CS 4400 / 5400 Programming Languages

[Introduction, Overview, Intro to Haskell]

Ferdinand Vesely

September 10, 2019

A bit about me...

Hi, I'm Ferdinand! I'm new here (started last week).
Born in what is today Slovakia, then part of Czechoslovakia.
Got my first computer at about 6 or 7 years old.

…a bit more about me…

After school, started studying philosophy, but my interests shifted back to computers

Worked as a software dev for a few years before deciding to study CS.

Moved to Swansea, Wales in UK to do a Bachelor's degree…

…and yet more about me

…stayed on for a PhD, which I did on programming language semantics and implementation.

- Thesis on component-based semantics, implementation, and program equivalence (bisimulation)

Postdoc at Tufts, then teaching faculty in Swansea, now Northeastern.

- Worked on automatically transforming semantic specifications

Random: I play guitar and speak 4 (to 5-ish) languages.

Programming Languages

What is this class about

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

Why study programming languages?

- Programming languages come in a wide variety
- Different styles / paradigms:
  - Imperative?
  - Functional?
  - Logic?
- What is a programming language?
  - syntax
  - **semantics**
  - pragmatics – idioms
  - ecosystem – libraries, tools

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

Semantics

- 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 $\boxed{E \vdash exp \Rightarrow v/p}$

$$\frac{E \vdash atexp \Rightarrow v}{E \vdash atexp \Rightarrow v} \tag{96}$$

$$\frac{E \vdash exp \Rightarrow vid \qquad vid \neq \texttt{ref} \qquad E \vdash atexp \Rightarrow v}{E \vdash exp\ atexp \Rightarrow (vid, v)} \tag{97}$$

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

$$\frac{s, E \vdash exp \Rightarrow \texttt{ref}\ , s' \qquad s', E \vdash atexp \Rightarrow v, s'' \qquad a \notin \mathrm{Dom}(mem\ \mathrm{of}\ s'')}{s, E \vdash exp\ atexp \Rightarrow a,\ s'' + \{a \mapsto v\}} \tag{99}$$

from: The Definition of Standard ML by Robin Milner, et al

Semantics

- formal?

$$\mathcal{C}[\![\text{<EmptyStmt>}]\!]\gamma\theta\sigma ::=$$
$$\quad \mathcal{C}[\![\text{;}]\!]\gamma\theta\sigma = \theta(\gamma, \sigma)$$

$$\mathcal{C}[\![\text{<LabeledStmt>}]\!]\gamma\theta\sigma ::=$$
$$\quad \mathcal{C}[\![\text{<Id> : <Stmt>}]\!]\gamma\theta\sigma = \mathcal{C}[\![\text{<Stmt>}]\!]\gamma_1\theta_1\sigma \text{ where}$$
$$\quad\quad \gamma_1 = \gamma[Id \leftarrow \theta_2] \text{ where}$$
$$\quad\quad\quad \forall\gamma_2, \sigma_2.\theta_2(\gamma_2, \sigma_2) = \mathcal{C}[\![\text{<Stmt>}]\!]\gamma_2\theta_1\sigma_2$$
$$\quad\quad \forall\gamma_1, \sigma_1.\theta_1(\gamma_1, \sigma_1) = \theta(\gamma, \sigma_1)$$

$$\mathcal{C}[\![\text{<ExprStmt> ;}]\!]\gamma\theta\sigma ::=$$
$$\quad \mathcal{C}[\![\text{<Expr>}]\!]\gamma\theta\sigma = \mathcal{E}[\![\text{<Expr>}]\!]\gamma\kappa\sigma \text{ where}$$
$$\quad\quad \forall r, \tau, \sigma_1.\kappa(r, \tau, \sigma_1) = \theta(\gamma, \sigma_1)$$

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/CS4400f19/syllabus.html
- Meeting once a week: Thursdays 6-9:15pm, Hurtig Hall 129
- Delivery mainly via lectures, possibly mixed with class / lab-like activities
- No required reading, but some resources will be useful

Grades

- Assignments: 60%
- Exams: midterm 15%, final 20%
- Participation: 5%

Contact

- Piazza: http://piazza.com/northeastern/fall2019/cs44005400
- Nightingale 132A
- Hours: Wednesdays, 2-6pm or by appointment – caveat:
  - Depending on how busy Nightingale will get, I might try to find a different location to hold office hours
  - I might also schedule additional office hours
  - I will update you
- Email: f.vesely@northeastern.edu
- Homepage: https://vesely.io
- Details about TAs to follow

Intro to Haskell

Haskell

- Functional programming language
- Statically typed
- Lazy
- Advanced Type System
  - problem: error messages

PL Basics

Abstract Syntax

Basic Interpreters