In this course, we concentrate on the study of deductive techniques.

A deductive methodology based on theorem-proving methods: Can accommodate infinite-state systems, but requires user interaction.

- Algorithmic verification methods for exploring verification of finite-state systems: Proof techniques for verifying that an implementation satisfies a specification.
- A specification language for specifying systems and their properties. We use a simple programming language (both synchronous and asynchronous). We use SPT, a simple programming language.
- A specification language for specifying systems and their properties. We use linear temporal logic (LTL).
- A specification language for specifying systems and their properties. We use linear temporal logic (LTL).
- A computational model providing an abstract syntactic base for all reactive systems. We use fair discrete systems (FDS).
- A computational model providing an abstract syntactic base for all reactive systems. We use fair discrete systems (FDS).
- A computational model providing an abstract syntactic base for all reactive systems. We use fair discrete systems (FDS).
- A computational model providing an abstract syntactic base for all reactive systems. We use fair discrete systems (FDS).

A Framework for Reactive Systems Verification

Example: $x + 1 = \bar{x}$

The program which computes $x + 1 = \bar{x}$ can be specified by the requirement $y = x^2$.

Such programs must be specified and verified in terms of their behaviors.

- Can be viewed as a green cactus (⊆).
- Can be viewed as a green cactus (⊆).

Example: Air traffic control systems, Programs controlling mechanical devices such as a train, a plane, or ongoing processes such as a nuclear reactor.

Programs whose role is to maintain an ongoing interaction with their environments.

Reactive Programs

Course grades will be determined based on assignments and a term project.


- Temporal verification of reactive systems: Safely by Z. Manna and A. Pnueli.

Textbooks:

http://www.cs.nyu.edu/courses/fall07/G22-3033-02/index.htm

Available at

Copies of presentations and lecture notes will be Wednesdays, 5:00-6:50 PM

Amir Pnueli

Deductive Verification of Reactive Systems

A Framework for Reactive Systems Verification

Can be modeled as a black box.

Computation Programs: Run in order to produce a final result on termination.

There are two classes of programs:

Classicalization of Programs
The Grand Vision

Formal verification will eventually prevail as the prime method for ensuring the correctness of systems.

What is Difficult in Deductive Verification?

Among the recognized difficulties in the application of deductive verification, we can count:

- Verifying general LTL properties.
- Verifying progress under compassion.
- Verifying invariance properties.
- The ∀∃ theorem prover.
- The PVS theorem prover.
- The PVS theorem prover.
- A simple programming language (SPS) and its translation into FDS.
- The computational model of Fair Discrete Systems (FDS).

Course Outline

- Verifying general LTL properties.
- Verifying progress under compassion.
- Verifying invariance properties.
- The ∀∃ theorem prover.
- The PVS theorem prover.
- A simple programming language (SPS) and its translation into FDS.
- The computational model of Fair Discrete Systems (FDS).

We will consider the following topics:

- Fair Discrete Systems
- A simple programming language (SPS) and its translation into FDS.
- The computational model of Fair Discrete Systems (FDS).
- The ∀∃ theorem prover.
- The PVS theorem prover.
- A simple programming language (SPS) and its translation into FDS.
- The computational model of Fair Discrete Systems (FDS).

The Grand Vision

Formal verification will eventually prevail as the prime method for ensuring the correctness of systems.

What is Difficult in Deductive Verification?

Among the recognized difficulties in the application of deductive verification, we can count:

- Verifying general LTL properties.
- Verifying progress under compassion.
- Verifying invariance properties.
- The ∀∃ theorem prover.
- The PVS theorem prover.
- A simple programming language (SPS) and its translation into FDS.
- The computational model of Fair Discrete Systems (FDS).

Course Outline

- Verifying general LTL properties.
- Verifying progress under compassion.
- Verifying invariance properties.
- The ∀∃ theorem prover.
- The PVS theorem prover.
- A simple programming language (SPS) and its translation into FDS.
- The computational model of Fair Discrete Systems (FDS).

We will consider the following topics:

- Fair Discrete Systems
- A simple programming language (SPS) and its translation into FDS.
- The computational model of Fair Discrete Systems (FDS).
- The ∀∃ theorem prover.
- The PVS theorem prover.
- A simple programming language (SPS) and its translation into FDS.
- The computational model of Fair Discrete Systems (FDS).
Examples of Computation:

**Computation set**:

\[
\emptyset \vdash 0\text{-positions.}
\]

**Justice set**:

\[
\{0\text{-positions}\} : J
\]

**Transition relation**:

\[
\begin{align*}
\delta = & \bar{r} \lor x = x' \lor \bar{m} = \bar{m} \lor 0^\bot \\
& \land (\bar{t}_j = 1^\bot \land 0^\bot) \lor 0^\bot = 0^\bot \\
\end{align*}
\]

**Initial condition**:

\[
\begin{align*}
\emptyset = & 0 \lor x = x \lor \bar{m} = \bar{m} \lor 0^\bot \\
& \land (\bar{t}_j = 1^\bot \land 0^\bot) \lor 0^\bot = 0^\bot \\
\end{align*}
\]

The Corresponding FDS

**State variables**:

\[
\begin{align*}
\{t_0, t_1, t_2, t_3\} : T
\end{align*}
\]

**Natural initial state**:

\[
\begin{align*}
\emptyset : \psi
\end{align*}
\]
expression, it becomes true and then

Semicritical

in the following, let \( q \) be a boolean expression, \( f \) be a natural variable, and by having no states in which this requirement holds.

Thus the sequence cannot terminate forever. While we can delay termination of the program for an arbitrary long time, we

**Conclusion:** Justice is not sufficient. We also need compassion.

The following program mux-sem, implements mutual exclusion by semaphores.

```
C := \{ (x < y \lor y < x) \}

The program consists of:

- Critical
- Non-critical

The semaphore instructions:

- release
- request
- loop forever do

```

Justice is not enough, you also need compassion.
A program $P$ has the form

\[ P = \langle \text{declaration}; \text{process}_1 \parallel \cdots \parallel \text{process}_m \rangle \]

where each $\text{process}_i$ is a process having the form

\[ \text{process}_i = \langle \text{declaration}; \text{statement} \rangle \]

### Deductive Verication

#### Programs

A program $P$ has the form

\[ P = \langle \text{declaration}; \text{process}_1 \parallel \cdots \parallel \text{process}_m \rangle \]

where each $\text{process}_i$ is a process having the form

\[ \text{process}_i = \langle \text{declaration}; \text{statement} \rangle \]

### Deductive Verication

#### Programs

A program $P$ has the form

\[ P = \langle \text{declaration}; \text{process}_1 \parallel \cdots \parallel \text{process}_m \rangle \]

where each $\text{process}_i$ is a process having the form

\[ \text{process}_i = \langle \text{declaration}; \text{statement} \rangle \]
can be attributed to one of their sub-statements.

Note that the conjunction and selection statements do not contribute any

\[(\land \text{pres}) \lor \text{d} \]

In addition to the above, the transition relation always contains the disjunct

\[\{\mu\} - a \land q \land \text{pres} \lor q^- \land \text{pres} \]

The while statement \(d\): while \(q\) do \(r\): \{\(S\)\} \: contributesto the disjunct

\[\{\{\mu\} - \land \text{pres}\} \lor \{i \land j \land \text{pres}\} \lor \{i \land j \land \text{pres}\}
\]

The release statement \(d\): release \(r\): \(\land \text{pres}\) \: contributesto the disjunct

\[\{\{\mu\} - \land \text{pres}\} \lor \{i \land j \land \text{pres}\} \lor \{i \land j \land \text{pres}\}
\]

The request statement \(d\): request \(r\): \(\land \text{pres}\) \: contributesto the disjunct

\[\{\{\mu\} - \land \text{pres}\} \lor \{i \land j \land \text{pres}\} \lor \{i \land j \land \text{pres}\}
\]

Which stays forever al \(d\) while \(q\) continuously holds:

\[\{\{\mu\} - \land \text{pres}\} \lor \{i \land j \land \text{pres}\} \lor \{i \land j \land \text{pres}\}
\]

The await statement \(d\): await \(q\): \(\land \text{pres}\) \: contributesto the disjunct

\[\{\{\mu\} - \land \text{pres}\} \lor \{i \land j \land \text{pres}\} \lor \{i \land j \land \text{pres}\}
\]
Formulas are satisifiable if there is a model of the formula. The semantics, then, is that for any model \( M \) of \( \mathcal{L} \), the formula \( \phi \) holds in \( M \) if and only if \( \phi \) is valid in \( M \).

### Modeling Systems

A system is modeled as a transition system, where the states are represented as atomic propositions and the transitions are represented as labeled edges. The semantics of a formula \( \phi \) is defined over the states of the system.

### Exercises

1. Given a model, prove that the formula \( \phi \) holds in the state \( s_0 \).
2. Determine whether the formula \( \phi \) is satisfiable for the system described above.
3. Prove that the formula \( \phi \) is valid for all models.

### Reading

Read the sections on semantics and models from the textbook. The key concepts to understand are:

- Temporal logic
- Satisfaction
- Validity
- Models

**Exercise:**

Prove that the formula \( \phi \) is satisfiable in the model described above. Use the semantics to show that there exists a model for \( \phi \).

---

**Problem:**

Given a model of a temporal formula, determine whether the formula is valid in the model. Use the semantics of temporal logic to prove your result.

---

**Solution:**

1. Use the semantics to determine the satisfaction of the formula in the model.
2. Prove that the formula is valid by constructing a model that satisfies the formula.
3. Use the properties of the model to show that the formula is valid.

---

**Exercise:**

Prove that the formula \( \phi \) is satisfiable in the model described above. Use the semantics to show that there exists a model for \( \phi \).

---

**Solution:**

1. Use the semantics to determine the satisfaction of the formula in the model.
2. Prove that the formula is satisfiable by constructing a model that satisfies the formula.
3. Use the properties of the model to show that the formula is satisfiable.

---

**Exercise:**

Prove that the formula \( \phi \) is valid in the model described above. Use the semantics to show that the formula is valid.

---

**Solution:**

1. Use the semantics to determine the validity of the formula in the model.
2. Prove that the formula is valid by constructing a model that satisfies the formula.
3. Use the properties of the model to show that the formula is valid.
Every temporal formula is equivalent to a conjunction of a reactivity formula, i.e.

\[ (\forall b \Box \Diamond \forall d \Box) \lor (\forall b \Box \Diamond) \lor (\forall d \Box) \]

By an equivalent characterization is the form \( d \Diamond \). The equivalence is justified by

\[ \leq d \Diamond \]

A formula of the form \( d \Diamond \) is called a safety formula.

A formula of the form \( d \Box \) is called a response formula.

A property is classified as a safety/response property if it can be specified by a

\[ \forall d \Diamond \]

Formal speciation of the main properties of program

Formal speciation of properties

Classificiation of Formulas/Properties

A formula of the form \( p \) is called a safety formula.

A formula of the form \( p \) is called a response formula.

Every temporal formula is equivalent to a conjunction of a reactivity formula, i.e.

\[ (\forall b \Box \Diamond \forall d \Box) \lor (\forall b \Box \Diamond) \lor (\forall d \Box) \]

By an equivalent characterization is the form \( d \Diamond \). The equivalence is justified by

\[ \leq d \Diamond \]

A formula of the form \( d \Diamond \) is called a safety formula.

A formula of the form \( d \Box \) is called a response formula.

A property is classified as a safety/response property if it can be specified by a

\[ \forall d \Diamond \]

Formal speciation of the main properties of program

Formal speciation of properties

Classification of Formulas/Properties

A formula of the form \( p \) is called a safety formula.

A formula of the form \( p \) is called a response formula.

Every temporal formula is equivalent to a conjunction of a reactivity formula, i.e.

\[ (\forall b \Box \Diamond \forall d \Box) \lor (\forall b \Box \Diamond) \lor (\forall d \Box) \]

By an equivalent characterization is the form \( d \Diamond \). The equivalence is justified by

\[ \leq d \Diamond \]

A formula of the form \( d \Diamond \) is called a safety formula.

A formula of the form \( d \Box \) is called a response formula.

A property is classified as a safety/response property if it can be specified by a

\[ \forall d \Diamond \]

Formal speciation of the main properties of program

Formal speciation of properties

Classification of Formulas/Properties

A formula of the form \( p \) is called a safety formula.

A formula of the form \( p \) is called a response formula.

Every temporal formula is equivalent to a conjunction of a reactivity formula, i.e.

\[ (\forall b \Box \Diamond \forall d \Box) \lor (\forall b \Box \Diamond) \lor (\forall d \Box) \]

By an equivalent characterization is the form \( d \Diamond \). The equivalence is justified by

\[ \leq d \Diamond \]

A formula of the form \( d \Diamond \) is called a safety formula.

A formula of the form \( d \Box \) is called a response formula.

A property is classified as a safety/response property if it can be specified by a

\[ \forall d \Diamond \]

Formal speciation of the main properties of program

Formal speciation of properties

Classification of Formulas/Properties

A formula of the form \( p \) is called a safety formula.

A formula of the form \( p \) is called a response formula.

Every temporal formula is equivalent to a conjunction of a reactivity formula, i.e.

\[ (\forall b \Box \Diamond \forall d \Box) \lor (\forall b \Box \Diamond) \lor (\forall d \Box) \]

By an equivalent characterization is the form \( d \Diamond \). The equivalence is justified by

\[ \leq d \Diamond \]

A formula of the form \( d \Diamond \) is called a safety formula.

A formula of the form \( d \Box \) is called a response formula.

A property is classified as a safety/response property if it can be specified by a

\[ \forall d \Diamond \]