30

In linguistics, semantics is the study of the connection between words and their meanings: the word “dog” is an arrangement of shapes on a page, or a sequence of vibrations in the air caused by someone’s vocal cords, which are very different things from an actual dog or the idea of dogs in general.

30

The Meaning of “Meaning”

33

Small-Step Semantics

45

Statements

a statement, on the other hand, is evaluated to make some change to the state of the abstract

35

Expressions

The purpose of an expression is to be evaluated to produce another expression

35

Number, Add, and Multiply

38

reducible?

reduce

40

maintaining a piece of state

41

Boolean true and false; Boolean and, or, and not

42

variables

45

DoNothing

```
class DoNothing ❶
  def to_s
    'do-nothing'
  end

  def inspect
    "«#{self}»"
  end

  def ==(other_statement) ❷
    other_statement.instance_of?(DoNothing)
  end

  def reducible?
    false
  end
end
```

45

==

We want to be able to compare any two statements to see if they’re equal. The other syntax classes inherit an implementation of #==== from Struct, but DoNothing has to define its own.

46

Assign

- If the assignment’s expression can be reduced, then reduce it, resulting in a reduced assignment statement and an unchanged environment.

- If the assignment’s expression can’t be reduced, then update the environment to associate that expression with the assignment’s variable, resulting in a «do-noth ing» statement and a new environment.

47

Assign

```
class Assign < Struct.new(:name, :expression)
  def to_s
    "#{name} = #{expression}"
  end

  def inspect
    "«#{self}»"
  end

  def reducible?
    true
  end

  def reduce(environment)
    if expression.reducible?
      [Assign.new(name, expression.reduce(environment)), environment]
    else
      [DoNothing.new, environment.merge({ name => expression })]
    end
  end
end
```

48

New Machine

```
class Machine < Struct.new(:statement, :environment)
  def step
    self.statement, self.environment = statement.reduce(environment)
  end

  def run
    while statement.reducible?
      puts "#{statement}, #{environment}"
      step
    end

    puts "#{statement}, #{environment}"
  end
end
```

48

If

if (x) { y = 1 } else { y = 2 }

- If the condition can be reduced, then reduce it, resulting in a reduced conditional statement and an unchanged environment.
- If the condition is the expression «true», reduce to the consequence statement and an unchanged environment.

- If the condition is the expression «false», reduce to the alternative statement and an unchanged environment.

49

If

```
class If < Struct.new(:condition, :consequence, :alternative)
  def to_s
    "if (#{condition}) { #{consequence} } else { #{alternative} }"
  end

  def inspect
    "«#{self}»"
  end

  def reducible?
    true
  end

  def reduce(environment)
    if condition.reducible?
      [If.new(condition.reduce(environment), consequence, alternative), environment]
    else
      case condition
      when Boolean.new(true)
        [consequence, environment]
      when Boolean.new(false)
        [alternative, environment]
      end
    end
  end
end
```

49

Output

```
>> Machine.new(
     If.new(
       Variable.new(:x),
       Assign.new(:y, Number.new(1)),
       Assign.new(:y, Number.new(2))
     ),
     { x: Boolean.new(true) }
   ).run
if (x) { y = 1 } else { y = 2 }, {:x=>#<true>}
if (true) { y = 1 } else { y = 2 }, {:x=>#<true>}
y = 1, {:x=>#<true>}
do-nothing, {:x=>#<true>, :y=>#<1>}
=> nil
```

50

sequence

- If the first statement is a «do-nothing» statement, reduce to the second statement and the original environment.
- If the first statement is not «do-nothing», then reduce it, resulting in a new sequence (the reduced first statement followed by the second statement) and a reduced environment.

50

Sequence

```
class Sequence < Struct.new(:first, :second)
  def to_s
    "#{first}; #{second}"
  end

  def inspect
    "«#{self}»"
  end

  def reducible?
    true
  end

  def reduce(environment)
    case first

    when DoNothing.new
      [second, environment]
    else
      reduced_first, reduced_environment = first.reduce(environment)
      [Sequence.new(reduced_first, second), reduced_environment]
    end
  end
end
```

51

Output

```
>> Machine.new(
     Sequence.new(
       Assign.new(:x, Add.new(Number.new(1), Number.new(1))),
       Assign.new(:y, Add.new(Variable.new(:x), Number.new(3)))
     ),
     {}
   ).run
x = 1 + 1; y = x + 3, {}
x = 2; y = x + 3, {}
do-nothing; y = x + 3, {:x=>#<2>}
y = x + 3, {:x=>#<2>}
y = 2 + 3, {:x=>#<2>}
y = 5, {:x=>#<2>}
do-nothing, {:x=>#<2>, :y=>#<5>}
=> nil
```

51

while

- Reduce «while (condition) { body }» to «if (condition) { body; while (condi tion) { body } } else { do-nothing }» and an unchanged environment.

52

While

```
class While < Struct.new(:condition, :body)
  def to_s
    "while (#{condition}) { #{body} }"
  end

  def inspect
    "«#{self}»"
  end

  def reducible?
    true
  end

  def reduce(environment)
    [If.new(condition, Sequence.new(body, self), DoNothing.new), environment]
  end
end
```

52

Output

```
>> Machine.new(
     While.new(
       LessThan.new(Variable.new(:x), Number.new(5)),
       Assign.new(:x, Multiply.new(Variable.new(:x), Number.new(3)))
     ),
     { x: Number.new(1) }
   ).run
while (x < 5) { x = x * 3 }, {:x=>#<1>}
if (x < 5) { x = x * 3; while (x < 5) { x = x * 3 } } else { do-nothing }, {:x=>#<3>}
if (x < 5) { x = x * 3; while (x < 5) { x = x * 3 } } else { do-nothing }, {:x=>#<3>}
if (true) { x = x * 3; while (x < 5) { x = x * 3 } } else { do-nothing }, {:x=>#<3>}
x = x * 3; while (x < 5) { x = x * 3 }, {:x=>#<3>}
x = 3 * 3; while (x < 5) { x = x * 3 }, {:x=>#<3>}
x = 9; while (x < 5) { x = x * 3 }, {:x=>#<9>}
do-nothing; while (x < 5) { x = x * 3 }, {:x=>#<9>}
while (x < 5) { x = x * 3 }, {:x=>#<9>}
if (x < 5) { x = x * 3; while (x < 5) { x = x * 3 } } else { do-nothing }, {:x=>#<9>}
if (9 < 5) { x = x * 3; while (x < 5) { x = x * 3 } } else { do-nothing }, {:x=>#<9>}
if (false) { x = x * 3; while (x < 5) { x = x * 3 } } else { do-nothing }, {:x=>#<9>}
do-nothing, {:x=>#<9>}
=> nil
```