Qualifying-Exam Syllabus: Programming Languages and Compilers

Spring 2003

Charles Fischer, Susan Horwitz, Somesh Jha, and Thomas Reps
University of Wisconsin

The course requirements are CS 536, CS 538, CS 701, and CS 704. You should consult the syllabi for these courses when you study for the exam. If you did not take these courses, please ask the faculty for copies from recent offerings of the courses. (Required material is also sometimes covered, or covered in more detail, in CS 703; however, the topic of CS 703 varies from year to year.)

This document lists topics and concepts that the exam may cover, and is provided as a study aid. The general references that are provided in the different sections are places where you should be able to find a discussion of the majority of the topics listed in that section. In many cases, several general references have been given so that you have a number of different options (e.g., to find a treatment of a particular topic at a level that you find accessible). Many of the references are books. You do not have to read the entire book; consult the table of contents and the index to find where particular topics are discussed.

Most of the books are available at Wendt Library; some may be available only at the reserve desk.

1. Programming Paradigms

General References: [Sethi 1989]

- Procedural languages
- Object-oriented languages (C++ and Java)
  - — Classes, objects, methods
  - — Inheritance & subclass extension
  - — Multiple inheritance
- Functional languages (Lisp or Scheme, ML) [Henderson 1980], [Field and Harrison 1988]
  - — Programs as expressions
  - — Programming without side-effects
  - — Lists and list operators (cons, head, tail)
  - — Tail recursion
  - — Functions as first-class objects
  - — Polymorphism
  - — Type variables
  - — Type inference
- λ-calculus [Stoy 1977], [Glaser, Hankin, and Till 1984]
  - — Substitution, reduction rules, normal form
  - — Expressing programming constructs in λ-calculus: booleans, numerals, pairing
  - — Fixed-point combinators [Stoy 1977, Chapter 5]
  - — Confluence (Church-Rosser property) [Rosser 1982, Section 4]
- Logic languages (Prolog)
  - — Logic vs. control

— Facts and rules

— Backtracking and unification

— Depth-first and breadth-first search

— Issues concerning the ordering of rules and literals

2. Features and Properties of Languages

General References: [Sethi 1989], [Pratt 1984]

• Scoping

— Dynamic (Lisp)

— Static

— Extensions for importing and exporting names

• Evaluation

— Lazy [Field and Harrison 1988, Chapter 4]

• Exceptions (Java, ML)

• Parameter-passing modes

— Value

— Result

— Value-result

— Reference

— Name

• Data types and user-defined data types

— Abstract data types

— Name/structural equivalence

• Data encapsulation (classes, modules)

• Overloading and coercion (C++ and Java)

• Infinite data objects [Friedman and Wise 1976]

• Multiple inheritance [Cardelli 1984]

3. Translation and Implementation

General References: [Aho, Sethi, Ullman 1985], [Fischer and LeBlanc 1988], [Waite and Goos 1983], [Wilhelm and Maurer 1995], [Muchnick 2000]

• Lexical analysis

• Parsing

— Recursive descent

— LL(1)

— LR(1)

• Symbol tables and type checking

• Code generation

— Algorithms on trees and directed acyclic graphs

— Syntax-directed translation

- Register allocation
  - Sethi-Ullman register allocation [Aho, Sethi, Ullman 1985, Section 9.10], [Sethi and Ullman 1970]
  - Graph-coloring methods [Chaitin 1982]
- Partial Evaluation [Jones et al. 1993]
  - Futamura projections
  - Binding-time analysis/specialization
  - Runtime code generation [Auslander et al. 1996]
- Runtime execution models
  - Activation records
  - Dynamic storage management (heap-allocated storage)
  - Garbage collection [Jones and Lins 1996]
- Interpretation

4. Formal Methods for Describing Languages and Reasoning About Programs

General References: [Aho, Sethi, Ullman 1985], [Schmidt 1986], [Field and Harrison 1988], [Nielson and Nielson 1992]

- Grammars
  - Regular expressions
  - Context-free grammars
  - Attribute grammars [Aho, Sethi, and Ullman 1985]
- Structural induction [Burstall 1969]
- Operational semantics [Nielson and Nielson 1992]
  - Small-step
  - Large-step
- Axiomatic semantics [Hoare 1969], [Nielson and Nielson 1992]
  - Partial correctness
  - Total correctness
- Denotational semantics [Schmidt 1986], [Stoy 1977], [Gordon 1979]
  - Continuation semantics [Gordon 1979]
- Domain theory [Schmidt 1986, Chapters 3, 6, and 11]

5. Intermediate Representations (IRs)

General References: [Aho, Sethi, and Ullman 1985], [Muchnick 2000]

- Parse trees
- Abstract syntax trees
- DAGs
- Control-flow graphs
- Call graphs

- Dependences and dependence-based IRs
  - Control dependence
  - Data dependence (flow, anti, output)
  - Program dependence graphs
  - SSA-form

6. Static and Dynamic Program Analysis

General References: [Aho, Sethi, Ullman 1985], [Wilhelm and Maurer 1995], [Muchnick 2000], [Nielson et al. 1999]

- Polymorphic type checking [Field and Harrison 1988], [Hancock 1987a], [Hancock 1987b]
  - Typed λ-calculus
  - Unification, substitutions, type environments, Algorithm W
- Intraprocedural dataflow analysis
  - Meet-over-all-paths (MOP) solution vs. solution to a set of equations
  - Flow-sensitive vs. flow-insensitive problems
- Interprocedural dataflow analysis
  - Meet-over-all-valid paths (MOVP) solution vs. solution to a set of equations
  - Context-sensitive vs. context-insensitive problems
- Abstract interpretation [Abramsky and Hankin 1987], [Nielson et al. 1999]
  - Collecting semantics
  - Galois connection, Galois insertion (abstraction/concretization)
  - Soundness requirements
- Declarative notations for specifying dataflow-analysis problems
  - Horn clauses
  - Set constraints [Aiken 1999]
  - Graph-reachability criteria [Reps 1998]
- Fixed-point-finding techniques
- Points-to analysis/alias analysis
- Optimization
  - Basic block
  - Intraprocedural
  - Interprocedural
- Software verification techniques (beginning Fall 2003)
  [Ball and Rajamani 2001], [Engler et al. 2000], [Corbett et al. 2000]"
- Profiling and instrumentation
  - Path profiling [Ball and Larus 1996]

7. Language-Based Security

- Software fault isolation (sandboxing) [Wahbe et al. 1993]
- Stack inspection [Erlingsson and Schneider 2000]

• Safe kernel extensions without run-time checking [Necula and Lee 1996]

• Static analysis for computer security
[Wagner et al. 2000], [Wagner and Dean 2001], [Chen and Wagner 2002], [Jensen et al. 1999], [Ashcraft and Engler 2002]

8. References

[Aiken 1999]
Aiken, A., Introduction to set constraint-based program analysis. *Science of Computer Programming 35* (1999), 79-111. [On-line copy available at http://www.cs.berkeley.edu/˜aiken/publications/papers/scp99.ps]

[Ashcraft and Engler 2002]
Ashcraft, K. and Engler, D., Using programmer-written compiler extensions to catch security holes. In *Proc. IEEE Security and Privacy 2002*. [On-line copy available at http://www.stanford.edu/˜engler/]

[Auslander et al. 1996]
Auslander, J., Philipose, M., Chambers, C., Eggers, S.J., and Bershad, B.N., Fast, effective dynamic compilation, In *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, May 1996. [On-line copy available at http://www.cs.washington.edu/research/dyncomp/Papers/pldi96-abstract.html]

[Abramsky and Hankin 1987]
Abramsky, S. and Hankin, C. An introduction to abstract interpretation, In *Abstract Interpretation of Declarative Languages*, S. Abramsky and C. Hankin (eds.), Ellis Horwood Limited, Chichester, West Sussex, UK, 1987, 9-31.

[Aho, Sethi, Ullman 1985]
Aho, A.V., Sethi, R., and Ullman, J.D., *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, Mass., 1985.

[Ball and Larus 1996]
Ball, T. and Larus, J., Efficient path profiling. *Proc. of MICRO-29*, Dec. 1996.

[Ball and Rajamani 2001]
T. Ball and S.K. Rajamani, Automatically validating temporal safety properties of interfaces. SPIN Workshop on Model Checking of Software, 2001.

[Burstall 1969]
Burstall, R.M., Proving properties of programs by structural induction. *Comp. J. 12*, 1 (Feb. 1969), 41-48.
[On-line copy available at http://www.cs.wisc.edu/˜cs704-1/Papers/induction.69.pdf.]

[Cardelli 1984]
Cardelli, L., A semantics of multiple inheritance. In *Semantics of Data Types: Proceedings of the International Symposium*, (Sophia-Antipolis, France, June 27-29, 1984), Lecture Notes in Computer Science, Vol. 173, G. Kahn, D. MacQueen, and G. Plotkin (eds.), Springer-Verlag, New York, NY, 1984, p. 51-67.

[Chaitin 1982]
Chaitin, G.J., Register allocation and spilling via graph coloring. *SIGPLAN Notices* 17, 6 (1982), pp. 98-105.

[Corbett et al. 2000]
J. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, Robby, S. Laubach, and H. Zheng, Bandera: Extracting finite-state models from Java source code. International Conference on Software Engineering, 2000, pp. 439-448.

[Engler et al. 2000]
D. Engler, B. Chelf, A. Chou, and S. Hallem, Checking system rules using system-specific, programmer-written compiler extensions. Proceedings of Operating Systems Design and Implementation (OSDI), September 2000.

[Erlingsson and Schneider 2000]
U. Erlingsson and F.B. Schneider, IRM enforcement of Java stack inspection. IEEE Symposium on Security and Privacy, May 2000, pp. 246-255.

[Field and Harrison 1988]
Field, A.J and Harrison, P.G., *Functional Programming*. Addison-Wesley, Reading, Mass., 1988.

[Fischer and LeBlanc 1988]
Fischer, C.N. and LeBlanc, R.J., *Crafting a Compiler*. Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1988.

[Friedman and Wise 1976]
Friedman, D.P. and Wise, D.S. Cons should not evaluate its arguments. In *3rd Int. Colloquium on Automata, Languages, and Programming*, Edinburgh University Press, 1976, pp. 257-284.

[Glaser, Hankin, and Till 1984]
Glaser, H., Hankin, C., and Till, D., *Principles of Functional Programming*, Prentice-Hall International, Englewood Cliffs, NJ, 1984.

[Gordon 1979]
Gordon, M. *The Denotational Description of Programming Languages*. Springer-Verlag, New York, 1979.

[Hancock 1987a]
Hancock, P., Polymorphic type-checking, Chapter 8 in *The Implementation of Functional Programming Languages*, Prentice-Hall, Englewood Cliffs, NJ, 1987, 139-162.

[Hancock 1987a]
Hancock, P., A type-checker, Chapter 9 in *The Implementation of Functional Programming Languages*, Prentice-Hall, Englewood Cliffs, NJ, 1987, 163-182.

[Hoare 1969]
Hoare, C.A.R. An axiomatic basis for computer programming. *Comm of the ACM 12*, 10 (Oct. 1969), 576-583.

[Jensen et al. 1999]
T. Jensen, D.L. Metayer, and T. Thorn, Verification of control flow based security properties. IEEE Symposium on Security and Privacy, May 1999.

[Jones et al. 1993]
N.D. Jones, C.K. Gomard, P. Sestoft, Chapters 1, 3, 4, and 5 in *Partial Evaluation and Automatic Program Generation*, Prentice-Hall Int., Englewood Cliffs, NJ, 1993.
[On-line copy available at http://www.dina.kvl.dk/˜sestoft/pebook/jonesgomardsestoft.ps, http://www.dina.kvl.dk/˜sestoft/pebook/jonesgomardsestoft.pdf.]

[Jones and Lins 1996]
R. Jones and R. Lins, *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. Wiley, NY, 1996.

[Muchnick 2000]
S.S. Muchnick, *Advanced Compiler Design and Implementation*. Morgan Kaufmann, 2000.

[Necula and Lee 1996]
Necula, G.C. and Lee, P., Safe kernel extensions without run-time checking. In *Proc. OSDI '96*, Oct. 1996. [On-line copy available at http://www.cs.berkeley.edu/~necula/papers.html]

[Nielson and Nielson 1992]
H.R. Neilson and F. Nielson, *Semantics with Applications: A Formal Introduction*, John Wiley and Sons, New York, NY, 1992.
[On-line copy available at http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.ps, http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.pdf.]

[Nielson et al. 1999]
F. Nielson, H.R. Nielson, and C. Hankin, *Principles of Program Analysis*, Springer-Verlag, 1999.

[Pratt 1984]
Pratt, T.W., *Programming Languages: Design and Implementation*. Prentice-Hall, 2nd. ed., 1984.

[Reps 1998]
Reps, T., Program analysis via graph reachability. *Information and Software Technology 40*, 11-12 (November/December 1998), pp. 701-726.
[On-line copy available at http://www.cs.wisc.edu/wpis/papers/tr1386.ps, http://www.cs.wisc.edu/wpis/papers/tr1386.pdf.]

[Rosser 1982]
Rosser, J.B. Highlights of the history of the lambda-calculus. In *Conf. Rec. of the 1982 ACM Symp. on Lisp and Functional Programming*, Pittsburgh, Penn., Aug. 15-18, 1982, pp. 216-225.

[Schmidt 1986]
Schmidt, D., *Denotational Semantics*. Allyn and Bacon, Inc., Boston, MA, 1986.

[Sethi 1989]
Sethi, R., *Programming Languages Concepts and Constructs*. Addison-Wesley, Reading, Mass., 1989.

[Sethi and Ullman 1970]
Sethi, R. and Ullman, J.D., The generation of optimal code for arithmetic expressions. *J. ACM* 17, 4 (1970).

[Stoy 1977]
Stoy, J., *Denotational Semantics*. The M.I.T. Press, Cambridge, MA, 1977.

[Wagner and Dean 2001]
Wagner, D. and Dean, D., Intrusion detection via static analysis. In *Proc. 2001 IEEE Symp. on Security and Privacy*. [On-line copy available at http://www.cs.berkeley.edu/~daw/papers/]

[Wagner et al. 2000]
Wagner, D., Foster, J.S., Brewer, E.A., and Aiken, A., A first step towards automated detection of buffer overrun vulnerabilities. In *Proc. Network and Distributed System Security Symposium*, 2000.
[On-line copy available at http://www.cs.berkeley.edu/~daw/papers/overruns-ndss00.ps]

[Wahbe et al. 1993]
Wahbe, R., Lucco, S., Anderson, T.E., and Graham, S.L., Efficient software-based fault isolation. In *Proc. of ACM Symp on Operating System Principles*, 1993, 203-216.

[Wilhelm and Maurer 1995]
R. Wilhelm and D. Maurer, *Compiler Design*. Addison-Wesley, 1995.