

**CMSC 631 – Program Analysis and Understanding
Fall 2006**

About this Class

- Topic: Analyzing and understanding software

- Three main focus areas:

- ‡ Formal systems and notations

- Vocabulary for talking about programs

- ‡ Static analysis

- Automatic reasoning about source code

- ‡ Programming language features

- Affects programs and how we reason about them

CMSC 631

2

Personnel

- Michael Hicks

- ‡ Office: 4131 AVW

- ‡ E-mail: mwh at cs.umd.edu

- ‡ Office hours: M 3:30-4:30pm, W 10am-11am
- Or by appointment

- James Rose

- ‡ E-mail: rosejr at cs.umd.edu

- ‡ Programming project grading

CMSC 631

3

Prerequisite

- CMSC 430 or equivalent compiler class

- ‡ Ideas we will use in this class:

- Parse trees/abstract syntax trees

- BNF notation for grammars

- Type checking (usually little coverage in a compilers class)

- Data flow analysis (coverage varies in a compilers class)

- Tools like yacc and lex may be useful for your project

- ‡ We won't use most of the other material

- So even without taking a compilers class, you may be OK

- Talk to me if you're not sure

CMSC 631

4

Textbooks

- No required textbooks

- Two recommended texts

- ‡ Pierce, *Types and Programming Languages*

- ‡ Huth and Ryan, *Logic in Computer Science*

- Neither covers everything in the course

- On reserve in CS library

CMSC 631

5

Forum

- Web forum on CS dept server

- ‡ <https://forum.cs.umd.edu/forumdisplay.php?f=40>

- Can use the forum to communicate with others in the class and ask questions of general interest

CMSC 631

6

Expectations: Homework

- Two kinds of assignments:
 - ‡ Programming assignments (20% of grade)
 - Every two weeks
 - Implement the ideas we see in lecture
 - ‡ Written assignments (10% of grade)
 - Every week
 - Short problem sets
- This is how you will learn things
 - ‡ Much more effective than listening to a lecture

CMSC 631

7

Late Policy on Assignments

- Programming Assignments: Due at midnight
 - ‡ We use Marmoset for submissions
- <http://submit.cs.umd.edu>
- Written assignments: Due at start of class
 - ‡ No late submissions
- Contact me about extenuating circumstances
 - ‡ E.g., religious holidays
 - ‡ Inform me as soon as possible

CMSC 631

8

Expectations: Participation

- Will need to read some papers for class
 - ‡ More later on in the semester
 - ‡ Should come prepared to contribute to discussion
- (Possible) student presentations later in the semester
 - ‡ Read 1-2 papers on a topic
 - ‡ Present a lecture in class about the material
- 10% of grade on class participation

CMSC 631

9

Expectations: Project

- Class goal: Teach you how to do research
 - ‡ So you have to do research as part of the class
- Substantial research project (35% of grade)
 - ‡ Any topic vaguely related to the class
 - Will post some suggestions for projects later on
 - May also be able to share project with other class
 - ‡ Completed in groups of size 2 (possibly 1 or 3)
- This will consume second-half of semester
 - ‡ Will ease up on homeworks, reading

CMSC 631

10

Expectations: Project (cont'd)

- Deliverables
 - ‡ Project proposal (one page) + talk with me
 - ‡ Project write-up
 - A conference-style paper (5-15 pages, as appropriate)
 - ‡ Implementation, if any
 - ‡ In-class presentation
 - 10-20 minutes, depending on # of projects

CMSC 631

11

Expectations: Exam

- Final exam (25% of grade)
 - ‡ Based on written and programming assignments
 - ‡ Take-home
- or in-class if you'd prefer; we can vote

CMSC 631

12

Academic Dishonesty

- Don't do it

CMSC 631

13

Software Chat

- <http://www.cs.umd.edu/projects/softchat>
- Weekly meeting about PL and SE research
- Mondays at 11am in 3118 CSIC this fall
 - ‡ Starting a week from Monday
- Topics include
 - ‡ Current research in the department
 - ‡ Practice talks
 - ‡ Interesting recent papers

CMSC 631

14

CMSC 631 – Program Analysis and Understanding Fall 2006

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

CMSC 631

16

Example

- $e ::= n \mid e + e$

$$\alpha(n) = \begin{cases} - & n < 0 \\ 0 & n = 0 \\ + & n > 0 \end{cases} \quad \begin{array}{c|ccc} + & - & 0 & + \\ - & - & - & ? \\ 0 & - & 0 & + \\ + & ? & + & + \end{array}$$

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

CMSC 631

17

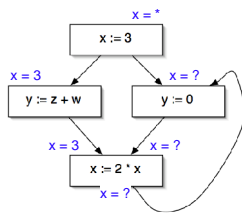
Dataflow Analysis

- 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

CMSC 631

18

Control-Flow Graph

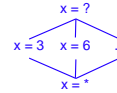


CMSC 631

19

Lattices and Termination

- Dataflow facts form a lattice



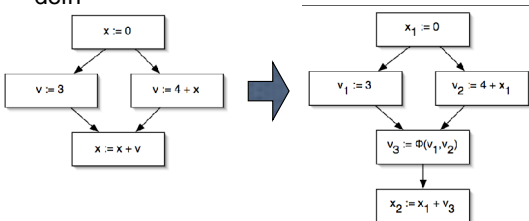
- Each statement has a transformation function
 - $\text{Out}(S) = \text{Gen}(S) \cup (\text{In}(S) - \text{Kill}(S))$
- Terminates because
 - Finite height lattice
 - Monotone transformation functions

CMSC 631

20

Static Single Assignment Form

- Transform CFG so each use has a single defn



CMSC 631

21

Lambda Calculus

- Three syntactic forms

- $e ::= x$ variable
- $\lambda x. e$ function
- $e_1 e_2$ 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!

CMSC 631

22

Example

- Conditionals

- $\text{true} = \lambda x. \lambda y. x$ $\text{false} = \lambda x. \lambda y. y$
- if a then b else $c = a b c$
 - if true then b else $c = (\lambda x. \lambda y. x) b c \rightarrow (\lambda y. b) c \rightarrow b$
 - if false then b else $c = (\lambda x. \lambda y. y) b c \rightarrow (\lambda y. y) c \rightarrow c$

- Can also represent numbers, pairs, data structures, etc.
- Result: Lingua franca of PL

CMSC 631

23

ML: Meta-Language

- ML designed originally for theorem provers
 - But after a while, realized could be general-purpose

- Mostly-functional language

- Similar to lambda-calculus
 - Encouraged avoid side-effects, call-by-value

- We'll use OCaml for programming assignments

CMSC 631

24

Program Semantics

- To be able to analyze programs, we have to know what they mean
 - ‡ *Semantics* comes from the Greek *semaino*, or "to mean"
- Three styles of formal semantics
 - ‡ Operational semantics
 - Like an interpreter
 - ‡ Denotational semantics
 - Like a compiler
 - ‡ Axiomatic semantics
 - Semantics is based on what you can prove about programs

CMSC 631

25

Operational Semantics

- Evaluation is depicted as *operationally*, as part of some *abstract machine*
 - ‡ Program states are reduced according to some transition relation \rightarrow . An example is our lambda calculus rule:
 - ‡ $(\lambda x.e_1) e_2 \rightarrow e_1[e_2/x]$
- There are different styles of abstract machine
 - ‡ Small-step (as above), big-step (natural semantics), SECD machine ...
- The *meaning* of a program is its fully reduced form (a.k.a. a *value*)

CMSC 631

26

Denotational Semantics

- The meaning of a program is defined as a mathematical object, like a function or number
 - ‡ Rather than a sequence of machine states
- The semantics is given in terms of an *interpretation* function $[[\cdot]]$
- Things get interesting when trying to define denotations for recursive constructs

CMSC 631

27

Denotational Semantics example

- $b ::= \text{true} \mid \text{false} \mid b \vee b \mid b \wedge b$
- $e ::= 0 \mid 1 \mid \dots \mid e + e \mid e * e$
- $s ::= e \mid \text{if } b \text{ then } s \text{ else } s$
 - ‡ $[[\text{true}]] = \text{true}$
 - ‡ $[[b_1 \vee b_2]] = [[b_1]] \text{ or } [[b_2]]$
 - ‡ $[[\text{if } b \text{ then } s_1 \text{ else } s_2]] = \begin{cases} [[s_1]] & \text{iff } [[b]] \text{ holds} \\ [[s_2]] & \text{iff } [[b]] \text{ does not hold} \end{cases}$
 - ‡ How would we handle a while loop?

CMSC 631

28

Axiomatic Semantics

- With the aforementioned semantics, we define the behavior of programs, and then reason about programs in terms of this behavior
 - ‡ Are two programs equivalent? Does a program terminate? Does a program implement a particular specification?
- Axiomatic semantics instead directly assigns meaning in terms of what one can prove
 - ‡ Hoare, Dijkstra, Gries, others

CMSC 631

29

Example: 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: $\{y > 3\} x := y \{x > 3\}$

CMSC 631

30

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

CMSC 631

31

Simply-typed λ -calculus

- $e ::= x \mid n \mid \lambda x:\tau.e \mid e e$

- $\tau ::= \text{int} \mid \tau \rightarrow \tau$

- $A \vdash e : \tau$ in type environment A , expression e has type τ

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

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

CMSC 631

32

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 .

CMSC 631

33

Other Technologies and Topics

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

CMSC 631

34

Applications: Parsing

- Syntactic bug pattern checkers

‡ ASTLog

‡ PREFast

- Buffer overflows! (sizeof() of wrong type in copy operations)

‡ FindBugs

- wait() not inside of a loop
- Pointer to internal array returned (unsafe)
- Dereference of null pointer

CMSC 631

35

Applications: Abstract Interpretation

- 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

- Stacktool

‡ Abstractly interprets machine code to check for possible stack overflow in embedded systems

CMSC 631

36

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
- ...

CMSC 631

37

Applications: Symbolic Evaluation

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

CMSC 631

38

Applications: Model Checking

- SLAM and BLAST
 - ‡ Focus on device drivers: lock/unlock protocol errors, and other errors sequencing of operations
- Uses alias analysis, predicate abstraction, and more
- *(We will not cover model checking extensively in this case; check out CMSC 630 in the Spring!)*

CMSC 631

39

Applications: Axiomatic Semantics

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

CMSC 631

40

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

CMSC 631

41

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

CMSC 631

42

This document was created with Win2PDF available at <http://www.win2pdf.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.