CSC2125 – Advanced Topics in Software Engineering: Program Analysis and Understanding
Fall 2006

About this Class

- Topic: Analyzing and understanding software
- Three main focus areas:
  - Static analysis
    - Automatic reasoning about source code
  - Formal systems and notations
    - Vocabulary for talking about programs
  - Programming language features
    - Affects programs and how we reason about them

2

Readings

- Nielson, Nielson, Hankin. Principles of Program Analysis, 2005, Springer.
- Supplemental readings from classical papers and from recent advances

3

Preparation

- A course in compilers would be helpful
- A course in model-checking would be most helpful

4

Expectations

- Periodic written assignments (not graded)
  - Short problem sets
  - This is how you will learn things
    - Much more effective than listening to a lecture
- Course participation (discussion of written assignments and course material)
- Presentation of part of course material
- Presentation of one application

5

What this course is about?

20 Ideas and Applications in Program Analysis in 40 Minutes

Abstract Interpretation

- Rice's Theorem: Any non-trivial property of programs is undecidable
  - Uh-oh! We can't do anything. So much for this course...
- Need to make some kind of approximation
  - Abstract the behavior of the program
  - ...and then analyze the abstraction
- Seminal papers: Cousot and Cousot, 1977, 1979

7

Example

- e ::= n | e + e

$$\alpha(n) = \begin{cases} - & n < 0 \\ 0 & n = 0 \\ + & n > 0 \end{cases}$$

+	-	0	+
-	-	-	?
0	-	0	+
+	?	+	+

- Notice the need for ? value
  - Arises because of the abstraction

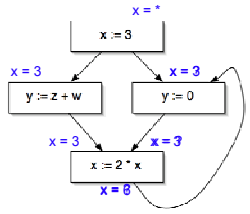
8

Dataflow Analysis

- Classic style of program analysis
- Used in optimizing compilers
  - Constant propagation
  - Common sub-expression elimination
  - etc.
- Efficiently implementable
  - At least, interprocedurally (within a single proc.)
  - Use bit-vectors, fixpoint computation

9

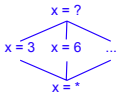
Control-Flow Graph



10

Lattices and Termination

- Dataflow facts form a lattice

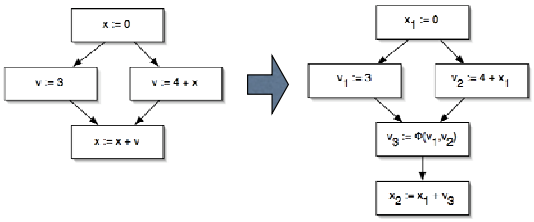


- Each statement has a transformation function
  - Out(S) = Gen(S) U (In(S) - Kill(S))
- Terminates because
  - Finite height lattice
  - Monotone transformation functions

11

Static Single Assignment Form

- Transform CFG so each use has a single defn



12

Lambda Calculus

- Three syntactic forms
  - e ::= x variable
  - | λx.e function
  - | e e function application
- One reduction rule
  - $(\lambda x.e_1)\ e_2 \rightarrow e_1[e_2/x]$ (replace x by e_2 in e_1)
- Can represent any computable function!

13

Example

- Conditionals
  - true = λx.λy.x false = λx.λy.y
  - if a then b else c = a b c
    - if true then b else c = (λx.λy.x) b c → (λy.b) c → b
    - if false then b else c = (λx.λy.y) b c → (λy.y) c → c
- Can also represent numbers, pairs, data structures, etc, etc.
- Result: Lingua franca of PL

14

Type Systems

- Machine represents all values as bit patterns
  - Is 00110110111100101100111010101000
    - A signed integer? Unsigned integer? Floating-point number? Address of an integer? Address of a function? etc.
- Type systems allow us to distinguish these
  - To choose operation (which + op), e.g., FORTRAN
  - To avoid programming mistakes
    - E.g., don't treat integer as a function address

15

Simply-typed λ-calculus

- e ::= x | n | λx:τ.e | e e
- τ ::= int | τ → τ
- A⊢ e : τ in type environment A, expression e has type τ

$$\frac{}{A \vdash n : int} \qquad \frac{x \in dom(A)}{A \vdash x : A(x)}$$

$$\frac{A[\tau/x] \vdash e : \tau'}{A \vdash \lambda x{:}\tau.e : \tau \rightarrow \tau'} \qquad \frac{A \vdash e1 : \tau \rightarrow \tau' \quad A \vdash e2 : \tau}{A \vdash e1\ e2 : \tau'}$$

16

Subtyping

- Liskov:
  - If for each object o_1 of type S there is an object o_2 of type T such that for all programs P defined in terms of o_1, the behavior of P is unchanged when o_2 is substituted for o_1 then S is a subtype of T.
- Informal statement
  - If anyone expecting a T can be given an S instead, then S is a subtype of T.

17

Axiomatic Semantics

- Old idea: Shouldn't just hack up code, try to prove programs are correct
- Proofs require reasoning about the meaning of programs
- First system: Formalize program behavior in logic
  - Hoare, Dijkstra, Gries, others

18

Hoare Triples

- {P} S {Q}
  - If statement S is executed in a state satisfying precondition P, then S will terminate, and Q will hold of the resulting state
  - Partial correctness: ignore termination
- Weakest precondition for assignment
  - Axiom: {Q[e\x]} x := e {Q}
  - Example: {y > 3} x := y {x > 3}

19

Other Technologies and Topics

- Control-flow analysis
- CFL reachablity and polymorphism
- Constraint-based analysis
- Alias and pointer analysis
- Region-based memory management
- Garbage collection
- More...

20

Applications: Abstract Interp.

- Everything!
- But in particular, Polyspace
  - Looks for race conditions, out-of-bounds array accesses, null pointer dereferences, non-initialized data access, etc.
  - Also includes arithmetic equation solver

21

Applications: Dataflow analysis

- Optimizing compilers
  - I.e., any good compiler
- ESP: Path-sensitive program checker
  - Example: can check for correct file I/O properties, like files are opened for reading before being read
- LCLint: Memory error checker (plus more)
- Meta-level compilation: Checks lots of stuff
- ...

22

Applications: Symbolic Evaluation

- PREFix
  - Finds null pointer dereferences, array-out-of bounds errors, etc.
  - Used regularly at Microsoft
- Also ESP

23

Applications: Model Checking

- SLAM, BLAST, Yasm
  - Focus on device drivers: lock/unlock protocol errors, and other errors sequencing of operations
- Uses alias analysis, predicate abstraction, analysis of recursive functions…

24

Applications: Axiomatic Semantics

- Extended Static Checker and Spec#
  - Can perform deep reasoning about programs
  - Array out-of-bounds
  - Null pointer errors
  - Failure to satisfy internal invariants
- Based on theorem proving

25

Applications: Type Systems

- Type qualifiers
  - Format-string vulnerabilities, deadlocks, file I/O protocol errors, kernel security holes
- Vault and Cyclone
  - Memory allocation and deallocation errors, library protocol errors, misuse of locks

26

Conclusion

- PL has a great mix of theory and practice
  - Very deep theory
  - But lots of practical applications
- Recent exciting new developments
  - Focus on program correctness instead of speed
  - Forget about full correctness, though
  - Scalability to large programs essential
- Source: Jeff Foster's course in Univ. of Maryland

27

Possible Course Syllabus

- Week 1 Introduction, course setup
- Week 2 Dataflow analysis
- Week 3 More dataflow. PA as MC of AI, monotone frameworks
- Week 4 Program semantics (Schmidt), worklist algorithms
- Week 5 Interprocedural analysis, context sensitive analysis
- (Pnueli), Bebob, Reps/Sagiv
- Week 6 Abstract Interpretation
- Week 7 More abstract interpretation (widening, shape analysis)
- Week 8 Lambda calculus, Type systems
- Week 9 Type systems (Cont'd), powersets
- Week 10 Axiomatic semantics
- Week 10 Axiomatic semantics, weakest precondition, C#, ESC/Java
- Week 12 Applications: Slicing and testcase generation
- Week 13 Applications: Security analysis

28

Introduction to the actual material

- Data-flow analysis – reaching definitions
  - From Chapter 1 of textbook
  - Slides 15, 18-37
- Abstract interpretation
  - From Chapter 1 of textbook
  - Slides 58-71

29