Formal Specification and Verification

Temporal logic (2)

17.06.2014

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Formal specification

- Specification for program/system
- Specification for properties of program/system

Verification tasks:

Check that the specification of the program/system has the required properties.

Temporal logic

Motivation

The purpose of temporal logic (TL) is:

- reasoning about time (in philosophy), and
- reasoning about the behaviour of systems evolving over time (in computer science).

Models of time

Which flow of time should we use?

This depends on the application!

Models of time

The main application of TL in computer science is the verification of finite-state reactive and concurrent systems.

A state is a snapshot of the system capturing the values of the variables at an instant of time.

- **Finite-state systems.**
  Finite-state systems can only take finitely many states.
  (Often, infinite-state systems can be abstracted into finite-state ones by grouping the states into a finite number of partitions.)

Models of time

The main application of TL in computer science is the verification of finite-state reactive and concurrent systems.

A state is a snapshot of the system capturing the values of the variables at an instant of time.

- **Reactive Systems.**
  A reactive system interacts with the environment frequently and usually does not terminate. Its correctness is defined via these interactions. This is in contrast to a classical algorithm that takes an input initially and then eventually terminates producing a result.

Models of time

The main application of TL in computer science is the verification of finite-state reactive and concurrent systems.

A state is a snapshot of the system capturing the values of the variables at an instant of time.

- **Concurrent Systems.**
  Systems consisting of multiple, interacting processes. One process does not know about the internal state of the others. May be viewed as a collection of reactive systems.

Models of time

The main application of TL in computer science is the verification of finite-state reactive and concurrent systems.

Task: Verificaton.

Given the (formal) description of a system and of its intended behaviour, check whether the system indeed complies with this behaviour.

Transition systems

We use an abstract model of reactive and concurrent systems.

Definition (Transition system, simplified version)

Let Π be a finite set of propositional variables.

A transition system is a tuple $(S, \rightarrow, S_i, L)$ with

- S a non-empty set of states;
- $\rightarrow \subseteq S \times S$ is a transition relation that is total, i.e. for each state $s \in S$, there is a state $s' \in S$ such that $s \rightarrow s'$;
- $S_i \subseteq S$ is a set of initial states;
- $L : S \rightarrow \{0, 1\}^{AP}$ is a valuation function which we will also regard as a function $L : AP \times S \rightarrow \{0, 1\}$

Example

Consider the following simple mutual-exclusion protocol:

```
    task body ProcA is
    begin
    loop
(0) Non_Critical_Section_A;
(1) loop [exit when Turn = 0] end loop;
(2) Critical_Section_A;
(3) Turn := 1;
    end loop;
    end ProcA;

    task body ProcB is
    begin
    loop
(0) Non_Critical_Section_B;
(1) loop [exit when Turn = 1] end loop;
(2) Critical_Section_B;
(3) Turn := 0;
    end loop;
    end ProcA;
```

Assume that the processes run asynchronously, i.e., either Process A or B makes a step, but not both. The order of executions is undetermined.

Example

$$\Pi = \{(T = i) \mid i \in \{0, 1\}\} \cup \{(X = i) \mid X \in \{A, B\}, i \in \{0, 1, 2, 3\}\}$$

$(T = i)$ means that Turn is set to i, and

$(X = i)$ means the process X is currently in Line i.

Example

We define the following transition system $(S, \rightarrow, S_i, L)$:

- $S = \{0, 1\} \times \{0, 1, 2, 3\} \times \{0, 1, 2, 3\}$

  $(t, i, j) \in S$: state in which Turn $= t$, A is at line i, B is at line j

- $S_i = \{(0, 0, 0), (1, 0, 0)\}$

- $\rightarrow = R_A \cup R_B$, where

  $$R_A = \{((t, i, j), (t', i', j)) \mid \begin{array}{l} (i \in \{0, 2, 3\} \wedge t = t') \rightarrow i' = i + 1\ (mod 4), \\ t = 0, i = 1 \rightarrow i' = 2 \\ t = 1, i = 1 \rightarrow i' = 1 \\ i = 3 \rightarrow t' = 1\} \end{array}$$

  and R_B is defined similarly

- $L((T = t'), (t, i, j)) = 1$ iff $t' = t$

  $L((A = i'), (t, i, j)) = 1$ iff $i' = i$

  $L((B = j'), (t, i, j)) = 1$ iff $j' = j$

Computations

Let $TS = (S, \rightarrow, S_i, L)$ be a transition system.

A computation (or execution) of TS is an infinite sequence $s_0 s_1 \ldots$ of states such that $s_0 \in S_i$ and $s_i \rightarrow s_{i+1}$ for all $i \geq 0$.

Example: computation (execution) of the transition system from the previous example:

$(0, 0, 0), (0, 1, 0), (0, 1, 1), (0, 2, 1), (0, 3, 1), (1, 0, 1), (1, 0, 2), \ldots$

This corresponds to an (asynchronous) execution of the concurrent system with Processes A and B.

Note that our formalization allows computations that are unfair, e.g., in which Process B is never executed. Such issues are not adressed on the level of transition systems.

Example

Interesting properties that can be verified in this Example include the following:

- **Mutual exclusion:** can A and B be at Line (2) at the same time?

- **Guaranteed accessibility:** if process $X \in \{A, B\}$ is at Line (2), is it guaranteed that it will eventually reach Line (3)?

  (holds, but only in computations that execute both Process A and Process B infinitely often)

Later, we will express such properties as temporal logic formulas.

Computation trees

Transition systems can be non-deterministic, i.e., for an $s \in S$, the set $\{s' \mid s \to s'\}$ can have arbitrary cardinality > 0.

Thus, in general there is more than a single computation.

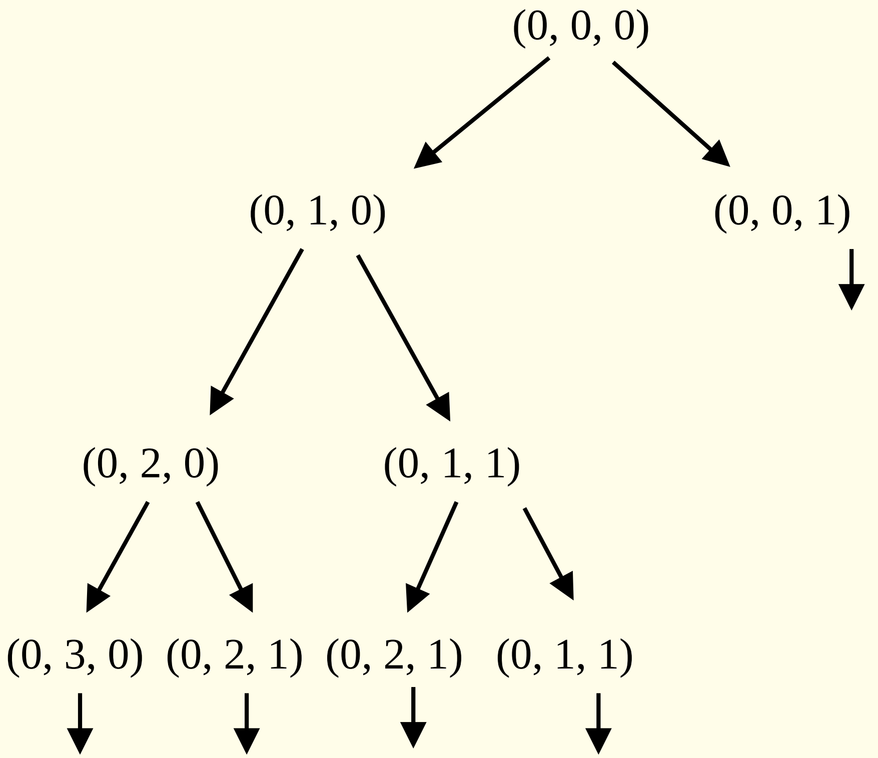
Instead of considering single computations in isolation, we can arrange all of them in a computation tree.

Informally, for $s \in S_i$, the (infinite) computation tree $T(TS, s)$ of TS at $s \in S$ is inductively constructed as follows:

- use s as the root node;
- for each leaf s' of the tree, add successors $\{t \in S \mid s' \to t\}$.

Computation trees

The computation tree of the transition system from the previous example starting at state (0, 0, 0) is:



Linear Time Logic

Syntax

Π set of propositional variables.

The set of LTL (linear time logic) formulae is the smallest set such that:

- $\bot, \top$ and each propositional variable $P \in \Pi$ are formulae;
- if F, G are formulae, then so are $F \wedge G, F \vee G, \neg F$;
- if F, G are formulae, then so are $\bigcirc F$ and $F \mathcal{U} G$

Linear Time Logic

Syntax

Π set of propositional variables.

The set of LTL (linear time logic) formulae is the smallest set such that:

- $\bot, \top$ and each propositional variable $P \in \Pi$ are formulae;
- if F, G are formulae, then so are $F \wedge G, F \vee G, \neg F$;
- if F, G are formulae, then so are $\bigcirc F$ and $F \mathcal{U} G$

Remark: Instead of $\bigcirc F$ in some books also XF is used.

Linear Time Logic

Semantics

- Transition systems $(S, \rightarrow, L)$
  (with the property that for every $s \in S$ there exists $s' \in S$ with $s \rightarrow s'$ i.e. no state of the system can "deadlock"[a])

  Transition systems are also simply called models in what follows.

---

[a] This is a technical convenience, and in fact it does not represent any real restriction on the systems we can model. If a system did deadlock, we could always add an extra state s_d representing deadlock, together with new transitions $s \rightarrow s_d$ for each s which was a deadlock in the old system, as well as $s_d \rightarrow s_d$.

Linear Time Logic

Semantics

- Transition systems $(S, \rightarrow, L)$
  (with the property that for every $s \in S$ there exists $s' \in S$ with $s \rightarrow s'$
  i.e. no state of the system can "deadlock"[a])

  Transition systems are also simply called models in what follows.

- Computation (execution, path) in a model $(S, \rightarrow, L)$
  infinite sequence of states $\pi = s_0, s_1, s_2, \ldots$ in S such that for each $i \geq 0$, $s_i \rightarrow s_{i+1}$.
  We write the path as $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$.

[a] This is a technical convenience, and in fact it does not represent any real restriction on the systems we can model. If a system did deadlock, we could always add an extra state s_d representing deadlock, together with new transitions $s \rightarrow s_d$ for each s which was a deadlock in the old system, as well as $s_d \rightarrow s_d$.

Linear Time Logic

Consider the path $\pi = s_0 \to s_1 \to \ldots.$

It represents a possible future of our system.

We write π^i for the suffix starting at s_i, e.g.,

$$\pi^3 = s_3 \to s_4 \to \ldots.$$

Linear Time Logic

Semantics

Let $TS = (S, \rightarrow, L)$ be a model and $\pi = s_0 \rightarrow ...$ be a path in TS.

Whether π satisfies an LTL formula is defined by the satisfaction relation $\models$ as follows:

- $\pi \models \top$
- $\pi \not\models \bot$
- $\pi \models p$ iff $p \in L(s_0)$, if $p \in \Pi$
- $\pi \models \neg F$ iff $\pi \not\models F$
- $\pi \models F \wedge G$ iff $\pi \models F$ and $\pi \models G$
- $\pi \models F \vee G$ iff $\pi \models F$ or $\pi \models G$
- $\pi \models \bigcirc F$ iff $\pi^1 \models F$
- $\pi \models F \mathcal{U} G$ iff $\exists m \geq 0$ $s.t.$ $\pi^m \models G$ and $\forall k \in \{0, \ldots, m-1\} : \pi^k \models F$

Linear Time Logic

Alternative way of defining the semantics:

An LTL structure M is an infinite sequence $S_0 S_1 \ldots$ with $S_i \subseteq \Pi$ for all $i \geq 0$. We define satisfaction of LTL formulas in M at time points $n \in \mathbb{N}$ as follows:

- $M, n \models p$ iff $p \in S_n$, $\qquad$ if $p \in \Pi$
- $M, n \models F \wedge G$ iff $M, n \models F$ and $M, n \models G$
- $M, n \models F \vee G$ iff $M, n \models F$ or $M, n \models G$
- $M, n \models \neg F$ iff $M, n \not\models F$
- $M, n \models \bigcirc F$ iff $M, n+1 \models F$
- $M, n \models F \mathcal{U} G$ iff $\exists m \geq n$ $s.t.$ $M, m \models G$ and $\forall k \in \{n, \ldots, m-1\} : M, k \models F$

Note that the time flow $(\mathbb{N}, <)$ is implicit.

Transition systems and LTL models

The connection between transition systems and LTL structures is as follows:

Every computation (evolution, path) of a transition system $s_0 \rightarrow s_1 \dots$ gives rise to an LTL structure.

To see this, let $TS = (S, \rightarrow, L)$ be a transition system.

A computation $s_0, s_1, ...$ of TS induces an LTL structure $L(s_0)L(s_1)\dots$

Such an LTL structure is called a trace of TS.

Abbreviations

- The future diamond

  $\Diamond\phi := \top\mathcal{U}\phi$

  $\pi \models \Diamond\phi$ iff $\exists m \geq 0 : \pi^m \models \phi$

- The future box

  $\Box\phi := \neg\Diamond\neg\phi$

  $\pi \models \Box\phi$ iff $\forall m \geq 0 : \pi^m \models \phi$

Abbreviations

- The future diamond

  $\Diamond\phi := \top\mathcal{U}\phi$ Sometimes denoted also $F\phi$

  $\pi \models \Diamond\phi$ iff $\exists m \geq 0 : \pi^m \models \phi$ $M, n \models \Diamond\phi$ iff $\exists m \geq n : M, m \models \phi$

- The future box

  $\Box\phi := \neg\Diamond\neg\phi$ Sometimes also denoted $G\phi$

  $\pi \models \Box\phi$ iff $\forall m \geq 0 : \pi^m \models \phi$ $M, n \models \Box\phi$ iff $\forall m \geq n : M, m \models \phi$

Abbreviations

- The infinitely often operator

  $\Diamond^{\infty}\phi := \Box\Diamond\phi$

  $\pi \models \Diamond^{\infty}\phi$ iff $\{m \geq 0 \mid \pi^m \models \phi\}$ is infinite

  $M, n \models \Diamond^{\infty}\phi$ iff $\{m \geq n \mid M, m \models \phi\}$ is infinite

- The almost everywhere operator

  $\Box^{\infty}\phi := \Diamond\Box\phi$

  $\pi \models \Box^{\infty}\phi$ iff $\{m \geq 0 \mid \pi^m \not\models \phi\}$ is finite.

  $M, n \models \Box^{\infty}\phi$ iff $\{m \geq n \mid M, m \not\models \phi\}$ is finite.

Abbreviations

- The release operator

  $\phi \mathcal{R} \psi := \neg(\neg\phi \mathcal{U} \neg\psi)$

  $\pi \models \phi \mathcal{R} \psi$ iff ($\exists m \geq 0 : \pi^m \models \phi$ and $\forall k < m$: $\pi^k \models \psi$) or
  ($\forall k \geq 0 : \pi^k \models \psi$)

  $M, n \models \phi \mathcal{R} \psi$ iff ($\exists m \geq n : M, m \models \phi$ and $\forall k < m : M, m \models \psi$) or
  ($\forall k \geq m : M, k \models \psi$)

  Read as
  "ψ always holds unless released by ϕ" i.e.,
  "ψ holds permanently up to and including the first point where ϕ holds (such an ϕ-point need not exist at all)".