Next, we consider methods for model checking general LTL formulas. Let $D$ be an FDS and $\phi$ be an LTL formula. Assume we wish to check whether $D \models \phi$. Then, we can consider the following steps:

1. **Construct a temporal acceptor $A(\phi)$ for a general LTL formula $\phi$.**

2. **Form the parallel composition $D \parallel A(\phi)$ of $D$ and $A(\phi)$.** This is an FDS whose computations are all the sequences falsifying $\phi$. If $D \parallel A(\phi)$ is feasible, then $D \models \phi$. Otherwise, $D \not\models \phi$.

Let $D$ be an FDS and $\phi$ be an LTL formula. Assume we wish to check whether $D \models \phi$. Then, we consider the following steps:

1. **Construct a temporal acceptor $A(\phi)$ for a general LTL formula $\phi$.**

2. **Form the parallel composition $D \parallel A(\phi)$ of $D$ and $A(\phi)$.** This is an FDS whose computations are all the sequences falsifying $\phi$. If $D \parallel A(\phi)$ is feasible, then $D \models \phi$. Otherwise, $D \not\models \phi$. 

### Analysis of Reactive Systems, NYU, Fall 2009
Lecture 5: Model Checking General LTL

Asynchronous parallel composition

That is, the FDS corresponding to the program $P_1 \parallel P_2$ is equivalent to the asynchronous parallel composition of the FDS's corresponding to $P_1$ and $P_2$.

Claim 7.

Asynchronous parallel composition represents the interleaving-based concurrency which is the assumed concurrency in shared-variables models.

The predicate $\text{pres}(U)$ stands for the assertion that all the variables in $U$ are preserved by the transition.

The asynchronous parallel composition of systems $\mathcal{D}_1$ and $\mathcal{D}_2$, denoted by $\mathcal{D}_1 \parallel \mathcal{D}_2$, is given by $\mathcal{D}_1 \parallel \mathcal{D}_2 = \mathcal{D}_1 \parallel \mathcal{D}_2 = \mathcal{D}_1 \parallel \mathcal{D}_2$

\[
\begin{align*}
\mathcal{C}_1 \cup \mathcal{C}_2 &= \mathcal{C} \\
\mathcal{L}_1 \cup \mathcal{L}_2 &= \mathcal{L} \\
((\mathcal{L}_1 - \mathcal{L}_2) \lor \mathcal{d}) \land (\mathcal{L}_1 - \mathcal{L}_2) &= \mathcal{d} \\
\mathcal{C} \lor \mathcal{D} &= \mathcal{C} \lor \mathcal{D} \\
\mathcal{L} \lor \mathcal{A} &= \mathcal{L} \lor \mathcal{A} \\
\mathcal{C}_1 \cup \mathcal{L}_2 &= \mathcal{C}_1 \cup \mathcal{L}_2 \\
\mathcal{L}_1 \cup \mathcal{A} &= \mathcal{L}_1 \cup \mathcal{A} \\
\end{align*}
\]

where $\langle \mathcal{D}_1, \mathcal{D}_2, \mathcal{L}, \mathcal{d}, \mathcal{C}, \mathcal{A} \rangle = \mathcal{D}_1 \parallel \mathcal{D}_2$.

Operations on FDS's: Asynchronous Parallel Composition
The synchronous parallel composition of systems $D_1$ and $D_2$, denoted $\bigoplus D_1 D_2$, is given by the FDS $\langle V; J; C_1 \cup C_2 \rangle$. The sequence $\bigoplus o \uparrow D_1 D_2$ is a computation of the combined $\bigoplus D_1 D_2$. Synchronous parallel composition can be used for model checking of LTL formulas.

Claim 8. The sequence $o \uparrow D_1 D_2$ is a computation of the combined $\bigoplus D_1 D_2$. Here, $o \uparrow D_i$ is a computation of $D_i$ and $o \uparrow V_i$ is a computation of $D_i$. Synchronous parallel composition can be used for model checking of LTL formulas. Here we use it for model checking of LTL formulas.

\begin{align*}
\mathcal{F} & \cup \mathcal{F} = \mathcal{F} \\
\mathcal{L} & \cup \mathcal{L} = \mathcal{L} \\
\mathcal{d} & \cup \mathcal{d} = \mathcal{d} \\
\Theta \cup \Theta & = \Theta \\
\Lambda \cup \Lambda & = \Lambda
\end{align*}

is given by the FDS $\langle \mathcal{F}, \mathcal{L}, \mathcal{d}, \Theta, \Lambda \rangle = \mathcal{D}$, where

The synchronous parallel composition of systems $D_1$ and $D_2$, denoted $\bigoplus D_1 D_2$, is a computation of the combined $\bigoplus D_1 D_2$. Synchronous parallel composition can be used for model checking of LTL formulas.
value of $\phi$ over the infinite sequence. The building blocks from which we construct acceptors are temporal testers. Let $\phi$ be a temporal formula over vocabulary $\bigwedge$, and let $x$ be a boolean variable. We can view $(\phi)I$ as a (possibly non-deterministic) transducer which incrementally reads the values of the variables $\bigwedge$, and outputs in the current state. Let $x \notin \bigwedge$ be a boolean variable disjoint from $\bigwedge$.

In the following, let $s_0, s_1, \ldots$ be an infinite sequence of states over $\bigwedge$. We say that $x$ matches $(\phi)I$ if, for every position $j$, the value of $x$ at position $j$ is true iff $(\phi)I$.

We can view $T(\phi)$ as a (possibly non-deterministic) transducer which incrementally reads the values of the variables $\bigwedge$, and outputs in the current state. Let $x \notin \bigwedge$ be a boolean variable disjoint from $\bigwedge$.

We say that $x$ matches $(\phi)I$ if, for every position $j$, the value of $x$ at position $j$ is true iff $(\phi)I$.

The infinite sequence $\sigma$ is a computation of $T(\phi)$ iff $x$ matches $\sigma$ in $\phi$.

A temporal tester for $\phi$ is an FDS $T$ over $\bigwedge$, satisfying the requirement:

\[ \phi \models \text{true} \iff (T, \sigma) \]

In the following, let $s_0, s_1, \ldots$ be an infinite sequence of states over $\bigwedge$. We say that $x$ matches $\phi$ in $\bigwedge$ if, for every position $j$, the value of $x$ at position $j$ is true iff $T(\phi)$.

We say that $x$ matches $\phi$ in $\bigwedge$ if, for every position $j$, the value of $x$ at position $j$ is true iff $T(\phi)$.

The infinite sequence $\sigma$ is a computation of $T(\phi)$ iff $x$ matches $\sigma$ in $\phi$.

A temporal tester for $\phi$ is an FDS $T$ over $\bigwedge$, satisfying the requirement:

\[ \phi \models \text{true} \iff (T, \sigma) \]

In the following, let $s_0, s_1, \ldots$ be an infinite sequence of states over $\bigwedge$. We say that $x$ matches $\phi$ in $\bigwedge$ if, for every position $j$, the value of $x$ at position $j$ is true iff $T(\phi)$.

The infinite sequence $\sigma$ is a computation of $T(\phi)$ iff $x$ matches $\sigma$ in $\phi$.

A temporal tester for $\phi$ is an FDS $T$ over $\bigwedge$, satisfying the requirement:

\[ \phi \models \text{true} \iff (T, \sigma) \]
A formula is called a principally temporal formula (PTF) if the main operator of A. Pnueli

A formula is called a principal temporal formula (PTF) if it contains no other PTFs.

Construction of Temporal Testers

We start our construction by presenting temporal testers for the basic temporal formulas.
The tester for the formula \( p \) is given by:

A Tester for \( \diamond p \)
The tester for the formula $b \eta d$ is given by:

\begin{align*}
\begin{array}{c}
\emptyset : \emptyset \\
x \land \neg b : \cal{F} \\
(x \lor d) \land b : \cal{I} \\
\{x\} \cap (b \cdot d) : \vars \\
\end{array}
\end{align*}

A Tester for $p \lor q$

Proof:

Claim 10.

The tester requirement demands that $\omega$ contains infinitely many positions at which both $x$ and $\neg b$ hold over all positions beyond $\omega$, which is impossible since it violates the semantic definition of the operator. The other case in which $b \eta d = (\omega, \omega)$ for all $x \lor d \lor b \leadsto \omega$ is obviously satisfied. If we locate a stopping run, we have either $b \models \omega$ or we have either locate a stopping run, or we have

\begin{align*}
\begin{array}{c}
\emptyset : \emptyset \\
x \land \neg b : \cal{F} \\
(x \lor d) \land b : \cal{I} \\
\{x\} \cap (b \cdot d) : \vars \\
\end{array}
\end{align*}

The tester is a temporal tester for $p \lor q$.
Next we consider the case that $\phi$ contains only finitely many $b$-positions. In that case, $x$ must be false at all positions beyond $c$ beyond $c$. Consequently, $x$ is false. In this case, there must exist a cut-off position $c$ such that no position beyond $c$ satisfies $\phi$. We will show that $\phi$ is a computation of $T(p \cup q)$.

From the semantic definition, it is enough at $x^- \land b = x$ holds at $(x \lor d) \land b = x$ and $d = \models s$. Then by induction if necessary, the transition relation $T(p \cup q)$ implies that $x$ matches at $b \eta d = | (\emptyset, \emptyset)$.

In the other direction, let $\phi$ be an infinite sequence such that $x$ matches at all positions. Let $\phi$ be a computation of $T(p \cup q)$.

Next we consider the case that $\phi$ contains only finitely many $q$-positions. In that case, $p \cup q$ must be false at all positions beyond $c$. Consequently, $x$ is false. In this case, there must exist a cut-off position $c$ such that no position beyond $c$ satisfies $\phi$. We will show that $\phi$ is a computation of $T(p \cup q)$.

From the semantic definition, it is enough at $x^- \land b = x$ holds at $(x \lor d) \land b = x$ and $d = \models s$. Then by induction if necessary, the transition relation $T(p \cup q)$ implies that $x$ matches at $b \eta d = | (\emptyset, \emptyset)$.

In the other direction, let $\phi$ be an infinite sequence such that $x$ matches at all positions. Let $\phi$ be a computation of $T(p \cup q)$.

Next we consider the case that $\phi$ contains only finitely many $q$-positions. In that case, $p \cup q$ must be false at all positions beyond $c$. Consequently, $x$ is false. In this case, there must exist a cut-off position $c$ such that no position beyond $c$ satisfies $\phi$. We will show that $\phi$ is a computation of $T(p \cup q)$.

From the semantic definition, it is enough at $x^- \land b = x$ holds at $(x \lor d) \land b = x$ and $d = \models s$. Then by induction if necessary, the transition relation $T(p \cup q)$ implies that $x$ matches at $b \eta d = | (\emptyset, \emptyset)$.

In the other direction, let $\phi$ be an infinite sequence such that $x$ matches at all positions. Let $\phi$ be a computation of $T(p \cup q)$.
Thus, the role of the justice requirement is to select among several solutions to the transition relation equation. This is also the only solution which satisfies the justice requirement. The transition relation equation is true at all positions. Only one in which is identically false and the other in which is identically true at all positions. This equation has two possible solutions, one in which is identically false and the other in which is identically true at all positions. Thus, the role of the justice requirement is to select among several solutions to the transition relation equation.

We wish to show that the justice requirement is essential for the correctness of the construction. Consider a state sequence. Consider a state sequence. In this case, the transition relation reduces to the equation.

For example, let 

\[
\begin{align*}
0 = x \\
x = 0
\end{align*}
\]

This is also the only solution which satisfies the justice requirement. The transition relation equation is true at all positions. Only one in which is identically false and the other in which is identically true at all positions. This equation has two possible solutions, one in which is identically false and the other in which is identically true at all positions. Thus, the role of the justice requirement is to select among several solutions to the transition relation equation.

Consider the temporal tester for

**Why Do We Need the Justice Requirement**

A. Pnueli

---

**Analysis of Reactive Systems**, NYU, Fall, 2009
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 $b \mathcal{M} d$.
After

Note that once 0 = d in the tester 0 = x in the tester.

Either fulfill all your promises or stop promising.

A formula such as p can be viewed as a "promise" for an eventual. The justice requirement p can be interpreted as suggesting:

p can be viewed as a "promise for an eventual."

A form such as p can be viewed as a "promise for an eventual."

They are given by □ and ◻ operators and □ ◻ operators. They are given by □ and ◻ operators. They are given by □ and ◻ operators.

Based on the testers for and , we can construct testers for the derived testers for the derived.
The following are testers for the basic past formulas:

\[
\begin{align*}
&\emptyset : C \\
&\emptyset : L \\
&\{x\} \cap (b,d)_{\text{Vars}} : \Lambda
\end{align*}
\]

\[
\begin{align*}
&\emptyset : C \\
&\emptyset : L \\
&\{x\} \cap (d)_{\text{Vars}} : \Lambda
\end{align*}
\]

\[
\begin{align*}
&b \leq d \\
&b = x : \Theta \\
&0 = x : \Theta
\end{align*}
\]

Note that testers for past formulas are not associated with any fairness requirements. On the other hand, they have a non-trivial initial condition.
formulas, and denote it by $\text{(f)}(\text{redux})$. When all possible substitution/composition steps are performed, we are left with a (non-temporal) assertion. We refer to this assertion as the \text{redux} of the original formula. Each basic formula nested within $f$ can be decomposed into a synchronous parallel composition of smaller testers, one for each basic formula contained within it. Following this recipe the temporal tester for an arbitrary formula $f$ can be constructed according to the following recipe: then the temporal tester for a temporal formula containing one or more occurrences of the basic formula $\phi$ can be constructed according to the following recipe.

**Testers for Compounded Temporal Formulas**
Having viewed testers as transducers, we can view their composition as a circuit. For example, in the following diagram we show how the tester for the compound formula $\phi \land \psi$ can be constructed by interconnecting the testers for $\phi$ and $\psi$. 

**Testers as Circuits**
For a formula containing one or more occurrences of the basic formula $\phi$:

$$\phi \land ((\phi x)f) = ((\phi)f)$$

The construction of an acceptor is defined recursively as follows:

- For an assertion $p$:
  $$A(p) = \begin{cases} Vars(p) : 1 \land J = C \end{cases}$$

- For a formula $f$ containing one or more occurrences of the basic formula $\phi$:
  $$A(f(x')) = A(f(x)) \cup T(f(x'))$$

Thus, unlike testers, an acceptor only accepts at position 0. Thus, testers are the essential building blocks for the construction of an acceptor.

For an acceptor $d$:

- $\emptyset : C = \emptyset$
- $I : d$
- $d : \emptyset$
- $(d)Vars : \lambda$

The sequence $\omega$ satisfies $\phi$ if $\omega \models (d)(\phi)$ such that acceptor for an LTL formula $(\phi)(\emptyset(\omega)(\phi))$ is an FDS $A(d)(\phi)$ over variables $\phi$. An acceptor for an acceptor.

Acceptors
An acceptor for $p$ following is a tester for the formula $p$ which is obtained by computing the parallel composition $A' (d \Box \Diamond 
abla) 
abla$

Note that the redux of $d \Box \Diamond 
abla$ is $\Diamond x \rightarrow d \Box \Diamond 
abla$

\[
\begin{align*}
\emptyset : \mathcal{C} \\
\{ \Diamond x \rightarrow \Box x \land \Diamond x \land d \rightarrow \} : \mathcal{L} \\
(\Diamond x \land \Box x = \Diamond x) \lor (\Box x \lor d = \Box x) : d \\
\Diamond x \rightarrow : \Theta \\
\{ \Box x \land \Diamond x \} \cap (d) \text{Vars} : \Lambda
\end{align*}
\]

Example: An acceptor for $\Diamond x \rightarrow d \Box \Diamond 
abla$
Model Checking General Temporal Formulas

To check whether $\varphi \models \phi$, perform the following steps:

1. Construct the acceptor $A(\neg \varphi)$.
2. Form the combined system $C = D \parallel A(\neg \varphi)$.
3. Check whether $C$ is feasible.
4. Conclude $\varphi \models \phi$ iff $C$ is infeasible.

A. Pnueli

Analysis of Reactive Systems, NYU, Fall, 2009
Consider the following system:

- $D$: 0; $p_1$; $p_2$; $p$

For which we wish to verify the property $\square p$.

Example

Analysis of Reactive Systems, NYU, Fall, 2009
Composing the system with the acceptor \( \text{A} \) (\( d \)), we obtain:

\[ \text{Example: Continued} \]
Claim 11.

For an FDS $D$ and temporal formula $\phi$, $\phi \models \square (\neg \phi)$ is infeasible.

**Proof:**

The proof is based on the observation that every computation of the combined system $C$ is a computation of $D$ which satisfies the negation of $\phi$. Therefore, the existence of such a computation shows that not all computations of $D$ satisfy $\phi$, and therefore, $\phi$ is not valid over $D$. Therefore, $\phi$ is not valid over $D$. 

A.Pnueli