CS256/Spring 2008 — Lecture #1
Zohar Manna

FORMAL METHODS FOR REACTIVE SYSTEMS

Instructor: Zohar Manna
Email: `zm@cs`
Office hours: by appointment

TA: Eric W. Smith
Email: `ewsmith@stanford`
Office hours: Tues. 3:45-4:45, Thurs. 3:45-4:45
Office: Gates 312

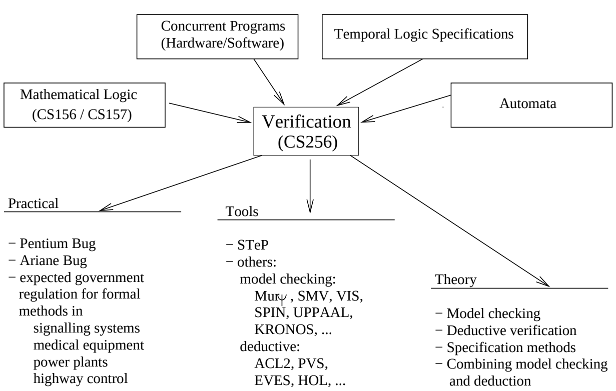
Web page:
`http://cs256.stanford.edu`
Course Meetings: TTh 12:50–2:05, Gates B12

Course work

- Weekly homeworks
- Final exam (3:30pm-6:30pm on Friday, June 6)

No collaboration on homeworks & exam
(but welcome otherwise).

No late homeworks.



Textbooks
Manna & Pnueli   Springer

Vol. I: "The Temporal Logic of Reactive and Concurrent Systems: Specification"
Springer 1992

Vol II: "Temporal Verification of Reactive Systems: Safety"
Springer 1995

Vol. III: "Temporal Verification of Reactive Systems: Progress"
Chapters 1–3, on Manna's web site.

Copies of lecture slides.

Papers.

Textbook Overview

(Volume II)

Chapter 0: Preliminary Concepts
[Summary of volume I]

Chapter 1: Invariance: Proof Methods

Chapter 2: Invariance: Applications

Chapter 3: Precedence

[**Chapter 4:** General Safety]

Chapter 5: Algorithmic Verification
("Model Checking")

Extra:

- ω-automata
- branching time logic CTL; BDDs

Transformational Systems

Observable only at the beginning and the end of their execution ("black box")

$$\xrightarrow{\text{input}} \boxed{\text{system}} \xrightarrow{\text{output}}$$

with no interaction with the environment.

- specified by

input-output relations
$\Downarrow$
state formulas (assertions)
First-Order Logic

- typically

terminating sequential programs
e.g., input $x \geq 0 \rightarrow$ output $z = \sqrt{x}$

Reactive Systems

Observable throughout their execution ("black cactus")

$\downarrow\ \uparrow\ \downarrow\ \uparrow\ \downarrow\ \uparrow$

system

$\downarrow\ \uparrow\ \downarrow\ \uparrow\ \downarrow\ \uparrow$

environment

$| \longrightarrow$ time

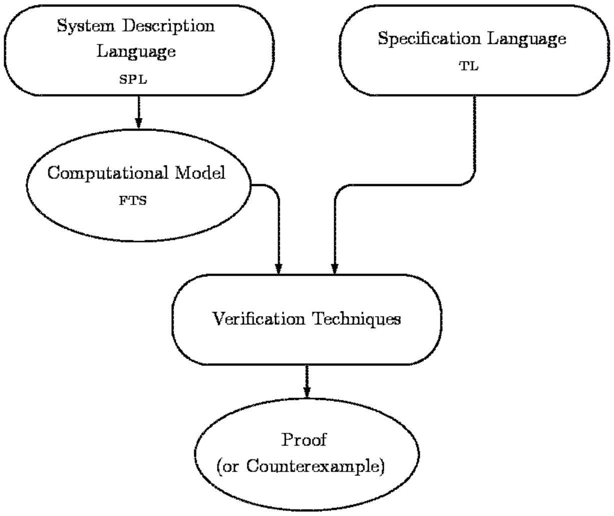
Interaction with the environment

- specified by

their on-going behaviors
(histories of interactions with their environment)
$\Downarrow$
sequence formulas
Temporal Logic

- Typically
  - Airline reservation systems
  - Operating systems
  - Process control programs
  - Communication networks

Overview of the Verification Process



The Components

- **System Description Language**
  SPL (Simple Programming Language)

  Pascal-like high-level language with constructs for

  - concurrency
  - nondeterminism
  - synchronous/asynchronous communication

- **Computational Model**
  FTS (Fair Transition System)

  Compact first-order representation of all sequences of states that can be generated by a system

The Components (cont.)

- **Specification Language**
  TL (temporal logic)

  models of a TL formula are infinite sequences of states

- **Verification Techniques**

  - algorithmic (model checking)
    search a state-graph for counterexample
  - deductive (theorem proving)
    prove first-order verification conditions

Reactive System	**Specification**
SPL Program P	TL formula ψ
$\downarrow$	
Fair Transition System (FTS) Φ	$\downarrow$
$\downarrow$	

Verification

Proof	**Counterexample**
$\mathrm{Com}(\Phi) \subseteq \mathrm{Mod}(\psi)$ i.e., all computations of Φ are models of ψ	computation σ of Φ, s.t. $\sigma \notin \mathrm{Mod}(\psi)$

Chapter 0:

Preliminary Concepts

States

- vocabulary $\mathcal{V}$ — set of typed variables
  (type defines the domain over which the values can range)
  - expression over $\mathcal{V}$ $\quad x + y$
  - assertion over $\mathcal{V}$ $\quad x > y$
- state s — interpretation over $\mathcal{V}$

Example:
$\mathcal{V} = \{x, y : \text{integer}\}$
$s = \{x : 2, y : 3\}$
(also written as $s[x] = 2, \quad s[y] = 3$)
$x + y$ is 5 on s
$x > y$ false on s

- Σ — set of all states

Fair Transition System (FTS)

$$\Phi = \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$$

(represents a Reactive Program)

- $V = \{u_1, \ldots, u_n\} \subseteq \mathcal{V}$ — vocabulary

  A finite set of system variables

  System variables = data variables + control variables

- Θ — initial condition

  First-order assertion over V that characterizes all initial states

Example:
$\Theta: \quad x = 5 \ \wedge \ 3 \leq y \leq 5$
initial states: $\{x : 5, y : 3\}$
$\{x : 5, y : 4\}$
$\{x : 5, y : 5\}$

- $\mathcal{T}$ — finite set of transitions

  For each $\tau \in \mathcal{T}$,

  $$\tau : \Sigma \rightarrow 2^{\Sigma}$$

  (τ is a function from states to sets of states)

  - s' is a τ-successor of s if $s' \in \tau(s)$
  - τ is represented by the transition relation ("next-state" relation) $\rho_\tau(V, V')$ where

    V – values of variables in the current state

    V' – values of variables in the next state

Example:
$\rho_\tau : x' = x + 1$ means
$s'[x] = s[x] + 1$

  - special idling (stuttering) transition τ_I,

    $$\rho_{\tau_I} : V = V'$$

Example:

$\langle x:5, y:3\rangle \xrightarrow{\tau} \{\langle x:5, y:4\rangle, \langle x:5, y:5\rangle\}$

"When in state $\langle x:5, y:3\rangle$ τ may increment y by either 1 or 2, and keep x unchanged."

$\langle x:5, y:4\rangle$ and $\langle x:5, y:5\rangle$ are τ-successors of $\langle x:5, y:3\rangle$.

- $\mathcal{J} \subseteq \mathcal{T}$: set of just (weakly fair) transitions
- $\mathcal{C} \subseteq \mathcal{T}$: set of compassionate (strongly fair) transitions

Enabled/Disabled/Taken Transition

- For each $\tau \in \mathcal{T}$,
  τ is enabled on s if $\tau(s) \neq \emptyset$
  τ is disabled on s if $\tau(s) = \emptyset$
- For an infinite sequence of states
  $\sigma : \; s_0, \; s_1, \; s_2, \; \ldots, \; s_k, \; s_{k+1}, \; \ldots$
  - $\tau \in \mathcal{T}$ is enabled at position k of σ if τ is enabled on s_k
  - $\tau \in \mathcal{T}$ is taken at position k of σ if s_{k+1} is a τ-successor of s_k

Example:

$\rho_\tau : \; x = 5 \;\wedge\; x' = x + 1 \;\wedge\; y' = y$

τ is enabled on all states s.t. $s[x] = 5$ and disabled on all other states

$\sigma : \ldots \overbrace{\langle x:5, y:3\rangle}^{s_k}, \overbrace{\langle x:6, y:3\rangle}^{s_{k+1}} \ldots$

τ is enabled at position k
τ is taken at position k

Computation

Infinite sequence of states

$$\sigma : \; s_0, \; s_1, \; s_2, \; \ldots$$

is a computation of an FTS Φ (Φ-computation), if it satisfies the following:

- Initiality: s_0 is an initial state (satisfies Θ)
- Consecution: For each $i = 0, 1, \ldots$,
  $s_{i+1} \in \tau(s_i)$ for some $\tau \in \mathcal{T}$.

- Justice: For each $\tau \in \mathcal{J}$, it is $\boxed{\text{not}}$ the case that τ is continually enabled beyond some position j in σ but not taken beyond j.

```
Example:
```

$V : \{x : \text{integer}\}$
$\Theta : x = 0$
$\mathcal{T} : \{\tau_I, \tau_{\text{inc}}\}$ with $\rho_{\tau_{\text{inc}}} : x' = x + 1$
$\mathcal{J} : \{\tau_{\text{inc}}\}$
$\mathcal{C} : \emptyset$

$\sigma : \langle x : 0 \rangle \xrightarrow{\tau_I} \langle x : 0 \rangle \xrightarrow{\tau_I} \langle x : 0 \rangle \xrightarrow{\tau_I} \ldots$

satisfies Initiality and Consecution, but not Justice.
Therefore σ is not a computation.

(In any computation of this system, x grows beyond any bound.)

$$\sigma : \left[\begin{array}{l} \langle x : 0 \rangle \longrightarrow \langle x : 1 \rangle \longrightarrow \langle x : 2 \rangle \longrightarrow \langle x : 2 \rangle \longrightarrow \\ \quad \langle x : 3 \rangle \longrightarrow \langle x : 3 \rangle \longrightarrow \langle x : 3 \rangle \longrightarrow \\ \quad \langle x : 4 \rangle \longrightarrow \cdots \end{array} \right.$$

is a computation

Question: $\rho_{\tau_{\text{inc}}} : (x = 0 \lor x = 1) \land x' = x + 1$
Is

$$\sigma : \left[\begin{array}{l} \langle x : 0 \rangle \longrightarrow \langle x : 1 \rangle \longrightarrow \langle x : 2 \rangle \longrightarrow \\ \quad \langle x : 2 \rangle \longrightarrow \langle x : 2 \rangle \longrightarrow \cdots \end{array} \right.$$

a computation?

- Compassion: For each $\tau \in \mathcal{C}$, it is $\boxed{\text{not}}$ the case that τ is enabled at infinitely many positions in σ, but taken at only finitely many positions in σ.

```
Example:
```

$V : \{x, y : \text{integer}\}$
$\Theta : x = 0 \land y = 0$
$\mathcal{T} : \{\tau_I, \tau_x, \tau_y\}$ with
$\quad \rho_{\tau_x} : x' = x + 1 \bmod 2$
$\quad \rho_{\tau_y} : x = 1 \land y' = y + 1$
$\mathcal{J} : \{\tau_x\}$
$\mathcal{C} : \{\tau_y\}$

$\sigma : \langle \overset{x}{0}, \overset{y}{0} \rangle \xrightarrow{\tau_x} \langle 1, 0 \rangle \xrightarrow{\tau_x} \langle 0, 0 \rangle \xrightarrow{\tau_x} \ldots$

is not a computation: τ_y is infinitely often enabled, but never taken.
(**Note**: If τ_y had only been just, σ would have been a computation, since τ_y is not continually enabled.)

FTS $\Phi = \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$

Run	=	Initiality + Consecution
Fairness	=	Justice + Compassion
Computation	=	Run + Fairness

Notation: $s_0 \overset{\tau_1}{\to} s_1 \overset{\tau_2}{\to} s_2 \overset{\tau_3}{\to} s_3 \to \ldots$

Note: For every two consecutive states s_i, s_{i+1}, there may be more than one transition that leads from s_i to s_{i+1}.
Therefore, several different transitions can be considered as taken at the same time.

Finite-State

- For a computation σ of Φ

$$\sigma: \ s_0, \ s_1, \ s_2, \ \ldots, \ s_i, \ \ldots \ ,$$

  state s_i is a Φ-accessible state.

- Φ is finite-state if the set of Φ-accessible states is finite. Otherwise, it is infinite-state.
  - If the domain of all variables of Φ is finite, (e.g., booleans, subranges, etc.), then Φ is finite-state.
  - Even if the domain of some variables of Φ is infinite (e.g., integer), Φ may still be finite-state.

`Example:`

$V : \{x : \text{integer}\}$
$\Theta : x = 1$
$\mathcal{T} : \{\tau_I, \tau_1, \tau_2\}$ with
$\quad \rho_{\tau_1} : x = 1 \wedge x' = 2$
$\quad \rho_{\tau_2} : x = 2 \wedge x' = 1$
$\mathcal{J}, \mathcal{C} : \emptyset$

has 2 accessible states:
$\langle x : 1 \rangle$ and $\langle x : 2 \rangle$