

Formal Verification Methods

Lectures by Doron Peled
Summary by Ari Feiglin (ari.feiglin@gmail.com)

Contents

1	Transition Systems	1
2	Specification Formalisms	4

1 Transition Systems

When modelling systems, one must take into consideration a variety of factors: for example, is the system sequential or concurrent? When investigating the transitions between states, how granular should they be? These questions are common questions in computer science, the terms may not be though. A sequential system is a system with only one thread of execution, while a concurrent system may be multi-threaded/multiprogrammed/multiprocessed. The granularity of a transition refers to how detailed we view the transition: is the command $x := y$ atomic? Or do the variables first need to be loaded into memory?

We now begin to discuss how we model systems.

1.0.1 Definition

A **transition system** over a first-order language \mathcal{L} is a triplet (\mathcal{S}, T, Θ) , where

- (1) \mathcal{S} is a (potentially many-sorted) \mathcal{L} -structure. The symbols of \mathcal{L} correspond to the symbols utilized within the program in question. For example, \mathcal{L} may contain the $+$ operator, $<$ relation, etc. As opposed to general first-order logic, the set of variables V is taken to be finite here. This set of variables correspond to precisely what you'd expect: the set of all variables in the program. This includes internal registers utilized by the program, called the *program counters*, for which there is one for each concurrent process, and they point to the location of the next instruction to be executed.
- (2) T is a *finite* set of **transitions**. Each transition $t \in T$ has the form ($\mathcal{T}_{\mathcal{L}}$ is the set of \mathcal{L} -terms)

$$p \longrightarrow (v_1, \dots, v_n) := (e_1, \dots, e_n) \quad (v_1, \dots, v_n \in V, e_1, \dots, e_n \in \mathcal{T}_{\mathcal{L}})$$

p is a quantifier-free formula in \mathcal{L} . Notice that even in concurrent systems, there is a single set of transitions, meaning all the transitions are grouped together.

- (3) Θ is the *initial condition*, a quantifier-free formula in \mathcal{L} .

In this model, a **state** is an assignment of the variables in V to elements of the domain of \mathcal{S} . In other words, a state is a valuation $s: V \longrightarrow S$ ($S = \text{dom}\mathcal{S}$), so \mathcal{S} together with a state form an \mathcal{L} -model. The **state space** is the set of all possible states, which can be taken to be S^V or a subset of this (if for example, S contains all the naturals, but our computer's memory is bound in size).

A transition of the form $p \longrightarrow (v_1, \dots, v_n) := (e_1, \dots, e_n)$ intuitively can execute from any state which satisfies the condition p . The condition p is called the *enabledness condition* of the transition t , and if p is satisfied by the state s , ie. $\mathcal{S}, s \models p$ (recall that \mathcal{S}, s is simply an \mathcal{L} -model), then t is said to be *enabled* at s . t transitions from a state s in which it is enabled to a state where the value of each v_i is set to e_i^S for $1 \leq i \leq n$, denoted $s' = t(s) = s[e_1/v_1, \dots, e_n/v_n]$.

Note that the assignment is simultaneous: $(x, y) := (y, x)$ has the effect of swapping the values of x and y . Allowing for simultaneous assignments may seem contrary to the idea of having transitions be atomic. But this again goes back to the notion of granularity: we decide what transitions are atomic, and it can be useful to view assignments, even simultaneous ones, as atomic.

1.0.2 Definition

Given a system (\mathcal{S}, T, Θ) , an **execution** is an infinite sequence of states s_0, s_1, s_2, \dots such that $\mathcal{S}, s_0 \models \Theta$ (we will also use the notation $s_0 \models^S \Theta$), meaning the first state satisfies the initial condition, and for every $i \geq 0$ one of the following holds:

- (1) There exists some transition $p \longrightarrow (v_1, \dots, v_n) := (e_1, \dots, e_n) \in T$ that is enabled at s_i , ie. $s_i \models^S p$, and s_{i+1} is obtained by this assignment, meaning $s_{i+1} = s_i[e_1^S/v_1, \dots, e_n^S/v_n]$.
- (2) There is no transition enabled at s_i , meaning for every transition $t \in T$ whose enabledness condition is p , $s_i \not\models^S p$. In this case, for every $j \geq i$ we set $s_j = s_i$. So in such a case, we manually extend the sequence if it can no longer be extended.

Instead of the second condition, we could add a new transition to T of the form $\neg(p_1 \vee \dots \vee p_n) \rightarrow (v := v)$ where p_1, \dots, p_n exhaust all the enabledness conditions of transitions in T , and $v \in V$ is arbitrary. Alternatively

we could allow for finite sequences of states, provided the final state enables no transition.

A state which appears in some execution of a program (system) is called *reachable*. Not every state needs to be reachable: consider a program that can hold (bounded) natural numbers with variables y_1, y_2 and the program is written in such a way that $y_1 \geq y_2$ always. But the state $s[y_1] = 1$ and $s[y_2] = 2$ is a valid, yet unreachable, state.

We can view the execution of a system as a *scheduler* which can generate interleaved sequences (sequences where a single transition is executed at a time)

1. **function** SCHEDULER(\mathcal{S}, T, Θ)
2. **choose** some initial state s such that $s \models^{\mathcal{S}} \Theta$
3. **while** (s has an enabled transition)
4. **choose** a transition t enabled by s
5. $s \leftarrow t(s)$
6. **end while**
 ▷ *Extend the sequence infinitely if the final state has no enabled transition*
7. **repeat** s forever
8. **end function**

This scheduler is non-deterministic as the choice for the initial state and the choices between transitions enabled at each state along the execution are made non-deterministically.

1.0.3 Example

Let us give an example of *mutual exclusion*: we have two programs sharing a shared *critical section* (here the variable **turn**):

routine PROGRAM1

1. **while** (true)
 ▷ *wait until turn is zero*
2. wait(**turn** = 0)
3. **turn** \leftarrow 1
- end while**
- end routine**

routine PROGRAM2

1. **while** (true)
 ▷ *wait until turn is one*
2. wait(**turn** = 1)
3. **turn** \leftarrow 0
- end while**
- end routine**

In this example, we have three variables: **turn**, the first program counter \mathbf{pc}_1 , and the second program counter \mathbf{pc}_2 . The transitions are as follows:

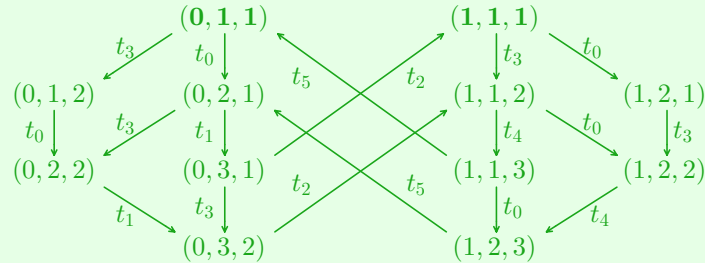
$$t_0: \mathbf{pc}_1 = 1 \longrightarrow \mathbf{pc}_1 := 2, \quad t_1: (\mathbf{pc}_1 = 2 \wedge \mathbf{turn} = 0) \longrightarrow \mathbf{pc}_1 := 3, \quad t_2: (\mathbf{pc}_1 = 3) \longrightarrow (\mathbf{pc}_1, \mathbf{turn}) := (1, 1)$$

$$t_3: \mathbf{pc}_2 = 1 \longrightarrow \mathbf{pc}_2 := 2, \quad t_4: (\mathbf{pc}_2 = 2 \wedge \mathbf{turn} = 1) \longrightarrow \mathbf{pc}_2 := 3, \quad t_5: (\mathbf{pc}_2 = 3) \longrightarrow (\mathbf{pc}_2, \mathbf{turn}) := (1, 0)$$

Then the initial condition is

$$\Theta = \mathbf{pc}_1 = 1 \wedge \mathbf{pc}_2 = 1$$

Viewing states as $(\mathbf{turn}, \mathbf{pc}_1, \mathbf{pc}_2)$, then we can draw the following diagram for the transition system, initial states are bold:



In the above example, the focus was more on the states and the possible transitions between them rather than the explicit content of each transition. We can generalize this idea to the concept of *state spaces*:

1.0.4 Definition

A **state space** is a triplet (S, Δ, I) , where S is a set of states, $\Delta \subseteq S \times S$ is the transition relations, and $I \subseteq S$ are the initial transitions. This defines a graph, called an **automaton**. A **run** of the automaton is a sequence $s_0 s_1 s_2 \dots$ such that $s_0 \in I$ is an initial state and for every $i \geq 0$, $(s_i, s_{i+1}) \in \Delta$. Such a run must be maximal, meaning it is either infinite or it reaches a state with no successor.

Sometimes we give names to the transitions in Δ in which case our state space becomes (S, Δ, Σ, I) where Δ now is a subset of $S \times \Sigma \times S$. Every transition gets its own name, so if $(s, \alpha, s'), (r, \alpha, r') \in \Delta$ then $s = r$ and $s' = r'$.

In particular, a transition system defines a state space where S is the set of all states, which are valuations $V \rightarrow \mathcal{S}$. Then $(s, s') \in \Delta$ if and only if there is a transition t enabled at s such that $s' = t(s)$. And I is the set of states which satisfy the initial condition, $I = \{s \in S \mid s \models \Theta\}$.

Suppose we have n concurrent processes, each with a variable v_i and the transitions

$$t_1^i: v_i = 1 \rightarrow v_i := 2, \quad t_2^i: v_i = 2 \rightarrow v_i := 3, \quad t_3^i: v_i = 3 \rightarrow v_i := 1$$

in other words, if v_i is 1, then it is 2, then it is 3, then it is 1. Since this is a concurrent system, we must combine these states together, and then we get that the number of global states becomes 3^n (each state is (v_1, \dots, v_n) and each v_i can take on three values). This is called *combinatorial explosion*: a relatively simple transition system becomes exponentially larger with the growth of concurrent processes.

Let us examine this above example more closely: notice how we took multiple transition systems and combined them into one. We will define this notion formally: suppose we have a transition system whose transitions T is constructed from *local* components T_1, \dots, T_n . Here, each T_i refers to a local component of the system, be it a concurrent process, a variable, or whatever. For each T_i we also have a set of *transition names* Σ_i and a *labelling function* which is a bijection $L_i: T_i \rightarrow \Sigma_i$. Importantly while the T_i s are disjoint, Σ_i need not be.

If two transitions have the same name, then we execute them together. Formally, from each global state s , we can execute all the transitions with the name d (meaning $L_i(t) = d$) provided *all of them* are enabled at s . So suppose we have the transitions $t_i: p_i \rightarrow (v_1^i, \dots, v_{n_i}^i) := (e_1^i, \dots, e_{n_i}^i)$ for $1 \leq i \leq k$ such that $L_i(t_i) = d$ for all $1 \leq i \leq k$. Then the resulting transition is

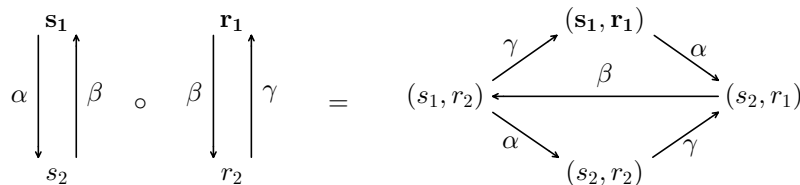
$$(p_1 \wedge \dots \wedge p_k) \rightarrow (v_1^1, \dots, v_{n_1}^1, \dots, v_1^k, \dots, v_{n_k}^k) := (e_1^1, \dots, e_{n_1}^1, \dots, e_1^k, \dots, e_{n_k}^k)$$

Now suppose that each local component can be represented as a local state space $G_i = (S_i, \Sigma_i, \Delta_i, I_i)$ which corresponds to the local component T_i . We assume that the set of local states S_i are disjoint, but Σ_i need not be. If the label α appears in both Σ_i and Σ_j , then G_i and G_j must be synchronized to perform α at the same time.

We define the operator \circ to combine two state spaces G_1 and G_2 as follows: let $G_1 \circ G_2 = (S, \Sigma, \Delta, I)$ as follows:

- (1) $S = S_1 \times S_2$, each state is a pair of a state from G_1 and G_2 ,
- (2) $\Sigma = \Sigma_1 \cup \Sigma_2$, the transition names include all the names in both G_1 and G_2 ,
- (3) The set of transitions Δ is the union of the following three sets:
 - (1) $\{((s, r), \alpha, (s', r')) \mid (s, \alpha, s') \in \Delta_1, \alpha \in \Sigma_1 \setminus \Sigma_2, r \in S_2\}$. In this case, we have a transition (s, α, s') in G_1 with no transition of the same name in G_2 , so we transition from (s, r) to (s', r) , leaving the state in G_2 unchanged.
 - (2) $\{((s, r), \beta, (s, r')) \mid (r, \beta, r') \in \Delta_2, \beta \in \Sigma_2 \setminus \Sigma_1, s \in S_1\}$. This is similar to the previous set, but for G_2 instead of G_1 .
 - (3) $\{((s, r), \gamma, (s', r')) \mid (s, \gamma, s') \in \Delta_1, \gamma \in \Sigma_1 \cap \Sigma_2, (r, r') \in \Delta_2\}$. Here, we have a transition in both G_1 and G_2 , so the transition is done simultaneously.

So for example, the following two state spaces combine together to give



We can also relate \mathbf{U} and \mathbf{V} by $\neg(\varphi \mathbf{V} \psi) \equiv (\neg\varphi) \mathbf{U} (\neg\psi)$. We will prove this directly from definition:

$$\xi^k \models \varphi \mathbf{V} \psi \iff ((\forall i \geq k) \xi^i \models \psi) \vee ((\exists j \geq k) (\forall k \leq i < j) \xi^i \models \psi \wedge \psi^j \models \varphi)$$

and so

$$\xi^k \models \neg(\varphi \mathbf{V} \psi) \iff ((\exists i \geq k) \xi^i \models \neg\psi) \wedge ((\forall j \geq k) (\exists k \leq i < j) \xi^i \models \neg\psi \vee \psi^j \models \neg\varphi)$$

So at every $j \geq k$, either $\neg\varphi$ or $\neg\psi$ holds, and eventually $\neg\psi$ holds. This just means that $\neg\varphi$ holds until $\neg\psi$ holds, ie. $(\neg\varphi) \mathbf{U} (\neg\psi)$.

Thus, we could've defined LTL with only the operators $\neg, \wedge, \bigcirc, \mathbf{U}$ (in other words, these form a *complete bundle*).

We can combine operators: for example $\Box \Diamond \varphi$ means that always, φ eventually happens; or equivalently φ happens infinitely many times. $\Diamond \Box \varphi$ means that at some point, φ will hold forever. $\bigcirc \bigcirc \varphi$ means that φ holds after two steps. Notice that $\xi \models \Diamond \varphi$ if and only if there exists some n such that $\xi \models \bigcirc^n \varphi$ (\bigcirc^n meaning $\bigcirc \cdots \bigcirc$ n times).

Let P be a system which has multiple executions, then we write $P \models \varphi$ if $\xi \models \varphi$ for all executions ξ of P . Importantlu $P \not\models \varphi$ does not imply $P \models \neg\varphi$, since one execution not satisfying φ does not mean all executions don't satisfy φ .

2.0.2 Example

Let us consider a simple model of a spring. The spring can be in one of the following three states: $\{initial, extended, extended \text{ and } malfunctioned\}$ which we denote s_1, s_2, s_3 respectively. So our propositional variables are $PV = \{extended, malfunctioned\}$. Since s_1 is neither extended nor malfunctioned, $s_1 \models \neg extended \wedge \neg malfunctioned$, $s_2 \models extended \wedge \neg malfunctioned$, $s_3 \models extended \wedge malfunctioned$.

We can transition from s_1 to s_2 via pulling the spring, and releasing the spring can either transition to s_1 or to s_3 . From s_3 we transition only to s_3 .

This system has an infinite number of executions, for example

$$\begin{aligned}\xi_0 &= s_1 s_2 s_1 s_2 s_3 s_3 s_3 \cdots \\ \xi_1 &= s_1 s_2 s_3 s_3 s_3 s_3 s_3 \cdots \\ \xi_2 &= s_1 s_2 s_1 s_2 s_1 s_2 s_1 \cdots\end{aligned}$$

Let us investigate ξ_0 :

- (1) $\xi_0 \not\models extended$ since $extended$ is a formula of the underlying logic of the LTL and so ξ_0 satisfies $extended$ if and only if its first state, s_1 , does. It does not.
- (2) $\xi_0 \models \bigcirc extended$ ("nexttime extended") since $\xi_0 \models \bigcirc extended \iff \xi_0^1 \models extended \iff s_2 \models extended$ which it does.
- (3) $\xi_0 \not\models \bigcirc \bigcirc extended$ ("nexttime nexttime extended") since ξ_0^2 begins with s_1 which does not satisfy $extended$.
- (4) $\xi_0 \models \Diamond extended$ ("eventually extended") since eventually the spring is extended (this is since $\xi_0 \models \bigcirc extended$).
- (5) $\xi_0 \not\models \Box extended$ ("always extended") since the spring is not always extended.
- (6) $\xi_0 \models \Diamond \Box extended$ ("eventually always extended") since eventually the spring remains in s_3 where it is extended.
- (7) $\xi_0 \not\models (\neg extended) \mathbf{U} malfunctioned$ ("not extended until malfunctioned") since the spring is not extended, then extended and not malfunctioned.

Let us now investigate the system P as a whole:

- (1) $P \models \Diamond extended$ since for the spring to not extend, it would need to forever remain in s_1 , which is impossible.
- (2) $P \models \Box (\neg extended \rightarrow \bigcirc extended)$ which means that always, if the spring is not extended then the next time it is. This is since in order for the spring to not be extended, it must be in s_1 , which means that the next time it is in s_2 , extended.

- (3) $P \not\models \Diamond \Box \text{extended}$, since ξ_2 is a counterexample: here we have that we never are only extended, in other words $\xi_2 \models \Box \Diamond \neg \text{extended}$.
- (4) $P \not\models \neg \Diamond \Box \text{extended}$, since ξ_0 is a counterexample: here we have that eventually we are only extended.
- (5) $P \not\models \Box (\text{extended} \rightarrow \bigcirc \neg \text{extended})$ since it is possible to go from extended to extended (s_2 to s_3). The only sequence in which this is true is ξ_2 .

We can form a Hilbert calculus to axiomatize LTL with respect to a system P . To form it we adjoin to the Hilbert calculus of \mathcal{L} (which is either first-order or propositional, usually propositional. But importantly these axioms now range over all LTL formulas, not just formulas in \mathcal{L}) the following eight axioms:

- | | | |
|---|--|--|
| (A1) $\neg \Diamond \varphi \leftrightarrow \Box \neg \varphi$ | (A2) $\Box(\varphi \rightarrow \psi) \rightarrow (\Box \varphi \rightarrow \Box \psi)$ | (A3) $\Box \varphi \rightarrow (\varphi \wedge \bigcirc \Box \varphi)$ |
| (A4) $\bigcirc \neg \varphi \leftrightarrow \neg \bigcirc \varphi$ | (A5) $\bigcirc(\varphi \rightarrow \psi) \rightarrow (\bigcirc \varphi \rightarrow \bigcirc \psi)$ | (A6) $\Box(\varphi \rightarrow \bigcirc \varphi) \rightarrow (\varphi \rightarrow \Box \varphi)$ |
| (A7) $(\varphi \cup \psi) \leftrightarrow (\psi \vee (\varphi \wedge \bigcirc(\varphi \cup \psi)))$ | (A8) $(\varphi \cup \psi) \rightarrow \Diamond \psi$ | |

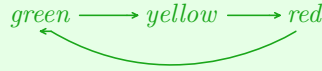
Here we take \mathbf{V} as defined by $(\varphi \mathbf{V} \psi) := \neg((\neg \varphi) \mathbf{V} (\neg \psi))$. We use the following rule of reference (temporal generalization) as well as MP

$$\frac{\varphi}{\Box \varphi}$$

meaning that if φ then $\Box \varphi$ (notice that here we do not have an initial state, and hence we obtain the soundness of generalization).

2.0.3 Example

Let us look at a model of a traffic light, which can transition between colors as follows:



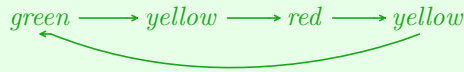
The traffic light is only ever in one color, which can be expressed in LTL as:

$$\Box(\neg(\text{green} \wedge \text{yellow}) \wedge \neg(\text{yellow} \wedge \text{red}) \wedge \neg(\text{red} \wedge \text{green}) \wedge (\text{green} \vee \text{yellow} \vee \text{red}))$$

Specifying the transition of colors can be done via

$$\Box((\text{green} \cup \text{yellow}) \vee (\text{yellow} \cup \text{red}) \vee (\text{red} \cup \text{green}))$$

Now suppose we alter the state graph to be



Specifying the colors is now harder, since $\Box(\text{yellow} \rightarrow \text{yellow} \cup \text{red})$ is no longer necessarily true. We could attempt $\Box(((\text{green} \vee \text{red}) \cup \text{yellow}) \vee (\text{yellow} \cup (\text{green} \vee \text{red})))$, but $(\text{green} \vee \text{red}) \cup \text{yellow}$ allows the light to switch between *green* and *red* before turning *yellow*. A correct specification would be

$$\begin{aligned} &\Box((\text{green} \rightarrow (\text{green} \cup (\text{yellow} \wedge (\text{yellow} \cup \text{red})))) \\ &\wedge (\text{red} \rightarrow (\text{red} \cup (\text{yellow} \wedge (\text{yellow} \cup \text{green})))) \\ &\wedge (\text{yellow} \rightarrow (\text{yellow} \cup (\text{red} \cup \text{green})))) \end{aligned}$$

The first line allows $\text{green} \rightarrow \text{yellow} \rightarrow \text{red}$, the second allows $\text{red} \rightarrow \text{yellow} \rightarrow \text{green}$. These two lines deal only with the case that the light begins on *green* or *red*, the third line deals with the case when it starts on *yellow*.