Software Verification : Introduction

Ranjit Jhala, UC San Diego

April 4, 2013
What is Algorithmic Verification?

Algorithms, Techniques and Tools to ensure that

- Programs
- Don’t Have
- Bugs

(What does that *mean*? Stay tuned...)
Most people here know what it means so more concretely . . .

1. Survey of *basics* of software verification [me]
2. Building up to *refinement type-based* verification [me]
3. Culminating with *recent topics* in verification. [you]
Goals

1. Train students in state of the art, preparation for research
2. Write a monograph synthesizing different lines of work
1. *Use* tools for different languages to see ideas in practice
2. *Develop* ideas in a *single, unified, simplified* (aka “toy”) PL
Plan

- **Part 1** Deductive Verification
- **Part 2** Type Systems
- **Part 3** Refinement Types
- **Part 4** Abstract Interpretation
- **Part 5** Heap and Dynamic Languages
- **Part 6** Project Talks
Plan: 1 Deductive Verification

- Logics & Decision Procedures
- Floyd-Hoare Logic
- Verification Conditions
- Symbolic Execution
Plan: 2 Type Systems

- Hindley-Milner
- Subtyping
- Bidirectional Type Checking
Plan: 3 Refinement Types

- Combining Types & Logic
- Reasoning about State
- Abstract Refinements
Plan: 4 Abstract Interpretation

- Horn Clause Constraints
- Galois Connections
- Predicate Abstraction/Liquid Types
- Interpolation
Plan: 5 Heap & Dynamic Languages

- Linear Types
- Separation Logic
- Hoare Type Theory
- Dependent JavaScript
Plan: 6 Project Talks

Link to README
Requirements & Evaluation

1. Scribe
2. Program
3. Present
Requirements: 1. Scribe

- Lectures will be black-board (not slides)
- You sign up for one lecture (Online URL)
- For that lecture, take notes
- Write up notes in LaTeX using provided template
Requirements: 2. Program

About **three** “programming” assignments

- *Implement* some of algorithms (in Haskell)
- *Use* some verification tools (miscellaneous)
Requirements: 3. Present

You will present one **40 minute talk**

1. Select 1-3 (related) papers from **reading list**
2. Select presentation date (**~** last 5 lectures)
3. Prepare slides, get vetted by me **1 week in advance**
4. Present lecture

- Can add other paper if I’m ok with it.
Questions
 Lets Begin . . .

- Logics & Decision Procedures
- Easily enough to teach (many) courses
- We will scratch the surface just to give a feel
Logic is the Calculus of Computation

May seem *abstract* now . . .

. . . why are we talking about these wierd symbols?!

Much/all of program analysis can be boiled down to logic

**Language** for reasoning about programs
Logics & Decision Procedures

We will look very closely at the following

1. Propositional Logic
2. Theory of Equality
3. Theory of Uninterpreted Functions
4. Theory of Difference-Bounded Arithmetic

(Why? Representative & have “efficient” decision procedures)
We will look very closely at the following

1. **Propositional Logic**
2. Theory of *Equality*
3. Theory of *Uninterpreted Functions*
4. Theory of *Difference-Bounded Arithmetic*

(Why? Representative & have “efficient” decision procedures)
Propositional Logic

A logic is a **language**

- *Syntax* of formulas (predicates, propositions...) in the logic
- *Semantics* of when are formulas *satisfied* or *valid*
Propositional Logic: Syntax

\begin{verbatim}
data Symbol -- a set of symbols

data Pred = PV Symbol
          | Not Pred
          | Pred `And` Pred
          | Pred `Or` Pred
\end{verbatim}

Predicates are made of

- Propositional symbols ("boolean variables")
- Combined with And, Or and Not
Propositional Logic: Syntax

```haskell
data Symbol -- a set of symbols

data Pred = PV Symbol
  | Not Pred
  | Pred `And` Pred
  | Pred `Or` Pred

Can build in other operators Implies, Iff, Xor etc.

p `imp` q = (Not p `Or` q)
p `iff` q = (p `And` q) `Or` (Not p `And` Not q)
p `xor` q = (p `And` Not q) `Or` (Not p `And` q)
```
Predicate is a **constraint**. For example,

\[ x_1 \text{ `xor` } x_2 \text{ `xor` } x_3 \]

States “only an **odd number** of the variables can be true”

- When is such a constraint **satisfiable** or **valid**?
Propositional Logic: Semantics

Let Values = True, False, ... be a universe of possible “meanings”

An **assignment** is a map setting value of each Symbol as True or False

data Asgn = Symbol -> Value

**Semantics/Evaluation Procedure**

Defines when an assignment s makes a formula p true.

eval :: Asgn -> Pred -> Bool

eval s (PV x) = s x  -- assignment s sets x to 'True'
eval s (Not p) = not (sat s p)  -- p is NOT satisfied
eval s (p 'And' q) = sat s p && sat s q  -- both of p, q are satisfied
eval s (p 'Or' q) = sat s p || sat s q  -- one of p, q are satisfied
Propositional Logic: Decision Problem

Decision Problem: Satisfaction
Does eval s p return True for some assignment s ?

Decision Problem: Validity
Does eval s p return True for all assignments s ?
Satisfaction: A Naive Decision Procedure

Does \textit{eval} \ s \ p return True for \textbf{some} assignment \textit{s}?

\textit{Enumerate} all assignments and run \textit{eval} on each!

\[
isSat :: \text{Pred} \rightarrow \text{Bool}
\]

\[
isSat \ p = \text{exists} (\ s \rightarrow \text{eval} \ s \ p) \ ss
\]

where

\[
\text{ss} = \text{asgns} \ \&\& \ \text{removeDuplicates} \ \&\& \ \text{vars} \ p
\]

\[
\text{exists} \ f \ [] = \text{False}
\]

\[
\text{exists} \ f \ (x:xs) = f \ x || \text{exists} \ f \ xs
\]
Satisfaction: A Naive Decision Procedure

Does \( \text{eval } s \ p \) return True for some assignment \( s \)?

Enumerate all assignments and run eval on each!

Enumerating all Assignments

\[
\text{asgns} \quad :: \quad \text{[PVar]} \to \text{[Asgn]}
\]
\[
\text{asgns } [] = [\ \lambda x \to \text{False}]
\]
\[
\text{asgns } (x:xs) = [\ 
\text{ext } s \ x \ t \mid s \leftarrow \text{asgns } xs, t \leftarrow \text{[True, False]}]
\]
\[
\text{ext } s \ x \ t = \lambda y \to \text{if } y == x \text{ then } t \text{ else } s \ x
\]

\[
\text{vars} \quad :: \quad \text{Pred} \to \text{[PVar]}
\]
\[
\text{vars } (\text{PV } x) = [x]
\]
\[
\text{vars } (\text{Not } p) = \text{vars } p
\]
\[
\text{vars } (p \ '\text{And' } q) = \text{vars } p ++ \text{vars } q
\]
\[
\text{vars } (p \ '\text{Or' } q) = \text{vars } p ++ \text{vars } q
\]

Obviously Inefficient... (guaranteed) exponential in
We will look very closely at the following

1. Propositional Logic
2. Propositional Logic + Theories
   - Equality
   - Uninterpreted Functions
   - Difference-Bounded Arithmetic

(Why? Representative & have “efficient” decision procedures)
Propositional Logic + Theory

Layer theories on top of basic propositional logic

Expressions

A new kind of term

data Expr

Theory

A Theory is Described by

1. Extend universe of Values
2. A set of Operator
   - Syntax: data Expr = ... | Op [Expr]
   - Semantics: eval :: Op -> [Value] -> Value
3. A set of Relation (i.e. [Expr] -> Pred)
   - Syntax: data Pred = ... | Symbol <=> (Rel [Expr])
   - Semantics: eval :: Rel -> [Value] -> Bool
Propositional Logic + Theory

Layer theories on top of basic propositional logic

Semantics

Extend eval semantics for Operator and Relation

$$\text{eval } s \ (\text{op } es) \ = \ \text{eval } \text{op} \ [\text{eval } s \ e \mid e \leftarrow es]$$

$$\text{eval } s \ (x \leftrightarrow r \ es) \ = \ \text{eval } r \ [\text{eval } s \ e \mid e \leftarrow es]$$

→

Satisfaction / Validity

- **Sat** Does eval \( s \ p \) return True for some assignment \( s \)?
- **Valid** Does eval \( s \ p \) return True for all assignments \( s \)?
Let's make things concrete!
Logics & Decision Procedures

We will look very closely at the following

1. Propositional Logic
2. Propositional Logic + Theories
   - Equality
   - Uninterpreted Functions
   - Difference-Bounded Arithmetic

(Why? Representative & have “efficient” decision procedures)
Propositional Logic + Theory of Equality

1. Values = ... + Integer
2. Operator none
3. Relation
   ▶ Syntax : a Eq b or a Ne b
   ▶ Semantics

\[
eval \ Eq \ [n, m] = (n == m) \\
eval \ Ne \ [n, m] = not \ (n == m)
\]

Example

\[
(x1 \ 'And' \ x2 \ 'And' \ x3) \\
'And' \ (x1 \ <=> \ a \ 'Eq' \ b) \\
'And' \ (x2 \ <=> \ b \ 'Eq' \ c) \\
'And' \ (x3 \ <=> \ a \ 'Ne' \ c)
\]
 Propositional Logic + Theory of Equality

Example

\[(x_1 \ 'And' \ x_2 \ 'And' \ x_3)\]

\[\ 'And' \ (x_1 \ <=> \ a \ 'Eq' \ b)\]

\[\ 'And' \ (x_2 \ <=> \ b \ 'Eq' \ c)\]

\[\ 'And' \ (x_3 \ <=> \ a \ 'Ne' \ c)\]

Decision Procedures?

- Sat Does eval s p return True for some assignment s ?

Can we enumerate over all assignments? [No]
We will look very closely at the following

1. Propositional Logic
2. Propositional Logic + Theories
   - Equality
   - Uninterpreted Functions
   - Difference-Bounded Arithmetic

(Why? Representative & have “efficient” decision procedures)
Propositional Logic + Theory of Equality + Uninterpreted Functions

1. Values: ... + functions \([\text{Value}] \rightarrow \text{Value}\)
2. Operator: \(\text{App} (\text{apply} \: \text{App} \: [f,a,b] \text{ or just } f(a,b))\)
3. Relation: Eq and Ne (from before)
4. Extended eval

\[
eval \: s \: (\text{App} \: (e : [e_1 \ldots e_n])) = (\eval \: s \: e) \: (\eval \: s \: e_1 \ldots \eval \: s \: e_n)
\]

Example

\[
(x_1 \text{'And'} \: x_2 \text{'And'} \: x_3)
\]
\[
\text{'And'} \: (x_1 \leftrightarrow a \text{'Eq'} \: g(g(g(a))))
\]
\[
\text{'And'} \: (x_2 \leftrightarrow a \text{'Eq'} \: g(g(g(g(g(a))))))
\]
\[
\text{'And'} \: (x_3 \leftrightarrow a \text{'Ne'} \: g(a))
\]

Decision Procedures?

- **Sat** Does \(\eval \: s \: p\) return True for **some** assignment \(s\)?
Logics & Decision Procedures

We will look very closely at the following

1. Propositional Logic
2. Propositional Logic + Theories
   ▶ Equality
   ▶ Uninterpreted Functions
   ▶ Difference-Bounded Arithmetic

(Why? Representative & have “efficient” decision procedures)
Propositional Logic + Difference Bounded Arithmetic

1. Values: ... + Integer
2. Operator: None
3. Relation: \( DBn(x, y) \) (or, \( x - y \leq n \))
4. Extended eval

\[
eval \ s \ DB (e1, e2, n) = (eval \ s \ e1) - (eval \ s \ e2) \leq n
\]

Example

\[
(x1 \ 'And' \ x2 \ 'And' \ x3) \\
'And' \ (x1 \ <=> \ a - b \leq 5 ) \\
'And' \ (x2 \ <=> \ b - c \leq 10 ) \\
'And' \ (x3 \ <=> \ c - a \leq -20 )
\]

Decision Procedures?

- **Sat** Does eval \( s \ p \) return True for some assignment \( s \) ?
- Can we enumerate over all assignments? [Hell, no!]
  - How are we possibly supposed to even call functional
Next Time: Decision Procedures for SAT/SMT