TYPECHECKING

A compiler must check that the source program follows both syntactic and semantic conventions of the source language. This checking, called static checking, detects and reports programming errors.

Some examples of static checks:

1. Type checks - A compiler should report an error if an operator is applied to an incompatible operand.Example: Ifanarrayvariableandfunctionvariableareaddedtogether.

2. Flow-of-control checks - Statements that cause flow of control to leave a construct must have some place to which to transfer the flow of control. Example: An enclosing statement, such as break, does not exist in switchstatement.

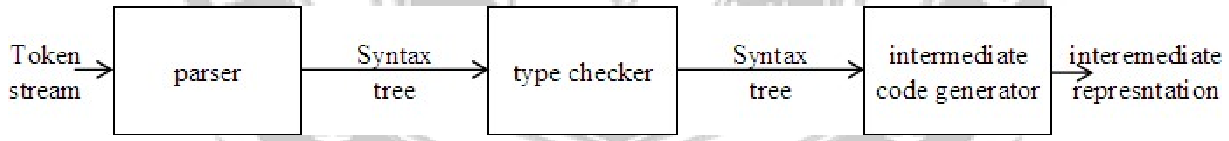


Fig. Position of type checker

A typechecker verifies that the type of a construct matches that expected by its context. For example :arithmetic operator mod in Pascal requires integer operands, so a type checker verifies that the operands of mod have type integer. Type information gathered by a type checker may be needed when code is generated.

Type Systems

The design of a type checker for a language is based on information about the syntactic constructs in the language, the notion of types, and the rules for assigning types to language constructs.

For example : “ if both operands of the arithmetic operators of +,- and * are of type integer, then the result is of type integer ”

Type Expressions

The type of a language construct will be denoted by a “type expression.” A type expression is either a basic type or is formed by applying an operator called a type constructor to other type expressions. The sets of basic types and constructors depend on the language to be checked. The following are the definitions of type expressions:

1. Basic types such as boolean, char, integer, real are typeexpressions.
   A special basic type, type_error , will signal an error during type checking; void denoting “the absence of a value” allows statements to be checked.
2. Sincetypeexpressionsmaybenamed,atypenameisatypeexpression.
3. A type constructor applied to type expressions is a typeexpression.

Constructors include:

Arrays : If T is a type expression then array (I,T) is a type expression denoting the type of an array with elements of type T and index set I.

Products :If T1 and T2 are type expressions, then their Cartesian product T1 X T2 is a type expression.

Records : The difference between a record and a product is that the names. The record type constructor will be applied to a tuple formed from field names and field types.

For example:

```
type row = record
          address: integer;
          lexeme:
          array[1..15
          ] of char
          end;
```

var table: array[1...101] of row;

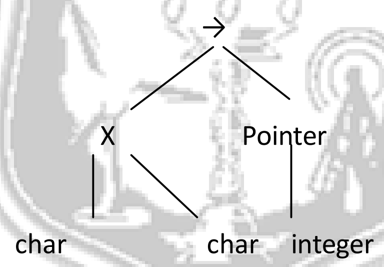
declares the type name row representing the type expression record((address X integer) X (lexeme X array(1..15,char))) and the variable table to be an array of records of this type.

Pointers : If T is a type expression, then pointer(T) is a type expression denoting the type “pointer to an object of type T”.

For example, var p: ↑ row declares variable p to have type pointer(row).

Functions : A function in programming languages maps a domain type D to a range type R. The type of such function is denoted by the type expression D → R

4. Type expressions maycontain variables whose values are type expressions.



Fg. 5.7 Tree representation for char x char → pointer (integer)

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. It is specified in a syntax-directed manner. Different type systems may be used by different compilers or processors of the same language.

Static and Dynamic Checking of Types

Checking done by a compiler is said to be static, while checking done when the target program runs is termed dynamic. Any check can be done dynamically, if the target code carries the type of an element along with the value of that element.

Sound type system

A sound type system eliminates the need for dynamic checking follows us to determine statically that these errors cannot occur when the target program runs. That is, if a sound type system assigns a type other than type_error to a program part, then type errors cannot occur when the target code for the program part isrun.

Strongly typed language

A language is strongly typed if its compiler can guarantee that the programs it accepts will execute without type errors.

Error Recovery

Since type checking has the potential for catching errors in program, it is desirable for type checker to recover from errors, so it can check the rest of the input. Error handling has to be designed into the type system right from the start; the type checking rules must be prepared to cope with errors.