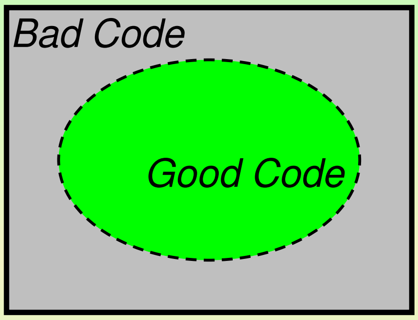
CS520, Handout 12, 5 Oct 2006

Retrospection

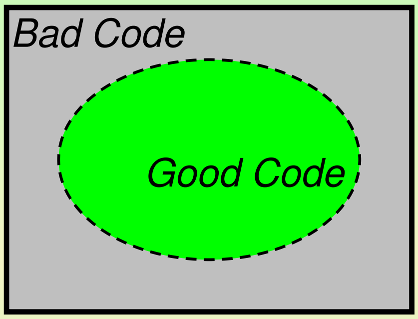
What Are Typing Systems Good For?

Writing code in your favorite programming language ...



What Are Typing Systems Good For?

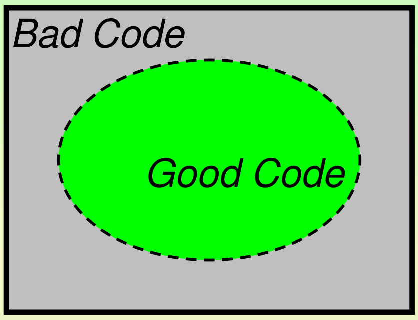
Writing code in your favorite programming language ...



Bad Code =
cannot be compiled, or can be compiled but harmful to run.

What Are Typing Systems Good For?

Writing code in your favorite programming language ...

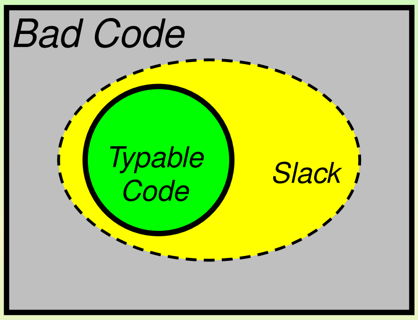


Good Code =

- can be compiled and executed,
- never passes data to a function that does not know how to use it,
- never passes data to a function that is not authorized to use it,
- never passes data to a function that uses it more than once,
- never passes data to a function that over-writes it to memory,
- etc. . . . . . .

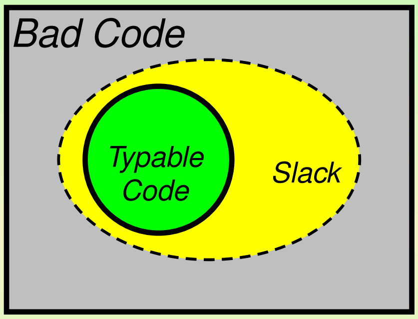
• A type system for your favorite programming language imposes invariants on all executions of the same code.
•These invariants are written in a formal syntax and can therefore be manipulated with the same rigor as any other formal system.

• A type system for your favorite programming language imposes invariants on all executions of the same code.
•These invariants are written in a formal syntax and can therefore be manipulated with the same rigor as any other formal system.



Typable Code =
good code that never breaks formal type-specified invariants.
Slack =
good code that may break formal type-specified invariants.

• A type system for your favorite programming language imposes invariants on all executions of the same code.
•These invariants are written in a formal syntax and can therefore be manipulated with the same rigor as any other formal system.



Typable Code =
good code that never breaks formal type-specified invariants.
Slack =
good code that may break formal type-specified invariants.

The ***slack*** contains all code that is unfairly rejected. The need to minimize it is a major driving force in the development of new type systems.

But the new type systems can become unduly complex, with many bad consequences.

Desirable properties of a type system:

1. Static discipline, i.e., code is judged type-safe at compile time.
2. Types are “easy” to write and insert into programs.
3. Types are “easy” to understand.
4. Type-checking (for fully type-annotated programs) is efficient.
5. Type-inference (for untyped or partially typed programs) is efficient.

Unavoidable tension between these different requirements.