Developing Verified Programs with Boogie

Nadia Polikarpova

Software Verification

October 15, 2014
Overview

What is Boogie?

The Language: how to express your intention?
  Imperative constructs
  Specification constructs

The Tool: how to get it to verify?
  Debugging techniques
  Boogaloo to the rescue
Overview

What is Boogie?

The Language
- Imperative constructs
- Specification constructs

The Tool
- Debugging techniques
- Boogaloo to the rescue
“Auto-active” verification

all interaction at the program level

- Specification
- Program
- Annotations

Verifier

Logical Formula

Reasoning Engine
Verifying imperative programs

Language A
Verifier A

Language B
Verifier B

Language C
Verifier C

Control flow & state
... 

Control flow & state, built-in types, framing,... 

Control flow & state
... 

Logical Formula

Reasoning Engine

reuse
Intermediate Verification Language

Language A
Verifier A

Language B
Verifier B

Language C
Verifier C

High-level constructs, built-in types and operations, framing, ...

Invariant inference, ...

IVL Program

Control flow & state

Logical Formula I
Reasoning Engine I

Logical Formula II
Reasoning Engine II

Logical Formula III
Reasoning Engine III
The Boogie IVL

Simple yet expressive
procedures
first-order logic
integer arithmetic

Great for teaching verification!
skills transferable to other auto-active tools

Getting started with Boogie

Try online [rise4fun.com/Boogie]
Download [boogie.codeplex.com]
User manual [Leino: This is Boogie 2]

Hello, world?
Overview

What is Boogie?

The Language
  Imperative constructs
  Specification constructs

The Tool
  Debugging techniques
  Boogaloo to the rescue
Types

Basic types: \textbf{bool}, \textbf{int}, \textbf{real}

User-defined: \textbf{type} Name \( t_1, \ldots, t_n \);

Maps: \(<t_1, \ldots, t_n>[\text{dom}_1, \ldots, \text{dom}_n]\)\text{range}

Synonyms: \textbf{type} Name \( t_1, \ldots, t_n = \text{type};\)

\begin{align*}
\text{definition} & \quad \text{usage} \\
\text{Definition} & \quad \text{Usage} \\
\text{Definition} & \quad \text{Usage} \\
\text{Definition} & \quad \text{Usage} \\
\text{Definition} & \quad \text{Usage} \\
\end{align*}
Imperative constructs

Regular procedural programming language

[Absolute Value & Fibonacci]

... and non-determinism

great to simplify and over-approximate behavior

```havoc x; // assign an arbitrary value to x

if (*) { // choose one of the branches non-deterministically
    statements
} else {
    statements
}

while (*) { // loop some number of iterations
    statements
}
```
Specification statements: `assert`

`assert` e: executions in which e evaluates to `false` at this point are bad

expressions in Boogie are pure, no procedure calls

Uses

explaining semantics of other specification constructs
encoding requirements embedded in the source language

```
assert lo <= i && i < hi; // bounds check
result := array[i];
```

```
assert this != null; // 0-0 void target check
call M(this);
```

debugging verification (see later)

[Absolute Value]
Specification statements: assume

**assume** e: executions in which e evaluates to **false** at this point are impossible

```
havoc x; assume x*x == 169; // assign such that
assume true; // skip    assume false; // this branch is dead
```

Uses

explaining semantics of other specification constructs
encoding properties guaranteed by the source language
```
havoc Heap; assume NoDangling(Heap); // managed language
```

debugging verification (see later)

Assumptions are dangerous! [Absolute Value]
Loop invariants

```plaintext
before_statements;
while (c)  
    invariant inv;
{
    body;
}
after_statements;
```

```plaintext
before_statements;
assert inv;

havoc all_vars;
assume inv && c;
body;
assert inv;

havoc all_vars;
assume inv && !c;
after_statements;
```

The only thing the verifier know about a loop simple invariants can be inferred

[Fibonacci]
Procedure contracts

procedure \( P(\text{ins}) \) returns (\( \text{outs} \))

free requires \( \text{pre}' \);
requires \( \text{pre} \);
modifies \( \text{vars} \); // global
ensures \( \text{post} \);
free ensures \( \text{post}' \);
{
body;
}

call \( \text{outs} := P(\text{ins}); \)

\[ \text{assume } \text{pre } \&\& \text{pre}'; \]
\[ \text{body}; \]
\[ \text{assert } \text{post}; \]

\[ \text{assert } \text{pre}; \]
\[ \text{havoc } \text{outs}, \text{vars}; \]
\[ \text{assume } \text{post } \&\& \text{post}'; \]

The only thing the verifier knows about a call this is called modular verification

[Abs and Fibonacci]
Enhancing specifications

How do we express more complex specifications?
   e.g. ComputeFib actually computes Fibonacci numbers

Uninterpreted functions

```
function fib(n: int): int;
```

Define their meaning using axioms

```
axiom fib(0) == 0 && fib(1) == 1;
axiom (forall n: int :: n >= 2 ==> fib(n) == fib(n-2) + fib(n-1));
```

[Fibonacci]
Overview

What is Boogie?
The Language
  Imperative constructs
  Specification constructs
The Tool
  Debugging techniques
  Boogaloo to the rescue
What went wrong?

Speciﬁcation
Program
Annotations

Boogie
Debugging techniques

Proceed in small steps [Swap]
  use **assert** statements to figure out what Boogie knows

Divide and conquer the paths
  use **assume** statements to focus on a subset of executions

Prove a lemma [Non-negative Fibonacci]
  write ghost code to help Boogie reason

Look at a concrete failing test case [Array Max]
  Boogaloo to the rescue!
Getting started with Boogaloo

Try online [cloudstudio.ethz.ch/comcom/#Boogaloo]
Download [bitbucket.org/nadiapolikarpova/boogaloo]
Features

Print directives

```
assume {: print "hello, world", x + y } true;
```

[Array Max, print the loop counter]

Bound on loop iterations

```
--loop-max=N -l=N
```

N = 1000 by default

[Array Max, comment out loop counter increment]
Conclusions

Boogie is an Intermediate Verification Language (IVL)
IVLs help develop verifiers

The Boogie language consists of:
imperative constructs ≈ Pascal
specification constructs (assert, assume, requires, ensures, invariant)
math-like part (functions + first-order axioms)

There are several techniques to debug a failed verification attempt