

Lecture 27

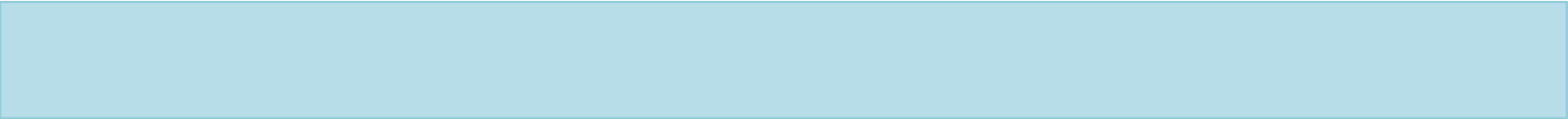
COMPILER DESIGN

Announcements

- **HW6:** Analysis & Optimizations (the final homework)
 - Alias analysis, constant propagation, dead code elimination, register allocation
- **Final Exam**
 - Scheduled for Friday, February 5th, 14:00-16:00
 - Waiting for further details
 - Past exam/solution from Fall 2019: <https://exams.vis.ethz.ch/>

Plan

- ~~Next: Register allocation~~
- Upcoming
 - ~~Dataflow analysis (part 2)~~
 - ~~Control flow analysis & SSA~~
 - ~~Garbage collection (GC)~~
 - ~~Compiler testing & validation~~
 - ~~How to find thousands of bugs in GCC & LLVM?~~
 - ~~Compiler verification~~
 - ~~How to build a fully verified *realistic* compiler?~~
 - ~~Guest lecture on MLIR~~
 - ~~Guest lecture on GraalVM, PGQL & Green Marlin~~
 - **The finale: (very) quick summary**



What have we learned?
Where else is it applicable?
What next?

SUMMARY

Final Exam

- Cover **everything until (and including) garbage collection** & the **6 HWs**
 - Lexing (regular expressions, DFA/NFA, lexer generator)
 - Parsing (top-down/bottom-up parsing, including LL, LR, & LALR)
 - Scope, type-checking, inference rules
 - Lambda calculus, closure conversion
 - Objects, inheritance, types, implementation of dynamic dispatch
 - Basic optimizations
 - Dataflow analysis (forward vs. backward, fixpoint computations, etc.)
 - Liveness, reaching definitions, available expressions, very busy expressions
 - Constant propagation
 - MOP solutions, distributed vs. non-distributed analyses
 - Linear-scan and graph-coloring register allocation
 - Control flow analysis
 - Loops, dominator trees
 - Garbage collection (mark & sweep, stop & copy, reference counting)
- Exam **focus** and **format**
 - Focus on the theory side, but may also include materials specific to the HWs
 - Focus on simple answers, computation, multiple choice, etc.
 - Sample exam/solution from Fall 2019: <https://exams.vis.ethz.ch/>

Why Compiler Design?

- You will learn:
 - Practical applications of theory
 - Parsing
 - How high-level languages are implemented in machine language
 - (A subset of) Intel x86 architecture
 - A deeper understanding of code
 - A little about programming language semantics
 - Functional programming in OCaml
 - How to manipulate complex data structures
 - How to be a better programmer

Did we meet these goals?

Materials we didn't cover

- We expectedly skipped many materials (both in depth and breadth)
- Concrete syntax/parsing
 - Much more to the theory of parsing
 - Good syntax is art, not science!
- Source language features
 - Exceptions, advanced type systems, type inference, concurrency
- Intermediate languages
 - Intermediate language design, bytecode, bytecode interpreters, just-in-time compilation (JIT)
- Compilation
 - Continuation-passing transformation, efficient representations, scalability
- Optimization
 - Scientific computing, cache, instruction selection/scheduling, etc.
- Runtime support
 - Advanced garbage collection algorithms

Where to go from here?

- Major relevant conferences
 - Programming Language Design and Implementation (PLDI)
 - Principles of Programming Languages (POPL)
 - Object Oriented Programming Systems, Languages & Applications (OOPSLA)
 - International Conference on Functional Programming (ICFP)
 - International Symposium on Code Generation and Optimization (CGO)
 - International Conference on Compiler Construction (CC)
 - European Symposium on Programming (ESOP)
 - ...
- Technologies / Open-Source Projects
 - Yacc, lex, bison, flex, ...
 - LLVM (low level virtual machine) → MLIR
 - Java virtual machine (JVM), Microsoft's Common Language Runtime (CLR)
 - WebAssembly
 - Languages: OCaml, F#, Haskell, Scala, Go, Rust, ...

Where else is this stuff applicable?

- General programming
 - In C/C++, better understanding of how the compiler works can help generate better code
 - Ability to read assembly output from compiler
 - Experience with functional programming can give different ways to think about how to solve a problem
- Writing domain specific languages
 - Tools like lex/yacc very useful for little utilities
 - Understanding abstract syntax and interpretation
- Understanding hardware/software interface
 - Different devices have different instruction sets, programming models

Thesis Projects

- A wealth of possible projects suitable for BSc (and MSc) theses
 - Language design & implementation
 - (Futuristic) Programming methodologies, environments and tools
 - Program analysis, verification, and testing
 - Software (including emerging software) quality, reliability, and security
 - Education technologies (for compilers, programming, and CS in general)
 - ...
- If interested, please feel free to get in touch
- Successful projects will likely lead to
 - Nice research results
 - Publications in the top CS conferences
 - PLDI, POPL, OOPSLA, ICSE, FSE, ...
 - More importantly, a taste of research & some tangible real impact

Thanks!

- To all the TAs for doing an excellent job helping with the course!
- To Theo, Dr. Grosser, Dr. Braun, & Dr. Wachsmuth for guest lecturing!
- To you all for taking and participating in the class!

- How can we improve the course for future offerings?
 - Hope you shared your feedback in the course evaluations!

- Happy Holidays, and good luck with your exams in early 2021!

