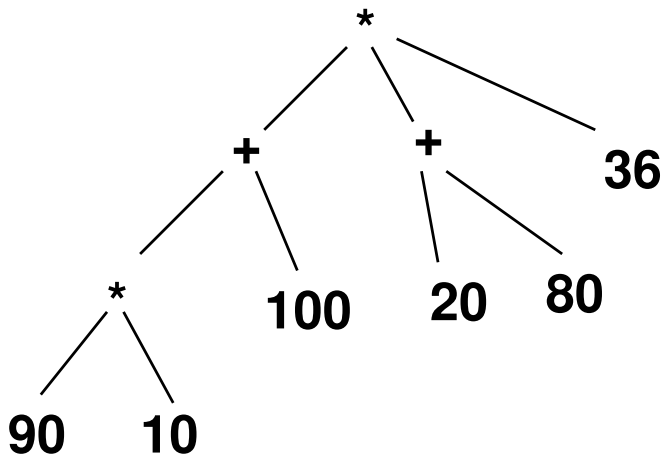
(Séquence 9.1

S-expression

S-expressions




```
(* (+ (* 90 10) 100)
   (+ 20 80)
   36)
```

L'expression Scheme ci-dessus ressemble à une liste (de quatre éléments) de type **LISTE(α)** mais les éléments de la liste ne sont pas de même type. On peut aussi voir cette expression comme un arbre général.
C'est une ***S-expression*** (pour *expression symbolique*)

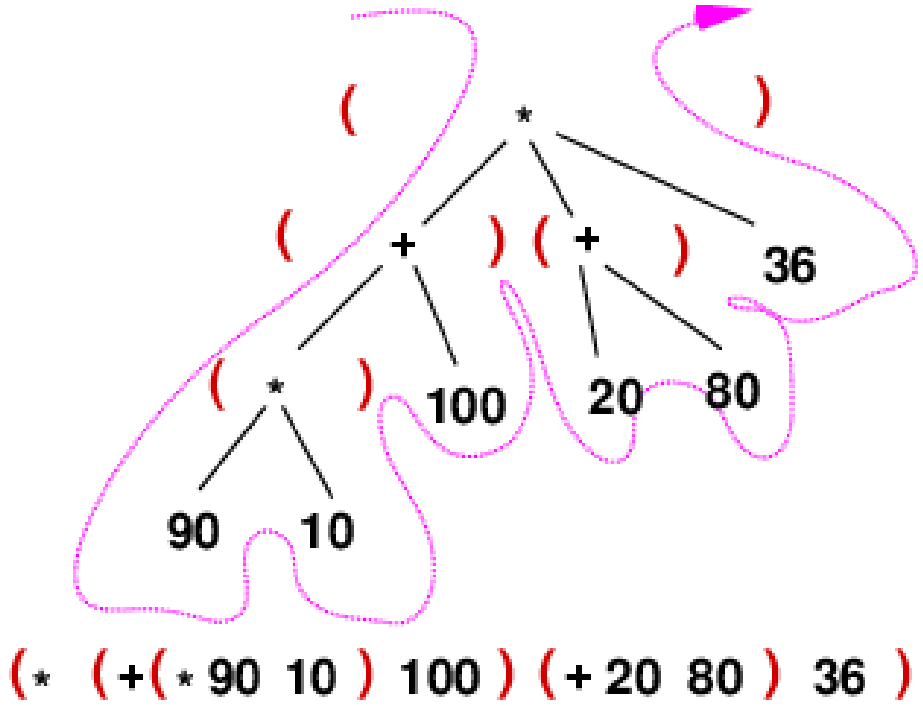
Syntaxe des S-expressions

$$
\begin{array}{lcl}
\langle Sexpression \rangle & \rightarrow & \langle atome \rangle \\
 & & \texttt{(}\ \langle Sexpression \rangle^{\star}\ \texttt{)} \\
\langle atome \rangle & \rightarrow & \langle Nombre \rangle \\
 & & \langle bool \rangle \\
 & & \langle string \rangle \\
 & & \langle Symbole \rangle
\end{array}
$$

Les ***lexèmes*** $\langle Nombre \rangle$, $\langle bool \rangle$, $\langle string \rangle$ et $\langle Symbole \rangle$ sont définis dans la carte de référence.

Arbres généraux et S-expressions

Un arbre général peut être représenté par une S-expression dont le premier élément est l'étiquette de la racine et dont le reste est constitué de la forêt de ses sous-arbres immédiats.



Implantation des AG par S-expressions

```
(define (ag-noeud e F)
  (cons e F))

(define (ag-etiquette ag)
  (car ag))

(define (ag-foret ag)
  (cdr ag))
```

Pourquoi alors avoir une barrière d'abstraction pour les arbres généraux ?

- Pour se concentrer sur la récursion dans les arbres généraux,
- parce qu'il y a d'autres implantations pour les arbres

Autre implantation des AG

```
(define (ag-noeud e F)
  (list "*AG*" e F))

(define (ag-etiquette ag)
  (if (and (pair? ag)
           (equal? (car ag) "*AG*") )
      (cadr ag)
      (erreur 'ag-etiquette "Pas un AG" ag) ) )

(define (ag-foret ag)
  (if (and (pair? ag)
           (equal? (car ag) "*AG*") )
      (caddr ag)
      (erreur 'ag-etiquette "Pas un AG" ag) ) )
```

Implantation par les S-expressions

Pourquoi des S-expressions ?

Implantation par les S-expressions

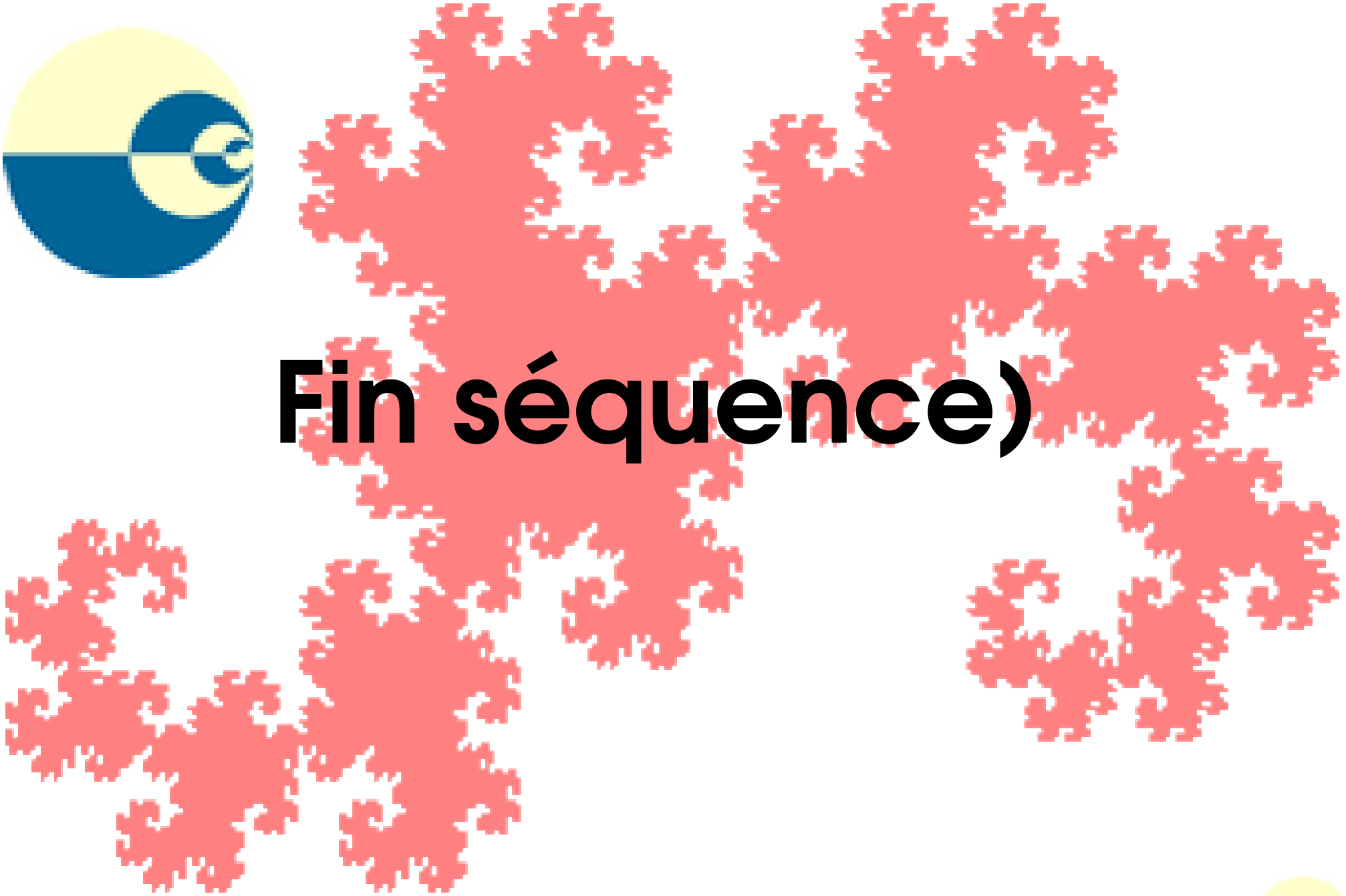
Pourquoi des S-expressions ?

- Parce qu'elles sont simples à citer.

```
(ag-noeud '+
          (ag-noeud '* (ag-noeud 90 (list))
                       (ag-noeud 10 (list)) )
          (ag-noeud 100 (list)) )
```

s'écrit directement :

```
'(+ (* 90 10) 100)
```



Fin séquence)