Linear-Time Properties

Lecture #5b of Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling and Verification

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

November 4, 2008
Overview Lecture #5

- Paths and traces
- Linear-time (LT) properties
- Trace equivalence and LT properties
- Invariants
Recall model checking

we now consider: what are properties?
Recall executions

- A **finite execution fragment** $\varrho$ of $TS$ is an alternating sequence of states and actions ending with a state:

  $$\varrho = s_0 \alpha_1 s_1 \alpha_2 \ldots \alpha_n s_n \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i < n.$$ 

- An **infinite execution fragment** $\rho$ of $TS$ is an infinite, alternating sequence of states and actions:

  $$\rho = s_0 \alpha_1 s_1 \alpha_2 s_2 \alpha_3 \ldots \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i.$$ 

- An **execution** of $TS$ is an initial, maximal execution fragment
  - a **maximal** execution fragment is either finite ending in a terminal state, or infinite
  - an execution fragment is **initial** if $s_0 \in I$
State graph

- The *state graph* of $TS$, notation $G(TS)$, is the digraph $(V, E)$ with vertices $V = S$ and edges $E = \{(s, s') \in S \times S \mid s' \in \text{Post}(s)\}$

  \[ \Rightarrow \text{omit all state and transition labels in } TS \text{ and ignore being initial} \]

- $Post^*(s)$ is the set of states reachable $G(TS)$ from $s$

  \[ Post^*(C) = \bigcup_{s \in C} Post^*(s) \quad \text{for } C \subseteq S \]

- The notations $Pre^*(s)$ and $Pre^*(C)$ have analogous meaning

- The set of reachable states: $Reach(TS) = Post^*(I)$
Path fragments

- A path fragment is an execution fragment without actions.

- A *finite path fragment* $\pi$ of $TS$ is a state sequence:
  \[ \pi = s_0 s_1 \ldots s_n \] such that $s_{i+1} \in \text{Post}(s_i)$ for all $0 \leq i < n$ where $n \geq 0$.

- An *infinite path fragment* $\pi$ of $TS$ is an infinite state sequence:
  \[ \pi = s_0 s_1 s_2 \ldots \] such that $s_{i+1} \in \text{Post}(s_i)$ for all $i \geq 0$.

- A *path* of $TS$ is an initial, maximal path fragment:
  - a *maximal* path fragment is either finite ending in a terminal state, or infinite.
  - a path fragment is *initial* if $s_0 \in I$.
  - $\text{Paths}(s)$ is the set of maximal path fragments $\pi$ with $\text{first}(\pi) = s$.
Semaphore-based mutual exclusion

\[ PG_1 : \]
\[
\text{noncrit}_1 \rightarrow \text{wait}_1 \rightarrow \begin{cases} 
\text{crit}_1 & \text{if } y > 0 \\
\text{noncrit}_1 & \text{if } y = 0
\end{cases}
\]
\[
y := y + 1
\]

\[ PG_2 : \]
\[
\text{noncrit}_2 \rightarrow \text{wait}_2 \rightarrow \begin{cases} 
\text{crit}_2 & \text{if } y > 0 \\
\text{noncrit}_2 & \text{if } y = 0
\end{cases}
\]
\[
y := y + 1
\]

\( y = 0 \) means “lock is currently possessed”; \( y = 1 \) means “lock is free”
Transition system $TS(PG_1 ||| PG_2)$
Example paths
Traces

- Actions are mainly used to model the (possibility of) interaction
  - synchronous or asynchronous communication

- Here, focus on the states that are visited during executions
  - the states themselves are not “observable”, but just their atomic propositions

- Consider sequences of the form $L(s_0) \, L(s_1) \, L(s_2) \ldots$
  - just register the (set of) atomic propositions that are valid along the execution
  - instead of execution $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \ldots$
  $\Rightarrow$ this is called a trace

- For a transition system without terminal states:
  - traces are infinite words over the alphabet $2^{AP}$, i.e., they are in $(2^{AP})^\omega$
Traces

- Let transition system $\mathit{TS} = (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ without terminal states
  - all maximal paths (and executions) are infinite

- The trace of path fragment $\pi = s_0 s_1 \ldots$ is $\mathit{trace}(\pi) = L(s_0) L(s_1) \ldots$
  - the trace of $\hat{\pi} = s_0 s_1 \ldots s_n$ is $\mathit{trace}(\hat{\pi}) = L(s_0) L(s_1) \ldots L(s_n)$

- The set of traces of a set $\Pi$ of paths: $\mathit{trace}(\Pi) = \{\mathit{trace}(\pi) \mid \pi \in \Pi\}$

- $\mathit{Traces}(s) = \mathit{trace}(\mathit{Paths}(s))$ \hspace{1cm} $\mathit{Traces}(\mathit{TS}) = \bigcup_{s \in I} \mathit{Traces}(s)$

- $\mathit{Traces}_{\mathit{fin}}(s) = \mathit{trace}(\mathit{Paths}_{\mathit{fin}}(s))$ \hspace{1cm} $\mathit{Traces}_{\mathit{fin}}(\mathit{TS}) = \bigcup_{s \in I} \mathit{Traces}_{\mathit{fin}}(s)$
Example traces

Let $AP = \{ \text{crit}_1, \text{crit}_2 \}$

Example path:

\[
\pi = \langle n_1, n_2, y = 1 \rangle \rightarrow \langle w_1, n_2, y = 1 \rangle \rightarrow \langle c_1, n_2, y = 0 \rangle \rightarrow \\
\langle n_1, n_2, y = 1 \rangle \rightarrow \langle n_1, w_2, y = 1 \rangle \rightarrow \langle n_1, c_2, y = 0 \rangle \rightarrow \ldots
\]

The trace of this path is the infinite word:

\[
\text{trace}(\pi) = \emptyset \emptyset \{ \text{crit}_1 \} \emptyset \emptyset \{ \text{crit}_2 \} \emptyset \emptyset \{ \text{crit}_1 \} \emptyset \emptyset \{ \text{crit}_2 \} \ldots
\]

The trace of the finite path fragment:

\[
\hat{\pi} = \langle n_1, n_2, y = 1 \rangle \rightarrow \langle w_1, n_2, y = 1 \rangle \rightarrow \langle w_1, w_2, y = 1 \rangle \rightarrow \\
\langle w_1, c_2, y = 0 \rangle \rightarrow \langle w_1, n_2, y = 1 \rangle \rightarrow \langle c_1, n_2, y = 0 \rangle
\]

is:

\[
\text{trace}(\hat{\pi}) = \emptyset \emptyset \emptyset \{ \text{crit}_2 \} \emptyset \{ \text{crit}_1 \}
\]
Linear-time properties

- Linear-time properties specify the traces that a TS must exhibit
  - LT-property specifies the admissible behaviour of system under consideration
    later, a logic will be introduced for specifying LT properties

- A *linear-time property* (LT property) over $AP$ is a subset of $(2^{AP})^\omega$
  - finite words are not needed, as it is assumed that there are *no terminal states*

- $TS$ (over $AP$) *satisfies* LT property $P$ (over $AP$):

$$TS \models P \text{ if and only if } \text{Traces}(TS) \subseteq P$$

- $TS$ satisfies the LT property $P$ if all its “observable” behaviors are admissible
- state $s \in S$ satisfies $P$, notation $s \models P$, whenever $\text{Traces}(s) \subseteq P$
How to specify mutual exclusion?

“Always at most one process is in its critical section”

- Let $AP = \{ crit_1, crit_2 \}$
  - other atomic propositions are not of any relevance for this property

- Formalization as LT property

  $$P_{mutex} = \text{set of infinite words } A_0 A_1 A_2 \ldots \text{ with } \{ crit_1, crit_2 \} \not\subseteq A_i \text{ for all } 0 \leq i$$

- Contained in $P_{mutex}$ are e.g., the infinite words:
  - $(\{ crit_1 \} \{ crit_2 \})^\omega$ and $\{ crit_1 \} \{ crit_1 \} \{ crit_1 \} \ldots$ and $\emptyset \emptyset \emptyset \ldots$
  - but not $\{ crit_1 \} \emptyset \{ crit_1, crit_2 \} \ldots$ or $\emptyset \{ crit_1 \}, \emptyset \emptyset \{ crit_1, crit_2 \} \emptyset \ldots$

  Does the semaphore-based algorithm satisfy $P_{mutex}$?
Does the semaphore-based algorithm satisfy $P_{mutex}$?

Yes as there is no reachable state labeled with $\{ crit_1, crit_2 \}$
How to specify starvation freedom?

“A process that wants to enter the critical section is eventually able to do so”

- Let $AP = \{ \text{wait}_1, \text{crit}_1, \text{wait}_2, \text{crit}_2 \}$

- Formalization as LT-property

$$P_{nostarve} = \text{set of infinite words } A_0 A_1 A_2 \ldots \text{such that:}$$

$$\left( \exists j. \text{wait}_i \in A_j \right) \Rightarrow \left( \forall j. \text{crit}_i \in A_j \right) \text{ for each } i \in \{1, 2\}$$

there exist infinitely many:

$$\left( \exists j. \text{wait}_i \in A_j \right) \equiv (\forall k \geq 0. \exists j > k. \text{wait}_i \in A_j)$$

Does the semaphore-based algorithm satisfy $P_{nostarve}$?
Does the semaphore-based algorithm satisfy $P_{nostarve}$?

No. Trace $\emptyset \{ \{ \text{wait}_2 \} \{ \text{wait}_1, \text{wait}_2 \} \{ \text{crit}_1, \text{wait}_2 \} \}^\omega \in \text{Traces}(TS)$, but $\not\in P_{nostarve}$
Mutual exclusion algorithm revisited

This algorithm satisfies $P_{mutex}$
Refining mutual exclusion algorithm

this variant algorithm with an omitted edge also satisfies $P_{mutex}$
Trace equivalence and LT properties

For \( TS \) and \( TS' \) be transition systems (over \( AP \) without terminal states:

\[
\text{Traces}(TS) \subseteq \text{Traces}(TS') \quad \text{if and only if} \quad \text{for any LT property } P: TS' \models P \implies TS \models P
\]

\[
\text{Traces}(TS) = \text{Traces}(TS') \quad \text{if and only if} \quad TS \text{ and } TS' \text{ satisfy the same LT properties}
\]
Two beverage vending machines

\[ AP = \{ \text{pay, sprite, beer} \} \]

there is no LT-property that can distinguish between these machines
Invariants

- Safety properties \( \approx \) “nothing bad should happen” \[\text{[Lamport 1977]}\]

- Typical safety property: mutual exclusion property
  - the bad thing (having \( > 1 \) process in the critical section) never occurs

- Another typical safety property is deadlock freedom

\( \Rightarrow \) These properties are in fact invariants

- An invariant is an LT property
  - that is given by a condition \( \Phi \) for the states
  - and requires that \( \Phi \) holds for all reachable states
  - e.g., for mutex property \( \Phi \equiv \neg \text{crit}_1 \lor \neg \text{crit}_2 \)
#5b: Linear-time properties

## Invariants

- An LT property $P_{\text{inv}}$ over $AP$ is an *invariant* if there is a propositional logic formula $\Phi$ over $AP$ such that:

  $$P_{\text{inv}} = \{ A_0A_1A_2\ldots \in (2^{AP})^\omega \mid \forall j \geq 0. \ A_j \models \Phi \}$$

  - $\Phi$ is called an *invariant condition* of $P_{\text{inv}}$

- Note that

  $$TS \models P_{\text{inv}} \iff \text{trace}(\pi) \in P_{\text{inv}} \text{ for all paths } \pi \text{ in } TS$$

  $$\iff \ L(s) \models \Phi \text{ for all states } s \text{ that belong to a path of } TS$$

  $$\iff \ L(s) \models \Phi \text{ for all states } s \in \text{Reach}(TS)$$

- $\Phi$ has to be fulfilled by all initial states and

  - satisfaction of $\Phi$ is invariant under all transitions in the reachable fragment of $TS$
Checking an invariant

• Checking an invariant for the propositional formula $\Phi$

  \[\Rightarrow \text{ check the validity of } \Phi \text{ in every reachable state} \]
  \[\Rightarrow \text{ use a slight modification of standard graph traversal algorithms (DFS and BFS)} \]
  \[- \text{ provided the given transition system } TS \text{ is } \textit{finite} \]

• Perform a forward depth-first search

  \[- \text{ at least one state } s \text{ is found with } s \not\models \Phi \Rightarrow \text{ the invariance of } \Phi \text{ is violated} \]

• Alternative: backward search

  \[- \text{ starts with all states where } \Phi \text{ does not hold} \]
  \[- \text{ calculates (by a DFS or BFS) the set } \bigcup_{s \in S, s \not\models \Phi} Pre^*(s) \]
A naive invariant checking algorithm

Input: finite transition system \( TS \) and propositional formula \( \Phi \)
Output: true if \( TS \) satisfies the invariant "always \( \Phi \)”, otherwise false

\( \begin{align*}
\text{set of state } R & := \emptyset; \\
\text{stack of state } U & := \varepsilon; \\
\text{bool } b & := \text{true;}
\end{align*} \)

\( \begin{align*}
\text{for all } s \in I \text{ do } & \quad \begin{align*}
\text{if } s \not\in R \text{ then } & \quad \text{visit}(s) \\
\text{fi } & \\
\text{od } & \\
\text{return } b
\end{align*} \)

(* the set of visited states *)

(* the empty stack *)

(* all states in \( R \) satisfy \( \Phi \) *)

(* perform a dfs for each unvisited initial state *)
A naive invariant checking algorithm

procedure visit (state s)
    push(s, U);
    R := R ∪ { s };
    repeat
        s′ := top(U);
        if Post(s′) ⊆ R then
            pop(U);
            b := b ∧ (s′ |= Φ);
        else
            let s'' ∈ Post(s′) \ R
            push(s'', U);
            R := R ∪ { s'' };
        end if
    until (U = ε)
endproc

error indication is state refuting Φ

initial path fragment s_0 s_1 s_2 . . . s_n with s_i |= Φ (i ≠ n) and s_n ⊭ Φ is more useful
Invariant checking by DFS

**Input:** finite transition system $TS$ and propositional formula $\Phi$

**Output:** "yes" if $TS \models \text{"always } \Phi\text{"}$, otherwise "no" plus a counterexample

```
set of states $R := \emptyset$; (* the set of reachable states *)
stack of states $U := \epsilon$; (* the empty stack *)
bool $b := \text{true}$; (* all states in } R \text{ satisfy } \Phi (*)
while ($I \setminus R \neq \emptyset$ $\land b$) do
  let $s \in I \setminus R$; (* choose an arbitrary initial state not in } R *)
  visit($s$); (* perform a DFS for each unvisited initial state *)
  od
if $b$ then
  return("yes") (* $TS \models \text{"always } \Phi\text{"} *)
else
  return("no", reverse($U$)) (* counterexample arises from the stack content *)
fi
```
Invariant checking by DFS

procedure visit (state s)
    push(s, U);
    \( R := R \cup \{ s \} \);
    repeat
        \( s' := \text{top}(U) \);
        if Post(s') \subseteq R then
            pop(U);
            \( b := b \land (s' \models \Phi) \);
        else
            let \( s'' \in \text{Post}(s') \setminus R \);
            push(s'', U);
            \( R := R \cup \{ s'' \} \);
        fi
    until ((U = \varepsilon) \lor \neg b)
endproc

(* push s on the stack *)
(* mark s as reachable *)
(* check validity of \( \Phi \) in \( s' \) *)
(* state \( s'' \) is a new reachable state *)
Time complexity

- Under the assumption that
  - $s' \in Post(s)$ can be encountered in time $\Theta(|Post(s)|)$
  - this holds for a representation of $Post(s)$ by adjacency lists

- The time complexity for invariant checking is $O(N \times (1 + |\Phi|) + M)$
  - where $N$ denotes the number of reachable states, and
  - $M = \sum_{s \in S} |Post(s)|$ the number of transitions in the reachable fragment of $TS$

- The adjacency lists are typically given implicitly
  - e.g., by a syntactic description of the concurrent processes as program graphs
  - $Post(s)$ is obtained by the rules for the transition relation