necessarily integral. Furthermore, the final solution to a linear program need not be integral; it is purely coincidental that this example has an integral solution.

Pivoting

We now formalize the procedure for pivoting. The procedure PIVOT takes as input a slack form, given by the tuple (N, B, A, b, c, v), the index l of the leaving variable x_l, and the index e of the entering variable x_e. It returns the tuple $(\hat{N}, \hat{B}, \hat{A}, \hat{b}, \hat{c}, \hat{v})$ describing the new slack form. (Recall again that the entries of the $m \times n$ matrices A and $\hat{A}$ are actually the negatives of the coefficients that appear in the slack form.)

```
PIVOT(N, B, A, b, c, v, l, e)
 1  // Compute the coefficients of the equation for new basic variable x_e.
 2  let Â be a new m × n matrix
 3  b̂_e = b_l / a_le
 4  for each j ∈ N − {e}
 5      â_ej = a_lj / a_le
 6  â_el = 1 / a_le
 7  // Compute the coefficients of the remaining constraints.
 8  for each i ∈ B − {l}
 9      b̂_i = b_i − a_ie b̂_e
10      for each j ∈ N − {e}
11          â_ij = a_ij − a_ie â_ej
12      â_il = −a_ie â_el
13  // Compute the objective function.
14  v̂ = v + c_e b̂_e
15  for each j ∈ N − {e}
16      ĉ_j = c_j − c_e â_ej
17  ĉ_l = −c_e â_el
18  // Compute new sets of basic and nonbasic variables.
19  N̂ = N − {e} ∪ {l}
20  B̂ = B − {l} ∪ {e}
21  return (N̂, B̂, Â, b̂, ĉ, v̂)
```

PIVOT works as follows. Lines 3–6 compute the coefficients in the new equation for x_e by rewriting the equation that has x_l on the left-hand side to instead have x_e on the left-hand side. Lines 8–12 update the remaining equations by substituting the right-hand side of this new equation for each occurrence of x_e. Lines 14–17 do the same substitution for the objective function, and lines 19 and 20 update the

In Section 29.5, we shall show how to determine whether a problem is feasible, and if so, how to find a slack form in which the initial basic solution is feasible. Therefore, let us assume that we have a procedure INITIALIZE-SIMPLEX(A, b, c) that takes as input a linear program in standard form, that is, an $m \times n$ matrix $A = (a_{ij})$, an m-vector $b = (b_i)$, and an n-vector $c = (c_j)$. If the problem is infeasible, the procedure returns a message that the program is infeasible and then terminates. Otherwise, the procedure returns a slack form for which the initial basic solution is feasible.

The procedure SIMPLEX takes as input a linear program in standard form, as just described. It returns an n-vector $\bar{x} = (\bar{x}_j)$ that is an optimal solution to the linear program described in (29.19)–(29.21).

```
SIMPLEX(A, b, c)
 1  (N, B, A, b, c, v) = INITIALIZE-SIMPLEX(A, b, c)
 2  let Δ be a new vector of length n
 3  while some index j ∈ N has c_j > 0
 4      choose an index e ∈ N for which c_e > 0
 5      for each index i ∈ B
 6          if a_ie > 0
 7              Δ_i = b_i / a_ie
 8          else Δ_i = ∞
 9      choose an index l ∈ B that minimizes Δ_i
10      if Δ_l == ∞
11          return "unbounded"
12      else (N, B, A, b, c, v) = PIVOT(N, B, A, b, c, v, l, e)
13  for i = 1 to n
14      if i ∈ B
15          x̄_i = b_i
16      else x̄_i = 0
17  return (x̄_1, x̄_2, ..., x̄_n)
```

The SIMPLEX procedure works as follows. In line 1, it calls the procedure INITIALIZE-SIMPLEX(A, b, c), described above, which either determines that the linear program is infeasible or returns a slack form for which the basic solution is feasible. The **while** loop of lines 3–12 forms the main part of the algorithm. If all coefficients in the objective function are negative, then the **while** loop terminates. Otherwise, line 4 selects a variable x_e, whose coefficient in the objective function is positive, as the entering variable. Although we may choose any such variable as the entering variable, we assume that we use some prespecified deterministic rule. Next, lines 5–9 check each constraint and pick the one that most severely limits the amount by which we can increase x_e without violating any of the nonnegativ-

$$\text{maximize} \quad -x_0 \tag{29.106}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j - x_0 \le b_i \quad \text{for } i = 1, 2, \ldots, m , \tag{29.107}$$

$$x_j \ge 0 \quad \text{for } j = 0, 1, \ldots, n . \tag{29.108}$$

Then L is feasible if and only if the optimal objective value of L_{aux} is 0.

Proof Suppose that L has a feasible solution $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$. Then the solution $\bar{x}_0 = 0$ combined with $\bar{x}$ is a feasible solution to L_{aux} with objective value 0. Since $x_0 \ge 0$ is a constraint of L_{aux} and the objective function is to maximize $-x_0$, this solution must be optimal for L_{aux}.

Conversely, suppose that the optimal objective value of L_{aux} is 0. Then $\bar{x}_0 = 0$, and the remaining solution values of $\bar{x}$ satisfy the constraints of L. ■

We now describe our strategy to find an initial basic feasible solution for a linear program L in standard form:

INITIALIZE-SIMPLEX(A, b, c)

```
1  let k be the index of the minimum b_i
2  if b_k ≥ 0                          // is the initial basic solution feasible?
3      return ({1, 2, ..., n}, {n + 1, n + 2, ..., n + m}, A, b, c, 0)
4  form L_aux by adding -x_0 to the left-hand side of each constraint
       and setting the objective function to -x_0
5  let (N, B, A, b, c, v) be the resulting slack form for L_aux
6  l = n + k
7  // L_aux has n + 1 nonbasic variables and m basic variables.
8  (N, B, A, b, c, v) = PIVOT(N, B, A, b, c, v, l, 0)
9  // The basic solution is now feasible for L_aux.
10 iterate the while loop of lines 3–12 of SIMPLEX until an optimal solution
       to L_aux is found
11 if the optimal solution to L_aux sets x̄_0 to 0
12     if x̄_0 is basic
13         perform one (degenerate) pivot to make it nonbasic
14     from the final slack form of L_aux, remove x_0 from the constraints and
           restore the original objective function of L, but replace each basic
           variable in this objective function by the right-hand side of its
           associated constraint
15     return the modified final slack form
16 else return "infeasible"
```