Formal Specification and Verification

Temporal logic (1)

13.12.2016

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).

How to define a TL?

To define a temporal logic (TL), we need to specify:

- the language for talking about time or temporal systems;
- our model of time.

Motivation

What model of time should we use?

What is the structure of time?

Motivation

What model of time should we use?

What is the structure of time?

A very liberal definition:

A flow of time is a pair $(T, <)$, where T is a non-empty set of time points, and $<$ is an irreflexive and transitive binary relation on T.

Depending on the intended application, we often require additional properties. One of the most fundamental decisions is whether or not time should be linear.

$(T, <)$ is linear if, for all $x, y \in T$ with $x \neq y$, we have $x < y$ or $y < x$.

Models of time

Important additional properties for linear flows of time:

Boundedness: We have four options by combining:

- *Bounded to the past:* there exists an $x \in T$ such that $x \leq y$ for all $y \in T$ (genesis).
- *Bounded to the future:* there exists a an $x \in T$ such that $y \leq x$ for all $y \in T$ (doomsday).

Discreteness: Existence of direct predecessors and successors:

- If $x \in T$ is not genesis, then there exists a $y \in T$ such that $y < x$ and $y < z < x$ holds for no $z \in T$.
- If $x \in T$ is not doomsday, then there exists a $y \in T$ such that $x < y$ and $x < z < y$ holds for no $z \in T$.

It can be seen that one does not follow from the other.

Models of time

Important additional properties for linear flows of time:

Density: For all $x, y \in T$ with $x < y$, there is a $z \in T$ such that $x < z < y$.

Dedekind completeness: Any non-empty subset $S \subseteq T$ that has an upper bound has a least upper bound:

Definitions:

Upper bound for S: $x \in T$ with $y \leq x$ for all $y \in S$;

Least upper bound for S: upper bound x for S such that there is no $x' \in T$ with $x' < x$ and x' upper bound for S.

Models of time

The following are among the most natural linear flows of time:

- **The natural numbers $\mathbb{N}$ with the usual order $<$.**

  Linear, discrete, bounded to the past, not bounded to the future.

  Note that other flows of time have these properties as well:

  $T := \mathbb{N} \times \{0\} \cup \mathbb{Z} \times \{1\}$, where:

  $(x, a) < (y, b)$ if (i) $a < b$ or (ii) $a = b$ and $x < y$.

  NOTE: above example not Dedekind complete.

Models of time

The following are among the most natural linear flows of time:

- **The rational numbers $\mathbb{Q}$.**

  A natural dense flow of time, though with gaps (e.g. π).

  The unique countable linear dense flow of time without endpoints (up to isomorphism).

- **The real numbers $\mathbb{R}$.**

  Up to isomorphism, the unique dense, Dedekind-complete flow of time without end points that is separable:

  There exists a countable subset $D \subseteq T$ such that, for all $x, y \in T$ with $x < y$, there is a $z \in D$ with $t < z < u$.

Models of time

The alternative to linear time is branching time.

Time can be:

- **Branching to the future** reflecting that there are many possible futures;
- **Branching to the past** reflecting that many different histories are considered possible (due to incomplete knowledge).

Branching to the future and linear to the past is the most popular option

for each $x \in T$, the set $\{y \in T \mid y < x\}$ is linearly ordered by $<$.

We can identify additional properties similar to the linear case. Usually, branching time is assumed to be discrete and has a genesis.

Models of time

Which flow of time should we use?

Models of time

Which flow of time should we use?

This depends on the application!

Models of time

Which flow of time should we use?

This depends on the application!

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

Which flow of time should we use?

This depends on the application!

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

Which flow of time should we use?

This depends on the application!

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

Which flow of time should we use?

This depends on the application!

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

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

  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: