Lecture 10 - February 5

Math Review Exercises, Model Checking

Nested Quantification
Model Checking Intro: $\vdash$ ***vs.*** $\models$
State Graph vs. (Computation) Paths

Announcements/Reminders

- ProgTest1 guide released
- Mockup Test scheduled in tomorrow's lab session
- Lab1 solution released
- Lab2 released
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- TA contact information (on-demand for labs) on eClass

Predicate Logic: Exercise 1

Consider the following predicate:

$$\forall\, x, y \bullet x \in \mathbb{N} \land y \in \mathbb{N} \Rightarrow x * y > 0$$

not a theorem.

to disprove give a witness of violation

Choose **all** statements that are **correct**.

- ✗ 1. It is a theorem, provable by (5, 4).
- ✗ 2. It is a theorem, provable by (2, 3).
- ✓ 3. It is not a theorem, witnessed by (5, 0).
- ✗ 4. It is not a theorem, witnessed by (12, -2).
- ✗ 5. It is not a theorem, witnessed by (12, 13).

$5 \in \mathbb{N} \land 0 \in \mathbb{N} \Rightarrow 5 * 0 > 0$ (T ⇒ F)

$12 \in \mathbb{N} \land 13 \in \mathbb{N} \Rightarrow 12 * 13 > 0$ (T ⇒ T)

$12 \in \mathbb{N} \land -2 \in \mathbb{N} \Rightarrow 12 * -2 > 0$ (F ⇒ T)

Predicate Logic: Exercise 2

Consider the following predicate:

$$\exists x, y \bullet x \in \mathbb{N} \land y \in \mathbb{N} \land x * y > 0$$

a theorem

to prove, show a witness of satisfaction.

Choose **all** statements that are **correct**.

1. ✓ It is a theorem, provable by (5, 4). $5 \in \mathbb{N} \land 4 \in \mathbb{N} \land 5 * 4 > 0$ T
2. ✓ It is a theorem, provable by (2, 3).
3. ✗ It is a theorem, provable by (-2, -3). $-2 \in \mathbb{N}$ $-3 \in \mathbb{N}$ $-2 * -3 > 0$ F
4. ✗ It is not a theorem, witnessed by (5, 0).
5. ✗ It is not a theorem, witnessed by (12, -2).
6. ✗ It is not a theorem, witnessed by (12, 13).

Nested Logical Quantifiers

$\forall i \bullet i \in \mathbb{Z} \Rightarrow (\exists j \bullet j \in \mathbb{N} \wedge i + j = 0)$

$\forall j \cdot j \in \mathbb{N} \wedge (i+j) \neq 0$

$\Leftrightarrow$

(F) To disprove it: choose some i s.t. $\neg(\exists j \cdot j \in \mathbb{N} \wedge i + j = 0)$

Witness: choose $i = 2$, $\neg\exists j \cdot j \in \mathbb{N} \wedge 2 + j = 0$

$\forall i \bullet \underline{i \in \mathbb{N}} \Rightarrow (\exists j \bullet j \in \mathbb{Z} \wedge i + j = 0)$ P

can't be negative.

(T) To prove it: argue that all possible values of i satisfy P

$i \in \mathbb{N} \Rightarrow i \geq 0$ for each i, choose $j = -i \in \mathbb{Z}$

$\exists i \bullet i \in \mathbb{N} \wedge (\forall j \bullet j \in \mathbb{Z} \Rightarrow i \cdot j > 0)$ P

show $\neg P$

(F) $i \in \mathbb{N} \Rightarrow i \geq 0$ witness: choose $j = 0$ or $j < 0$

$\exists i \bullet i \in \mathbb{N} \wedge (\forall j \bullet j \in \mathbb{Z} \Rightarrow i \cdot j \geq 0)$

(T) Witness: choose $i = 0$ $\Rightarrow$ each $j \in \underline{\mathbb{Z}}$ is s.t. $0 \cdot j \geq 0$

Use of Model Checking in Industry

Pentium FDIV bug: https://en.wikipedia.org/wiki/Pentium_FDIV_bug

*The Pentium FDIV bug is a hardware bug affecting the **floating-point unit (FPU)** of the early Intel Pentium processors. Because of the bug, the processor would return incorrect binary floating point results when dividing certain pairs of high-precision numbers.*

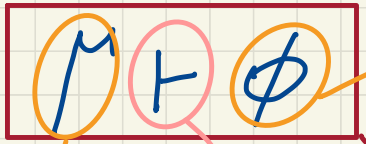
*In December 1994, Intel **recalled** the defective processors ... In its 1994 annual report, Intel said it incurred "**a $475 million pre-tax charge** ... to recover replacement and write-off of these microprocessors."*

*In the aftermath of the **bug** and subsequent **recall**, there was a marked increase in the use of formal verification of hardware floating point operations across the **semiconductor industry**. Prompted by the discovery of the bug, a technique ... called "word-level **model checking**" was developed in 1996. Intel went on to use **formal verification** extensively in the development of later CPU architectures. In the development of the Pentium 4, symbolic trajectory evaluation and **theorem proving** were used to **find a number of bugs that could have led to a similar recall incident** had they gone undetected.*

instruction spec.

Formal Verification: Proof Based vs. Check Based

3342 4315

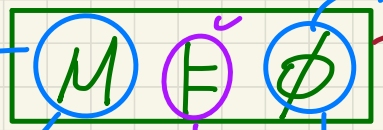


system formula written as predicates (axioms, invariants, guards, before-after predicates)

a series of applications of inference rules.

undecidable problem

proofs in general are done:
(1) manually
(2) interactively (guide the prover)



automated RG
(a) build the
(b) traverse the graph

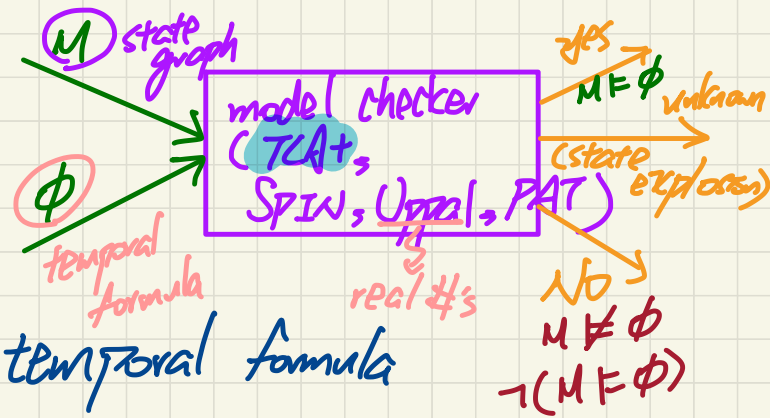
invariant
temporal logic
- LTL
- CTL

system descriptions (formulated as a labelled transition system)

"models"/ satisfies

LTS (a graph with:
(a) finite # of states & transitions
and
(b) infinite # of possible paths

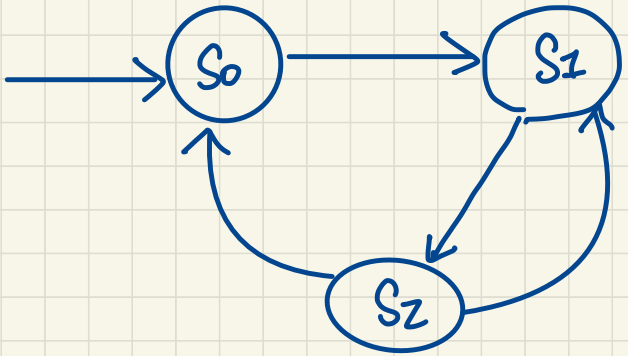
Temporal Logic



- Syntax: structural rules for writing compound temporal formula
  - → syntax tree
  - ≒ context-free grammar

- semantics: given a temporal formula ϕ that's syntactically correct, what's its meaning?
  - (1) how to express
  - (2) how to check if a formula is satisfies ($M \stackrel{?}{\models} \phi$)
  - (3) when the check fails, how to interpret & find error traces?

State Graph vs. (Computation) Path



Finite # states : S_0, S_1, S_2

# of paths is infinite :
there's at least one cycle.

(P1) $S_0 \to S_1 \to S_2 \to S_0 \to \cdots$

(P2) $S_0 \to S_1 \to S_2 \to S_1 \to S_2 \to S_1 \to \cdots$

trace.

An infinite sequence of states models the time

temporal logic