CS 267: Automated Verification

Lecture 1: Brief Introduction. Transition Systems. Temporal Logic LTL.

Instructor: Tevfik Bultan

What do these people have in common?

2013 Leslie Lamport

2007 Clarke, Edmund M

2007 Emerson, E Allen

2007 Sifakis, Joseph

1996 Pnueli, Amir

1991 Milner, Robin

1980 Hoare, C. Antony R.

1978 Floyd, Robert W

1972 Dijkstra, E. W.

State of the art in automated verification: Model Checking

- What is model checking?
  - Automated verification technique
  - Focuses on bug finding rather than proving correctness
  - The basic idea is to exhaustively search for bugs in software
  - Has many flavors
    - Explicit-state model checking
    - Symbolic model checking
    - Bounded model checking

Hardware to Software Model Checking

- In 90s model checking was mainly used in industry as a technique for analyzing hardware designs
  - Most hardware companies had their in house automated verification tools
- In the last ten years very promising results have been obtained in verification of software
  - Microsoft started using a model checker to verify device drivers
    - Based on a research project from Microsoft Research
  - Model checking tools found numerous bugs in Linux code

Is There More Research Left To Do?

- Model checking does not scale very well
  - To verify a program you need to investigate all possible states (configurations) of the program somehow
  - In theory: inifinite state $\Rightarrow$ undecidable
  - In practice: finite but large number of states $\Rightarrow$ run out of memory
- We look for ways to reduce the state space while showing that properties we are interested are preserved in the transformed system
  - symbolic representations
  - modularity
  - abstraction
  - symmetry reduction, etc.

Beyond Model Checking

- Promising results obtained in the model checking area created a new interest in automated verification
- Nowadays, there is a wide spectrum of verification/analysis/ testing techniques with varying levels of power and scalability
  - Bounded verification using SAT solvers
  - Symbolic execution using Satisfiability Modulo Theories (SMT) solvers
  - Dynamic symbolic execution (aka concolic execution)
  - Various types of symbolic analysis: shape analysis, string analysis, size analysis, etc.
- Taking this course should give you a better understanding of all these techniques

What to Verify

- Before we start talking about automated verification techniques, we need to identify what we want to verify

- It turns out that this is not a very simple question

- For the rest of this lecture we will discuss issues related to this question

A Mutual Exclusion Protocol

Two concurrently executing processes are trying to enter a critical section without violating mutual exclusion

```
Process 1:
while (true) {
   out:  a := true; turn := true;
   wait: await (!b or !turn);
   cs:   a := false;
}
||
Process 2:
while (true) {
   out:  b := true; turn := false;
   wait: await (!a or turn);
   cs:   b := false;
}
```

Reactive Systems: A Very Simple Model

- We will use a very simple model for reactive systems
- A reactive system generates a set of ***execution paths***
- An execution path is a concatenation of the states (configurations) of the system, starting from some ***initial state***
- There is a ***transition relation*** which specifies the *next-state* relation, i.e., given a state what are the states that can follow that state

State Space

- The state space of a program can be captured by the valuations of the variables and the program counters

- For our example, we have
  - two program counters: `pc1, pc2`

    domains of the program counters: `{out, wait, cs}`
  - three boolean variables: `turn, a, b`

    boolean domain: `{True, False}`

- Each ***state*** of the program is a valuation of all the variables

State Space

- Each state can be written as a tuple
  `(pc1,pc2,turn,a,b)`

- Initial states: `{(o,o,F,F,F), (o,o,F,F,T), (o,o,F,T,F), (o,o,F,T,T), (o,o,T,F,F), (o,o,T,F,T), (o,o,T,T,F), (o,o,T,T,T)}`
  - initially: `pc1=o and pc2=o`

- How many states total?
  3 * 3 * 2 * 2 * 2 = 72
  exponential in the number of variables and the number of concurrent components

Transition Relation

- Transition Relation specifies the next-state relation, i.e., given a state what are the states that can come immediately after that state
- For example, given the initial state `(o,o,F,F,F)`

  Process 1 can execute:

  ```
  out:  a := true; turn := true;
  ```

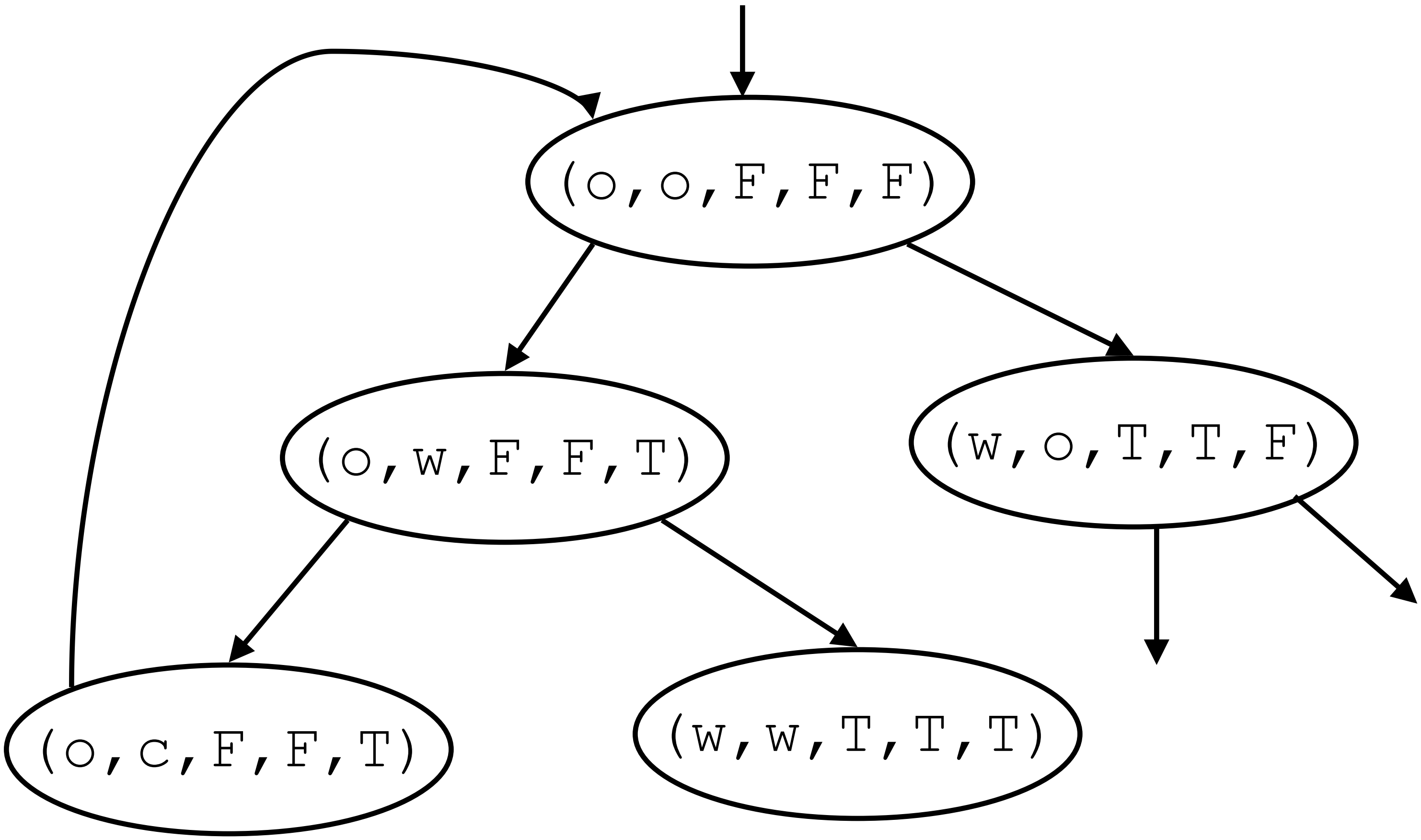
  or Process 2 can execute:

  ```
  out:  b := true; turn := false;
  ```

- If process 1 executes, the next state is `(w,o,T,T,F)`
- If process 2 executes, the next state is `(o,w,F,F,T)`
- So the state pairs `((o,o,F,F,F),(w,o,T,T,F))` and `((o,o,F,F,F),(o,w,F,F,T))` are included in the transition relation

Transition Relation

The transition relation is like a graph, edges represent the next-state relation



Transition System

- A ***transition system*** $T = (S, I, R)$ consists of
  - a set of states S
  - a set of initial states $I \subseteq S$
  - and a transition relation $R \subseteq S \times S$

- A common assumption in model checking
  - R is total, i.e., for all $s \in S$, there exists s' such that $(s,s') \in R$

Execution Paths

- A ***path*** in $T = (S, I, R)$ is an infinite sequence of states

  $x = s_0, s_1, s_2, \ldots$

  such that for all $i \geq 0$, $(s_i, s_{i+1}) \in R$

Notation: For any path x

x_i denotes the i' th state on the path (i.e., s_i)

x^i denotes the i' th suffix of the path (i.e., $s_i, s_{i+1}, s_{i+2}, \ldots$)

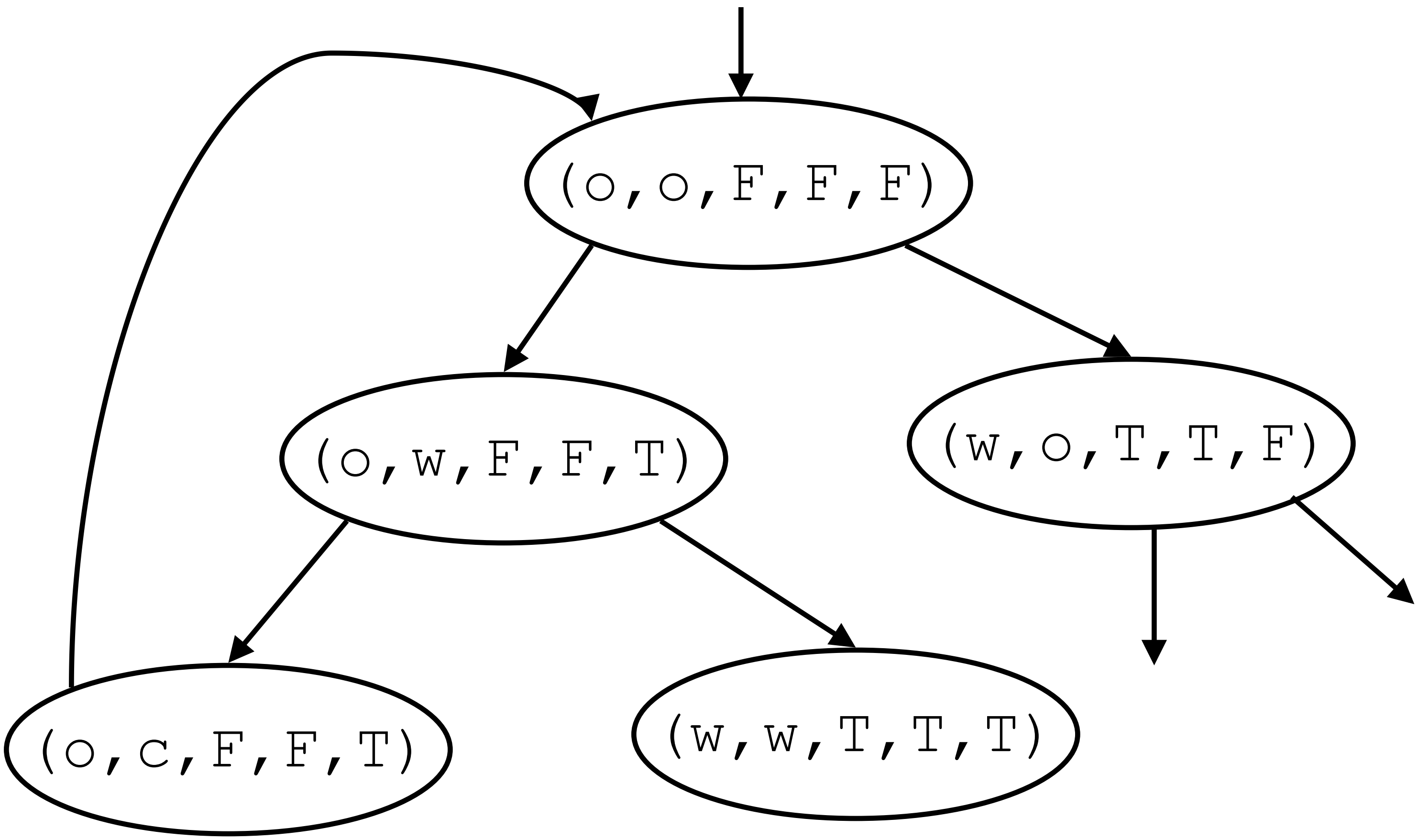
- An ***execution path*** in $T = (S, I, R)$ is a path x in $T = (S, I, R)$ where $x_0 \in I$

Execution Paths

A possible execution path:

`((o,o,F,F,F), (o,w,F,F,T), (o,c,F,F,T))`$^{\omega}$

(ω means repeat the above three states infinitely many times)



Temporal Logics

- Pnueli proposed using temporal logics for reasoning about the properties of reactive systems

- Temporal logics are a type of modal logics
  - Modal logics were developed to express modalities such as "necessity" or "possibility"
  - Temporal logics focus on the modality of temporal progression

- Temporal logics can be used to express, for example, that:
  - an assertion is an invariant (i.e., it is true all the time)
  - an assertion eventually becomes true (i.e., it will become true sometime in the future)

Temporal Logics

- We will assume that there is a set of basic (***atomic***) ***properties*** called AP
  - These are used to write the basic (non-temporal) assertions about the program
  - Examples: `a=true, pc0=c, x=y+1`

- We will use the usual boolean connectives: $\neg$, $\wedge$, $\vee$

- We will also use four ***temporal operators***:

Invariant *p*	:	G *p*	(aka $\Box p$)	(Globally)
Eventually *p*	:	F *p*	(aka $\Diamond p$)	(Future)
Next *p*	:	X *p*	(aka $\bigcirc p$)	(neXt)
p **Until** *q*	:	*p* U *q*		

Atomic Properties

- In order to define the semantics we will need a function L which evaluates the truth of atomic properties on states:

$$L : S \times AP \rightarrow \{\text{True, False}\}$$

L((o,o,F,F,F), pc1=o) = True

L((o,o,F,F,F), pc1=w) = False

L((o,o,F,F,F), turn) = False

L((o,o,F,F,F), turn=false) = True

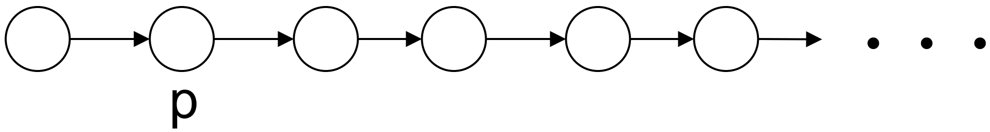
Linear Time Temporal Logic (LTL) Semantics

Given a path x and LTL properties p and q

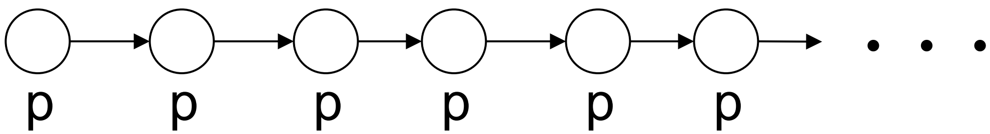
$x \models p$	iff	$L(x_0, p) = \text{True}$, where $p \in AP$
$x \models \neg p$	iff	not $x \models p$
$x \models p \wedge q$	iff	$x \models p$ and $x \models q$
$x \models p \vee q$	iff	$x \models p$ or $x \models q$
$x \models X\ p$	iff	$x^1 \models p$
$x \models G\ p$	iff	for all $i \geq 0$, $x^i \models p$
$x \models F\ p$	iff	there exists an $i \geq 0$ such that $x^i \models p$
$x \models p\ U\ q$	iff	there exists an $i \geq 0$ such that $x^i \models q$ and for all $0 \leq j < i$, $x^j \models p$

LTL Properties

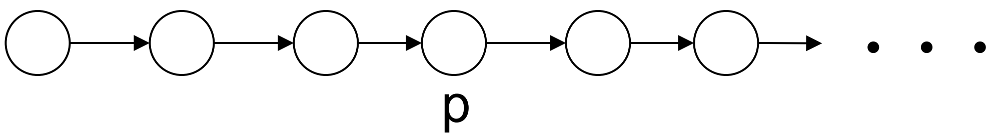
X p



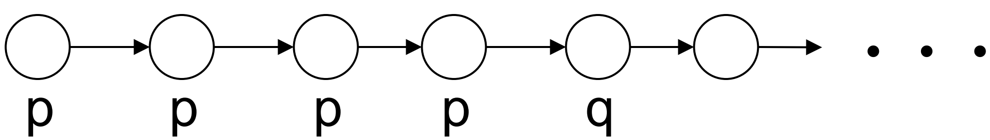
G p



F p



p U q



Example Properties

mutual exclusion: $G(\neg(pc1=c \wedge pc2=c))$

starvation freedom:

$$G(pc1=w \Rightarrow F(pc1=c)) \wedge G(pc2=w \Rightarrow F(pc2=c))$$

Given the execution path:

$x = ((o,o,F,F,F),\ (o,w,F,F,T),\ (o,c,F,F,T))^{\omega}$

$x \models pc1=o$

$x \models X\,(pc2=w)$

$x \models F\,(pc2=c)$

$x \models (\neg turn)\ U\ (pc2=c \wedge b)$

$x \models G(\neg(pc1=c \wedge pc2=c))$

$x \models G(pc1=w \Rightarrow F(pc1=c)) \wedge G(pc2=w \Rightarrow F(pc2=c))$

LTL Equivalences

- We do not really need all four temporal operators
  - X and U are enough (i.e., X, U, AP and boolean connectives form a basis for LTL)

$F\ p = true\ U\ p$

$G\ p = \neg\ (F \neg p) = \neg\ (true\ U\ \neg p)$

LTL Model Checking

- Given a transition system T and an LTL property p

  T |= p   iff   for all execution paths x in T, x |= p

For example:

T |=? $G(\neg(pc1=c \wedge pc2=c))$

T |=? $G(pc1=w \Rightarrow F(pc1=c)) \wedge G(pc2=w \Rightarrow F(pc2=c))$

Model checking problem: Given a transition system T and an LTL property p, determine if T is a model for p (i.e., if T |=p)