Type Checking

66.648 Compiler Design Lecture (02/25/98)

Computer Science

Rensselaer Polytechnic

Lecture Outline

- Types and type checking
- Poly-morphic functions
- Administration

Errors

We are in chapter 6 of the text book. Please read

that chapter.

Compiler must check that the source program follows both the syntactic and semantic conventions of the source language.

Static checks - reported during the compilation phase.

Dynamic checks - occur during the execution of the program.

Static Checks

1. Type checks: A compiler should report an error if an operator is applied to an incompatible operand.
2. Flow of control checks: Places for transfer of control should be specified.
3. Uniqueness checks: An object must be defined exactly (like the type of an identifier, statements inside a case/switch statement.)
4. Name Related checks: Same name must appear two or more times.

Example

Procedure foo

end foo;

Type information gathered by a type checker may be needed when code is generated.

In general operators could be overloaded. (What is the case in Java?)

Type System

The design of a type checker for a language is based on the information about the syntactic constructs in the language.

Each expression has a type associated with it.

Types are either basic or constructed. Basic types are atomic with no internal structure as far as the program is constructed. Pointers, arrays, functions, structures can be treated as constructor types.

Type Expressions

A type expression is either a basic type or is formed by applying an operator called type constructor to other expressions.

1. A basic type is a type expression. “void” is also a basic type.
2. A type name is a type expression.
3. A type constructor applied to a type expression is a type expression.

Constructors

1. Array : If T is a type expression, then array(I,T) is a type expression, where I is an index set.
2. Products: If T_1 and T_2 are type expressions, then so is T_1 x T_2
3. Structures: differs from the products as fields have names. The struct type constructor applied to tuple formed from field names and field types.
4. Pointers: If T is a type expression, then pointer(T) is a type expression.

Constructors -Contd

5. Functions: Functions in a programming language map a domain type D to a range type R. Often, for implementation reasons, there are limitations on the types a function can return.
6. Objects: Objects are also type. We will talk about objects in the next lecture.

Type Expressions may contain variables whose values are type expressions.

Type Systems

A type system is a collection of rules for assigning type expressions to the various parts of a program. A type checker implements a type system.

The type of an array includes the index set of an array, so a function with an array argument can only be applied to arrays with that index set.

Static and Dynamic Checking of Types

A sound type system eliminates the need for dynamic checking for type errors. A language is strongly typed if its compiler can guarantee that the programs it accepts will execute without type errors.

E.g. array bound checking.

Error Recovery

Since type checking has the potential for catching errors in programs, it is important to do something when an error is discovered.

The inclusion of error handling may result in a type system that goes beyond the one needed to specify correct programs.

Specifications of a simple type checker

The type of each identifier must be declared before the identifier is used. The type checker is a translation scheme that synthesizes the type of each expression from the type of its sub-expressions.

P --> D; E

D--> D; D | id: T

T--> char | integer | array[num] of T | * T

E--> literal | num | id | E mod E | E[E] | E*

Base Types: char, integer, type-error

Translation Scheme

P --> D; E

D--> D;D

D--> id :T { addtype(id.entry,T.type);}

T--> char {T.type= char;}

T--> integer {T.type=integer;}

T-->*T_1 {T.type=pointer(T_1.type);}

T--> array[num] of T_1 { T.type =array(1..num.val,T_1.type); }

Type Checking of Expressions

E--> literal { E.type=char;}

E--> num { E.type =integer;}

E--> id { E.type =lookup(id.entry);}

E--> E_1 mod E_2 { E.type =If (E_1.type ==integer) if (E_2. Type ==integer) integer; else type-error;

E--> E_1[E_2] { E.type=if ((E_2.type==integer)&& (E_1.type==array(s,t)) t; else type-error;}

E--> *E_1 { E.type = if (E_1.type ==pointer(t)) t else type-error;

Type Checking for Statements

S--> id=E { if (id.type==E.type) void; else type-error;}

S--> if E then S { if (E.type==boolean) S_1.type; else type-error;}

S--> While E do S { if (E.type==boolean) S_1.type; else type-error;

S--> S; S; { if (S_1.type==void) if (S_2.type ==void) void; else type-error;}

Type Checking of Functions

E--> E(E)

T--> T '->' T { T.type = T1.type -> T2.type}

E--> E(E) {E.type = I f ((E_2.type ==s) && (E_1.type == s--> t)) t; else type-error;

Comments and Feedback

Project 2 is out. Please start working. PLEASE do not wait for the due date to come.

We are in chapter 6. We will do the rest of chapter 6 in the next class. Please keep studying this material. It may look difficult.