

Transducers Compose while Acceptors do not! On the Merits of Temporal Testers

Amir Pnueli

New York University and Weizmann Institute of Sciences

25MC, August, 2006

Based on Joint work with:

Yonit Kesten

Elad Shahar

Oded Maler, Dejan Nickovic

Alex Zaks

BGU

Weizmann

Verimag, Grenoble, France

NYU

The Common Practice of Model Checking

In order to model check $S \models \varphi$, for an LTL formula φ :

- Represent S as a Büchi automaton
- Construct an acceptor $\mathcal{A}_{\neg\varphi}$ — a non-deterministic Büchi automaton which accepts all sequences falsifying φ
- Construct the automaton $S \times \mathcal{A}_{\neg\varphi}$ and check that it is empty

The construction of $\mathcal{A}_{\neg\varphi}$ is done by either a tableau construction, or through alternating automata. It usually produces an explicit-state automaton.

An Alternative Approach

For a while, we have been proposing (and successfully applying) an alternative approach which is a variation on the above theme.

The main differences are:

- The automata we use are **Kripke machines**, rather than **Mealy/Moore**. These are automata in which both inputs and outputs label states. Transitions remain unlabeled.
- Both system and acceptor include both **Justice** (weak fairness) and **compassion** (strong fairness) requirements. Thus, instead of **Büchi** acceptance condition, we use **Streett** acceptance.
- Most importantly, instead of an acceptor \mathcal{A}_φ which accepts a sequence σ at position 0 iff $(\sigma, 0) \models \varphi$, we use a transducer $T[\varphi]$ which, **at every position** $j \geq 0$, outputs a value x such that, $x[j] = 1$ iff $(\sigma, j) \models \varphi$.

In this talk we will attempt to identify the advantages of using **transducers** instead of **acceptors**.

Fair Discrete Systems

As our computational model, we take **fair discrete systems**. An **FDS**

$\mathcal{D} = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ consists of:

- V – A finite set of **typed state variables**. A V -state s is an interpretation of V . Denote by Σ_V – the set of all V -states.
- Θ – An **initial condition**. A satisfiable assertion that characterizes the **initial states**.
- ρ – A **transition relation**. An assertion $\rho(V, V')$, referring to both **unprimed (current)** and **primed (next)** versions of the state variables. For example, $x' = x + 1$ corresponds to the assignment $x := x + 1$.
- $\mathcal{J} = \{J_1, \dots, J_k\}$ A set of **justice (weak fairness)** requirements. Ensure that a computation has **infinitely many J_i -states** for each J_i , $i = 1, \dots, k$.
- $\mathcal{C} = \{\langle p_1, q_1 \rangle, \dots, \langle p_n, q_n \rangle\}$ A set of **compassion (strong fairness)** requirements. **Infinitely many p_i -states** imply **infinitely many q_i -states**.

Synchronous Parallel Composition

The **synchronous parallel composition** of systems S_1 and S_2 , denoted by $S_1 \parallel S_2$, is given by the **FDS** $S = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$, where

$$\begin{array}{lclclcl} V & = & V_1 & \cup & V_2 \\ \Theta & = & \Theta_1 & \wedge & \Theta_2 \\ \rho & = & \rho_1 & \wedge & \rho_2 \\ \mathcal{J} & = & \mathcal{J}_1 & \cup & \mathcal{J}_2 \\ \mathcal{C} & = & \mathcal{C}_1 & \cup & \mathcal{C}_2 \end{array}$$

Synchronous parallel composition can be used for hardware verification, where it is the natural operator for combining two circuits into a composed circuit. Here we use it for **model checking** of **LTL** formulas.

Claim 1. *The sequence σ of V -states is a computation of the combined $S_1 \parallel S_2$ iff $\sigma \downarrow_{V_1}$ is a computation of S_1 and $\sigma \downarrow_{V_2}$ is a computation of S_2 .*

Here, $\sigma \downarrow_{V_i}$ denotes the sequence obtained from σ by restricting each of the states to a V_i -state.

Predecessors and Their Transitive Closure

For an assertion $\varphi(V)$ and a transition relation $\rho(V, V')$, we define the **existential predecessor predicate transformer**:

$$\rho \diamond \psi = \exists V' : \rho(V, V') \wedge \psi(V')$$

Obviously

$$\|\rho \diamond \varphi\| = \{s \mid s \text{ is an } \rho\text{-predecessor of a } \varphi\text{-state}\}$$

For example

$$(x' = x + 1) \diamond (x = 1) = \exists x' : x' = x + 1 \wedge x' = 1 \sim x = 0$$

The immediate predecessor transformer can be iterated to yield the **eventual predecessor transformer**:

$$\begin{aligned} \rho^* \diamond \varphi &= \\ \varphi \vee \rho \diamond \varphi \vee \rho \diamond (\rho \diamond \varphi) \vee \rho \diamond (\rho \diamond (\rho \diamond \varphi)) \vee \dots &= \\ \mu Y. (\varphi \vee \rho \diamond Y) & \end{aligned}$$

being the **minimal solution** to the **fix-point** equation $Y = \varphi \vee \rho \diamond Y$.

Checking for Feasibility

A state s is called **feasible** if it participates in some computation of \mathcal{D} . System \mathcal{D} is **feasible** if it has at least one computation. Equivalently, if one of the initial states is feasible. For simplicity, we consider systems without compassion.

Assume an assertion φ which characterizes a set of feasible states. There are several implications that φ should satisfy:

φ	\rightarrow	$reachable_{\mathcal{D}}$	Every φ -state is reachable
φ	\rightarrow	$\rho \diamond \varphi$	Every φ -state has a φ -successor
φ	\rightarrow	$\mu Y.(J \wedge \varphi \vee \rho \diamond Y)$	For every $J \in \mathcal{J}$, every φ -state has a path leading to a $J \wedge \varphi$ -state

This can be summarized as

$$\varphi \rightarrow \left(reachable_{\mathcal{D}} \wedge \rho \diamond \varphi \wedge \bigwedge_{J \in \mathcal{J}} \mu Y.(J \wedge \varphi \vee \rho \diamond Y) \right)$$

Since we are interested in the maximal set of feasible states, this set can be computed by:

$$\nu Z. \left(reachable_{\mathcal{D}} \wedge \rho \diamond Z \wedge \bigwedge_{J \in \mathcal{J}} \mu Y.(J \wedge Z \vee \rho \diamond Y) \right)$$

Temporal Testers

For every LTL formula φ , there exists an FDS $T[\varphi]$ called the **temporal tester** for φ . This tester has a distinguished boolean variable x , such that, in every σ , a computation of $T[\varphi]$ and every position $j \geq 0$, $x[s_j] = 1$ iff $(\sigma, j) \models \varphi$.

We can view $T[\varphi]$ as a (possibly non-deterministic) **transducer** which incrementally reads the values of the variables U and outputs in x the current value of φ over the infinite sequence.

Construction of Temporal Testers

A formula φ is called a **principally temporal formula** (PTF) if the main operator of p is temporal. With no serious loss of generality, we restrict our attention to **LTL** formulas which are PTF's.

A PTF is called a **basic temporal formula** if it contains no other PTF as a proper sub-formula. We start our construction by presenting temporal testers for the basic temporal formulas.

A Tester for $\bigcirc p$

The tester for the formula $\bigcirc p$ is given by:

$$T[\bigcirc p] : \begin{cases} V : \text{Vars}(p) \cup \{x\} \\ \Theta : 1 \\ \rho : x = p' \\ \mathcal{J} = \mathcal{C} : \emptyset \end{cases}$$

Claim 2.

$T[\bigcirc p]$ is a temporal tester for $\bigcirc p$.

Proof:

Let σ be a computation of $T[\bigcirc p]$. We will show that x matches $\bigcirc p$ in σ . Let $j \geq 0$ be any position. By the transition relation, $x = 1$ at position j iff $s_{j+1} \models p$ iff $(\sigma, j) \models \bigcirc p$.

Let σ be an infinite sequence such that x matches $\bigcirc p$ in σ . We will show that σ is a computation of $T[\bigcirc p]$. For any position $j \geq 0$, $x = 1$ at j iff $(\sigma, j) \models \bigcirc p$, iff $s_{j+1} \models p$. Thus, x satisfies $x = p'$ at every position j . ▀

A Tester for $p\mathcal{U}q$

The tester for the formula $p\mathcal{U}q$ is given by:

$$T[p\mathcal{U}q] : \begin{cases} V : \text{Vars}(p, q) \cup \{x\} \\ \Theta : 1 \\ \rho : x = q \vee (p \wedge x') \\ \mathcal{J} : q \vee \neg x \\ \mathcal{C} : \emptyset \end{cases}$$

Claim 3. $T[p\mathcal{U}q]$ is a temporal tester for $p\mathcal{U}q$.

Proof:

Based on the expansion formula

$$p\mathcal{U}q = q \vee (p \wedge \bigcirc(p\mathcal{U}q))$$

The role of the justice requirement $q \vee \neg x$ is to rule out a computation in which continuously $p = 1, q = 0$ but $x = 1$. Note that in such a case, the transition relation reduces to

$$x = x',$$

which has both $x = 0$ and $x = 1$ as possible solutions.

A Tester for $p\mathcal{W}q$

A supporting evidence for the significance of the justice requirements is provided by the tester for the formula $p\mathcal{W}q$:

$$T[p\mathcal{W}q] : \begin{cases} V : \text{Vars}(p, q) \cup \{x\} \\ \Theta : 1 \\ \rho : x = q \vee (p \wedge x') \\ \mathcal{J} : \neg p \vee x \\ \mathcal{C} : \emptyset \end{cases}$$

Note that the transition relation of $T[p\mathcal{W}q]$ is identical to that of $T[p\mathcal{U}q]$, and they only differ in their respective justice requirements.

The role of the justice requirement in $T[p\mathcal{W}q]$ is to eliminate the solution $x = 0$ over a computation in which $p = 1$ and $q = 0$ at all positions.

Testers for the Derived Operators

Based on the testers for \mathcal{U} and \mathcal{W} , we can construct testers for the derived operators \diamond and \square . They are given by

$$T[\diamond p] : \begin{cases} V : \text{Vars}(p) \cup \{x\} \\ \Theta : 1 \\ \rho : x = p \vee x' \\ \mathcal{J} : p \vee \neg x \\ \mathcal{C} : \emptyset \end{cases} \quad T[\square p] : \begin{cases} V : \text{Vars}(p) \cup \{x\} \\ \Theta : 1 \\ \rho : x = p \wedge x' \\ \mathcal{J} : \neg p \vee x \\ \mathcal{C} : \emptyset \end{cases}$$

A formula such as $\diamond p$ can be viewed as a “promise for an eventual p ”. The justice requirement $p \vee \neg x$ can be interpreted as suggesting:

Either fulfill all your promises or stop promising.

Note that once $x = 0$ in the tester $T[\diamond p]$, it remains 0 and requires $p = 0$ ever after.

Testers for the Basic Past Formulas

The following are testers for the basic past formulas $\ominus p$ and $p\mathcal{S}q$:

$$T[\ominus p] : \begin{cases} V : \text{Vars}(p) \cup \{x\} \\ \ominus : x = 0 \\ \rho : x' = p \\ \mathcal{J} : \emptyset \\ \mathcal{C} : \emptyset \end{cases} \quad T[p\mathcal{S}q] : \begin{cases} V : \text{Vars}(p, q) \cup \{x\} \\ \ominus : x = q \\ \rho : x' = q' \vee (p' \wedge x) \\ \mathcal{J} : \emptyset \\ \mathcal{C} : \emptyset \end{cases}$$

Note that testers for past formulas are not associated with any fairness requirements. On the other hand, they have a non-trivial initial conditions.

Testers for Compound Temporal Formulas

Up to now we only considered testers for basic formulas. The construction for non-basic formulas is based on the following reduction principle. Let $f(\varphi)$ be a temporal formula containing one or more occurrences of the basic formula φ . Then the temporal tester for $f(\varphi)$ can be constructed according to the following recipe:

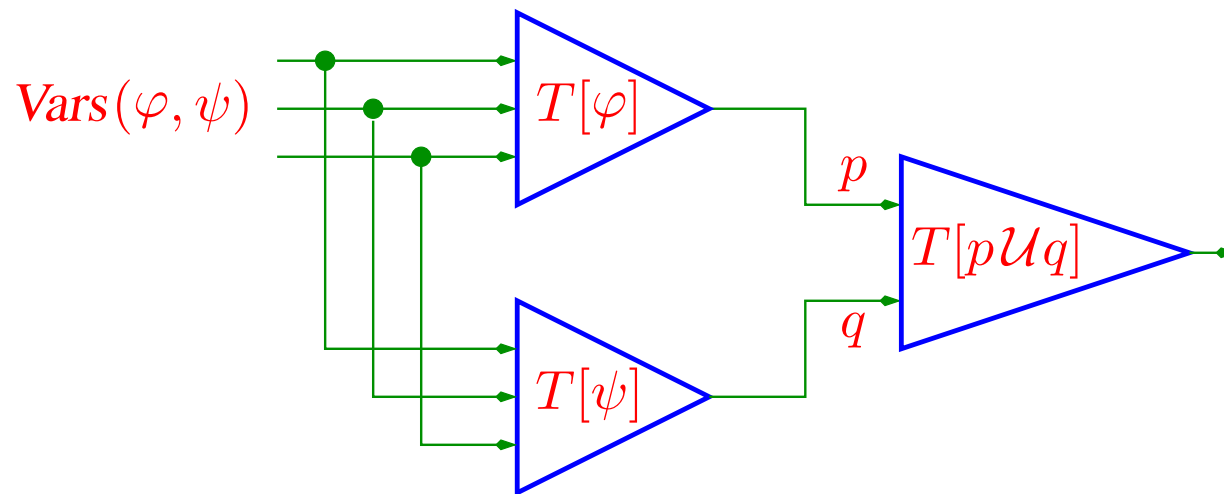
$$T[f(\varphi)] = T[f(x_\varphi)] \parallel T[\varphi]$$

where, x_φ is the boolean output variable of $T[\varphi]$, and $f(x_\varphi)$ is obtained from $f(\varphi)$ by replacing every instance of φ by x_φ .

Following this recipe the temporal tester for an arbitrary formula f can be decomposed into a synchronous parallel composition of smaller testers, one for each basic formula nested within f .

Testers as Circuits

Having viewed testers as **transducers**, we can view their composition as a **circuit interconnection**. For example, in the following diagram we show how a tester for the compound formula $\varphi \mathcal{U} \psi$ can be constructed by interconnecting the testers for φ , ψ , and the tester for the basic formula $p \mathcal{U} q$.



Model Checking General Temporal Formulas

To check whether $\mathcal{D} \models \varphi$, perform the following steps:

- Construct the tester $T[\varphi]$.
- Form the combined system $C = \mathcal{D} \parallel T[\varphi] \parallel \langle \Theta : \neg x\varphi, 1, \emptyset, \emptyset \rangle$, where $\langle \Theta : \neg x\varphi, 1, \emptyset, \emptyset \rangle$ is the trivial FDS which imposes the constraint that all initial states falsify $x\varphi$.
- Check whether C is feasible.
- Conclude $\mathcal{D} \models \varphi$ iff C is infeasible.

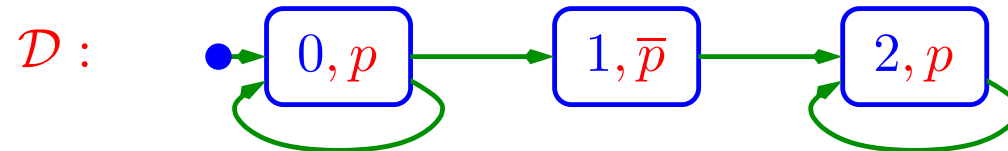
The Advantages of Using Temporal Testers

- **Testers** are easier and more natural to explain than either **tableaus** or **alternating automata**.
- Their construction is **compositional** and **modular**. It is sufficient to construct a tester for each temporal operator. Inclusion of **past** operators is straightforward.
- If one wants to extend the logic, it is only necessary to provide a tester for each new operator. Cases in point: the **PSL** operator $\langle r \rangle \varphi$ [ZP06], and the **MITL** timed operator $p \mathcal{U}_{[a,b]} q$ [MNP06].
- Testers enable verification of **LTL** (and **CTL***) properties which is **compositional** in the structure of the formula. Reduces the verification of $\mathcal{D} \models f(\varphi)$ into $(\mathcal{D} \parallel T[\varphi]) \models f(x_\varphi)$.

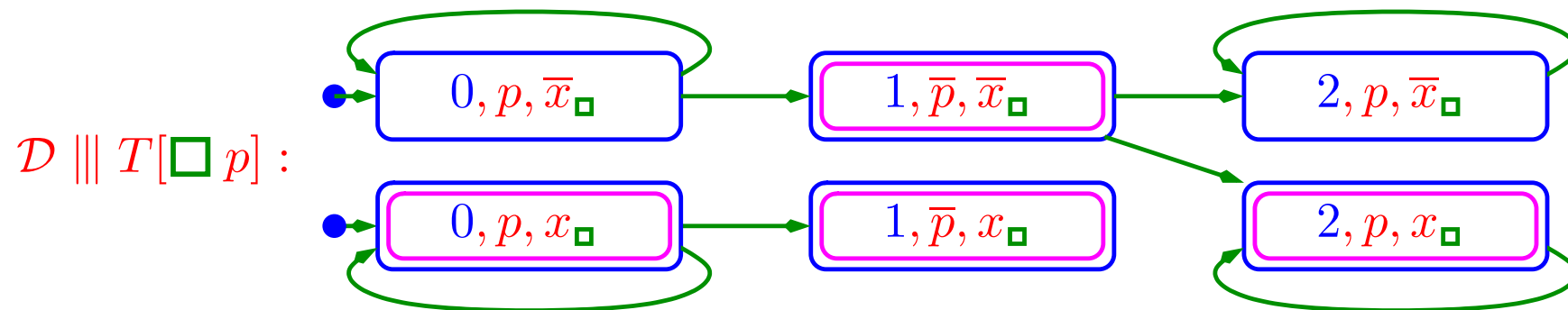
This provides an elegant solution to the long-standing problem of **deductive verification** of arbitrary **LTL** formulas.

Example: Compositional Verification

In order to verify $\diamond \square p$ over the system



it is sufficient to verify $\diamond x_{\square}$ over



with the justice requirement $\neg p \vee x_{\square}$.

A Tester for $\varphi : \langle (pq)^* \rangle b$

The following tester [PZ06] illustrates the constructions of a tester for the PSL operator $\langle r \rangle b$.

$$v \models \langle (pq)^* \rangle b \iff \exists j \geq 0 \text{ s.t. } v^{0..j} \models (pq)^*, v^j \models b$$

The associated right-linear grammar for the SERE $(pq)^*$ is given by:

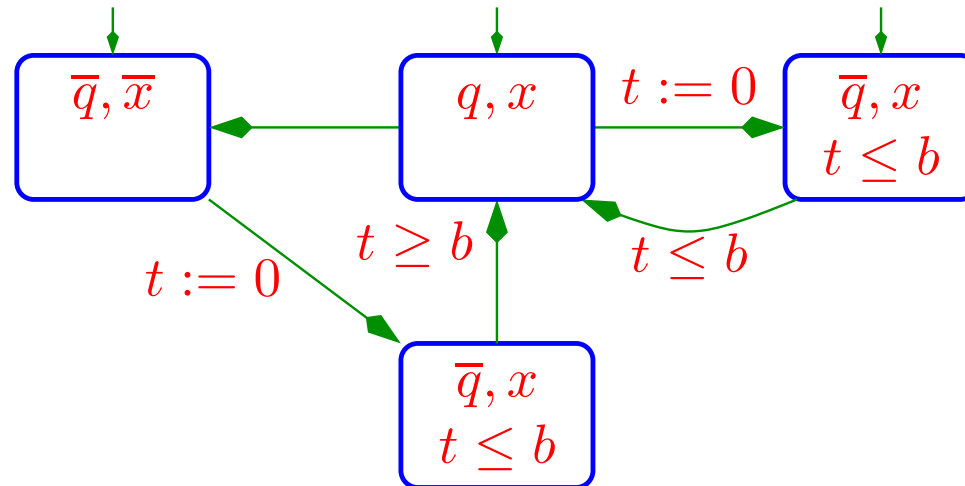
$$\begin{array}{l} V_1 \rightarrow pV_2 \\ V_2 \rightarrow q \mid qV_1 \end{array}$$

$$T[\varphi] : \left\{ \begin{array}{l} V : \text{Vars}(p, q, b) \cup \{x_\varphi, X_1, X_2, Y_1, Y_2\} \\ \Theta : 1 \\ \rho(V, V') : \left(\begin{array}{l} (X_1 = (p \wedge X'_2)) \quad \wedge \\ (X_2 = (q \wedge b) \vee (q \wedge X'_1)) \quad \wedge \\ (Y_1 \rightarrow (p \wedge Y'_2)) \quad \wedge \\ (Y_2 \rightarrow (q \wedge b) \vee (q \wedge Y'_1)) \quad \wedge \\ x_\varphi = X_1 \end{array} \right) \\ \mathcal{J} : \{\neg Y_1 \wedge \neg Y_2, \quad X_1 = Y_1 \wedge X_2 = Y_2\} \end{array} \right.$$

Note that $\langle V_1 \rangle b \iff p \wedge \bigcirc [\langle V_2 \rangle b]$ and $\langle V_2 \rangle b \iff (q \wedge b) \vee (q \wedge \bigcirc [\langle V_1 \rangle b])$.

A Tester for $\diamond_{[0,b]}q$

Following [MNP06], we present a (timed automaton) tester for the MITL formula $\diamond_{[0,b]}q$ for $b > 0$. This formula holds at time $\tau \geq 0$ if there is an occurrence of q at time $t \in [\tau, \tau + b]$.



For $0 < a < b$, we can express

$$\begin{aligned} \diamond_{[a,b]}q &= \diamond_a(\diamond_{[0,b-a]}q) \\ p\mathcal{U}_{[a,b]}q &= \square_{[0,a]}(p\mathcal{U}q) \wedge \diamond_{[a,b]}q \\ p\mathcal{U}_{[0,b]}q &= p\mathcal{U}q \wedge \diamond_{[0,b]}q \end{aligned}$$

where the formula $\diamond_a q$ holds at time $\tau \geq 0$ if there is an occurrence of q at time $\tau + a$.