Programming Language Design 2024

Scopes, functions and parameters

Hans Hüttel

19 February 2024

Part I

The topic for today

Learning goals for scope rules

- To understand the definitions of dynamic and static scope rules
- To be able to apply the definitions of scope rules to reason about the behaviour of a program, given specific scope rules
- To be able to reason about the strengths and weaknesses of dynamic and static scope rules.
- To be able to describe how function calls are implemented for dynamic and static scope rules and how memory is allocated (deep/shallow binding and activation records).

Learning goals for parameter passing mechanisms

- To be able to explain the differences between different parameter passing mechanisms.
- To be able to apply the definition of a parameter passing mechanism to reason about the behaviour of a program.
- To be able to reason about the strengths and weaknesses of parameter passing mechanisms.

Part II

Some problems for discussion

For proponents of Python

Here are two fragments of Python code.

```
xs = []
def foo():
    xs.append(42)
foo()
```

and

```
xs = []
def foo():
    xs +=[42]
foo()
```

The first one will succeed. The second one gives a run-time error – which error is that? And why is there a difference here?

Definitions, declarations and scopes

Python uses *declaration by definition*: A variable is created the moment one first assigns a value to it. A function scope contains the names that one *defines* in the function body. These are only visible within the body.

In

```
xs = []
def foo():
    xs.append(42)
foo()
```

we call a function of the global variable `xs` (this function is defined to be there!) but never assign a value to it.

Definitions, declarations and scopes

```
# UnboundLocalError: local variable 'xs'
# referenced before assignment
```

In

```
xs = []
def foo():
    xs +=[42]
foo()
```

we attempt to update the value of a new local `xs` that gets implicitly declared when it is mentioned – but whose value was never defined.


BAFTA WINNER
ADAPTED SCREENPLAY
2018 ACADEMY AWARD WINNER
BEST ADAPTED SCREENPLAY
2018 AFI AWARDS WI
MOVIE OF THE YEAR
CALL ME BY YOUR NAME

What does this procedure do?

Here is an ALGOL60 procedure that uses call-by-name:

```
procedure svop (a, b);
integer a, b, temp;
begin
  temp := a;
  a := b;
  b:= temp
end;
```

What is the intended effect of `svop`? Now suppose `x` is an integer-indexed array. What will happen if we call `svop(i, x[i])`? Any explanation why? Is that what we wanted?

Call-by-name

```
procedure svop (a, b);
integer a, b, temp;
begin
  temp := a;
  a := b;
  b:= temp
end;
```

Here is what happens with `svop(i, x[i])`. Suppose that our array x has that $x[2] \mapsto 5$. We get

Before the call	$i \mapsto 2$	$x[2] \mapsto 5$
After the call	$i \mapsto 5$	$x[5] \mapsto 2$

Call-by-name

The reason for this unexpected behaviour can be found by reading the informal description of procedure calls. The procedure call results in the execution of the procedure with actual parameters syntactically substituted for the formal parameters. This means that we execute the following code.

```
begin
  temp := i;
  i := x[i];
  x[i]:= temp
end;
```

What is wrong here?

```
BigType& SumBig(const BigType& a, const BigType& b)
{
     BigType& result
     result = ... (* a and b are used somewhere in here *)
     return result
}
...

BigType sum = SumBig(actuala,actualb)
```

What is wrong with this piece of C++ code? Explain it using what we known about implementing static scope rules.

Pop goes the stack . . .

```
BigType& SumBig(const BigType& a, const BigType& b)
{
     BigType& result
     result = ... (* a and b are used somewhere in here *)
     return result
}
...
BigType sum = SumBig(actuala,actualb)
```

After the call of `SumBig` finishes, the activation record for the call is popped from the run-time stack, and the reference to the value that has type `BigType` is forever gone.

Off the record

In Pascal, one can "open" a record and allow statements to mention the fields without mentioning the enclosing struct.

```
var x : record
            value : integer
            known : boolean
        end ;
begin
    with x do
       begin
          value := 7
          known := false
    end
end
```

What are the advantages and disadvantages of this? Can we find a better syntax?

Implicit bindings and values that have no syntax

The advantage is that one has to write less.

The disadvantage is that the fields are only implicitly bound in the **with**-block. In someone's opinion (mine) we should make the implicit bindings explicit to improve readability.

We could write e.g.

```
with (value: integer,known: boolean) = x do
...
```

In Pascal but also in many other imperative languages, structs and records are values but there is no syntax for describing such values!

This is somewhat reminiscent of ...

Named parameters (but not call-by-name)

In Java, a method call lists the actual parameters as

```
window.addNew("Title", 20, 50, 100, 50, true);
```

but Python lets one associate the formal and actual parameters directly in a call:

```
window.addNew(title="Title",
                      xPosition=20,
                      yPosition=50,
                      width=100,
                      height=50,
                      drawingNow=True)
```

What do you think of this notion of named parameters?

Pros and cons

- We can write the actual parameters in any order we like!
- We can make sure that the actual parameters are bound to what they were intented to be bound to.
- But what are the names of the parameters in a call? Are they binding or bound occurrences?

Pros and cons

- We can write the actual parameters in any order we like!
- We can make sure that the actual parameters are bound to what they were intented to be bound to.
- But what are the names of the parameters in a call? Are they binding or bound occurrences? In order for them to be bound occurrences, we have to have a somewhat peculiar notion of the scope of formal parameters: It extends beyond the body of the function in which it was declared – to all the lines scattered across the program in which the function gets called.

Part III

About today (summing up)

Evaluation of the course so far

- What are the most important questions about the course that you have right now?
- Is there a particular topic/problem that we should follow up on?
- What can we do to get more students to attend?