Lecture 26

- Announcements

- My grading deadline : May 18

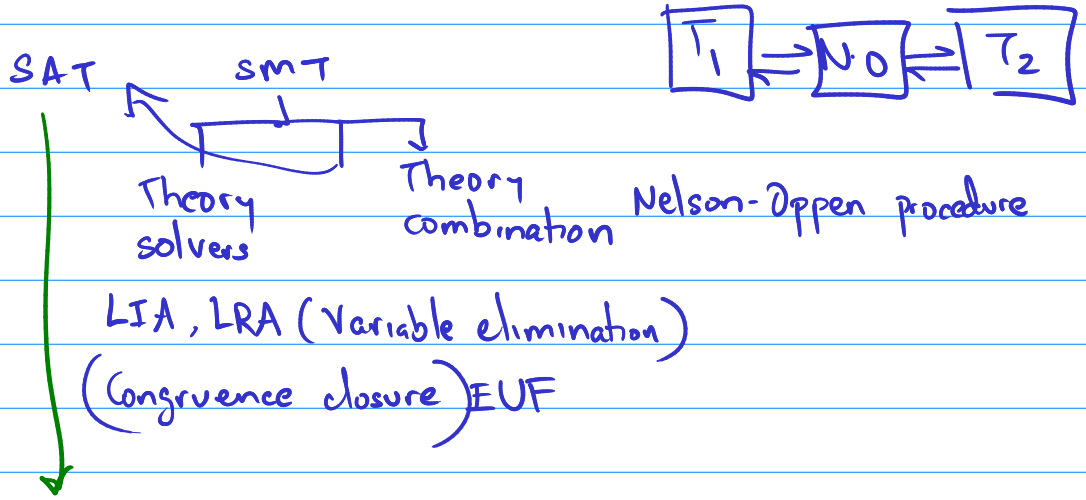
  May 14

- What remains still :- HW3 + HW4 ← May 10?

- Problem background + challenges ← - Project presentation
- What did you do? + Demo? - Project report (+ github links) ↖ May 14
- How well did it work? / Experiments
- What did you learn? What worked, what didn't work, why, etc.

① What did we discuss?

Unit 1: Constraint solvers



① What if the formula was unsatisfiable?
- Convince me that it is unsat / Why was it unsat?
  Unsat cores.
- How close is it to being satisfiable?
  Max SAT ⟶ Max SMT
  Weighted Max SAT

④ Theorem provers for first order logic. Automatic.
Vampyre
see TPTP Problems
Thousands of theorem provers

② Ok, it is satisfiable. But how many satisfying assignments does it admit?

$p \vee (p \wedge q)$ → 2 sat asgnts #SAT, random sampling

$(x > 5) \vee (x > 5 \wedge y < 10)$ ~~#SMT~~

③ What if the variables were quantified?

See M. Vardi's talk
NP is the new P.

QBF

Emina Torlak
Rosette

Sample complexity

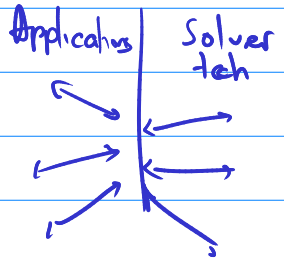
PROSE

Adding synthesis technology to an existing language.

Unit 2: Program synthesis

Flexible ways of specifying intent

- Input-output examples
- Logical specifications
- Snippets of executable code.



$\forall x.\ \varphi(f, x)$ Syntax Guided Synthesis

$\varphi(f, I_1) \wedge \varphi(f, I_2) \wedge \ldots$

Counterexample Guided Inductive Synthesis (CEGIS)

- Enumerative ESolver
- Stochastic solver
- Symbolic solver

① Synthesis from types $f : List\langle Int \rangle \to Int$

Type inhabitation, STLC ← simply typed lambda calculus

Curry-Howard correspondence

Pspace complete

Types ⟷ Propositions

Expressions ⟷ Proofs

$x + 5$

```
B foo<A,B>(A a, A->B f)
{
    return f(a);
}
```

$\forall A.B.\ A \to (A \to B) \to B$

$A \to (A \Rightarrow B) \Rightarrow B$

Unit 3: Proof techniques

Claim: Verifying loop-free code (over finite data domains) is easy.

Inductive invariants. vs. (non-inductive) invariants.

Hoare logic

- Safety properties / partial correctness
- Liveness / termination checking

Resource usage analysis.

Assume input program needs $O(n^2)$ time an^2+bn+c

When we reach line 5, $x != null$.

Whenever we reach a line of code, some assertion always holds

We will eventually reach some line of code.

1. Crash!
2. Does not terminate
3. Exits successfully.

Think about fairness. COMPAS.

$f(x) \approx f(x+\delta)$

Hyperproperties.

"Something is always true"

"Something will eventually be true".

Logics of time — Amir Pnueli

Modeling the heap

Concurrency

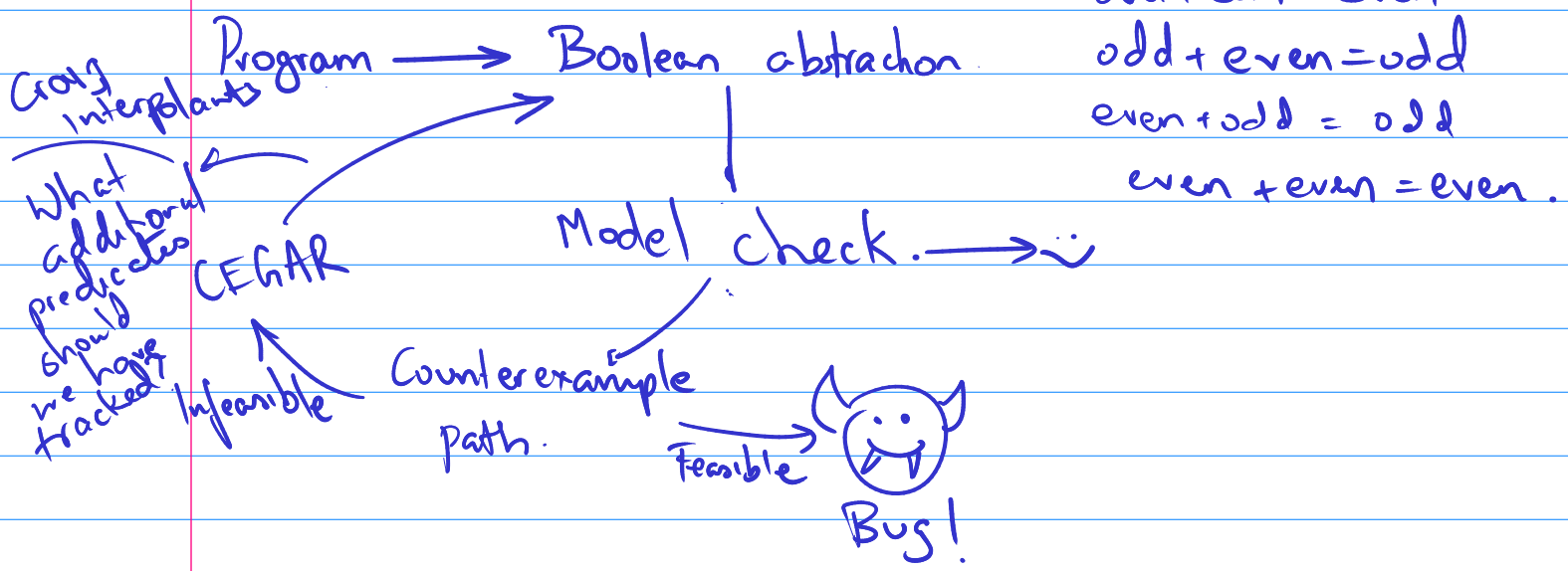
```
int y=10;
int * x = &y;
*x = 3.
int * x = malloc(sizeof(int));
```

Separation logic: Peter O'Hearn. → What is the value of y here?

Shape analysis

y = 3.

Unit 4: Static analysis / predicate abstraction.



odd + odd = even
odd + even = odd
even + odd = odd
even + even = even.

② What research is happening today?

- Concurrency, fairness + privacy
  - int x = 10;
  - int x ~ Unif(1,10);
- verification for AI ⟷ probabilistic programs
- AI for verification?
- Quantum programming languages
- Blockchains / smart contracts / distributed ledgers.
- Software defined networks
- How do we verify compilers themselves?

④ Big Picture

- Can we help programmers write better code?

Intent: Verifying code. Synthesis.

- Can we help programmers understand existing code?

Easier human-machine interfaces?