Preventing Bugs in Programs

Programs have bugs

- What problems does this cause?

How can we get rid of bugs?

- Testing
- Verifying – proving the code is right
- Using safer languages,
  - eg “buffer overflow” not possible in Java, vs C
- Automated checking in the compiler
  - eg type checking in Java, vs Python
  - eg assertions and annotations
- How good could we get?

Perfect checking isn’t possible

- Fundamental results of Theoretical Computer Science!
  - Can prove that some things we would like to do are not possible

```
/* lunar lander in game: smooth descent */
double height = 1000;   // start 1000 meters up
double speed = 200;
while (height > 0){
    moveDown(speed);          //  move down for 1 second
    speed = speed * 0.75;     // slow down a bit
}
UI.println(“Landed”);
```

Will it stop?

Will it halt?

-

```
public void simulate(int rabbits, int foxes){
    while (rabbits + foxes > 0) {
        if (rabbits >  foxes) { rabbits = rabbits – foxes/2;  }
        else                  { foxes = foxes – rabbits/10; }
        if (foxes < 100) { foxes++; }
        if ( rabbits < 10)  { rabbits = rabbits*2; }
    }
}
```

Could you always detect if a program will halt?

- No! proof by contradiction:
- Suppose you gave me a perfect loop checker:

```
checker(program, input)
    → "OK"     if  program(input) halts
    → "Loops" if program(input) loops for ever.
```

- Then I could make a new program out of your program:

```
tricky (program)
    if ( checker (program, program) == "OK" )  { while (true) {}  }
    if ( checker (program, program) == "Loops")  { return; }
```


tricky tricky

```
tricky (program)
    if ( checker (program, program) == "OK" )  { while (true) {}  }
    if ( checker (program, program) == "Loops")  { return; }
```

What does tricky(tricky) do?

if checker(tricky, tricky) == "OK" then → loops forever

ie, if tricky(tricky) halts, then → loops forever

if checker(tricky, tricky) == "Loops" then → halts

ie, if tricky(tricky) loops forever then → halts

Contradiction!

if tricky(tricky) halts, then tricky(tricky) doesn't halt

if tricky(tricky) doesn't halt, then tricky(tricky) halts

The Halting problem

- Contradictions aren't possible.
  Therefore the assumption must be false.

➔ You can't make a checker that always tells whether an arbitrary program will halt.

- The Halting Problem is just one example of non-computable (undecidable) problems.
- Gödel's Theorems: You cannot make a theorem prover that could prove all the true statements in some logical system.

What does it mean for preventing bugs

- You can make a useful compiler/program checker that can identify lots of problems, but you cannot make a perfect one.
  Don't waste time trying to make it perfect, just make it better
- You can write theorem provers that will help you prove that a program is correct, but you can't make a perfect one.
- Designing languages, compilers, automated provers, is important for improving software, but there are fundamental limitations.