Semantic Analysis

Semantic analysis

- Goals
  - Connects variable definitions to their uses,
  - checks that each expression has a correct type, and
  - translate the abstract syntax into a simple representation suitable for generating machine code.

SYMBOL TABLES

- Symbol Tables
  - Mapping identifiers t their types and locations.
- Each local variables in a program has a scope in which it is visible.
- An environment is a set of bindings, denoted $\rightarrow$ arrow.

  Ex. The bindings {g$\rightarrow$int,a $\rightarrow$ int}

Program	***Environment***
	σ_0
1.Function f(a:int,b:int,c:int}	$\sigma_1 = \sigma_0 + \{a \rightarrow int, b \rightarrow int, c \rightarrow int\}$
2.(	
3. print_int(a+c);	σ_1
4. let var j:=a+b;	$\sigma_2 = \sigma_1 + \{j \rightarrow int\}$
5. var a:="hello"	$\sigma_3 = \sigma_2 + \{a \rightarrow string\}$
6. in print(a);print_int(j);	σ_3
7. end;	σ_1
8. print_int(b);	σ_1
9.)	σ_0

We say that X+Y for table is not the same as Y+X;

bindings in the right-hand table override those in the left.

How to implement ?

- Two choices
  - functional style
    - We make sure to keep σ_0 in pristine condition while we create σ_1 and σ_2 .Then when we need again, it's rested and ready.
  - Imperative style
    - We modify σ_1 until it becomes σ_2. The destructive update "destroys" σ_1; while σ_2 exists, we cannot look thing up in σ_1.
    - But where we are done with σ_2, we can undo the modification to get σ_1 back again.

MULTIPLE SYMBOL TABLES

- In some languages there can be several environments at once, each module or class or record, in the program has a symbol table σ of its own.

An example In Java

```
package M;
class E {
    static int a=5;
}
class N {
    static int b=1-;
    static int a=E.a+b;
}
class D {
    static int d=E.a+N.a;
}
```

In JAVA, ***forward reference*** is allowed , so N and D are both compiled in the environment σ_7

The result is still $\{M \rightarrow \sigma_7\}$

An example In ML

```
Structure M = struct
    structure   E= struct
        val a=5;
    end
    structure N = struct
        val b=10
        val a=E.a+b
    end
    structure D= struct
        val d=E.a+N.a
    end
end
```

σ_0 : Base environment

$\sigma_1=\{a \to \text{int}\}$

$\sigma_2 =\{E \to \sigma_1 \}$

$\sigma_3 =\{b \to \text{int}, a \to \text{int}\}$

$\sigma_4 =\{N\to \sigma_3 \}$

$\sigma_5 =\{d \to \text{int}\}$

$\sigma_6 =\{D \to \sigma_5 \}$

$\sigma_7 = \sigma_2 + \sigma_4 + \sigma_6$

The N is compiled using environment $\sigma_0 + \sigma_2$

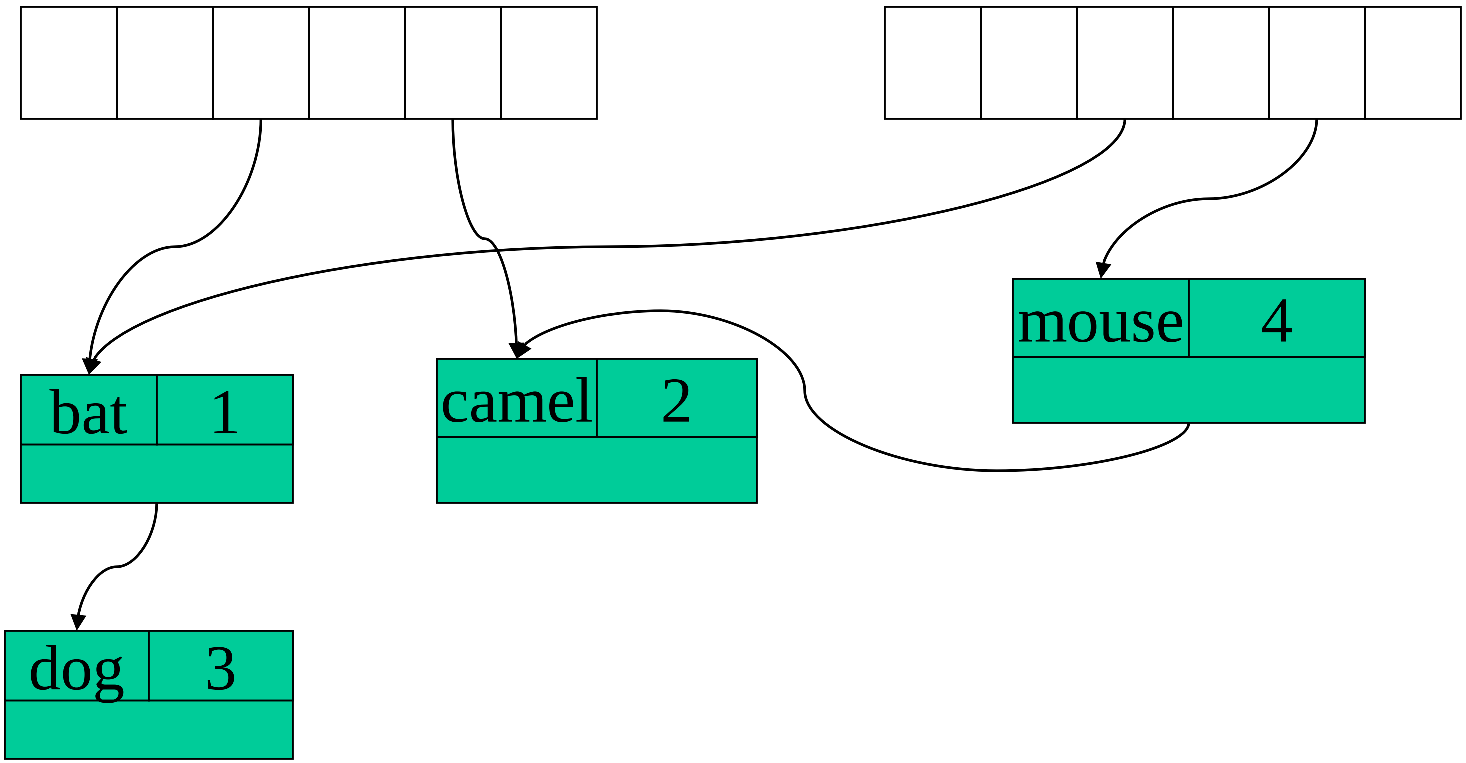
The D is compiled using environment $\sigma_0 + \sigma_2 + \sigma_4$

The result of the analysis is $\{M \to \sigma_7 \}$

EFFICIENT IMPERATIVE SYMBOL TABLES

- Usually implemented using hash tables.
- The operation $\sigma' = \sigma + \{a \to \tau\}$ be implemented by inserting τ in the hash table with key a.
- A simple hash table with external chaining work well and supports deletion easily to recover σ at the end of the scope of a.

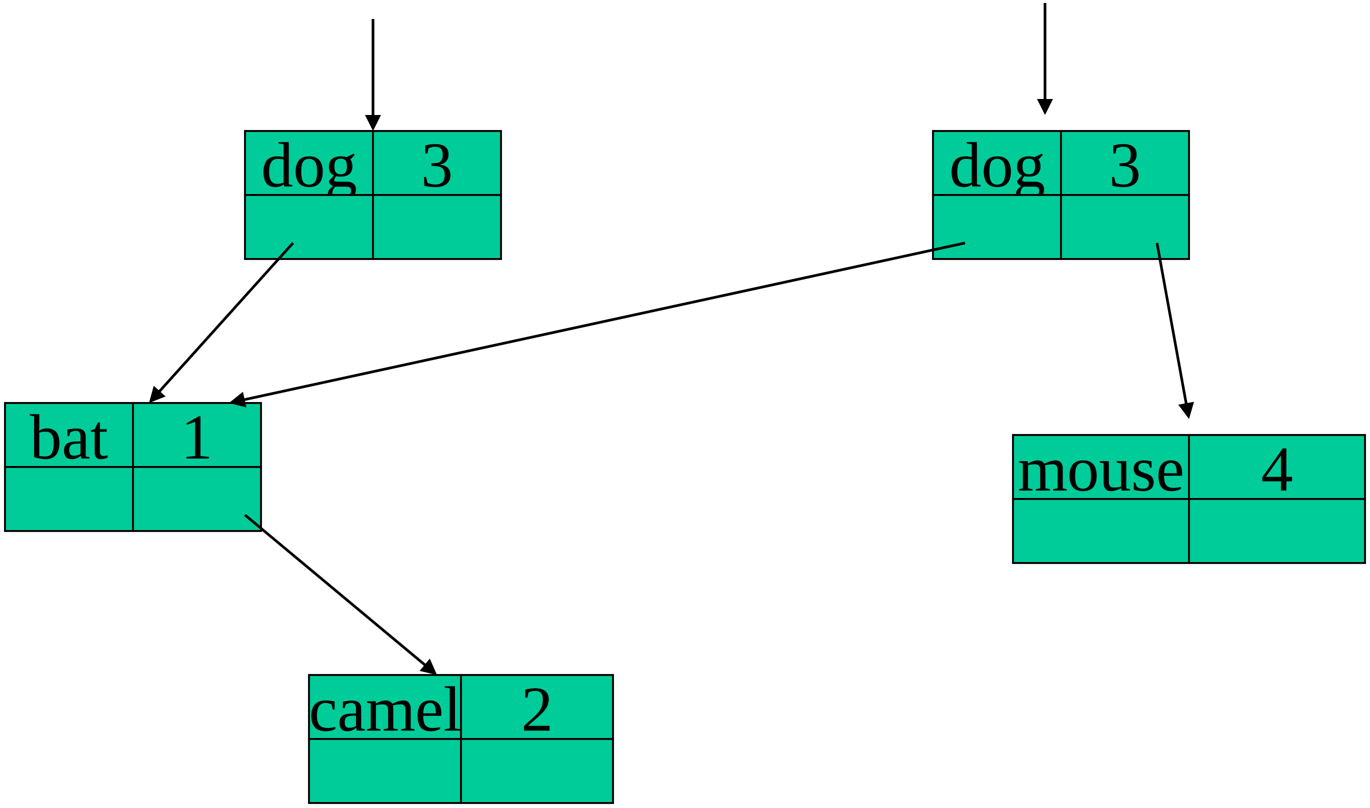
Hash Tables



EFFICIENT FUNCTIONAL SYMBOL TABLES

- In the functional style, we wish to compute $\sigma' = \sigma + \{a \rightarrow \tau\}$ in such a way that we still have σ available to look up identifiers.

Binary search trees



M1={bat->1 camel->2 dog->3}

M2= {bat->1 camel->2 dog->3 **mouse->4**} without destroy M1