Autour du concept d'algorithme

intensionnalité, complétude algorithmique, programmation et modèles de calcul

Pierre Valarcher
LACL, IUT Fontainebleau - Sénart
Université Paris Est Créteil

Plan

- Intensionnalité, complexité : du fonctionnel à l'impératif

Langages fonctionnels	intensionnalité
Langages impératifs	façon de calculer
	complexité

- Complétude algorithmique : une notion de classe d'algorithmes pour les fonctions primitives récursives

Classe d'algorithmes	Langage fonctionnel
	Langage impératif

- Caractérisation des algorithmes issus de modèles de calcul séquentiel

Modèle de calcul séquentiel	Classe d'algorithmes

Intensionnalité et complexité
du fonctionnel à l'impératif

Intensionnalité des programmes

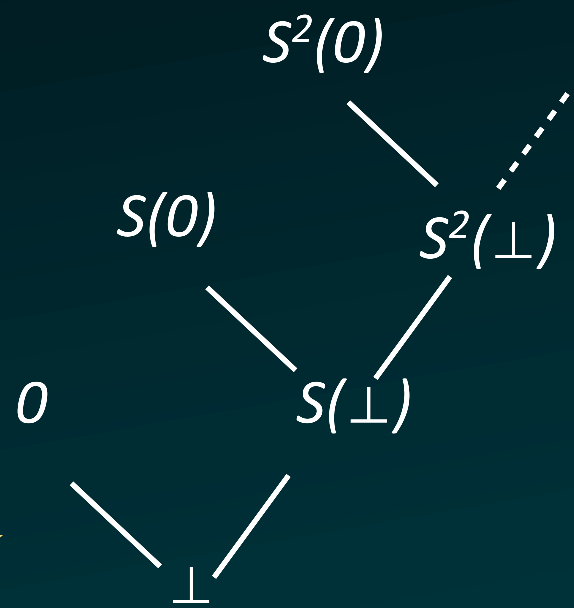
$add : x,y \mapsto x+y$

extension

programme fonctionnel

```
add1(0,y) = y
add1(S(x), y) = S(add1(x,y))
```

```
add2(0,y) = y
add2(S(x), y) = add2(x,S(y))
```



interprétation

$[\![\mathtt{add1}]\!](S^n(\bot),S^p(\bot))= S^n(\bot)$

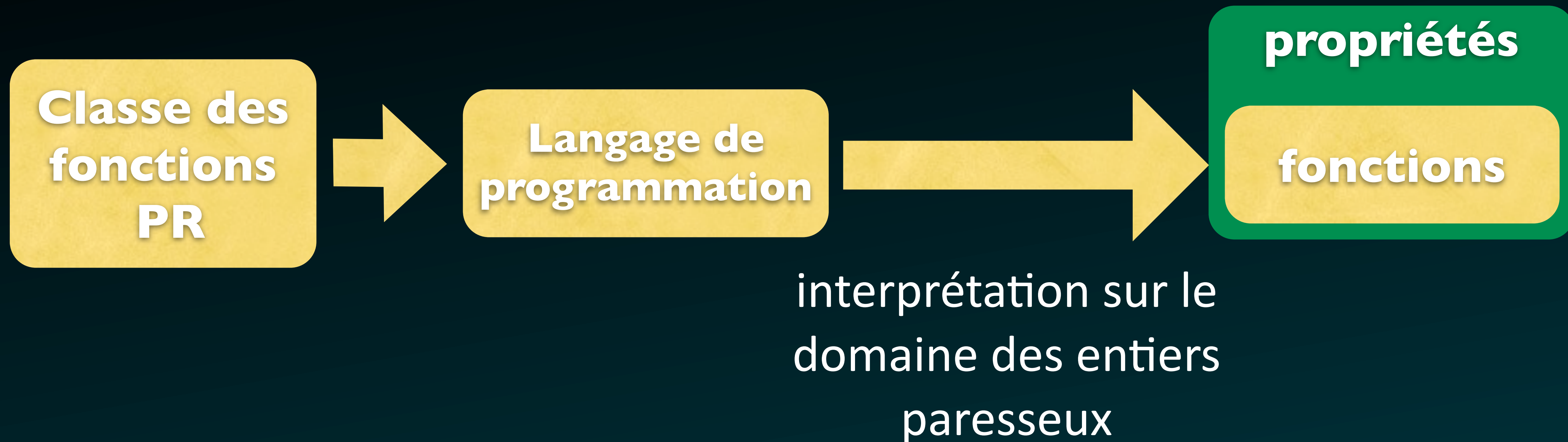
$[\![\mathtt{add1}]\!](S^n(0),S^p(0))= S^{n+p}(\bot)$

comportement intensionnel

$[\![\mathtt{add2}]\!](S^n(\bot),S^p(\bot))= \bot$

$[\![\mathtt{add2}]\!](S^n(0),S^p(0))= S^{n+p}(0)$

Intensionnalité



- [1991] L. Colson : le langage de **la récursion primitive classique** a la propriété d'**ultime obstination** : ne permet pas de calculer alternativement sur ses entrées.

$$f(0,y) = g(y)$$
$$f(S(x), y) = h(x, f(x,y),y)$$

Intensionnalité des langages

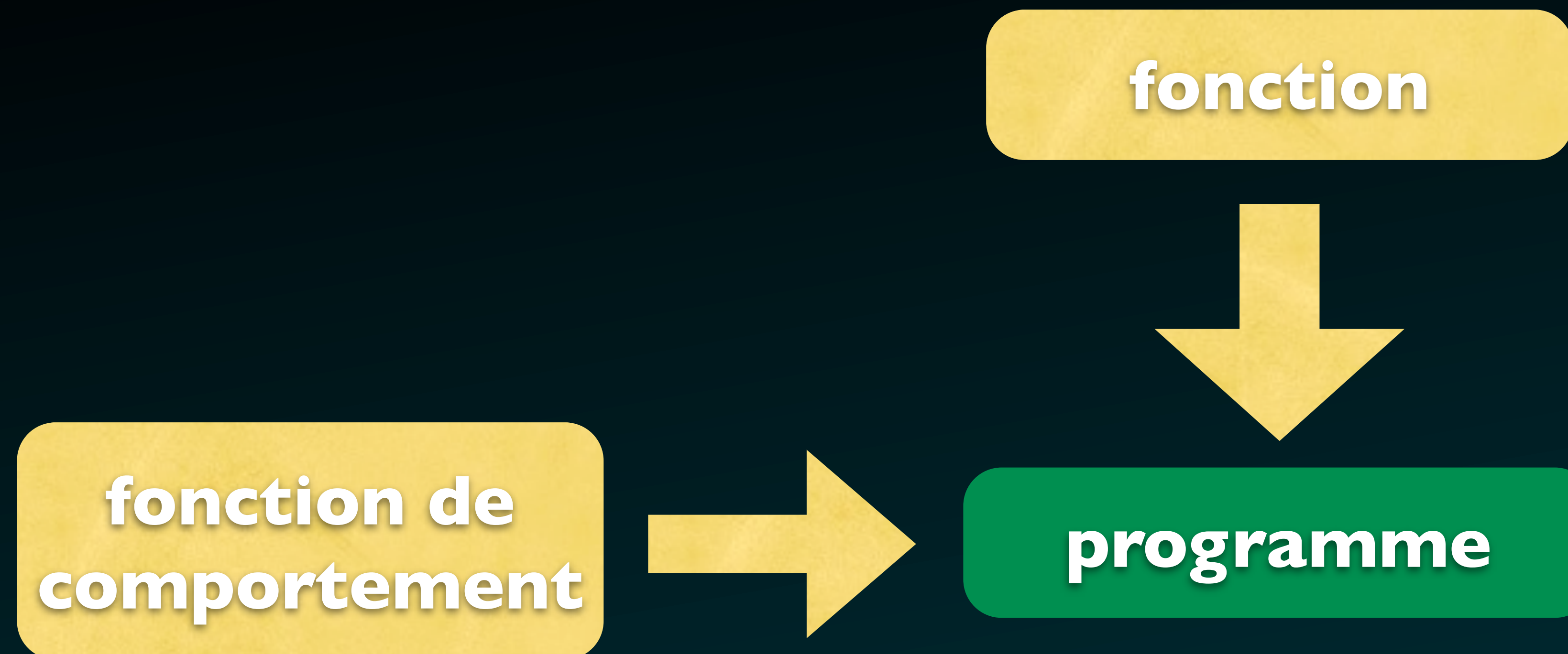
interprétation sur le domaine
des entiers paresseux

- [1996] P. Valarcher : la **classe des fonctions** qui sont des comportement du langage de la récursion primitive classique est **définissable** et est exactement la classe des fonctions primitives récursives avec **le cas de base nul**.

- (2003) Étendu avec R. David pour les langages PR alternée et PR mutuelle

$$f(0,y) = 0$$
$$f(S(x), y) = h(x, f(x,y),y)$$

Théorèmes de représentation

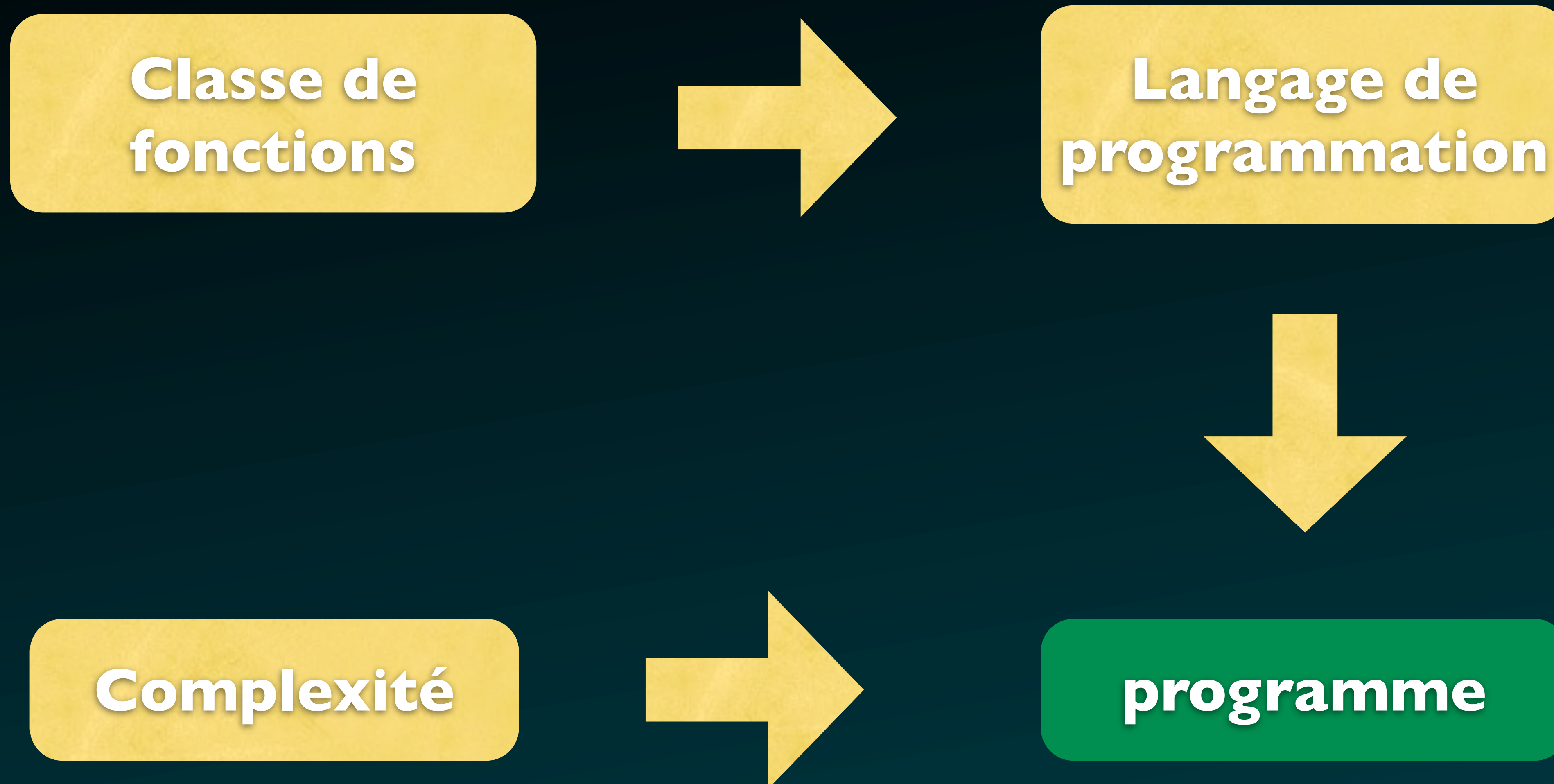


- [1991] L. Colson : étant données **une fonction PR unaire** f et **une fonction PR de comportement unaire** g (croissante et $\leq f$) alors on peut construire un programme qui **calcule f avec le comportement g** mais dans le **système T**.

- [1996] P. Valarcher : **récursion avec paramètres variables**
- [2002] P. Valarcher : **système T**

***n*-aires**

Intensionnalité et complexité structurelle

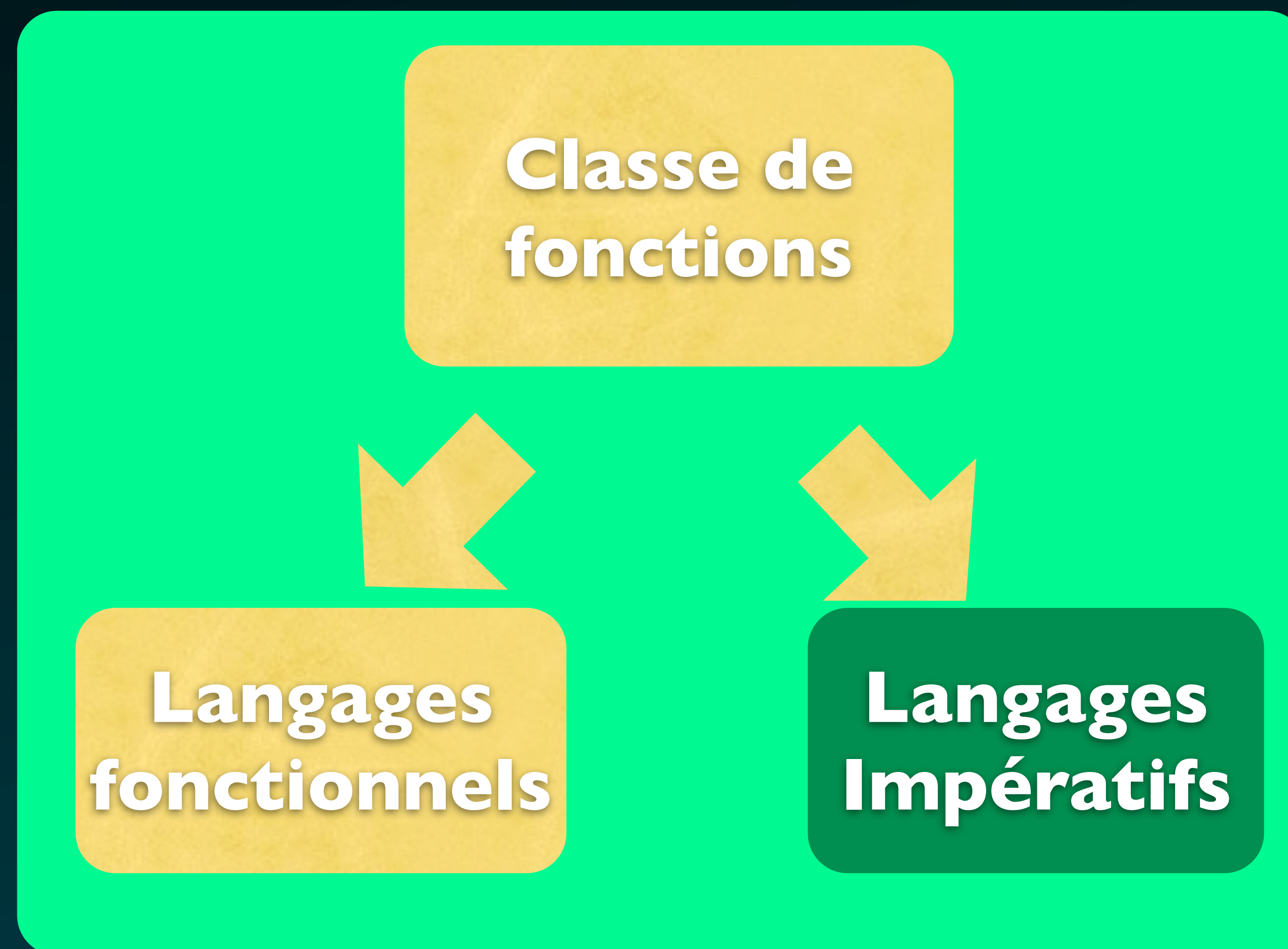


l'algorithme *min* qui calcule le minimum de deux entiers en parcourant alternativement ses entrées est en temps le minimum de ses entrées

Complexité structurelle

- [1991] L. Colson : le *min* **n'est pas programmable** dans le langage de la **récursion primitive classique** en temps O(min) (quelle que soit **la stratégie**)
- [1996] D. Fredholm : le *min* n'est pas programmable dans le langage de la **récursion primitive étendue aux listes (**avec stratégie d'**appel par valeur)**
- [1998] L. Colson & D. Fredholm : le *min* n'est pas programmable dans le langage de la **récursion primitive étendue avec des paramètres fonctionnels** (Système *T* avec stratégie d'**appel par valeur**)
- [2003] Y. Moschovakis : les algorithmes en ***logtime*** ne sont pas programmables dans le langage de **récursion primitive étendue aux linéaires par morceaux**: exemple le ***pgcd*** **de Stein** (*O(log x + log y)*)
- [2004] L. van den Dries: l'**algorithme d'Euclide** n'est pas programmable dans le langage de la **récursion primitive classique** : le calcul du reste n'est pas linéaire par morceaux

Le cas des langages impératifs ?



Un langage impératif pour les fonctions PR

- [1970] Meyer et Ritchie: langage LOOP
  - ✓ des variables, des expressions : variables, 0, +1, -1
  - ✓ l’affectation
  - ✓ la séquence
  - ✓ l’itération bornée

un **programme** pour calculer le minimum de *a* et de *b*

```
x := a;
y := b;
r := b;
for i := 1 to y loop  x := x - 1;   end loop;

for i := 1 to x loop  r := a;   end loop;
```

Complexité : le cas des langages impératifs

- [2006] T. Crolard, S. Lacas, P. Valarcher : **le langage LOOP** a les **mêmes lacunes** que le schéma de la **récursion primitive classique** (stratégie d'appel par valeur)

- [2009] T. Crolard, E. Polonowski, P. Valarcher : **le langage LOOP$^{\omega}$** (LOOP + **paramètres procéduraux**) a les **mêmes lacunes** que le **système T** (stratégie d'appel par valeur)

✓ description d'une **sémantique opérationnelle** pour chacun des langages

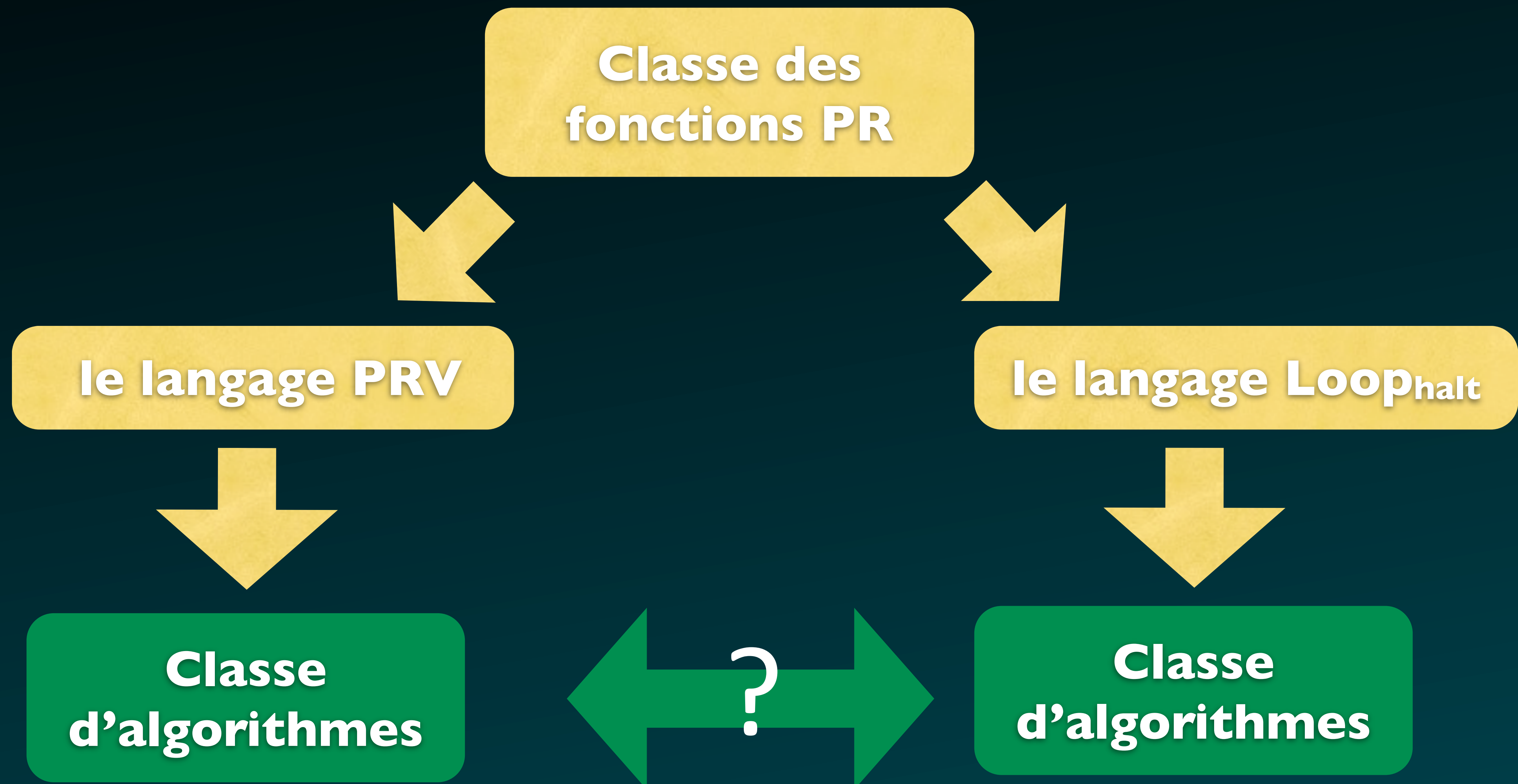
✓ **traduction** du langage impératif vers le langage fonctionnel avec **simulation pas à pas**

Des intuitions sur l'expressivité de certains langages

Les lacunes algorithmiques comme le *min* et le *pgcd* sont comblées par les langages :

- **PRV** : **récursion primitive avec paramètres variables**
- **$\mathbf{LOOP_{halt}}$** : **langage LOOP** étendu avec **le test à zéro** et **la sortie de boucles**

Classe d'algorithmes



Complétude algorithmique

ou comment lier une classe d'algorithmes
et des langages de programmation

Qu’est ce qu’un algorithme ?

- Très souvent une **définition informelle**

An algorithm is a finite, definite, effective procedure, with some output.

Knuth, 1968

- et en fait au travers d’un **modèle de calcul ou d’un langage de programmation**

an algorithm is a computational process defined by a Turing machine

Savage, 1987

Algorithms in C

Sedgewick, 1998, 3rd

- **TM, RAM, CA, système equationnel, ..., Pascal, C, Java, ML, ...**

Peut on définir une classe d’algorithmes **indépendante des langages de programmation ou des modèles de calcul ?**

Des modèles de calculs aux algorithmes

- **Théorie de la calculabilité**
  - intensionnalité est inconsciente
  - Une équivalence du point de vue de l'**extensionnalité** (entrée/sortie)
- **Théorie des Algorithmes (indépendants des langages)**
  - [1953, 1958] Kolmogorov , Uspensky : KU machine
  - [1976] D. Grigoryev :«KU machine more powerful than TM»
  - [1970] Schönhage : Storage Modification Machine, KU + pointeur
  - [1985] Y. Gurevich : **Abstract State Machines** (ASM)
  - [1995] Y. N. Moschovakis : un algorithme se définit à partir d'**équations à la Herbrand/Gödel**

Deux définitions pour la notion d’algorithme

Un algorithme est relatif à des opérations de bases

- Approche Y. N. Moschovakis

système d’**équations récursives** + point-fixe

- Approche Y. Gurevich

système de **transitions** dont les états sont des **structures logiques du 1er ordre**
Abstract State Machine

Abstract State Machine

- symboles de fonctions (l'état) + interprétations

statique et dynamique

- un programme (ensemble de règles R)
  - conditionnelle : ***if C then R_1 else R_2***
  - mise à jour (affectation) : ***$f(t_1, \dots, t_N) := t_0$***
  - simultanéité : ***R_1 || … || R_N***

Une **meta-boucle** : on exécute toutes les règles applicables

Arrêt : deux règles en conflit ou pas de changement d'état

- Un algorithme pour le *min* avec (+1, -1, =0) et (or, and, ¬)

```
if ¬(x = 0) and ¬(y = 0) then x := x-1 || y := y-1
if (x = 0) then result := a
if (y = 0) then result := b
```

Thèse de Y. Gurevich

- Axiome 1 : Un algorithme est un **système de transitions**
- Axiome 2 : Les états du système sont des **structures du 1er ordre**; la signature de la structure ne varie pas
- Axiome 3 : une **famille finie fixée** de termes est suffisante pour effectuer un nombre borné de modifications de l'état

[2000] Y. Gurevich : tout **algorithme** est **simulable pas à pas** par une **ASM**

Une classe d'algorithmes pour les fonctions PR

- les ASMs sont **trop générales** :
  - ✓ l'interprétation de la partie statique peut être une fonction non primitive récursive
  - ✓ la partie dynamique peut comporter des fonctions *n*-aires
  - ✓ on peut exprimer des algorithmes qui ne terminent pas : $x := x + 1$
- On va restreindre :
  - ✓ le **vocabulaire et son interprétation** : on doit aboutir à un langage de programmation pour les fonctions PR
  - ✓ on **borne en temps** les exécutions par une fonction PR

la classe $Algo_{PR}$

$Algo_{PR}$ *c'est l'ensemble des (A, f)*

- Une **ASM** (*A*) construite sur
  - ✓ un vocabulaire statique unaire ou 0-aire (*+1, -1, =0, ...)* et un vocabulaire dynamique *0*-aire (les variables)

- Une **fonction primitive récursive** (*f*)

  un majorant de la complexité

Complétude algorithmique

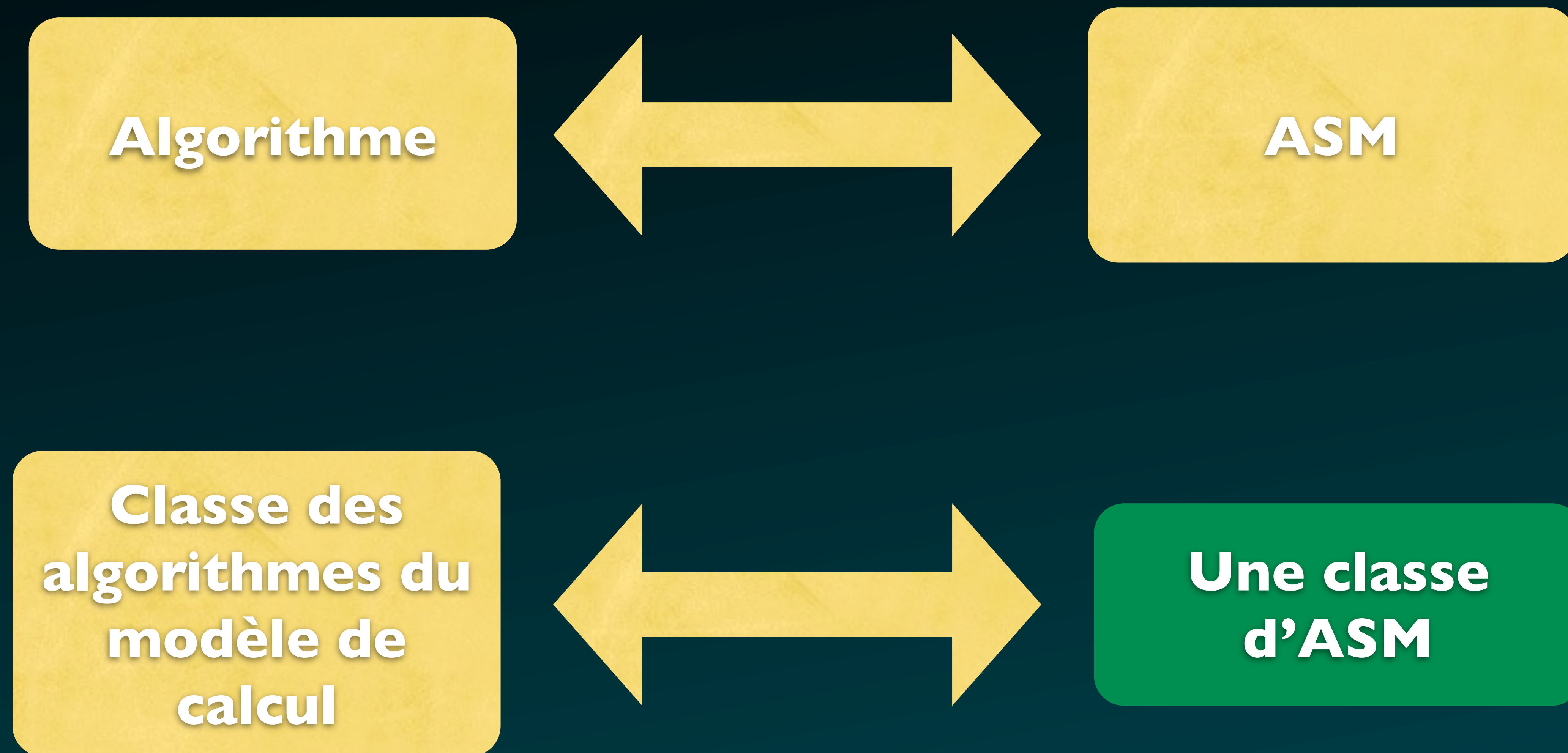
- [2005, 2010] Ph. Andary, B. Patrou, P. Valarcher : la **classe des algorithmes** Algo_{PR} correspond **exactement** à la **classe des programmes** de LOOP_{halt} .
  - ✓ par injection de *A* (traduit) dans le programme calculant *f*

- [2010] D. Michel, P. Valarcher : la **classe des algorithmes** Algo_{PR} est capturée par la **classe des programmes** de PRV (stratégie mixte)
  - ✓ par simulation de LOOP_{halt}

Abstract State Machine

Système d'équations récursives

Les modèles de calcul comme classes d'algorithmes



Caractérisation des modèles de calcul séquentiel

Identification littérale

- On cherche non seulement **simuler pas à pas** les modèles de calcul mais on cherche aussi à **identifier les éléments** du modèle **avec les fonctions statiques et dynamiques** de l'ASM
    - ✓ les programmes du modèle sont définissables
    - ✓ fidèle à la structure du modèle
- La solution est de modifier les Abstract State Machine en **Evolving Multi-Algebra**

Evolving Multi-Algebra

- Modification des ASMs
  - en ne permettant pas le mélange des symboles de fonctions nécessaires au modèle étudié : état est **multi-sorté** (typage des symboles)
  - en introduisant une **fonctionnelle** pour rendre compte du programme à simuler

[2010] S. Grigorieff, P. Valarcher : une classe d'**Evolving Multi-Algebra** pour chaque **modèle de calcul séquentiel**

- La classe d'EMA définit en fait un **modèle plus large** qui met en évidence le **paradigme de base du modèle** :
  - Machine de Turing : actions locales
  - RAM : indirections

Résumé

PR

Système T

Langages fonctionnels

Langages impératifs

intensionnalité

algorithmes

complexité

Classes de fonctions

ASM

PR

Classe d'algorithmes

Langage fonctionnel

Langage impératif

Calculables

Modèles de calcul séquentiel

Classes d'EMA

Perspectives

- La complétude algorithmique
  - ✓ d'autres classes d'algorithmes
  - ✓ d'autres langages plus «réalistes»
- Evolving Multi-algebra
  - ✓ permettre l'identification de modèle de plus haut niveau
    - ➡ paraléllisme
    - ➡ récursion
  - ✓ revisiter les axiomes
- Algorithme de Y. Moschovakis
  - ✓ lien avec les ASM/EMA

Merci de votre attention