COMPILER CONSTRUCTION · WEEK 1

What is Selfie?

The talk, then the spine of the semester in an hour: the axis, the three theorems, and the definition a compiler class earns.

WHAT THIS CLASS IS FOR

A compiler is a proof system for syntax and a constructor of semantics — and every semantic question it seems to answer is a chosen approximation.

That sentence is the class. Fourteen weeks of building starc, the compiler in selfie, extension by extension, and at each step asking which of the three parts of the sentence the code you just wrote belongs to.

The purpose is the same as in the companion classes: a deep understanding of the basic principles, deep enough to position generative AI, and whatever comes next, properly. A compiler is where notation meets meaning most exactly, so it is the best place to learn what a machine that produces notation can and cannot do.

THE ASSIGNMENTS

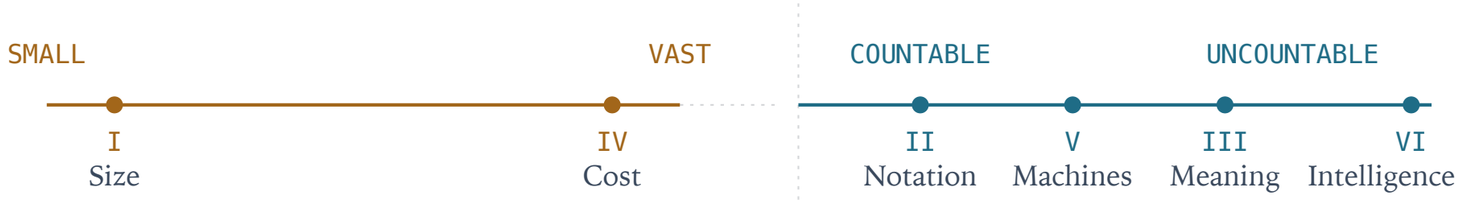
Twelve autograded extensions of selfie, from hexadecimal literals to structs, one per week from week 2. Plus one new one in week 12: verify your own extension with a model checker.

THE BOOK

What is Intelligence? — the Programming chapter is this class; the Meaning and Cost chapters are its theory. Read the Introduction and the Selfie chapter this week.

THE AXIS

One line carries the semester.



the six stations of this book, on its one axis

the six stations, and where a compiler class dwells

Finite, countable. Bits, and everything you can write down: programs, grammars, binaries. Weeks 2 to 5 live here — scanner, parser, code generator — and they never leave the decidable.

Uncountable. What programs mean. Weeks 6 to 9 — statements, procedures, self-compilation, optimisation — are where the compiler constructs meaning and where Gödel, Turing and Rice say what it cannot decide about it.

Cost. Weeks 10 to 12 turn the compiler's output into a formula, hand it to a solver, and meet NP-completeness. Then a week on generated code, and the definition.

THREE COMMANDS, THREE THEOREMS

Understand the first and you understand compilers.

```
$ ./selfie -c selfie.c -o selfie1.m -m 2 -c selfie.c -o selfie2.m && diff -s selfie1.m selfie2.m
Files selfie1.m and selfie2.m are identical
$ ./selfie -c selfie.c -o selfie.m -m 2 -l selfie.m -m 1
$ ./selfie -c selfie.c -o selfie.m -m 3 -l selfie.m -y 2 -l selfie.m -y 1
```

SELF-COMPILATION · A FIXED POINT

The compiler compiles its own source and gets the same bytes twice. Gödel's self-reference, runnable. It proves self-agreement — and week 8 says what it cannot prove.

SELF-EXECUTION · THE UNIVERSAL MACHINE

An emulator that runs every program for its machine, itself included. Turing's move, forwards. The target of every instruction you will emit.

SELF-HOSTING · THE BOOTSTRAP PROBLEM

A hypervisor that must isolate itself from the machines it isolates. The systems class. Here, only the reminder that the loop is the subject.

THE THREE THEOREMS, BRIEFLY

Cantor, Gödel, Rice — what a compiler class needs from each.

CANTOR · 1891

Programs are countable; behaviours are not. So almost every behaviour has no program, and no analysis of programs can list what they do. Semantic analysis is approximation, by counting alone.

GÖDEL · 1931

No strong system certifies itself. A compiler that agrees with itself has certified nothing; the outside check is a second compiler, or a solver. Week 8, with Thompson.

RICE · 1953

Every non-trivial property of behaviour is undecidable; only questions about the text are safe. Type checking is about the text. Dead code is about behaviour. Week 4 and week 9.

All three are one move: assume a complete list, ask each item about itself, answer the opposite. The introduction class proves them; here they are the constraints the compiler is built under.

THE SYSTEM

Nothing in the middle is left out.

and in the same file: in-memory linker · disassembler · garbage collector

L1 caches · profiler · debugger with replay

selfie.c → scanner → parser → code generator → RISC-U

Twelve thousand lines, one file: the compiler starc, the emulator mipster, the hypervisor hypster, and a small library. The compiler is about four thousand of those lines and you will read all of them.

C*: 7 keywords, 22 symbols, LL(1), one type and pointers to it. RISC-U: 14 instructions, 32 registers, 4 GB. Small enough to hold in one head, which is the only argument for the size.

FOURTEEN WEEKS

The route.

WK	LECTURE	ASSIGNMENT
2	Regular languages, FSMs, the scanner	hex-literal
3	Context-free grammars, LL(1), recursive descent: a parse is a proof	–
4	Symbol table, types: type checking is proof checking	bitwise-shift-compilation
5	Code generation for expressions; instruction encoding is Gödel numbering	bitwise-shift-execution
6	Statements: while + assignment is universal	bitwise-and-or-not, logical-and-or-not
7	Procedures, calling convention, recursion; where halting lives	for-loop, lazy-evaluation
8	Self-compilation: the fixed point, trusting trust	array-access
9	Optimisation and Rice; register allocation is NP-hard	array-allocation
10	Semantics as a formula: rotor, BTOR2	array-multidimensional
11	SAT: NP-completeness, DPLL, CDCL, babysat	struct-declaration
12	SMT, bounded model checking with bitme	struct-execution, rotor-check
13	Generated code and the gate	–
14	What a compiler is; the definition	–

THIS WEEK

Take a selfie.

1. Install selfie, run the three commands, and read the grader's README: private clone, upstream, commit links.
2. Do `print-your-name`: make selfie print your name right after initialisation, prefixed like every other status message. Part of the assignment is finding out where.
3. Check your grade with `./grader/self.py print-your-name`, and that `self-compile` still passes.
4. Read the Introduction, the Selfie chapter, and the Programming chapter's opening callout.

NEXT WEEK

The scanner: characters into symbols, one finite state machine per kind of symbol, and your first extension of the grammar — hexadecimal integer literals.