10.2. Message Passing: Sum Product

Algorithm 10.A.1 — Efficient implementation of a factor product operation.

```plaintext
Procedure Factor-Product (phi1 over scope X1, phi2 over scope X2)
    // Factors represented as a flat array with strides for the variables
    j ← 0, k ← 0
    for l = 0,...,|X1 ∪ X2|
        assignment[l] ← 0
     for i = 0,...,|Val(X1 ∪ X2)| − 1
         psi[i] ← phi1[j] · phi2[k]
     for l = 0,...,|X1 ∪ X2|
        assignment[l] ← assignment[l] + 1
     if assignment[l] = card[l] then
        assignment[l] ← 0
        j ← j − (card[l] − 1) · phi1.stride[l]
        k ← k − (card[l] − 1) · phi2.stride[l]
     else
        j ← j + phi1.stride[l]
        k ← k + phi2.stride[l]
     break
    return (psi)
```

and vice versa

assignment[i] = ⌊index/phi.stride[i]⌋ mod card[i]

With this factor representation, we can now design a library of operations: product, marginalization, maximization, reduction, and so forth. Since many inference algorithms involve multiple iterations over a series of factor operations, it is important that these be high-performance. One of the key design decisions is indexing the appropriate entries in each factor for the operations that we wish to perform. (In fact, when one uses a naive implementation of index computations, one often discovers that 90 percent of the running time is spent on that task.)

Algorithm 10.A.1 provides an example for the product between two arbitrary factors. Here we define phi.stride[X] = 0 if X ∉ Scope[phi]. The inner loop (over l) advances to the next assignment to the variables in ψ and calculates indexes into each other factor array on the fly. It can be understood by considering the equation for computing index shown earlier. Similar on-the-fly index calculations can be applied for other factor operations. We leave these as an exercise (exercise 10.3).

For iterative algorithms or multiple queries, where the same operation (on different data) is performed a large number of times, it may be beneficial to cache these index mappings for later use. Note, however, that the index mappings require the same amount of storage as the factors themselves, that is, are exponential in the number of variables. Thus, this design choice offers a