5. DES PREDICATS ET DE LA SELECTION

Avec les fonctions dont nous disposons, il est déjà possible d'écrire un grand nombre de petits programmes utiles. Reste toutefois que tous ces programmes sont *linéaires*. Par cela, nous entendons qu'ils se constituent de suites d'instructions (de suites d'appels de fonctions) qui seront exécutées l'une après l'autre, sans possibilité de faire des tests ou des répétitions. Dans ce chapitre des fonctions particulières seront introduites, les *prédicats*, et une fonction de *sélection* ou de *test* qui se sert de ces prédicats.

Rappelons que des programmes peuvent être considérés comme des descriptions d'activités, comme des recettes de cuisine. Naturellement, comme dans les recettes de cuisine, il doit y avoir des possibilités de tester l'état du monde, par exemple il faut une possibilité de tester si l'eau est bouillante ou pas. Des fonctions de test sont, en logique, appelées des *prédicats*. Ce sont des fonctions posant des questions auquelles la réponse est soit oui, soit non. Elles testent la vérification d'une condition, et peuvent donc ramener *deux* valeurs possibles : *oui* ou *non*, comme à la question "est-ce que l'eau est bouillante ?" vous ne pouvez répondre que par oui ou par non (des réponses du style "pas encore" ne sont pas admises dans notre jeu), la machine ne peut, à l'évaluation d'un prédicat, que répondre par oui ou par non, *vrai* ou *faux*.

Non se dit en LISP **NIL** ou **()**, et *oui* se dit **T**. **T** est une abréviation de **T**rue, mot anglais pour *vrai*. **Nil** signifie en Latin et en 'colloquial english' *rien*, faux, nul. Cette association entre le mot 'faux' et le mot 'rien' se retrouve en LISP : rappelons que **NIL** est également un mot pour désigner la liste vide **()**. **NIL** a donc en LISP plusieurs rôles :
• c'est un atome. En LE_LISP, **NIL** est un atome, mais pas une variable : on ne peut pas lui associer une valeur.
• c'est un nom pour la liste vide (la liste à 0 élément)
• et c'est le mot LISP pour *logiquement faux*.

5.1. QUELQUES PREDICATS

Voici la définition syntaxique d'un premier prédicat - la fonction **ATOM** :

(ATOM *arg***)**
→ **()** si *arg* n'est pas un atome
→ **T** si l'argument est un atome

Cette fonction teste donc si la valeur de son argument est un *atome*. Voici quelques exemples :

(ATOM 'BRAVO) → **T**
(ATOM NIL) → **T**
remarquez qu'il ne faut pas *quoter* l'atome **NIL**. De même, il ne faut pas quoter l'atome **T** et les nombres. On dit que de tels atomes sont des *constantes*
(ATOM ()) → **T** *; par définition !!!! ;*
(ATOM '(A B)) → **()** *; c'est une liste ! ;*
(ATOM 1024) → **T**
(ATOM '(GIRLS (ARE (COSIER)))) → **()**

On a un prédicat qui teste si son argument est un atome, mais naturellement il existe également des prédicats testant les autres types d'objets LISP. Ainsi le prédicat **NUMBERP** teste si son argument est un *nombre* et le prédicat **CONSP**[1] teste si son argument est une *liste*. Voici leurs définitions syntaxiques et quelques exemples d'appels :

(CONSP *arg***)** → **NIL** si *arg* n'est pas une liste
→ **T** si l'argument est une liste

quelques exemples de **CONSP** :

(CONSP '(A B C)) → **T**
(CONSP 'BRAVO) → **NIL**
(CONSP ()) → **NIL** *; !!!! par définition !!!! ;*
(CONSP '((A)(B C))) → **T**
(CONSP -1024) → **NIL**

et voici le prédicat **NUMBERP** :

(NUMBERP *arg***)** → **NIL** si *arg* n'est pas un nombre
→ *arg* si l'argument est un nombre

Une première remarque s'impose : tout à l'heure il était signalé qu'en LISP *vrai* se dit **T**. Ce n'est pas tout à fait correct, il aurait fallu dire que *faux* se disait **NIL** en LISP, mais que LISP considère toute expression différente de **NIL** comme équivalente à **T**, c'est-à-dire *vrai*. C'est pourquoi dans la fonction **NUMBERP** le résultat peut être soit **NIL**, si l'argument n'est pas un nombre, soit, si l'argument est effectivement un nombre, la valeur de l'argument même. Tout les prédicats qui permettent de ramener comme valeur *vrai* l'argument de l'appel même, vont adhérer à cette convention qui est fort utile si l'on veut utiliser, dans la suite du programme, la valeur de l'argument. Nous y reviendrons à l'occassion. D'ailleurs, cette convention n'est pas possible pour la fonction **ATOM**, puisque l'appel

(ATOM NIL)

doit ramener en valeur *vrai*, c-à-d. **T**. Elle ne peut décemment pas, dans ce cas, ramener la valeur de l'argument, qui est **NIL**, donc la valeur LISP disant 'logiquement faux'.

Voici quelques exemples d'appel de la fonction **NUMBERP** :

[1] En VLISP le prédicat **CONSP** s'appelle **LISTP** et le prédicat **NUMBERP** s'appelle **NUMBP**.

(NUMBERP 0)	→	**0**
(NUMBERP -110)	→	**-110**
(NUMBERP NIL)	→	**()**
(NUMBERP 'HMMM)	→	**()**
(NUMBERP 3214)	→	**3214**
(NUMBERP '(DO RE))	→	**()**

Donnons tout de suite un prédicat supplémentaire : la fonction **NULL**. Cette fonction teste si son argument est une liste vide ou non. Voici sa définition :

(NULL *arg***)**
→ **NIL** si *arg* n'est pas la liste vide
→ **T** si l'argument est la liste vide

NULL peut être traduite par la question 'est-il vrai que la valeur de l'argument est égale à **NIL** ou **()** ?". Voici quelques exemples :

(NULL ())	→	**T**
(NULL NIL)	→	**T**
(NULL T)	→	**()**
(NULL '(TIENS))	→	**()**
(NULL 13)	→	**()**
(NULL 'UN-ATOME)	→	**()**
(NULL (NULL NIL))	→	**()**
(NULL (NULL T))	→	**T**

Evidemment, puisque **NIL** est l'équivalent LISP du *faux* logique, ce prédicat peut être également traduit en "est-il vrai que la valeur de l'argument est faux ?" ! C'est une fonction extrèmement utile, comme nous le verrons dans le chapitre suivant.

Ces prédicats peuvent être utilisés dans des fonctions comme toute autre fonction LISP. Ecrivons par exemple une fonction qui teste si le **CAR** (abus de langage ! **CAR** veut dire : le premier élément) d'une liste est un atome :

```
(DE ATOM-CAR? (L)
    (ATOM (CAR L)))
```

Si nous donnons cette définition de fonction à la machine LISP, elle nous répondra en retournant l'atome **ATOM-CAR?**, indiquant ainsi que cette fonction lui est connue maintenant. On peut donc l'appeler ensuite :

(ATOM-CAR? '((DO RE)(RE DO)))	→	**NIL**
ou		
(ATOM-CAR? '(DO RE RE DO))	→	**T**
ou encore		
(ATOM-CAR? (CONS 'HELLO '(YOU THERE)))	→	**T**

5.1.1. exercices

1. Ecrivez une fonction **NUMBERP-CADR**, qui teste si le deuxième élément de la liste passée en argument est un nombre.
2. Ecrivez une fonction de nom **LISTE-CAR?** qui ramène la liste **(EST** x**)**, avec x égal soit à **T** soit à **NIL**, suivant que le premier élément de sa liste argument est une liste ou non. Exemples d'appels :

(LISTE-CAR? '(A B C)) → **(EST NIL)**
(LISTE-CAR? '((A) (B C))) → **(EST T)**
(LISTE-CAR? '(AHA)) → **(EST NIL)**

3. Quels sont les résultats de ces appels de fonctions :

(NUMBERP-CADR '(1 2 3)) → ?
(NUMBERP-CADR '(A -24 B)) → ?
(NUMBERP-CADR '(256 PETITS BITS) → ?
(NUMBERP-CADR (CONS '1 '(1 2 3 5 8 13))) → ?

avec **NUMBERP-CADR** définie comme :

```
(DE NUMBERP-CADR (L)
    (NUMBERP (CADR (LISTE3 (CADDR L)(CAR L)(CADR L)))))
```

et **LISTE3** définie comme au chapitre précédent de cette introduction.

solutions

5.2. LA FONCTION IF

Revenons sur notre analogie de recettes de cuisine (ou de manuels quelconques) : si l'on teste la température de l'eau, c'est-à-dire si l'on se pose la question "l'eau est-elle bouillante ?", c'est probablement qu'on désire *agir différemment* suivant le résultat du test. On veut, par exemple, éteindre le feu ou verser l'eau dans la cafetière si l'eau est bouillante, et attendre que l'eau soit vraiment bouillante dans le cas contraire. Donc, on veut, suivant les résultats de l'application des tests, sélectionner entre différentes activités celle qui est adéquate.

En LISP, les prédicats servent exactement à la même chose : guider la suite de l'exécution du programme suivant les résultats de tests sur les données.

La fonction LISP responsable de ce 'guidage´ est la fonction **IF**. Donnons en tout de suite la définition syntaxique :

(IF *test* *action-si-vrai*
{$action_1$-*si-faux*
$action_2$-*si-faux*
. . .
$action_n$-*si-faux*}**)**

→ valeur de *action-si-vrai* si l'évaluation du test donne *vrai*

→ valeur de $action_n$-*si-faux* si l'évaluation du test donne *faux*

Cette définition peut se lire comme :

> **IF** est une fonction à nombre variable d'arguments. Le premier argument, *test*, est un prédicat. Le deuxième argument, *action-si-vrai*, est une expression LISP quelconque. Si le résultat de *test* est différent de **NIL** (donc, si le test donne le résultat *vrai*) la valeur ramenée du **IF** sera la valeur de l'expression *action-si-vrai*. Sinon, si le résultat de l'évaluation de *test* est égal à **NIL**, la machine LISP évalue séquentiellement une *action-si-faux* après l'autre et retourne la valeur de l'évaluation de $action_n$-*si-faux*.

Regardons un exemple. Voici un appel possible de **IF** :

(IF (NUMBERP 1) '(UN NOMBRE) '(PAS UN NOMBRE)) $\rightarrow$ **(UN NOMBRE)**

L'évaluation du test **(NUMBERP 1)** ramène en valeur le nombre **1**, qui est évidemment différent de **NIL**. Donc le résultat de l'évaluation de *test* donne *vrai*. Ceci implique, après la définition de la fonction **IF**, que la valeur ramenée par le **IF** tout entier soit la valeur de *action-si-vrai*, donc la valeur de **'(UN NOMBRE)**, donc la liste **(UN NOMBRE)**. Si au lieu de donner le test **(NUMBERP 1)**, on avait donné à LISP l'appel suivant :

(IF (NUMBERP 'HELLO) '(UN NOMBRE) '(PAS UN NOMBRE)) $\rightarrow$ **(PAS UN NOMBRE)**

En effet, puisque l'atome **HELLO** n'est pas un nombre, la valeur ramenée est la liste **(PAS UN NOMBRE)**, la valeur de l'unique *action-si-faux*.

Donnons encore quelques exemples :

(IF (NULL '(A B)) (CAR '(DO RE))(CDR '(DO RE))) $\rightarrow$ **(RE)**
(IF (CONSP (CONS NIL NIL)) (CONS NIL NIL) 'BRRR) $\rightarrow$ **(NIL)**
(IF NIL 1 2) $\rightarrow$ **2**

dans l'exemple précédent, le premier argument du **IF** n'est pas l'appel d'un prédicat. Toutefois, l'atome **NIL** se trouve en situation de prédicat, simplement par le fait qu'il se trouve dans la position de *test*. Le résultat de l'évaluation de *test* est donc égal à **()**.

Voici encore un exemple :

(IF '(A B C) 'AHA 'HMMM) $\rightarrow$ **AHA**

la même remarque que précédemment s'impose : la liste **(A B C)** n'est pas un prédicat en soi, mais de par sa situation en position de *test*, elle joue le rôle d'un *vrai* logique

Munis de cette fonction, nous pouvons enfin commencer à écrire de vraies (petites) fonctions LISP. Construisons d'abord une nouvelle fonction **NUMBERP-CADR?** qui ramène la liste **(**deuxième-élément **EST UN NOMBRE)** si le deuxième élément de sa liste argument est un nombre, et dans les autres cas, elle ramène la liste **(**deuxième-élément **N EST PAS UN NOMBRE)** :

```
(DE NUMBERP-CADR? (L)
    (IF (NUMBERP (CADR L))
        (CONS (CADR L) '(EST UN NOMBRE))
        (CONS (CADR L) '(N EST PAS UN NOMBRE))))
```

et voici quelques appels de cette fonction :

(NUMBERP-CADR? '(1 2 3)) $\rightarrow$ **(2 EST UN NOMBRE)**
(NUMBERP-CADR? '(DO RE MI)) $\rightarrow$ **(RE N EST PAS UN NOMBRE)**
(NUMBERP-CADR? NIL) $\rightarrow$ **(NIL N EST PAS UN NOMBRE)**

encore quelques exemples :

```
(DE CONS? (ELE L)
    (IF (CONSP L)(CONS ELE L)(CONS L '(N EST PAS UNE LISTE, MSIEUR))))
```

et voici quelques appels :

```
(CONS? 1 '(2 3 4))
                    →  (1 2 3 4)
(CONS? 1 2)
                    →  (2 N EST PAS UNE LISTE, MSIEUR)
(CONS? 'UNE (CONS? 'SOURIS (CONS? 'VERTE ())))
                    →  (UNE SOURIS VERTE)
(CONS? 'UNE (CONS? 'SOURIS 'VERTE))
                    →  (UNE VERTE N EST PAS UNE LISTE, MSIEUR)
```

Examinons la fonction suivante :

```
(DE TYPE? (ARG)
    (IF (NUMBERP ARG) 'NOMBRE
        (IF (ATOM ARG) 'ATOME 'LISTE)))
```

Tous les arguments du **IF** pouvant être des expressions LISP quelconques, rien n'exclut que l'un ou plusieurs des arguments soient des appels de la fonction **IF** elle-même. De telles imbrications servent en général à distinguer entre plus de 2 possibilités. Ici, dans la fonction **TYPE?**, nous distinguons 3 cas possibles : soit l'argument donné à l'appel est un nombre, la machine répond alors avec la valeur **NOMBRE**, soit il est un atome, la machine ramène l'atome **ATOME**, soit l'argument n'est ni un nombre, ni un atome, alors la machine suppose que c'est une liste et ramène en valeur l'atome **LISTE**. Voici quelques appels :

```
(TYPE? '(((IRREN) IST) MENSCHLICH))     →  LISTE
(TYPE? '(ERRARE HUMANUM EST))           →  LISTE
(TYPE? (CAR '(1 + 2)))                  →  NOMBRE
(TYPE? (CADR '(1 + 2)))                 →  ATOME
(TYPE? '(CECI EST FAUX))                →  LISTE
```

5.2.1. quelques exercices

1. Ecrivez une fonction **3NOMBRES** qui ramène l'atome **BRAVO**, si les 3 premiers éléments de sa liste argument sont des nombres, sinon elle doit ramener l'atome **PERDANT**. exemple :

```
(3NOMBRES '(1 2 3))                          →  BRAVO
(3NOMBRES '(1 CAT))                          →  PERDANT
(3NOMBRES (CONS 1 (CONS -100 (CONS -1 ()))))  →  BRAVO
```

2. Ecrivez la fonction **REV** qui inverse une liste de 1, 2 ou 3 éléments. Exemples d'appels :

```
(REV '(DO RE MI))          →  (MI DO RE)
(REV '(EST IL FATIGUE))    →  (FATIGUE IL EST)
(REV '(JO GEHNS))          →  (GEHNS JO)
```

3. Voici une fonction bizarre :

```
(DE BIZARRE (ARG1 ARG2)
    (IF (CONSP ARG2)
        (CONS ARG1 ARG2)
        (IF (CONSP ARG1)
            (CONS ARG2 ARG1)
            (IF (NULL ARG2)
                (IF (NULL ARG1) '(ALORS, BEN, QUOI)
                        (CONS ARG1 ARG2))
                (IF (NULL ARG1) (CONS ARG2 ARG1)
                        (CONS ARG1 (CONS ARG2 NIL)))))))
```

Que fait cette fonction ? Donnez les valeurs des appels suivants :

(BIZARRE ()()) → ?
(BIZARRE 1 '(2 3)) → ?
(BIZARRE '(2 3) 1) → ?
(BIZARRE 'TIENS 'C-EST-TOI) → ?
(BIZARRE () (TYPE? 432)) → ?
(BIZARRE (CDR (CONS? (TYPE? 'INTEL) 432)) (CAR '(DO RE MI))) → ?

Avant de continuer, assurez-vous que jusqu'ici vous avez bien compris. Si vous avez des problèmes, reprenez les exercices et les exemples. Et n'oubliez pas de les faire tourner sur une machine, vérifiez les, modifiez les légèrement et trouvez les erreurs.

chapitre précédent | solutions | chapitre suivant