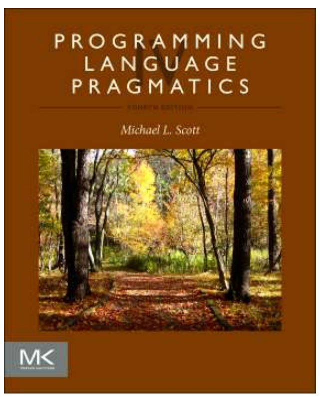
Introduction

17-363/17-663: Programming Language Pragmatics



Next edition: Scott & Aldrich!

Prof. Jonathan Aldrich

ELSEVIER

Introduction

- Language Design and Language Implementation go together
  - An implementor has to understand the language
  - A language designer has to understand implementation issues
  - A good programmer has to understand both!

Introduction

- Why are there so many programming languages?
  - evolution -- we've learned better ways of doing things over time
  - socio-economic factors: proprietary interests, commercial advantage
  - orientation toward special purposes
  - orientation toward special hardware
  - diverse ideas about what works well (and what people like)

Introduction

- What makes a language successful?
  - easy to learn (BASIC, Python, LOGO, Scheme)
  - expressive, powerful (C++, Common Lisp, Scala, Rust)
  - easy to implement (BASIC, Forth)
  - possible to compile to very good (fast/small) code (Fortran, C, Rust)
  - backing of a powerful sponsor (C#, Ada, Swift)
  - wide dissemination at minimal cost (Pascal, Java)
  - market lock-in (JavaScript)

Introduction

- Why do we have programming languages? What is a language for?
  - way of thinking / way of expressing algorithms
    - languages from the user's point of view
  - abstraction of virtual machine -- way of specifying what you want the hardware to do without getting down into the bits
    - languages from the implementor's point of view

Why study programming languages?

- Help you choose a language.
  - C++ vs. Rust for systems programming
  - Fortran vs. Julia for numerical computations
  - Python vs. JavaScript for web applications
  - Ada vs. C for embedded systems
  - Common Lisp vs. Scheme vs. ML for symbolic data manipulation
  - Java vs. Scala for application servers

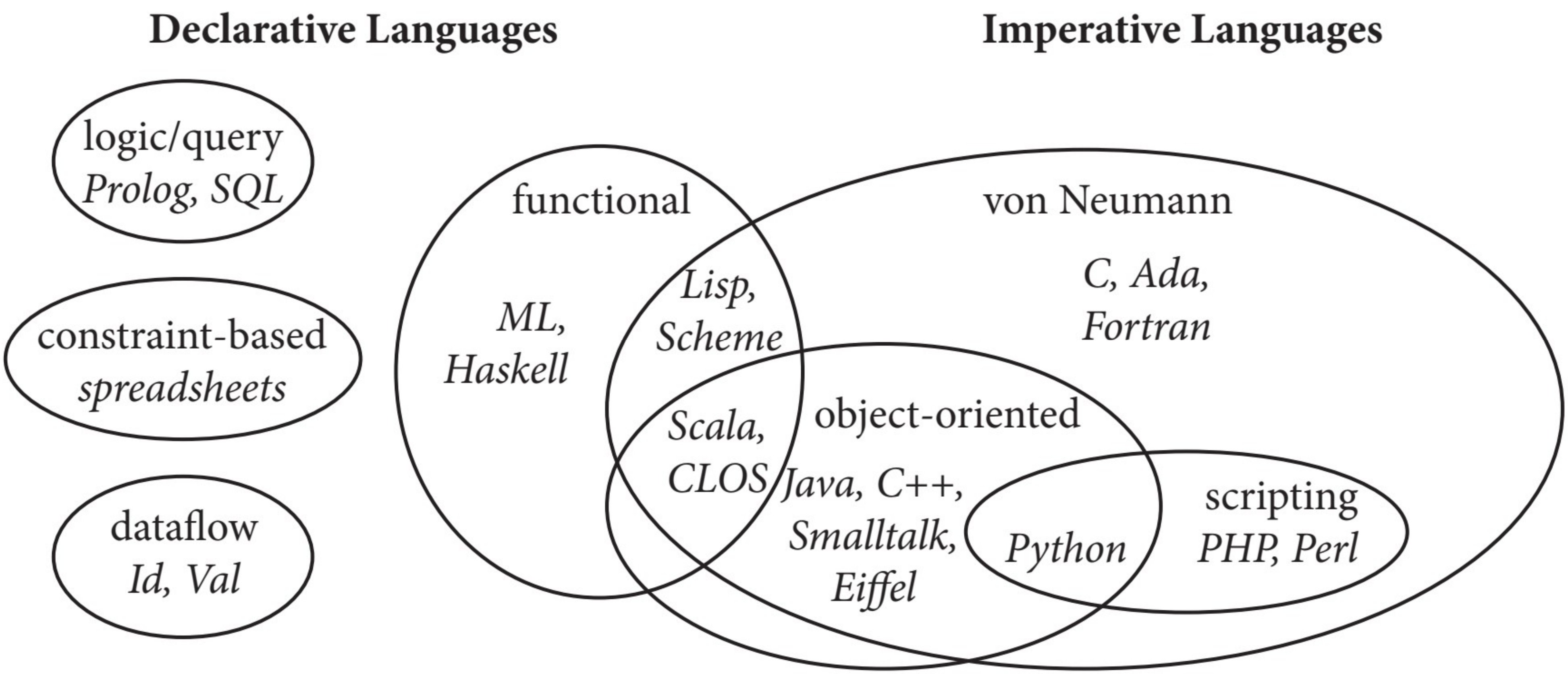
Why study programming languages?

- Make it easier to learn new languages
  - Familiarity with related languages
  - Understanding core concepts that reappear
- Use language/compiler ideas in your projects
  - Almost every complex system has a language somewhere!
- Learn how to reason rigorously
  - PL has some of the best intellectual tools!

Why study programming languages?

- Help you make better use of whatever language you use
  - Specialized features
    - unions, first-class functions, …
  - Implementation costs
    - Garbage collection, tail recursion
  - Emulating missing features
    - Recursion (with loops and stacks)
    - First-class functions (with objects)…or vice versa!

Language Paradigms



How is this course different?

- Overall: emphasizes the interaction between language design and implementation
- Vs. 15-410
  - More focus on language design and theory; fulfills the Logic & Languages elective, not the Systems elective
- Vs. 15-312
  - "Pragmatic" focus – we study ideas and theory in the context of industrial languages and their design choices
  - Use of an educational proof assistant to make theory both more approachable and rigorous

Course Staff

Prof. Jonathan Aldrich

TA Anrui Liu

Course Administration

- Lectures 2x/week
  - Active learning exercises in every class
  - In person expectation
    - If you can't make it (COVID is not gone, but there may be other reasons too), email me—we'll get you a video & exercises
- Textbook: Programming Language Pragmatics
  - We'll provide a PDF of the upcoming 5th edition
  - Please do not share
- Recitation
  - Lab-like, helpful for homework. Bring your laptop!

“How do I get an A?”

- 50% Homework –due Friday 11:59pm
  - Build a compiler (5 coding assignments, plus a warmup this week)
    - Implementation in Rust – good language for compilers & interesting to study
  - Reason about languages (4 theory assignments)
    - SASyLF educational theorem proving tool
- 20% - 2 midterm exams covering core concepts
- 25% Project
  - Extend the compiler in some interesting way, or explore theory
- 5% Participation (assessed via in-class exercises)
  - Can miss up to 2 sessions (lecture or recitation) w/o losing credit

Communication

- Website
  - Schedule, syllabus, slides
- Piazza for announcements, communication
  - Use Piazza as much as possible
  - Make questions public if possible, so others can benefit!
- Canvas
  - Assignments, grades
- Office hours TBA shortly (or just come by)

Read the Syllabus

A high level summary of some policies:

- Late work: 5 free late days
  - 10% penalty per day after these are used up
  - No credit more then 5 days late
  - Special circumstances: contact the instructor
- Collaboration policy
  - Your work must be your own
  - 100% penalty for cheating
  - Read full policy carefully
- No electronics in lecture
  - But bring them to recitation!

CMU can be pretty intense

- A 12-credit course is expected to take ~12 hours a week.
- We aim to provide a rigorous but tractable course.
  - More frequent assignments rather than big monoliths
  - Two midterm exams to cover core material as you learn it
- Please keep us apprised of how much time the class is actually taking and whether it is interfacing badly with other courses.
  - We have no way of knowing if you have three midterms in one week.
  - Sometimes, we misjudge assignment difficulty.
- If it’s 2 am and you’re panicking…put the homework down, send us an email, and go to bed.

Executing programs

- Consider the following program
  - In a simple imperative language, Hoare's WHILE

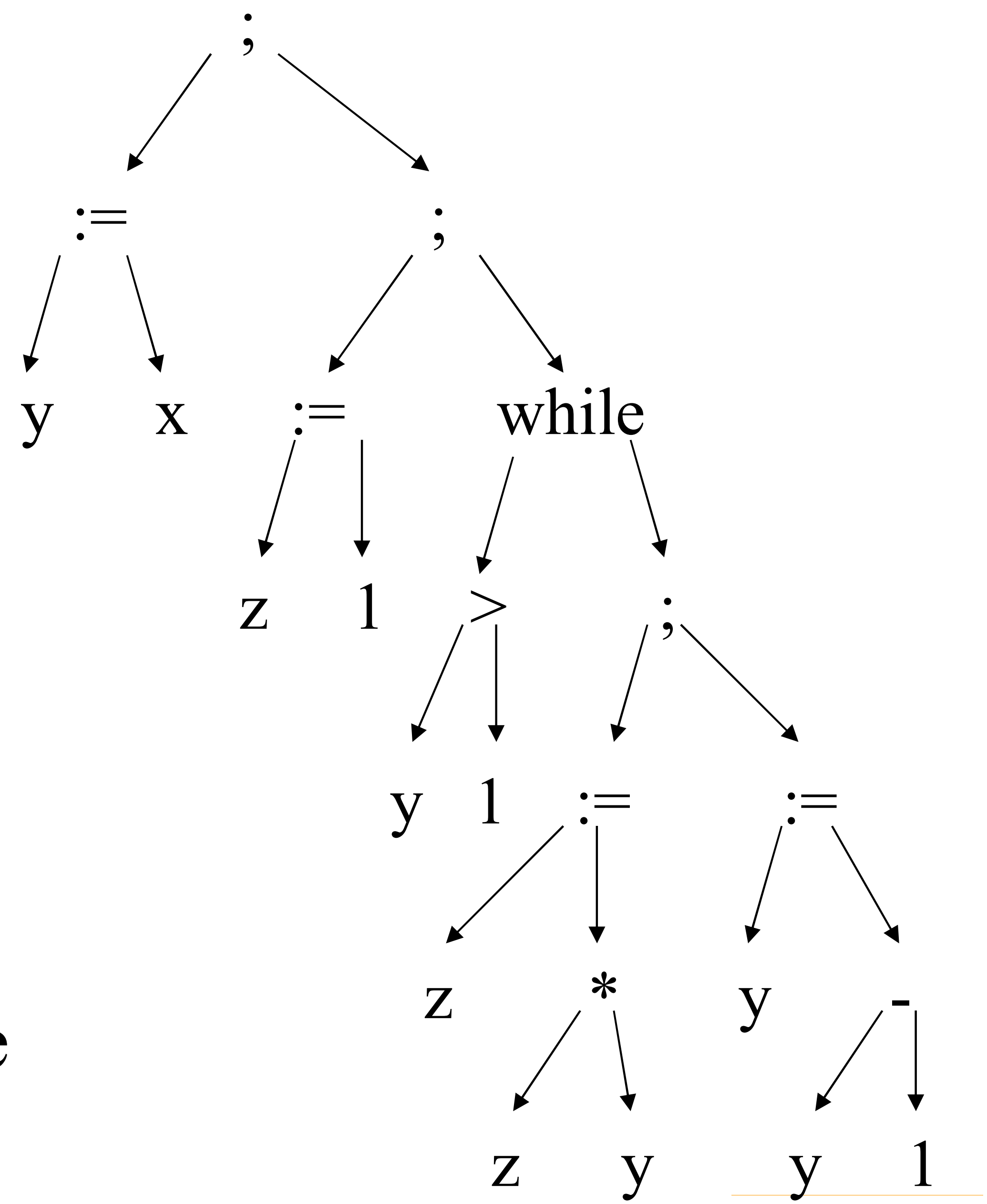
```
y := x;
z := 1;
while y > 1 do
   z := z * y;
   y := y - 1
```

- How do we run this sequence of characters?

Programs as trees

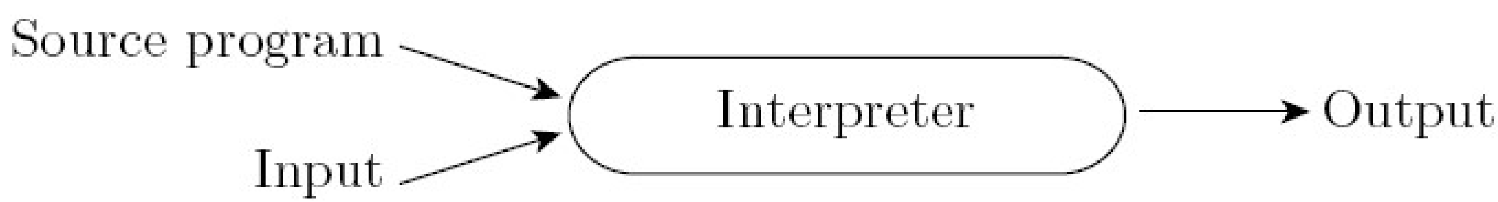
- What if we organize it as a tree in memory

```
y := x;
z := 1;
while y > 1 do
  z := z * y;
  y := y - 1
```



- Now we can walk the tree and execute it

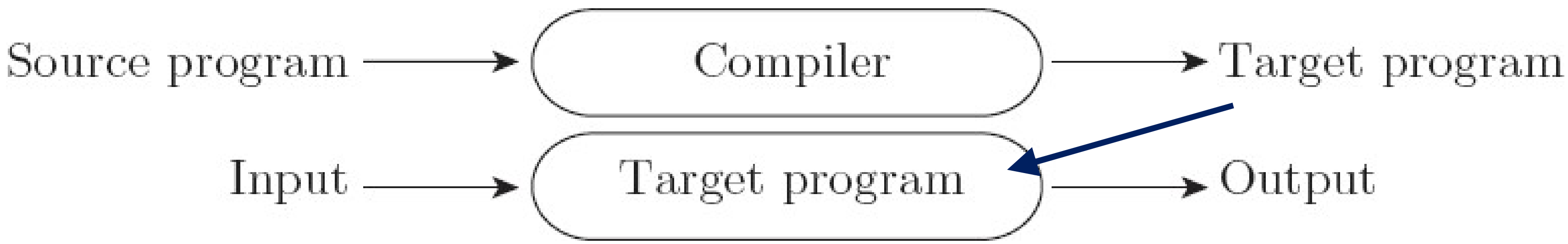
Interpreters



- Interpreter runs at execution time
  - Operates over the program as a data structure
- A simple and flexible approach—but slow
  - We examine the program to determine what to do, over and over again

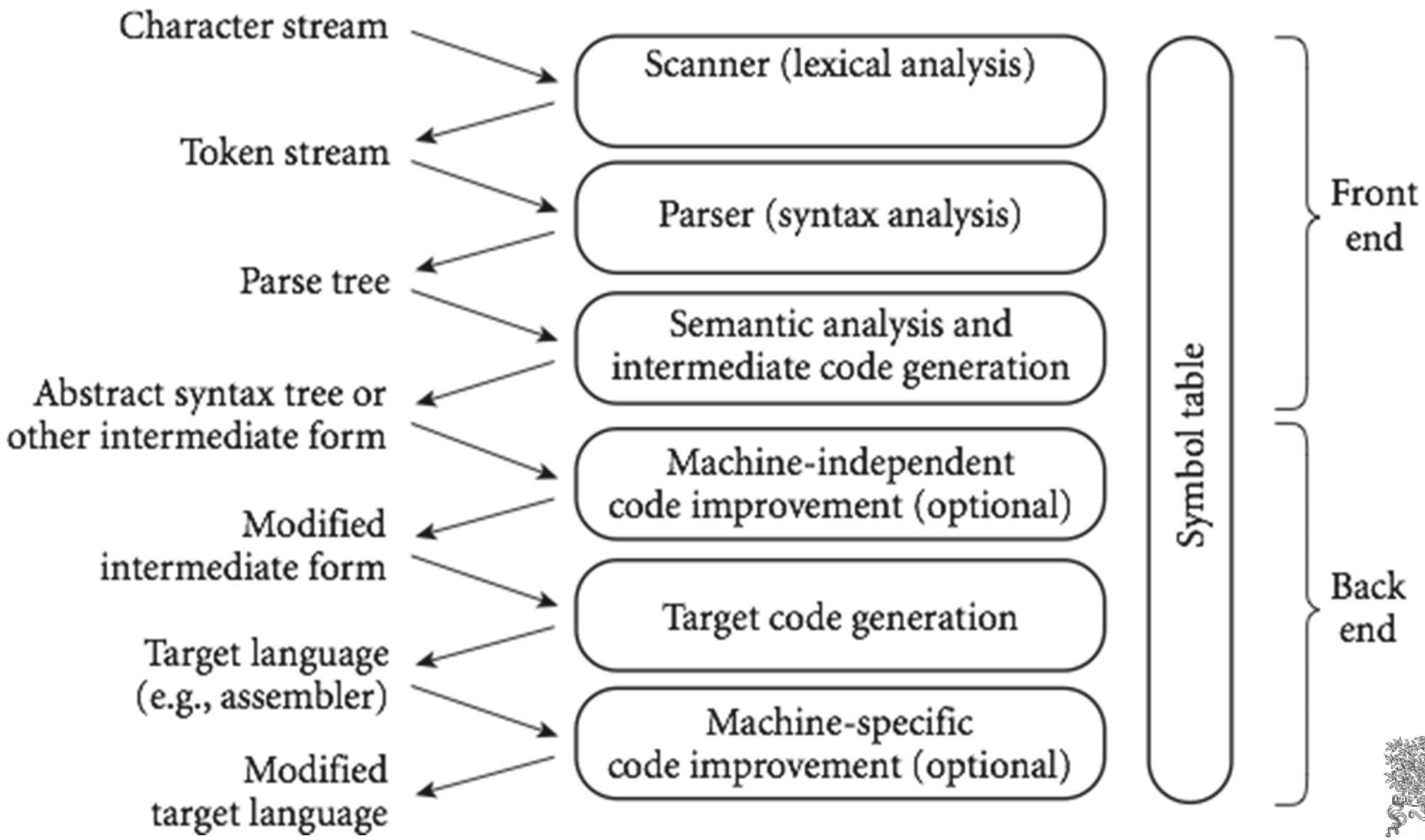
Compilers

- A compiler translates the high-level source program into an equivalent target program (typically in machine language), and then goes away:



An Overview of Compilation

- Phases of Compilation



Programming Language Pragmatics

- PL is an exciting field to study
  - Interesting theory
  - Important impact on practice
  - Lots of applications
  - Will help you become a better programmer
- For next time:
  - Reading: PLP chapter 1
  - Homework "zero" is out today, due Friday. Useful:
    - Rust book chapters 1-6, esp. "Programming a Guessing Game" https://doc.rust-lang.org/book
    - x86 quick references
      - Stanford https://web.stanford.edu/class/archive/cs/cs107/cs107.1196/guide/x86-64.html
      - Brown https://cs.brown.edu/courses/cs033/docs/guides/x64_cheatsheet.pdf