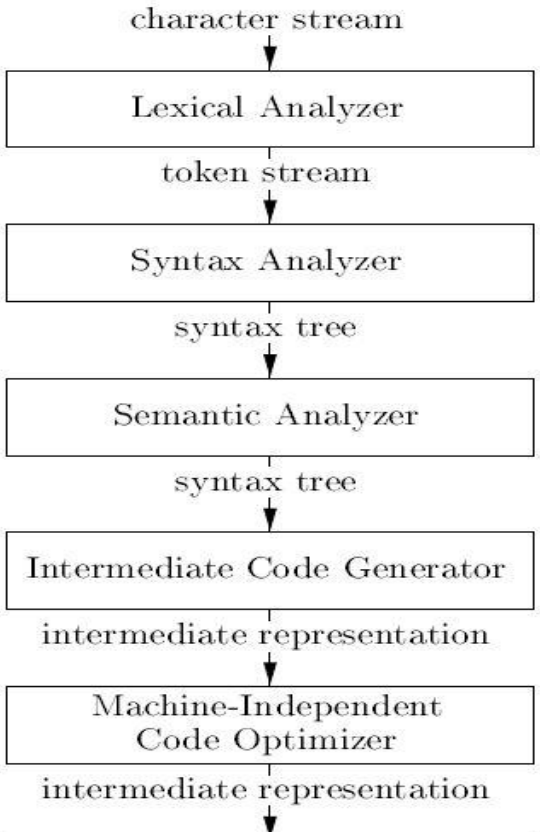
Semantic Analysis II

Tutorial (10)

The Semantic Analyzer

- The semantic analyzer deals with types management and any remaining checks after parsing.
- Type management can be grouped under two categories.
  a. Type checking: ensures that the types of the operands match the type expected by an operator.
  b. Translation Applications: from the type of a name, a compiler can determine the storage that will be needed for that name at run time.

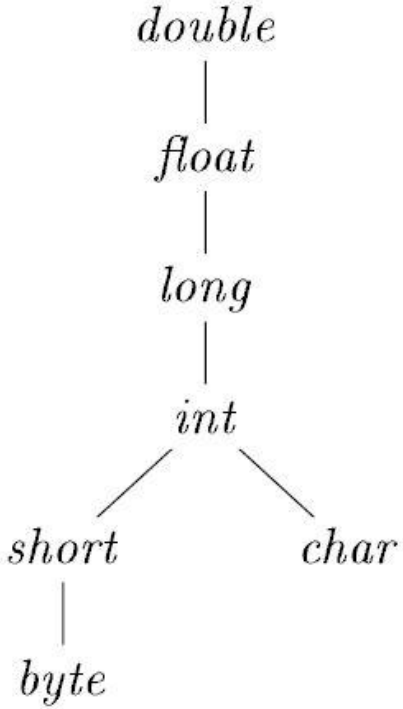


Type Checking

- Type checking is assigning a type expression to each program component and checking that these types observes the type system of the language.
- A strongly typed language does type checking at compile time.
- Type checking can be done by:
  a. Type Synthesis: if f has the type $s \rightarrow t$ and x has type s, then f(x) has the type t. Requires the program to specify types of the basic components of the program.
  b. Type Inference: If f(x) is an expression, there must be types α and β such that f(x) has type $\alpha \rightarrow \beta$ and x has the type α.

Type Synthesis

- The compiler does implicit type conversions called coercions.
- Coercions are limited to widening operations.
- If $E \rightarrow E_1 + E_2$, to synthesize a value for E two functions are used.
  - max(t1,t2): max type according to the hierarchy.
  - widen(a,t,w): widen address a from type t to type w.



Type Synthesis

$$
\begin{aligned}
E \rightarrow E_1 + E_2 \quad \{\ & E.type = max(E_1.type, E_2.type); \\
& a_1 = widen(E_1.addr, E_1.type, E.type); \\
& a_2 = widen(E_2.addr, E_2.type, E.type); \\
& E.addr = \textbf{new}\ Temp\,(); \\
& gen(E.addr\ '='\ a_1\ '+'\ a_2);\ \}
\end{aligned}
$$

```
Addr widen(Addr a, Type t, Type w)
        if ( t = w ) return a;
        else if ( t = integer and w = float ) {
                temp = new Temp();
                gen(temp '=' '(float)' a);
                return temp;
        }
        else error;
}
```

Exercise 10-2

Assume the widen function can handle any type in the widening hierarchy. Suppose that c,d are chars, s,t are shorts, and x is a float. Translate the following.

1. x=s+c
2. x=(s+c)+(t+d)

Type Inference

- Useful for strongly typed languages that does not require the programmer to declare types.
- Exercise 10-3: Consider the following polymorphic function. What is the type of reverse?

```
fun reverse(x) =
    if length(x)==1 then x
    else append(head(x), reverse(tail(x))
```

Translation Applications

- SDT for the static memory management of a sequence of variable declarations.
- Exercise 10-1: Determine the types and relative addresses for the identifiers in the following sequence of declarations:

  float x;

  record {float x; float y;} p;

  record {int tag; float x; float y;} q;

$$
\begin{aligned}
&P \rightarrow \{offset = 0\} \quad D \\
&D \rightarrow T\ id; \quad \{table.put(id.lexeme, T.type, offset) \\
&\qquad offset+ = T.width\} \\
&\quad D_1 \\
&D \rightarrow \varepsilon \\
&T \rightarrow int \quad \{T.type = int \quad T.width = 4\} \\
&T \rightarrow float\ \{T.type = float \quad T.width = 8\} \\
&T \rightarrow record\{ \quad \{Stack.push(table) \\
&\qquad\qquad\qquad table = newTable() \\
&\qquad\qquad\qquad Stack.push(offset) \\
&\qquad\qquad\qquad offset = 0\} \\
&D \quad \{T.type = record(table) \\
&\qquad T.width = offset \\
&\qquad offset = Stack.pop() \\
&\qquad table = Stack.pop()\}\} \quad \}
\end{aligned}
$$