INF222 Crashcourse

Procedures - v2023

Sander Wiig

Bergen Language Design Laboratory
Institute for Informatics
University of Bergen

May 15, 2023

UNIVERSITY OF BERGEN
Faculty of Mathematics and Natural Scien

Script and Presentation

Figure: https://github.com/Swi005/Book-of-Magne/tree/v2023

What is a Programming Language?

A Programming language is ...

- used to tell a computer what to do

What is a Programming Language?

A Programming language is ...

- used to tell a computer what to do
- usually artificial

Grouping Languages by domain

Languages are usually grouped into two categories when based on their specificity

- DSLs, small, targeted at specific problems. Internal/embedded vs external

Grouping Languages by domain

Languages are usually grouped into two categories when based on their specificity

- DSLs, small, targeted at specific problems. Internal/embedded vs external
- GPLs, large, many uses.

Grouping Languages by domain

Characteristic	DSL	GPL
Domain	Small and well-defined domain	Generality, many use cases
Size	Small ASTs	Large ASTs, often user extensible
Lifespan	As long as their domain	years to decades
Extensibility	Usually not extendible	Extendable

Figure: Comparison between GPLs and DSLs

Syntax and Semantics

Definition

All languages consist of two parts

- **Syntax** - Defines shape

Syntax is defined by a grammar.
Grammar is not covered in this course :)

Syntax and Semantics

Definition

All languages consist of two parts

- **Syntax** - Defines shape
- **Semantics** - Defines meaning

Syntax is defined by a grammar.
Grammar is not covered in this course :)

Meta Programming

A metaprogram is a program that works on *other* programs.
Compilers and Interpreters are examples of metaprograms

Definition

- **Object Language** - Langage that gets compiled/interpreter
- **Meta Language** - Language used to implement the compiler/interpreter

!OBS Compilers vs Interpreters!

We often end up using the term Compiler to talk about both Compilers and Interpreters.

Sum of Products

```
1  data SomeType = A Bool Bool Bool
2                | B Bool
3                | C
```

Examples

$$\underbrace{(\text{Bool} \times \text{Bool} \times \text{Bool})}_{A} + \underbrace{\text{Bool}}_{B} + \underbrace{1}_{C}$$

`Bool` is either `True` or `False`.
The total number of values of type SomeType is $8 + 2 + 1 = 11$.

Sum of Products

```
1   interface SomeType {}
2   class A implements SomeType {
3       boolean a;
4       boolean b;
5       boolean c ;
6   }
7   class B implements SomeType {
8       boolean a;
9   }
10  class C implements SomeType {
11  }
```

Figure: Sum of Products in Java

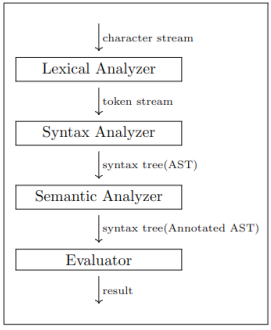
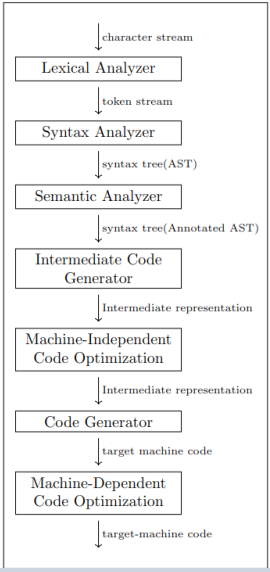


Figure 2.3: Phases of an interpreter



What is a Procedure

- Procedures are "programs within programs"
- Procedures have their own environment
- Functions $\neq$ Procedures

Anatomy of a Procedure

```
1
2 procedure <procedure_name> (<params>)
3     <procedure code>
```

Figure: Procedure structure

Declaration vs. Calling

```
1  program Proc_Example
2  begin
3     procedure swap (upd x: integer, upd y:integer)
4     begin
5        var tmp : integer; //
6        tmp := x;
7        x := y;
8        y := tmp;
9     end
10
11    procedure main ()
12    begin
13       var a = 4;
14       var b = 5;
15       call
```

Parameter Semantics

- **OBS** - "read only"
- **UPD** - "read/write"
- **OUT** - "write only"

Reference Semantics

- Parameters become aliased to arguments
- Points to same memory address
- Unsafe, but sometimes useful

Running a procedure with reference semantics

1. Get stackframe
2. Wipe environment
3. Add parameters to the environment with same address as arg
4. run the procedure code
5. restore the environment

Copy Semantics

- Parameters are declared as variables and initialized with args' value
- Safer
- More intuitive behavior
- More complicated to implement

Running a procedure with copy semantics

1. Get stackframe
2. Get values of args
3. Wipe environment
4. Add parameters to environment
5. init those parameters with the arg values
6. run the procedure code
7. get the values of the parameters
8. restore the environment
9. copy the parameter values back to the args

Swap example v2

```
1  procedure GroupSwap (upd x: integer, upd y :integer)
2  begin
3      y := x + y;
4      x := y - x;
5      y := y - x;
6  end;
7
8  procedure SelfSwap();
9  begin
10     var a = 5;
11     call GroupSwap (a, a);
12 end;
```

Reference semantics

```
1  procedure GroupSwap (upd x: integer, upd y :integer)
2  begin
3      y := x + y;
4      x := y - x;
5      y := y - x;
6  end;
7
8  procedure SelfSwap();
9  begin
10     var a = 5;
11     call GroupSwap (a, a); //x =5, y = 5
12 end;
```

Reference semantics

```
1  procedure GroupSwap (upd x: integer, upd y :integer)
2  begin
3      y := x + y;// y = x + y = 5+5 => y = 10
4      x := y - x;
5      y := y - x;
6  end;
7
8  procedure SelfSwap();
9  begin
10     var a = 5;
11     call GroupSwap (a, a); //x =5, y = 5
12 end;
```

Reference semantics

```
1  procedure GroupSwap (upd x: integer, upd y :integer)
2  begin
3      y := x + y;// y = x + y = 5+5 => y = 10
4      x := y - x;// x = y - x = 10 - 5 => x = 5
5      y := y - x;
6  end;
7
8  procedure SelfSwap();
9  begin
10     var a = 5;
11     call GroupSwap (a, a); //x =5, y = 5
12 end;
```

Reference semantics

```
1  procedure GroupSwap (upd x: integer, upd y :integer)
2  begin
3      y := x + y;// y = x + y = 5+5 => y = 10
4      x := y - x;// x = y - x = 10 - 5 => x = 5
5      y := y - x;// y = y - x = 10 - 5 => y = 5
6  end;
7
8  procedure SelfSwap();
9  begin
10     var a = 5;
11     call GroupSwap (a, a); //x =5, y = 5
12 end;
```

Copy semantics

```
1  procedure GroupSwap (upd x: integer, upd y :integer)
2  begin
3      y := x + y;
4      x := y - x;
5      y := y - x;
6  end;
7
8  procedure SelfSwap();
9  begin
10     var a = 5;
11     call GroupSwap (a, a); //x =a, y = a
12 end;
```

Copy semantics

```
1  procedure GroupSwap (upd x: integer, upd y :integer)
2  begin
3      y := x + y;// a = a + a = 5+5 => a = 10
4      x := y - x;
5      y := y - x;
6  end;
7
8  procedure SelfSwap();
9  begin
10     var a = 5;
11     call GroupSwap (a, a); //x =a, y = a
12 end;
```

Copy semantics

```
1  procedure GroupSwap (upd x: integer, upd y :integer)
2  begin
3      y := x + y;// a = a + a = 5+5 => a = 10
4      x := y - x;// a = a - a = 10 - 10 => a = 0
5      y := y - x;
6  end;
7
8  procedure SelfSwap();
9  begin
10     var a = 5;
11     call GroupSwap (a, a); //x =a, y = a
12 end;
```

Copy semantics

```
1  procedure GroupSwap (upd x: integer, upd y :integer)
2  begin
3      y := x + y;// a = a + a = 5+5 => a = 10
4      x := y - x;// a = a - a = 10 - 10 => a = 0
5      y := y - x;// a = a - a = 0 - 0 => a = 0
6  end;
7
8  procedure SelfSwap();
9  begin
10     var a = 5;
11     call GroupSwap (a, a); //x =a, y = a
12 end;
```