Contents

Type Systems and Type Inference

6.1 TYPES IN PROGRAMMING

type

is a collection of computational entities that share some common property. Some examples of types are the type int of integers, the type int→int of functions from integers to integers,

there is no such thing as an untyped programming language.

6.1.1 Program Organization and Documentation

Using types to organize a program

makes it easier for someone to read, understand, and maintain the program. Types therefore serve an important purpose in documenting the design and intent of the program.

An important advantage of type information,

in comparison with comments written by a programmer, is that types may be checked by the programming language compiler. In contrast, many programs contain incorrect comments,

6.1.2 Type Errors

A **type error** occurs when a computational entity, such as a function or a data value, is used in a manner that is inconsistent with the concept it represents.

Hardware Errors.
a machine instruction that results in a hardware error.

If x is an integer variable with value 256, then executing x() will cause the machine to jump to location 256 and begin executing the instructions stored at that place in memory. If location 256 contains data that do not represent a valid machine instruction, this will cause a hardware interrupt.

Unintended Semantics.
compiled code does not contain the same information as the program source code does.

int add(3, 4.5)

is a type error, as int add is an integer operation and is applied here to a floating-point number. Most hardware would perform this operation. int add is intended to perform addition, but the result of int add(3, 4.5) is not the arithmetic sum of the two operands.

6.1.3 Types and Optimization

Type information in programs can be used for many kinds of optimizations.

Some operations can be computed more efficiently if the type of the operand is known at compile time.

6.2 TYPE SAFETY AND TYPE CHECKING

6.2.1 Type Safety

A programming language is **type safe** if no program is allowed to violate its type distinctions. a function has a different

type from an integer. Therefore, any language that allows integers to be used as functions is not type safe.

form of type error

Type Casts.
Type casts allow a value of one type to be used as another type.

Pointer Arithmetic.
an assignment like x = *(p+i) may store a value of one type into a variable of another type and therefore may cause a type error.

Explicit Deallocation and Dangling Pointers.
the location reached through a pointer may be deallocated (freed) by the programmer. This creates a **dangling pointer,** If p is a pointer to an integer, then after we deallocate the memory referenced by p, the program can allocate new memory to store another type of value. This new memory may be reachable through the old pointer p,

6.2.2 Compile-Time and Run-Time Checking

Run-Time Checking.
the compiler generates code so that, when an operation is performed, the code checks to make sure that the operands have the correct type. **An advantage** of run-time type checking is that it catches type errors. **A disadvantage** is the run-time cost associated with making these checks.

Compile-Time Checking.
reject programs that do not pass the compile-time type checks. **An advantage** is that it catches errors earlier than run-time checking does: compiletime checking can make it possible to produce more efficient code.

Conservativity of Compile-Time Checking.
A property of compile-time type checking is that the compiler must be conservative. This mean that compile-time type checking will find all statements and expressions that produce run-time type errors, but also may flag statements or expressions as errors even if they do not produce run-time errors.

Combining Compile-Time and Run-Time Checking.
Most programming languages actually use some combination of compile-time and run-time type checking.

6.3 TYPE INFERENCE

Type inference is the process of determining the types of expressions based on the known types of some symbols that appear in them.

difference between type inference and compile-time type checking

is really a matter of degree. A **type-checking** algorithm goes through the program to check that the types declared by the programmer agree with the language requirements. In **type inference,** the idea is that some information is not specified, and some form of logical inference is required for determining the types of identifiers from the way they are used.

6.3.2 Type-Inference Algorithm

The ML type-inference algorithm uses the following three steps:

1. A assign a type to the expression and each subexpression. For any

compound expression or variable, use a type variable. For **known operations or constants,** such as + or 3, use the type that is known for this symbol.

2. Generates a set of constraints on types, using the parse tree of the expression.
3. Solve these constraints by means of unification, which is a substitution-based algorithm for solving systems of equations.

6.4 POLYMORPHISM AND OVERLOADING

Polymorphism,

which literally means "having multiple forms," refers to constructs that can take on different types as needed.

forms of polymorphism

parametric polymorphism,

in which a function may be applied to any arguments whose types match a type expression involving type variables;

ad hoc polymorphism,
another term for overloading, in which two or more implementations with different types are referred to by the same name;

subtype polymorphism,
in which the subtype relation between types allows an expression to have many possible types.

6.4.1 Parametric Polymorphism

main characteristic
the set of types associated with a function or other value is given by a type expression that contains type variables.

In parametric polymorphism,
a function may have infinitely many types, as there are infinitely many ways of replacing type variables with actual types.

explicit parametric polymorphism,
the program text contains type variables that determine the way that a function or other value may be treated polymorphically.

implicit parametric polymorphism
programs that declare and use polymorphic functions do not need to contain types

6.4.3 Overloading

A symbol is **overloaded** if it has two (or more) meanings, distinguished by type, and resolved at compile time.