Type Expressions

Language to define types.

\[ \text{Type} \rightarrow \text{int} \mid \text{float} \mid \text{char} \ldots \]

| void         |
| error        |
| name         |
| array(size, Type) |
| record( (name, Type)* ) |
| pointer( Type) |
| tuple( (Type)* ) |
| arrow( Type, Type) |

Examples of Type Expressions

- \( \text{float xform}[3][3]; \)
  \( \text{xform} \in \text{array}(3, \text{array}(3, \text{float})) \)
- \( \text{char *string}; \)
  \( \text{string} \in \text{pointer}(\text{char}) \)
- \( \text{struct list} \{ \text{int element; struct list *next; } \} 1; \)
  \( \text{list} \equiv \text{record}( (\text{element, int}), (\text{next, pointer(list)})) \)
  \( 1 \in \text{list} \)
- \( \text{int max(int, int);} \)
  \( \text{max: int} \times \text{int} \rightarrow \text{int} \)
  \( \text{max} \in \text{arrow( tuple( int, int), int)} \)

Type Checking: An Example

\[ E \rightarrow \text{int const} \quad \{ \text{E.type = int}; \} \]
\[ E \rightarrow \text{float const} \quad \{ \text{E.type = float}; \} \]
\[ E \rightarrow \text{id} \quad \{ \text{E.type = symbol(id, entry, type)}; \} \]
\[ E \rightarrow \text{E1 + E2} \quad \{ \]
  \( \text{if E1.type} \equiv \text{E2.type} \equiv \text{int} \)
  \( \quad \text{E.type = int}; \)
  \( \) else
  \( \quad \text{E.type = float}; \)
\( \} \)

Type Checking: Another Example

\[ E \rightarrow \text{int const} \quad \{ \text{E.type = int}; \} \]
\[ E \rightarrow \text{float const} \quad \{ \text{E.type = float}; \} \]
\[ E \rightarrow \text{id} \quad \{ \text{E.type = symbol(id, entry, type)}; \} \]
\[ E \rightarrow \text{E1 + E2} \quad \{ \]
  \( \text{if (E1.type} \in \{\text{int, float}\}) \) OR
  \( \quad \text{E2.type} \in \{\text{int, float}\} \) \)
  \( \quad \text{E.type = error}; \)
  \( \) else \( \text{if E1.type} \equiv \text{E2.type} \equiv \text{int} \)
  \( \quad \text{E.type = int}; \)
  \( \) else
  \( \quad \text{E.type = float}; \)
\( \} \)
Function Application

\[ E \rightarrow E_1 E_2 \quad \text{if } E_1 \text{type} \equiv \text{arrow}(S, T) \ \text{AND} \]

\[ E_2 \text{type} \equiv S \]

\[ E, \text{type} = T \]

\[ \text{else} \]

\[ E, \text{type} = \text{error} \}

\[ E \rightarrow (E_1, E_2) \quad \{ \text{E.type} = \text{tuple}(E_1 \text{type}, E_2 \text{type}) \}

Type Equivalence

When are two types “equal”?

- Name Equivalence: When they have the same name, \( x \) and \( y \) have same type, but \( z \) has different type.

- Structural Equivalence: When they have the same structure, \( x \), \( y \) and \( z \) have same type.

Type Checking with Type Expressions

\[ E \rightarrow E_1[E_2] \quad \{ \text{if } E_1 \text{type} \equiv \text{array}(S, T) \ \text{AND} \]

\[ E_2 \text{type} \equiv \text{int} \]

\[ E, \text{type} = T \]

\[ \text{else} \]

\[ E, \text{type} = \text{error} \}

\[ E \rightarrow *E_1 \quad \{ \text{if } E_1 \text{type} \equiv \text{pointer}(T) \]

\[ E, \text{type} = T \]

\[ \text{else} \]

\[ E, \text{type} = \text{error} \}

\[ E \rightarrow \&E_1 \quad \{ E, \text{type} = \text{pointer}(E_1 \text{type}) \}

Functions and Operators

Functions and Operators have Arrow types,

- \( \text{max} : \text{int} \times \text{int} \rightarrow \text{int} \)

- \( \text{sort} : \text{numlist} \rightarrow \text{numlist} \)

Functions and operators are applied to operands.

- \( \text{max}(x,y) : \)

\[ \text{max} : \text{int} \times \text{int} \rightarrow \text{int} \]

\[ x : \text{int} \]

\[ y : \text{int} \]

\[ (x,y) : \text{int} \times \text{int} \]

\[ \text{max}(x,y) : \text{int} \]
### Structural Subtyping

\[ S \subseteq T \text{ iff:} \]

- \( S \) and \( T \) are the same basic type;
- \( S = \text{array}(S_1) \), \( T = \text{array}(T_1) \), and \( S_1 \subseteq T_1 \);
- \( S = \text{pointer}(S_1) \), \( T = \text{pointer}(T_1) \), and \( S_1 \subseteq T_1 \);
- \( S = \text{tupled}(S_1, S_2) \), \( T = \text{tupled}(T_1, T_2) \), and \( S_1 \subseteq T_1 \) and \( S_2 \subseteq T_2 \);
- \( S = \text{arrow}(S_1, S_2) \), \( T = \text{arrow}(T_1, T_2) \), and \( S_1 \subseteq T_1 \) and \( T_2 \subseteq S_2 \).

### Subtyping

Object-oriented languages permit subtyping.

```java
class Rectangle {
    private int x, y;
    int area() { ... }
}

class Square extends Rectangle {
    ...
}
```

Square is a subclass of Rectangle.

Since all methods on \text{Rectangle} are inherited by \text{Square} (unless explicitly overridden)

\text{Square} is a subtype of \text{Rectangle}.