16 Constructing a Pure Interpreter Level 2

Introduction

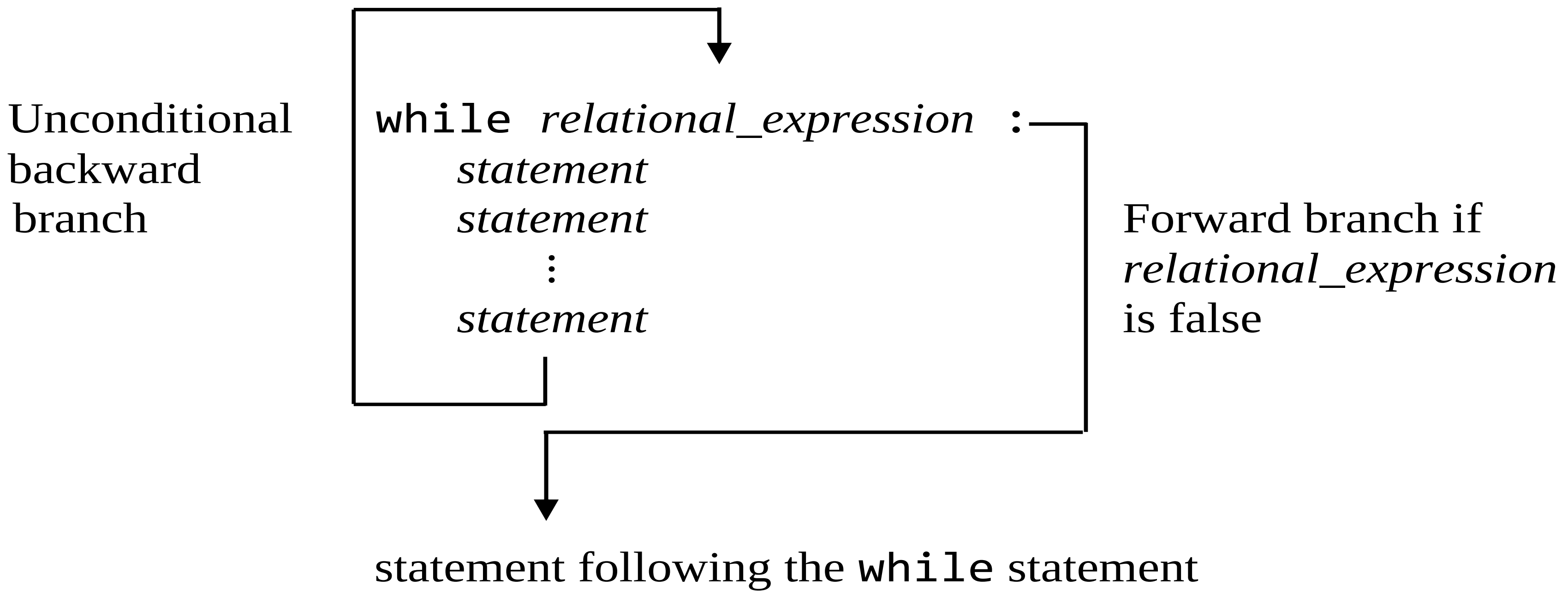

Unconditional backward branch
while *relational_expression* :
statement
statement
⋮
statement
Forward branch if *relational_expression* is false
statement following the while statement

Figure 16.1

Determining the Branch-to Address in a Backward Branch

```
def whilestmt():
   global tokenindex, token
   advance()                         # advance past while keyword
   savetokenindex = tokenindex       # save address of exit-test expr
   while True:                       # parser while loop
      relexpr()                      # pushes value of exit-test expr
      consume(COLON)
      if operandstack.pop():         # is exit-test relexpr true?
         codeblock()                 # execute loop body
         tokenindex = savetokenindex # backward branch rel expr
         token = tokenlist[tokenindex]
      else:
         break                       # must now do forward branch
```

Figure 16.2

Implementing a Forward Branch

```
consume(NEWLINE)
indentcol = token.column  # save column of INDENT token
consume(INDENT)
while True:
   # check if dedent is to left of indent column
   if token.category == DEDENT and token.column < indentcol:
      advance()           # advance past dedent token
      break               # now past the end of while loop
   advance()              # still in body of while loop so advance
```

Figure 16.3

Branching in an `if-else` statement:

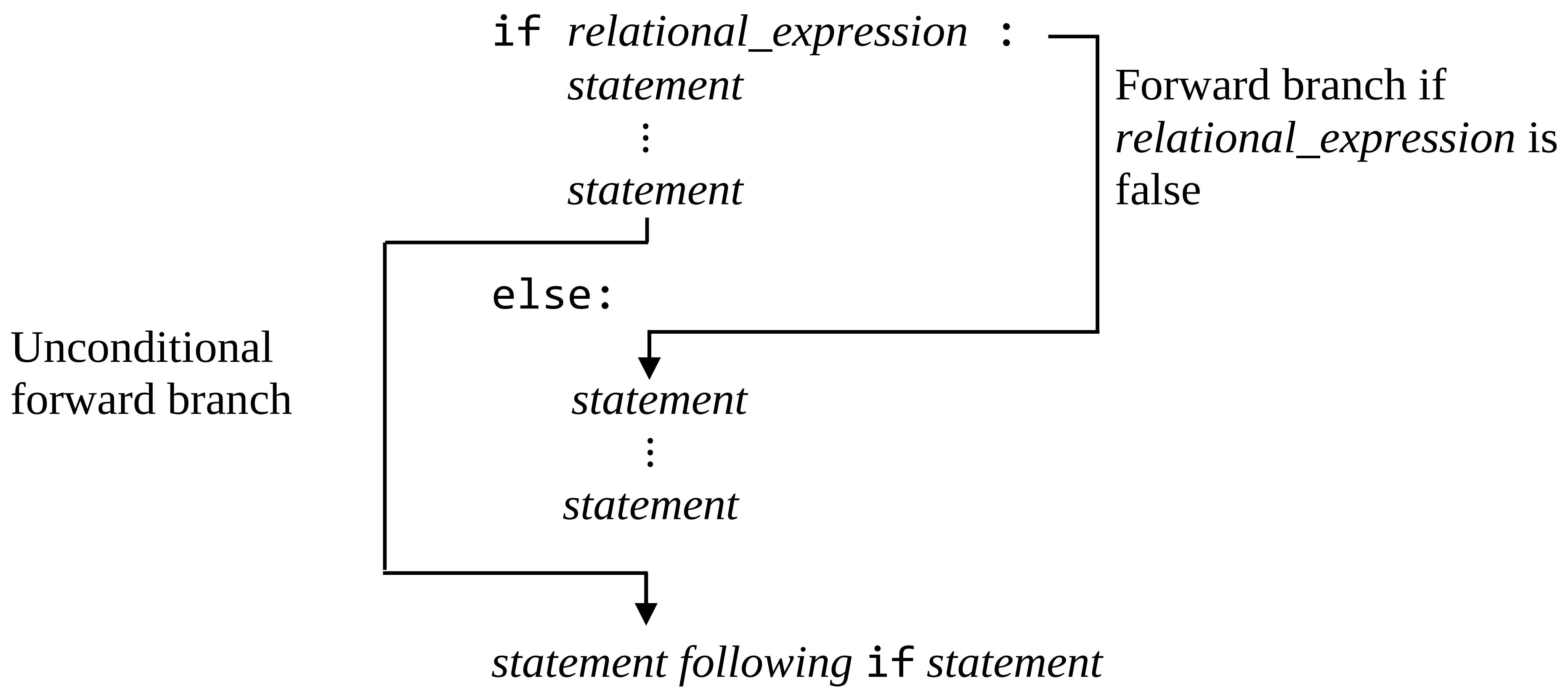
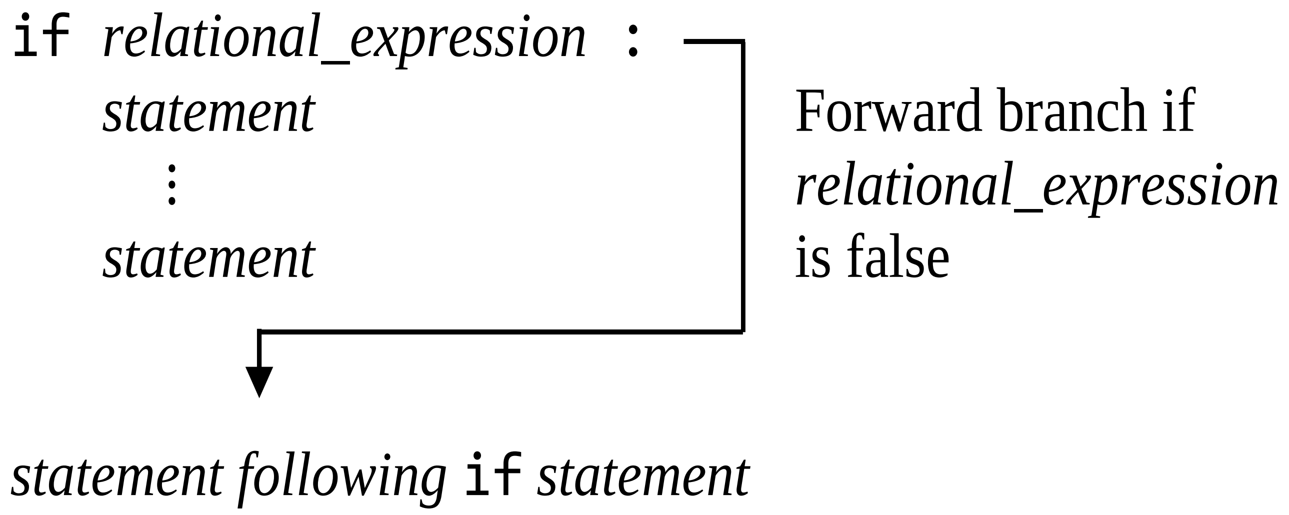

if *relational_expression* :
statement
⋮
statement
else:
statement
⋮
statement
statement following if *statement*
Forward branch if *relational_expression* is false
Unconditional forward branch

Figure 16.4

Branching in an `if` statement without an `else`:



```
if relational_expression :
    statement
        ⋮
    statement

statement following if statement
```

Figure 16.5

```
 1 def ifstmt():
 2    advance()
 3    relexpr()
 4    consume(COLON)
 5    saveval = operandstack.pop() # save it for line 15
 6    if saveval:
 7       codeblock()            # do codeblock() if saveval true
 8    else:                     # do forward branch if saveval false
 9       # code from Fig. 16.3
10       ...
11    if token.category == ELSE:
12       advance()
13       consume(COLON)
14       if not saveval:
15          codeblock()
16       else:                  # do another forward branch
17          # code from Fig. 16.3
18    …
```

Executing a Relational Expression

```
<relexpr> → <expr> [ ('==' | '!=' | '<' | '<=' | '>' | '>=') <expr> ]
```

```
def relexpr():
   expr()
   if token.category in [EQUAL, NOTEQUAL, LESSTHAN, LESSEQUAL,
                         GREATERTHAN, GREATEREQUAL]:
      savecat = token.category
      advance()
      expr()
      right = operandstack.pop()
      left = operandstack.pop()
      if savecat == EQUAL:
         operandstack.append(left == right) # push True or False
      elif savecat == NOTEQUAL:
         operandstack.append(left != right) # push True or False
      ... <============================== missing instructions
```

Figure 16.6