(Séquence 1.4

Grammaire

Grammaire

Grammaire : ensemble des règles à suivre pour composer un texte

Une expression est **syntaxiquement** correcte si elle respecte la grammaire

Toute expression syntaxiquement correcte n'est pas forcément **sémantiquement** correcte !

Erreur de syntaxe : `(define pi)`
Erreur de type : `(+ 1 "un")`
Erreur téléologique : `(+ 1 2 10)` au lieu de `(+ 12 10)`

Règles de composition

<programme> → *<expression-ou-définition>**
<expression-ou-définition> → *<expression>*
<expression-ou-définition> → *<définition>*
<expression> → *<constante>*
<variable>
<forme-spéciale>
<application>

L'étoile marque une répétition quelconque, un signe « + » marque une répétition quelconque mais non vide.

Application de fonction

<application> → **(** *<fonction>* *<argument>** **)**
<fonction> → *<expression>*
<argument> → *<expression>*

et par exemple :

```
(+ (quotient 7 2) (* 5 6 2) 8 (abs -3))

(positive? (abs -3))
```

Définition de fonction

La définition de fonction permet :

- de définir de nouvelles fonctions

$$x \to x^2$$

- de nommer ces nouvelles fonctions.

$$\text{carré} : x \to x^2$$

- pour pouvoir ensuite les appliquer

$$\text{carré}(a+1)$$

Le pourquoi de cette grammaire

Ainsi définir que carré$(x) = x^2$ s'écrit :

- On enserre entre parenthèses :

```
( carré(x) = x^2 )
```

- On place un mot en tête pour indiquer ce que c'est :

```
(define carré(x) = x^2 )
```

- On convertit en polonaise parenthésée préfixée

```
(define (carré x) = (* x x))
```

- On élimine le bruit syntaxique

```
(define (carré x) (* x x))
```

- et finalement, l'on obtient :

```
;;; carre: Nombre -> Nombre
;;; (carre x) rend le carré du nombre x
(define (carre x)
   (* x x) )
```

Grammaire des définitions

<définition> → **(define (** *<nom-fonction>* *<variable>** **)** *<corps>* **)**

<corps> → *<définition>** *<expression>*

Quatre étapes pour une définition

1. Donner la spécification de la fonction
2. Inventer la composition d'opérations menant au résultat cherché : l'algorithme,
3. Écrire l'algorithme en une expression syntaxiquement correcte
4. Tester la fonction en automatisant les tests avec `verifier`

Ainsi

```
;;; carre: Nombre -> Nombre
;;; rend le carré d'un nombre
(define (carre x)
   (* x x) )
(verifier carre
   (carre 1)   => 1
   (carre -2)  => 4
   (carre 2.1) => 4.41 )
```

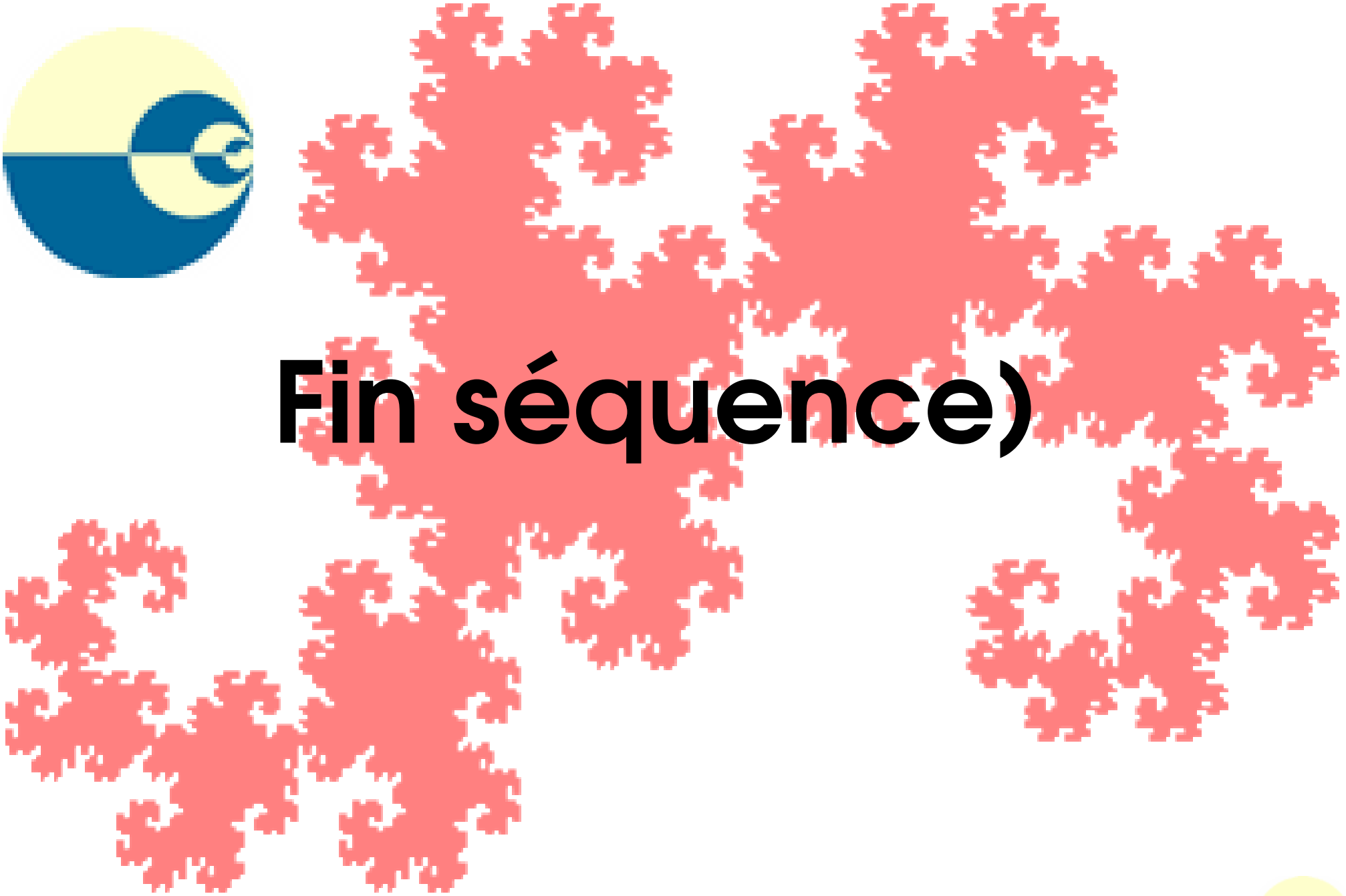
Alternative

<alternative> → **(if** *<condition>* *<consequence>* *<alternant>* **)**
<condition> → *<expression>*
<conséquence> → *<expression>*
<alternant> → *<expression>*

Une **condition** est une expression ayant pour valeur

- soit vrai c'est-à-dire `#t` pour *true*
- soit faux c'est-à-dire `#f` pour *false*

Ainsi `(if (positive? 3) "oui" "non")` vaut `"oui"`



Fin séquence)