Compiler Construction

What is type-checking?

Preliminary remark (1/3)

Consider the MIPS ASM fragment:

```
add $1, $2, $3
```

Types are not necessary!
Assembly language is untyped

Preliminary remark (2/3)

- There is none at machine/assembly level operators are “typed” though
- There are type-less languages e.g., in Tcl or M4 everything is a string

Preliminary remark (3/3)

- It does not make sense to add a function pointer and an integer in C
- It does make sense to add two integers
- But both have the same assembly language implementation!

Goal of type-checking

Reject impossible values!

Types are usefull

- More control from the compiler
- Catching “impossible but expressible” situations
- Optimizing
- Abstraction (arrays, records, etc.)
- Memory management (automatic or not)
- Violations of abstraction boundaries, such as using a private field from outside a class

Russel's Paradox

Russel's Paradox

$$E = \{x \notin x\} \qquad E \in E \qquad E \notin E$$

Based on the conjunction of:

- Any predicate is an object
- Any predicate can be applied to any object

Rejecting one leads to:

- Type theory (1909)
- Zermelo Fraenkel's set theory (1922)

Types

The data types of a language are a large part of what determines that language's style and usefulness (along with control structures).

- **Primitive (built-in) types**: data type provided by a programming language.
- **Composite types**: recursively constructed starting from primitive types

Types in Tiger

- int
- string
- User-defined structure
- Arrays
- **Void?**

Types in some real language

- Numerics
- Booleans
- User-defined enumerations
- Subranges
- Arrays (static, stack dynamic, heap dynamic)
- Unions (discrimated/free)
- Structures/Records/Objects
- Tuples/Lists
- References/Pointers
- etc.

Type Checking

Type Checking is the activity of ensuring that the operands of an **operator** are of compatible types

A compatible type is one that is

- either legal for the operator
- **or** allowed under language rules to be implicitly converted by compiler-generated code (or the interpreter) to a legal type

Coercion

Coercion is the automatic (implicit) conversion from a type to another.

There are 2 kind of coercions:

- widening convertions: from a ”smaller” type to a ”larger one”

```
int i = 42;
float f = i;
```

- narrowing convertions: from a ”larger” type to a ”smaller one”

```
float f = 42.0;
int i = f;
```

Java only allows only widening coercions.

Strong Typing (1/2)

Strong Typing

A programming language is strongly typed if type errors are always detected.

- The types of all operands can be determined, either at **compile time** or at **runtime**
- At run time, detection of incorrect type values in variables that can store values of more than one type

Strong Typing (2/2)

- Ada is nearly strongly typed due to *Unchecked_Conversion*
- C and C++ are not strongly typed languages because both include union types
- F♯ and ML are strongly typed

Type Equivalence (1/2)

Two types are equivalent if an operand of one type in an expression is substituted for one of the other type, without coercion.

In other words, Type equivalence is a strict form of compatibility type compatibility without coercion.

Type Equivalence (2/2)

- Name type equivalence two variables have equivalent types if they are defined either in the same declaration or in declarations that use the same type name

```
int i = 42; int j = 51;
```

- Structural type equivalence two variables have equivalent types if their types have identical structures

```
struct A { int a; int b;};
struct B { int c; int d;};
```

Static Typing vs. Dynamic Typing

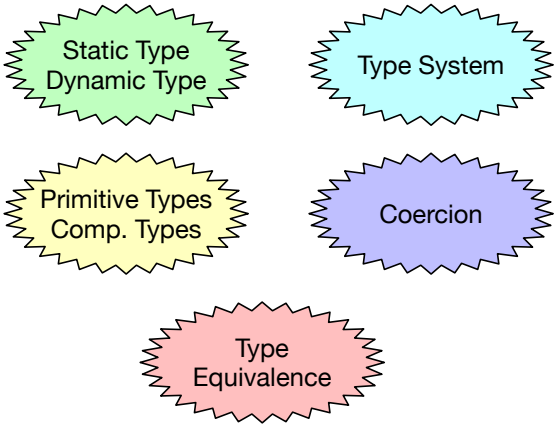
- Statically typed languages: all or almost all type checking occurs at compilation time.
  - C, Java, etc.
- Dynamically typed languages: almost all checking of types is done as part of program execution.
  - Scheme
- Untyped languages: no type checking
  - Assembly, Machine code

Static and Dynamic Types

- The dynamic type of an object is the class C that is used in the *new* $C()$ expression that construct the object
  - Runtime notion
  - Even langages that are not statically typed have the notion of dynamic types

- The static type of an object is a notation that encapsulates all possible types the expression could take.
  - Compile-time notion

Summary


Static Type
Dynamic Type
Type System
Primitive Types
Comp. Types
Coercion
Type
Equivalence