

CS 4400

Programming Languages

[Overview & Intro to Haskell]

Ferdinand Vesely

September 11, 2020

Programming Languages

What is this class about

- A study of programming languages
- Through examples
- Language features from an implementation-based perspective

So, Programming Languages...

What is a programming language?

```
public static int Main(string[] args)
{
    var currentPath = new FileInfo(System.Reflection.Assembly.GetEntryAssembly().Location);
    var isWindows = RuntimeInformation.IsOSPlatform(OSPlatform.Windows);

    string platformFolder = isWindows ? WinFolderName : UnixFolderName;

    string argsString = args.Length > 0 ? string.Join(" ", args) : null;
    var pwshPath = Path.Combine(currentPath, platformFolder, PwshDllName);
    string processArgs = string.IsNullOrEmpty(argsString) ? $"\"{pwshPath}\"" : $"\"{pwshPath}\" {argsString}";

    if (File.Exists(pwshPath))
    {
        Console.CancelKeyPress += (sender, e) =>
        {
            e.Cancel = true;
        };
    }
}
```

What is a programming language?

```
(define generate-vector
  (lambda (size proc)
    (let ((ans (make-vector size)))
      (letrec ((loop
                 (lambda (i)
                   (cond ((= i size) ans)
                         (else
                          (vector-set! ans i (proc i))
                          (loop (+ i 1)))))))
        (loop 0)))))
```

What is a programming language?

```
impl<D: Disk> FileSystem<D> {
    pub fn open(mut disk: D, block_opt: Option<u64>) -> Result<Self> {
        for block in block_opt.map_or(0..65536, |x| x..x + 1) {
            let mut header = (0, Header::default());
            disk.read_at(block + header.0, &mut header.1)?;

            if header.1.valid() {
                let mut root = (header.1.root, Node::default());
                disk.read_at(block + root.0, &mut root.1)?;

                let mut free = (header.1.free, Node::default());
                disk.read_at(block + free.0, &mut free.1)?;

                return Ok(FileSystem {
                    disk: disk,
                    block: block,
                    header: header,
                });
            }
        }
    }
}
```

What is a programming language?

```
fvariable ci fvariable c fvariable zi fvariable z
: >2? z f@ fdup f* zi f@ fdup f* f+ 4.0e f> ;
: nextr z f@ fdup f* zi f@ fdup f* f- c f@ f+ ;
: nexti z f@ zi f@ f* 2.0e f* ci f@ f+ ;
: pixel c f! ci f! 0e z f! 0e zi f! 150 50 do nextr nexti zi
: left->right -1.5e 80 0 do fover fover pixel emit 0.026e f+
: top->bottom cr -1e 40 0 do left->right cr 0.05e f+ loop fd
top->bottom bye
```


What is a programming language?

do_trace_entry:

```
    movel    #-ENOSYS,%sp@(PT_OFF_D0) | needed for strace
    subql    #4,%sp
    SAVE_SWITCH_STACK
    jbsr     syscall_trace
    RESTORE_SWITCH_STACK
    addql    #4,%sp
    movel    %sp@(PT_OFF_ORIG_D0),%d0
    cmpl     #NR_syscalls,%d0
    jcs     syscall
```

badsys:

```
    movel    #-ENOSYS,%sp@(PT_OFF_D0)
    jra     ret_from_syscall
```

What is a programming language?

```
+++++++  
[  
  >++++  
  [  
    >++ >+++ >+++ >+ <<<<-  
  ]  
  >+ >+ >- >>+ [<] <-  
]  
>>. >---. +++++ ++..+++ . >>.  
<-. <. +++ .----- .----- .  
>>+. >+.
```

What is a programming language?

What is a programming language?



What is a programming language?

- syntax

What is a programming language?

- syntax
- **semantics**

What is a programming language?

- syntax
- **semantics**
- pragmatics – idioms

What is a programming language?

- syntax
- **semantics**
- pragmatics – idioms
- ecosystem – libraries, tools

Why study programming languages?

- Different styles / paradigms:

Why study programming languages?

- Different styles / paradigms:
- Imperative?

Why study programming languages?

- Different styles / paradigms:
- Imperative?
- Functional?

Why study programming languages?

- Different styles / paradigms:
- Imperative?
- Functional?
- Logic?

Why study programming languages?

- Different styles / paradigms:
- Imperative?
- Functional?
- Logic?
- ???

Why study programming languages?

- Different styles / paradigms:
- Imperative?
- Functional?
- Logic?
- ???
- How do they relate?

Why study programming languages?

- Different styles / paradigms:
- Imperative?
- Functional?
- Logic?
- ???
- How do they relate?
- Understanding what programs mean

Why? Watman!



Semantics

- defines (precise?) meaning of constructs in a programming language
- various styles – “main” ones are:
 - operational – big-step, small-step, reduction semantics, rewriting semantics
 - denotational – translating a PL into pure math
 - axiomatic – by means of properties satisfied by language constructs
- combinations and variations of the above

- informal – language manuals

15.26.1. Simple Assignment Operator =

If the type of the right-hand operand is not assignment compatible with the type of the variable (§5.2), then a compile-time error occurs.

Otherwise, at run time, the expression is evaluated in one of three ways.

If the left-hand operand expression is a field access expression $e.f$ (§15.11), possibly enclosed in one or more pairs of parentheses, then:

- First, the expression e is evaluated. If evaluation of e completes abruptly, the assignment expression completes abruptly for the same reason.
- Next, the right hand operand is evaluated. If evaluation of the right hand expression completes abruptly, the assignment expression completes abruptly for the same reason.
- Then, if the field denoted by $e.f$ is not `static` and the result of the evaluation of e above is `null`, then a `NullPointerException` is thrown.
- Otherwise, the variable denoted by $e.f$ is assigned the value of the right hand operand as computed above.

If the left-hand operand is an array access expression (§15.10.3), possibly enclosed in one or more pairs of parentheses, then:

- First, the array reference subexpression of the left-hand operand array access expression is evaluated. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason; the index subexpression (of the left-hand operand array access expression) and the right-hand operand are not evaluated and no assignment occurs.

from: <https://docs.oracle.com/javase/specs/jls/se12/html/jls-15.html#jls-15.26.1>

Semantics

- formal?

Expressions

$$E \vdash exp \Rightarrow v/p$$

$$\frac{E \vdash atexp \Rightarrow v}{E \vdash atexp \Rightarrow v} \quad (96)$$

$$\frac{E \vdash exp \Rightarrow vid \quad vid \neq \mathbf{ref} \quad E \vdash atexp \Rightarrow v}{E \vdash exp atexp \Rightarrow (vid, v)} \quad (97)$$

$$\frac{E \vdash exp \Rightarrow en \quad E \vdash atexp \Rightarrow v}{E \vdash exp atexp \Rightarrow (en, v)} \quad (98)$$

$$\frac{s, E \vdash exp \Rightarrow \mathbf{ref}, s' \quad s', E \vdash atexp \Rightarrow v, s'' \quad a \notin \text{Dom}(\text{mem of } s'')}{s, E \vdash exp atexp \Rightarrow a, s'' + \{a \mapsto v\}} \quad (99)$$

Semantics

- formal?

$$\begin{aligned} \mathcal{C}[\langle \text{EmptyStmt} \rangle] \gamma \theta \sigma &::= \\ \mathcal{C}[\langle ; \rangle] \gamma \theta \sigma &= \theta(\gamma, \sigma) \end{aligned}$$

$$\begin{aligned} \mathcal{C}[\langle \text{LabeledStmt} \rangle] \gamma \theta \sigma &::= \\ \mathcal{C}[\langle \text{Id} \rangle : \langle \text{Stmt} \rangle] \gamma \theta \sigma &= \mathcal{C}[\langle \text{Stmt} \rangle] \gamma_1 \theta_1 \sigma \text{ where} \\ \gamma_1 &= \gamma[\text{Id} \leftarrow \theta_2] \text{ where} \\ \forall \gamma_2, \sigma_2. \theta_2(\gamma_2, \sigma_2) &= \mathcal{C}[\langle \text{Stmt} \rangle] \gamma_2 \theta_1 \sigma_2 \\ \forall \gamma_1, \sigma_1. \theta_1(\gamma_1, \sigma_1) &= \theta(\gamma, \sigma_1) \end{aligned}$$

$$\begin{aligned} \mathcal{C}[\langle \text{ExprStmt} \rangle ;] \gamma \theta \sigma &::= \\ \mathcal{C}[\langle \text{Expr} \rangle] \gamma \theta \sigma &= \mathcal{E}[\langle \text{Expr} \rangle] \gamma \kappa \sigma \text{ where} \\ \forall r, \tau, \sigma_1. \kappa(r, \tau, \sigma_1) &= \theta(\gamma, \sigma_1) \end{aligned}$$

Semantics

- formal! + executable

```
Inductive ceval : com -> state -> state -> Prop :=
| E_Skip : forall st,
  SKIP / st \\ st
| E_Ass : forall st a1 n x,
  aeval st a1 = n ->
  (x ::= a1) / st \\ (t_update st x n)
| E_Seq : forall c1 c2 st st' st'',
  c1 / st \\ st' ->
  c2 / st' \\ st'' ->
  (c1 ;; c2) / st \\ st''
| E_IfTrue : forall st st' b c1 c2,
  beval st b = true ->
  c1 / st \\ st' ->
  (IFB b THEN c1 ELSE c2 FI) / st \\ st'
| E_IfFalse : forall st st' b c1 c2,
  beval st b = false ->
  c2 / st \\ st' ->
  (IFB b THEN c1 ELSE c2 FI) / st \\ st'
| E_WhileEnd : forall b st c,
  beval st b = false ->
  (WHILE b DO c END) / st \\ st
| E_WhileLoop : forall st st' st'' b c,
  beval st b = true ->
  c / st \\ st' ->
  (WHILE b DO c END) / st' \\ st'' ->
  (WHILE b DO c END) / st \\ st''
```

where "c1 '/' st '\\ st'" := (ceval c1 st st').

Our Approach

- Semantics = interpreters
- Implemented in Haskell
- In effect, we relate the meaning of our example languages to that of Haskell
- Our semantics – executable

Why bother with semantics?

- precise meaning of a program
- Is my program correct?
 - what does “correct” even mean?
- Is my program equivalent to another one?
- Does this compiler correctly implement the language?

Why bother with semantics?

- Program verification – static, runtime
- Generation of test cases
- Tool generation
- Language design ... ?

Course Particulars

- Syllabus / course webpage:
<https://vesely.io/teaching/CS440of19/syllabus.html>
- Meeting: twice a week: Tue/Fri 9:50-11:30am, Cargill 097 and Zoom
- Zoom links (+ possibly other info) on Canvas
- No required reading, but some resources will be useful

Grades

- 60% assignments
- 35% other (quizzes or exams)
- 5% karma / participation

Contact

- Piazza: <https://piazza.com/northeastern/fall2020/cs4400>
- Hours: TBD, via Zoom or Teams
- Email: f.vesely@northeastern.edu
- Homepage: <https://vesely.io>

On to Haskell!

