Can we compose software with building blocks and verify it automatically?

Francis Leung

Fall, 2010
Outline

• Why we work on these two problems
• Challenges and our approach
• What we are working on
Two Problems in Software Engineering

• Almost all programmers in the industry make their living by changing code and in testing
  – Can we add features without changing code?

• Software testing often takes half the time and resources of a software project
  – Can we verify software by assertions automatically?
What It Takes to Add Features to SW Today

POTS as An Example

- The programmer develops **Call Waiting** by changing the code of **POTS**
- **Call Forwarding** is developed by changing the code of **Call Waiting** and **POTS**
- **Retry** is developed by changing the code of **Call Forwarding**, **Call Waiting**, and **POTS**, and so on
- The programmer is not just implementing one feature. He must thoroughly understand change the code and test many features
- Features are not reusable without each other
  - They are “entangled”
How does entangled code look like?
Why Are We Still Writing Entangled Code?

The programmer is not experienced
The programmer is not well trained
The programmer is not given enough time and resources.
Because they use top-down
No, because they use bottom-up

It is due to object oriented programming
It is because they do not follow a plan (agile programming)

It is due to waterfall

“Whoever blames will later be blamed. The times they are a changing” – I. Bob Dylan

The programmers cannot help it!

Why do we keep on changing code?

try {
   if (agent.isAvailable())
      call.transferTo(agent);
   ....
}
catch (AgentCrashed) {
   // Transfer call to another agent
   ....
}
catch (NetworkCongestion) {
   // Do other things
   ....
}
catch (NetworkDown) {
   // Do yet other things
}
When a function throws a new exception, programs that call the function may have to be changed.

Programs that transitively (indirectly) call the function may also have to be changed.

Existing programming languages offer very little help to programmers.

Many popular programs crash often because exceptions are not handled.
Features and feature packages written in FLX are reusable

- Features are written based on a model instead of the code of other features
- Features and feature packages are integrated in a feature package
- One can integrate different combinations of features and feature packages into different feature packages
Challenges

• Automatic assertion based software verification
  – Grand Challenge (Hoare 2003)
  – Holy Grail of Computer Science (Gates 2005)

• Changing code is as hard
  – Assertion verification is to determine whether an assertion is true at some point in the code.
  – Finding out where to make changes is to determine where in the code an assertion becomes true
Getting to the Holy Grail

• We can automate assertion based verification of hardware circuits
  – Finite state machine
  – Boolean logic (Advances in SAT solvers in the late 90’s)

• But with Software
  – State variables may not be bounded (e.g. integer)
    • FLX compiles an application into a FSM
  – Must reason on predicate logic whose variables have complex data structures (e.g. Is the linked list empty?)
    • A first order solver for assertions written in Java has been implemented
What are we working on?

• The FLX compiler.

• The FLX satisfiability solver for first order formulas

• A call center developed using FLX

• A set of self healing features