CSE P 501 – Compilers

Exam Review

Hal Perkins

Autumn 2021

When/Where

- This Thursday, Dec. 2, 6:30-8:00
  - Will plan to end @8:00 but if people are still working we can let it run longer, maybe up to 8:20 or so (i.e., want to avoid time pressure)
  - And we might start a little late if we can't get into the UW room before 6:30 (reservation starts then)
- Two locations
  - UW CSE2 G01 (the twin classroom across the hall from our regular room)
  - Microsoft Building 99 room 1919 (large lecture hall)
    - Open to anyone in the class
    - Everyone (MSFT and others) must do online health check-in ahead of time and wear masks at all times.

What

- Info, topic list, and old exams on course web
- Exam is (almost) closed book
  - Brief reference info and definitions will be supplied on the exam as needed (see old exams for examples)
  - OK to bring two 5x8 index cards with any *hand-written* notes you wish

Exam topics 1 (from online list)

- Interpreters and compilers - key differences
- Gross structure of compilers - tasks of front/middle/back ends
- Basic notions of grammars - productions, terminals, non-terminals
- Regular expressions and DFAs
  - RE operators (as done in class, not Bash/PERL/Python/…)
  - Constructing REs and DFAs (but you do not need to know the full RE -> NFA -> DFA construction algorithms)
- Scanners - transforming character streams to token streams

Exam topics 2 (from online list)

- Context-free grammars
  - Derivations, leftmost, rightmost, etc.
  - Constructing grammars for sets of strings
  - Ambiguity
  - First, follow, and nullable
- LR parsing
  - Shift-reduce parsing
  - Construction of LR(0) and SLR(1) parse tables
    - Items, item sets, and parser states
    - Shift-reduce and reduce-reduce conflicts
    - LR(0) vs. SLR grammars and how lookahead is used in SLR
- LL and recursive-descent parsers
  - Constructing hand-written recursive-descent parsers
  - Grammar problems and solutions - left recursion removal, left-factoring common prefixes, etc.

Exam topics 3 (from online list)

- Static semantics & symbol tables
  - Kinds of things checked in this phase of a compiler
  - Attribute grammar formalism and how to use it to think about where in the AST to perform checks
  - Basic symbol table structures for languages like MiniJava
  - Representation of type information in a compiler

Exam topics 4 (from online list)

- Basic x86-64 architecture
  - Core instruction set. Don't memorize details – handouts or ref. info will be provided – but be able to read and write simple code
- Code shape
  - Representation of common high-level language constructs in x86-64 assembly language
  - Representation of objects and implementation of new
  - Implementation of method calls and dynamic dispatch
    - Method tables and overriding
    - Be sure you understand basic Java rules for method overriding and field hiding in extended classes
  - Be able to translate simple C or Java code to x86-64, including calling conventions

Exam topics 5 (from online list)

- Intermediate representations, particularly:
  - Abstract syntax trees (ASTs)
  - Control-flow graphs, basic blocks
- Analysis and optimization
  - Value numbering techniques
  - General form of dataflow equations (def, use, in, and out sets) and how these are used to analyze typical problems like liveness; be able to solve simple problems like the ones we saw in class & on HW4
  - Dominators and immediate dominators; how to find a loop in a CFG
  - Basic idea of SSA - what it means; dominance frontiers; be able to hand translate a simple CFG into SSA with appropriate phi functions (do not need to precisely simulate the detailed algorithms for this);
  - Interaction between analysis and optimizations - what can we do with the information that is discovered by the analysis; when is a transformation safe .

Exam non-topics

- Not responsible for details of algorithms or examples from today's lecture on compiler backend (e.g., tree pattern matching for instruction selection, list scheduling, register allocation by graph coloring)
  - Still, good stuff to know and you should understand where this is done in the compiler (i.e., what does each part of the compiler do?)

Questions?