

Verification

Formal methods have been used to verify the correctness of critical aspects of various computer systems (hardware or software or a combination).

Formal verification requires:

- A framework for *modelling* the system in question, i.e., a *description language*.
- A *specification language* for describing the properties to be verified.
- A *verification method* to establish whether the system description satisfies the specification.

Several approaches to verification have been developed that differ in whether they are model-based or proof-based, in the degree of automation they realize, and in the expressiveness of the specification language they allow.

We will discuss a verification method, called *model checking*, that is intended to be used for concurrent, reactive systems and illustrates an automatic, model-based, property-verification approach.

Linear-Time Temporal Logic

Linear-time temporal logic (LTL) provides connectives that allow one to refer to the future. Time is viewed as a sequence of time instances (and in that sense we speak of a “linear-time” logic) and formulas are interpreted with reference to such (computation) paths. (We will also consider *branching-time* logics where formulas are interpreted with respect to computation trees.)

(Syntactically correct) LTL formulas are recursively defined as follows:

$$\begin{aligned}\phi & ::= P \mid \top \mid \perp \mid (\neg\phi) \mid (\phi \wedge \phi) \\ & \quad \mid (X\phi) \mid (F\phi) \mid (G\phi) \\ & \quad \mid (\phi U\phi) \mid (\phi W\phi) \mid (\phi R\phi) \\ P & ::= p_1 \mid p_2 \mid p_3 \mid \dots\end{aligned}$$

The connectives X , F , G , U , W , and R are called *temporal connectives*. Intuitively, X means “next state,” F means “some future state,” and G means “all future states” (i.e., “globally”). The connectives U , W , and R are called “until,” “weak-until,” and “release” respectively.

Transition Systems

Definition

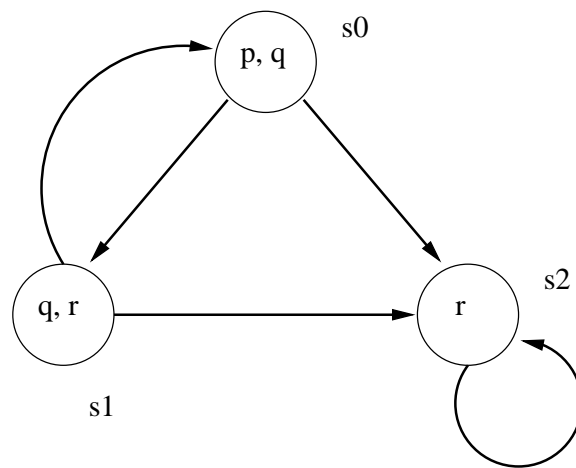
A *transition system* is a triple $\mathcal{M} = (S, \rightarrow, L)$, where

1. S is a set, the elements of which are called *states*;
2. \rightarrow is a subset of $S \times S$, called the *transition relation*, such that for every $s \in S$ there is an $s' \in S$ with $s \rightarrow s'$; and
3. L is a function, called a *labelling function*, from S to the powerset $\mathcal{P}(A)$ of the set A of all atomic descriptions.

For example, suppose the atomic descriptions are p , q , and r . Let S be the set $\{s_0, s_1, s_2\}$; L be a labelling function with $L(s_0) = \{p, q\}$, $L(s_1) = \{q, r\}$, and $L(s_2) = \{r\}$; and \rightarrow be a transition relation with $s_0 \rightarrow s_1$, $s_0 \rightarrow s_2$, $s_1 \rightarrow s_0$, $s_1 \rightarrow s_2$, and $s_2 \rightarrow s_2$. Then $\mathcal{M} = (S, \rightarrow, L)$ is a transition system.

Representation of Transition Systems

Transition systems are usually represented as labelled, directed graphs.



Transition systems are Kripke models with an accessibility relation where each possible world (state) has at least one successor. Computationally this constraint implies that no state can “deadlock.” This is not a severe restriction as one can always add an additional state, say s_d , representing deadlock, together with transitions $s_d \rightarrow s_d$ and $s \rightarrow s_d$, for all states s that are deadlocks in the original transition relation.

Semantics of LTL

The semantics of LTL formulas is defined with respect to computation paths of a transition system $\mathcal{M} = (S, \rightarrow, L)$. We define a satisfaction relation

$$\pi \models \phi$$

for computation paths π and LTL formulas ϕ by structural induction:

1. $\pi \models p$ iff $p \in L(s)$, where s is the first state of π .
2. $\pi \models \neg\phi$ iff $\pi \not\models \phi$.
3. $\pi \models \phi_1 \wedge \phi_2$ iff $\pi \models \phi_1$ and $\pi \models \phi_2$.
4. $\pi \models X\phi$ iff $\pi^2 \models \phi$.
5. $\pi \models G\phi$ iff, for all $i \geq 1$, $\pi^i \models \phi$.
6. $\pi \models F\phi$ iff, for some $i \geq 1$, $\pi^i \models \phi$.
7. $\pi \models \phi U \psi$ iff there is some $i \geq 1$, such that $\pi^i \models \psi$ and for all j with $1 \leq j < i$, $\pi^j \models \phi$.
8. $\pi \models \phi W \psi$ iff either there is some $i \geq 1$, such that $\pi^i \models \psi$ and for all j with $1 \leq j < i$, $\pi^j \models \phi$; or else $\pi^k \models \phi$, for all $k \geq 1$.
9. $\pi \models \phi R \psi$ iff either there is some $i \geq 1$, such that $\pi^i \models \phi$ and for all j with $1 \leq j \leq i$, $\pi^j \models \psi$; or else $\pi^k \models \psi$, for all $k \geq 1$.

LTL Equivalences

Two LTL formulas ϕ and ψ are said to be (*semantically equivalent*), written $\phi \equiv \psi$, iff they are true for the same paths.

We also say that an LTL formula ϕ is *satisfied* in a state s if it is true for all paths starting at s .

One can easily establish various equivalences for LTL, including the following:

$$\begin{aligned} G\phi &\equiv \neg F\neg\phi \\ X\phi &\equiv \neg X\neg\phi \\ F(\phi \vee \psi) &\equiv F\phi \vee F\psi \\ G(\phi \wedge \psi) &\equiv G\phi \wedge G\psi \\ F\phi &\equiv \top U\phi \\ G\phi &\equiv \perp R\phi \\ \neg(\phi U\psi) &\equiv \neg\phi R\neg\psi \\ \neg(\phi R\psi) &\equiv \neg\phi U\neg\psi \\ \phi W\psi &\equiv \phi U\psi \vee G\phi \\ \phi W\psi &\equiv \psi R(\phi \vee \psi) \\ \phi R\psi &\equiv \psi W(\phi \wedge \psi) \end{aligned}$$

Another important equivalence is:

$$\phi U \psi \equiv \neg(\neg\psi U (\neg\phi \wedge \neg\psi)) \wedge F\psi,$$

which holds for all LTL formulas ϕ and ψ .

Adequate Sets of Connectives

The basic equivalences for LTL formulas listed previously, and the fact that propositional equivalences carry over to LTL formulas in general, imply that all LTL formulas can be expressed in equivalent form with a selected few of the connectives.

Theorem

The following sets of connectives are adequate for LTL: $\{U, X\}$, $\{R, X\}$, and $\{W, X\}$.

Example: Mutual Exclusion

We will next design a transition system to *model* a situation in which several processes may share a common resource but are not allowed access to it at the same time.

For each process certain *critical sections* are identified that include all the access to the shared resource. Only one process may be in its critical section at any time. We want to find a *protocol* for determining which process is allowed to enter its critical section at which time.

We then need to *verify* that the proposed solution satisfies certain expected properties, including the following:

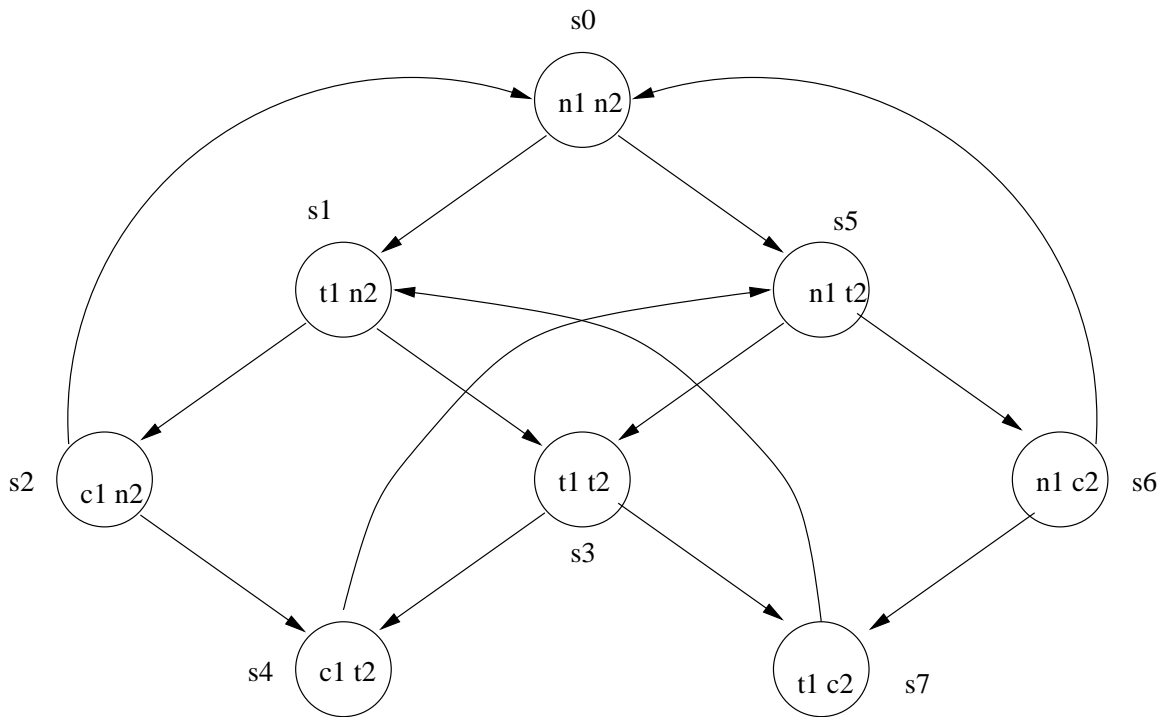
Safety: The protocol allows only one process to be in its critical section at any time.

Liveness: Whenever a process wants to enter its critical section, it will eventually be permitted to do so.

Non-blocking: A process can always request to enter its critical section.

No strict sequencing: Processes need not enter their critical section in strict sequence.

First Model



Formal Description of Properties

We next try to formalize the desired properties in LTL.

Safety:

$$G\neg(c_1 \wedge c_2)$$

Liveness:

$$G(t_1 \rightarrow Fc_1)$$

Non-blocking: Unfortunately this property can not be expressed in LTL, though it can be expressed in other logics we discuss.

No strict sequencing: We can not express this property directly, but we can formalize its complement:

$$G(c_1 \rightarrow c_1W(\neg c_1 \wedge \neg c_1Wc_2)).$$

Note that the proposed model does not satisfy the liveness property. We refine it by splitting the state s_3 into two states so as to ensure liveness.

Second Model

