Decision Procedures: Motivation

Ruzica Piskac
Yale University
Why Software Verification?

Maiden flight of the Ariane 5 rocket on the 4th of June 1996

- The reason for the explosion was a software error
- Financial loss: $500,000,000 (including indirect costs: $2,000,000,000)
Boeing could not assemble and integrate the fly-by-wire system until it solved problems with the databus and the flight management software. Solving these problems took more than a year longer than Boeing anticipated. In April, 1995, the FAA certified the 777 as safe.

Total development cost: $3 billion
Software integration and validation cost: one third of total
Examples of Software Errors

Radio Therapy Machine
software error
→ 6 people overdosed

Year 2010 Bug
30 million debit and credit cards have been rendered unreadable by the software bug

software in modern cars
>100K LOC
2006: error in pump control software
→ 128000 vehicles recalled

link
Financial Impact of Software Errors

Recent research at Cambridge University (2013, link) showed that the global cost of software bugs is around 312 billion of dollars annually.

Goal: to increase software reliability
How to obtain Software Reliability?

- Testing, testing, testing, ...
  - Many software errors are detected this way
  - Does not provide any correctness guarantee
  - “Murphy’s Law”
- Verification
  - Provides a formal mathematical proof that a program is correct w.r.t. a certain property
  - A formally verified program will work correctly for every given input
  - Verification is algorithmically very hard task (problem is in general undecidable)
A Mathematical Proof of Program Correctness?

Can you verify my program?

Which property are you interested in?

```java
public void add (Object x)
{
    Node e = new Node();
    e.data = x;
    e.next = root;
    root = e;
    size = size + 1;
}
```
Example Questions in Verification

- Will the program crash?
- Does it compute the correct result?
- Does it leak private information?
- How long does it take to run?
- How much power does it consume?
- Will it turn off automated cruise control?
A Mathematical Proof of Program Correctness?

I just want to be sure that no element is lost in the list – if I insert an element, it is really there.

```java
public void add (Object x)
{
    Node e = new Node();
    e.data = x;
    e.next = root;
    root = e;
    size = size + 1;
}
```
A Mathematical Proof of Program Correctness?

```java
//: L = data[root.next*]

public void add (Object x)
{
    Node e = new Node();
    e.data = x;
    e.next = root;
    root = e;
    size = size + 1;
}
```

Let L be a set (a multiset) of all elements stored in the list ...
A Mathematical Proof of Program Correctness?

```java
public void add (Object x) {
    Node e = new Node();
    e.data = x;
    e.next = root;
    root = e;
    size = size + 1;
}
```

Annotations:

- `//: L = data[root.next*]`
- `//: invariant: size = card L`
- `//: ensures L = old L + {x}`
Annotatons

- Written by a programmer or a software analyst
- Added to the original program code to express properties that allow reasoning about the programs

**Examples:**
- **Preconditions:**
  - Describe properties of an input
- **Postconditions:**
  - Describe what the program is supposed to do
- **Invariants:**
  - Describe properties that have to hold in every program point
Decision Procedures for Collections

```java
public void add (Object x)
//: ensures L = old L + {x}
{
    Node e = new Node();
e.data = x;
e.next = root;
root = e;
size = size + 1;
}
```

Prove that the following formula always holds:

\[ \forall X. \forall L. |X| = 1 \rightarrow |L \cup X| = |L| + 1 \]
Verification Conditions

- Mathematical formulas derived based on:
  - Code
  - Annotations
- If a verification condition always holds (valid), then the code is correct w.r.t. the given property
- It does not depend on the input variables
- If a verification condition does not hold, we should be able to detect an error in the code
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
    ??
    return y
}
Verification Condition: Example

//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
    val y = x - 2
    return y
}

Verification condition:

∀ x. ∀ y. x > 0 ∧ y = x - 2 → y > 0
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
    val y = x - 2
    return y
}

∀ x. ∀ y. x > 0 ∧ y = x - 2 → y > 0

Preconditions
Verification Condition: Example

```java
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
  val y = x - 2
  return y
}
```

Verification condition:

\[ \forall x. \forall y. x > 0 \land y = x - 2 \rightarrow y > 0 \]
Verification Condition: Example

```java
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
    val y = x - 2
    return y
}
```

Verification condition:

\[ \forall x. \forall y. x > 0 \land y = x - 2 \rightarrow y > 0 \]
Verification Condition: Example

```java
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
    val y = x - 2
    return y
}
```

Verification condition:

\[ \forall x. \forall y. x > 0 \land y = x - 2 \rightarrow y > 0 \]

Formula does not hold for input \( x = 1 \)
Automation of Verification

- Windows XP has approximately 45 millions lines of source code
  \[\approx 300.000 \text{ DIN A4 papers} \]
  \[\approx 12m \text{ high paper stack}\]

Verification should be automated!!!
Software Verification

Prove formulas automatically!
A decision procedure is an algorithm which answers whether the input formula is satisfiable or not

- formula $x \leq y$ is satisfiable for $x=0$, $y=1$
- formula $x \leq y \land x+1 > y+1$ is unsatisfiable
Combining Various Logics

```java
//: L = data[root.next*]
//: invariant: size = card L
public void add (Object x)
//: ensures L = old L + {x}
{
    Node e = new Node();
    e.data = x;
    e.next = root;
    root = e;
    size = size + 1;
}
```
Combining Various Logics

```java
//: L = data[root.next*]
//: invariant: size = card L
public void add (Object x)
//: ensures L = old L + {x}
{
    Node e = new Node();
e.data = x;
e.next = root;
root = e;
size = size + 1;
}
```

Verification condition:

\[ \neg \text{next}_o^*(\text{root}_o,n) \land x \not\in \{\text{data}_o(v) \mid \text{next}_o^*(\text{root}_o,v)\} \land \text{next} = \text{next}_o[n:=\text{root}_o] \land \text{data} = \text{data}_o[n:=x] \rightarrow \\
\{|\text{data}(v) . \text{next}^*(n,v)\}| = \\
\{|\text{data}_o(v) . \text{next}_o^*(\text{root}_o,v)\}| + 1\]
Software synthesis = a technique for automatically generating code given a specification

Why?
- ease software development
- increase programmer productivity
- fewer bugs

Challenges
- synthesis is often a computationally hard task
- new algorithms are needed
Software Synthesis

val bigSet = ....

val (setA, setB) = choose(((a: Set, b: Set) ) =>
    ( a.size == b.size && a union b == bigSet && a intersect b == empty))

Code
assert (bigSet.size % 2 == 0)
val n = bigSet.size/2
val setA = take(n, bigSet)
val setB = bigSet -- setA
Course Textbooks

Aaron R. Bradley, Zohar Manna: *The calculus of computation - decision procedures with applications to verification*. Springer 2007