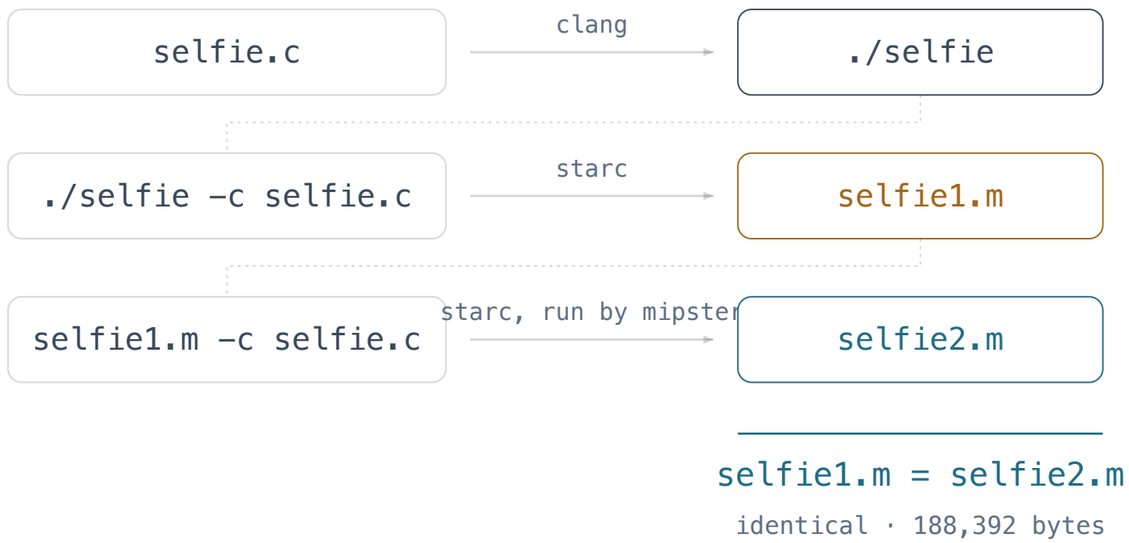
COMPILER CONSTRUCTION · WEEK 8 · STATION III

Self-Compilation

The fixed point, bootstrapping, trusting trust, diverse double-compiling — and what the fixed point proves.

THE FIXED POINT

The compiler compiles its own source and gets the same bytes twice.



selfie1.m = selfie2.m · 188,392 bytes

```
$ make self-self-check
./selfie -c selfie.c -o selfie1.m -s selfie1.s -m 2
…
diff -q selfie1.m selfie2.m
diff -q selfie1.s selfie2.s
```

Stage 0: gcc compiles selfie.c. Stage 1: that selfie compiles selfie.c into selfie1.m. Stage 2: selfie1.m, on mipster, compiles selfie.c into selfie2.m. Identical. 1,228,259,585 instructions in total.

Every assignment in this class has passed this check. Today: what, exactly, did it check?

WHY IT IS A FIXED POINT

The boot levels, and why the bytes must agree.

Selfie1.m and selfie2.m were produced by two different compilers: one compiled by gcc, one compiled by selfie. If both compilers implement the same semantics for C*, they produce the same output for the same input. Selfie.c is the input both times.

So agreement is evidence that gcc's reading of C* and selfie's reading of C* coincide on selfie.c — and, since selfie's reading is *defined* by selfie.c, that the definition is consistent with itself. A dictionary whose entries, when applied to the dictionary, produce the dictionary.

WHAT CAN DIFFER, AND DOES

Filenames in strings. The value of register a6. Anything that depends on the boot level. Selfie is written to keep those out of the binary, which is engineering, not theory — and the reason the assembly is compared as well as the bytes.

TRUSTING TRUST · 1984

The source does not determine the binary.

Thompson's compiler: recognises when it compiles login and inserts a back door; recognises when it compiles a compiler and inserts the code that does both. Remove the trick from the source, recompile with the infected binary — the back door survives, because the source is not what compiled the compiler.

Such a compiler passes `self-self-check`. It agrees with itself perfectly. The identical bytes prove self-agreement and nothing more.

> *You can't trust code that you did not totally create yourself.*
>
> KEN THOMPSON, REFLECTIONS ON TRUSTING TRUST

DIVERSE DOUBLE-COMPILING · WHEELER 2005

Compile the compiler's source with two unrelated compilers, then compile it again with each result. If the source is honest, the two final binaries agree; an infection in either first compiler shows up as a difference. The check comes from *outside*.

THEOREM

No strong system can certify itself.

GÖDEL · SECOND INCOMPLETENESS · 1931

> *A consistent formal system rich enough for arithmetic cannot prove its own consistency.*

Read as engineering. A trustworthy system cannot be the source of its own trust. Confidence comes from outside: a second compiler, a solver, a test on hardware, a person.

Read as a rule for this class. Every self-check — the fixed point, a test suite the compiler compiles, a compiler that verifies its own output — establishes agreement, never correctness. Whenever you build a check, ask what it is outside of.

The first theorem is the companion: there are truths about the compiler that no analysis inside the compiler reaches. The same diagonal as Cantor's, and the same as next week's Rice: assume a complete list, ask each item about itself, answer the opposite.

MEANWHILE, IN THE SOURCE

A compiler that mistranslated one construct would reproduce the mistake perfectly.

Suppose `compile_term` emitted `mul` for `/`. Selfie.c uses division in a few places; the miscompiled selfie would compute wrong results there — and, compiling selfie.c, emit `mul` for `/` again, byte for byte the same as stage 1. The fixed point holds. The compiler is wrong.

What catches it: gcc's selfie disagrees with selfie's selfie at stage 1 — the outside check — or a test that divides, or the model checker of week 12 asked whether the output of `1 / 2` can ever be 2.

TRY IT

Change one emit in your selfie so that the change is invisible to selfie.c — a construct selfie.c never uses — and run the check. It passes. Now add a use of that construct to selfie.c. Which stage disagrees, and why?

ASSIGNMENT

array-access: [and] on heap-allocated arrays.

1. Specify. Extend the grammar so that a factor and the left-hand side of an assignment may be `identifier [ expression ]`, for pointers obtained from `malloc`.
2. Semantics. `a[i]` is `*(a + i)` with the scaling of week 5: the address is the pointer plus eight times the index. Decide it, write it down, then implement it.
3. Generate. A load for an rvalue, a store for an lvalue; the index in a temporary, scaled, added, and freed. Assertions on the temporaries, both paths.
4. `./grader/self.py array-access` and `self-compile`. Then rewrite one pointer expression in selfie.c as an array access and self-compile again — your first use of your own language extension in the compiler that implements it.

THE POINT

From this week on, every extension is expected to be used in selfie.c itself. That is the fixed point as a habit: a feature exists when the compiler compiles itself through it.