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...)

Topics

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

Goals

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

Logics & Decision Procedures

- **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)

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)

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

```
data Symbol -- a set of symbols

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

Predicates are made of

- Propositional symbols ("boolean variables")
- Combined with `And`, `Or` and `Not`

Propositional Logic: Syntax

```
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)
```

Propositional Logic: Semantics

Predicate is a **constraint**. For example,

```
x1 `xor` x2 `xor` x3
```

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 s
eval s (Not p)      = not (sat s p)          -- p is NOT satis
eval s (p `And` q)  = sat s p && sat s q     -- both of p , q
eval s (p `Or`  q)  = sat s p || sat s q    -- one of  p , q a
```

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 `eval s p` return `True` for **some** assignment `s` ?

Enumerate all assignments and run `eval` on each!

```
isSat    :: Pred -> Bool

isSat p = exists (\s -> eval s p) ss
  where
    ss  = asgns $ removeDuplicates $ vars p

exists f []     = False
exists f (x:xs) = f x || exists f xs
```

Satisfaction: A Naive Decision Procedure

Does `eval s p` return `True` for **some** assignment `s` ?

Enumerate all assignments and run `eval` on each!

Enumerating all Assignments

```
asgns               :: [PVar] -> [Asgn]
asgns []            = [\x -> False]
asgns (x:xs)        = [ext s x t | s <- asgns xs, t <- [True,

ext s x t           = \y -> if y == x then t else s x

vars                :: Pred -> [PVar]
vars (PV x)         = [x]
vars (Not p)        = vars p
vars (p `And` q)    = vars p ++ vars q
vars (p `Or`  q)    = vars p ++ vars q
```

Obviously Inefficent. . . *(guaranteed) exponential in*

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

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`

```
eval s (op es)       = eval op [eval s e | e <- es]
eval s (x <=> r es) = eval r  [eval s e | e <- 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` ?

Lets 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

```
       (x1 `And` x2 `And` x3)
`And` (x1 <=> a `Eq` b)
`And` (x2 <=> b `Eq` c)
`And` (x3 <=> a `Ne` c)
```

Decision Procedures?

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

Can we *enumerate* over all assignments? [No]

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 + Uninterpreted Functions

1. `Values` : `...  + functions [Value] -> Value`
2. `Operator` : `App` (*apply* `App  [f,a,b]` or just `f(a,b)`)
3. `Relation` : Eq and Ne (from before)
4. Extended `eval`

```
eval s (App (e : [e1...en])) = (eval s e) (eval s e1 ... ev
```

Example

```
        (x1 `And` x2 `And` x3             )
`And` (x1 <=> a `Eq` g(g(g(a)))       )
`And` (x2 <=> a `Eq` g(g(g(g(g(a))))))
`And` (x3 <=> a `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 <= n`)
4. Extended `eval`

```
eval s (DB (e1, e2, n)) = (eval s e1) - (eval s e2) <= n
```

Example

```
      (x1 `And` x2 `And` x3)
`And` (x1 <=> a - b <= 5   )
`And` (x2 <=> b - c <= 10  )
`And` (x3 <=> c - a <= -20 )
```

Decision Procedures ?

- **Sat** Does `eval s p` return `True` for **some** assignment `s` ?
- Can we *enumerate* over all assignments? [Hell, no!]

Next Time: Decision Procedures for SAT/SMT