

CSC 347 - Concepts of Programming Languages

Formal Semantics

Instructor: Stefan Mitsch



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 $\langle \text{prg}, \xi \rangle$ or $\langle v, \xi \rangle$
 - Judgments: $\langle \text{prg}, \xi \rangle \Downarrow \langle v, \xi' \rangle$
 - Executing program **prg** in state ξ *yields* ($\Downarrow$) value v and new state ξ'
 - Reduction rules:
$$\frac{\text{premise}_1, \dots, \text{premise}_n}{\text{conclusion}}$$
 - Conclusion follows from all premisses 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

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

- Operations map to integer operations

$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 + e_2, \xi_0 \rangle \Downarrow \langle v_1 + v_2, \xi_2 \rangle} \text{ (Add)}, \dots, \frac{\langle e, \xi \rangle \Downarrow \langle v, \xi_1 \rangle}{\langle (e), \xi \rangle \Downarrow \langle v, \xi_1 \rangle} \text{ (Par)}$$



Reduction Example: Arithmetic Expression

Rules

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

- $$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 + e_2, \xi_0 \rangle \Downarrow \langle v_1 + v_2, \xi_2 \rangle} \text{ (Add)}$$

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

- Reduce $2 + 3$ to its semantic value

-  Deduction tree

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



Reduction Exercises: Arithmetic Expression

Rules

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

- $$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 + e_2, \xi_0 \rangle \Downarrow \langle v_1 + v_2, \xi_2 \rangle} \text{ (Add)}$$

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

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

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



Implement a Language in Scala

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

```
1 enum Expr:
2   // Arithmetic expressions
3   case Number(n:Int)
4   case Plus(l:Expr, r:Expr)
5   case Minus(l:Expr, r:Expr)
6   case Times(l:Expr, r:Expr)
7
8   // Identifiers a,...,x,y,z
9   case Ident(c:Char)
10
11   // Parenthesized expressions (e)
12   case Par(e:Expr)
```

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

```
1 val expr =
2   Plus(
3     Number(2),
4     Times(Number(3), Ident('x'))
5   )
```




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: $\langle e, \xi \rangle \Downarrow \langle v, \xi' \rangle$)

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



Implement a Language in Scala

- Reduction rules for integer expressions

- Numbers

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

- Addition etc.

$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 + e_2, \xi_0 \rangle \Downarrow \langle v_1 + v_2, \xi_2 \rangle} \text{ (Add), } \dots$$

- Parentheses

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

```
1 def eval(e: Expr, s: Store): (Int, Store) = e match
2   case Number(n)      => (n, s)
3
4   case Plus(e1, e2)   =>
5     val (v1, s1) = eval(e1, s)
6     val (v2, s2) = eval(e2, s1)
7     (v1+v2, s2)
8
9   ...
10
11  case Par(e)          => eval(e, s)
```



Assignments and Sequential Composition

Syntax

```

1 Expr ::= Number
2       | Expr '+' Expr
3       | Expr '-' Expr
4       | Expr '*' Expr
5       | Ident
6       | PrgSeq
7       | '(' Expr ')'
8 Ident ::= 'a' | 'b' | ... | 'z'
9 PrgSeq ::= Prg | Prg ';' PrgSeq
10 Prg    ::= Ident ':' Expr
    
```

Example

```

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

Semantics

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

- Look up variable value in store

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

- Assignment: evaluate expression, then update state

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

- Sequence: evaluate subexpressions, chain results

$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 ; e_2, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (;)}$$



Implement a Language in Scala

- Extend algebraic data type with assignment and sequential composition

```
1 enum Expr:
2   // Arithmetic expressions
3   case Number(n:Int)
4   case Plus(l:Expr, r:Expr)
5   case Minus(l:Expr, r:Expr)
6   case Times(l:Expr, r:Expr)
7
8   // Identifiers a,...,x,y,z
9   case Ident(c:Char)
10
11   // Parenthesized expressions (e)
12   case Par(e:Expr)
```

```
1   // Programs
2
3   // Assignment x := v
4   case Assign(x:Ident, v:Expr)
5
6   // Sequential composition p ; r
7   case Seq(p:Expr, r:Expr)
8 end Expr
```



Implement a Language in Scala

- Reduction rules for assignment

- Variable lookup

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

- Assignment

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

- Sequential expression composition

$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 ; e_2, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (;)}$$

```
1 def eval(e: Expr, s: Store): (Int, Store) = e match
2   ...
3
4   case Ident(c)          => (s(c), s)
5
6   case Assign(Ident(x), e) =>
7     val (v, s1) = eval(e, s)
8     (v, s1.updated(x, v))
9
10  case Seq(e1, e2)       =>
11    val (v1, s1) = eval(e1, s)
12    val (v2, s2) = eval(e2, s1)
13    (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:
$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 \neq 0 \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \text{ fi}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (iftrue)}$$
- False:
$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 = 0 \quad \langle e_3, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \text{ fi}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (iffalse)}$$

- Loop

- End:
$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 = 0}{\langle \text{while } e_1 \text{ do } e_2 \text{ od}, \xi_0 \rangle \Downarrow \langle 0, \xi_1 \rangle} \text{ (whileend)}$$
- Recurse:
$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 \neq 0 \quad \langle e_2; \text{while } e_1 \text{ do } e_2 \text{ od}, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{while } e_1 \text{ do } e_2 \text{ od}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (whilerec)}$$



Implement a Language in Scala

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

```
1 enum Expr:
2   // Arithmetic expressions
3   case Number(n:Int)
4   case Plus(l:Expr, r:Expr)
5   case Minus(l:Expr, r:Expr)
6   case Times(l:Expr, r:Expr)
7
8   // Identifiers a,...,x,y,z
9   case Ident(c:Char)
10
11   // Comparisons l>=r
12   case GEq(l:Expr, r:Expr)
13
14   // Parenthesized expressions (e)
15   case Par(e:Expr)
```

```
1   // Programs
2
3   // Assignment x := v
4   case Assign(x:Ident, v:Expr)
5
6   // Conditional if p then t else e
7   case If(p:Expr, t:Expr, e:Expr)
8
9   // Loop while p do b
10  case While(p:Expr, b:Expr)
11
12  // Sequential composition p ; r
13  case Seq(p:Expr, r:Expr)
14 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

$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 \geq e_2, \xi_0 \rangle \Downarrow \langle \max(0, v_1 - v_2 + 1), \xi_2 \rangle} (\geq)$$

- Conditional

$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 \neq 0 \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \text{ fi}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} (\text{iftrue})$$

$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 = 0 \quad \langle e_3, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \text{ fi}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} (\text{iffalse})$$

- Loop

$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 = 0}{\langle \text{while } e_1 \text{ do } e_2 \text{ od}, \xi_0 \rangle \Downarrow \langle 0, \xi_1 \rangle} (\text{whileend})$$

$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 \neq 0 \quad \langle e_2; \text{while } e_1 \text{ do } e_2 \text{ od}, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{while } e_1 \text{ do } e_2 \text{ od}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} (\text{whilerec})$$

```

1 def eval(e: Expr, s: Store): (Int, Store) = e match
2   ...
3
4   case GEq(e1, e2) =>
5     val (v1, s1) = eval(e1, s)
6     val (v2, s2) = eval(e2, s1)
7     (math.max(0, v1 - v2 + 1), s2)
8
9   case If(e1, e2, e3) =>
10    val (v1, s1) = eval(e1, s)
11    if v1 != 0
12      then eval(e2, s1)
13      else eval(e3, s1)
14
15   case w@While(e1, e2) =>
16    val (v1, s1) = eval(e1, s)
17    if v1 == 0
18      then (0, s1)
19      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

- $$\frac{}{\langle n, \xi \rangle \Downarrow \langle n, \xi \rangle} \text{ (Num)}$$
- $$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 + e_2, \xi_0 \rangle \Downarrow \langle v_1 - v_2, \xi_2 \rangle} \text{ (Sub)}$$

- Assignment

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

- Sequential composition

- $$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 ; e_2, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (;)}$$

- Comparison

- $$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 >= e_2, \xi_0 \rangle \Downarrow \langle \max(0, v_1 - v_2 + 1), \xi_2 \rangle} (\geq)$$

- Conditional

- $$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 \neq 0 \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \text{ fi}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (iftrue)}$$
- $$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 = 0 \quad \langle e_3, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \text{ fi}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (iffalse)}$$



Program Reduction Example: Absolute Value

$$\frac{\frac{\overline{\langle -2, \{\} \rangle \Downarrow \langle -2, \{\} \rangle}(\text{Num})}{\langle i := -2 \rangle \Downarrow \langle -2, \{i \mapsto -2\} \rangle}(\text{:=}) \quad \frac{\text{Lemma } i \geq 0 \quad 0 = 0 \quad \text{Lemma } i := 0 - i}{\langle \text{if } i \geq 0 \text{ then } i := i \text{ else } i := 0 - i \text{ fi} \rangle \Downarrow \langle 2, \{i \mapsto 2\} \rangle}(\text{iffalse})}{\langle i := -2; \text{ if } i \geq 0 \text{ then } i := i \text{ else } i := 0 - i \text{ fi}, \{\} \rangle \Downarrow \langle 2, \{i \mapsto 2\} \rangle}(\text{;})$$

- Lemma $i \geq 0$, where $\xi = \{i \mapsto -2\}$
- Lemma $i := 0 - i, \xi = \{i \mapsto -2\}$

$$\frac{\frac{\overline{\langle i, \xi \rangle \Downarrow \langle -2, \xi \rangle}(\text{Var})}{\langle i \geq 0, \xi \rangle \Downarrow \langle 0, \xi \rangle}(\text{:=}) \quad \frac{\overline{\langle 0, \xi \rangle \Downarrow \langle 0, \xi \rangle}(\text{Num})}{\langle 0, \xi \rangle \Downarrow \langle 0, \xi \rangle}(\geq)}$$

$$\frac{\frac{\overline{\langle 0 - i, \xi \rangle \Downarrow \langle 2, \xi \rangle}(\text{Num})}{\langle i := 0 - i, \xi \rangle \Downarrow \langle 2, \xi \{i \mapsto 2\} \rangle}(\text{:=}) \quad \frac{\overline{\langle i, \xi \rangle \Downarrow \langle -2, \xi \rangle}(\text{Var})}{\langle i, \xi \rangle \Downarrow \langle -2, \xi \rangle}(\text{Sub})$$



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

- $\frac{}{\langle n, \xi \rangle \Downarrow \langle n, \xi \rangle} \text{ (Num)}$
- $\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 - e_2, \xi_0 \rangle \Downarrow \langle v_1 - v_2, \xi_2 \rangle} \text{ (Sub)}$
- $\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 * e_2, \xi_0 \rangle \Downarrow \langle v_1 \cdot v_2, \xi_2 \rangle} \text{ (Mul)}$

- Assignment

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

- Sequential composition

- $\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 ; e_2, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (;)}$

- Conditional

- $\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 >= e_2, \xi_0 \rangle \Downarrow \langle \max(0, v_1 - v_2 + 1), \xi_2 \rangle} (\geq)$
- $\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 \neq 0 \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \text{ fi}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (iftrue)}$
- $\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 = 0 \quad \langle e_3, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \text{ fi}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (iffalse)}$

- Loop

- $\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 = 0}{\langle \text{while } e_1 \text{ do } e_2 \text{ od}, \xi_0 \rangle \Downarrow \langle 0, \xi_1 \rangle} \text{ (whileend)}$
- $\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad v_1 \neq 0 \quad \langle e_2 ; \text{while } e_1 \text{ do } e_2 \text{ od}, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle \text{while } e_1 \text{ do } e_2 \text{ od}, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} \text{ (whilerec)}$



Program Reduction Example: Factorial

$$\begin{array}{c}
 \frac{}{\langle -5, \{\} \rangle \Downarrow \langle -5, \{\} \rangle} \text{(Num)} \\
 \frac{}{\langle n := -5, \{\} \rangle \Downarrow \langle -5, \{n \mapsto -5\} \rangle} \text{(:=)} \\
 \frac{}{\langle n := -5; \text{if} \dots \text{fi} \rangle \Downarrow \langle 0, \{n \mapsto -5, i \mapsto 0, f \mapsto 120\} \rangle} \text{(:)} \\
 \frac{}{\langle n := -5; \text{if} \dots \text{od}, \{\} \rangle \Downarrow \langle 0, \{n \mapsto -5, i \mapsto 0, f \mapsto 120\} \rangle} \text{(:)}
 \end{array}$$

- Lemma $\text{if} \dots \text{fi}$, where $\xi = \{n \mapsto -5\}$

$$\begin{array}{c}
 \frac{}{\langle n, \xi \rangle \Downarrow \langle -5, \xi \rangle} \text{(Var)} \quad \frac{}{\langle 0, \xi \rangle \Downarrow \langle 0, \xi \rangle} \text{(Num)} \quad \frac{}{\langle 0, \xi \rangle \Downarrow \langle 0, \xi \rangle} \text{(Num)} \quad \frac{}{\langle n, \xi \rangle \Downarrow \langle -5, \xi \rangle} \text{(Var)} \\
 \frac{}{\langle n \geq 0, \xi \rangle \Downarrow \langle 0, \xi \rangle} (\geq) \quad \frac{}{\langle 0 - n, \xi \rangle \Downarrow \langle 5, \xi \{i \mapsto 5\} \rangle} \text{(Sub)} \\
 \frac{}{\langle \text{if } n \geq 0 \text{ then } i := n \text{ else } i := 0 - n \text{ fi}, \xi \rangle \Downarrow \langle 5, \xi \{i \mapsto 5\} \rangle} \text{(iffalse)}
 \end{array}$$

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

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



Program Reduction Example: Factorial

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

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

- Lemma `f := f * i`

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

- Lemma `i := i - 1`

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

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

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