Completion of the course

Second progress report

Submit **before** Tuesday March 24.

Presentations

Monday March 30, 9:00-10:30 and Wednesday April 1, 9:00-10:30 on Zoom. (Project need not be completed by this time. Just present what you have done so far and what you still plan to do.)

Final exam

"Take home" exam on Wednesday April 8, 19:00-21:00. The questions will be available online at 19:00. Students have two hours to complete the exam and submit their answers electronically.

Report and code

Submit **before** Tuesday April 21.

Undecidability

EECS 4315

www.eecs.yorku.ca/course/4315/

Undecidability

Question

Why do we study undecidability in this course?

Undecidability

Question

Why do we study undecidability in this course?

Answer

It is important to know that there are problems that cannot be solved.

Undecidability

Question

Why do we study undecidability in this course?

Answer

It is important to know that there are problems that cannot be solved.

The course EECS 2001 has the following learning outcomes:

- Demonstrate limits of computing by proving that a problem is not solvable (within a particular model of computation).
- Show how one problem can be reduced to another.

Undecidability

Question

Why do we study undecidability in this course?

Answer

It is important to know that there are problems that cannot be solved.

The course EECS 2001 has the following learning outcomes:

- Demonstrate limits of computing by proving that a problem is not solvable (within a particular model of computation).
- Show how one problem can be reduced to another.

This course used to be required for Software Engineering students.

Undecidability

1. Formulate a problem and show that it cannot be solved.
2. Reduce some model checking problems to this problem: this shows that those model checking problems cannot be solved either.

Terminates?

Assume the following method.

```
/**
 * Checks whether the given Java app terminates for
 * the given input.
 *
 * @param code file name of the Java app
 * @param input file name of the input for the Java app
 * @return true if the given Java app terminates for
 * the given input, false otherwise.
 */
public static boolean isTerminating(String code,
                                String input)
```

Input

Convention

All Java app read all their input from a file that is given as the first command line argument.

Terminates?

```java
import java.util.Scanner;

public class Example {
  public static void main(String[] args) {
    Scanner input = new Scanner(args[0]);
    int value = input.nextInt();
    if (value == 0) {
      while (true);
    } else {
      // do nothing
    }
    input.close();
  }
}
```

Terminates?

Question

Assume that the file `example` starts with `"0 "`. What should the call `isTerminating("Example.java", "example")` return?

Terminates?

Question

Assume that the file `example` starts with `"0 "`. What should the call `isTerminating("Example.java", "example")` return?

Answer

`false`

Terminates?

Question

Assume that the file `example` starts with `"1 "`. What should the call `isTerminating("Example.java", "example")` return?

Terminates?

Question

Assume that the file `example` starts with `"1 "`. What should the call `isTerminating("Example.java", "example")` return?

Answer

`true`

Terminates?

Question

Assume that the file `example` starts with `"public "`. What should the call `isTerminating("Example.java", "example")` return?

Terminates?

Question

Assume that the file `example` starts with `"public "`. What should the call `isTerminating("Example.java", "example")` return?

Answer

`true`

Mystery

```
public class Mystery {
  public static void main(String[] args) {
    if (isTerminating(args[0], args[0])) {
      while (true);
    } else {
      // do nothing
    }
  }
}
```

Terminates?

`isTerminating("Mystery.java", "Mystery.java")` returns `true`

Terminates?

`isTerminating("Mystery.java", "Mystery.java")` returns `true`

if and only if (specification of isTerminating method)

Terminates?

`isTerminating("Mystery.java", "Mystery.java")` returns `true`

if and only if (specification of isTerminating method)

Java app `Mystery` terminates for input in file `Mystery.java`

Terminates?

`isTerminating("Mystery.java", "Mystery.java")` returns `true`

if and only if (specification of isTerminating method)

Java app `Mystery` terminates for input in file `Mystery.java`

if and only if (convention about input)

Terminates?

`isTerminating("Mystery.java", "Mystery.java")` returns `true`

if and only if (specification of isTerminating method)

Java app `Mystery` terminates for input in file `Mystery.java`

if and only if (convention about input)

Java app `Mystery` terminates with command line argument `Mystery.java`

Terminates?

`isTerminating("Mystery.java", "Mystery.java")` returns `true`

if and only if (specification of isTerminating method)

Java app `Mystery` terminates for input in file `Mystery.java`

if and only if (convention about input)

Java app `Mystery` terminates with command line argument `Mystery.java`

if and only if (definition of Mystery class)

Terminates?

`isTerminating("Mystery.java", "Mystery.java")` returns `true`

if and only if (specification of isTerminating method)

Java app `Mystery` terminates for input in file `Mystery.java`

if and only if (convention about input)

Java app `Mystery` terminates with command line argument `Mystery.java`

if and only if (definition of Mystery class)

`isTerminating("Mystery.java", "Mystery.java")` returns `false`

Terminates?

`isTerminating("Mystery.java", "Mystery.java")` returns `true`

if and only if (specification of isTerminating method)

Java app `Mystery` terminates for input in file `Mystery.java`

if and only if (convention about input)

Java app `Mystery` terminates with command line argument `Mystery.java`

if and only if (definition of Mystery class)

`isTerminating("Mystery.java", "Mystery.java")` returns `false`

This cannot be true.

Terminates?

Question

Did we make a mistake in the derivation on the previous slide?

Terminates?

Question

Did we make a mistake in the derivation on the previous slide?

Answer

No.

Terminates?

Question

Did we make a mistake in the derivation on the previous slide?

Answer

No.

Question

Did we make any assumptions? (If we start from an assumption that does not hold, then we can derive anything.)

Terminates?

Question

Did we make a mistake in the derivation on the previous slide?

Answer

No.

Question

Did we make any assumptions? (If we start from an assumption that does not hold, then we can derive anything.)

Answer

Yes, we assumed the existence of the method `isTerminating`.

Terminates?

Conclusion

The method `isTerminating` cannot be implemented as specified, that is, given a Java app and its input we cannot in general determine whether that app terminates for the input.

Terminates?

Conclusion

The method `isTerminating` cannot be implemented as specified, that is, given a Java app and its input we cannot in general determine whether that app terminates for the input.

Determining whether a given Java app terminates for a given input is an example of a decision problem.

Terminates?

Conclusion

The method `isTerminating` cannot be implemented as specified, that is, given a Java app and its input we cannot in general determine whether that app terminates for the input.

Determining whether a given Java app terminates for a given input is an example of a decision problem.

Definition

A decision problem is a problem that can be posed as a yes-no question of the input values.

The halting problem

Problem

Determine, given an arbitrary computer program and an input for that program, whether the program will finish running, or continue to run forever.

The halting problem

Problem

Determine, given an arbitrary computer program and an input for that program, whether the program will finish running, or continue to run forever.

The halting problem is a decision problem.

The halting problem

Problem

Determine, given an arbitrary computer program and an input for that program, whether the program will finish running, or continue to run forever.

The halting problem is a decision problem.

Theorem

The halting problem cannot be solved, that is, it is undecidable.

The halting problem

Problem

Determine, given an arbitrary computer program and an input for that program, whether the program will finish running, or continue to run forever.

The halting problem is a decision problem.

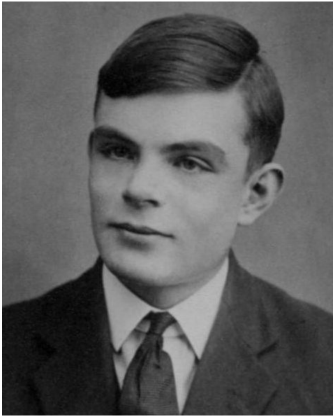
Theorem

The halting problem cannot be solved, that is, it is undecidable.

This theorem was proved by Alan Turing in 1936.

Alan Turing (1912-1954)

- English mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist
- Formalized the concepts of algorithm and computation
- During the Second World War, worked on cracking messages of the Enigma machine



Source: unknown

Concurrency

EECS 4315

www.eecs.yorku.ca/course/4315/

Increment (Java code)

```
public class Increment extends Thread {
  private int i;

  public Increment(int i) {
    this.i = i;
  }

  public void run() {
    this.i++;
    System.out.println(this.i);
  }
}
```

Increment (Java bytecode)

Java bytecode of the `run` method:

```
aload_0
dup
getfield
iconst_1
iadd
putfield
getstatic
aload_0
getfield
invokevirtual
```

Increment (Java bytecode)

Java bytecode of part of the `run` method:

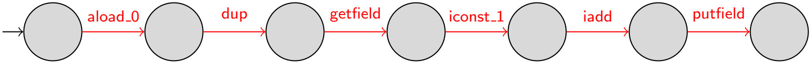
```
aload_0
dup
getfield
iconst_1
iadd
putfield
```

Question

Draw the state-transition diagram.

State-transition diagram

```
aload_0
dup
getfield
iconst_1
iadd
putfield
```



Two Increments

```
public class Main {
  public static void main(String[] args) {
    (new Increment(1)).start();
    (new Increment(2)).start();
  }
}
```

Two Increments

```
public class Main {
  public static void main(String[] args) {
    (new Increment(1)).start();
    (new Increment(2)).start();
  }
}
```

Question

What output does this app produce?

Two Increments

```
public class Main {
  public static void main(String[] args) {
    (new Increment(1)).start();
    (new Increment(2)).start();
  }
}
```

Question

What output does this app produce?

Answer

23 or 32.

Two Increments (Java bytecode)

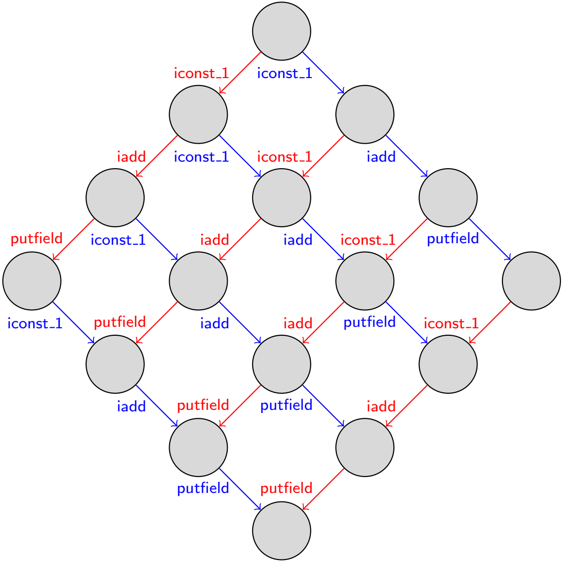
Java bytecode of parts of the run methods:

```
iconst_1      iconst_1
iadd          iadd
putfield      putfield
```

Question

Draw the state-transition diagram.

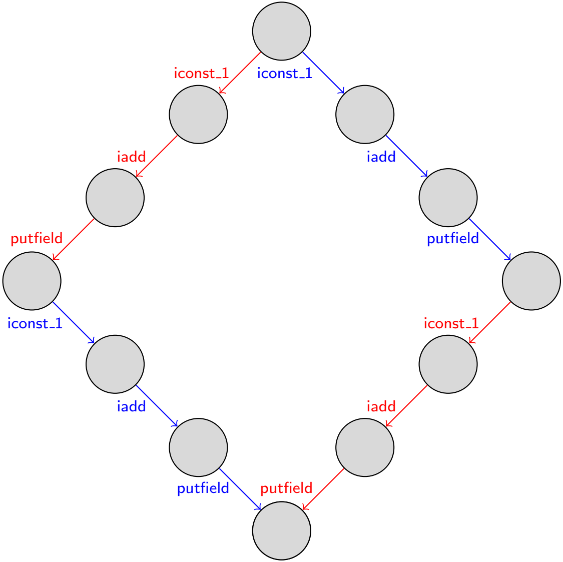
State-transition diagram



Interleavings

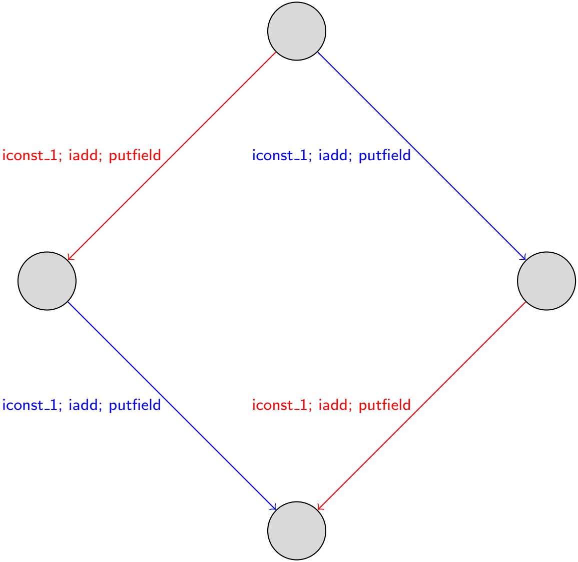
The bytecode instructions of each thread manipulate only an attribute that is not shared with the other thread. As a consequence, the bytecode instructions of one thread do not impact the bytecode instructions of the other thread. Hence, not all interleavings need to be considered.

State-transition diagram

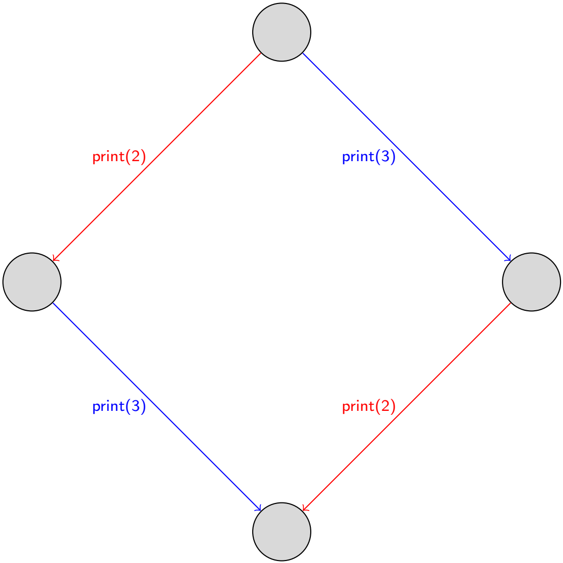


State-transition diagram

Combine transitions



State-transition diagram



Interleavings

For the three bytecode instructions of the `run` method

All interleavings: 16 states and 24 transitions
Essential interleavings: 4 states and 4 transitions

Interleavings

For the three bytecode instructions of the `run` method

All interleavings: 16 states and 24 transitions
Essential interleavings: 4 states and 4 transitions

For all ten bytecode instructions of the `run` method

All interleavings: 121 states and 220 transitions
Essential interleavings: 4 states and 4 transitions

Combining bytecode instructions

- We combine the bytecode instructions when there is only one thread.
- We combine the bytecode instructions that do not impact the other threads.

Combining bytecode instructions

Problem

Given all the (byte)code of a multi-threaded app, determine for a specific bytecode instruction of a specific thread whether it impacts other threads.

Combining bytecode instructions

Problem

Given all the (byte)code of a multi-threaded app, determine for a specific bytecode instruction of a specific thread whether it impacts other threads.

Question

Give an algorithm that solves the problem.

Combining bytecode instructions

Problem

Given all the (byte)code of a multi-threaded app, determine for a specific bytecode instruction of a specific thread whether it impacts other threads.

Question

Give an algorithm that solves the problem.

Answer

Impossible!

Proving impossibility

Question

Which other problems cannot be solved?

Proving impossibility

Question

Which other problems cannot be solved?

Answer

The halting problem: given code and input for that code, determine whether the code terminates for the input.

Proving impossibility

Problem

Given all the (byte)code of a multi-threaded app, determine for a specific bytecode instruction of a specific thread whether it impacts other threads.

Question

Explain (informally) why the problem cannot be solved.

Writer

```java
public class Writer extends Thread {
  public static boolean shared = false;

  public void run() {
    Writer.shared = true;
  }
}
```

Writer

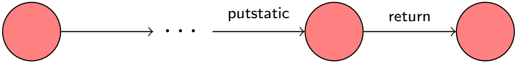
```
public class Reader extends Thread {
  public void run() {
    this.code();
    if (Writer.shared) {
      ...
    }
  }

  public void code() {
    ...
  }
}
```

Main

```java
public class Main {
  public static void main(String[] args) {
    (new Reader()).start();
    (new Writer()).start();
  }
}
```

Writer

Transitions of the Writer thread:



Assume that the code method does not use the attribute Writer.shared. Then the bytecode instruction putstatic of the Writer thread impacts the Reader thread if and only if the method call to code terminates.

Combining bytecode instructions

General idea

Combine those bytecode instructions for which we can prove that they do not impact other threads.

Combining bytecode instructions

General idea

Combine those bytecode instructions for which we can prove that they do not impact other threads.

The idea of combining consecutive transitions labelled with invisible (outside the current thread) actions into a single transition is due to Patrice Godefroid.

Combining bytecode instructions

General idea

Combine those bytecode instructions for which we can prove that they do not impact other threads.

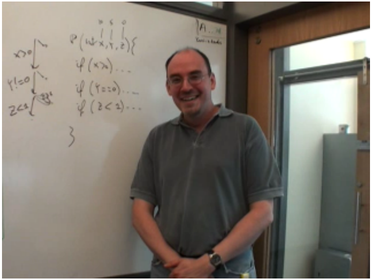
The idea of combining consecutive transitions labelled with invisible (outside the current thread) actions into a single transition is due to Patrice Godefroid.

Examples of invisible actions

- Reading or writing an attribute that can be proved to be not shared.
- Reading or writing a local variable.
- ...

Patrice Godefroid

- Ph.D. degree in Computer Science from the University of Liege, Belgium.
- Worked at Bell Laboratories.
- Currently at Microsoft Research.



Source: Patrice Godefroid

Course evaluation

Please complete the course evaluation at https://courseevaluations.yorku.ca/.

Normally, I bring cupcakes if more than 90% of the students completes the course evaluation. Unfortunately, this year I cannot provide cupcakes.

Instead, everyone will get a 1% bonus mark if we surpass that 90% participation rate by Wednesday April 1. This is not a joke and it is of course not as good as a cupcake.