A **fully optimizing compiler** $C_{opt}$ transforms a program $P$ into a program $\text{Opt}(P)$ that is the smallest program with the same I/O behavior.

Let $Q$ be a program with no output that does not halt. Then

$$\text{Opt}(Q) = L: \text{goto } L;$$

This implies that we could use $C_{opt}$ to solve the halting problem:

For a program $P$, check if $\text{Opt}(P)$ is just an infinite loop.

Thus, only **optimizing compilers** can exist.
Proposition [Full employment of compiler writers; Rice] :
For any optimizing compiler, there always exists a more optimizing compiler.

Proof Sketch:

Let A be an optimizing compiler. For a program P that does not halt, it holds that $A(P) \neq \text{Opt}(P)$ because otherwise A would be a fully optimizing compiler.

Thus, there exists a compiler B that is more optimizing:

$B(Q) = \text{if } Q = P \text{ then } [L: \text{ goto } L;] \text{ else } A(Q)$
Structure of an Optimizing Compiler

- **frontend** → Optimization → **backend**
- Control Flow Analysis
- Data Flow Analysis
- Transformations
Optimization Techniques

In the following, common optimization techniques are introduced.

Tasks:

1. Develop criteria to distinguish the introduced optimization techniques, e.g., with respect to the optimized structures, required program information, ... .

2. Classify the introduced optimization techniques according to your criteria.