CSC 347 - Concepts of Programming Languages

Formal Semantics

Instructor: James Riely

Learning Objectives

❓ How to unambiguously define the semantics of a programming language?

- Identify the difference between syntactic and semantic expressions in operational semantics
- Identify judgments and reduction rules
- Apply reduction rules to execute a program

A Small Programming Language

Language Constructs

- Integer expressions
- Statements
  - Assignment
  - Statement lists
  - Conditional
  - Loop

Example Program: Factorial

```
1 n := -5;
2 if n>=0
3   then i := n
4   else i := 0-n
5 fi;
6 f := 1;
7 while i do
8   f := f * i;
9   i := i-1
10 od
```

Operational Semantics

- Operational semantics defines meaning of programs relative to an *abstract machine*
- **Reduction machine**: operates on a program and reduces it to its semantic "value"
  - Uses a store ξ (e.g., a map from variables to values)
  - State of the machine is a program or value and the store $\langle \mathtt{prg}, \xi \rangle$ or $\langle v, \xi \rangle$
  - Judgments: $\langle \mathtt{prg}, \xi \rangle \Downarrow \langle v, \xi' \rangle$

    > Executing program `prg` in state ξ *yields* ($\Downarrow$) value v and new state ξ'

  - Reduction rules: $\frac{\text{premise}_1, \ldots, \text{premise}_n}{\text{conclusion}}$

    > Conclusion follows from all premises satisfied

Language of Integer Expressions

Syntax

```
1 Expr ::= Number
2         | Expr '+' Expr
3         | Expr '-' Expr
4         | Expr '*' Expr
5         | '(' Expr ')'
```

Example

```
1 2+3*4
2 (2+3) * 4
```

Semantics

- Semantic domain: integer arithmetic
- Value v is the meaning of an expression
  - For example, $(2 + 3) * 4$ means 20
- Numbers evaluate to their value

$$\frac{}{\langle \mathtt{n}, \xi \rangle \Downarrow \langle n, \xi \rangle}\ (\text{Num})$$

- Operations map to integer operations

$$\frac{\langle \mathtt{e}_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle \mathtt{e}_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \mathtt{e}_1 \mathtt{+} \mathtt{e}_2, \xi_0 \rangle \Downarrow \langle v_1 + v_2, \xi_2 \rangle}\ (\text{Add}), \ldots, \frac{\langle \mathtt{e}, \xi \rangle \Downarrow \langle v, \xi_1 \rangle}{\langle \mathtt{(e)}, \xi \rangle \Downarrow \langle v, \xi_1 \rangle}\ (\text{Par})$$

Reduction Example: Arithmetic Expression

Rules

- $\dfrac{}{\langle \mathtt{n}, \xi \rangle \Downarrow \langle n, \xi \rangle}\ (\text{Num})$
- $\dfrac{\langle \mathtt{e}_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle \mathtt{e}_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \mathtt{e}_1 \mathtt{+} \mathtt{e}_2, \xi_0 \rangle \Downarrow \langle v_1 + v_2, \xi_2 \rangle}\ (\text{Add})$
- $\dfrac{\langle \mathtt{e}, \xi \rangle \Downarrow \langle v, \xi_1 \rangle}{\langle \mathtt{(e)}, \xi \rangle \Downarrow \langle v, \xi_1 \rangle}\ (\text{Par})$

- Reduce 2 + 3 to its semantic value
- Deduction tree

$$\dfrac{\dfrac{}{\langle 2, \xi \rangle \Downarrow \langle 2, \xi \rangle}(\text{Num}) \qquad \dfrac{}{\langle 3, \xi \rangle \Downarrow \langle 3, \xi \rangle}(\text{Num})}{\langle 2 + 3, \xi \rangle \Downarrow \langle 5, \xi \rangle}(\text{Add})$$

Reduction Exercises: Arithmetic Expression

Rules

- $\dfrac{}{\langle \mathtt{n}, \xi \rangle \Downarrow \langle n, \xi \rangle}\ (\text{Num})$
- $\dfrac{\langle \mathtt{e}_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle \mathtt{e}_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \mathtt{e}_1 \mathtt{+} \mathtt{e}_2, \xi_0 \rangle \Downarrow \langle v_1 + v_2, \xi_2 \rangle}\ (\text{Add})$
- $\dfrac{\langle \mathtt{e}, \xi \rangle \Downarrow \langle v, \xi_1 \rangle}{\langle \mathtt{(e)}, \xi \rangle \Downarrow \langle v, \xi_1 \rangle}\ (\text{Par})$

- Define the rule for multiplication
- Reduce $\mathtt{(2 + 3) * 4}$ to its semantic value

$$\dfrac{\dfrac{\text{see previous slide: } \langle \mathtt{2 + 3}, \xi \rangle \Downarrow \langle \mathtt{5}, \xi \rangle}{\langle \mathtt{(2 + 3)}, \xi \rangle \Downarrow \langle \mathtt{5}, \xi \rangle}\ (\text{Par}) \qquad \dfrac{}{\langle \mathtt{4}, \xi \rangle \Downarrow \langle \mathtt{4}, \xi \rangle}\ (\text{Num})}{\langle \mathtt{(2 + 3) * 4}, \xi \rangle \Downarrow \langle \mathtt{20}, \xi \rangle}\ (\text{Mul})$$

Implement a Language in Scala

- Implement grammar to represent abstract syntax tree as an algebraic data type

```
enum Expr:
  // Arithmetic expressions
  case Number(n:Int)
  case Plus(l:Expr, r:Expr)
  case Minus(l:Expr, r:Expr)
  case Times(l:Expr, r:Expr)

  // Identifiers a,...,x,y,z
  case Ident(c:Char)

  // Parenthesized expressions (e)
  case Par(e:Expr)
```

- Integer expression as an abstract syntax tree: 2+3*x

```
val expr =
  Plus(
    Number(2),
    Times(Number(3), Ident('x'))
  )
```

Implement a Language in Scala

- Implement an interpreter following the formal semantics
- Define a store for the values of variables

```
1 type Store = Map[Char, Int]
```

- Define the interpreter signature (follows from shape of judgments: $\langle \mathbf{e}, \xi \rangle \Downarrow \langle v, \xi' \rangle$)

```
1 def eval(e: Expr, s: Store): (Int, Store)
```

Implement a Language in Scala

- Reduction rules for integer expressions
- Numbers

$$\frac{}{\langle \mathtt{n}, \xi \rangle \Downarrow \langle n, \xi \rangle}\ (\text{Num})$$

- Addition etc.

$$\frac{\langle \mathtt{e}_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle \mathtt{e}_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \mathtt{e}_1 \mathtt{+} \mathtt{e}_2, \xi_0 \rangle \Downarrow \langle v_1 + v_2, \xi_2 \rangle}\ (\text{Add}), \ldots$$

- Parentheses

$$\frac{\langle \mathtt{e}, \xi \rangle \Downarrow \langle v, \xi_1 \rangle}{\langle \mathtt{(e)}, \xi \rangle \Downarrow \langle v, \xi_1 \rangle}\ (\text{Par})$$

```scala
def eval(e: Expr, s: Store): (Int, Store) = e match
  case Number(n)      => (n, s)

  case Plus(e1, e2)   =>
    val (v1, s1) = eval(e1, s)
    val (v2, s2) = eval(e2, s1)
    (v1+v2, s2)

  ...

  case Par(e)         => eval(e, s)
```

Assignments and Sequential Composition

Syntax

```
Expr   ::= Number
         | Expr '+' Expr
         | Expr '-' Expr
         | Expr '*' Expr
         | Ident
         | PrgSeq
         | '(' Expr ')'
Ident  ::= 'a' | 'b' | ... | 'z'
PrgSeq ::= Prg | Prg ';' PrgSeq
Prg    ::= Ident ':=' Expr
```

Example

```
a := 2+3;
b := (a:=a+1)*4;
a := b-5
```

Semantics

❓ How do we store/look up values of variables?

- Look up variable value in store

$$\frac{}{\langle \mathtt{x}, \xi \rangle \Downarrow \langle \xi(\mathtt{x}), \xi \rangle}\ (\mathrm{Var})$$

- Assignment: evaluate expression, then update state

$$\frac{\langle \mathtt{e}, \xi_0 \rangle \Downarrow \langle v, \xi_1 \rangle}{\langle \mathtt{x} := \mathtt{e}, \xi_0 \rangle \Downarrow \langle v, \xi_1\{\mathtt{x} \mapsto v\} \rangle}\ (:=)$$

- Sequence: evaluate subexpressions, chain results

$$\frac{\langle \mathtt{e}_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \qquad \langle \mathtt{e}_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \mathtt{e}_1 ; \mathtt{e}_2, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle}\ (;)$$

Implement a Language in Scala

- Extend algebraic data type with assignment and sequential composition

```
enum Expr:
  // Arithmetic expressions
  case Number(n:Int)
  case Plus(l:Expr, r:Expr)
  case Minus(l:Expr, r:Expr)
  case Times(l:Expr, r:Expr)

  // Identifiers a,...,x,y,z
  case Ident(c:Char)

  // Parenthesized expressions (e)
  case Par(e:Expr)
```

```
  // Programs

  // Assignment x := v
  case Assign(x:Ident, v:Expr)

  // Sequential composition p ; r
  case Seq(p:Expr, r:Expr)
end Expr
```

Implement a Language in Scala

- Reduction rules for assignment
- Variable lookup

$$\frac{}{\langle \mathtt{x}, \xi \rangle \Downarrow \langle \xi(\mathtt{x}), \xi \rangle} \text{(Var)}$$

- Assignment

$$\frac{\langle \mathtt{e}, \xi_0 \rangle \Downarrow \langle v, \xi_1 \rangle}{\langle \mathtt{x} := \mathtt{e}, \xi_0 \rangle \Downarrow \langle v, \xi_1\{\mathtt{x} \mapsto v\} \rangle} (:=)$$

- Sequential expression composition

$$\frac{\langle \mathtt{e}_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \qquad \langle \mathtt{e}_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \mathtt{e}_1 \,;\, \mathtt{e}_2, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} (;)$$

```scala
def eval(e: Expr, s: Store): (Int, Store) = e match
  ...

  case Ident(c)       => (s(c), s)

  case Assign(Ident(x), e)   =>
    val (v,s1) = eval(e, s)
    (v, s1.updated(x,v))

  case Seq(e1, e2)    =>
    val (v1, s1) = eval(e1, s)
    val (v2, s2) = eval(e2, s1)
    (v2, s2)
```

Summary

- **Operational semantics**: defines language in terms of operations of an abstract machine
  - Alternative semantics definitions: denotational semantics, axiomatic semantics
- **Judgments** and **reduction rules**: describe steps of the abstract machine, expressed as conclusions from premises
- **Deduction tree**: makes conclusions about program from the meaning of the program's components

Conditionals and Loops

Syntax

```
1  Expr   ::= Number
2           | Expr '+' Expr
3           | Expr '-' Expr
4           | Expr '*' Expr
5           | Ident
6           | Expr '>=' Expr
7           | PrgSeq
8           | '(' Expr ')'
9  Ident  ::= 'a' | 'b' | ... | 'z'
10 PrgSeq ::= Prg | Prg ';' PrgSeq
11 Prg    ::= Ident ':=' Expr
12          | 'if' Expr
13              'then' Expr
14              'else' Expr 'fi'
15          | 'while' Expr 'do'
16              Expr
17            'od'
```

Semantics

- Conditional
  - True: $$\frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1 \neq 0 \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{if\ e}_1\ \mathtt{then\ e}_2\ \mathtt{else\ e}_3\ \mathtt{fi},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}\ (\text{iftrue})$$
  - False: $$\frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1 = 0 \quad \langle \mathtt{e}_3,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{if\ e}_1\ \mathtt{then\ e}_2\ \mathtt{else\ e}_3\ \mathtt{fi},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}\ (\text{iffalse})$$
- Loop
  - End: $$\frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1 = 0}{\langle \mathtt{while\ e}_1\ \mathtt{do\ e}_2\ \mathtt{od},\xi_0\rangle \Downarrow \langle 0,\xi_1\rangle}\ (\text{whileend})$$
  - Recurse: $$\frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1 \neq 0 \quad \langle \mathtt{e}_2;\mathtt{while\ e}_1\ \mathtt{do\ e}_2\ \mathtt{od},\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{while\ e}_1\ \mathtt{do\ e}_2\ \mathtt{od},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}\ (\text{whilerec})$$

Implement a Language in Scala

- Extend algebraic data type with comparison, conditional, and loop

```
enum Expr:
  // Arithmetic expressions
  case Number(n:Int)
  case Plus(l:Expr, r:Expr)
  case Minus(l:Expr, r:Expr)
  case Times(l:Expr, r:Expr)

  // Identifiers a,...,x,y,z
  case Ident(c:Char)

  // Comparisons l>=r
  case GEq(l:Expr, r:Expr)

  // Parenthesized expressions (e)
  case Par(e:Expr)
```

```
  // Programs

  // Assignment x := v
  case Assign(x:Ident, v:Expr)

  // Conditional if p then t else e
  case If(p:Expr, t:Expr, e:Expr)

  // Loop while p do b
  case While(p:Expr, b:Expr)

  // Sequential composition p ; r
  case Seq(p:Expr, r:Expr)
end Expr
```

Implement a Language in Scala

- Express a program as an abstract syntax tree: absolute value of an expression

```
1 val abs = (n: Expr) =>
2   Seq(
3     Assign(Ident('i'), n),                                  // i := <n>
4     If(                                                     // if
5       GEq(Ident('i'), Number(0)),                           //   i >= 0
6       Assign(Ident('i'), Ident('i')),                       //   then i := i
7       Assign(Ident('i'), Minus(Number(0), Ident('i')))      //   else i := 0-i
8     )                                                       // fi
9   )
```

Implement a Language in Scala

Reduction rules for conditionals and loops

- Comparison

$$\frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{e}_1\mathtt{>=}\mathtt{e}_2,\xi_0\rangle \Downarrow \langle \max(0,v_1-v_2+1),\xi_2\rangle}\ (\geq)$$

- Conditional

$$\frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1\neq 0 \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{if\ e}_1\ \mathtt{then\ e}_2\ \mathtt{else\ e}_3\ \mathtt{fi},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}\ (\text{iftrue})$$

$$\frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1=0 \quad \langle \mathtt{e}_3,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{if\ e}_1\ \mathtt{then\ e}_2\ \mathtt{else\ e}_3\ \mathtt{fi},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}\ (\text{iffalse})$$

- Loop

$$\frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1=0}{\langle \mathtt{while\ e}_1\ \mathtt{do\ e}_2\ \mathtt{od},\xi_0\rangle \Downarrow \langle 0,\xi_1\rangle}\ (\text{whileend})$$

$$\frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1\neq 0 \quad \langle \mathtt{e}_2\mathtt{;while\ e}_1\ \mathtt{do\ e}_2\ \mathtt{od},\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{while\ e}_1\ \mathtt{do\ e}_2\ \mathtt{od},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}\ (\text{whilerec})$$

```scala
def eval(e: Expr, s: Store): (Int, Store) = e match
  ...

  case GEq(e1, e2)    =>
    val (v1, s1) = eval(e1, s)
    val (v2, s2) = eval(e2, s1)
    (math.max(0,v1-v2+1), s2)

  case If(e1, e2, e3) =>
    val (v1, s1) = eval(e1, s)
    if v1!=0
      then eval(e2, s1)
      else eval(e3, s1)

  case w@While(e1, e2)  =>
    val (v1, s1) = eval(e1, s)
    if v1==0
      then (0, s1)
      else eval(Seq(e2, w), s1)
```

Program Example

- Program as an abstract syntax tree: absolute value of an expression

```
1 val abs = (n: Expr) =>
2   Seq(
3     Assign(Ident('i'), n),                                  // i := <n>
4     If(                                                     // if
5       GEq(Ident('i'), Number(0)),                           //   i >= 0
6       Assign(Ident('i'), Ident('i')),                       //   then i := i
7       Assign(Ident('i'), Minus(Number(0), Ident('i')))      //   else i := 0-i
8     )                                                       // fi
9   )
```

Program Reduction Example: Absolute Value

```
1 i := -2;
2 if i>=0
3   then i := i
4   else i := 0-i
5 fi
```

- Integer arithmetic
  - $\dfrac{}{\langle \mathtt{n},\xi\rangle \Downarrow \langle n,\xi\rangle}$ (Num)
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{e}_1\mathtt{+}\mathtt{e}_2,\xi_0\rangle \Downarrow \langle v_1 - v_2,\xi_2\rangle}$ (Sub)
- Assignment
  - $\dfrac{\langle \mathtt{e},\xi_0\rangle \Downarrow \langle v,\xi_1\rangle}{\langle \mathtt{x}\mathtt{:=}\mathtt{e},\xi_0\rangle \Downarrow \langle v,\xi_1\{\mathtt{x}\mapsto v\}\rangle}$ (:=)
- Sequential composition
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{e}_1\,\mathtt{;}\,\mathtt{e}_2,\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}$ (;)
- Comparison
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{e}_1\mathtt{>=}\mathtt{e}_2,\xi_0\rangle \Downarrow \langle \max(0,v_1-v_2+1),\xi_2\rangle}$ ($\geq$)
- Conditional
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1\neq 0 \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3\ \mathtt{fi},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}$ (iftrue)
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1=0 \quad \langle \mathtt{e}_3,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3\ \mathtt{fi},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}$ (iffalse)

Program Reduction Example: Absolute Value

$$\dfrac{\dfrac{\dfrac{}{\langle -2, \{\}\rangle \Downarrow \langle -2, \{\}\rangle}(\text{Num})}{\langle \texttt{i := } -2\rangle \Downarrow \langle -2, \{\texttt{i} \mapsto -2\}\rangle}(:=) \qquad \dfrac{\text{Lemma } \texttt{i >= 0} \qquad 0 = 0 \qquad \text{Lemma } \texttt{i := 0 - i}}{\langle \texttt{if i >= 0 then i := i else i := 0 - i fi}\rangle \Downarrow \langle 2, \{\texttt{i} \mapsto 2\}\rangle}(\text{iffalse})}{\langle \texttt{i := } -2\texttt{; if i >= 0 then i := i else i := 0 - i fi}, \{\}\rangle \Downarrow \langle 2, \{\texttt{i} \mapsto 2\}\rangle}(;)$$

- Lemma `i >= 0`, where $\xi = \{\texttt{i} \mapsto -2\}$

$$\dfrac{\dfrac{}{\langle \texttt{i}, \xi\rangle \Downarrow \langle -2, \xi\rangle}(\text{Var}) \qquad \dfrac{}{\langle 0, \xi\rangle \Downarrow \langle 0, \xi\rangle}(\text{Num})}{\langle \texttt{i >= 0}, \xi\rangle \Downarrow \langle 0, \xi\rangle}(\geq)$$

- Lemma `i := 0 - i`, $\xi = \{\texttt{i} \mapsto -2\}$

$$\dfrac{\dfrac{\dfrac{}{\langle 0, \xi\rangle \Downarrow \langle 0, \xi\rangle}(\text{Num}) \qquad \dfrac{}{\langle \texttt{i}, \xi\rangle \Downarrow \langle -2, \xi\rangle}(\text{Var})}{\langle \texttt{0 - i}, \xi\rangle \Downarrow \langle 2, \xi\rangle}(\text{Sub})}{\langle \texttt{i := 0 - i}, \xi\rangle \Downarrow \langle 2, \xi\{\texttt{i} \mapsto 2\}\rangle}(:=)$$

Program Reduction Example: Factorial

```
 1 n := -5;
 2 if n>=0
 3   then i := n
 4   else i := 0-n
 5 fi;
 6 f := 1;
 7 while i do
 8   f := f*i;
 9   i := i-1
10 od
```

- Integer arithmetic
  - $\dfrac{}{\langle \mathtt{n},\xi\rangle \Downarrow \langle n,\xi\rangle}$ (Num)
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{e}_1 - \mathtt{e}_2,\xi_0\rangle \Downarrow \langle v_1 - v_2,\xi_2\rangle}$ (Sub)
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{e}_1 * \mathtt{e}_2,\xi_0\rangle \Downarrow \langle v_1 \cdot v_2,\xi_2\rangle}$ (Mul)
- Assignment
  - $\dfrac{\langle \mathtt{e},\xi_0\rangle \Downarrow \langle v,\xi_1\rangle}{\langle \mathtt{x} := \mathtt{e},\xi_0\rangle \Downarrow \langle v,\xi_1\{\mathtt{x} \mapsto v\}\rangle}$ (:=)
- Sequential composition
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{e}_1\,;\mathtt{e}_2,\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}$ (;)
- Conditional
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{e}_1 \texttt{>=} \mathtt{e}_2,\xi_0\rangle \Downarrow \langle \max(0,v_1 - v_2 + 1),\xi_2\rangle}$ ($\geq$)
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1 \neq 0 \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3\ \mathtt{fi},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}$ (iftrue)
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1 = 0 \quad \langle \mathtt{e}_3,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3\ \mathtt{fi},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}$ (iffalse)
- Loop
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1 = 0}{\langle \mathtt{while}\ \mathtt{e}_1\ \mathtt{do}\ \mathtt{e}_2\ \mathtt{od},\xi_0\rangle \Downarrow \langle 0,\xi_1\rangle}$ (whileend)
  - $\dfrac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad v_1 \neq 0 \quad \langle \mathtt{e}_2;\mathtt{while}\ \mathtt{e}_1\ \mathtt{do}\ \mathtt{e}_2\ \mathtt{od},\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{while}\ \mathtt{e}_1\ \mathtt{do}\ \mathtt{e}_2\ \mathtt{od},\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}$ (whilerec)

Program Reduction Example: Factorial

$$\dfrac{\dfrac{}{\langle -5, \{\}\rangle \Downarrow \langle -5, \{\}\rangle}(\text{Num})}{\langle \texttt{n := } -5, \{\}\rangle \Downarrow \langle -5, \{\texttt{n} \mapsto -5\}\rangle}(:=) \qquad \dfrac{\text{Lemma } \texttt{if...fi} \qquad \dfrac{\text{Lemma } \texttt{f := 1} \qquad \text{Lemma } \texttt{while...od}(5)}{\langle \texttt{f := 1; while...od}, \{\texttt{n} \mapsto -5, \texttt{i} \mapsto 5\}\rangle \Downarrow \langle 0, \{\texttt{n} \mapsto -5, \texttt{i} \mapsto 0, \texttt{f} \mapsto 120\}\rangle}(;)}{\langle \texttt{if...fi; f := 1; while...od}, \{\texttt{n} \mapsto -5\}\rangle \Downarrow \langle 0, \{\texttt{n} \mapsto -5, \texttt{i} \mapsto 0, \texttt{f} \mapsto 120\}\rangle}(;)$$

$$\langle \texttt{n := } -5\texttt{; if...od}, \{\}\rangle \Downarrow \langle 0, \{\texttt{n} \mapsto -5, \texttt{i} \mapsto 0, \texttt{f} \mapsto 120\}\rangle \quad (;)$$

- Lemma `if...fi`, where $\xi = \{\texttt{n} \mapsto -5\}$

$$\dfrac{\dfrac{\dfrac{}{\langle \texttt{n}, \xi\rangle \Downarrow \langle -5, \xi\rangle}(\text{Var}) \qquad \dfrac{}{\langle 0, \xi\rangle \Downarrow \langle 0, \xi\rangle}(\text{Num})}{\langle \texttt{n >= 0}, \xi\rangle \Downarrow \langle 0, \xi\rangle}(\geq) \qquad 0{=}0 \qquad \dfrac{\dfrac{\dfrac{}{\langle 0, \xi\rangle \Downarrow \langle 0, \xi\rangle}(\text{Num}) \qquad \dfrac{}{\langle \texttt{n}, \xi\rangle \Downarrow \langle -5, \xi\rangle}(\text{Var})}{\langle \texttt{0-n}, \xi\rangle \Downarrow \langle 5, \xi\{\texttt{i} \mapsto 5\}\rangle}(\text{Sub})}{\langle \texttt{i := 0-n}, \xi\rangle \Downarrow \langle 5, \xi\{\texttt{i} \mapsto 5\}\rangle}(:=)}{\langle \texttt{if n >= 0 then i := n else i := 0-n fi}, \xi\rangle \Downarrow \langle 5, \xi\{\texttt{i} \mapsto 5\}\rangle}(\text{iffalse})$$

- Lemma `f := 1`, where $\xi = \{\texttt{n} \mapsto -5, \texttt{i} \mapsto 5\}$

$$\dfrac{\dfrac{}{\langle 1, \xi\rangle \Downarrow \langle 1, \xi\rangle}(\text{Num})}{\langle \texttt{f := 1}, \xi\rangle \Downarrow \langle 1, \xi\{\texttt{f} \mapsto 1\}\rangle}(:=)$$

Program Reduction Example: Factorial

- Lemma $\texttt{while}\ldots\texttt{od}(5)$, where $\xi = \{\texttt{n} \mapsto -5, \texttt{i} \mapsto 5, \texttt{f} \mapsto 1\}$

$$\dfrac{\dfrac{}{\langle \texttt{i}, \xi\rangle \Downarrow \langle 5, \xi\rangle}(\text{Var}) \qquad 5 \neq 0 \qquad \dfrac{\text{Lemma } \texttt{f := f * i} \qquad \dfrac{\text{Lemma } \texttt{i := i - 1} \qquad \text{Lemma } \texttt{while}\ldots\texttt{od}(4)}{\langle \texttt{i := i - 1; while i do}\ldots\texttt{od}, \xi\{\texttt{f} \mapsto 5\}\rangle \Downarrow \langle 0, \xi\{\texttt{i} \mapsto 0, \texttt{f} \mapsto 120\}\rangle}(;)}{\langle \texttt{f := f * i; i := i - 1; while i do}\ldots\texttt{od}, \xi\rangle \Downarrow \langle 0, \xi\{\texttt{i} \mapsto 0, \texttt{f} \mapsto 120\}\rangle}(;)}{\langle \texttt{while i do f := f * i; i := i - 1 od}, \xi\rangle \Downarrow \langle 0, \xi\{\texttt{i} \mapsto 0, \texttt{f} \mapsto 120\}\rangle}(\text{whilerec})$$

- Lemma $\texttt{f := f * i}$

$$\dfrac{\dfrac{}{\langle \texttt{f}, \xi\rangle \Downarrow \langle 1, \xi\rangle}(\text{Var}) \qquad \dfrac{}{\langle \texttt{i}, \xi\rangle \Downarrow \langle 5, \xi\rangle}(\text{Var})}{\langle \texttt{f} * \texttt{i}, \xi\rangle \Downarrow \langle 1 \cdot 5, \xi\rangle}(\text{Mul})}{\langle \texttt{f := f * i}, \xi\rangle \Downarrow \langle 5, \xi\{\texttt{f} \mapsto 5\}\rangle}(:=)$$

- Lemma $\texttt{i := i - 1}$

$$\dfrac{\dfrac{}{\langle \texttt{i}, \xi\rangle \Downarrow \langle \xi(\texttt{i}), \xi\rangle}(\text{Var}) \qquad \dfrac{}{\langle 1, \xi\rangle \Downarrow \langle 1, \xi\rangle}(\text{Num})}{\langle \texttt{i} - 1, \xi\rangle \Downarrow \langle 5 - 1, \xi\rangle}(\text{Sub})}{\langle \texttt{i := i - 1}, \xi\rangle \Downarrow \langle 4, \xi\{\texttt{i} \mapsto 4\}\rangle}(:=)$$

- Lemma $\texttt{while}\ldots\texttt{od(0)}$, where $\xi = \{\texttt{n} \mapsto -5, \texttt{i} \mapsto 0, \texttt{f} \mapsto 120\}$

$$\dfrac{\dfrac{}{\langle \texttt{i}, \xi\rangle \Downarrow \langle 0, \xi\rangle}(\text{Var}) \qquad 0 = 0}{\langle \texttt{while i do f := f * i; i := i - 1 od}, \xi\rangle \Downarrow \langle 0, \xi\rangle}(\text{whileend})$$