\ldots$
Are (C0) – (C2) sufficient?
Are (C0) – (C2) sufficient?

No!

(C3) we forbid cycles \( C \) such that \( \exists \beta \; \forall s \in C \; \beta \in \text{en}_s \setminus \text{ample}_s \)
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) for every path \( \Pi \) starting in \( s \):

\[
\text{if } \alpha \in \text{ample}_s, \ \beta \notin \text{ample}_s, \ \alpha D \beta \\
\text{then } \beta \text{ can not be executed in } \Pi \\
\text{before some transition from } \text{ample}_s \text{ is executed}
\]

(C3) we forbid cycles \( C \) such that \( \exists \beta \ \forall s \in C \ \beta \in \text{en}_s \setminus \text{ample}_s \)
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 $\text{ample}_s \neq \text{en}_s$ then $\forall \alpha \in \text{ample}_s \; \alpha(s) \notin \text{stack}$
Implementation decision:

\[ \text{ample}_s = \text{all transitions of some process } i \text{ enabled in } s \]
Implementation decision:

\[
\text{ample}_s = \text{all transitions of some process } i \text{ enabled in } s
\]

whenever

- they are \textit{independent} from all operations of all other processes
- no operation of any other process may \textit{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)

- \( \alpha \) \( i \) \( \beta \) refer to the same global variable and at least one of them modifies the variable (over-appr.)

- \( \alpha \) \( i \) \( \beta \) belong to the same process; synchronous communication is understood as belonging to both processes

- \( \alpha \) \( i \) \( \beta \) write to/read from the same buffer

However reading from and writing to the same buffer is 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 $\text{ample}_s$ should be the same

– condition (C3') is applied to $M \times A_\neg \phi$ instead of $M$. 