

15/09

ES54

CHAPITRE 3: RECURSION

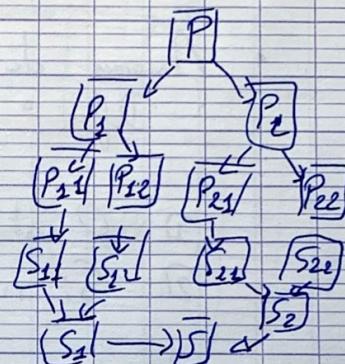
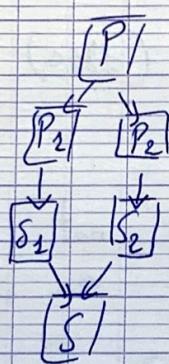
→ Diviser et conquérir

↳ Pb trop complexe ? Diviser en pb's simples

↓
les résoudre

↓

Grand pb résolu.



P
↓
P
↓
P
↓
S
↓
S
↓
S
↓
S.

Def: Object récursif

→ Défini avec Poi - m.

Ex :

- Suites
- Chaîne de caract. (Caract + chaîne ou + rien)

Toujours.

→ Transformable en fonc° itératives.

- Cas de base

→ Cas trivial, résolvable immédiatement.

→ Evite les boucles ∞ .

- Lin avec les maths.

- Axiomes de Peano (1880)

Déf. 0 de \mathbb{N} .

1) 0 est un ent. nat.

2) Si $n \in \mathbb{N}$, son succ. $(n+1)$ aussi.

- Algorithmes :

- Fonc° et procédures récursives (Generative)
- Structure de donnée (Naturelle / "Structural")

- Aucun pb n'est intrinsèquement récursif

↳ Certains sont + facilement solvables récursivement

- Tt algo. recu. peut être "récursive".

Fct et procédures recu.

→ S'appelle efficacement

Ex : Factorielle

$$\left\{ \begin{array}{l} n! = n \times (n-1)! \\ 0! = 1 \end{array} \right.$$

Contraintes sur la fct et les param.
□ Précondi° : Factorielle(n) n'est def que si $n \geq 0$
□ Postcondi° : Factorielle(n) = $n!$.

Ce qui est attendu.

Principe général :

```
if CondBase Then CasBase  
else CasGen  
end.
```

→ Cond : Cond° Binaire,

→ Cond = True → Cas de base (Pas d'appel rec.)

→ Cond = False → Cas gen (Appel recu).

Autres types de reco

- Recursion multipli : (Pascal)

- Recursion croisée (Pair / Impair)

- Reco. imbriquée (Param. de l'appel recu. est un app. recu.)

• Struct. données recu

Def. Structure de données qui s'utilise elle-même.

Ex : liste : ~~Liste~~

→ Recu. simple à comprendre, difficile à implementer
(\oplus Trust issue).

→ 2 :

Holistique : Le tout est mieux que ses parties

Reductionniste : Le tout se comprend si on comprend ses parties et leur Σ .

Comment résoudre un pb récursivement ?

1 Choix du param.

2 Résoudre cas simples = Cas de base

3 Mettre en place la récurs.

- Supposer qu'on peut résoudre le pb
sur une tabl. (\oplus) petite et exploiter ce résultat

4 Résoud Ecrire le cas général.

5 Ecrire les ~~instructions~~ d'annts (Cas de base)
cond: 0

Tri Fusion

1. Diviser la liste en 2 parties
2. Trier les 2 sous-listes récursivement
3. Fusionner les sous-listes triés.

Complexité :

- Temps : $\text{Pog}(n)$ appels réc., les appels sont fins:
 $\rightsquigarrow O(n \times \text{Pog}(n))$.
- Mémoire: Copier le tableau $\rightsquigarrow 2n$
+ $\text{Pog}(n)$ pour la pile

Utilisé dans C.

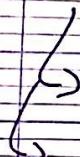
Tri rapide

1. Choisir un pivot
2. Parcours de la liste.
 - ↳ Ⓛ petit que le pivot à gauche
 - ⓑ gd que le pivot à droite

CS54

1/09

Pb de la RECUR : on répète P_c schema d'appel de fact



Temps :

Coût Θ élevé.

Mémoire :

Appels $\Theta \delta^t$ d'espace.

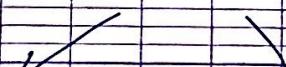
Si P_c résultat correspond au calcul du cas de base
on parle de RECURSIVITÉ TERMINALE (Tail recursion).

Toute fonc^o recursive peut être transfo. en itérative
(\neq méthodes).

\hookrightarrow Rec. terminale : Très simple.

\hookrightarrow Rec. non-terminale : 2 méthodes

Transform en
rec. terminale et
transfo.



Méth. générique
(Θ long)

Algo recu^{form} basique :

$f^*(n)$

if cond (a) then BaseCase (a)
else : $T(a)$, $r \leftarrow f^*(a)$

Equi. itératif :

$P^*(a)$:

$v \leftarrow a$

repete

until P cond (v) do

$T(v)$

$v \leftarrow R(v)$

end

BaseCase (v)

Si rec. non terminée.

↳ On peut parfois stocker le résultat d'un calcul avec un seul paramètre en plus f



Si calcul pas dans P branche. Commutativité.

Opérations
Associativité.

Cas général : Si on ne peut pas passer à l'arc en ciel.

→ On doit simuler le fonctionnement de la pile.

Backtracking

Utilisé en combinatoire ou optimisation

↳ Problème avec trop d'un grand nombre de solutions possibles (Combinaison d'un coffee, "traveling salesman" etc...)

Backtracking :

↳ On essaie des solutions et on coupe les branches des mauvaises solutions.

Schéma général : Récursivité dans une itera-

Principes du bt :

- Ne pas oublier de défaire les transformations