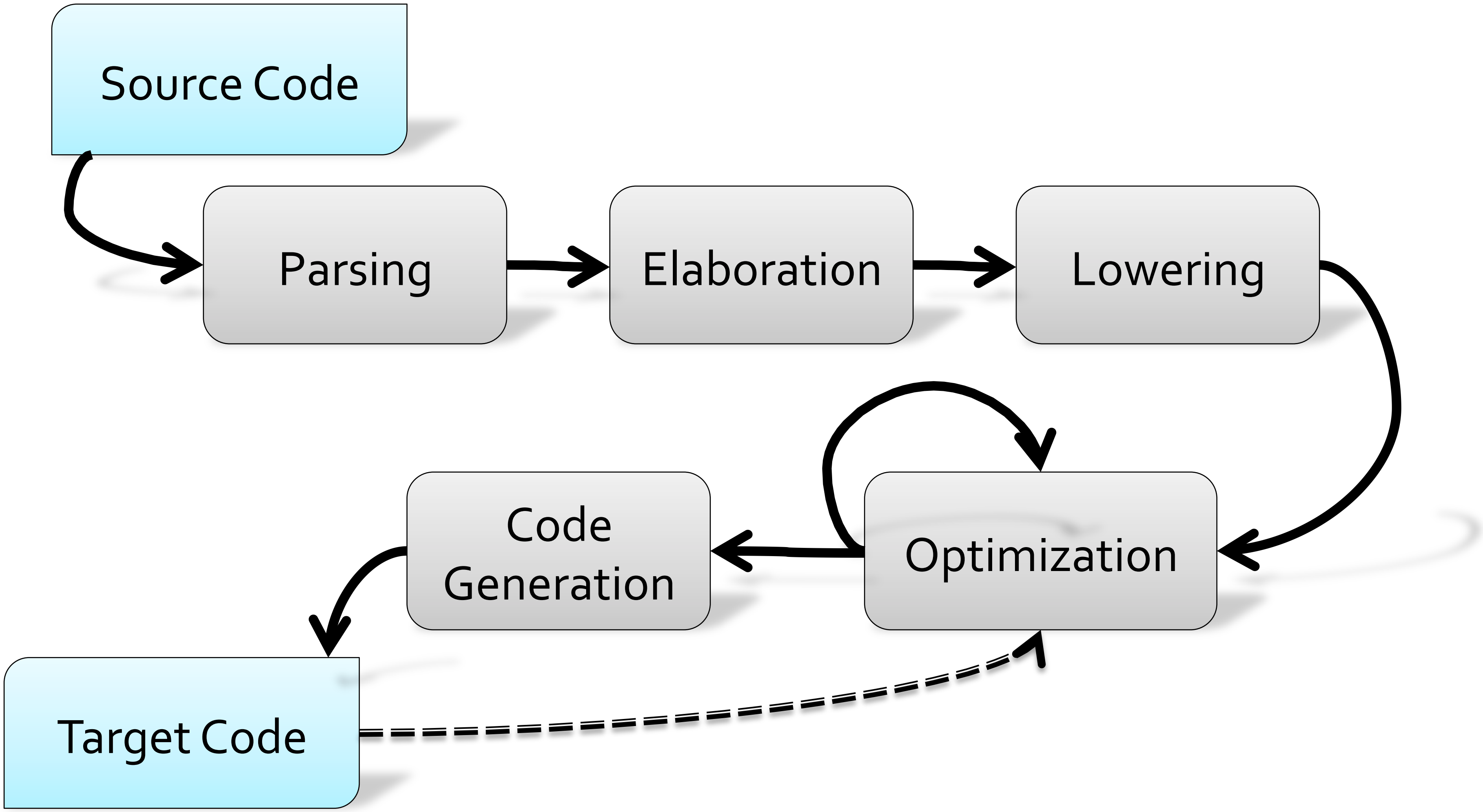
Amal Ahmed Spring 2013

CS4410: Compilers

What is the course about?

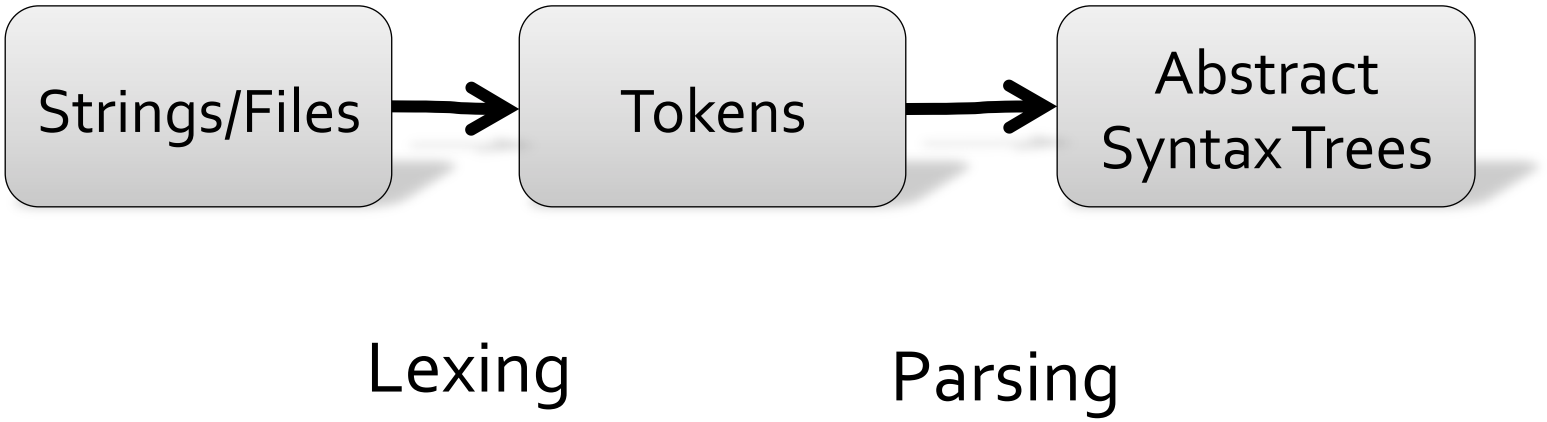
- Compilers.
  - Translating one programming language into another.
- Also interpreters.
  - Translating and running a language from within another language.

Basic Architecture



Front End

- Lexing & Parsing
  - From strings to data structures
  - Usually split into two phases:



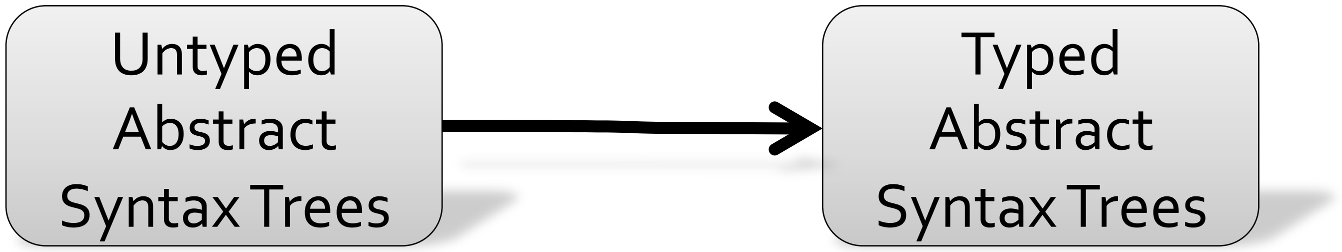
Parsing Tools

- Parsing is something that happens in essentially all applications.
  - E.g., Google search bar, calendar, etc.
  - First step in going from raw data to information
- Lots of CS theory (cs3800) to help out
  - Regular expressions (finite state automata)
  - Context-free grammars (push-down automata)
- Lots of tool support
  - E.g., Lex and Yacc; parsing combinators; etc.

Elaboration

Type-checking.

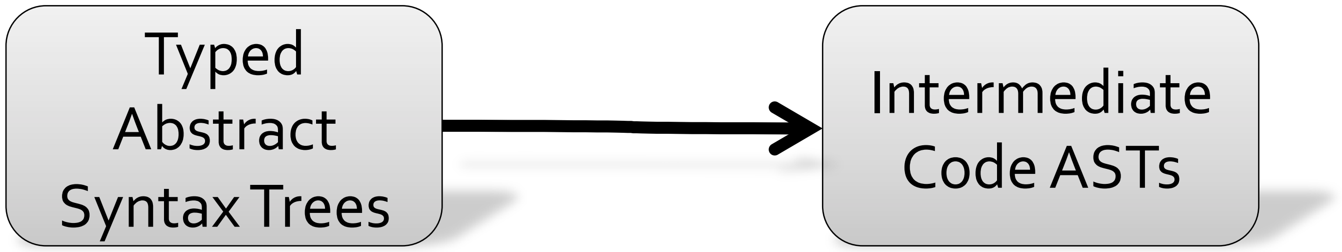
- Resolve variables, modules, etc.
- Check that operations are given values of the right types.
- Infer types for sub-expressions.
- Check for other safety/security problems.



Lowering

Translate high-level features into a small number of target-like constructs.

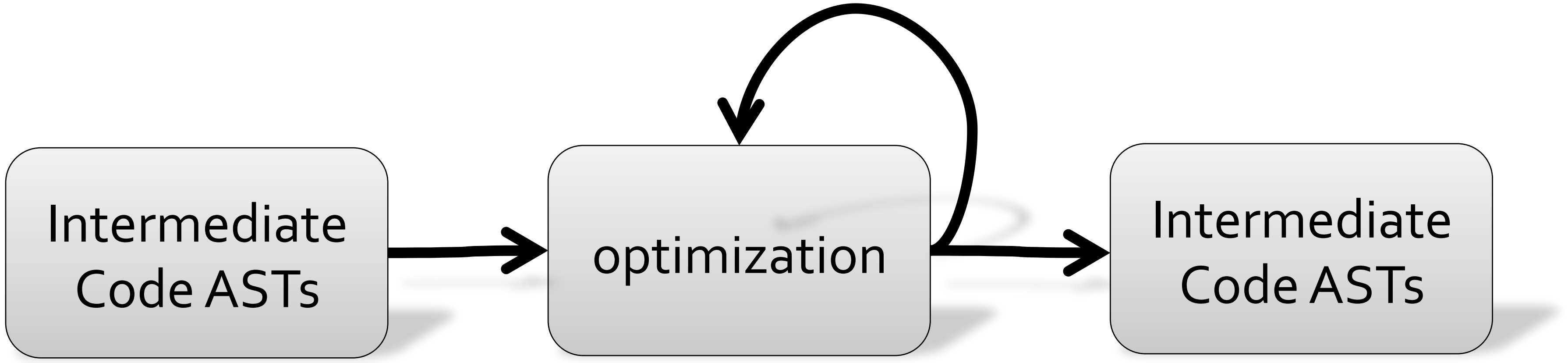
- e.g., while, for, do-loops all compiled to code using goto's.
- e.g., objects, closures to records and function pointers.
- e.g., make type-tests, array-bounds checks, etc. explicit.



Optimization

Rewrite expensive sequences of operations into less expensive.

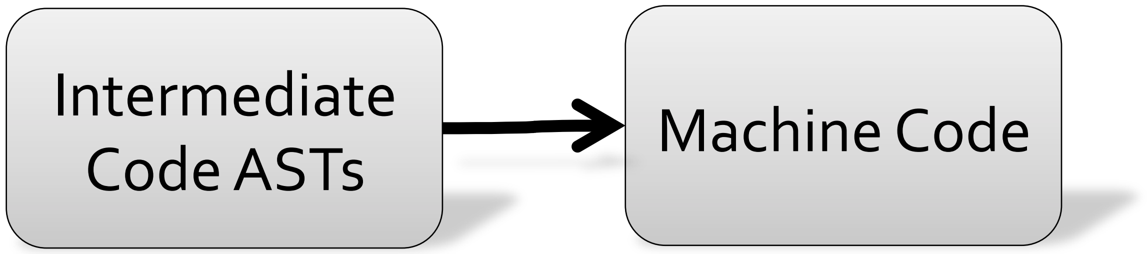
- e.g., constant folding: 3+4 → 7
- e.g., lift an invariant computation out of a loop
- e.g., parallelize a loop



Code Generation

Translate intermediate code into target code.

- Register assignment
- Instruction selection
- Instruction scheduling
- Machine-specific optimization



Who Should Take cs4410?

- People fascinated by languages & compilers
  - Why does[n't] this language include this feature?
- Systems programmers
  - Know the one tool that you use every day.
  - What can[`t] a compiler do well?
- Architecture designers
  - See 432 and itanium disasters
  - These days, architecture folks use compilers too!
- API designers
  - A language is the ultimate API
  - c.f., Facebook

What background?

- Some assumptions:
  - Assume you know a little bit about how machines work.
  - Assume you can write functional programs.
  - Assume you can pick up OCaml on your own.
  - Assume you are willing to work your tail off.

Administrivia

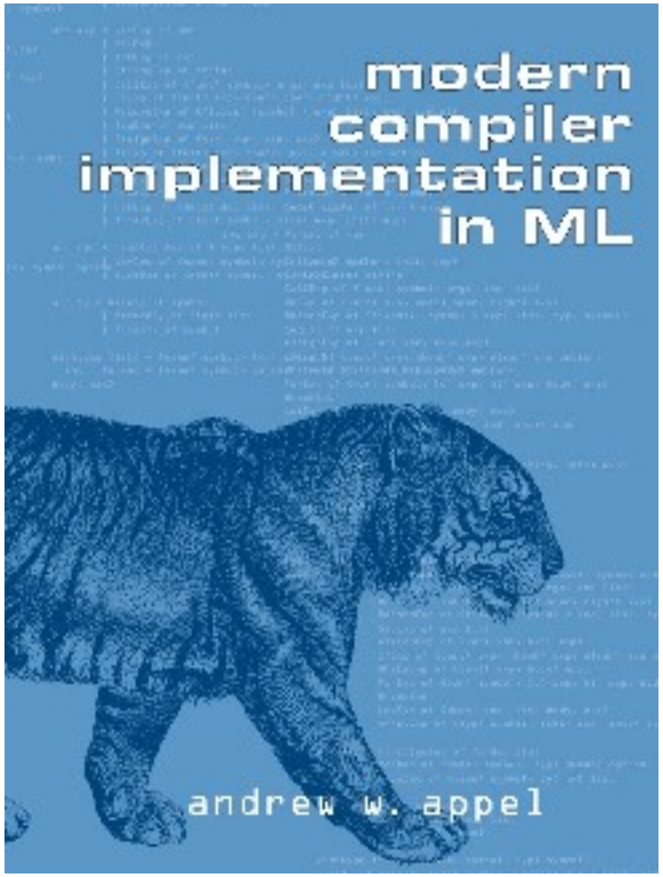
- 9 programming assignments.
  - Bulk of your grade.
  - Work with a partner.
  - No late days.
- Mid-term (Tuesday, Feb 26) and Final exams.
- See course website for more

Programming Environment

- Ocaml
  - Not familiar? Pick up along the way.
  - Ideal for writing compilers.
- SPIM
  - MIPS simulator.
  - Ideal target for compilers.

Book

Modern Compiler Implementation in ML, by Andrew Appel.



Projects

- MIPS simulator
  - Understand the machine we are targeting
- Fortran-ish -> MIPS
  - Simple front-end
  - Simple lowering & code generation
- C-ish -> MIPS
  - 1st-order procedures, structs, arrays
- Scheme-ish -> C-ish
  - Higher-order procedures, type-tests
- ML-ish -> Scheme-ish
  - Type inference

Projects, continued

- Algebraic optimization
  - Tree-based rewriting
- Control-flow graphs
  - Rip out naïve C-ish backend and replace with one based on control-flow graphs.
  - Implement liveness analysis
  - Implement graph-coloring register allocation

Key Outcomes

- Understand how compilers work
  - Parsing, type-checking & inference, lowering, analysis, optimization, code generation.
  - How to rewrite your code so that the compiler can do its job better.
  - Significant engineering experience on a big project.
- Understand language and architecture design tradeoffs.
  - e.g., why are error messages in compilers so bad?
  - e.g., why do advanced languages essentially require garbage collection?
  - e.g., what prevents us from parallelizing C/C++?

Along the way…

- What’s happening with architecture?
  - Multi-core
  - Heterogeneous units
- What’s happening with languages?
  - Security may matter more than performance
  - Productivity may matter more than performance
  - Cluster-level performance (e.g., Map/Reduce)
  - Feedback-driven optimization