Today

Wrapping up static and dynamic program analysis
- Reasoning about programs
- Invariants and Daikon
- A primer on solver-aided reasoning

Course overview: the big picture

- **Week 1**: Introduction & static vs. dynamic analysis
  - HW 1
- **Week 2**: Abstract Interpretation
- **Week 3**: Abstract Interpretation
  - HW 2
- **Week 4**: Testing
- **Week 5**: Delta Debugging
  - In-class exercise
- **Week 6**: Slicing & Invariants
- **Week 7**: Program Repair
- **Week 8**: Empirical Software Engineering
- **Week 9**: ML for Software Engineering
- **Week 10**: Wrap up
  - Project presentation

Let's take a step back...
Reasoning about programs

Use cases
- Verification/testing: ensure code is correct
- Prove facts to be true, e.g.:
  - \( x \) is never null
  - \( y \) is always greater than 0
  - Input array \( a \) is sorted
- Debugging: understand why code is incorrect

Approaches
- Abstract interpretation
- Testing
- Slicing
- Delta debugging

Forward vs. backward reasoning

Forward reasoning
- Usually more intuitive (simulate the code)
- Reasoning about what will happen
- May introduce irrelevant facts
- Set of learned facts may get (too) large

Backward reasoning
- Usually more helpful (starting from a known point of interest)
- Reasoning about what must have happened
  - Given a specific goal, how to achieve it?
  - Given an error, what test case exposes it?

Preconditions and postconditions

```java
public double avgAbs(double[] nums)
{
    int n = nums.length;
    double sum = 0;
    int i = 0;
    while (i != n) {
        if(nums[i] > 0) {
            sum = sum + nums[i];
        } else {
            sum = sum - nums[i];
        }
        i = i + 1;
    }
    return sum / n;
}
```

What are preconditions and postconditions of this method?
Preconditions and postconditions

```java
double avgAbs(double[] nums) {
  int n = nums.length;
  double sum = 0;
  int i = 0;
  while (i != n) {
    if(nums[i] > 0) {
      sum = sum + nums[i];
    } else {
      sum = sum - nums[i];
    }
    i = i + 1;
  }
  return sum / n;
}
```

**Preconditions**
- `nums` is not null
- `nums.length > 0`

**Postconditions**
- `nums` has not changed
- `return val >= 0`
- ...

Does this loop terminate?
What are preconditions, postconditions, and loop invariants?

Explicitly stating invariants is hard -- reasoning about inferred variants might be easier.

Daikon live example

https://plse.cs.washington.edu/daikon/download/doc/daikon.html#StackAr-example
Daikon: general workflow

A primer on solver-aided reasoning
(507 covers this topic in detail)

What is Z3?
- An SMT (Satisfiability Modulo Theories) solver.
- Uses a standard language (SMT-LIB).
  - Print to the screen.
  - Declare variables and functions.
  - Define constraints.
  - Check satisfiability and obtain a model.
  - ...

Which question does this code answer?

```lisp
(echo "Running Z3...")
(declare-const a Int)
(assert (> a 0))
(check-sat)
(get-model)
```
What is Z3?

- An SMT (Satisfiability Modulo Theories) solver.
- Uses a standard language (SMT-LIB).
  - Print to the screen.
  - Declare variables and functions.
  - Define constraints.
  - Check satisfiability and obtain a model.
  - ...

```
(echo "Running Z3...")
(declare-const a Int)
(assert (> a 0))
(check-sat)
(get-model)
```

This code is asking the question:
Does an integer greater than 0 exist?

A first example

```
int simpleMath(int a, int b) {
  assert(b>0);
  if(a + b < a) {
    return 1;
  }
  return 0;
}
```

Does this method ever return 1? Let's ask Z3...

```
(declare-const a Int)
(declare-const b Int)
(assert (> b 0))
(assert (< (+ a b) a))
(check-sat)
```

Does this method ever return 1? Let's ask Z3...

A first example: limited precision

```
int simpleMath(int a, int b) {
  assert(b>0);
  if(a + b < a) {
    return 1;
  }
  return 0;
}
```

Z3 supports Bitvectors of arbitrary size.
Let's model Java ints (32 bits) and ask the same question...

```
(declare-const a (_ BitVec 32))
(declare-const b (_ BitVec 32))
(assert (bvsgt b #x00000000))
(assert (bvslt (bvadd a b) a))
(check-sat)
(get-model)
```
A more complex example

```c
int getNumber(int a, int b, int c) {
    if (c==0) return 0;
    if (c==4) return 0;
    if (a + b < c) return 1;
    if (a + b > c) return 2;
    if (a * b == c) return 3;
    return 4;
}
```

Does this method ever return 3?

All of the following must be true (why?):
- !(c == 0)
- !(c == 4)
- !(a + b < c)
- !(a + b > c)
- a * b == c

Reasoning about program equivalence

```c
int add1(int a, int b) {
    return a + b;
}
```

```c
int add2(int a, int b) {
    return a * b;
}
```

Are these two methods semantically equivalent?
Are these two methods semantically equivalent?

Yes, for $a=0$ and $b=0$.

What have we actually proven here?

Our goal is to prove the absence of counter examples (i.e., the defined constraints are unsatisfiable)!

What's next?

- **Week 1:** Introduction & static vs. dynamic analysis
- **Week 2:** Abstract Interpretation
- **Week 3:** Abstract Interpretation
- **Week 4:** Testing
- **Week 5:** Delta Debugging
- **Week 6:** Slicing & Invariants
- **Week 7:** Program Repair
- **Week 8:** Empirical Software Engineering
- **Week 9:** ML for Software Engineering
- **Week 10:** Wrap up

HW 1

HW 2

In-class exercise

Project presentation