Type system implementation

- We extend our Cuppa3 language to Cuppa4 with the addition of a type system with four types:
  - int
  - float
  - string
  - void
- We also assume that int is a subtype of float and float is a subtype of string, that is, a compiler/interpreter is allowed to insert widening conversions and should flag errors for narrowing conversions.

Type system implementation

- We want to be able to write programs such as these:

```
int inc(int x) return x+1;
int y = inc(3);
put "the result is" + y;
```

```
float pow(float b,int p) {
   if (p == 0)
      return 1.0;
   else
      return b*pow(b,p-1);
}

float v;
get v;
int p;
get p;
float result = pow(v,p);
put v + " to the power of " + p +" is "+result;
```

Type system implementation: Syntax

```
program : stmt_list

stmt_list : stmt stmt_list
          | empty

stmt : VOID_TYPE ID '(' opt_formal_args ')' stmt
     | data_type ID '(' opt_formal_args ')' stmt
     | data_type ID opt_init opt_semi
     | ID '=' exp opt_semi
     | GET ID opt_semi
     | PUT exp opt_semi
     | ID '(' opt_actual_args ')' opt_semi
     | RETURN opt_exp opt_semi
     | WHILE '(' exp ')' stmt
     | IF '(' exp ')' stmt opt_else
     | '{' stmt_list '}'

data_type : INTEGER_TYPE
          | FLOAT_TYPE
          | STRING_TYPE
```

```
opt_formal_args : formal_args
                | empty

formal_args : data_type ID ',' formal_args
            | data_type ID

opt_init : '=' exp
         | empty

opt_actual_args : actual_args
                | empty

actual_args : exp ',' actual_args
            | exp

opt_exp : exp
        | empty

opt_else : ELSE stmt
         | empty

opt_semi : ';'
         | empty
```

Type system implementation: Syntax

```
exp : exp PLUS exp
    | exp MINUS exp
    | exp TIMES exp
    | exp DIVIDE exp
    | exp EQ exp
    | exp LE exp
    | INTEGER
    | FLOAT
    | STRING
    | ID
    | ID '(' opt_actual_args ')'
    | '(' exp ')'
    | MINUS exp %prec UMINUS
    | NOT exp
```

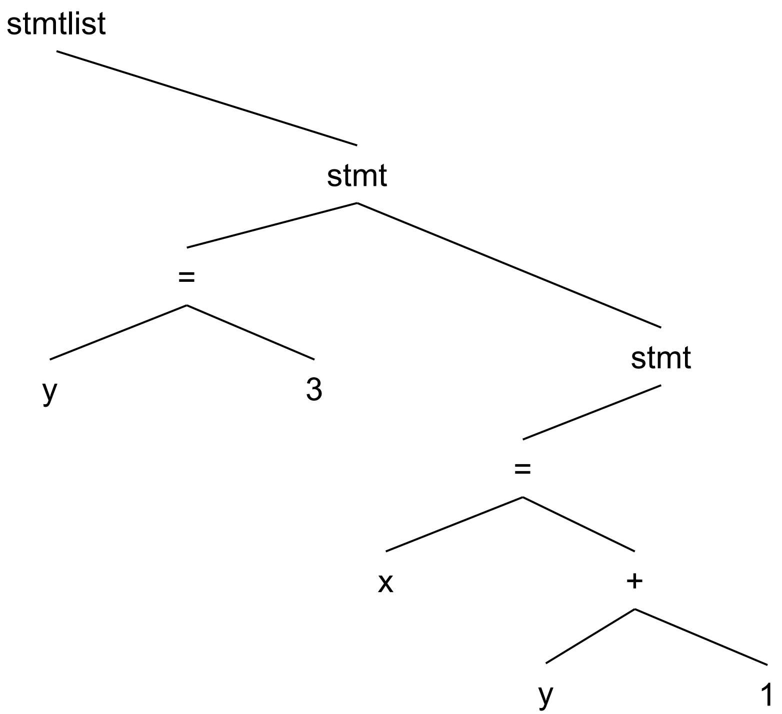
Type system implementation: Semantics

- At the semantic level we *annotate* all ASTs with type information
- We use *type propagation* to check that expressions/statements are properly typed.
  - Type propagation is the systematic tagging of an AST from leafs up with type information.

Type system implementation: Semantics

- Consider the simple example:

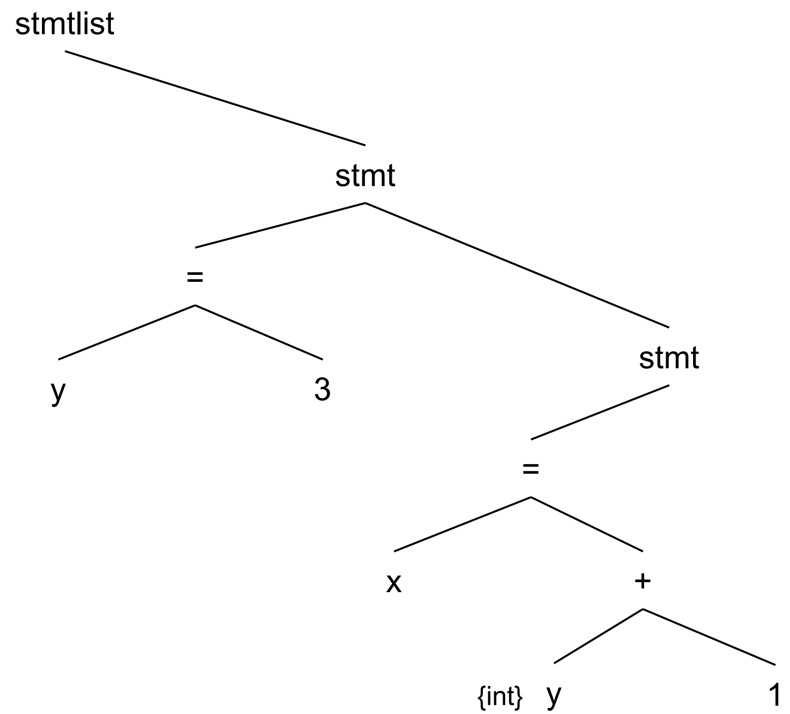
```
int y;
int x;
y = 3;
x = y + 1;
```



Type system implementation: Semantics

- Consider the simple example:

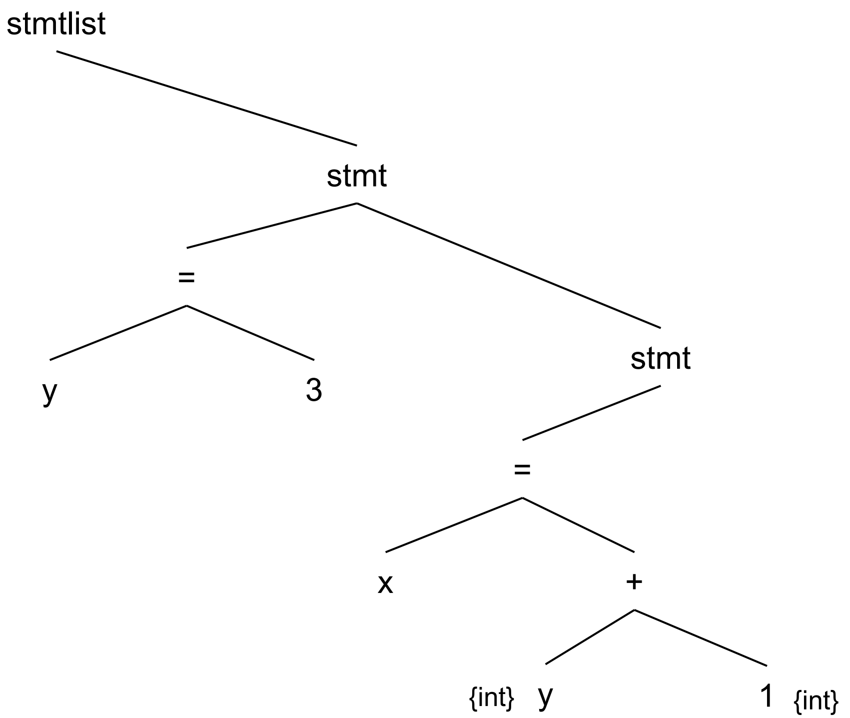
```
int y;
int x;
y = 3;
x = y + 1;
```



Type system implementation: Semantics

- Consider the simple example:

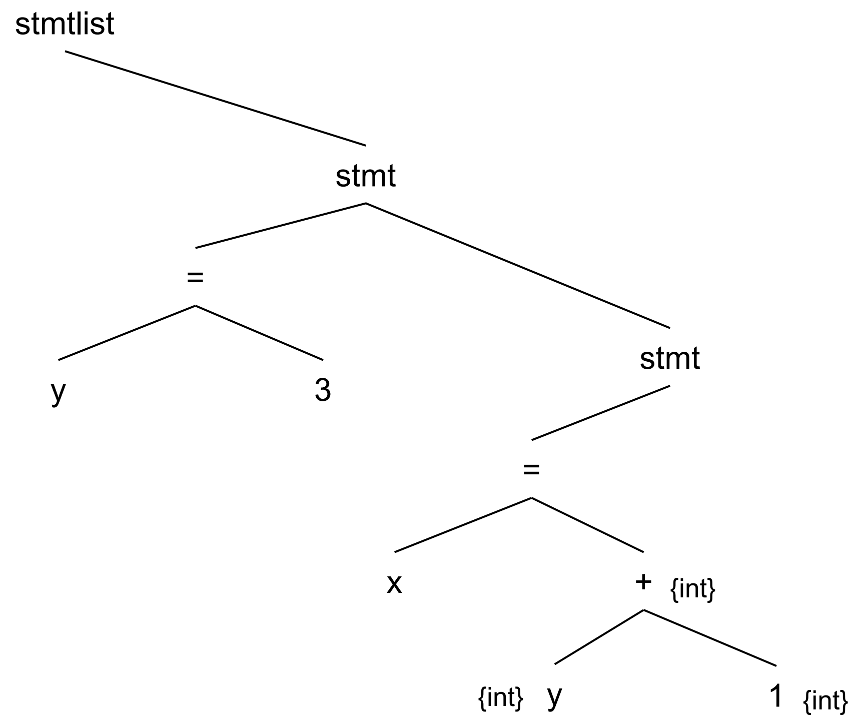
```
int y;
int x;
y = 3;
x = y + 1;
```



Type system implementation: Semantics

- Consider the simple example:

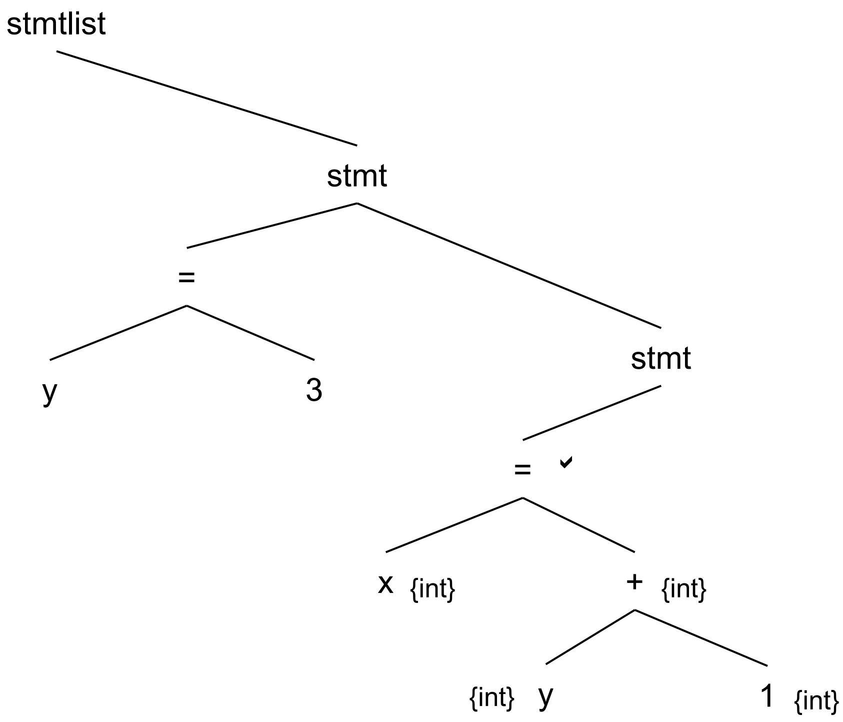
```
int y;
int x;
y = 3;
x = y + 1;
```



Type system implementation: Semantics

- Consider the simple example:

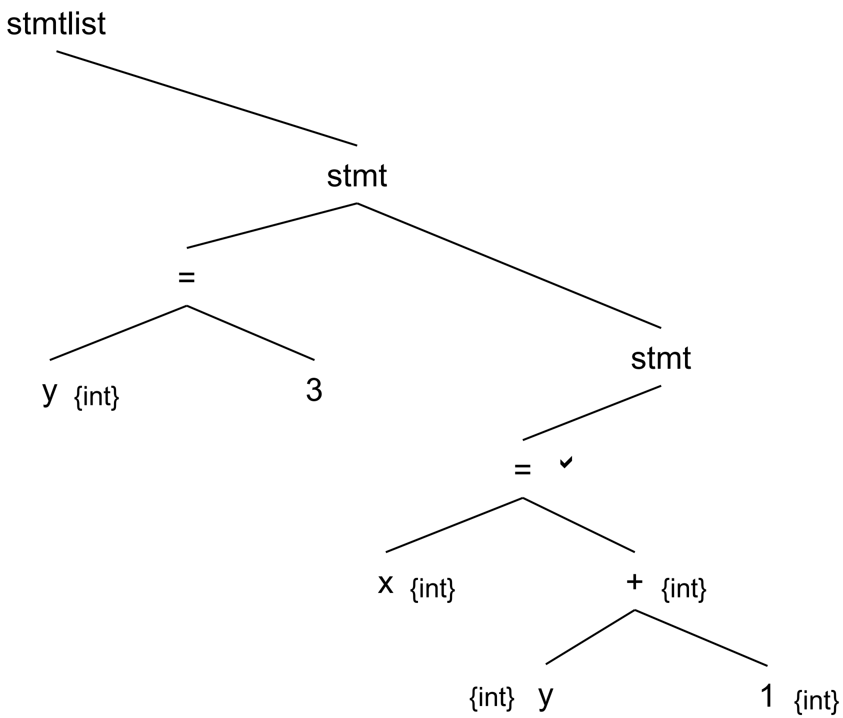
```
int y;
int x;
y = 3;
x = y + 1;
```



Type system implementation: Semantics

- Consider the simple example:

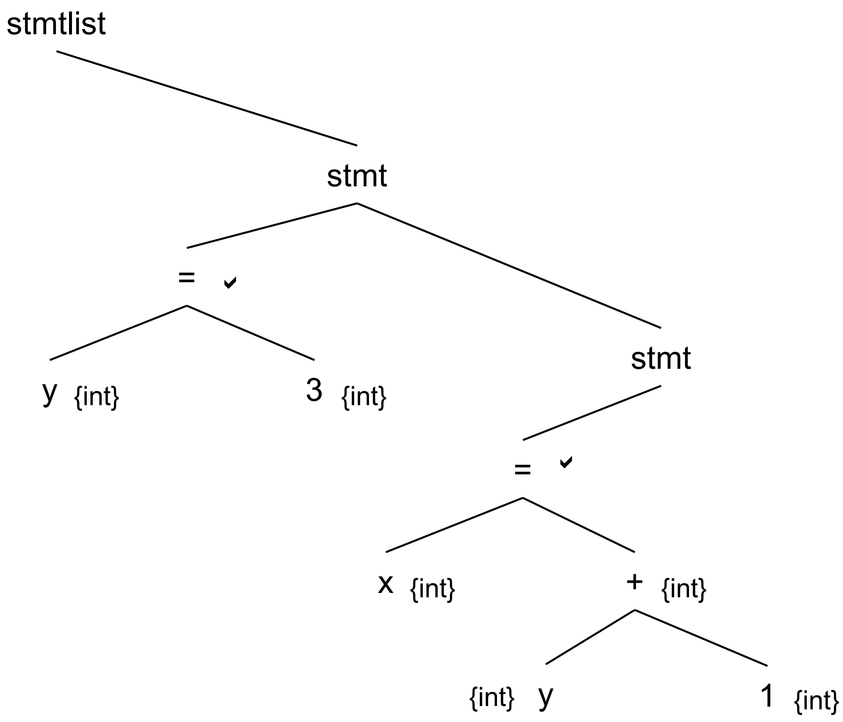
```
int y;
int x;
y = 3;
x = y + 1;
```



Type system implementation: Semantics

- Consider the simple example:

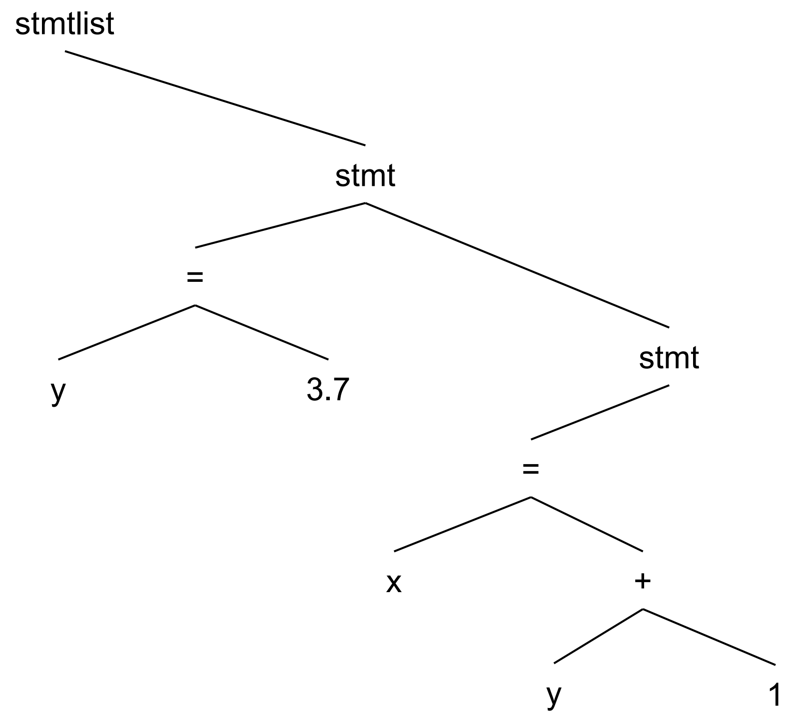
```
int y;
int x;
y = 3;
x = y + 1;
```



Type system implementation: Semantics

- Consider this example which has a typecheck error:

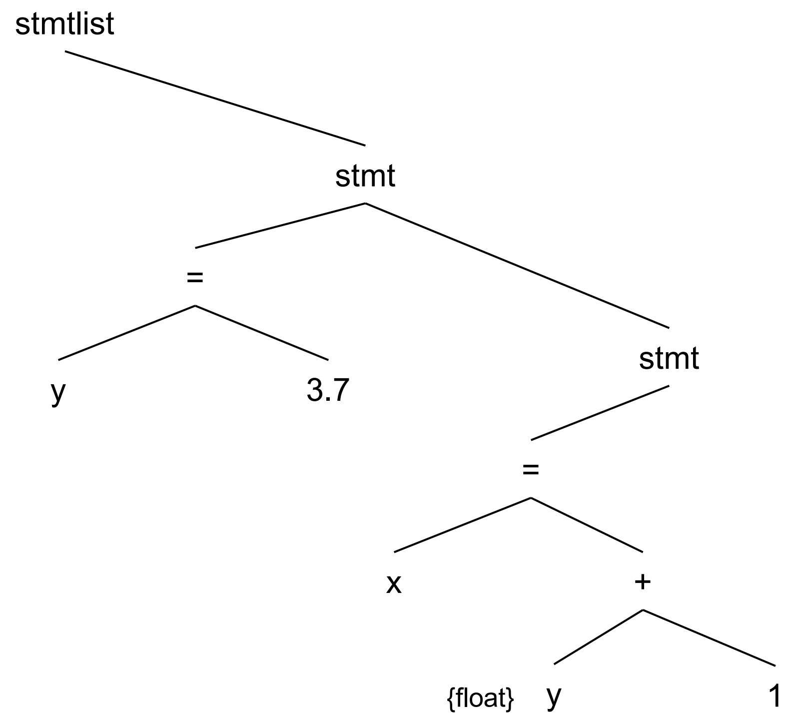
```
float y;
int x;
y = 3.7;
x = y + 1;
```



Type system implementation: Semantics

- Consider this example which has a typecheck error:

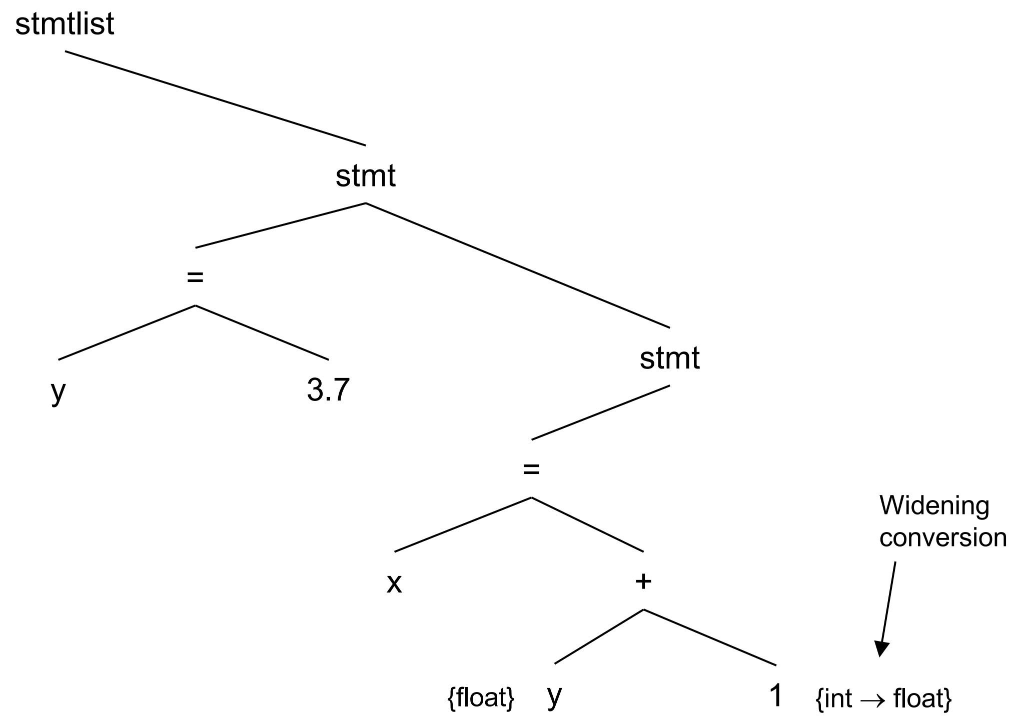
```
float y;
int x;
y = 3.7;
x = y + 1;
```



Type system implementation: Semantics

- Consider this example which has a typecheck error:

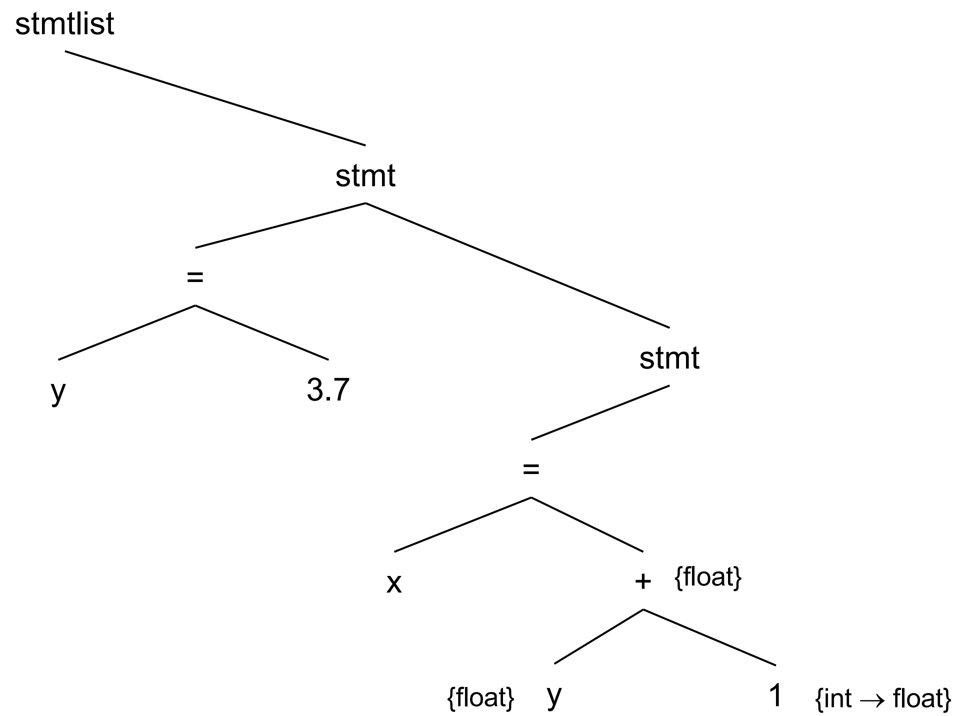
```
float y;
int x;
y = 3.7;
x = y + 1;
```



Type system implementation: Semantics

- Consider this example which has a typecheck error:

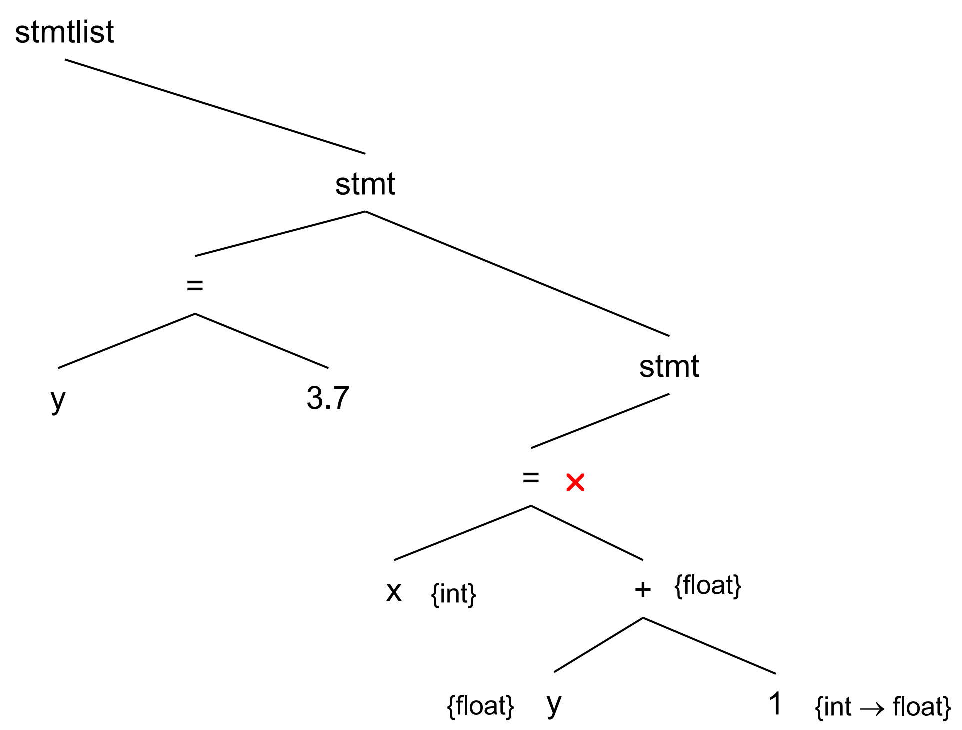
```
float y;
int x;
y = 3.7;
x = y + 1;
```



Type system implementation: Semantics

- Consider this example which has a typecheck error:

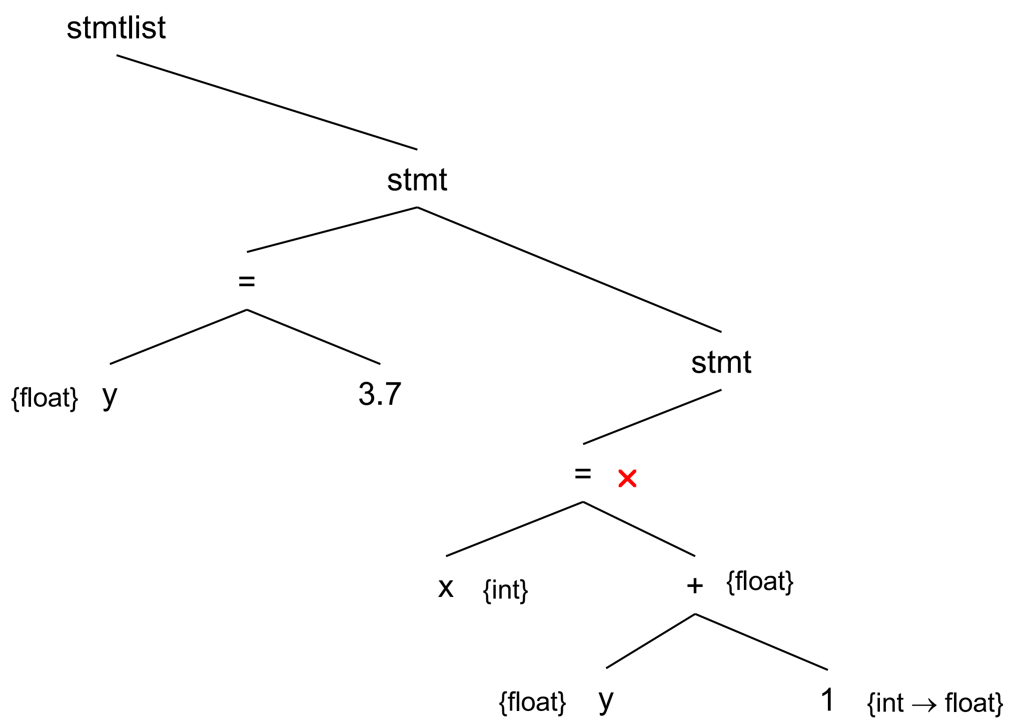
```
float y;
int x;
y = 3.7;
x = y + 1;
```



Type system implementation: Semantics

- Consider this example which has a typecheck error:

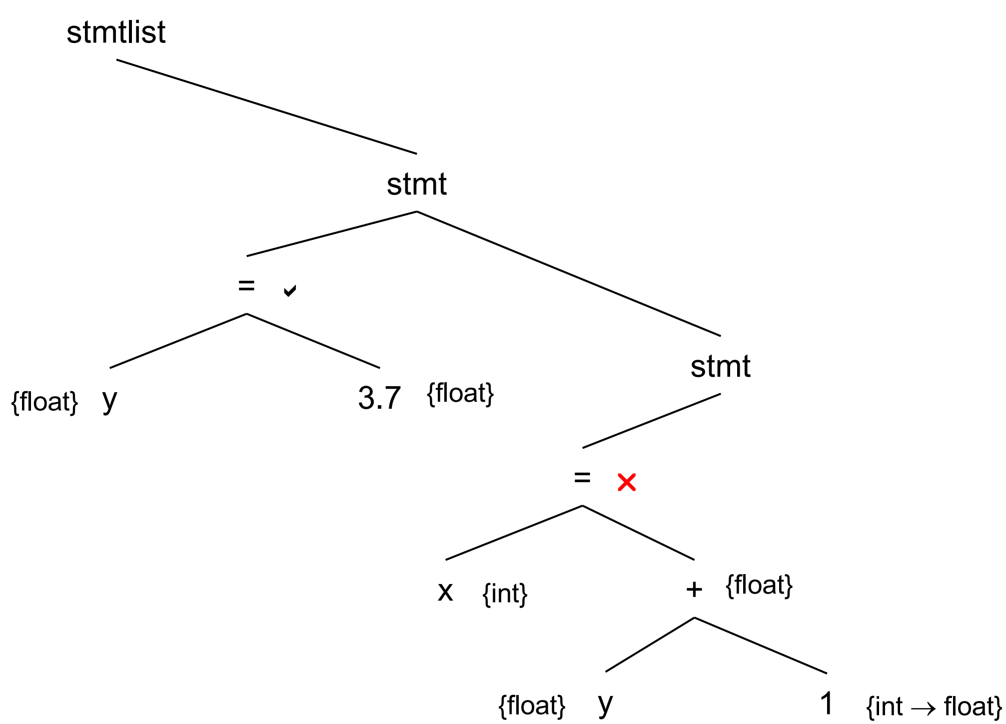
```
float y;
int x;
y = 3.7;
x = y + 1;
```



Type system implementation: Semantics

- Consider this example which has a typecheck error:

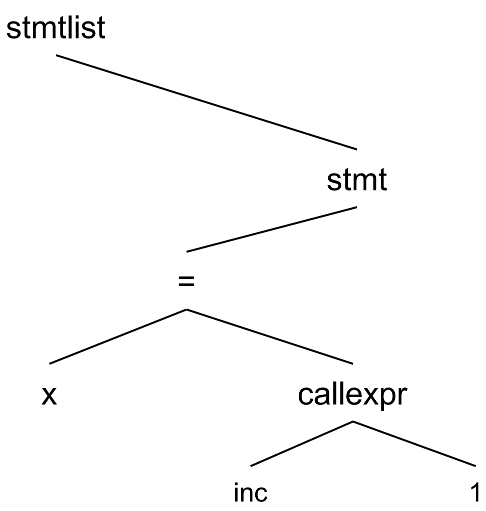
```
float y;
int x;
y = 3.7;
x = y + 1;
```



Type system implementation: Semantics

- Here is an example with a function call:

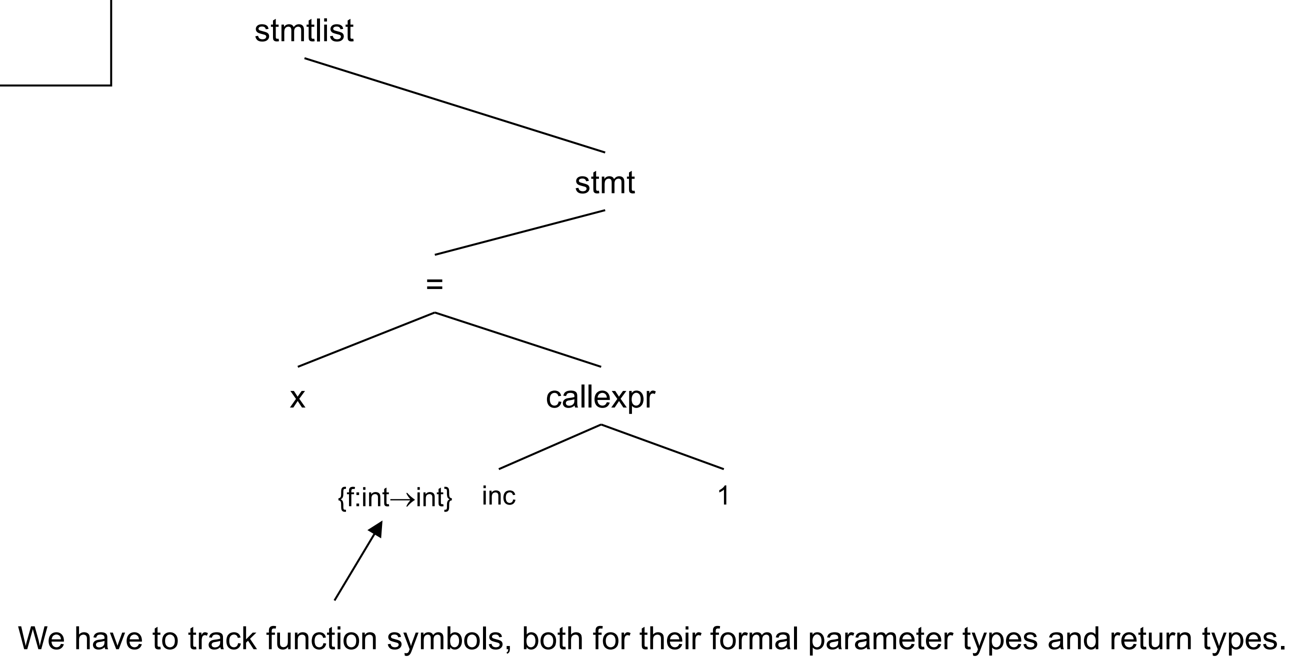
```
int inc(int i) return i+1;
int x;
x = inc(1);
```



Type system implementation: Semantics

- Here is an example with a function call:

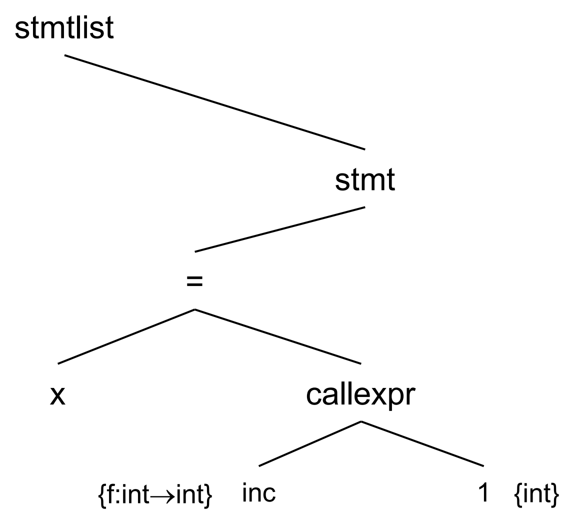
```
int inc(int i) return i+1;
int x;
x = inc(1);
```



Type system implementation: Semantics

- Here is an example with a function call:

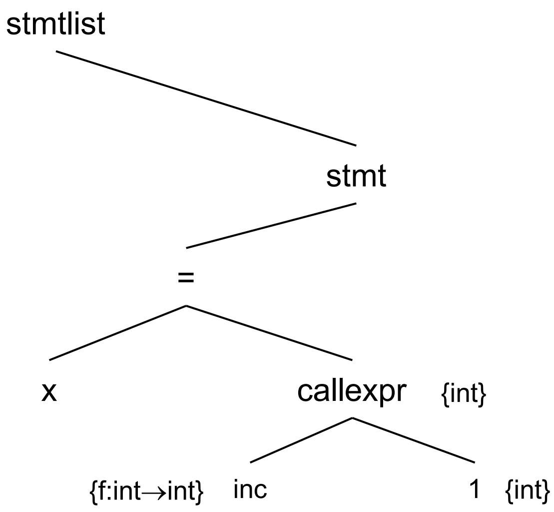
```
int inc(int i) return i+1;
int x;
x = inc(1);
```



Type system implementation: Semantics

- Here is an example with a function call:

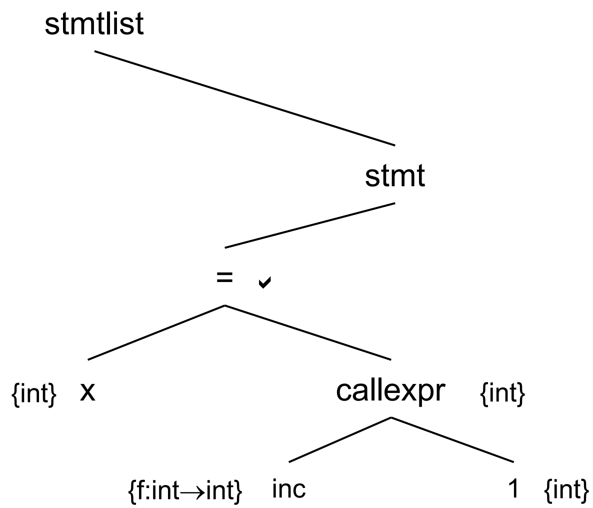
```
int inc(int i) return i+1;
int x;
x = inc(1);
```



Type system implementation: Semantics

- Here is an example with a function call:

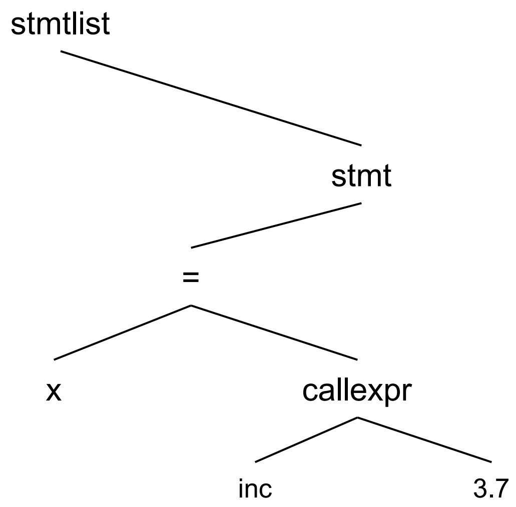
```
int inc(int i) return i+1;
int x;
x = inc(1);
```



Type system implementation: Semantics

- Here is an example with a function call and a type error:

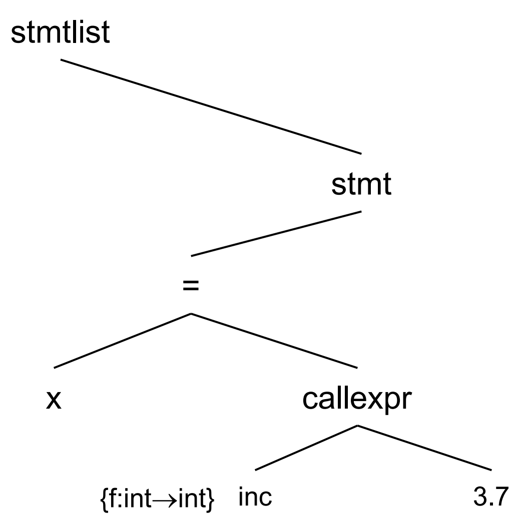
```
int inc(int i) return i+1;
int x;
x = inc(3.7);
```



Type system implementation: Semantics

- Here is an example with a function call and a type error:

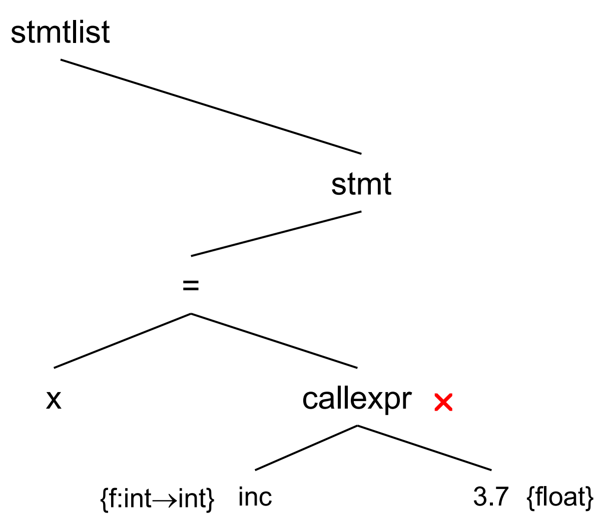
```
int inc(int i) return i+1;
int x;
x = inc(3.7);
```



Type system implementation: Semantics

- Here is an example with a function call and a type error:

```
int inc(int i) return i+1;
int x;
x = inc(3.7);
```



Type system implementation

- Even though Cuppa4 is statically typed we will implement a dynamic type checker
  - Type propagation is done as part of the interpretation!
- Central to our implementation is the type promotion table that implements our type hierarchy.
- We use the type promotion table to implement our type propagation and type checking

Type system implementation: Type Promotion Table

```
# type promotion tables for builtin primitive types.  these tables
# implement the type hierarchy
#
#              integer < float < string
#              void

_promote_table = {
  'string' : {'string': 'string', 'float': 'string', 'integer': 'string',  'void': 'void'},
  'float'  : {'string': 'string', 'float': 'float',  'integer': 'float',   'void': 'void'},
  'integer': {'string': 'string', 'float': 'float',  'integer': 'integer', 'void': 'void'},
  'void'   : {'string': 'void',   'float': 'void',   'integer': 'void',    'void': 'void'},
}

_conversion_table = {
  'string' : {'string': str,  'float': str,   'integer': str,   'void': None},
  'float'  : {'string': str,  'float': float, 'integer': float, 'void': None},
  'integer': {'string': str,  'float': float, 'integer': int,   'void': None},
  'void'   : {'string': None, 'float': None,  'integer': None,  'void': None},
}

_safe_assign_table = {
  'string' : {'string': True,  'float': True,  'integer': True,  'void': False},
  'float'  : {'string': False, 'float': True,  'integer': True,  'void': False},
  'integer': {'string': False, 'float': False, 'integer': True,  'void': False},
  'void'   : {'string': False, 'float': False, 'integer': False, 'void': False},
}

def promote(type1, type2):
    return _promote_table.get(type1).get(type2)

def conversion_fun(ltype, rtype):
    return _conversion_table.get(ltype).get(rtype)

def safe_assign(ltype, rtype):
        return _safe_assign_table.get(ltype).get(rtype)
```

Type system implementation: Symbol Table

```python
# symbol functions
# types of symbol values:
#     ('scalar-val', type, val)
#     ('function-val', return_data_type, arglist, body, context)

def declare_scalar(self, sym, data_type, init_val):
    '''
    declare a symbol in the current scope.
    '''
    # first we need to check whether the symbol was already declared
    # at this scope
    if sym in self.scoped_symtab[CURR_SCOPE]:
        raise ValueError("symbol {} already declared".format(sym))

    # enter the symbol with its value in the current scope
    value = ('scalar-val', data_type, init_val)
    self.scoped_symtab[CURR_SCOPE].update({sym : value})

def declare_fun(self, sym, return_data_type, arglist, body, context):
    '''
    declare a symbol in the current scope.
    '''
    # first we need to check whether the symbol was already declared
    # at this scope
    if sym in self.scoped_symtab[CURR_SCOPE]:
        raise ValueError("symbol {} already declared".format(sym))

    # enter the symbol with its value in the current scope
    value = ('function-val', return_data_type, ('lambda', arglist, body, context))
    self.scoped_symtab[CURR_SCOPE].update({sym : value})
```

Type system implementation: Walk

```
def assign_stmt(node):

    (ASSIGN, name, exp) = node
    assert_match(ASSIGN, 'assign')

    (sym_type, data_type, *_) = state.symbol_table.lookup_sym(name)

    if sym_type != 'scalar-val':
        raise ValueError("Cannot assign a value to the name {}"\
                         .format(name))

    (t, v) = walk(exp)

    if not safe_assign(data_type, t):
        raise ValueError(\
          "a value of type {} cannot be assigned to the variable {} of type {}"\
          .format(t,name,data_type))

    value = (sym_type, data_type, conversion_fun(data_type,t)(v))
    state.symbol_table.update_sym(name, value)
```

```
def plus_exp(node):

    (PLUS,c1,c2) = node
    assert_match(PLUS, '+')

    (t1, v1) = walk(c1)
    (t2, v2) = walk(c2)

    type = promote(t1, t2)

    if type in ['integer', 'float']:
        return (type, v1 + v2)
    elif type == 'string':
        return ('string', str(v1) + str(v2))
    else:
        raise ValueError('unsupported type {} in + operator'.format(type))
```

Type system implementation

In [2]:
```python
from cuppa4_interp import interp
```

In [13]:
```python
str_concat = \
'''
float x = 3.0 + 1;
put "x = " + x;
'''
interp(str_concat)
```

```
x = 4.0
```

In [4]:
```python
add = \
'''
float add (float a, float b)
{
    return a+b;
}

float x = add(3.0,2);
put x;
'''

interp(add)
```

```
5.0
```

In [9]:
```python
factrec = \
'''
// recursive implementation of factorial
int fact(int x)
{
     if (x <= 1)
        return 1;
     else
        return fact(x-1) * x;
}

put fact(3);
'''
interp(factrec)
```

```
6
```