Computer aided verification

Lecture 4: Model checking for LTL
(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) \[ M \mapsto A_M \]
\[
M \mapsto A_M
\]
(iii) $L_\omega(A) \neq \emptyset$?
(1) On the fly verification

for each successor $s_i$ of $s$ do ...
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 $q_0 \in Q^0$ do $dfs1(q_0)$;
  terminate(False);
end procedure
Proof of correctness

Assume an accepting state $p$ with a cycle not detected by $dfs_2(p)$. Let $p$ – the first such state.

Let $r$ – the first flagged state inspected by $dfs_2(p)$ that is on a $p$-cycle.

Let $p'$ – the accepting state such that $r$ visited by $dfs_2(p')$. 

\[\text{Proof Diagram} \]
Partial-order reductions
Motivation
Motivation

\[ F \neg p \]

t, u niezależne
**Def.:** \( M = \langle S, S_{init}, T, L \rangle \)  

\( T \) – operations (transitions)

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

path: \( \Pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \ldots \)  
\( s_0 = s_{init} \)

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

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

**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 \text{(correctness)}
- significantly less states visited
- time overhead reasonable \text{(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:

```
    p, q
       |            |            |            |
    ¬p, q ──────→ ──────→ ──────→ ──────→ ¬p, ¬q

    p, q ──────→ ──────→ ¬p, q ──────→ ¬p, ¬q
```

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

$\forall \Pi \ w \ M \ \exists \Pi' \ w \ M' \ \Pi \equiv \Pi'$

$\forall \Pi' \ w \ M' \ \exists \Pi \ w \ M \ \Pi \equiv \Pi'$
LTL\(_{-X}\) = 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:** \(\text{LTL}_{-X} = \text{FO}_{\equiv}\)
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

\[ ss_1r \equiv ss_2r \]
Sufficient condition for correctness

(C0) ample_s = ∅ ⇐⇒ \text{ens} = ∅

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

(C2) ...

(C3) ...

Idea: Instead of doing sth now, do it in future!
Problem 1: Property may depend on state \( \lnot p \).

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

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

$(s \in \text{en}_\alpha \cap \text{en}_\beta)$
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 $\alpha \parallel \beta$. 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) ? \( (\text{en}_s \setminus \text{ample}_s) \) \( I \) \( \text{ample}_s \)

(C3) . . .

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 $\text{ample}_s$
can not be executed
before some transition from $\text{ample}_s$ 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 \((\text{ens} \setminus \text{ample}_s) \cup \text{ample}_s\).

Proof: Let \(\beta \in \text{ens} \setminus \text{ample}_s\), \(\alpha \in \text{ample}_s\), \(\alpha D \beta\).

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

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

E.g., let $\alpha \in \text{ample}_s$, $\beta \notin \text{ample}_s$

By (C2) applied to $\beta \gamma \ldots$, we deduce $\gamma I \alpha$
Problem 2: $s_2$—successors unreachable otherwise.

$\alpha$ invisible, thus $ss_1rr' \equiv ss_2s'_2$
Problem 2\( \infty \): \( s_2 \)–path unreachable otherwise.

By (C2) we deduce \( \gamma I \alpha, \gamma' I \alpha, \ldots \)

\( \alpha \) 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 $\exists \beta \ \forall s \in C \ \beta \in \text{en}_s \setminus \text{ample}_s$
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 \( s \):

- 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

(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 $\forall \alpha \in \text{ample}_s \quad \alpha(s) \notin \text{stack}$
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 independent from all operations of all other processes
- no operation of any other process may enable any other operation of process \( i \)
\( \beta \) enabling \( \alpha \) (over-approximation)

- if \( \beta \) modifies pc so that \( \alpha \) may be executed

- if Promela enabling condition for \( \alpha \) depends on global variables, then any \( \beta \) that modifies these variables

- if \( \alpha \) is reading from/writing to a buffer then any \( \beta \) that reads from/writes to this buffer
\( \alpha \Delta \beta \) (over-approximation)

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

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

- \( \alpha \) and \( \beta \) 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 \texttt{xr} flag set
- writing to a buffer with \texttt{xs} flag set
- \texttt{test nempty(q)} if \texttt{xr} flag is set for \texttt{q}
- \texttt{test nfull(q)} if \texttt{xs} flag is set for \texttt{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 $M$.

Is it correct?