

# 1.1 Types in Computer Science

Consider the following quote from *TAPL* (p. 1):

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

# 1.1 Types in Computer Science

Consider the following quote from *TAPL* (p. 1):

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

Expanding on the above, type systems:

- are static (compile time);

# 1.1 Types in Computer Science

Consider the following quote from *TAPL* (p. 1):

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

Expanding on the above, type systems:

- are static (compile time);
- are always conservative (some correct programs are rejected);

# 1.1 Types in Computer Science

Consider the following quote from *TAPL* (p. 1):

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

Expanding on the above, type systems:

- are static (compile time);
- are always conservative (some correct programs are rejected);
- only guarantee freedom from certain kinds of misbehavior;

# 1.1 Types in Computer Science

Consider the following quote from *TAPL* (p. 1):

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

Expanding on the above, type systems:

- are static (compile time);
- are always conservative (some correct programs are rejected);
- only guarantee freedom from certain kinds of misbehavior;
- can enforce high-level properties (like respecting data abstractions);

# 1.1 Types in Computer Science

Consider the following quote from *TAPL* (p. 1):

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

Expanding on the above, type systems:

- are static (compile time);
- are always conservative (some correct programs are rejected);
- only guarantee freedom from certain kinds of misbehavior;
- can enforce high-level properties (like respecting data abstractions);
- are structural (typing of whole based on that of parts);

# 1.1 Types in Computer Science

Consider the following quote from *TAPL* (p. 1):

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

Expanding on the above, type systems:

- are static (compile time);
- are always conservative (some correct programs are rejected);
- only guarantee freedom from certain kinds of misbehavior;
- can enforce high-level properties (like respecting data abstractions);
- are structural (typing of whole based on that of parts);
- are automatic—no programmer intervention, except for hints in programs;

# 1.1 Types in Computer Science

Consider the following quote from *TAPL* (p. 1):

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

Expanding on the above, type systems:

- are static (compile time);
- are always conservative (some correct programs are rejected);
- only guarantee freedom from certain kinds of misbehavior;
- can enforce high-level properties (like respecting data abstractions);
- are structural (typing of whole based on that of parts);
- are automatic—no programmer intervention, except for hints in programs;
- are tractable—reasonably efficient.

## 1.2 What Type Systems Are Good For

## 1.2 What Type Systems Are Good For

- Early detection of some errors (but programmer must make good use of type system);

## 1.2 What Type Systems Are Good For

- Early detection of some errors (but programmer must make good use of type system);
- Good maintenance tools;

## 1.2 What Type Systems Are Good For

- Early detection of some errors (but programmer must make good use of type system);
- Good maintenance tools;
- Support modularity;

## 1.2 What Type Systems Are Good For

- Early detection of some errors (but programmer must make good use of type system);
- Good maintenance tools;
- Support modularity;
- Good documentation;

## 1.2 What Type Systems Are Good For

- Early detection of some errors (but programmer must make good use of type system);
- Good maintenance tools;
- Support modularity;
- Good documentation;
- Support efficient code generation.

## 1.2 (Cont.) Language Safety

According to *TAPL* (pp. 6 and 7):

A “safe language” is one that protects its own abstractions.

or

A “safe language” is completely defined by its programmer’s manual.

## 1.2 (Cont.) Language Safety

According to *TAPL* (pp. 6 and 7):

A “safe language” is one that protects its own abstractions.

or

A “safe language” is completely defined by its programmer’s manual.

Discussion points:

- Are the above definitions equivalent?

## 1.2 (Cont.) Language Safety

According to *TAPL* (pp. 6 and 7):

A “safe language” is one that protects its own abstractions.

or

A “safe language” is completely defined by its programmer’s manual.

Discussion points:

- Are the above definitions equivalent?
- Language safety can be achieved statically or dynamically.

## 1.2 (Cont.) Language Safety

According to *TAPL* (pp. 6 and 7):

A “safe language” is one that protects its own abstractions.

or

A “safe language” is completely defined by its programmer’s manual.

Discussion points:

- Are the above definitions equivalent?
- Language safety can be achieved statically or dynamically.
- Can dynamic checks be completely avoided?

## 1.2 (Cont.) Language Safety

According to *TAPL* (pp. 6 and 7):

A “safe language” is one that protects its own abstractions.

or

A “safe language” is completely defined by its programmer’s manual.

Discussion points:

- Are the above definitions equivalent?
- Language safety can be achieved statically or dynamically.
- Can dynamic checks be completely avoided?
- Is language safety absolute?

## 1.2 (Cont.) Language Safety

According to *TAPL* (pp. 6 and 7):

A “safe language” is one that protects its own abstractions.

or

A “safe language” is completely defined by its programmer’s manual.

Discussion points:

- Are the above definitions equivalent?
- Language safety can be achieved statically or dynamically.
- Can dynamic checks be completely avoided?
- Is language safety absolute?
- Is C really unsafe?

## 1.3 Type Systems and Language Design

Let's consider the following quotations from *TAPL* (pp. 9 and 10):

Retrofitting a type system onto a language not designed with typechecking in mind can be tricky; ideally, language design should go hand-in-hand with type system design.

and

... in typed languages, the type system itself is often taken as the foundation of the design and the organizing principle in light of which every other aspect of the design is considered.