L'implantation de fonctions pour les ensembles en APL (une proposition)

Joseph de Kerf

Abstract :
Proposal are formulated for the implementation of the primitive set functions : unique, union, intersection and difference. User-defined functions are given, simulating those proposed primitive functions illustrated with some examples.

Dans une note ajoutée à une lettre de l'éditeur publiée dans « Les Nouvelles d'APL » [Réf.1], la rédaction insiste sur le besoin d'étendre l'APL avec des fonctions primitives pour la manipulation des ensembles. Dans cet article, nous proposons des définitions pour de telles fonctions : la fonction monadique unique et les fonctions dyadiques union, intersection et différence. Nous donnons les fonctions définies équivalentes aux fonctions primitives proposées illustrées de quelques exemples. Les concepts sont fondés sur ceux décrits par D. Livingstone (et les autres auteurs mis en référence) dans sa communication à APL86 [Réf.2]. Le nombre d'occurrences d'entités dans les résultats sont adaptés aux besoins imposés par la manipulation de l'algèbre des bases de données relationnelles.

Unique : `∪←ZR`

L'argument R peut être n'importe quel tableau, simple ou enclos, homogène ou hétérogène. Le résultat Z est un vecteur contenant une copie de chaque entité unique dans R. L'ordre des entités dans Z est l'ordre de la première occurrence dans la linéarisation de R. Cette fonction utilise la tolérance de comparaison.

Union : `Z←L∪R`

Les arguments L et R peuvent être des tableaux quelconques, simples ou enclos, homogènes ou hétérogènes. Le résultat Z est un vecteur contenant toutes les entités de L et de R. Le nombre d'occurrences des entités dans Z est égal au plus grand nombre d'occurrences de ces entités dans L et R respectivement. L'ordre des entités dans Z est l'ordre de leur première occurrence dans la concaténation de la linéarisation de L avec la linéarisation de R. Cette fonction utilise la tolérance de comparaison.

Intersection : `Z←L∩R`

Les arguments L et R peuvent être des tableaux quelconques, simples ou enclos, homogènes ou hétérogènes. Le résultat Z est un vecteur contenant toutes les entités de L et de R. Le nombre d'occurrences des entités dans Z est égal au plus petit nombre d'occurrences de ces entités dans L et R respectivement. L'ordre des entités dans Z est l'ordre de leur première occurrence dans la concaténation de la linéarisation de L avec la linéarisation de R. Cette fonction utilise la tolérance de comparaison.

Différence : `Z←L~R`

Les arguments L et R peuvent être des tableaux quelconques, simples ou enclos, homogènes ou hétérogènes. Le résultat Z est un vecteur contenant toutes les entités de L qui numériquement ne sont pas contenues dans R. Soit x le nombre d'occurrences d'une telle entité dans L et y le nombre

d'occurrences d'une telle entité dans R. Si x>y, l'entité est reproduite x-y fois dans Z. L'ordre des entités dans Z est l'ordre de leur première occurrence dans la concaténation de la linéarisation de L avec la linéarisation de R. Cette fonction utilise la tolérance de comparaison.

Fonctions définies

Nous donnons ci-dessous les fonctions définies qui se substituent aux primitives proposées. Ces fonctions ne se conforment pas à l'APL ISO standard, car ce standard ne comprend ni les tableaux enclos ni les tableaux hétérogènes. De plus le standard ne connaît que la compression et pas la réplication [Réf.3]. Ces fonctions sont conformes cependant au standard proposé dans l'APL ISO étendu qui admet les tableaux enclos et les tableaux hétérogènes ainsi que l'extension de la compression à la réplication [Réf.4]. L'APL étendu ne reconnaît pas les entiers négatifs comme argument gauche de la réplication mais c'est sans importance puisqu'on n'utilise pas de tels entiers négatifs dans les fonctions que l'on montre.
La programmation a été réalisée sur un MicroLine Pentium-S 100, avec Dyalog APL/W Version 7.1.2 sous WINDOWS 3.11 [Réf.5]

```
    ∇ Z←UNIQUE R
[1]   Z←((R⍳R)=⍳⍴R)/R←,R
   ∇

    ∇ Z←L UNION R
[1]   Z←((Z⍳Z)=⍳⍴Z)/Z←(L←,L),R←,R
[2]   Z←((+/Z∘.=L)⌈+/Z∘.=R)/Z
   ∇

    ∇ Z←L INTERSECTION R
[1]   Z←((Z⍳Z)=⍳⍴Z)/Z←(L←,L),R←,R
[2]   Z←((+/Z∘.=L)⌊+/Z∘.=R)/Z
   ∇

    ∇ Z←L DIFFERENCE R
[1]   Z←((Z⍳Z)=⍳⍴Z)/Z←(L←,L),R←,R
[2]   Z←((D>0)×D←(+/Z∘.=L)-+/Z∘.=R)/Z
   ∇
```

Exemples

```
      UNIQUE 'ABRACADABRA'
ABRCD

      UNIQUE 'YABADABADOE'
YABDOE

     'ABRACADABRA'UNION 'YABADABADOE'
AAAAABBRRCDDYOE

     'YABADABADOE'UNION 'ABRACADABRA'
YAAAAABBDDOERRC

     'ABRACADABRA'INTERSECTION 'YABADABADOE'
AAAABBD
```

```
      'YABADABADOE'INTERSECTION 'ABRACADABRA'
AAAABBD

      'ABRACADABRA'DIFFERENCE 'YABADABADOE'
ARRC

      'YABADABADOE'DIFFERENCE 'ABRACADABRA'
YDOE
```

Quelques commentaires

L'APL ISO n'inclut aucune des fonctions primitives proposées. L'APL étendu n'inclut que la fonction primitive *unique* (Z←∪R). Cependant le domaine de l'argument R est restreint aux scalaires et aux vecteurs. Dyalog APL et VAX APL inclut cette fonctionnalité, mais sur VAX APL le domaine de l'argument R est étendu à des tableaux quelconques.
Seuls Dyalog APL et VAX APL comprennent la fonction primitive *union* (Z←L∪R). Dans Dyalog APL, le domaine des arguments L et R est restreint aux scalaires et aux vecteurs, alors que dans VAX APL les arguments L et R peuvent être des tableaux quelconques. Dans Dyalog APL, l'union rend un vecteur Z formé par la concaténation de L avec tous les items de R non trouvés dans L. Les entités dupliquées ne sont pas enlevées. Dans VAX APL, l'union rend un vecteurs Z formé de la concaténation des linéarisations de L et R dont on enlève les entités dupliquées.
Seuls Dyalog APL et VAX APL comprennent la fonction primitive *intersection* (Z←L∩R). Dans Dyalog APL, le domaine des arguments L et R est restreint aux scalaires et aux vecteurs, alors que dans VAX APL les arguments L et R peuvent être des tableaux quelconques. Dyalog APL, l'intersection rend un vecteur Z toutes les entités de L trouvées aussi dans R. Les entités dupliquées ne sont pas enlevées. Dans VAX APL, l'intersection rend un vecteurs Z formé des entités communes à L et à R dont on enlève les entités dupliquées. L'ordre des entités dans Z est imprédictible.
Aucune des implantations actuelles d'APL n'offre la primitive proposée pour la *différence* (Z←L~R). Bien sûr, l'introduction du nouveau caractère « tilde souligné » est ouvert à la discussion. Une meilleure proposition serait la bienvenue. On peut éviter l'usage d'un nouveau caractère en substituant la fonction primitive proposée à l'actuelle fonction primitive *sans* ou *excluant* (Z←L~R). L'idiome pour évaluer *sans* ou *excluant* est très simple :

```
    ∇ Z←L SANS R
[1]   Z←(~L∈R)/L←,L
     ∇

       'ABRACADABRA' SANS 'YABADABADOE'
RCR
       'YABADABADOE' SANS 'ABRACADABRA'
YOE
```

Cependant, cette idée peut-être inacceptable, puisque la fonction primitive *sans* ou *excluant* (Z←L~R) est une fonctionnalité requise dans l'APL ISO et dans l'APL étendu et qu'elle est offerte par toutes les implantations actuelles d'APL.

Les tableaux vides comme arguments

Les définitions des fonctions primitives proposées sont seulement des descriptions informelles - on ne donne aucune séquence d 'évaluation. La question reste de savoir quel résultat doit avoir Z quand les

arguments L et R des fonctions dyadiques proposées sont des tableaux vides de types différents, par exemple le vide numérique (⍳0) et le vide caractère ('') respectivement ou vice-versa.
En utilisant les fonctions définies ci-dessus, présentées comme des substitues aux fonctions primitives proposées, le résultat Z dépend du comportement de l'implantation de la concaténation de la linéarisation des arguments L et R. En APL ISO, quand les arguments de la concaténation sont des tableaux vides de types différents une « erreur de domaine » doit être signalée. En APL étendu cependant, un vecteur vide pourrait être produit dont le type est déterminé par l'implantation de « l'algorithme pour les éléments typiques mixtes ». On donne ci-dessous [Réf.6] le comportement de quelques implantations courantes sur micro-ordinateurs :

Implantation	(⍳0),''	'',⍳0
VAX APL (DEC)	⍳0	''
Dyalog APL/W	⍳0	''
APL2/PC (IBM)	''	⍳0
APL*PLUS III	''	⍳0
APL 68000 II	''	⍳0

VAX APL et Dyalog APL/W rendent l'argument gauche. APL2/PC, APL*PLUS III et APL.68000 II rendent l'argument droit. On pourrait souhaiter que toutes les implantations donnent le même résultat. D'un autre coté, on pourrait affirmer que le type du résultat est sans importance puisque le résultat est un vecteur vide.

Références

[1] J. de Kerf ; Courrier des lecteurs ; Les Nouvelles d'APL, N° 20, septembre 1996, pp. 100-101.
[2] D. Livingstone ; APL86 Conférence Proceedings, Manchester, England, 7-11 July 1986 ; APL Quote-Quad, Vol. 16, N° 4, & special Edition of VECTOR, 1986, pp. 211-220
[3] International Standard ISO 8485 : 1989 (E), First Edition ; 1989-11-01 ; Programming Languages - APL ; International Organisation for Standardization, Geneva, Switzerland, 1989.
[4] ISO Document CD13751 ; Programming Language APL - Extended ; Committee Draft prepared by the APL Working Group ISO-IEC/JTC/SC22/WG3 - Version 1 ; International Organization for Standardization, Geneva, Switzerland, August 1993.
[5] Dyalog APL/W Language Reference - Version 7 ; Dyadic Systems Limited, Basingstoke, Hampshire, United Kingdom, 1994.
[6] J. de Kerf ; A comparison of Microcomputer APLs - A Preliminary Report (Part 2 : Primitive Functions and Operators) ; APL-CAM Journal, Vol. 18, N°.2, 16 june 1996, pp. 202-261.