1
Boolean and Boolean Expressions

As I presented in the two previous chapters, conditional expressions and conditional loops require expressions whose value are true or false. Such expressions are called *boolean* expressions and that until now I only presented it superficially. In this chapter I go deeper into the notion of boolean. as it is a key programming language concept. I show how to write basic boolean expressions, and how you can combine them to express complex conditions. Finally I present some of the most common errors that happen because of missing parentheses.

1 Booleans and Booleans Expressions

Boolean expressions are expressions that return true or false. Such values are called boolean[1]. Booleans can *only* be true or false. In programming languages booleans are important because they serve as basis for conditional execution.

Booleans. Booleans represent the truth or falsity of statements. For example, a true statement is *2 + 2 is equals 4* or *the earth accomplishes a complete rotation around its axis in 24 hours*. In Smalltalk, there are two objects to represent booleans: true and false. The object true represents the meaning "it is true" and the object false "it is false". The objects true and false understands all the key messages that allow you to use booleans as you shall see in a minute.

Boolean Expressions. A boolean expression is an expression that returns a boolean object. (2 > 1) is a boolean expression — it returns a boolean. You can think of a boolean expression as a question whose answer is true or false. Script 1.1 shows some examples of boolean expressions and the kinds of questions they express. Try to print the expression Time now > (Time new hours: 8) you get either true or false depending on the time you execute it.

[1]The adjective *boolean* comes from George Boole, an English mathematician of the nineteenth century. He discovered that boolean expressions –logical propositions – could be manipulated as mathematical objects.

Script 1.1 (*Examples of simple boolean expressions)*

```
Bot new color = Color red
```
Is the color of a newly created robot red?

```
Bot new center = (100@200)
```
Is a newly created robot located at the position
100 pixels to the right of the left edge and 153 pixels down from the top edge?

```
Time now > (Time new hours: 8)
```
Is the time now after 8 o'clock?

```
| pica |
pica := Bot new.
pica go: 100.
(Rectangle origin: 100@200 corner: 300@400)
   containsPoint: pica center
```
Is the center of the robot inside the rectangle 100@200, 300@400?

Notice that the first two questions could be answered from just the information in the boolean expression. The others require knowing what happens in the script before the boolean expression. Evaluate and print the results of the boolean expressions. After you try each example, experiment by changing the expression and checking your new prediction.

Simple boolean expressions are based on the messages = which returns whether two objects are equal, ∼= which returns whether two objects are different and some messages such as >, <=, <, >= which returns whether two objects are in certain order relations.

2 Combining Basic Boolean Expressions

The expressions presented in the previous sections are simple; and they are often not sufficient by themselves. However they can combined to express complicated conditions Complex boolean expressions can be composed from simpler ones using logical *negation* (not), *conjunction* (and), and *alternation* (or). Note that negation does not really combine boolean expressions but it is common to present it with conjunction and alternation.

In Smalltalk there are three messages that build compound boolean expressions from simpler ones: not for negation, **&** for conjunction and **|** for alternation (or). To compose complex expressions, you just send these messages to the simple boolean parts they are built from. The messages are used like this:

aBooleanExpression **not**
aBooleanExpression **&** *anotherBooleanExpression*
aBooleanExpression **|** *anotherBooleanExpression*

script 1.2 presents some examples of composed boolean expressions. I detail below the main way of composing boolean expressions that is by combining them using the messages not, **|** and **&**.

Script 1.2 (*Examples of composed boolean expressions)*

```
(Bot new color = Color red) not
```
Is the color of a newly created robot different than red?

```
| pica |
pica := Bot new.
(pica center = (100@100)) & (pica direction = 90)
```
Is a newly created robot pointing to the located at the position 100@100 and pointing to the north?

```
Time now > (Time new hours: 8) |  (Date today  weekday asString = 'Sunday')
```
Is the time now after 8 o'clock or are we Sunday?

```
Time now > (Time new hours: 8) |  (Date today  weekday asString = 'Sunday') not
```
Is the time now after 8 o'clock or are we not Sunday?

Negation (not)

Negation is useful to express the contrary of something. In Smalltalk it is expressed using the message not which simply negates the boolean expression to which it is sent. In the last line of script 1.3, the message not is sent to the expression (aBot color = Color red). If such an expression is true then its negation will be false and vice versa.

Script 1.3 (*Example of negation)*

```
| aBot |
aBot := Bot new.
aBot color: Color green.
(aBot  color = Color red) not
```
Is the color of a robot something else than red?

Note that you can always negate (logically reverse) a condition to switch from one form to the other. For example in method 1.1, distanceFromCenter >= 200 is the negation of the expression distanceFromCenter < 200 as shown in method 1.2. Again I suggest you add a trace to understand what is happening.

Method 1.1

```
redWhenCloseToCenter

  | distance |
  distance := self distanceFrom: World  center.
  distance >= 200
    ifFalse: [ self color: Color red ]
```

Method 1.2

```
redWhenCloseToCenter

  | distance |
  distance := self distanceFrom: World  center.
  distance < 200
    ifTrue: [ self color: Color red ]
```

Conjunction (and)

The term *conjunction* literally means together. Conjunction is used to express that the combination is true only when the two boolean sub-expressions are true. In Squeak, a conjunction is defined by sending the binary message & to a boolean expression with another boolean expression as argument. A conjunction is only true when both sub-expressions that composed it are true. In the script 1.4, the composed expression will only be true, if (aBot center = 100@100) *and* (aBot direction = 90) are true.

Script 1.4 (*Example of conjunction)*

```
(aBot center = 100@100) & (aBot direction = 90)
```
Is a robot located at the position 100@100 and pointing to the north?

Alternation (or)

Alternation is used to express the idea of choice. An alternation is defined by sending the binary message | to a boolean expression with another boolean expression as argument. An alternation is used to express that you want at least one of the boolean expressions to be true. Therefore a conjunction is true as long as one the expressions it is composed of is true.

In the script 1.5, the composed expression is true, as soon as one of the two expressions (aBot center = 100@100) or (aBot direction = 90) is true.

Script 1.5 (*Example of alternation)*

```
(aBot center = (100@100)) | (aBot direction = 90)
```
Is a robot located at the position 100@100 or heading at the north?

The last example of script 1.2 shows that you can combine boolean expressions multiple times, negate them, and group them by alternation (or) or conjunction (and) to represent complex conditions.

3 Some Smalltalk Points

In the same way that a particular robot is created by the class Bot, we say a robot is an instance of the class Bot, in Smalltalk, booleans are objects. true is an instance of the class True that defines the behavior of the boolean true. false is created by the class False that defines the behavior of the boolean false. Note that even if true and false are objects in the same sense that a robot was created by the class Bot, they are so central to Smalltalk that true and false are special variables. Hence, you do not have to create them using new, true and false exist and you do not have to worry about their creation. true and false start with a lowercase letter.

As I explained in the first section of this chapter, when you evaluate an expression such as Time now > (Time new hours: 8) you get a boolean object: true or false depending on the time you execute it. I said that to compose boolean expressions, the messages &, not, or | are sent to boolean expressions. Or the result of boolean expressions returns either true either false. Therefore, the message &, not, or | are methods defined on the classes of the objects true or false. The classes True and False defines the behavior such as and (&, not, |) that allows you to compose expressions.

The following table shows the most common boolean operations.

Kind	Message	Results
Negation	not	
Examples	false not	true
	true not	false
	(Bot new color = Color red) not	true
Conjunction (and)	&	
Examples	true & true	true
	false & true	false
	true & false	false
	false & false	false
	(aBot center = 100@100) & (aBot direction = 90)	true or false
Alternation (or)	|	
Examples	true | true	true
	false | true	true
	true | false	true
	false | false	false
	Time now > (Time new hours: 8) | (Date today weekday asString = 'Sunday')	true or false

4 Missing Parenthesis: A Frequent Mistake

It may happen that you get some trouble with the syntax of Smalltalk. Consider that everybody falls into that problem. Even experienced programmers do. The difference between a beginner and an experienced programmer is not that one does mistakes and the other don't. The main difference is that an experienced programmer goes much faster to identify and fix them.

Missing parenthesis is a frequent source of mistakes, therefore I show you how to analyze the errors you may get. Basically when you are composing a boolean expression you have to identify clearly to which expression the messages not, |, and & are sent to. Let us illustrate the problem.

A Case Study

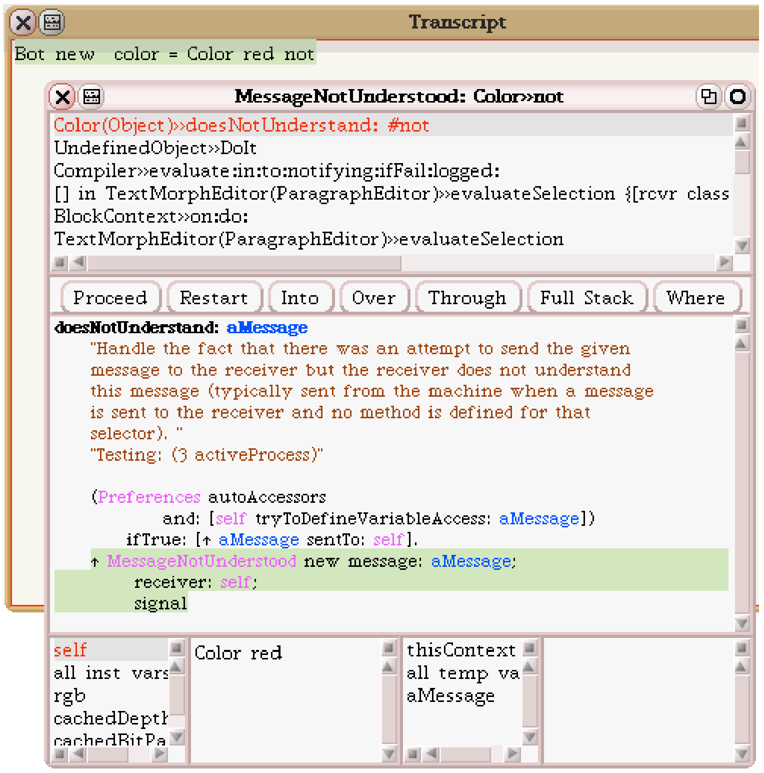
The script 1.6 shows a boolean expression that fails to represent the following question: is the color of a newly created robot different than red (not red)? Executing this script leads to an error. Execute the expression described in script 1.6, open the debugger on the error and select the first line in the top pane to obtain Figure 1.1.

Script 1.6 (*Missing Parenthesis to Identify the receiver of a not*)

```
Bot new  color = Color red not
```

Using the Debugger. The title window of the debugger already gives us some information. MessageNotUnderstood: Color»not. It states that a color object does not understand the message not.

Now when you select the topmost line of the top pane you see the body of the method doesNotUnderstand: which was called as the receiver did not understand the message not. When you click on self in the left bottom pane, you see that the receiver is not a boolean as it should be but a color



```
Bot new  color = Color red not
```

```
Color(Object)»doesNotUnderstand: #not
UndefinedObject»DoIt
Compiler»evaluate:in:to:notifying:ifFail:logged:
[] in TextMorphEditor(ParagraphEditor)»evaluateSelection {[rcvr class
BlockContext»on:do:
TextMorphEditor(ParagraphEditor)»evaluateSelection
```

```
doesNotUnderstand: aMessage
    "Handle the fact that there was an attempt to send the given
    message to the receiver but the receiver does not understand
    this message (typically sent from the machine when a message
    is sent to the receiver and no method is defined for that
    selector). "
    "Testing: (3 activeProcess)"

    (Preferences autoAccessors
            and: [self tryToDefineVariableAccess: aMessage])
        ifTrue: [↑ aMessage sentTo: self].
    ↑ MessageNotUnderstood new message: aMessage;
        receiver: self;
        signal
```

```
self                Color red        thisContext
all inst vars                        all temp va
rgb                                  aMessage
cachedDeptl
cachedBitPa
```

Figure 1.1: The message not is not sent to the complete boolean expression Bot new color = Color red, but sent to the expression Color red which returns a color. Therefore it is not understood.

Color red! If you click on the aMessage on the right bottom pane, you will see which message was not understood. In our case, you see not which means that the message not has not be sent to the right receiver. The message not is sent to the result of the expression Color red which is a color and does not understand the message not.

Understanding the Problem. The reason why the message not is sent to the expression Color red and not to the complete boolean expression is related to the way Smalltalk executes expressions as explained in Chapter **??**. First the expressions surrounded by parentheses are executed, then the unary messages, then the binary and finally the keyword-based messages. In our case the message not is an unary message. Therefore it is evaluated before the binary message = and it is sent to the result of the expression Color red. To get the correct execution order, you have to surround the expression in parentheses as shown in the script 1.3, this way the message not will be sent to the result of the = message.

The messages are executed in the wrong expression as follows: The expression Bot new color = Color red **not** is executed as it would be written fully parenthesized as follow (((Bot new) color) = ((Color red) not)). Therefore first both parts of the binary method = are evaluated that is the expression ((Bot new) color) which returns aColor and the expression ((Color red) not). The execution of the expression ((Color red) not) evaluates first Color red which returns a color, then the message not is sent to it which leads to an error.

To get the expected behavior, the expression should be parenthesised so that the not message is sent to the result of the = message. The expression is then **(**Bot new color = Color red**)** **not**. This expression is executed as follow: first both parts of the binary method = are evaluated. Both return a color possibly equal. Then the = message is executed, that is send to the color result of the right hand expression. The execution of the message = returns a boolean to which the message not is sent.

When you are not sure about the order of messages, I suggest you use the fact that expressions in parenthesis are executed before the other ones. Therefore you can always put parenthesis around the expressions to make sure that the messages are executed the way you want.

Similar Problems and Solutions

It would be tedious to explain the similar problems you may encounter with the other messages such as & and |. Try to execute the scripts 1.7 and 1.9, and to understand the problems. I show you the corresponding correctly parenthesized scripts 1.8 and 1.10.

Script 1.7 (*Missing Parenthesis to identify the receiver of a &.)*

```
| aBot |
aBot := Bot new.
aBot center = 100@100 & aBot penSize = 5
```

Script 1.8 (*Identifying the receiver of a & using parenthesis.)*

```
| aBot |
aBot := Bot new.
(aBot center = 100@100) & (aBot penSize = 5)
```

Script 1.9 (*Missing Parenthesis to identify the receiver of |.)*

```
| aBot |
aBot := Bot new.
aBot center = 100@100 | aBot direction = 90
```

Script 1.10 (*Identifying the receiver of | using parenthesis.)*

```
| aBot |
aBot := Bot new.
(aBot center = 100@100) | (aBot direction = 90)
```

Summary

- Booleans are true or false. true represents a true fact and false a false one.
- Booleans expressions are expressions that manipulate booleans and that return booleans.
- Complex boolean expressions can be composed of simple boolean expressions using conjunction (and), alternation (or), and negation (not).

Kind	Message	Results
Negation	not	
Examples	false not	true
	true not	false
	(aBot color = Color red) not	true or false
Conjunction (and)	&	
Examples	true & true	true
	false & true	false
	true & false	false
	false & false	false
	(aBot center = 100@100) & (aBot direction = 90)	true or false
Alternation (or)	|	
Examples	true | true	true
	false | true	true
	true | false	true
	false | false	false
	Time now > (Time new hours: 8) | (Date today weekday asString = ’Sunday’)	true or false