Power of software

**Can you write any program I describe to you?**

Can you write:

A program HALTS?
INPUT: the source code of a method
OUTPUT: false if the method enters an infinite loop, true if it does not.
What’s HALTS?(method)?

```java
method() {
    print "hello, world";
}
```

What’s HALTS?(method)?

```java
method() {
    for (int x=0; x<5; x++)
        print "hello, world";
}
```

What’s HALTS?(method)?

```java
method() {
    for (int x=0; x<-1; x++)
        print "hello, world";
}
```

What’s HALTS?(method)?

```java
method() {
    while (true);
}
```
What’s HALTS?(method)?

```java
method() {
    int x = $785^{th}$ digit of π;
    if (x == 7)
        while(true);
}
```

What’s HALTS?(method)?

```java
method() {
    int x = 785$^{th}$ digit of π;
    int y = $x^{x^{x^{x^{x+1}}}}$;
    int z = $y^{th}$ digit of π;
    if (z == 0)
        while(true);
}
```

What’s HALTS?(method)?

```java
method() {
    int x = $785^{th}$ digit of π;
    int y = $x^{x^{x^{x^{x+1}}}}$;
    int[] z[] = the $y^{th}$ through $(x+y)^{th}$ digits of π;
    if (z ever repeats in π again)
        while(true);
}
```

How about the general case?

- Let’s count programs. How many programs are there?
Specifications

- And how many specifications are there?
  - let’s limit ourselves to simple specifications:
    - given a set of numbers, e.g., \( \{2, 4, 6\} \)
    - on input \( i \), return 1 if \( i \) is in the set, and 0 otherwise

set size -> number of specs

- Suppose I can only write 4 programs.
- I start with the smallest set specification:
  \( \{\} \)
- that’s 1 program. (return \textit{false} on all inputs)
- With 4 programs, I can do
  \( \{\}, \{1\}, \{2\}, \{1, 2\} \)

First 64 programs

- With 64 programs, how large can my specification sets get (if I am being compact)
  - (a) 64
  - (b) 32
  - (c) 8
  - (d) 6
  - (e) 2

\( \{\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{1, 5\}, \{1, 6\}, \{2, 3\} \ldots \{1, 2, 3\}, \{1, 3, 4\}, \ldots, \{1, 2, 3, 4, 5\} \ldots \)

Scalability Problem

- To cover subsets of a set of \( n \) numbers, I need \( 2^n \) programs.
- But I only have as many programs as there are natural numbers.
- That’s exponentially smaller than the number of specifications there are.
  
  Can’t do it for all subsets!
Can HALTS? exist?

• Imagine that you wrote HALTS?
• I will write a new program NALTS?:
  NALTS?(Method p) {
    if (HALTS?(p)==false) return 1;
    else while (true);
  }
  
  Key: run the program on itself
  What is the value of NALTS?(NALTS?)

What is the value of NALTS?(NALTS?)

• Two cases:
  1. If NALTS?(NALTS?) goes into an infinite loop, then
     HALTS?(NALTS?)==true, which means that
     NALTS? terminates.
     So case 1 is impossible.
  2. If NALTS?(NALTS?) does not go into an infinite
     loop, then HALTS?(NALTS?)==false, which
     means that NALTS? does not terminate.
     So case 2 is impossible.

Conclusion

• The program HALTS cannot exist!
• Many programs cannot exist!
• Learn more in CS 401 or CS 601

What is Software Engineering?

More than just writing code
The complete process of specifying, designing, developing, analyzing, deploying, and maintaining
a software system.
• Common Software Engineering tasks include:
  ○ Requirements engineering
  ○ Software architecture and design
  ○ Programming
  ○ Verification & Validation
  ○ Debugging
What is Software Engineering?

**More than just writing code**

The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  1. **Requirements engineering**
  2. Software architecture and design
  3. Programming
  4. Verification & Validation
  5. Debugging

**Requirements engineering: Phases**

1. Elicitation
2. Specification
3. Analysis
4. Management

[Diagram of development process]
1. The 'Three in a Row' game must use the model-view-controller architecture pattern.
   a. There must be a single game model.
   b. Once the new game model is created, that model can have its state changed.
   c. If the game model has a state change, then its game view must be updated.
   d. ...

2. ...

Specification: FSA

Once the new game model is created, that model can have its state changed.

gameModelNew  gameModelStateChange

1  2  3  2

gameModelStateChange
What is Software Engineering?

More than just writing code
The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Software architecture and design
  - Programming
  - Verification & Validation
  - Debugging

Software architecture vs. design

Architecture (what components are developed?)
- Considers the system as a whole:
  - High-level view of the overall system.
  - What components exist?
  - What type of storage, database, communication, etc?

Design (how are the components developed?)
- Considers individual components:
  - Data representation
  - Interfaces, Class hierarchies
  - ...

Software architecture: Pipe and Filter

Pipe and Filter: Compiler

- Multiple passes
  - Each operate on a complex IR
  - Lot of information passing
  - Very complex Rep Invariant
  - Code generation at the end

The architecture doesn’t specify the design or implementation details of the individual components (filters)
OO design principles

- Information hiding (and encapsulation)
- Open/closed principle
- Inheritance, the diamond of death, Liskov substitution principle
- Composition/aggregation over inheritance to prevent the diamond of death

What is a design pattern?

- Addresses a recurring, common design problem.
- Provides a generalizable solution.
- Provides a common terminology.

Pros

- Improves communication and documentation.
- "Toolbox" for novice developers.

Cons

- Risk of over-engineering.
- Potential impact on system performance.

More than just a name for common sense and best practices

What is Software Engineering?

More than just writing code

The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Software architecture and design
  - Programming
  - Verification & Validation
  - Debugging
Commonly used best programming practices

- Version control
- Documentation (e.g., README, javadoc, internal comments)
- Coding best practices such as modularity (e.g., packages, classes), encapsulation (e.g., field/method modifiers), constants/enumerations, type checking inputs (e.g., defensive programming)
- Unit test suites (e.g., JUnit)

Achieving usability

- User testing and field studies
  - having users use the product and gathering data
- Evaluations and reviews by UI experts
- Prototyping
  - Paper prototyping
  - Code prototyping
- Good UI design focuses on the user, not on the developer, not on the system environment

What is Software Engineering?

More than just writing code
The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Software architecture and design
  - Programming
  - Verification & Validation
  - Debugging
Ways to get your code right

- **Verification & Validation**
  - Purpose is to uncover problems and increase confidence
  - Combination of manual techniques (e.g., code reviews, informal reasoning) and automated techniques (e.g., testing, model checkers, theorem provers)

- **Debugging**
  - Purpose is finding out why a program is not functioning as intended
  - Pinpoint location + cause of problem

- **Defensive programming**
  - Programming with validation and debugging in mind

Software testing

- Blackbox vs. whitebox testing
- Unit testing (vs. integration vs. system testing)
- Test adequacy
  - Structural code coverage
  - Statement coverage
  - Decision coverage
  - Condition coverage
  - Mutation analysis

Model Checker

- [e.g., Java PathFinder or JPF]

![Diagram showing the Model Checker process]

Formal reasoning

- Forward reasoning is more intuitive for most people
  - Helps understand what will happen (simulates the code)
  - Introduces facts that may be irrelevant to goal
  - Set of current facts may get large
  - Takes longer to realize that the task is hopeless

- Backward reasoning is usually more helpful
  - Helps you understand what should happen
  - Given a specific goal, indicates how to achieve it
  - Given an error, gives a test case that exposes it
Theorem Prover
[e.g. Z3]

Program (Constraints)  Question (Constraints)

Z3 theorem prover [SAT constraint solver]

SAT (+ positive example) [Variable assignments]
-OR- UNSAT -OR- UNK

Debugging: Overview

- **Goal:** Finding out why a program is not functioning as intended. Pinpoint location + cause of bug.

- **Approach:** Design **experiments** to gain information about bug
  - Fairly easy in a program with good modularity, representation hiding, specs, unit tests etc.
  - Much harder and more painstaking with a poor design, e.g., with rampant representation exposure

Debugging:
Commonly used techniques

- Println debugging (e.g., echos in script.sh)
- UI debuggers
- Fault localization algorithms (e.g., used by APR)
- Delta debugging (e.g., in-class exercise 3 & homework 2)

Automated program repair (APR) tool

Buggy program Test suite, including passing tests and failing tests

APR tool Patched program (now passes ALL tests)
Coming up

- Final projects deliverables:
  - Presentations: Tue April 28, in class (over Zoom)
  - Documentation submission: Tue April 28, 11:55 PM (a little before midnight)

Project Final Presentations

- Think of this as a science fair
- Each team will talk for 5 minutes about their project: demo, poster, or slides
  - Discuss the design, test/experimental plan, and results
  - Practice what you want to say
- Chance to hear about other projects too