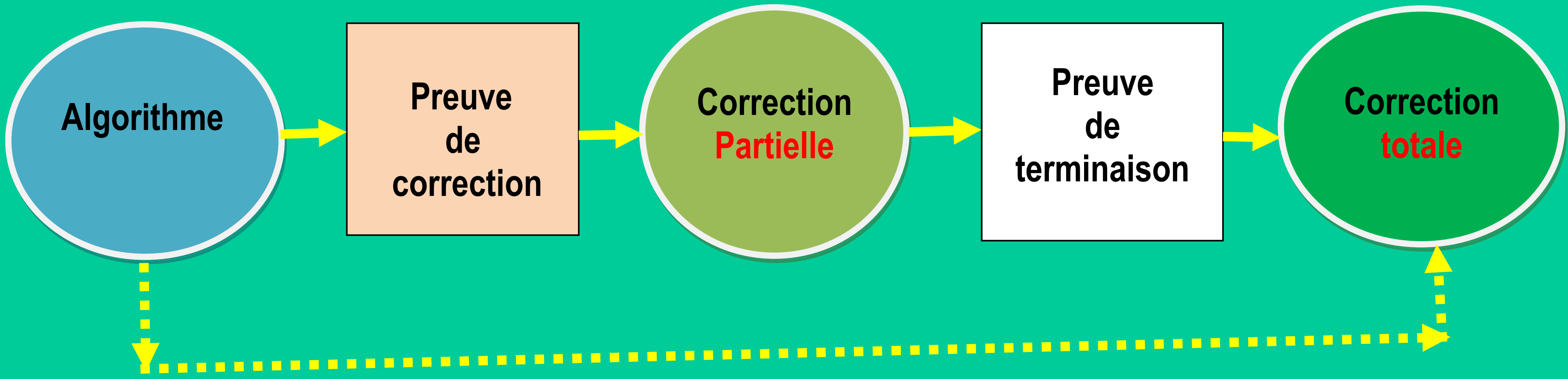
La correction d'un algorithme

Un algorithme:

- qui est **partiellement correct**
- qui se termine **toujours**

est alors dit **totalement correct**.

En résumé


Algorithme
Preuve de correction
Correction Partielle
Preuve de terminaison
Correction totale

Comment apporter la preuve de correction ?

Preuve par induction

Preuve par assertion

I-La preuve par induction

En résumé, une **preuve par induction** est constituée de trois parties:

- une **hypothèse de récurrence**,
- une **base**
- et une **étape de récurrence**.

L'**hypothèse de récurrence** décrit l'énoncé à **prouver** : c'est le but de l'étape (3) dans l'exemple.

La **base prouve** le cas de départ : c'est l'étape (1) dans l'exemple.

L'**étape de récurrence** permet d'aller :

- de la **base**
- vers des **cas progressivement supérieurs**.

II- La preuve par assertions

- **Spécification**

$\{x \geqslant 0 \wedge y \geqslant 0 \wedge x = x_0 \wedge y = y_0\}$ — Précondition

```
while (y > 0)
    {
        int Save = y;
        y = x % y;
        x = Save;
    }
```

Code à vérifier

$\{x = pgcd(x_0, y_0)\}$ — Postcondition

Triplet de Hoare

Une portion du programme ou **code** est **correcte** si le **triplet de Hoare**:

{ P} **code** {Q}

est **vrai**.

P : **précondition**
Q : **post-condition**

Correction d'une boucle

On part du triplet de base suivant:

```
{P}
INIT
WHILE  C
        CORPS
FIN
 {Q}
```

pour construire le triplet:

$$
\begin{array}{l}
\{P\} \\
\textbf{INIT} \\
\{\, I \,\} \\
\textbf{WHILE } C \\
\qquad\qquad \{I \wedge C\} \\
\qquad\qquad \textbf{CORPS} \\
\qquad\qquad \{I\} \\
\{I \wedge \neg C\} \\
\textbf{FIN} \\
\{Q\}
\end{array}
$$

Pour prouver que le triplet est correct :

1-on met en évidence une assertion particulière **I**, appelée **invariant de boucle**.

L'**invariant** décrit une propriété **pendant** la boucle.

2-on doit prouver que successivement:

-**avant** la boucle :

{**P**} **INIT** { I } est correct

-**pendant** la boucle

$\{I \wedge C\}$ **CORPS** { I } est correct

-**à la fin** de la boucle :

$\{I \wedge \neg C\}$ **FIN** {**Q**} est correct

Si on a plusieurs boucles **imbriquées**, on les traite :

- **séparément**,
- en démarrant avec la boucle **la plus interne**.

Formalisation de la technique de preuve:

#Inv: invariant de la boucle while

#Inv : propriété vraie **avant** la boucle
while condition:

--montrer que si #Inv est vrai en **haut** de la boucle
[**itération du while**]
--alors #Inv est vrai en **bas** de la boucle

finWhile
-- en déduire que #Inv est vrai **après** la boucle

Exercice 1 : calcul de la suite de Fibonacci

La suite de Fibonacci notée **(F_n)** est définie comme suit:

$F_0 = 0$;

$F_1 = 1$;

$F_2 = F_1 + F0$;

…

$F_n = F_{n-1} + F_{n-2}$

```
Fibonacci(n)
if n ≤ 1
            prev := n;
else
            { pprev := 0 ; prev := 1;
              i := 2
            }
while (i ≤ n)
            { f := prev + pprev;
               pprev := prev;     prev := f;
               i := i+1
            }
return (prev)
```

Montrer que ce programme calcule les termes de la suite de Fibonacci.

Exercice 2 : calcul de la factorielle

Soit l'algorithme suivant qui calcule n !

```
(1)  lire (n)
(2)  i := 2
(3)  fact := 1
(4)  tant que i ≤ n faire
(5)        fact := fact * i
(6)        i := i + 1
   fintantque
(7)  écrire (fact)
```

Montrer la correction totale de l'algorithme précédent qui calcule n ! pour les entiers n ≥ 1.