Type expression for the function

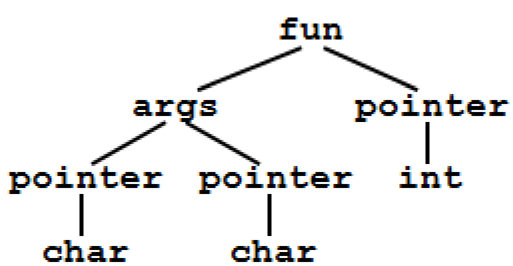
We may treat functions in a programming language as mapping from a domain type D to a range type R. So, the type of a function can be denoted by the type expression *D→R* where D are R type expressions. Ex: int→int represents the type of a function which takes an int value as parameter, and its return type is also int.

Ex: int f(double x, char y) { ... }

f: $\underbrace{\text{double} \times \text{char}}_{\text{argument types}} \rightarrow \underbrace{\text{int}}_{\text{return type}}$

Graph representation

```
int *f(char*,char*)
```



arrays: If T is a type expression, then *array(I,T)* is a type expression where I denotes index range. Ex: array(0..99,int)

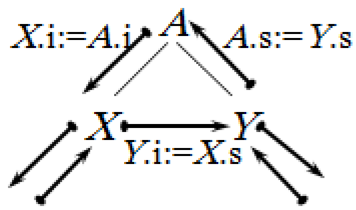
L - Attributed Definitions

Syntax-directed definition is *L-attributed* if each inherited attribute of X_j on the right side of $A \rightarrow X_1 X_2 \ldots X_n$ depends only on

- the attributes of the symbols $X_1, X_2, \ldots, X_{j-1}$
- the inherited attributes of *A*

L-attributed definitions allow for a natural order of evaluating attributes: depth-first and left to right

$A \rightarrow X\ Y$



$X.\mathrm{i} := A.\mathrm{i}$
$Y.\mathrm{i} := X.\mathrm{s}$
$A.\mathrm{s} := Y.\mathrm{s}$

Evaluating L-Attributed

Production	Semantic Rule
$D \rightarrow T\ L$	$L.\mathrm{in} := T.\mathrm{type}$
$T \rightarrow \textbf{int}$	$T.\mathrm{type} :=$ 'integer'
$T \rightarrow \textbf{real}$	$T.\mathrm{type} :=$ 'real'
$L \rightarrow L_1\ \textbf{, id}$	$L_1.\mathrm{in} := L.\mathrm{in};\ \mathit{addtype}(\textbf{id}.\mathrm{entry}, L.\mathrm{in})$
$L \rightarrow \textbf{id}$	$\mathit{addtype}(\textbf{id}.\mathrm{entry}, L.\mathrm{in})$

$D \rightarrow T\ \{\ L.\mathrm{in} := T.\mathrm{type}\ \}\ L$
$T \rightarrow \textbf{int}\ \{\ T.\mathrm{type} := \text{'integer'}\ \}$
$T \rightarrow \textbf{real}\ \{\ T.\mathrm{type} := \text{'real'}\ \}$
$L \rightarrow \{\ L_1.\mathrm{in} := L.\mathrm{in}\ \}\ L_1\ \textbf{, id}\ \{\ \mathit{addtype}(\textbf{id}.\mathrm{entry}, L.\mathrm{in})\ \}$
$L \rightarrow \textbf{id}\ \{\ \mathit{addtype}(\textbf{id}.\mathrm{entry}, L.\mathrm{in})\ \}$

Lexical Errors

Though error at lexical analysis is normally not common, there is possibility of errors. When the error occurs the lexical analyzer must not halt the process. So it can print the error message and continue. Error in this phase is found when there are no matching string found as given by the pattern. Some error recovery techniques includes like deletion of extraneous character, inserting missing character, replacing incorrect character by correct one, transposition of adjacent characters etc. Lexical error recovery is normally expensive process. For e.g. finding the number of transformation that would make the correct tokens.