8

Semantic Interpretation

Exercises

Note: for many of these exercises, instructors will need to provide students with the Java sources for the Clite interpreter. These sources are available only at the instructor's web site for the book.

8.1 a. If `++` and `--` are allowed only as free-standing statements, the parser can convert statements of the form `a++` to the equivalent `a=a+1`. No changes to the abstract syntax are required.

b. With this approach, no changes to the semantics are required.

c. And no changes to the interpreter are required.

d. If these operators were allowed in expressions, then tests in loops and conditionals could change the state of a computation. Similarly, any occurrence of an expression, e.g., as the source expression in an assignment, could change the state of a computation.

Since the interpreter is based on single-valued mathematical functions, such a change would require major revision in the definition of semantics and in the interpreter itself.

8.2 a. The concrete syntax in Fig. 2.7 would have the following rules added:

$$\begin{aligned} \mathit{Statement} &\rightarrow \mathit{InputStatement} \mid \mathit{OutputStatement} \\ \mathit{InputStatement} &\rightarrow \text{cin } \mathit{InVarList} \text{ ;} \\ \mathit{OutputStatement} &\rightarrow \text{cout } \mathit{OutExprList} \text{ ;} \\ \mathit{InVarList} &\rightarrow \{ \texttt{>>} \mathit{Identifier} \} \\ \mathit{OutExprList} &\rightarrow \{ \texttt{<<} \mathit{Expression} \} \end{aligned}$$

b. The abstract syntax in Fig. 2.14 would have the following additions:

$$\mathit{Statement} = \mathit{Skip} \mid \ldots \mid \mathit{Loop} \mid \underline{\mathit{Input} \mid \mathit{Output}}$$

$$Input = Variable^*$$
$$Output = Exrpession^*$$

c. Instructors will need to make the Clite sources available to students for this question.

d. Instructors will need to make the Clite sources available to students for this question.

8.3 An advantage is that *undef* can be explicitly tested as a value in the program (e.g., to see if a variable has been initialized). A disadvantage is that programs operating with *undef* values can propagate some very obscure run-time errors before the program crashes or delivers incorrect results. A whole new "arithmetic" with undefined values would have to be understood by programmers and language designers alike in order to properly introduce *undef* into the semantic domain of a programming language.

Readers may note that the current Clite implementation permits such an *Assignment* but does not permit the value *undef* to be used in a *Binary* or *Unary* expression. The simplest way to prevent *undef* from being assigned is for the semantic rule for *Assignement* to disallow an *undef* source expression.

8.4 a. No additions are needed; see *MulOp* in Figure 2.7, p. 38.

b. In Figure 2.14, p. 53, add the operator to the list of *ArithmeticOp* operators.

c. This is already done. To assign this project, instructors will need to remove all references to the remainder operator in the lexer and parser sources, and then distribute the altered sources to the students.

d. Modify method *applyBinary* in `DynamicTyping.java` by adding a test for a remainder operator, and then add the appropriate computation. Hint: copy and edit the code for DIV for type INT.

8.5 a. Modify Rule 8.8, 5(a) so that the rule uses logical and.

b. Modify Rule 8.8, 5(b) so that the rule uses logical or.

c. Find the method `applyBinary` in class `Semantics` and make the corresponding changes.

8.6 In file `Semantics.java` find the `M` function for *Statement*; before the first `if`, insert a statement to display the *State*.

8.7 This exercise is harder than others becuase of the large number of changes required over many files. The changes themselves are fairly minor. First add a line number to the state of an *AbstractSyntax* or *Statement* object.

The display for a *State* can then be modified to add the line number to the display.

To get the line number into an *AbstarctSyntax* object requires adding the line number variable to a *Token* and modifying the *Lexer* to assign its value in the *Token*. Then, in constructing an *AbstractSyntax* object, the *Parser* should move the line number from the *Token* to the corresponding *AbstractSyntax* object.

8.8 Here is a trace table:

	Before	**Variables**		
Step	**Statement**	**i**	**a**	**z**
1	3	*undef*	*undef*	*undef*
2	4	5	*undef*	*undef*
3	5	5	2	*undef*
4	6	5	2	1
5	7	5	2	1
6	8	5	2	1
7	9	5	2	2
8	10	2	2	2
9	6	2	4	2
10	7	2	4	2
11	9	2	4	2
12	10	1	4	2
13	6	1	16	2
14	7	1	16	2
15	8	1	16	2
16	9	1	16	32
17	10	0	256	32
18	6	0	256	32
19	12	0	256	32

8.9 Parts a. and b. can be answered by running the Clite interpreter with this program as input.

a.
```
Assignment:
  Variable: z
  Binary:
    Operator: *
    Variable: z
    Variable: a
```

b.
```
Assignment:
  Variable: z
  Binary:
    Operator: INT*
    Variable: z
    Variable: a
```

c. The value of `z` and then `a` are fetched from the current state. Since the operator is `INT*`, an integer multiplication is performed on these two values, giving an integer value for the source expression of the assignment statement. This value is finally *onioned* to `z`, thus changing the current state.

8.10 Unfortunately, integer overflow is not an exception or error caught by the Java virtual machine. The easiest way to catch such errors is to use *long*s for performing arithmetic for the class `IntValue`. Then for each arithmetic operation, the result must be checked to see if it exceeds `Integer.MAX_VALUE` or `Integer.MIN_VALUE`. This would be done in `Semantics.java`

8.11 To support side-effects in an expression, the evaluation of an expression must not only return a value but also an updated state. Since the state object is passed as a parameter, an updated state can be returned through the parameter list. The operations on the *State* class have to be modified so that they update the current state rather than create a new one. This is particularly true for the `onion` function that takes a single *Identifier* and a new *Value* as paramters.

8.12 Here is a trace table.

	Before	**Variables**		
Step	**Statement**	**i**	**a**	**z**
1	2			
2	3	5		
3	4	5	2.0	
4	5	5	2.0	1.0
5	6	5	2.0	1.0
6	7	5	2.0	1.0
7	8	5	2.0	2.0
8	9	2	2.0	2.0
9	5	2	4.0	2.0
10	6	2	4.0	2.0
11	8	2	4.0	2.0
12	9	1	4.0	2.0
13	5	1	16.0	2.0
14	6	1	16.0	2.0
15	7	1	16.0	2.0
16	8	1	16.0	32.0
17	9	0	256.0	32.0
18	5	0	256.0	32.0
19	11	0	256.0	32.0

8.13 a. The following abstract syntax can be obtained by running the Clite interpreter with this program as input.

```
Assignment:
  Variable: z
  Binary:
    Operator: *
    Variable: z
    Variable: a
```

b. The values of the variables `z` and `a` are fetched from the current state. Since both variables are of type *FloatValue*, a floating point multiplication is performed resulting in a *FloatValue* for the expression.

8.14 To class *IntValue* in `AbstractSyntax.java`, add a method `floatValue` (overriding the default) that returns the integer value converted to floating point. Then in method `applyBinary` in `DynamicTyping.java` relax the initial check for equal types to allow one to be integer and the other to be floating point. Finally, modify the check for `v1.type` to be floating point so that either `v1` or `v2` can be floating point.

8.15 a. $\{ < x, 1 >, < y, 5 >, < z, 3 > \}$

b. $\{ < x, 1 >, < y, 2 >, < z, 3 >, < w, 1 > \}$

c. $\{ < y, 5 >, < w, 1 > \}$

d. $\{ < y, 5 > \}$

e. ϕ

f. ϕ

g. $\{ < x, 1 >, < y, 2 >, < z, 3 >, < w, 1 > \}$

8.16 a. $M((z + 2) * y, \{ < x, 2 >, < y, -3 >, < z, 75 > \}) =$
$M(M((z + 2), \{ < x, 2 >, < y, -3 >, < z, 75 > \}) * y,$
$\{ < x, 2 >, < y, -3 >, < z, 75 > \}) = \ldots =$
$M(77 * y, \{ < x, 2 >, < y, -3 >, < z, 75 > \}) = -231$
Note: some intermediate steps are elided.

b. $M(2 * x + 3/y - 4, \{ < x, 2 >, < y, -3 >, < z, 75 > \}) = \ldots =$
$M(4 + 3/y - 4, \{ < x, 2 >, < y, -3 >, < z, 75 > \}) = \ldots =$
$M(4 - 1 - 4, \{ < x, 2 >, < y, -3 >, < z, 75 > \}) = -1$
Note: some intermediate steps are elided.

c. 1

8.17 Again, some intermediate steps are elided:
$M(z = 2 * x + 3/y - 4, \{ < x, 6 >, < y, 12 >, < z, 75 > \}) = \ldots =$
$M(z = 9, \{ < x, 6 >, < y, 12 >, < z, 75 > \}) = \ldots =$
$\{ < x, 6 >, < y, 12 >, < z, 9 > \}$

8.18 The file `Semantics.java` in the Clite sources provides an answer to this question.

8.19 A state change will occur only if the source is not *undef*.

$$\begin{aligned} M(\mathit{Assignment}\ \mathrm{a}, \mathit{State}\ \mathit{state}) &= \mathit{state} \cup \{\langle \mathrm{a.target}, M(\mathrm{a.source}, \mathit{state})\rangle\} \\ &\quad \text{if } M(\mathrm{a.source}, \mathit{state}) \neq \mathit{undef} \\ &= \mathit{state} \\ &\quad \text{otherwise} \end{aligned}$$

8.20

8.21

8.22