Spring 2017
Program Analysis and Verification

Introduction

Roman Manevich
Ben-Gurion University
Agenda

• What is program verification?

• Course overview and details

• What is program analysis?
What is program verification?
Code review

• Is this program correct? (take 2 min. to answer privately)

```java
// Negates the values of the vector in=(0|1)*.
void negate(int[] in, int[] out) {
    x = 0;
    while (x<in.length) {
        x = x+1;
        out[x] = in[1-x];
    }
    return out;
}
```

• If answer is no, can you offer a fix?
What does correct even mean?

• We need a specification

```java
// requires in!=null && out!=null && in.length==out.length
// requires foreach 0≤z<in.length : 0≤in[z]≤1
// ensures foreach 0≤z<in.length : out[z] = 1-in[z] && 0≤out[z]≤1
// Negates the values of the vector in=(0|1)*.
void negate(int[] in, int[] out) {
    x = 0;
    while (x<in.length) {
        x = x+1;
        out[x] = in[1-x];
    }
    return out;
}
```

• Does the program satisfy its specification?

Language intrinsic properties (avoids which runtime exceptions?)

Functional correctness
How do we know the spec is satisfied?

• Common practice: write a few tests

```java
// requires in!=null && out!=null && in.length==out.length
// requires foreach 0≤z<in.length : 0≤in[z]≤1
// ensures foreach 0≤z<in.length : out[z] = 1-in[z] &&
// 0≤out[z]≤1

// Negates the values of the vector in=(0|1)*.
void negate(int[] in, int[] out) {
    x = 0;
    while (x<in.length) {
        out[x] = 1-in[x];
        x = x+1;
    }
    return out;
}
```

• Can we actually prove correctness?

In general, no (Rice’s theorem)
Program verification

• A discipline for formally proving that a program (or a system) satisfies its specification
• Semi-automated
• Different tools exist
• We will use Dafny: a language and program verifier for functional correctness
  – Developed at Microsoft Research (RiSE group)
  – And now... a demo [proved]
method array_negate(a: array<int>, b: array<int>) returns ()
    requires a != null;
    requires b != null;
    requires a.Length == b.Length;
    requires (forall z :: (0 <= z < a.Length) ==> (a[z] == 0 | a[z] == 1));
    ensures (forall z :: (0 <= z < a.Length) ==> b[z] == 1 - a[z]);
    modifies b;
{
    var i := 0;
    while (i < a.Length)
    {
        b[i] := 1 - a[i];
        i := i + 1;
    }
}
method array_negate(a: array<int>, b: array<int>) returns ()
  requires a != null;
  requires b != null;
  requires a != b;
  requires a.Length == b.Length;
  requires (forall z :: (0 <= z < a.Length) ==> (a[z] == 0 | | a[z] == 1));
  ensures (forall z :: (0 <= z < a.Length) ==> b[z] == 1 - a[z]);
  modifies b;
{
  var i := 0;
  while (i < a.Length)
    invariant 0 <= i <= a.Length;
    invariant (forall z :: 0 <= z < i ==> b[z] == 1 - a[z]);
    decreases a.Length - i;
    { b[i] := 1 - a[i];
      i := i + 1;
    }
}
ANOTHER EXAMPLE
30GB Zunes all over the world fail en masse

December 31, 2008
What happens when days=366?

```
while (days > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  }
  } else {
    days -= 365;
    year += 1;
  }
}
```

Suggested solution: wait for tomorrow
while (days > 365) {
    if (IsLeapYear(year)) {
        if (days > 366) {
            days -= 366;
            year += 1;
        }
    } else {
        break;
    }
} else {
    break;
}
} else {
    days -= 365;
    year += 1;
}
Learning outcomes

• How to think of programs mathematically
• How to write basic specifications
• How to write annotations to help the verifier succeed
• The basic theory behind such tools
COURSE OVERVIEW
AND DETAILS
A little about me

• History
  – Studied B.Sc., M.Sc., Ph.D. at Tel-Aviv University
    • Research in program analysis with IBM and Microsoft
  – Post-doc in UCLA and in UT Austin
    • Research in program analysis and synthesis
  – Joined Ben-Gurion University on 2012
My research interests

• **Shape analysis**: inferring properties of programs manipulating dynamic data structures
  – Sequential and parallel
  – Automatically learn the shape of the heap

• **Synthesis**: automatically generate correct code from a very high-level language
  – Sequential and parallel
  – Advanced optimizations

• **Concurrent programming**

• Interested in doing a project or an M.Sc.? Come talk to me
What do you get in this course?

• Learn basic principles of program verification and program analysis
  – Understand jargon/papers

• Obtain some experience via assignments

• Put to practice what you learned in logic, automata, data structures, and programming
Course requirements

1. 50%: 2-3 theoretical assignments and programming assignments
   – Must obtain passing grade for each assignment

2. 50%: Home exam
   – Must obtain passing grade
Typesetting assignments

• LaTeX: THE way to produce scientific documents

• Recommended cloud-based editor
My obligation

• Prepare an interesting and accessible course
• Help you gain new knowledge and capabilities
• Answer questions in class and offer my help

• Course [web-page](#)
  – Announcements
  – Forum
  – Slides + lecture recordings
• Office hours: Monday 12:00-14:00
• E-mail: [romanm@cs.bgu.ac.il](mailto:romanm@cs.bgu.ac.il)
• Skype: rmanevich
How to succeed in this course

• Attend class regularly
  – Make sure you understand material **at class**
  – Ask right when you lost me, don’t delay
  – Solve worksheets
  – Engage by asking questions and raising ideas
• Missed a lecture – make up for it before next one
• Be on top of assignments
  – Submit on time
  – Don’t get stuck or give up on exercises – get help – ask me
  – Don’t start working on assignments the day before
• Be ethical

Joe (a day before assignment deadline):
“I don’t really understand what you want from me in this assignment, can you help me/extend the deadline”?  

22
Course policies

• Don’t be late, be inside room at 14:10, 15:10
• If you happen to be late (for a good reason) more than 15 minutes, wait for recess
• Basic mutual dignity
  – No eating, sleeping, reading newspapers...

• Follow announcements on course web-page
  – Course material page will also show schedule and any time/room changes
Tentative syllabus

Program Verification
- Operational semantics
- Hoare Logic
- Applying Hoare Logic
- Weakest Precondition Calculus
- Proving Termination
- Data structures
- Automated Verification

Program Analysis Basics
- From Hoare Logic to Static Analysis
- Control Flow Graphs
- Equation Systems
- Collecting Semantics
- Using Soot

Abstract Interpretation fundamentals
- Lattices
- Fixed-Points
- Chaotic Iteration
- Galois Connections
- Domain constructors
- Widening/Narrowing

Analysis Techniques
- Numerical Domains
- Pointer analysis
- Shape Analysis
- Interprocedural Analysis
What is program analysis?
“The algorithmic discovery of properties of a program by inspection of its source text\(^1\)”
-- Manna, Pnueli

A generalization of the dataflow analysis framework that you learned in the compiler construction course

1 Does not have to literally be the source text, just means w/o running it
Why study program analysis?

• Learn how to use abstraction to deal with intractable (usually undecidable) problems

• Understand how to systematically
  – Reason about correctness / find bugs (connections to cyber-security)
  – Design compiler optimizations

• Some techniques may be applied in other domains
  – Computational learning
  – Analysis of biological systems
method array_negate(a: array<int>, b: array<int>) returns ()
    requires a != null;
    requires b != null;
    requires a != b;
    requires a.Length == b.Length;
    requires (forall z :: (0 <= z < a.Length) ==> (a[z] == 0 | a[z] == 1));
    ensures (forall z :: (0 <= z < a.Length) ==> b[z] == 1 - a[z]);
    modifies b;
{
    var i := 0;
    while (i < a.Length)
        invariant 0 <= i <= a.Length;
        invariant (forall z :: 0 <= z < i ==> b[z] == 1 - a[z]);
        decreases a.Length - i;
        {
            b[i] := 1 - a[i];
            i := i + 1;
        }
}
Possible usage 2: compiler optimization

• How can we optimize this program?

```c
// Negates the values of the vector in=(0|1)*.
void negate(int[] in, int[] out) {
    int x = 0;
    while (x<in.length) {
        out[x] = 1-in[x];
        x = x+1;
    }
    return out;
}
```

The array length doesn’t change – access memory just once for the whole loop
How can we infer this optimization?

```java
// Negates the values of the vector in=(0|1)*.
void negate(int[] in, int[] out) {
    x = 0;
    int len = in.length;
    while (x<len) {
        out[x] = 1-in[x];
        x = x+1;
    }
    return out;
}
```
Implementing program analyses

• We will use Soot
  – A framework for analyzing and transforming Java and Android Applications

• Singly-linked lists analysis demo

• Tentative:

  CPAchecker
  The Configurable Software-Verification Platform
Learning outcomes

• The theory of *abstract interpretation*
• Learn about some existing analyses
• How to develop sound static analyses
• How to combine analyses
• Gain a little experience with Soot and Java
Math refresher

• Read Appendix A (pages 213-216) of *Semantics with Applications* book
Logical symbols

- **Vocabulary**: set of **predicate** and **function** symbols
  - $\text{domain}(p)$ is the **domain** of a predicate $p$
  - $\text{rank}(p)$ the number of arguments for predicate symbol $p$
  - $\text{domain}(f)$ is the **domain** of a function $f$
  - $\text{rank}(f)$ the number of arguments for function symbol $f$
  - A nullary function ($\text{rank}(f)=0$) called a **constant**
  - $\text{range}(f)$ is the **range** of a function $f$
  - $V^{\text{fun}}$ – the set of function symbols in $V$
  - $V^{\text{pred}}$ – the set of predicate symbols in $V$
Logical structures

• A logical structure $S$ over a vocabulary $V$ is a pair $(U, I)$
  – $U$ is a set of abstract individuals called the universe
  – Notation: $\text{vec}(U, k)$ all vectors of length $k$ from $U$
  – $I : \{\text{vec}(\text{domain}(p), \text{rank}(p)) \mid p \in V^{\text{pred}} \rightarrow \{0, 1\}\} \cup \{\text{vec}(\text{domain}(f), \text{rank}(f)) \mid p \in V^{\text{fun}} \rightarrow \text{range}(f)\}$ is an interpretation assignment function
Logical formulas

• Let \textit{vars} be a countable set of logical variables

• First-order logic \textit{terms} are defined as
  \[ T ::= v \mid f(T,\ldots,T) \]

• First-order logic \textit{sentences} are defined as
  \[ F ::= \text{QF} \mid \forall v. \ F \mid \exists v. \ F \]
  \[ \text{QF} ::= p(T,\ldots,T) \mid \neg F \mid F \land F \mid F \lor F \]
see you next time