Software Engineering: Where are we? And where do we go from here?

V22.0474-001 Software Engineering
Lecture 24

Clark Barrett
New York University
4/20/2005
Between 1985 and 1987, at least 6 accidental radiation overdoses were administered.

All the victims were injured, and 3 of them later died.
In 1996, an unmanned Ariane 5 rocket launched just seconds after its lift off.

, alue of roce et and car o 5 million
In August, 3, the largest lac out in our country's history occurred.

Projected cost to renew 3 or do it alone: $1.1 trillion.
Caused by Software Bugs!

- Each of the overdoses from the 5 herac 5 was the result of a user in the controlin software.

- The Ariane 5 explosion was the result of an unsafe floating oint to integer conversion in the rocket's software system.

- A software user caused an alarm system failure at 61rs in Akron, Ohio. An early response to those alarms would likely have prevented the lack out.
Top Oxymoron from OxymoronList.com: Microsoft Works
- When we build a ride, we don’t expect it to crumble and have to rebuild it twice a week. This is software so much less reliable than rides or buildings.

- You have the knowledge and skills you need to create quality software.

- What have you learned in this class that can help?

- What tools and techniques do you think future software engineers will use to create more reliable systems?
• Formal software verification has been the evolution of computer science for many decades. AB Billates.

• Formal verification techniques can be used to prove that a piece of software is correct.

• Here are still many challenges to make in this radical, but there are also some success stories.
What is Formal Verification?
Model Checking
Theorem Proving
Systems and Tools
- reate a mathematical model of the system
  - An inaccurate model can introduce or mas u s.
  - 6 ortunately, this can often e done automaticall.

- ecif formall what the ro erties of the s stem should e

- rove that the model has the desired ro erties
  - uch etter than an testin method
  - overs all ossi le cases
  - 5 his is the hard art

- 5 here are a variet of tools and techni ues
odel hec in
- 5 icali relies on low level Boolean logic
- roof is full automatic
- 9 oes not scale to lar e stem

5 heorem rovin
- 5 icali uses more e ressive lo ic hi her order lo ic
- roof is manuall directed
- nlimited scala ilit

Advanced technique uses com ine elements of oth
- What is Formal Verification?
- Model Checking
- Theorem Proving
- Systems and Tools
5. A formal model is a graph in which each vertex represents a state of the system, and each edge represents a transition from one state to another.

Consider this simple system:

```c
int x, y;
x = 0;
y = 0;
while (x < 3) {
    x++;
    y = y + x;
}
```

5. The states of this system are all possible states of the variables and .

6. Fortunately, we can restrict our attention to the edge states.
A formal model is a graph in which each vertex represents a state of the program, and each edge represents a transition from one state to another.

```
int x, y;
x = 0;
y = 0;
while (x < 3) {
    x++;
    y = y + x;
}
```
7 can check a root verification that it is a root in every reachable state. If the root is e, then there is a unique.

```
int x, y;
x = 0;
y = 0;
while (x < 3) {
    x++;
    y = y + x;
}
```
7 e can check a root verification that it is root in every reachable state. If the root is \( e \), then there is a \( u \).

```
int x, y;
x = 0;
y = 0;
while (x < 3) {
    x++;
y = y + x;
}
```
7 e can check a root verification that it is true in every reachable state. If the root is e, then there is a u.

\[
\begin{align*}
    x &= 0 \\
    y &= \frac{x(x+1)}{2}
\end{align*}
\]

Initial State

Final State

int x, y;
x = 0;
y = 0;
while (x < 3) {
    x++;
y = y + x;
}
7 e can check a ro ert verif in that it is re in ever
reacha le state. f the ro ert is e, then there is a u.

\[ x \]
\[ x \]
\[ y = \frac{x(x+1)}{2} \]

\[ = \rightarrow \]
\[ y = \frac{x(x+1)}{2} \]

\[ 2,0,0 \]
\[ 2,1,1 \]
\[ 3,1,0 \]
\[ 3,2,1 \]
\[ 3,3,3 \]

**Initial State**

\[ \]
\[ int x, y; \]
\[ x = 0; \]
\[ y = 0; \]
\[ while (x < 3) \{ \]
\[ \quad x++; \]
\[ \quad y = y + x; \]
\[ \} \]

\[ 2,3,6 \]
\[ 2,2,3 \]
\[ 3,3,3 \]

**Final State**
In practice, models of real ro rams would have too man states to modelchech.

5 here are a num er of techni ues which can hel
  • A straction
  • 9 ecom osition
  • m olic model chec in

Itimatel, model chec in alone cannot rove ro erties of lar e ro rams.
• What is Formal Verification?
• Model Checking
• Theorem Proving
• Systems and Tools
Theorem rovin relies on human in eunuit and som olie maniulation to rove that a ro ram satisfies some ro ert.

Icall, rovin a sin le ro ert a out a ro ram will reuire rovin man other ro erties as well.

ne a roach is to annotate the ro ram with theorems to e roved also called vr or er o, and then rove that each theorem reall does hold.
The re

- Consider a slight modified version of our simple ro ram from efore this time there are man more reachable states.

- u ose we wish to rove that at the end of the ro ram, \( x(x+1) \)

- e can annotate the end of the ro ram with this roert and wor ac wards from there.

```c
int x, y;
x = 0;
y = 0;
while (x < 30) {
    x++;
    y = y + x;
}
```
To show this ert, we must look at the two ession locations in the ram.

For these two locations, it will e sufficient to rove that the ram either doesn’t end or that the ert holds.

With a bit of insights, we can see that these two formulas are more complicated than necessary. We can e a formula less than it with a formula which lies it.

```java
int x, y;
x = 0;
y = 0;
while (x < 30) {
    x++;
    y = y + x;
    x < 30; = x + x;
}
x(x )
```
Theorem

- To show this property, we must look at the two possible previous locations in the memory.

- For these two locations, it will be sufficient to prove that the memory either doesn’t end or that the property holds.

- With a bit of insight, we can see that these two formulas are more complicated than necessary. We can reuse a formula recursively in it with a formula which implies it.

```plaintext
int x, y;
int x, y;
x = 0;
y = 0;
x <= 0 \lor y = x(x + 1)
while (x < 30) {
    x++;
    y = y + x;
    = x(x + 1)
}
x(x)
```
The reason that will guarantee the reen assertion after executing \( y = y + x \) is:

- 7 hat is the condition that will guarantee the reen assertion after executing \( y = y + x \).
- To find out, we imagine trying to represent the reen condition using imed variables to represent the value after \( y = y + x \) and unim ed variables for the value before.

\[
( \rightarrow x \land x = x \rightarrow = x (x + ) \\
( \rightarrow x x(x+ ) \\
( \rightarrow x(x ) \\
\]

```c
int x, y;
x = 0;
y = 0;
x=0
while (x < 30) {
    x++;
y = y + x;
    x(x+)
}
x(x )
```
Now we must find an assertion which is implied by the loop end condition and the re loo condition, and which implies the reen condition after executing $x++$.

To do this, we first must strengthen the re loo condition.

```c
int x, y;
x = 0;
y = 0;
x = 0 \land y = 0
while (x < 30) {
    x++;
    y = y + x;
}
```

$x(x)$
Now we must find an assertion which is implied by the loop end condition and the loop condition, and which implies the loop condition after executing $x++$.

5. To do this, we first must strengthen the loop condition.

7. We finish with a set of assertions, each of which can be proven to follow from the annotations at all possible previous points in the program.

```
int x, y;
x = 0;
y = 0;
x = 0  \land y = 0
while (x < 30) {
  x++; = x(x+)
  y = y + x;
}
```

```
x(x)
```
- Notice that the final set of conditions 
\( o e \ o e p e \) on the number of loop iterations.

- In fact, this same roof can be used regardless of what the loop condition is.

- His is one advantage theorem rovin has over model check in.

```c
int x, y;
int x = 0;
y = 0;
x = 0 \land \ y = 0
while (x < 30) {
    y = y + x;
    x++;
    x = x + x;
}
x = x
```
The re

- ach roof from the assertion efore a statement to the assertion after the statement is called a ver o o o.

- rification conditions can e roved usin an o e eore prover.

- owever, comin u with the assertions usuall re uires human uidance and can e quite challen in.
- What is Formal Verification?
- Model Checking
- Theorem Proving
- Systems and Tools
odel check er for finite state stems
Based on e tremel efficient data structures for re resentin Boolean logic
, er successful for hardware

odel check er for arallel stems
imited to small state aces
Theorem

- Some interactive 5 theoremrovers
  - ,
  - A
  - sa elle

- Some automated domain specific theoremrovers
  - im lif
  - ,
* stochastic search enter at
  * formerl om a , formerl 9

* 5 heorem rovin a roach for sim le ro erties in ava

* ser annotates code with e ected invariants

* nvariants are verified usin automated theorem rover im lif
lever combination of model checking and automated theorem proving

- An abstract model is created, in which all conditions are replaced with Boolean variables
- The result is a Boolean model that is model checked
- If the model checks in fails, the potential error ath is checked in the original model using an automated theorem prover

- Successfull used to find uses in 7 indows drivers.
- Reduced the frequency of Blue screens of death
• 6ormal software, verification is startin to ecome ractical
till lots of wor to e done

• ow can it ma e ou a etter ro rammer?
  ﬀ 9 ocument our code with the ro erties and invariants
  that ou thin should e true
  ﬀ 7 hen ou modif code, convince ourself that ou are
  not rea in an invariants
  ﬀ earn more a out formal verification

• oefull, someda software will e as safe and relia le as
  the other ojects uilt en ineers