Define *type system*.

"A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute."

What is problematic with the term *dynamically typed language*.

Those "types" are tags on the heap for identifying different kinds of values. They are not static *approximations* of runtime values.

In what sense are type systems *conservative*?

They can only prove the *absence* of some behaviors, never their presence. They therefore must reject some programs that will also lack these behaviors at runtime.

What is the main research goal in the study of type systems?

To allow more programs to be typed by improving the accuracy of static type approximations.

The kinds of bad behaviors a type system eliminates are called ...

... run-time type errors.

What are the benefits of type systems?

- Early detection of some programming errors.
- Maintenance/refactoring.
- Abstraction.
- Documentation.
- Language safety.
- Efficiency.

What makes a language *safe*?

Pierce: A safe language guarantees the integrity of its abstractions.
Cardelli: A safe language traps its errors, meaning they halt computation immediately or raise an exception that can be handled. Unsafe languages have untrapped errors that allow computation to proceed.
Also: A safe language lacks undefined behavior. Or in other words, it's portable between implementations.

Why are there virtually no unsafe dynamically checked languages?

There is little marginal cost to checking the safety of all operations at run-time once most are.

What was the goal of the first type systems for programming languages?

To distinguish between integers and floating-point numbers, for the sake of efficiency.