# Semantic Analysis

Sudakshina Dutta

IIT Goa

$30^{th}$ March, 2022

# Type Checking

- To do type checking, compiler assigns types to expressions of the program
- Next, compiler determines type expressions conform to the rules given for the programming language
- An implementation of a language is strongly typed if a compiler guarantees that the program it accepts will run without type errors

# Type synthesis

- Type checking can take on two forms: synthesis and inference
- Type synthesis builds the type of an expression from the sub-expressions
  - The type of $E_1$ and $E_2$ determines the type of $E_1 + E_2$
- Every non-terminal is associated with attributes
- Every terminal is associated with lexical values

# Type inference

- Type inference determines the type of a language from the way it is used
- From the usage of $null(x)$, we can tell that $x$ is a list
- A typical rule –

if $f(x)$ is an expression,
then for some $\alpha$ and $\beta$, $f$ has type $\alpha \rightarrow \beta$ and $x$ has type $\alpha$

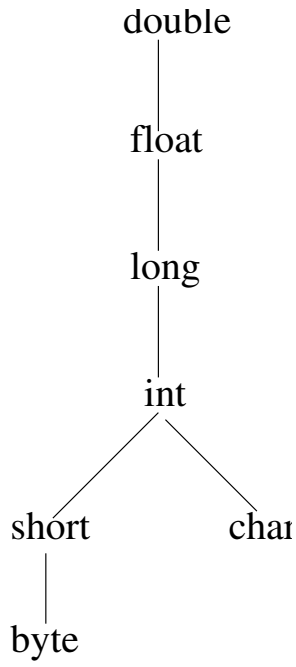
# Type Conversion

- Consider the expression like $x + i$ where $x$ is float and $i$ is integer
- As different machine instructions are used for operations on integers and float, compiler needs to convert one operand so that both are of the same type
- Example :
  $t_1 = 2,\ t_2 = t_1 * 5.2$
  Here compiler needs to convert 2 to float for computing $t_1 + t_2$
- A typical rule for $E \rightarrow E_1 + E_2$

  $\text{if}(E_1.type = integer \text{ and } E_2.type = integer) E.type = integer;$
  $\text{else if } (E_1.type = float \text{ and } E_2.type = integer) \cdots$

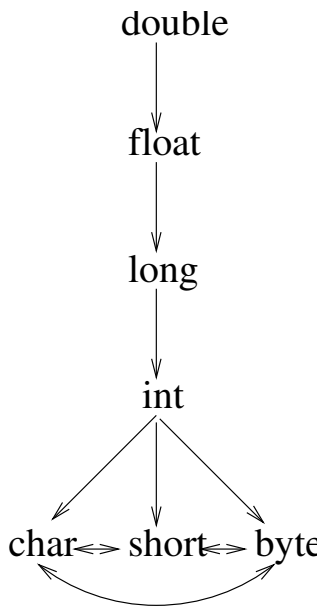
# Widening conversion

- Varies from language to language
- Widening conversion is intended to preserve the information
- Any type lower in the hierarchy can be widened to higher type



# Narrowing conversion

- A type $s$ can be narrowed to a type $t$ if there is a path from $s$ to $t$
- A *char* cannot be widened to a *short*
- Note that *char*, *short*, and *byte* are pairwise convertible to each other



# Type conversion

- Conversion from one type to another type is done automatically by the compiler — *implicit conversion*
- Implicit conversion is called coercion
- Conversion is explicit if programmer writes something to cause the conversion

# Type conversion

The semantic action for checking $E \rightarrow E_1 + E_2$ uses two functions:

- $max(t_1, t_2)$ takes two types $t_1$ and $t_2$ and returns the maximum of the two types in the widening hierarchy. Declares error if they are not in hierarchy e.g., if either type is array or a pointer type
- $widen(a, t, w)$ generates type conversion if needed to widen the content of an address $a$ of type $t$ into a value of type $w$. It return $a$ itself if $t$ and $w$ are of the same type. Otherwise, it generates an instruction to do the conversion and place the result in temporary which is returned as result

**How are these all achieved ?**

# Attribute Translation Grammar

- Context-free grammars with program fragments embedded in it
- Program fragments are called *semantic actions*
- Every non-terminals are associated with attributes and terminals are associated with lexical values