Synthesis, Analysis & Verification Topics Index

Diggory Hardy

April 17, 2012

Abstract

will the program crash, does it terminate, is the result correct, how long will it take to run, how much power will it consume, will it interfere with other jobs, will it leak private information This is a summary of notes available at `http://lara.epfl.ch/w/sav12:top`.

Contents

1 Introduction

1.1 Verification Condition Generation

The first step in proving program correctness is generating verification conditions. Example in slides (lecture 3, p3).

Section 1.2 covers representation of code and predicates.

Providing a specification

Intro slides (lecture 1, p33): add preconditions `require( ... )` on inputs and postcondition `ensuring( ... )` on output.

Loop invariants: Intro slides (lecture 1, p38): a condition which holds when initially encountered, which is preserved by the loop, and is strong enough to prove the post-condition.

1.2 Proving VCs

Once you've got verification conditions, you can attempt to prove them or find counter-examples. Section logic covers the logic systems needed to reason about VCs, while section abstrinterp looks at how abstractions can simplify code sufficiently to reason about non-trivial functions.

2 Code representations

2.1 Guarded Command Language

slides (lecture 3, p17):

- `assume(F)` — F is required to hold in any execution (assume any exceptions didn't happen)
- `s1 ; s2` — do s_1 then s_2
- `s1 [] s2` — do s_1 or s_2 arbitrarily (drunk if)
- `s*` — execute s any number of times
- `havoc(x)` — after execution, x may have any value while other variables remain unchanged

Guarded commands can be mapped to relations: slides (lecture 3, 18).

$x = E$ can be written as `havox(x); assume(x==E)` (slide 25).

Computing from a program: slides (lecture 4, p3)

Map `while(C){s1}; s2` to `( assume(C); s1 )* ; assume(!C)`

.

2.2 Relations

Programs are not always deterministic (in theory, they allow randomness): slides (lecture 3, p15). We use relations: $(x, x') \in r$ if and only if the operation specified by r can map state x to x'. Relations are not automatically functions: for some initial state x there can be zero, one or several results x'.

Identity relation: The identity relation on set S is $\Delta_S = \{(s, s) | s \in S\}$.

Computing: pages 2 onwards, slide set 4, week 2.

Normal Form Theorem: slides (lecture 4, p13)

2.2.1 Hoare Logic

Basics: Hoare Logic Basics, week 2. Hoare triples are some $\{P\}r\{Q\}$ where P is a precondition, r a relation and Q a postcondition. Usually P, Q are subsets of S, the set of all states.

$\{P\}r\{Q\}$ means $\forall s, s' \in S.(s \in P \wedge (s, s') \in r \rightarrow s' \in Q$

One can calculate some P or Q satisfying the triple: the **weakest precondition** or **strongest postcondition**.

$$wp(r, Q) = \{s \in S | \forall s' \in S.(s, s') \in r \rightarrow s' \in Q\}$$
$$sp(P, r) = \{s' | \exists s.s \in P \wedge (s, s') \in r\}$$

Obtaining verification conditions for Hoare triples: Compositional VCG, weeks 2-3(at end).

Manipulations (strengthening, weakening, some stuff on loops and composition): Syntactic rules for Hoare logic, week 3.

2.2.2 Operations on relations

From Compositional VCG, weeks 2-3:

Assignment, assume, havoc.

Union, sequential composition :

$$\{(x, y) | f(x, y)\} \circ \{(y, z) | g(y, z)\} = \{(x, z) | \exists y.f(x, y) \wedge g(y, z)\}$$

More: Forward VCG, week 3, Backward VCG, week 3.

2.2.3 Invariants and loops

Introduction to invariants on loops for Hoare triples: Monday, week 4.

2.3 Control Flow Graphs

Transition systems are essentially relations between states: Monday, week 4 (also contains a note about nested loops). CFGs are graphs of states and transitions.

3 Abstract Domains

3.1 Introduction

The basic idea of abstract interpretation is to consider the program as a control flow graph on sets of states. From Monday, week 4:

- slides
- *Sets of states at each program point*
- *Range analysis*

3.2 Lattices

Introduced Tuesday week 4 (slides).

3.2.1 Partial order

Partial order ($\leq$): must, for all x, y, z, satisfy

- $x \leq y$ (reflexivity)
- $x \leq y \wedge y \leq x \rightarrow x = y$ (antisymmetry)
- $x \leq y \wedge y \leq x \rightarrow x \leq z$ (transitivity)

Upper bound, *maximal element*, *greatest element*, *least upper bound* (lub, $\sqcup$), etc. and *monotonic functions* are defined on the same page.

3.2.2 Lattices

Defn: a **lattice** is a partial order in which every two-element set has a least upper bound and greatest lower bound. Definition and corollaries.

Lattices are used to approximate states.

Products of lattices can themselves be lattices: Monday week 5.

3.2.3 Fixed points

Given a function $f : A \rightarrow A$ on some domain A, $x \in A$ is a fixed point of f iff $f(x) = x$. See wiki.

Tarski's fixed point theorem: week 4 More definitions:

- $Post = \{x | G(x) \sqsubseteq x\}$
- $Pre = \{x | x \sqsubseteq G(x)\}$
- $Fix = \{x | G(x) = x\}$

Then (Tarski's thm): Let $(A, \sqsubseteq)$ be a complete lattice and $G : A \rightarrow A$ be a monotonic function. Then $\bigsqcap Post$ is the least element of Fix and $\bigsqcup Pre$ is the greatest element of Fiv.

3.2.4 Galois Connection

From wiki: a Galois connection is defined by two monotonic functions $\alpha : C \rightarrow A$ and $\gamma : A \rightarrow C$ where $(C, \leq)$ and $(A, \sqsubseteq)$ are partial orders and for all $a \in A$ and $c \in C$,

$$\alpha(c) \sqsubseteq a \Leftrightarrow c \leq \gamma(a)$$

Usage terminology: wiki.

3.3 Transition systems

Transition systems are given by a set of initial states $Init \subseteq S$ and a relation $r \subseteq S^2$. The set of reachable states is then $sp(Init, r^*)$. Introduction, with notes on fixed points: Monday, week 5.

3.3.1 Collecting semantics

Collecting semantics give us, for every program point/vertex in CFG a set of possible variable states (i.e. a mapping $V \rightarrow 2^S$ for program points V and variable states S). Introduction: *Transition systems and collecting semantics*, Tuesday week 5. More on this: example, additional notes.

From Monday week 6, example using ranges as abstract domain.

3.3.2 Abstract reachability

Abstract reachability tells us, given a set of predicates, which predicates hold at each node of the CFG. Hossein's slides (Friday, week 5).

4 Logic

4.1 Presburger Arithmetic

Introduced slides (lecture 2, p3).

Propositional formulas: expressions involving the constants $true$, $false$, propositional variables $p, q, ...$ and logical operators $\wedge \vee \neg \rightarrow \leftrightarrow$. $p \rightarrow q$ is equivalent to $\neg p \vee q$.

First order logic: extension of propositional logic with equality $=$, predicate symbols $P, Q, ...$, function symbols $f, g, ...$, variables $x, y, ...$ in some domain D, for-all and exists quantifiers $\forall$ $\exists$. First order logic formulas are the closure of the set:

$$\{\neg P, P \wedge Q, P \vee Q, P \rightarrow Q, P \leftrightarrow Q, \forall x.P, \exists x.P | P, Q \text{ are first-order formulas}\}$$

Formulas are **valid** if for all interpretations of unqualified symbols they are true, **satisfiable** if for some interpretation they are true, and **unsatisfiable** if not satisfiable.

4.1.1 Quantifier Elimination

This was covered in the Friday, week 3, and allows rewriting Presburgher arithmetic formulas without existential quantifiers.

4.1.2 More

Solving (exercise, hints at efficient code): Monday, week 4

5 Loop Invariant Inference

???

predicate abstraction
abstract interpretation and data-flow analysis
pointer analysis, typestate

Index