Reactive Systems

Analysis

incrementally reads the values of the variables and outputs in state. We can view $\Delta$ as a (possibly non-deterministic) transducer which be extended into a computation of $\phi, \Delta$. By interpreting $\Delta$ at position $x$ of $\phi$, the value of $\Delta$ at position $x$ of $\phi$ can be extended into a computation of $\phi, \Delta$. Here, by restricting each of the states denoted by $\phi, \Delta$, we define the sequence obtained from a computation of $\phi, \Delta$. The initial sequence $\phi, \Delta$ is a computation of $\phi, \Delta$. We say that $\phi$ satisfies the requirement $\phi, \Delta$. The initial sequence $\phi, \Delta$ is a computation of $\phi, \Delta$. In the following, let $A, B, C, D$ be an initial sequence of states over $\{x\} \cap \{y\} \cap \{z\}$. The initial sequence $\phi, \Delta$ is a computation of $\phi, \Delta$. Let $A, B, C, D$ be a transducer formula over vocabulary $\{x\} \cup \{y\} \cup \{z\}$, and let $A$ be a boolean variable.

Temporal Tests

Synchronous Parallel Composition

temporal tests for is an EDS $\Delta$ over $\{x\} \cup \{y\} \cup \{z\}$, satisfying the requirement:

$$\phi = \{x\} \cup \{y\} \cup \{z\}$$

The initial sequence $\phi, \Delta$ is a computation of $\phi, \Delta$. We say that $\phi$ satisfies the requirement $\phi, \Delta$. The initial sequence $\phi, \Delta$ is a computation of $\phi, \Delta$. In the following, let $A, B, C, D$ be an initial sequence of states over $\{x\} \cap \{y\} \cap \{z\}$. The initial sequence $\phi, \Delta$ is a computation of $\phi, \Delta$. Let $A, B, C, D$ be a transducer formula over vocabulary $\{x\} \cup \{y\} \cup \{z\}$, and let $A$ be a boolean variable.

Synchronous Parallel Composition

Lecture 5: Models Composition LTL

The synchronous parallel composition of FDS's corresponding to $\phi, \Delta$ and $\psi, \Delta$ is the EDS corresponding to the program $\phi, \Delta \parallel \psi, \Delta$, which is the least existential compatible with the interleaving-based concurrent model, being preserved by the transition.

The predicate $\text{pres} \phi, \Delta$ stands for the assertion $\Delta, \phi \models \phi$. We say that $\phi, \Delta$ is feasible if the following holds:

$$\text{pres} \phi, \Delta \models \phi \models \Delta \models \phi$$

Claim 7: The synchronous parallel composition of $\phi, \Delta$ and $\psi, \Delta$ is a computation of $\phi, \Delta$. Next, we consider methods for model checking general LTL formulas.

Model Checking General LTL Formulas
either $b$ is false or $x$ is true.

In this case, $x = \top$ and $b$ hold at all positions beyond $\ell$. In that case, there must exist a cut-off position $\ell'$ such that no position beyond $\ell'$ satisfies $b$.

In that case, we consider the case that $x$ satisfies the justice requirement above $b$. The other case in which $Up \neq 0$ is handled similarly. The semantic definition of the operator $\text{test}$ is given by:

$$\text{test}(\phi, \theta) = \begin{cases} 0 : \phi = \top & \\
\text{VAR} : \theta & \end{cases}$$

The test for the formula $\phi$ is given by:

$$\text{test}(\phi) = \begin{cases} 0 : \phi = \top & \\
\text{VAR} : \phi & \end{cases}$$

We start our construction by presenting temporal testers for the basic temporal operators.

**Lemma 10.**

**Claim 9.**

If $d = x \land \neg b$, then $d \models (f \lor g)$.

**Proof:**

If $d = x \land \neg b$, then $d \models (f \lor g)$.

Let $d$ be a computation of $L(d)$. We will show that $d \models (f \lor g)$.

**Proof:**

If $d = x \land \neg b$, then $d \models (f \lor g)$.

Let $d$ be a computation of $L(d)$. We will show that $d \models (f \lor g)$.

**Proof:**

If $d = x \land \neg b$, then $d \models (f \lor g)$.

Let $d$ be a computation of $L(d)$. We will show that $d \models (f \lor g)$.
A Tester for \( \mu^W \)

We wish to show that the justice requirement \( \forall p. \forall q. \forall x. x = x' \) is essential for the correctness of the construction. Consider a state sequence \( \sigma \ldots s \ldots s \ldots \) in which \( q \) is identically false and \( p \) is identically true at all positions. In this case, the transition relation reduces to the equation:

\[ x = x' \]

This equation has two possible solutions, one in which \( x \) is identically false and the other in which \( x \) is identically true at all positions. Only \( x = 0 \) matches \( \mu^W \).

Thus, the role of the justice requirement is to select among several solutions to the transition relation equation, a unique one which matches the basic temporal formula at all positions.

We have thus established the following claim:

The following are testers for the basic past formulas \( p \) and \( \mu_S \):

\[
\begin{align*}
T(p): & \begin{cases} 
\Theta: & x = 0 \\
\gamma: & 0 = 0 \\
\zeta: & 0 
\end{cases} \\
T(\mu_S): & \begin{cases} 
\Theta: & x = 0 \\
\gamma: & (p \land q) \land x = 0 \\
\zeta: & 0 
\end{cases}
\end{align*}
\]

The role of the justice requirement in \( T(\mu^W) \) is to eliminate the solution \( x = 0 \) over a computation in which \( p = 1 \) and \( q = 0 \) at all positions.

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

A formula such as \( \diamond p \) can be viewed as a ‘promise for an eventual \( p \)’. The justice requirement \( q \land x = 0 \) can be interpreted as suggesting:

**Either fulfill 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.

A-penel
Testers for Compound Temporal Formulas

Having viewed testers as transducers, we can view their composition as a circuit, with the compound formula as the output. We then decompose into a synchronous parallel composition of subformula testers, one for each subformula. The following recursion allows the construction of a temporal tester for an arbitrary formula. A tester for a compound formula can be constructed according to the following rules:

1. For a non-temporal formula, we refer to this assertion as the reduced formula of the original formula. We refer to this subformula as the reduced subformula of the original subformula. When all possible substitution/composition steps are performed, we are left with a formula.
2. For each subformula, we define a subformula tester for the subformula, where the subformula is obtained from the original subformula by replacing every instance of the atom in the subformula tester for the subformula with the atom in the original subformula.

Example: An acceptor for \( p \):

\[
\begin{align*}
A(p) : & \begin{cases}
\forall x : \mathcal{L} \\
\forall x : \mathcal{L} \land \phi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \land \chi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \land \chi(x) \land \\
\end{cases}
\end{align*}
\]

Note that the reduced formula of the tester for \( p \) is \( \phi(p) \) by reflexivity.

Thus, unlike testers, an acceptor only accepts at position 0. The construction of an acceptor is defined recursively as follows:

- For an assertion \( p \), \( A(p) : \forall x : \mathcal{L} \).
- For a formula \( f(p) \) containing one or more occurrences of the basic formula \( p \), \( A(f(p)) = A(f(x)) \).
- For an assertion \( d \).

Acceptors

The construction of an acceptor is defined recursively as follows:

- For an assertion \( d \).

The subformulas of the reduced formula of a compound formula are in \( \mathcal{L} \), and each subformula is a reduced subformula of a subformula tester, one for each subformula. We can view their composition as a circuit, with the compound formula as the output. We then decompose into a synchronous parallel composition of subformula testers, one for each subformula. The following recursion allows the construction of a temporal tester for an arbitrary formula. A tester for a compound formula can be constructed according to the following rules:

1. For a non-temporal formula, we refer to this assertion as the reduced formula of the original formula. We refer to this subformula as the reduced subformula of the original subformula. When all possible substitution/composition steps are performed, we are left with a formula.
2. For each subformula, we define a subformula tester for the subformula, where the subformula is obtained from the original subformula by replacing every instance of the atom in the subformula tester for the subformula with the atom in the original subformula.

Example: An acceptor for \( p \):

\[
\begin{align*}
A(p) : & \begin{cases}
\forall x : \mathcal{L} \\
\forall x : \mathcal{L} \land \phi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \land \chi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \land \chi(x) \land \\
\end{cases}
\end{align*}
\]

Note that the reduced formula of the tester for \( p \) is \( \phi(p) \) by reflexivity.

Thus, unlike testers, an acceptor only accepts at position 0. The construction of an acceptor is defined recursively as follows:

- For an assertion \( d \).

Acceptors

The construction of an acceptor is defined recursively as follows:

- For an assertion \( d \).

The subformulas of the reduced formula of a compound formula are in \( \mathcal{L} \), and each subformula is a reduced subformula of a subformula tester, one for each subformula. We can view their composition as a circuit, with the compound formula as the output. We then decompose into a synchronous parallel composition of subformula testers, one for each subformula. The following recursion allows the construction of a temporal tester for an arbitrary formula. A tester for a compound formula can be constructed according to the following rules:

1. For a non-temporal formula, we refer to this assertion as the reduced formula of the original formula. We refer to this subformula as the reduced subformula of the original subformula. When all possible substitution/composition steps are performed, we are left with a formula.
2. For each subformula, we define a subformula tester for the subformula, where the subformula is obtained from the original subformula by replacing every instance of the atom in the subformula tester for the subformula with the atom in the original subformula.

Example: An acceptor for \( p \):

\[
\begin{align*}
A(p) : & \begin{cases}
\forall x : \mathcal{L} \\
\forall x : \mathcal{L} \land \phi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \land \chi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \land \chi(x) \land \\
\end{cases}
\end{align*}
\]

Note that the reduced formula of the tester for \( p \) is \( \phi(p) \) by reflexivity.

Thus, unlike testers, an acceptor only accepts at position 0. The construction of an acceptor is defined recursively as follows:

- For an assertion \( d \).

Acceptors

The construction of an acceptor is defined recursively as follows:

- For an assertion \( d \).

The subformulas of the reduced formula of a compound formula are in \( \mathcal{L} \), and each subformula is a reduced subformula of a subformula tester, one for each subformula. We can view their composition as a circuit, with the compound formula as the output. We then decompose into a synchronous parallel composition of subformula testers, one for each subformula. The following recursion allows the construction of a temporal tester for an arbitrary formula. A tester for a compound formula can be constructed according to the following rules:

1. For a non-temporal formula, we refer to this assertion as the reduced formula of the original formula. We refer to this subformula as the reduced subformula of the original subformula. When all possible substitution/composition steps are performed, we are left with a formula.
2. For each subformula, we define a subformula tester for the subformula, where the subformula is obtained from the original subformula by replacing every instance of the atom in the subformula tester for the subformula with the atom in the original subformula.

Example: An acceptor for \( p \):

\[
\begin{align*}
A(p) : & \begin{cases}
\forall x : \mathcal{L} \\
\forall x : \mathcal{L} \land \phi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \land \chi(x) \\
\forall x : \mathcal{L} \land \phi(x) \land \psi(x) \land \chi(x) \land \\
\end{cases}
\end{align*}
\]

Note that the reduced formula of the tester for \( p \) is \( \phi(p) \) by reflexivity.

Thus, unlike testers, an acceptor only accepts at position 0. The construction of an acceptor is defined recursively as follows:

- For an assertion \( d \).

Acceptors

The construction of an acceptor is defined recursively as follows:

- For an assertion \( d \).

The subformulas of the reduced formula of a compound formula are in \( \mathcal{L} \), and each subformula is a reduced subformula of a subformula tester, one for each subformula. We can view their composition as a circuit, with the compound formula as the output. We then decompose into a synchronous parallel composition of subformula testers, one for each subformula. The following recursion allows the construction of a temporal tester for an arbitrary formula. A tester for a compound formula can be constructed according to the following rules:

1. For a non-temporal formula, we refer to this assertion as the reduced formula of the original formula. We refer to this subformula as the reduced subformula of the original subformula. When all possible substitution/composition steps are performed, we are left with a formula.
2. For each subformula, we define a subformula tester for the subformula, where the subformula is obtained from the original subformula by replacing every instance of the atom in the subformula tester for the subformula with the atom in the original subformula.
A.Pnueli

Example

Consider the following system:

\[ D: \quad p_1; p_2; p \]

For which we wish to verify the property \( p \).

Consider the following system:

\[ D' \]

To check whether \( D' \) is reachable.

From the combined system \( C = D' \sim p \).

Construct the acceptor \( A \).

Check whether \( C \) is feasible.

If \( C \) is feasible, then \( D' \) is not reachable.

Example: Continued

Correctness of the Algorithms

Claim 1.

For an FDS \( D \) and temporal formula \( \phi \), \( D \models \phi \iff \forall D' \models \phi \).

Proof: The proof is based on the observation that every computation of the combined system is a computation of \( D \) which satisfies the negation of \( \phi \). Therefore, the existence of such a computation shows that \( D \) does not satisfy \( \phi \).

Theorem: For an FDS \( D \) and temporal formula \( \phi \), \( D \models \phi \iff C = D \sim A(\phi) \) is feasible.

Proof: The proof is based on the observation that every computation of the combined system 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 \( D \).