Programmation et confiance

HDR
13 juin 2008

Pierre-Etienne Moreau

8

45

Tom

- Filtrage
- Règles
- Stratégies

dans des langages classiques

Constructions de haut niveau

Fondations théoriques solides

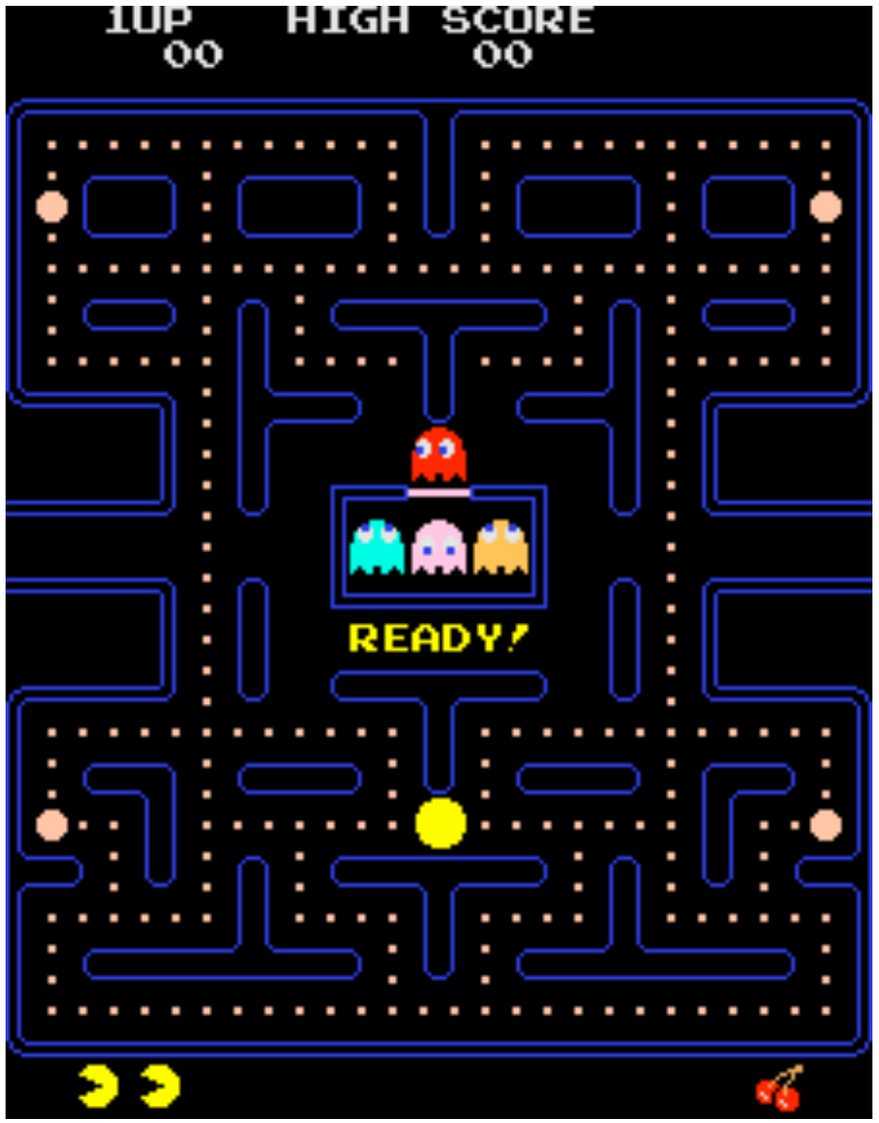
Utilisé dans les milieux académiques et industriels

- décrire des transformations
- support à la recherche / prototypage
- compilateurs
- outils de preuve
- traduction de requêtes

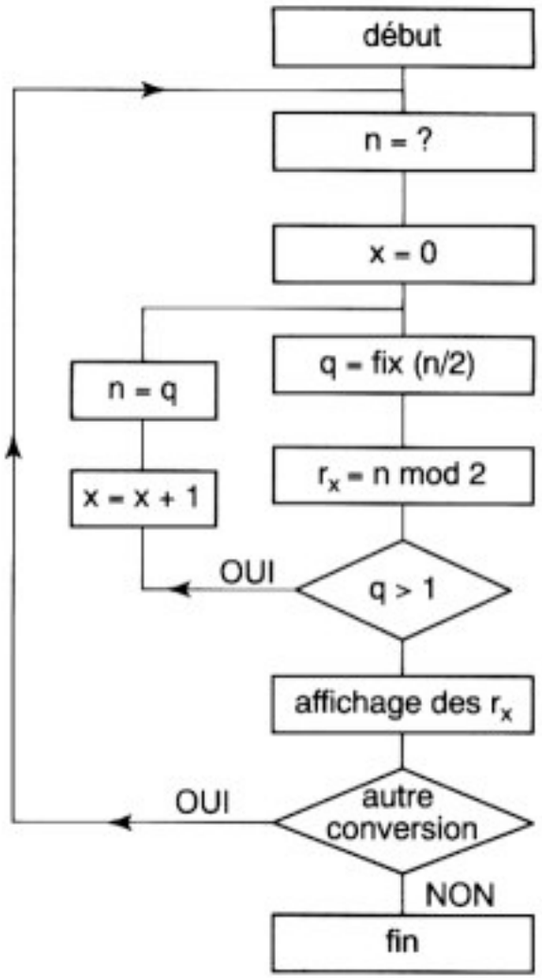
Fin

Comment écrire plus vite des programmes qui fonctionnent mieux ?

Programmer c'est ...



décrire une tâche

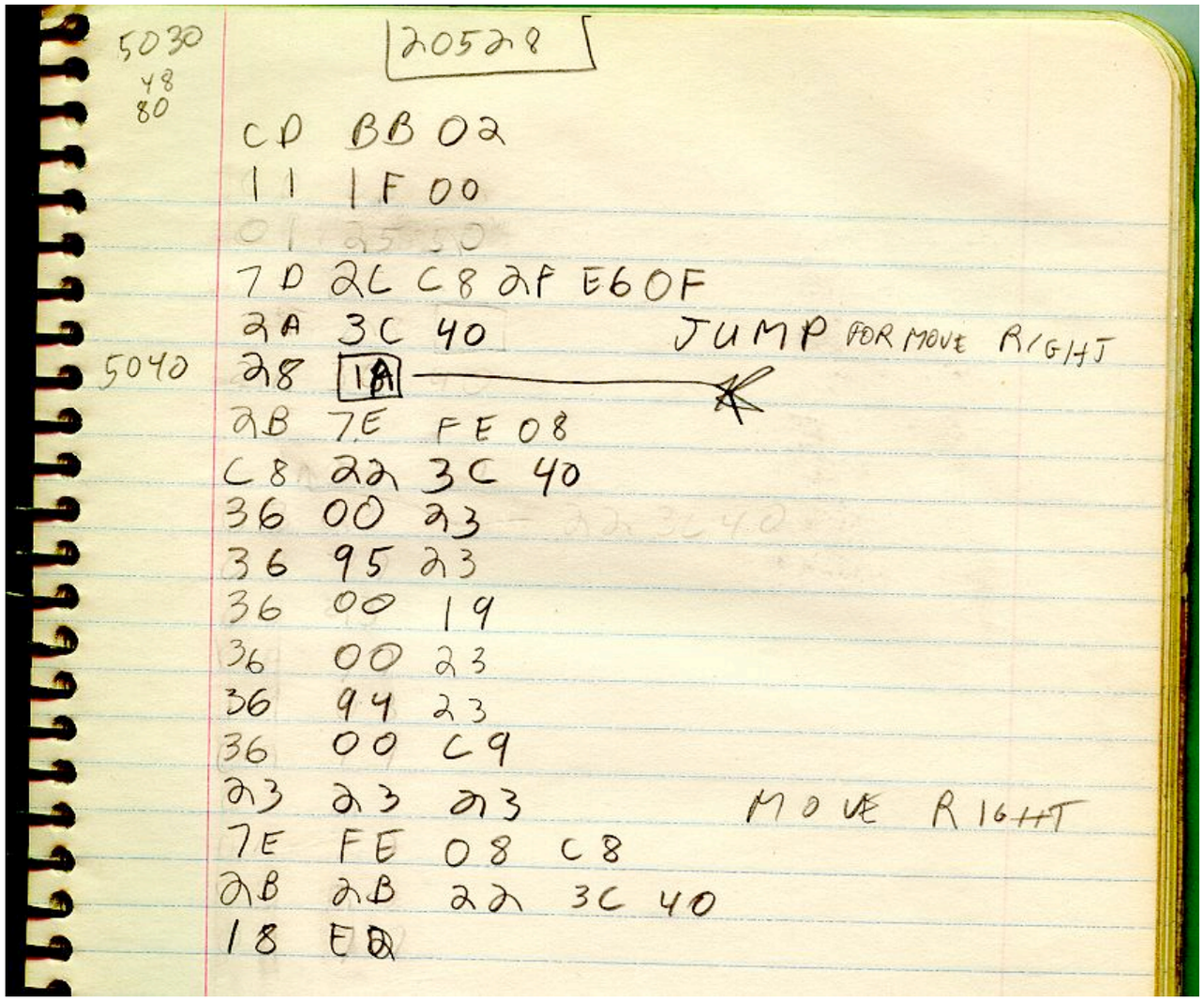


de manière suffisamment précise

pour pouvoir l'exécuter sur un matériel donné

Approche directe

- langage machine
- utilisation d'un assembleur



Utilisation de langages de plus haut niveau

- tableaux
- boucles
- sous-routines
- entrées-sorties

```
5 REM BREAKOUT GAME - BY DAN PINAL
10 DIM C(3),SCORE(3)
20 C(0)=-0.5:C(1)=-1:C(2)=1:C(3)=0.5
30 GRAPHICS 5:POKE 752,1
40 FOR L1=0 TO 3:COLOR L1
50 FOR L2=0 TO 2 STEP 2
...
340 GOSUB 1100
350 IF BX<1 OR BX>78 THEN DX=-DX:GOSUB 1000
360 IF BY=0 THEN DY=1:GOSUB 1000
...
1000 FOR L1=15 TO 0 STEP -1
1010 SOUND 0,30,10,L1
1020 NEXT L1:RETURN
```

1. *Introduction*
2. Îlots formels - Tom
3. Certification
4. Stratégies
5. Anti-patterns

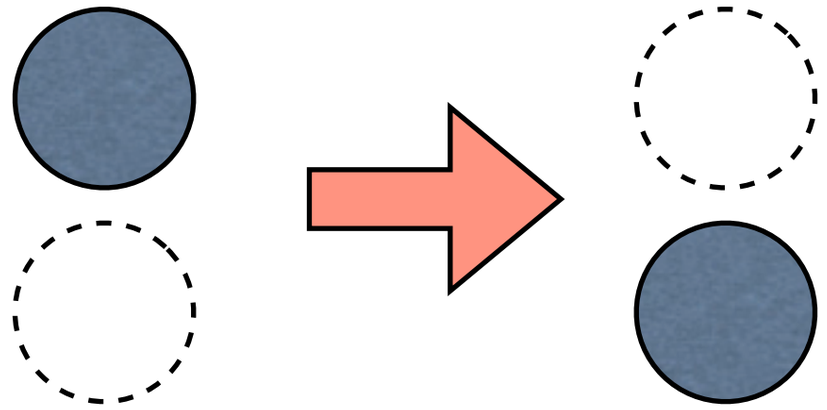
1. 2. 3. 4. 5.

Comment réduire les coûts de développement ?

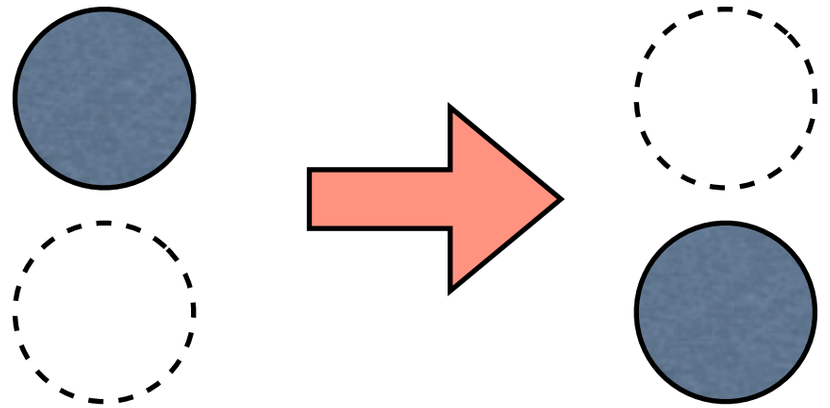
Travailler plus

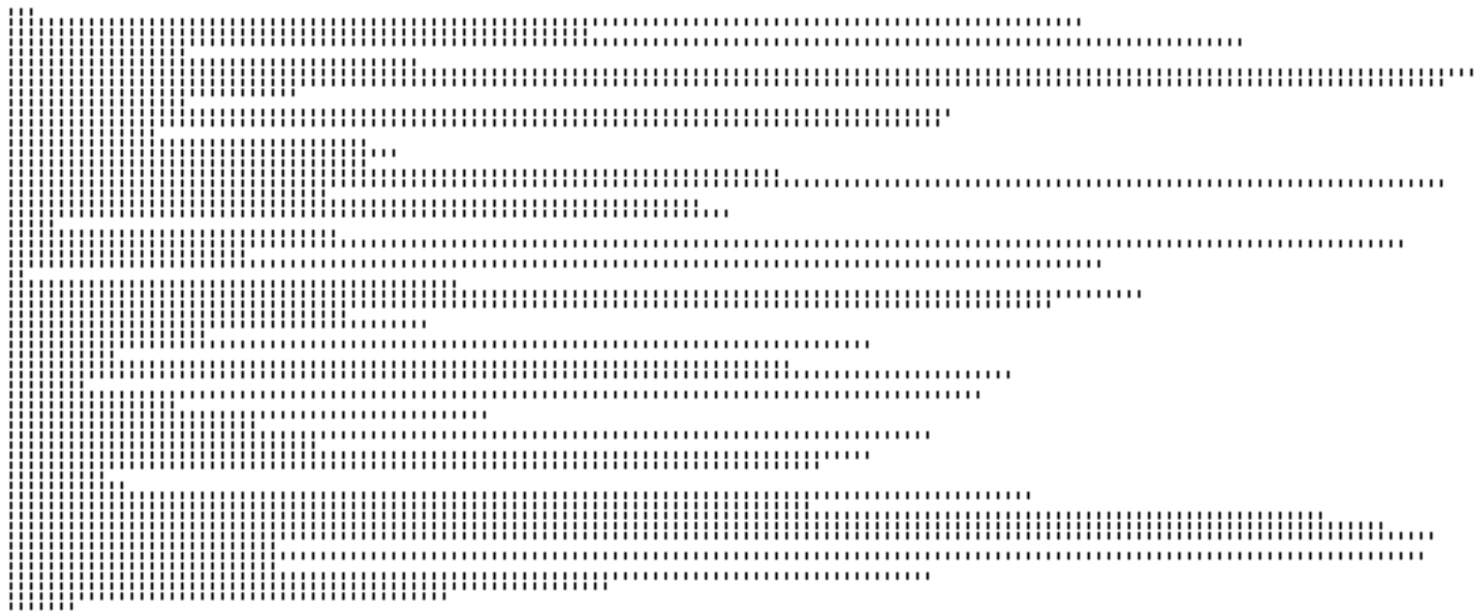
- Chine, Inde, ...
- utiliser de meilleurs outils

Parmi les bons outils



Parmi les bons outils





Réécriture

Réécriture

- concept fondamental permettant de modéliser des calculs
- modèle universel intégrant la notion de structure (termes, graphes,...)

Réécriture

- concept fondamental permettant de modéliser des calculs
- modèle universel intégrant la notion de structure (termes, graphes,...)
- introduit en 1970 [Knuth et Bendix]
- école nancéienne depuis 1980

  J.-P. Jouannaud, P. Lescanne, C. Kirchner, H. Kirchner, J.-L. Remy, M. Rusinowitch, I. Gnaedig, F. Klay, E. Domenjoud, C. Ringeissen, H. Cirstea, F. Blanqui, Y. Guiraud, ...

Concept principal

a → b : *règle* décrivant une transformation

Réécriture

- expressif
- exécutable
- formel

Expressif

x,y → y,x si y<x

algorithme de tri

Exécutable

Exécutable

(x,y → y,x si y<x) 1,2,5,3,4

Exécutable

(x,y → y,x si y<x) 1,2,5,3,4

➠ 1,2,3,5,4

Exécutable

(x,y → y,x si y<x) 1,2,5,3,4

➠ 1,2,3,5,4

(x,y → y,x si y<x) 1,2,3,5,4

Exécutable

(x,y → y,x si y<x) 1,2,5,3,4

➠ 1,2,3,5,4

(x,y → y,x si y<x) 1,2,3,5,4

➠ 1,2,3,4,5

Exécutable

(x,y → y,x si y<x) 1,2,5,3,4

➠ 1,2,3,5,4

(x,y → y,x si y<x) 1,2,3,5,4

➠ 1,2,3,4,5

opération fondamentale : C_1,x,y,C_2 << 1,2,3,4,5

$\{\sigma \mid \sigma(C_1,x,y,C_2) = 1,2,3,4,5 \}$

Permet de raisonner

- terminaison
- confluence
- complexité
- combinaison

Réécriture

à la base de nombreux langages

- OBJ[1976, 1985]
- *ELAN* [1993, 1999]
- Maude [1996]
- ASF+SDF[1985]
- Stratego[1998], DMS[1998], Hats[2003], ...

Applications principales

- formes canoniques
- calculs symboliques
- outils de preuve
- transformations de programmes
- modéliser

1. **2.** 3. 4. 5.

Comment rendre utilisables les concepts liés à la réécriture ?

Situation actuelle

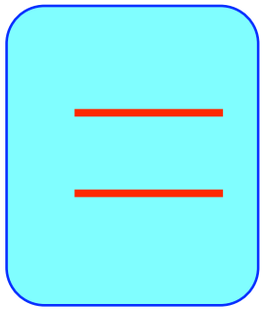
Situation actuelle

- grands acteurs dominent le marché (Microsoft, Sun, Oracle, Google,...)
- environnements incontournables (bibliothèques, éditeurs)

Situation actuelle

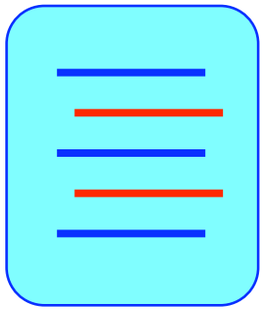
- grands acteurs dominent le marché (Microsoft, Sun, Oracle, Google,...)
- environnements incontournables (bibliothèques, éditeurs)
- ingénieurs formés
- les programmes existent
- *changer d'environnement* **n'est pas rentable**

Création d'un langage



Création d'un langage

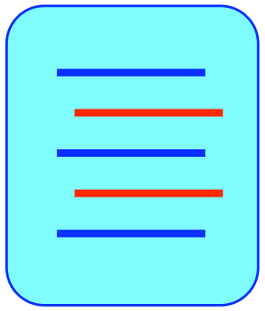
int



Création d'un langage

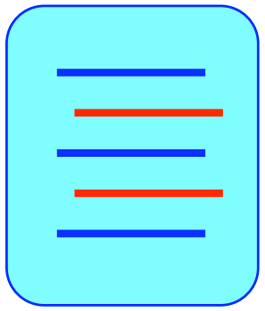
int

String



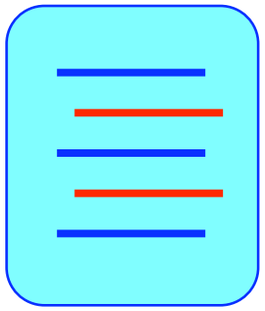
Création d'un langage

int String I/O

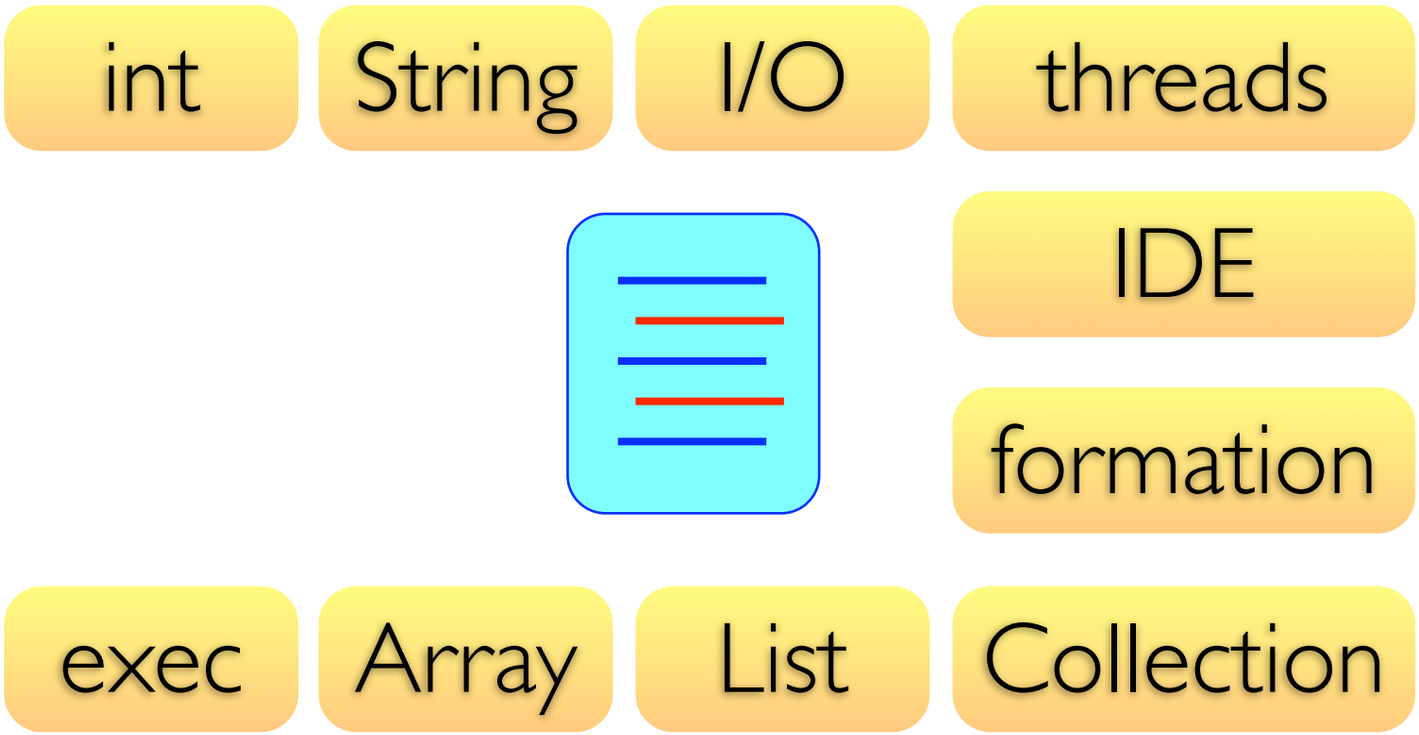


Création d'un langage

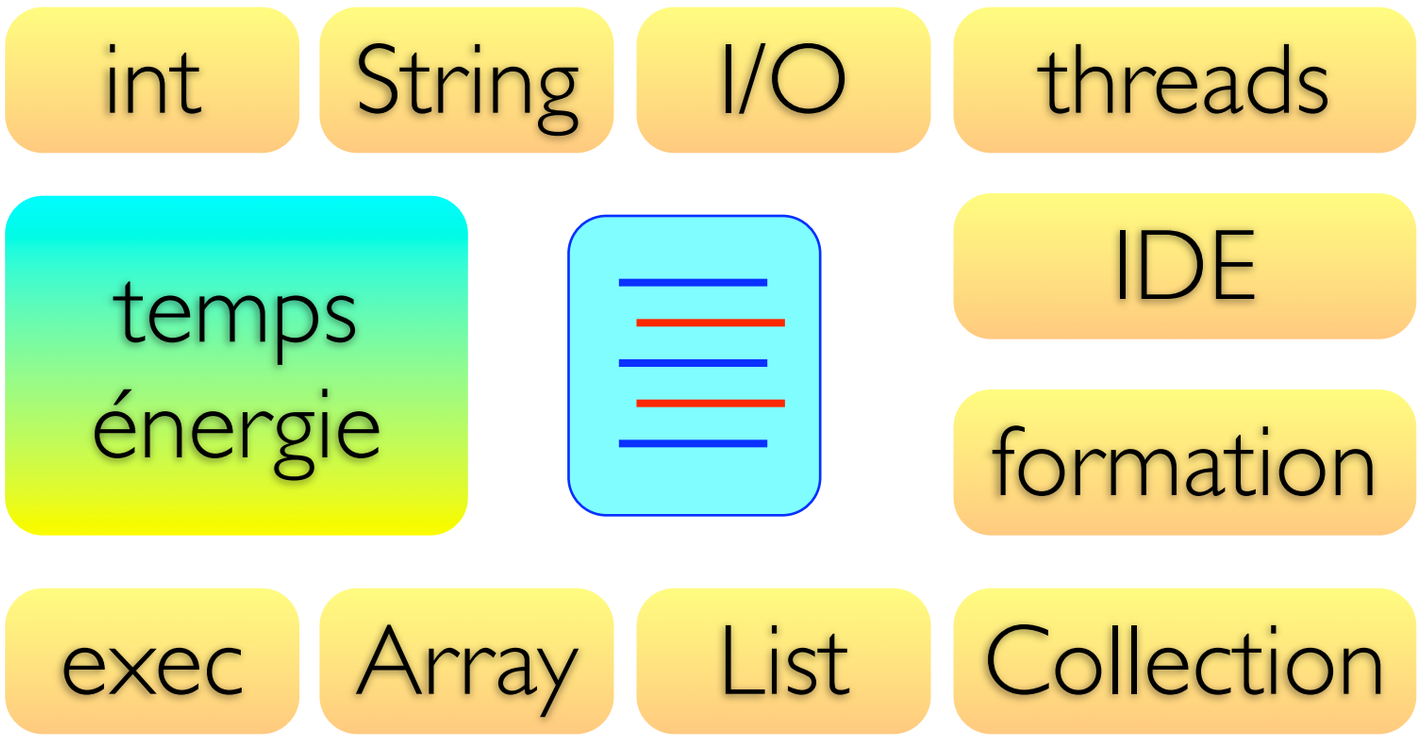
int String I/O threads



Création d'un langage

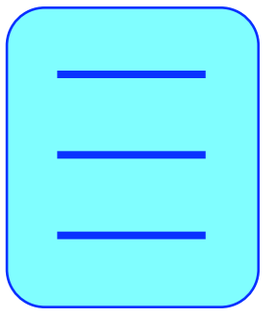


Création d'un langage



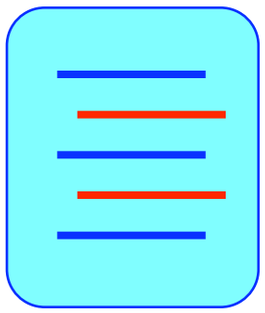
- *ELAN*, **Maude, ASF+SDF, Stratego, ...**
- **Caml, Clean, Haskell, Scala, ...**

A piggyback ride



hôte

A piggyback ride

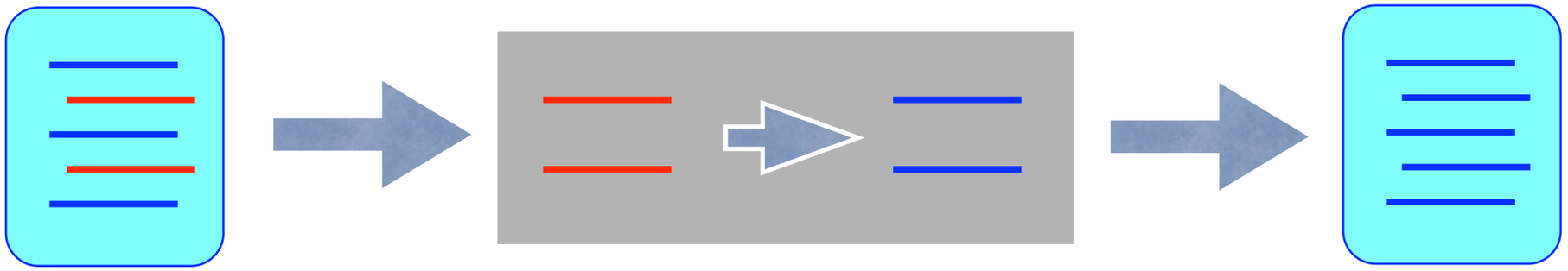


hôte

A piggyback ride

hôte hôte

A piggyback ride



hôte dissolution hôte

Îlot Formel

Petit espace isolé dans un ensemble d'une autre nature.
Dont la précision et la netteté excluent toute méprise.

De nombreuses questions

- quels *îlots* (constructions) ajouter ?
- quels *océans* (langages hôtes) considérer ?
- comment relier les structures de données ?
- est-ce réalisable ?

Je voulais

- pouvoir simuler tout type de réécriture
- être indépendant des structures de données
- une sémantique claire

Quels îlots ?

lhs → **rhs** if **condition** where **v:=expr**

Quels îlots ?

lhs → **rhs** if **condition** where **v:=expr**

- pas assez expressif (I/O, non-déterminisme)
- difficile (impossible) d'assurer que les règles sont toujours appliquées

Quels îlots ?

```
match(s) {
  lhs → { action }
  ...
  lhs → { return ⌈ rhs ⌉ ; }
}
```

Insertion triée

Insertion triée

insert(e,cons(head,tail)) → cons(head,insert(e,tail)) if head<e
insert(e,cons(head,tail)) → cons(e,cons(head,tail)) if head≥e
insert(e,nil) → cons(e,nil)

Insertion triée

```
List insert(Element e, List l) {
  match(l) {
    cons(head,tail) → {
      if(head<e) return ⌈ cons(head,insert(e,tail)) ⌉ ;
    }
  }
  return ⌈ cons(e,l) ⌉ ;
}
```

insert(e,cons(head,tail)) → cons(head,insert(e,tail)) if head<e
insert(e,cons(head,tail)) → cons(e,cons(head,tail)) if head≥e
insert(e,nil) → cons(e,nil)

Quels océans ?

- la plupart des langages impératifs
- en particulier C, Java, C#
- et pourquoi pas Caml, ...

Comment relier les structures de données ?

Comment relier les structures de données ?

- *ancrage* : voir les objets comme des termes
- correspondance : sorte algébrique ↔ type

Comment relier les structures de données ?

- *ancrage* : voir les objets comme des termes
- correspondance : sorte algébrique ↔ type
- accéder au symbole de tête (is_fsym)

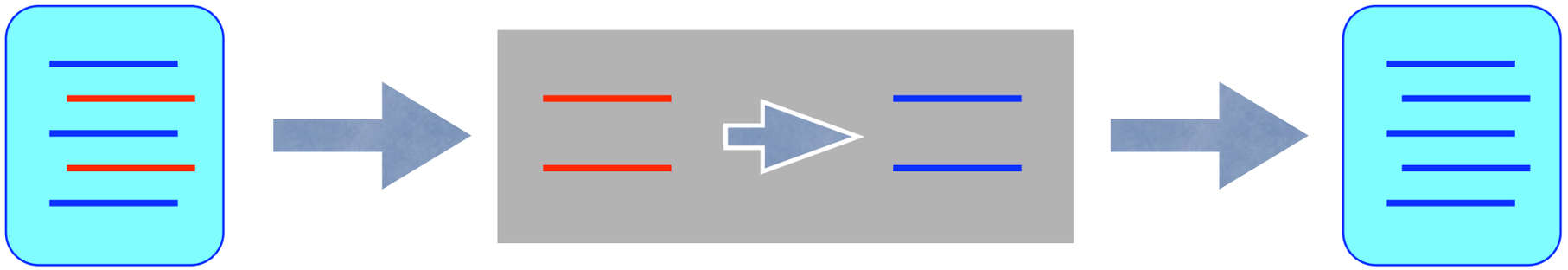
Comment relier les structures de données ?

- *ancrage* : voir les objets comme des termes
- correspondance : sorte algébrique ↔ type
- accéder au symbole de tête (is_fsym)
- accéder à un sous-terme (subterm)

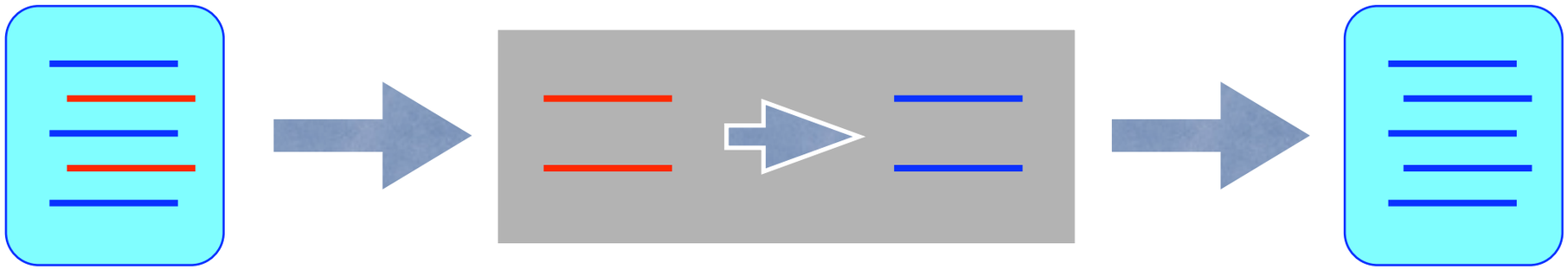
Comment relier les structures de données ?

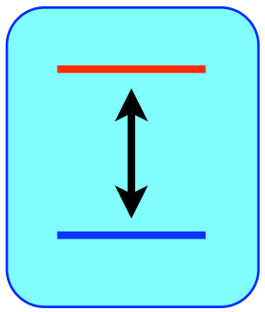
- *ancrage* : voir les objets comme des termes
- correspondance : sorte algébrique ↔ type
- accéder au symbole de tête (is_fsym)
- accéder à un sous-terme (subterm)
- allouer un constructeur (make)

Est-ce réalisable ?



Est-ce réalisable ?

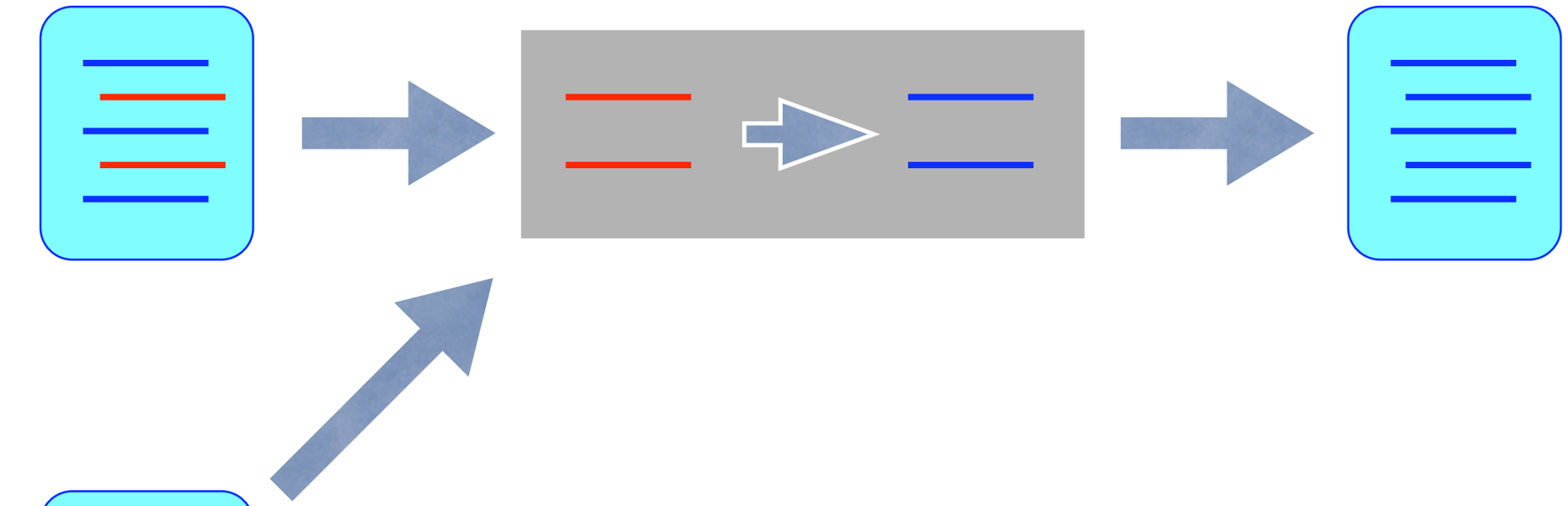


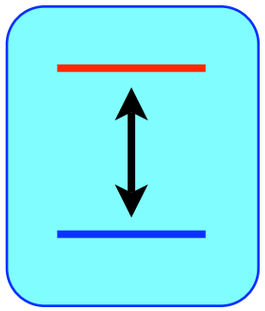


ancrage

is_fsym(s,f), subterm$_f$, make$_f$

Est-ce réalisable ?

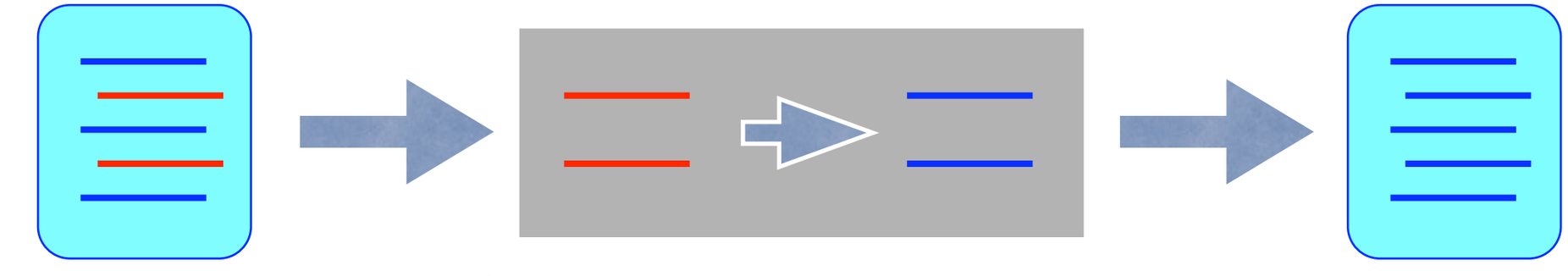




ancrage

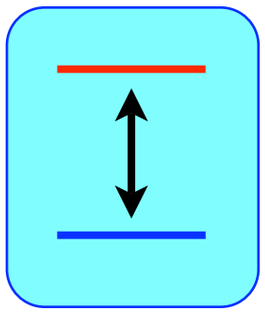
is_fsym(s,f), subterm_f, make_f

Est-ce réalisable ?



if, let, while

C, Java, Caml,...



ancrage

is_fsym(s,f), subterm$_f$, make$_f$

difficultés :
- identifier le bon formalisme d'ancrage
- compiler avec un nombre réduit d'instructions

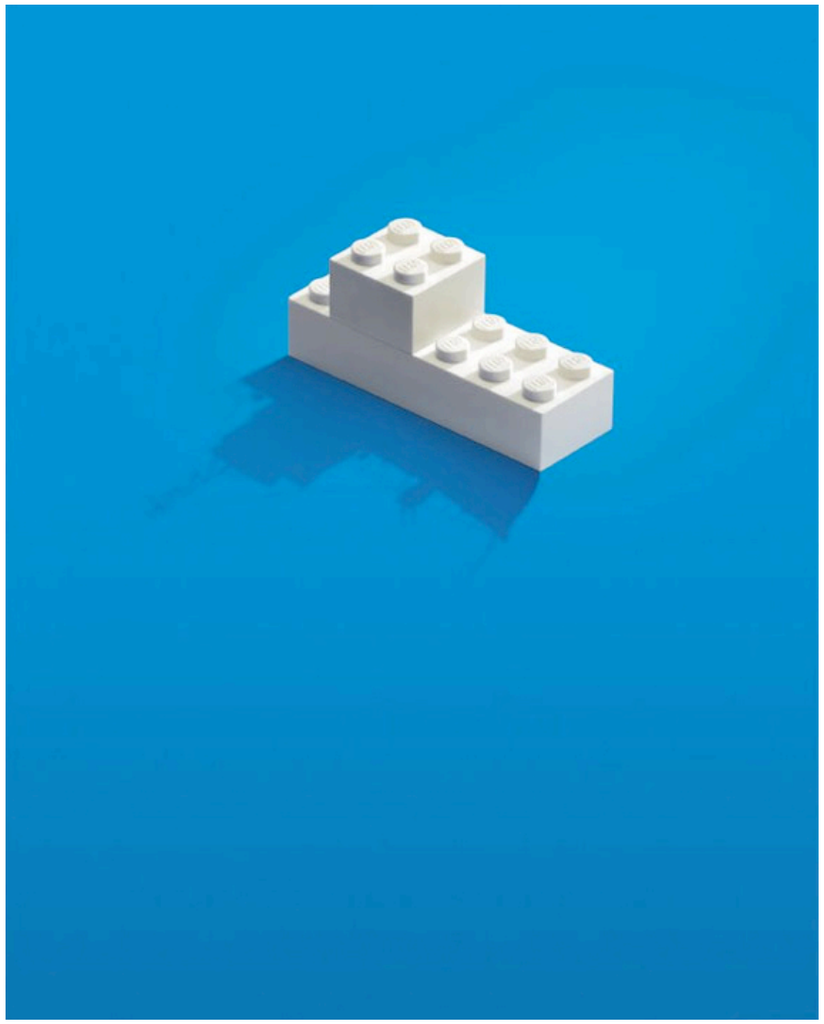
Tom [2001]

Un moyen de rendre utilisable la réécriture

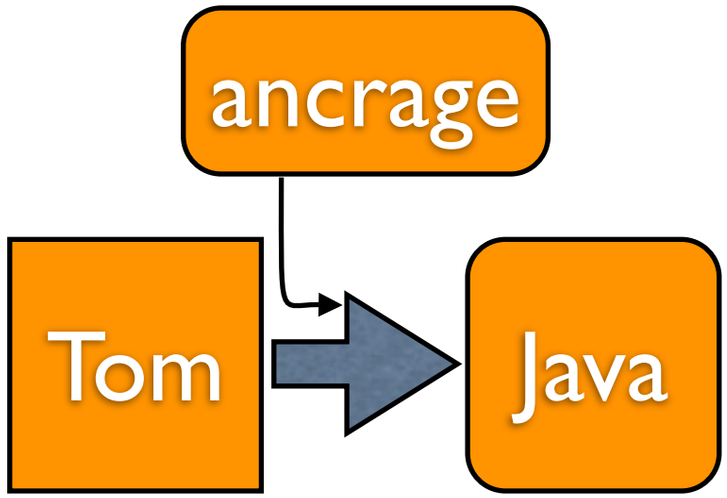
M. Vittek

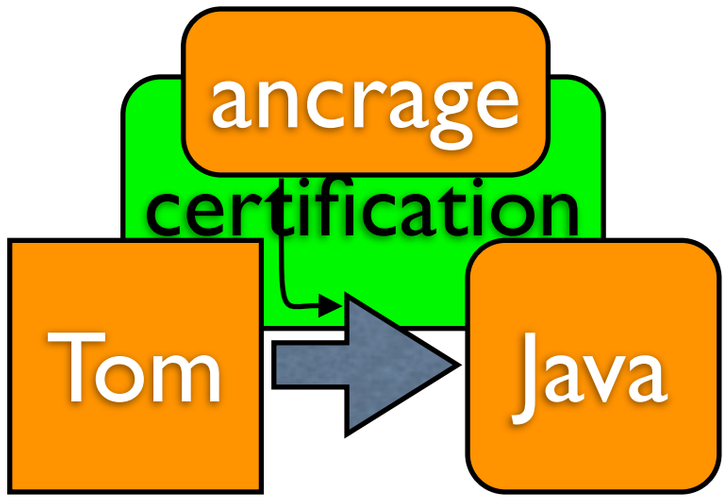
C. Ringeissen

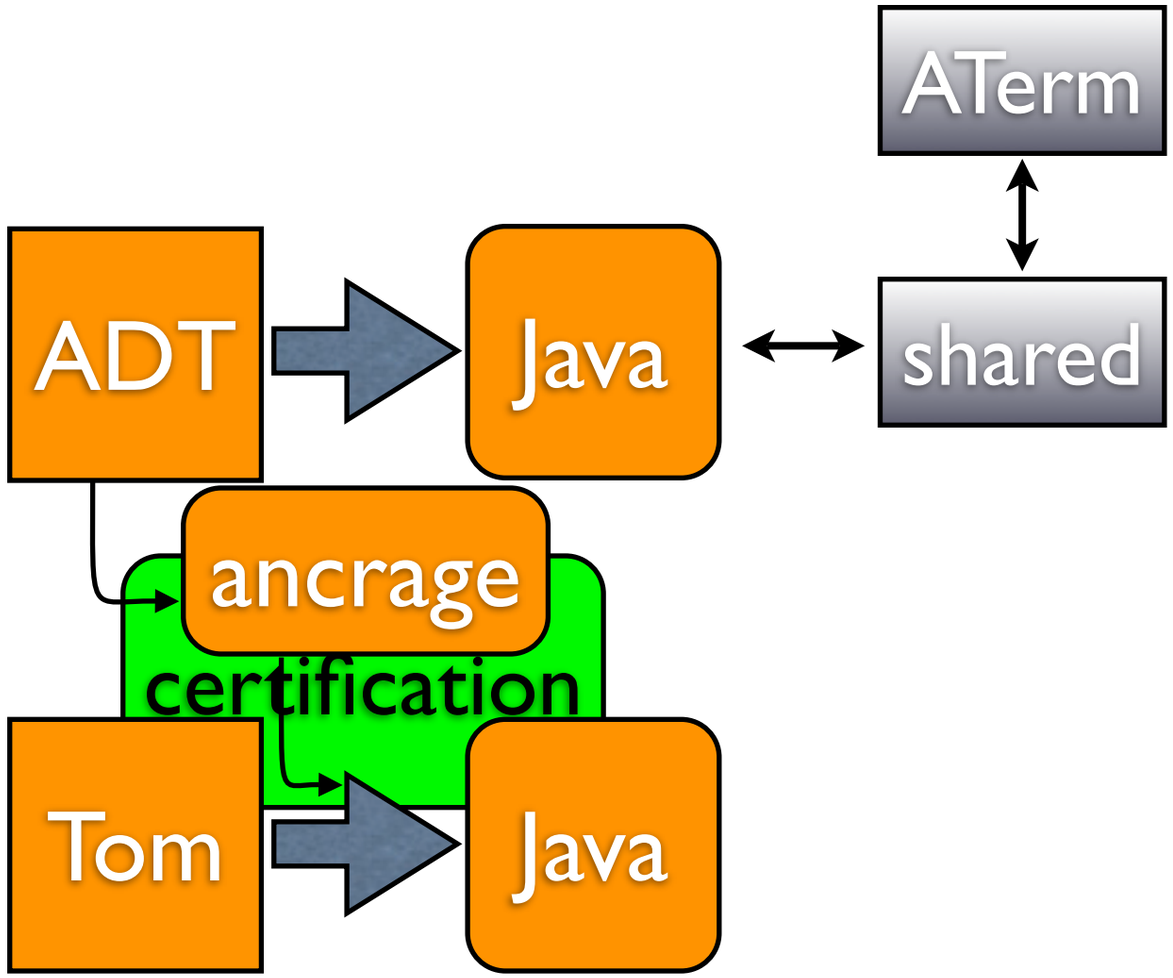
Roadmap

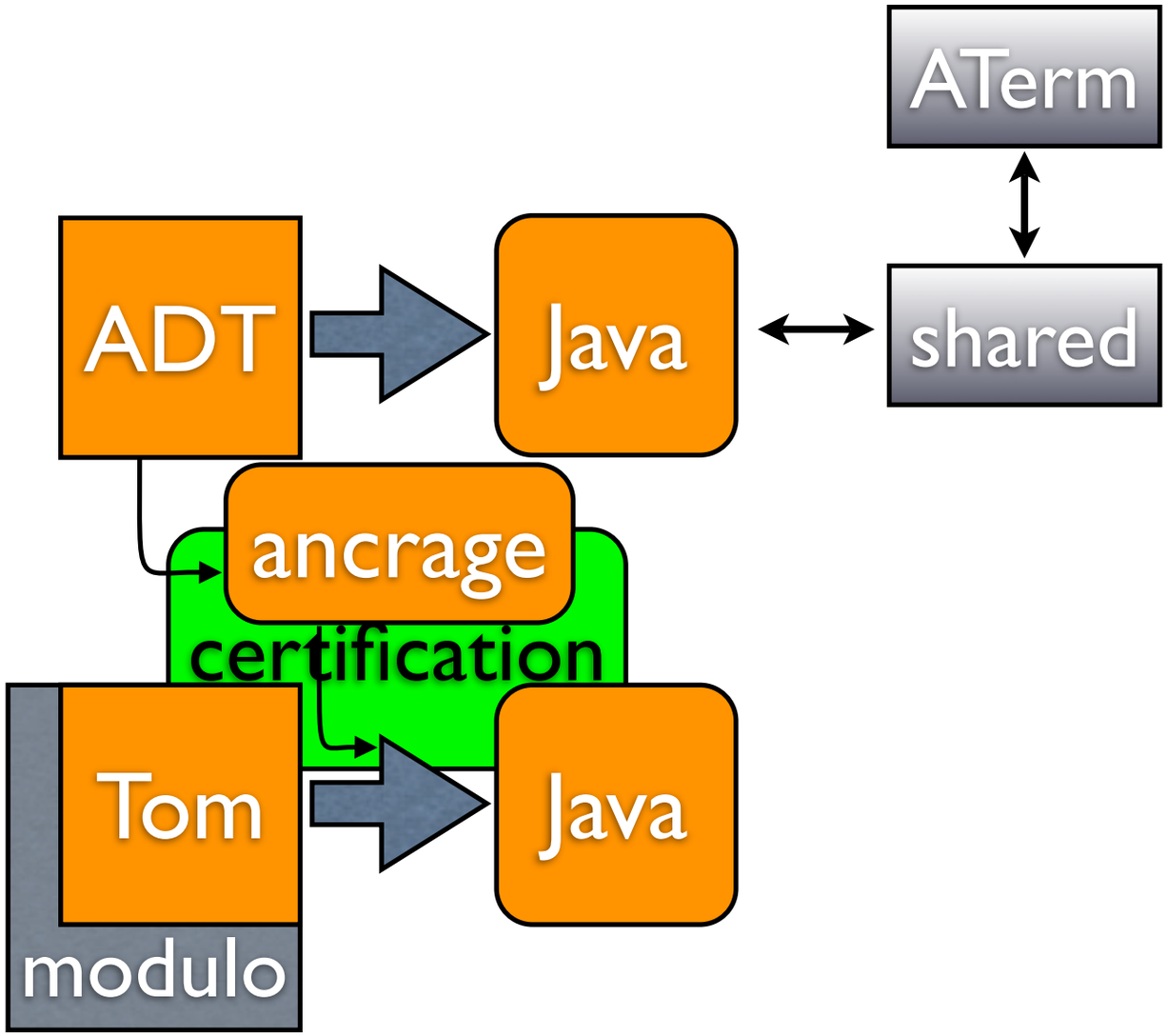


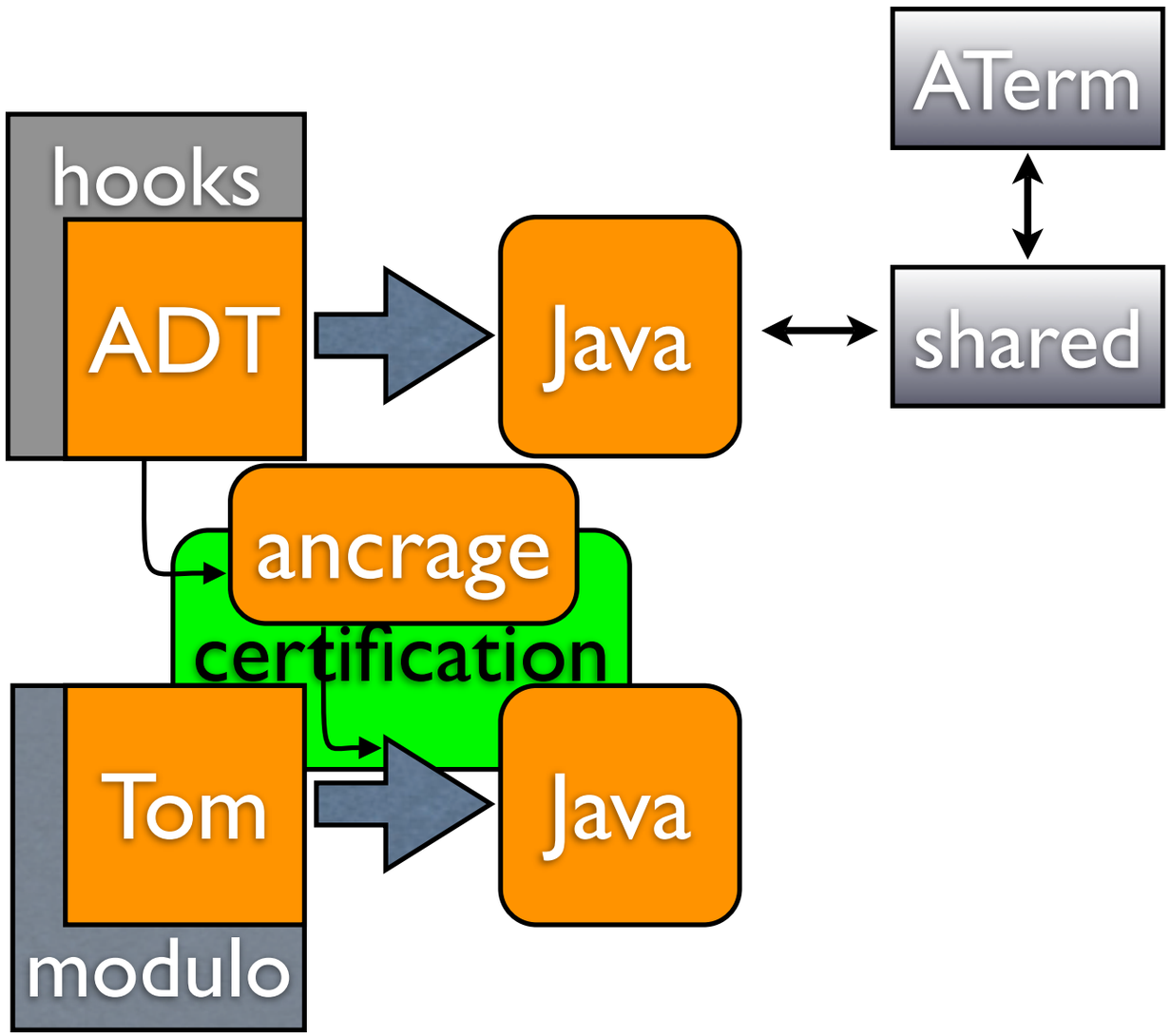
- plus d'expressivité : modulo, anti-patterns
- augmenter la confiance : comprendre, certifier, exprimer des invariants
- contrôler les règles : stratégies
- simplifier l'utilisation : types algébriques, IDE

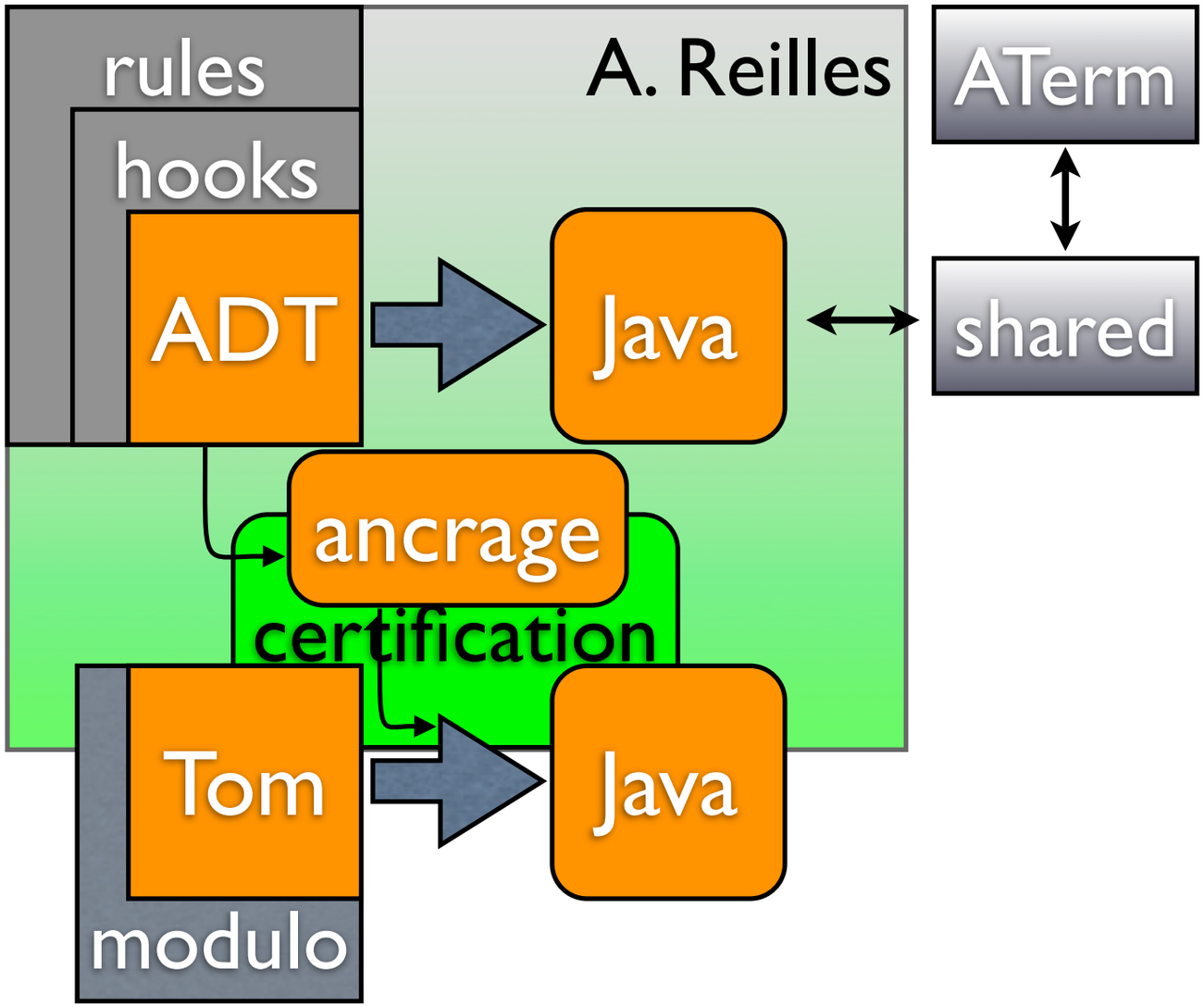

ancrage
Tom
Java

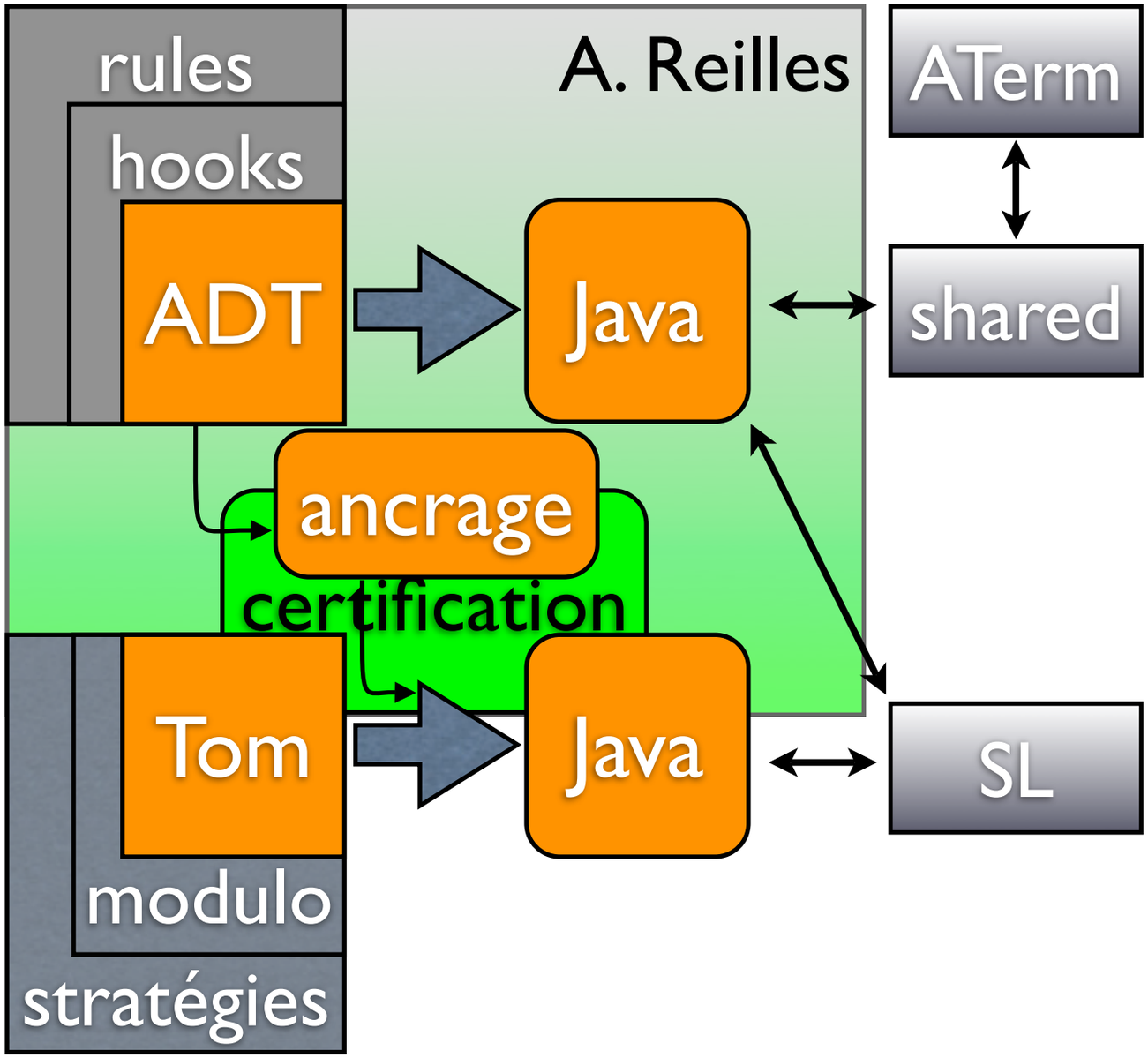

ancrage
certification
Tom
Java

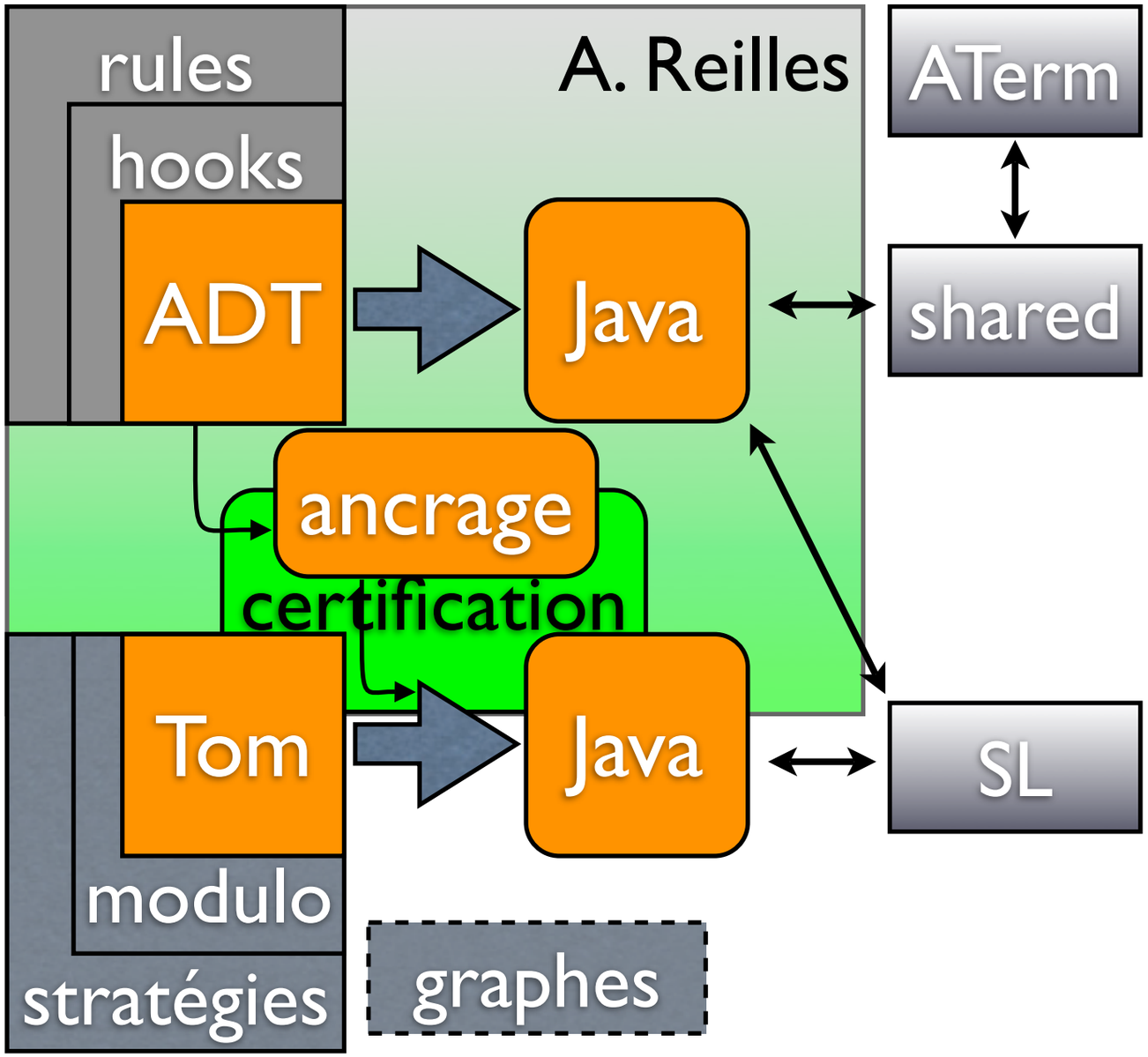

ATerm
ADT
Java
shared
ancrage
certification
Tom
Java

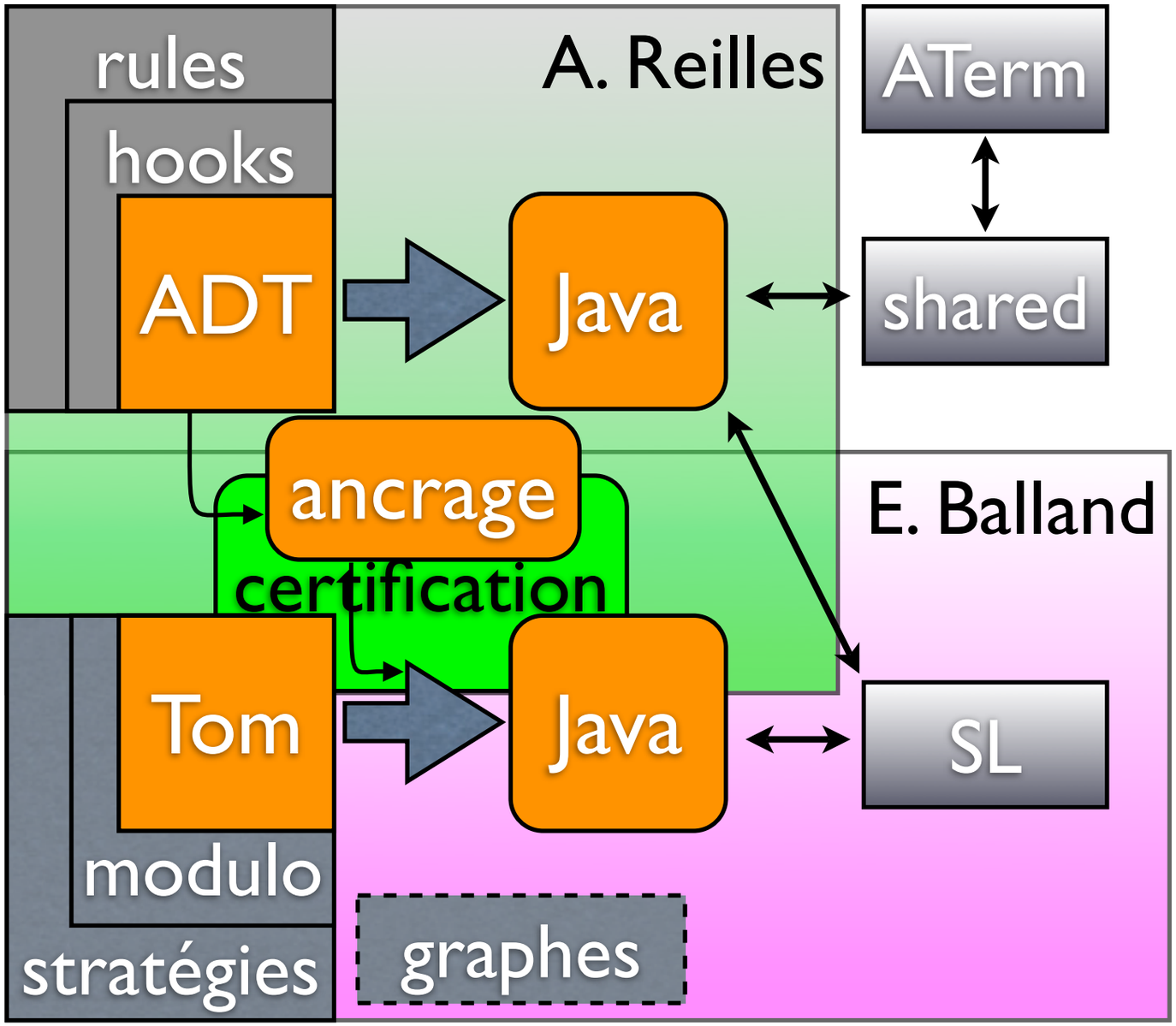

ATerm
ADT
Java
shared
ancrage
certification
Tom
Java
modulo

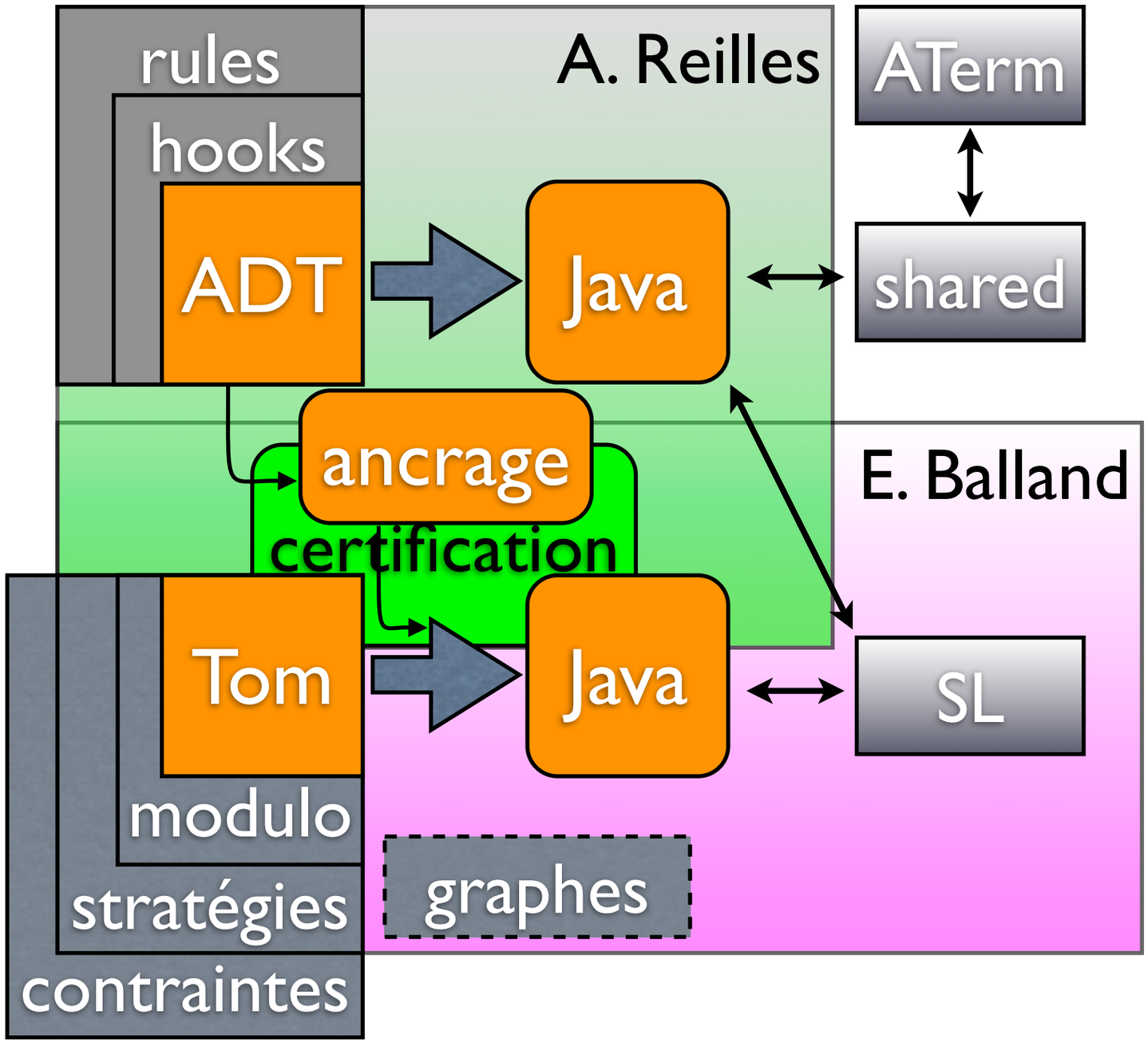

ATerm
hooks
ADT
Java
shared
ancrage
certification
Tom
Java
modulo

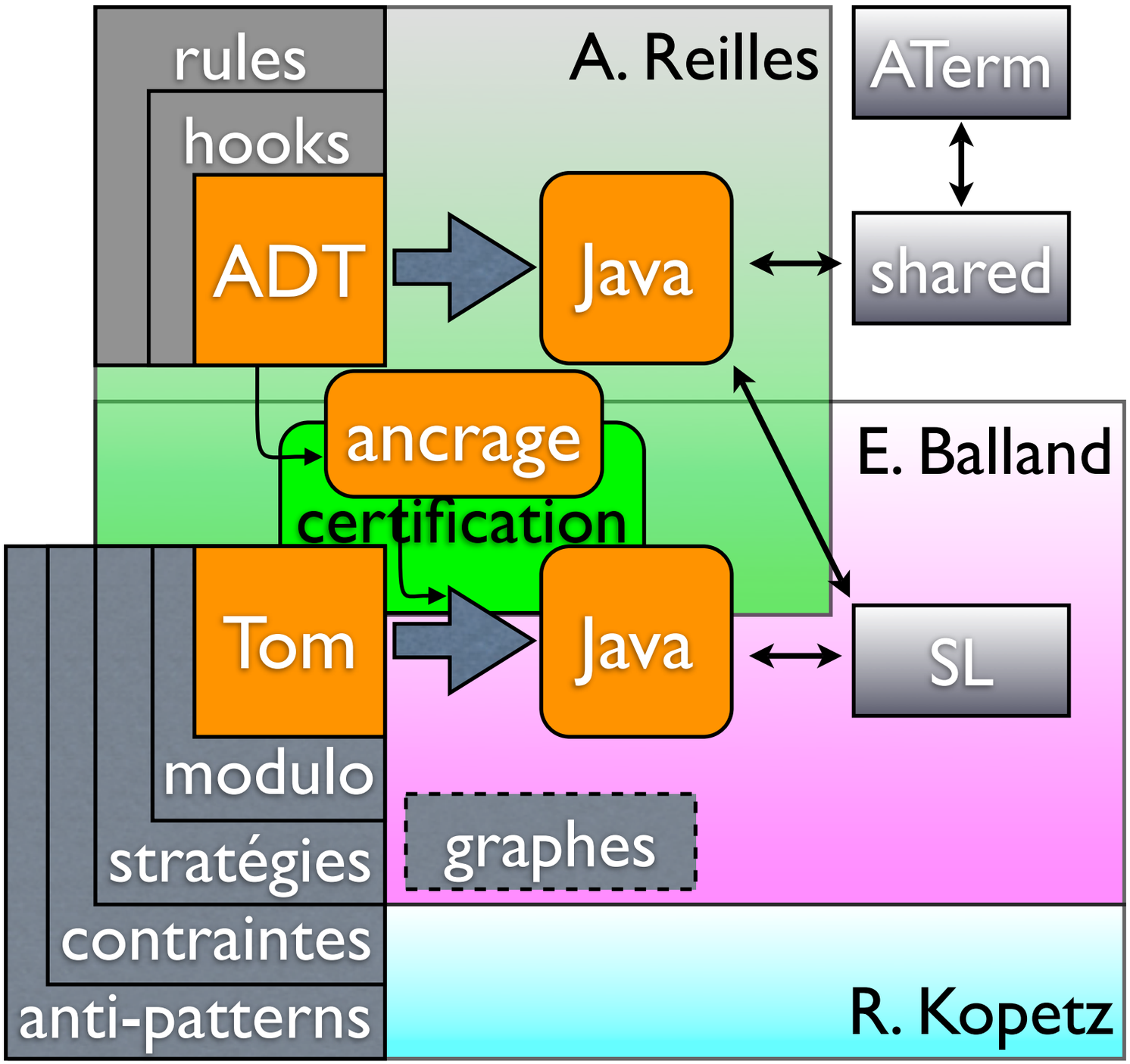

rules
hooks
ADT
A. Reilles
Java
ATerm
shared
ancrage
certification
Tom
Java
modulo


rules
hooks
ADT
A. Reilles
Java
ATerm
shared
ancrage
certification
Tom
Java
SL
modulo
stratégies

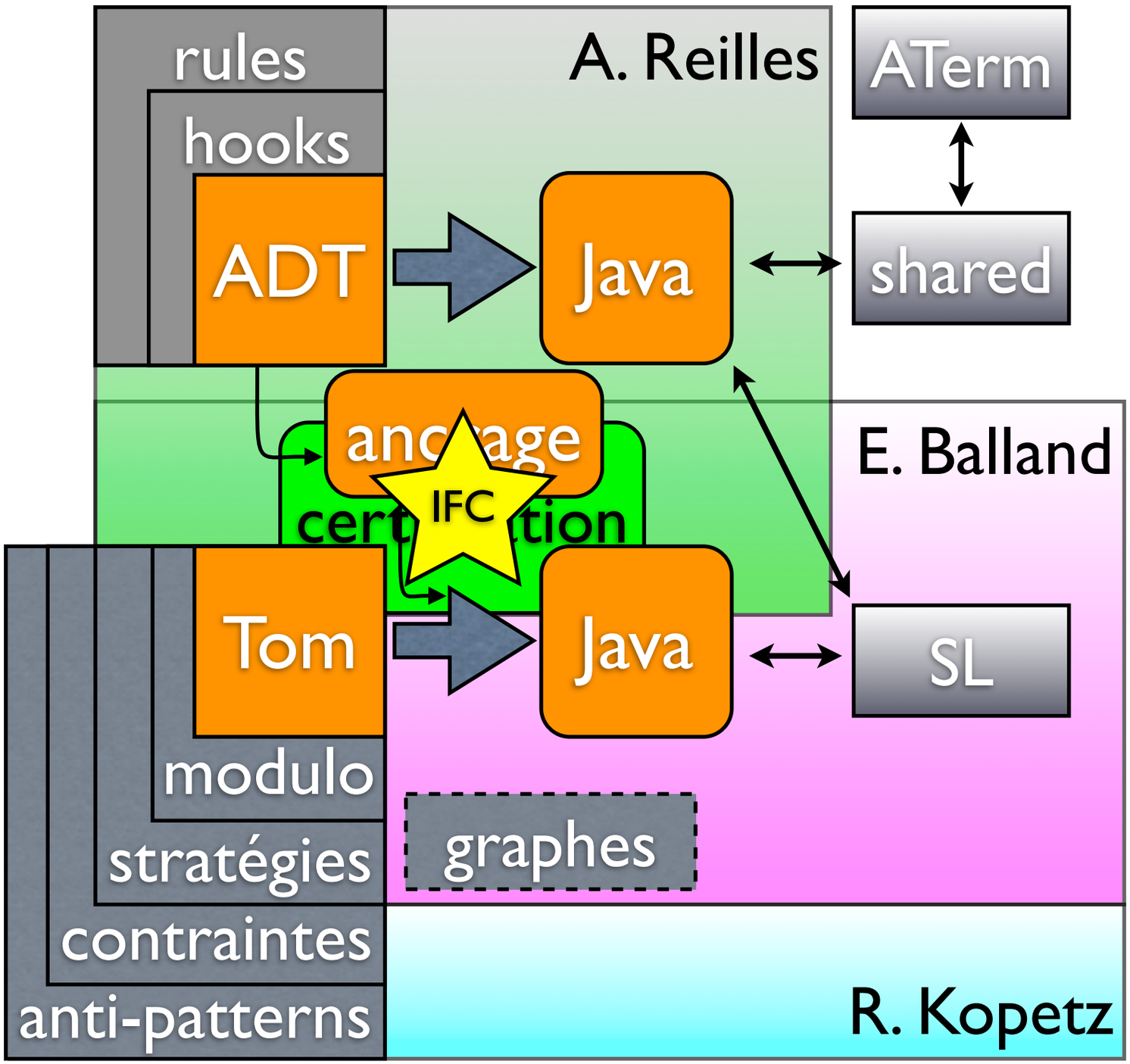

rules
hooks
ADT
A. Reilles
Java
ATerm
shared
ancrage
certification
Tom
Java
SL
modulo
stratégies
graphes


rules
hooks
ADT
A. Reilles
Java
ATerm
shared
ancrage
certification
E. Balland
Tom
Java
SL
modulo
stratégies
graphes


rules
hooks
ADT
A. Reilles
Java
ATerm
shared
ancrage
certification
E. Balland
Tom
Java
SL
modulo
stratégies
contraintes
graphes

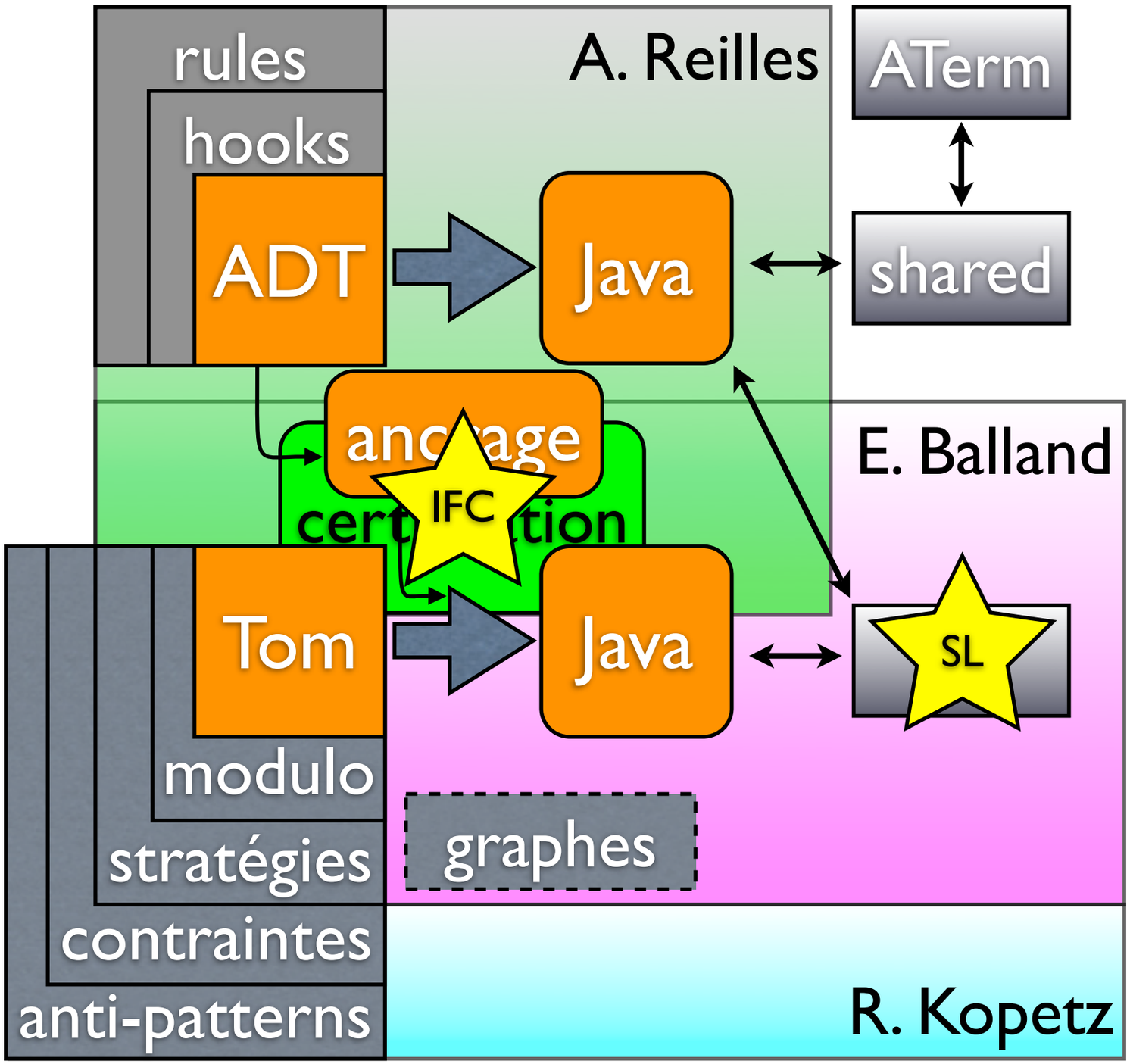


difficultés :
- extensions cohérentes
- sémantique claire
- système maintenable



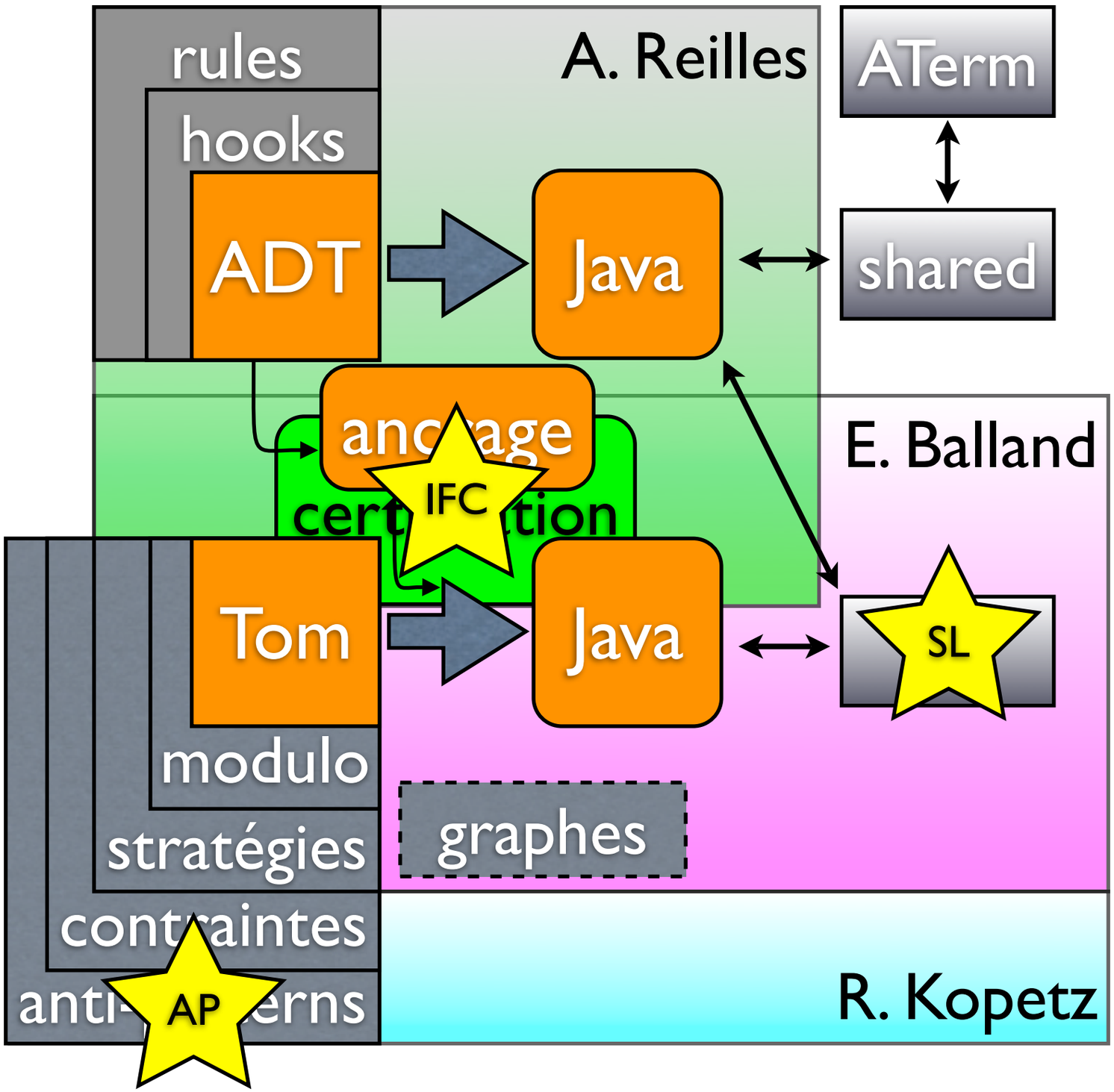
- certification

difficultés :
extensions cohérentes
sémantique claire
système maintenable



- certification
- stratégies

difficultés :
extensions cohérentes
sémantique claire
système maintenable



- certification
- stratégies
- anti-patterns

difficultés :
extensions cohérentes
sémantique claire
système maintenable

1. Introduction
2. Îlots formels - Tom
3. *Certification*
4. Stratégies
5. Anti-patterns

1. 2. **3.** 4. 5.

Peut-on avoir confiance dans les outils proposés ?

Éléments de confiance

Éléments de confiance

- équipe reconnue
- nombreux exemples
- jeux de tests

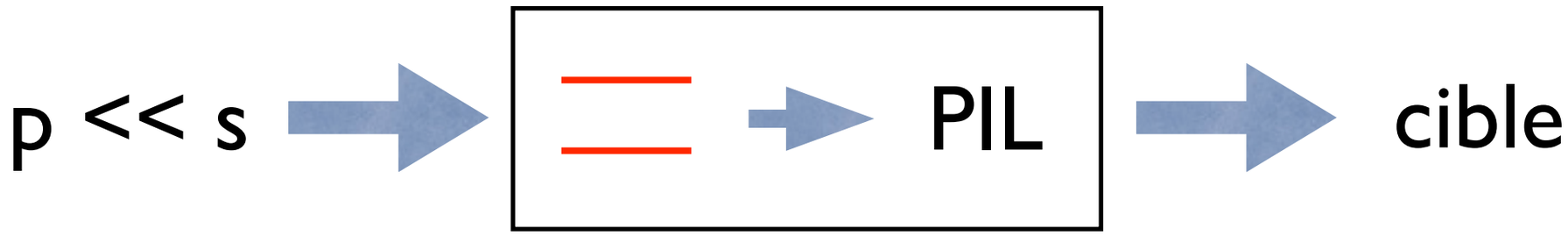
Éléments de confiance

- équipe reconnue
- nombreux exemples
- jeux de tests
- bootstrap du compilateur

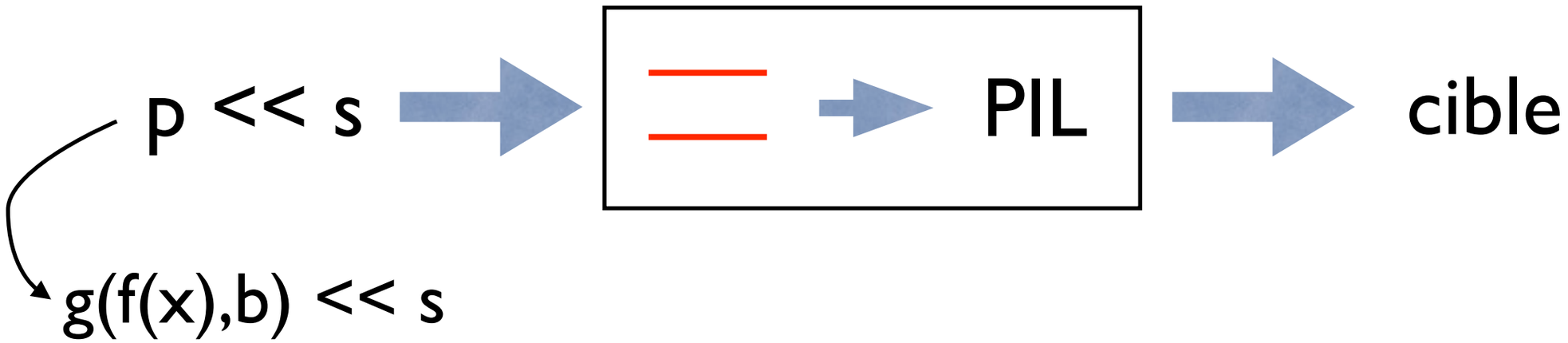
Éléments de confiance

- équipe reconnue
- nombreux exemples
- jeux de tests
- bootstrap du compilateur
- certification du compilateur

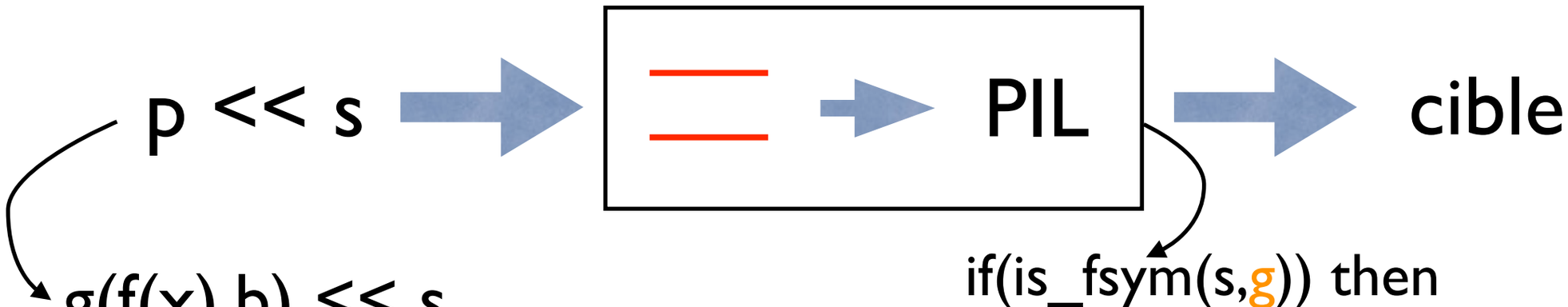
Certification



Certification

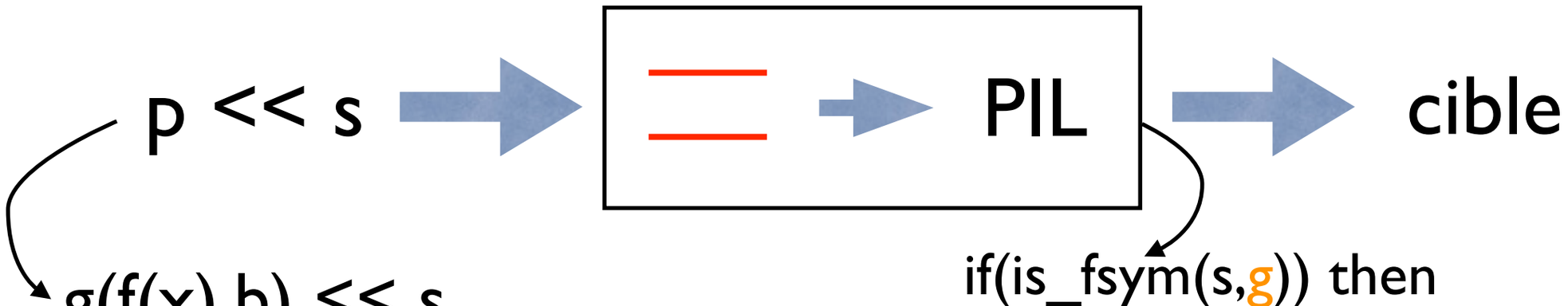


Certification




```
if(is_fsym(s,g)) then
  if(is_fsym(subterm_g(s,1),f) then
  let x = subterm_f(subterm_g(s,1),1) in
    if(is_fsym(subterm_g(s,2),b) then
      action(…)
    else echec
…
```

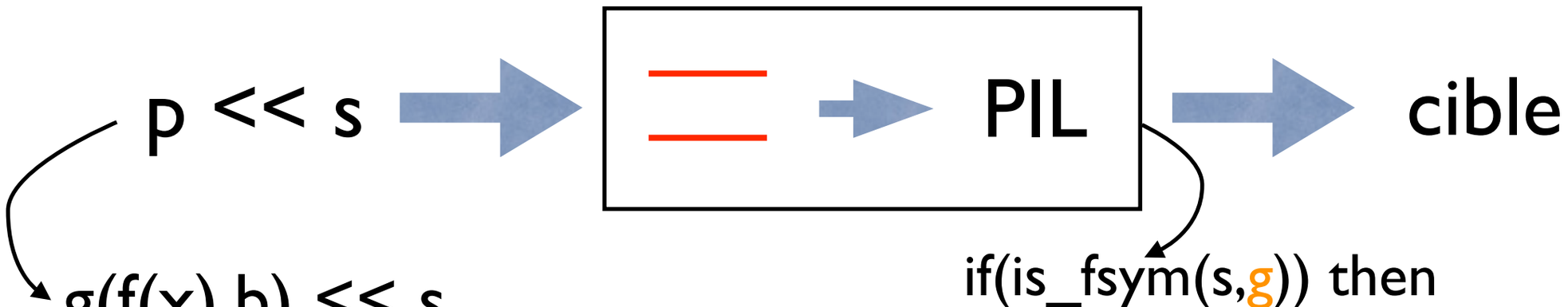
Certification




```
if(is_fsym(s,g)) then
  if(is_fsym(subterm_g(s,1),f) then
    let x = subterm_f(subterm_g(s,1),1) in
      if(is_fsym(subterm_g(s,2),b) then
        action(…)
      else echec
  …
```

```
is_fsym(s,g)
is_fsym(subterm_g(s,1),f)
x = subterm_f(subterm_g(s,1),1)
is_fsym(subterm_g(s,2),b)
```

Certification

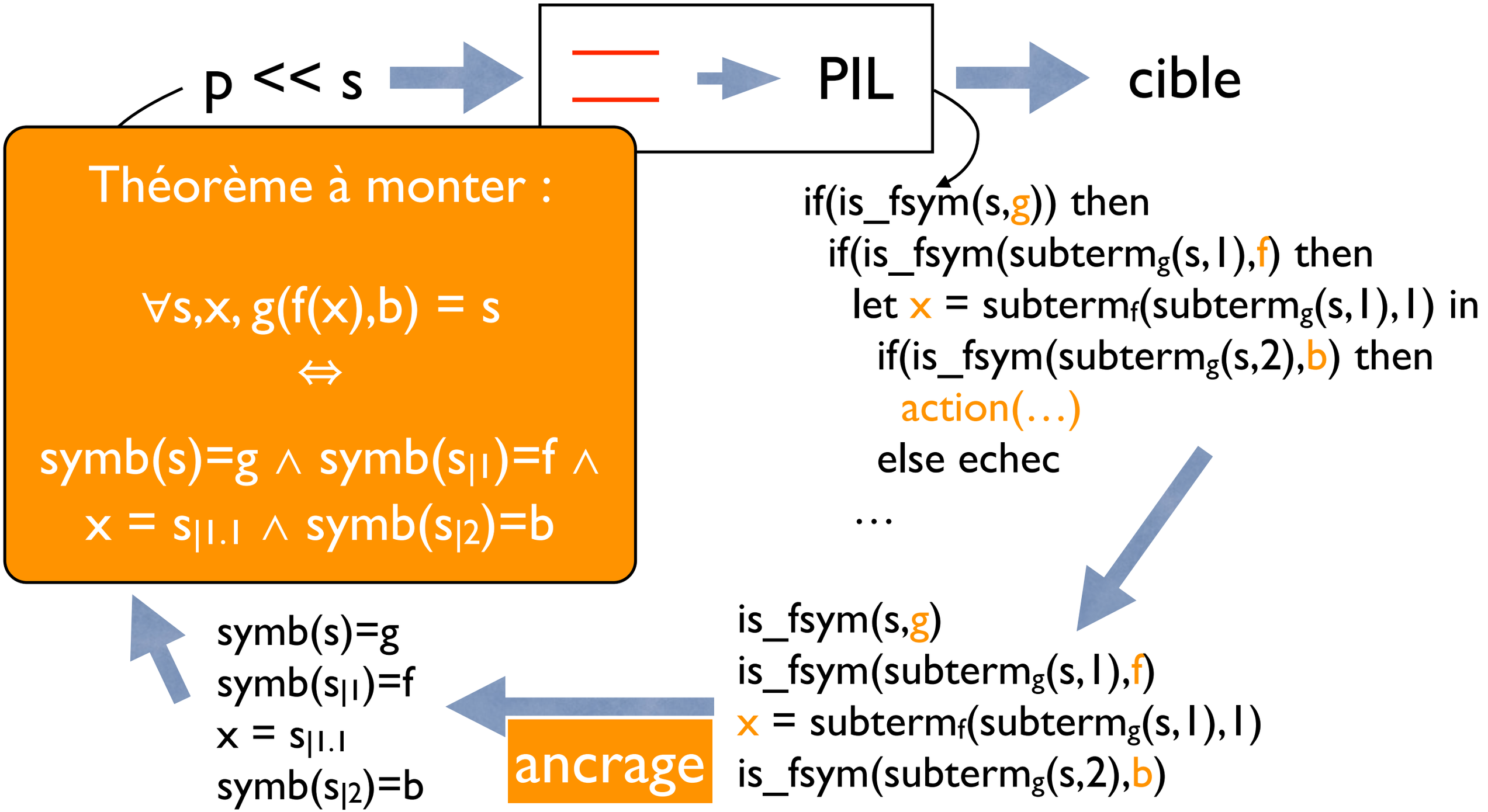



```
if(is_fsym(s,g)) then
  if(is_fsym(subterm_g(s,1),f) then
    let x = subterm_f(subterm_g(s,1),1) in
      if(is_fsym(subterm_g(s,2),b) then
        action(…)
      else echec
…
```

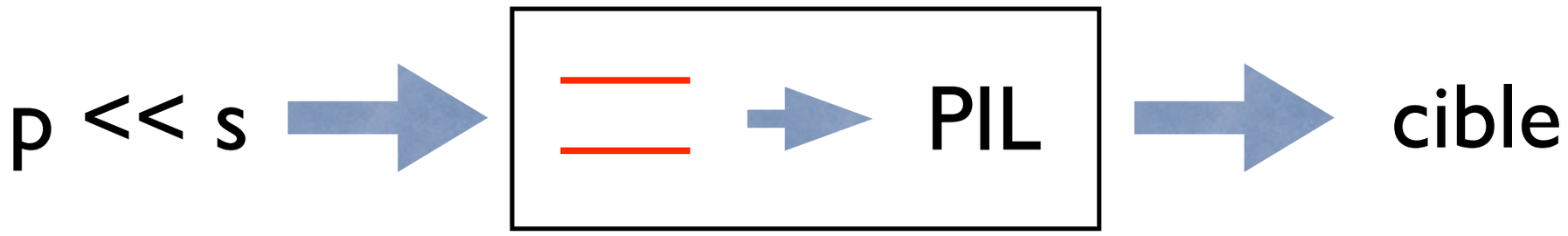

```
is_fsym(s,g)
is_fsym(subterm_g(s,1),f)
x = subterm_f(subterm_g(s,1),1)
is_fsym(subterm_g(s,2),b)
```

```
symb(s)=g
symb(s|1)=f
x = s|1.1
symb(s|2)=b
```

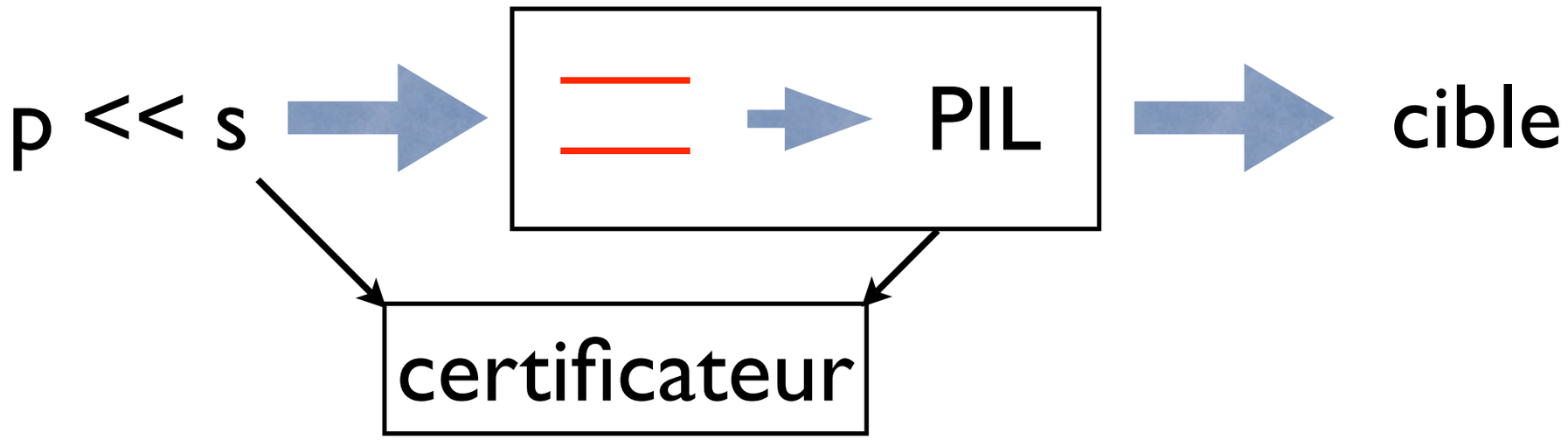
Certification



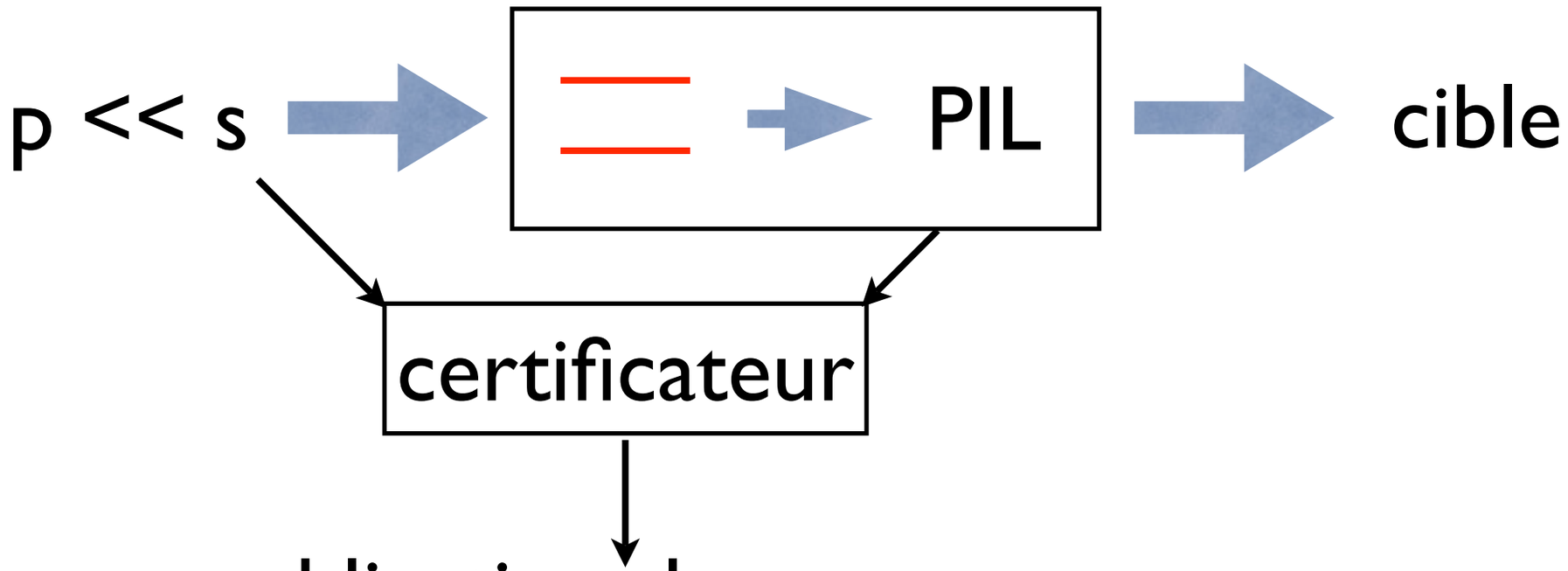
Mécanisme de certification



Mécanisme de certification



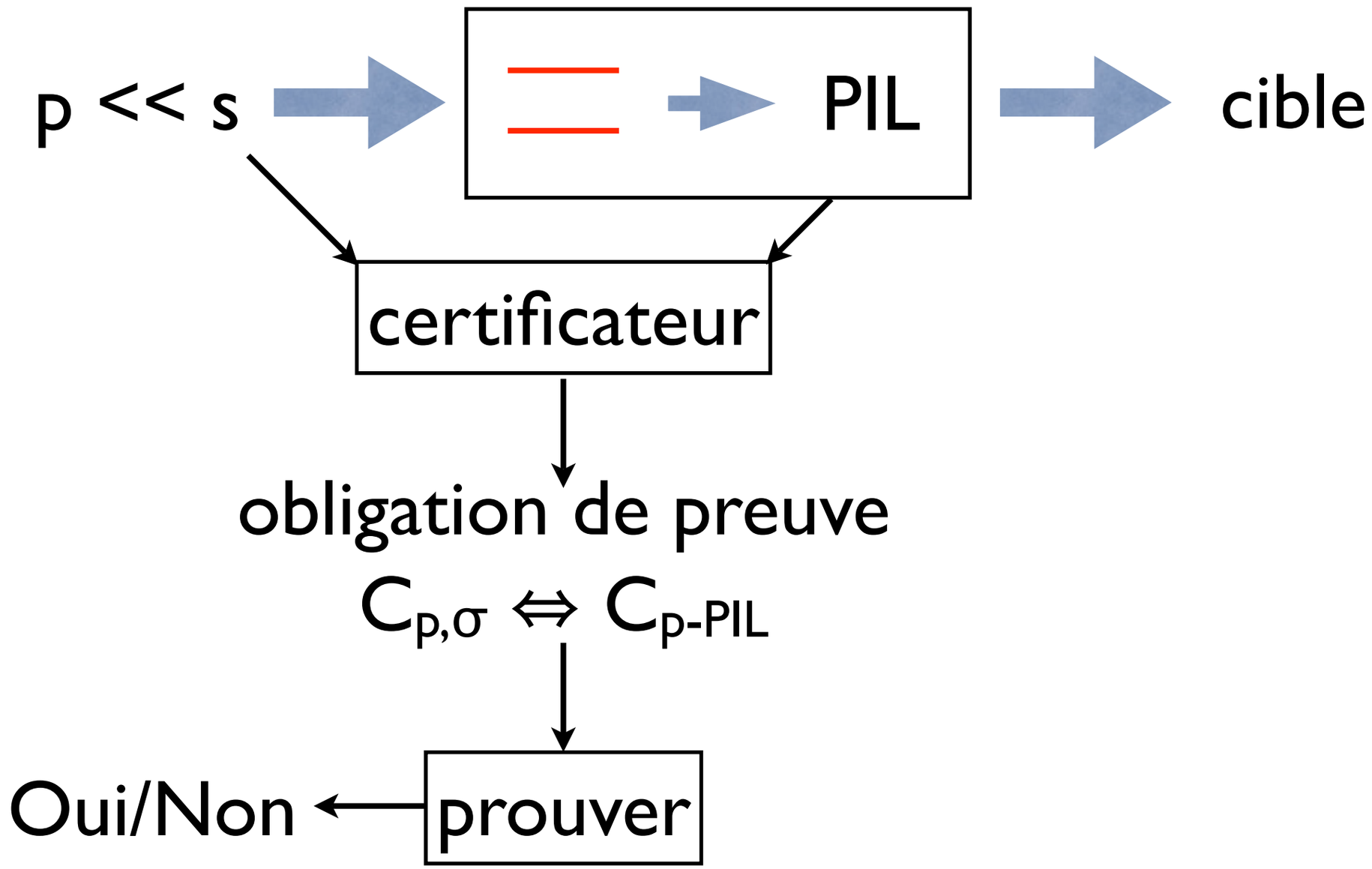
Mécanisme de certification



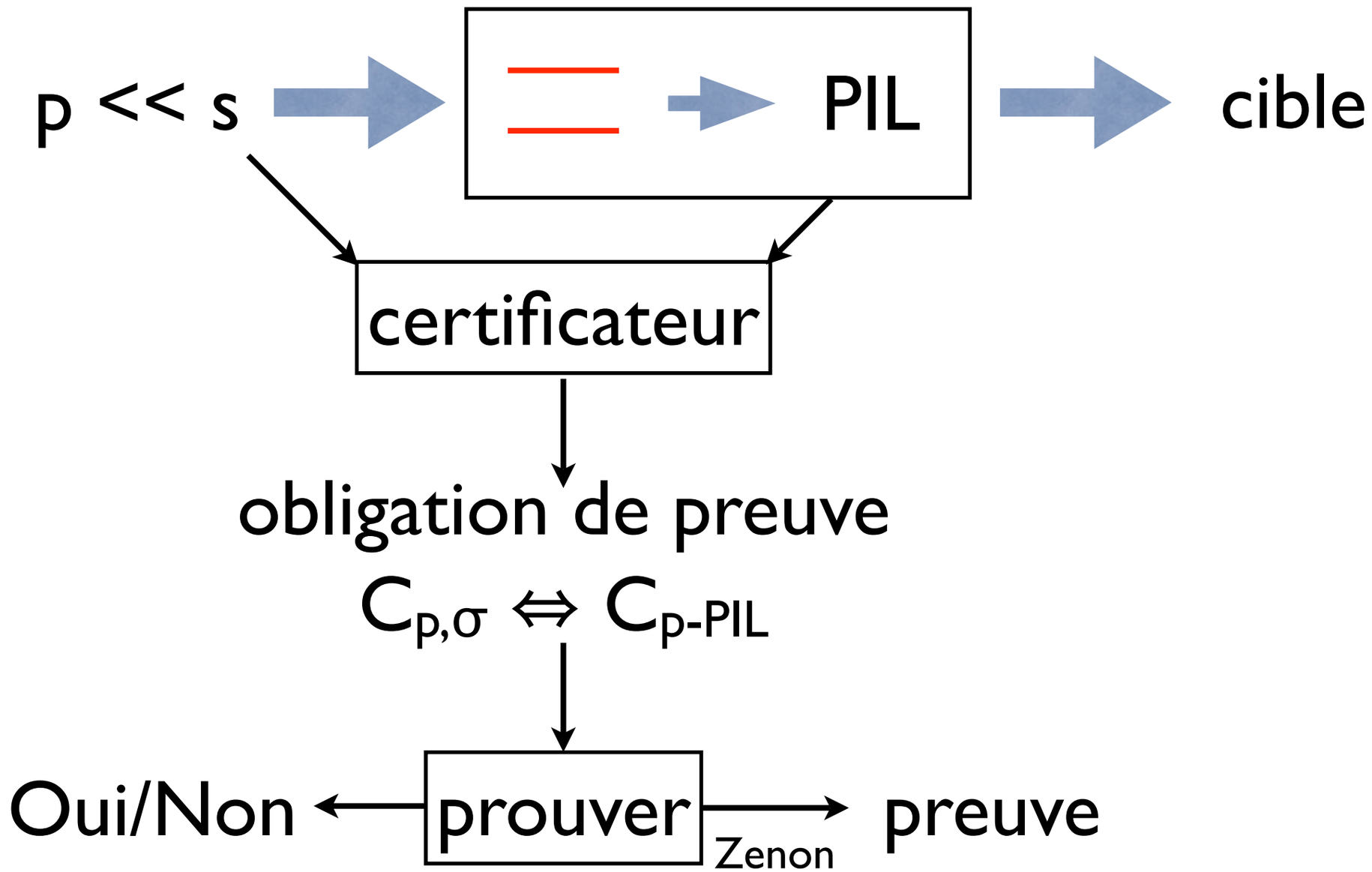
obligation de preuve

$C_{p,\sigma} \Leftrightarrow C_{p\text{-PIL}}$

Mécanisme de certification



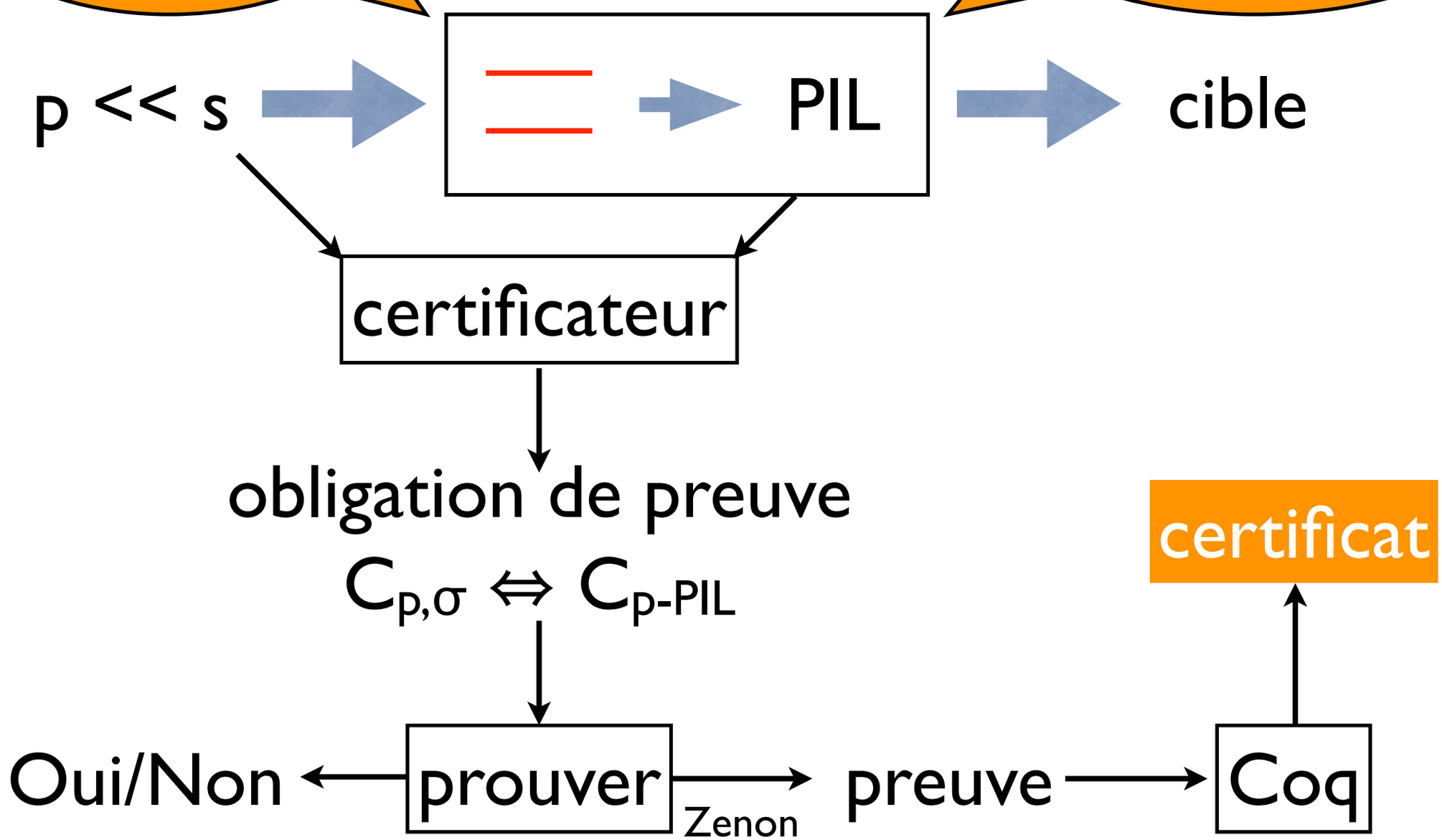
Mécanisme de certification



Mécanisme de certificatio

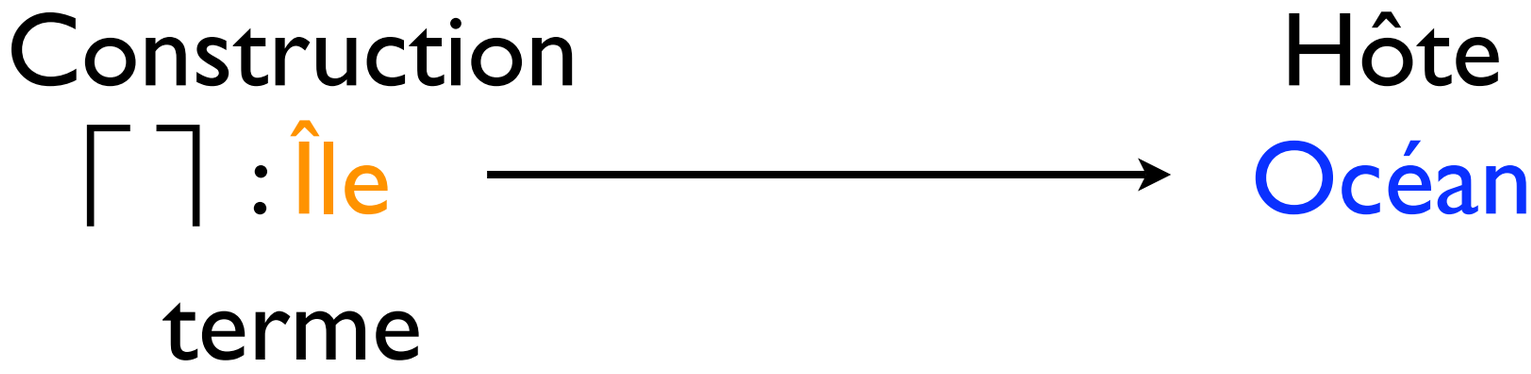
compilateur certifiant

peut évoluer facilement

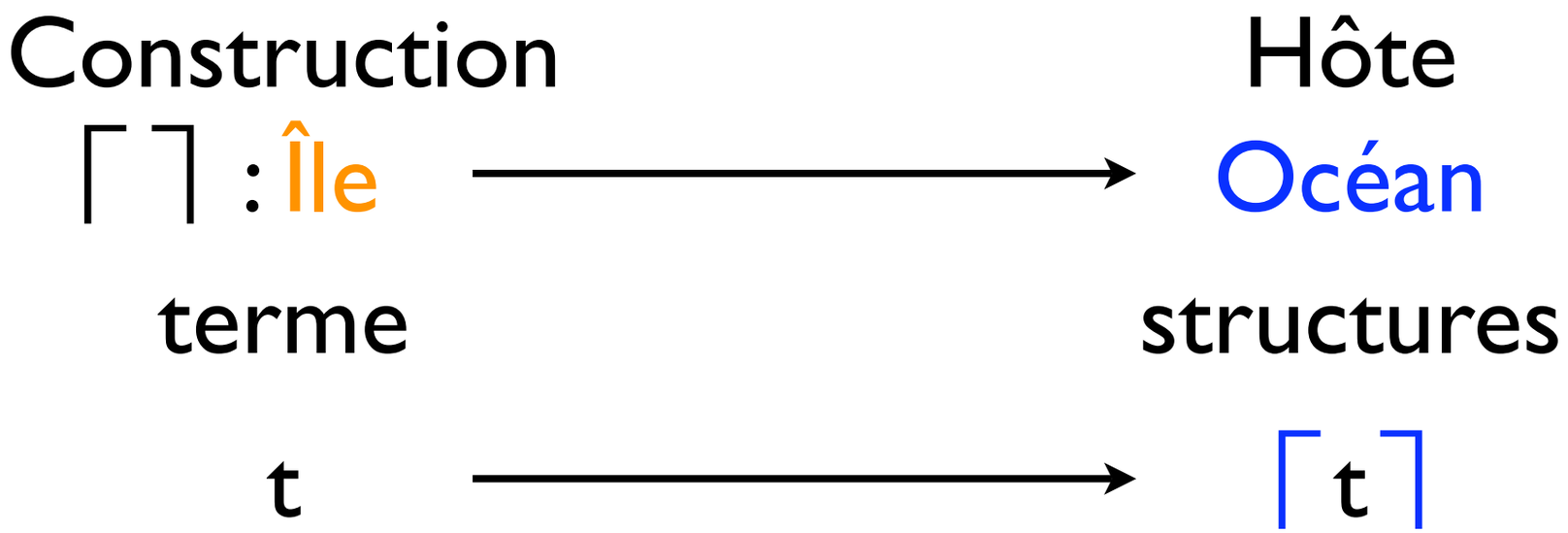


Ancrage formel

Ancrage formel



Ancrage formel



Ancrage formel

Construction		Hôte
$\ulcorner\ \urcorner$: Île	⟶	Océan
terme		structures
t	⟶	$\ulcorner t \urcorner$
Φ : prédicat	⟶	prédicat

Ancrage formel

Construction

$\ulcorner\ \urcorner$: Île ⟶ Océan

terme

structures

t ⟶ $\ulcorner t \urcorner$

Φ : prédicat ⟶ prédicat

$P(t_1,\ldots,t_n) \Leftrightarrow \Phi(P)(\ulcorner t_1 \urcorner, \ldots, \ulcorner t_n \urcorner)$

Ancrage formel

Construction		Hôte
$\ulcorner\ \urcorner$: Île	⟶	Océan
terme		structures
t	⟶	$\ulcorner t \urcorner$
Φ : prédicat	⟶	prédicat

$$P(t_1,\ldots,t_n) \Leftrightarrow \Phi(P)(\ \ulcorner t_1 \urcorner\ ,\ldots,\ \ulcorner t_n \urcorner\ )$$

$$\forall f \in \mathcal{F}, \forall t \in \mathcal{T}(\mathcal{F}),\quad \text{is\_fsym}(\ \ulcorner t \urcorner\ , f) = \ulcorner \text{Symb}(t)=f \urcorner$$

$$\forall f \in \mathcal{F}, \forall t \in \mathcal{T}(\mathcal{F}), \forall i \in [1..ar(f)],\ \text{subterm}_f(\ \ulcorner t \urcorner\ ,\ \ulcorner i \urcorner\ ) = \ulcorner t_{|i} \urcorner$$

îlots formels : formalisation & certification

E. Balland

C. Kirchner

A. Reilles

1. Introduction
2. Îlots formels - Tom
3. Certification
4. *Stratégies*
5. Anti-patterns

Comment contrôler les réécritures ?

Règle et contrôle

Règle et contrôle

- séparer les transformations de leur application

Règle et contrôle

- séparer les transformations de leur application
- différentes façons d'appliquer une règle
  - **stratégies classiques** : *repeat*, *innermost*
  - **stratégies non-déterministes** : *exploration*

Règle et contrôle

- séparer les transformations de leur application
- différentes façons d'appliquer une règle
  - **stratégies classiques** : *repeat*, *innermost*
  - **stratégies non-déterministes** : *exploration*
- quel langage de contrôle choisir ?

Règle et contrôle

- séparer les transformations de leur application
- différentes façons d'appliquer une règle
  - **stratégies classiques** : *repeat*, *innermost*
  - **stratégies non-déterministes** : *exploration*
- quel langage de contrôle choisir ?
- langage hôte : un méta-langage expressif

Stratégies

Stratégies

- ELAN[1994,1998] repeat ; dont-care dont-know
- Stratego[1998] ; <+ all one μ

Stratégies

- ELAN[1994,1998] repeat ; dont-care dont-know
- Stratego[1998] ; <+ all one μ
- JJTraveler[2001] : combinateurs en Java

Stratégies

- ELAN[1994,1998] repeat ; dont-care dont-know
- Stratego[1998] ; <+ all one μ
- JJTraveler[2001] : combinateurs en Java
- Tom : les combinateurs sont des constructeurs
- extensible : pselect(p,q,s_1,s_2)

Terme de stratégie

Terme de stratégie

- une stratégie est un terme

Terme de stratégie

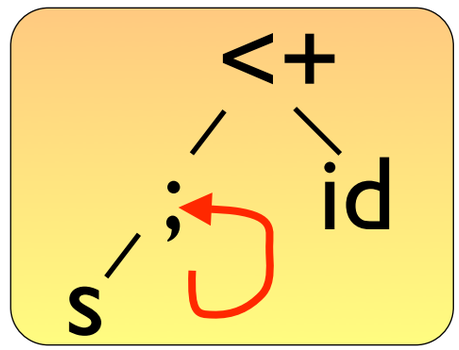
- une stratégie est un terme
- repeat(s) = μ x . (s ; x) <+ id

Terme de stratégie

- une stratégie est un terme
- repeat(s) = μ x . (s ; x) <+ id
- once-bottom-up(s) = μ x . one(x) <+ s
- innermost(s) = repeat(once-bottom-up(s))

Terme de stratégie

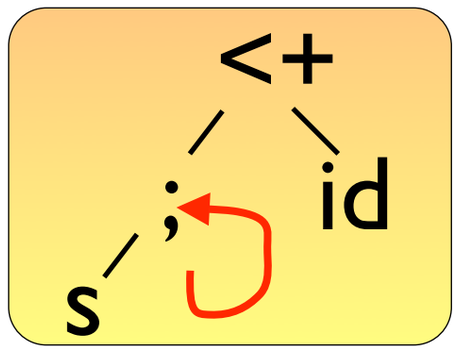
- une stratégie est un terme
- repeat(s) = μ x . (s ; x) <+ id



- once-bottom-up(s) = μ x . one(x) <+ s
- innermost(s) = repeat(once-bottom-up(s))
- des termes aux graphes : μ-expansion

Terme de stratégie

- une stratégie est un terme
- repeat(s) = μ x . (s ; x) <+ id



- once-bottom-up(s) = μ x . one(x) <+ s
- innermost(s) = repeat(once-bottom-up(s))
- des termes aux graphes : μ-expansion

Contribution 1
- construction dynamique
- stratégie applicable sur une stratégie

Stratégie multi-résultats

Stratégie multi-résultats

- stratégies paramètrées par des objets

Stratégie multi-résultats

- stratégies paramètrées par des objets
- Strategy s(Collection c) { x → c.add(x) ... }

Stratégie multi-résultats

- stratégies paramètrées par des objets
- Strategy s(Collection c) { x → c.add(x) ... }
- c = new ArrayList()
- bottom-up(s(c)) : collecte les sous-termes

Contribution 2
- stratégie paramètrée

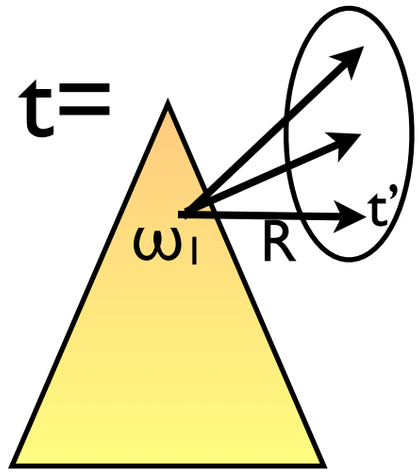
Un problème à résoudre

Un problème à résoudre

- soit R = lhs → rhs et t
- trouver les t' tels que $t \rightarrow_{R,\omega} t'$

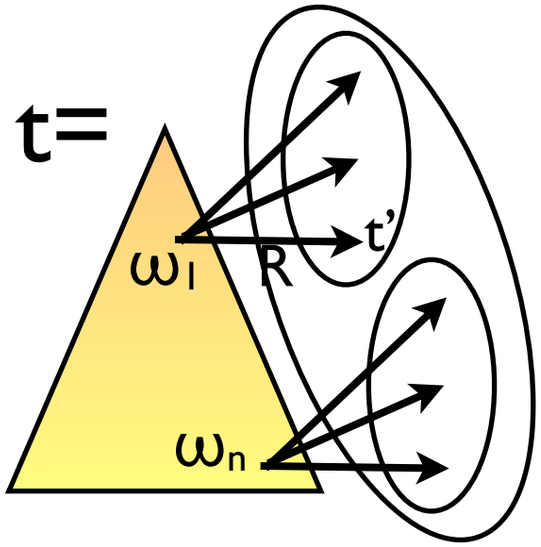
Un problème à résoudre

- soit R = lhs → rhs et t
- trouver les t' tels que $t \rightarrow_{R,\omega} t'$



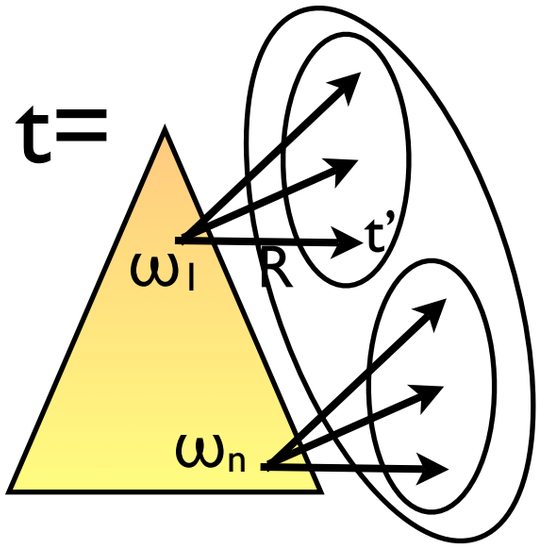
Un problème à résoudre

- soit R = lhs → rhs et t
- trouver les t' tels que $t \rightarrow_{R,\omega} t'$
- $\{ t[\sigma_1\ rhs]_{\omega 1},...,t[\sigma_m\ rhs]_{\omega n} \}$



Un problème à résoudre

- soit R = lhs → rhs et t
- trouver les t' tels que $t \rightarrow_{R,\omega} t'$
- $\{ t[\sigma_1\ rhs]_{\omega 1},...,t[\sigma_m\ rhs]_{\omega n} \}$
- *exercice* : calculer cet ensemble en Java, C, ELAN, Maude, Stratego, ASF+SDF, ... *(not easy)*
- *difficulté* : se souvenir du contexte $t[...]_\omega$



Positions explicites

Positions explicites

- *solution* : { $t[\sigma_1\ rhs]_{\omega 1}$,...,$t[\sigma_m\ rhs]_{\omega n}$ }

Positions explicites

- *solution* : { t[σ_1 rhs]ω_1,...,t[σ_m rhs]ω_n }
- R(Collection c) { lhs → c.add(rhs) }

- *calcule* { σ_1 rhs,...,σ_m rhs }

Positions explicites

- *solution* : { t[σ_1 rhs]ω_1,...,t[σ_m rhs]ω_n }
- R(Collection c) { lhs → c.add(rhs) }

- *calcule* { σ_1 rhs,...,σ_m rhs }
- R(Collection c) { lhs → c.add(t[σ rhs]ω) }

Positions explicites

- *solution* : { t[σ_1 rhs]ω_1,...,t[σ_m rhs]ω_n }
- R(Collection c) { lhs → c.add(rhs) }

- *calcule* { σ_1 rhs,...,σ_m rhs }
- R(Collection c) { lhs → c.add(t[σ rhs]ω) }
- lhs → c.add(replace(t,rhs,getPosition())

Contribution 3
- positions explicites
- accès aux ancêtres

Stratégies

Séparer transformation et contrôle

A. Reilles

E. Balland

1. Introduction
2. Îlots formels - Tom
3. Certification
4. Stratégies
5. *Anti-patterns*

1. 2. 3. 4. **5.**

Comment être plus expressif ?

Motifs et gardes

Motifs et gardes

- match(s) { lhs → if(c) action }

Motifs et gardes

- match(s) { lhs → if(c) action }
- match(s) { lhs when c → action }

Motifs et gardes

- match(s) { lhs → if(c) action }
- match(s) { lhs when c → action }
- **match** { condition → **action** }
- condition = **<<** | **==** | **P** | ∧ | ∨ | **...**

Motifs et gardes

- match(s) { lhs → if(c) action }
- match(s) { lhs when c → action }
- **match** { condition → **action** }
- condition = **<<** | **==** | **P** | ∧ | ∨ | **...**
- **match { (f(x)<<s ∧ g(y)<<x) ∨ ... → action }**

Motifs et gardes

- match(s) { lhs → if(c) action }
- match(s) { lhs when c → action }
- **match** { condition → **action** }
- condition = **<<** | **==** | **P** | ∧ | ∨ | **...**
- **match { (f(x)<<s ∧ g(y)<<x) ∨ ... → action }**

faut-il ajouter la *négation* **?**

Anti-patterns

Anti-patterns

- !a << b

Anti-patterns

- !a << b

  même chose que (a !<< b)

Anti-patterns

- !a << b

  même chose que (a !<< b)

- g(!a) << g(b)
- f(x,!x) << f(a,b)

Listes associatives

Listes associatives

- list(*,x,*,x,*) : 2 éléments identiques

Listes associatives

- list(*,x,*,x,*) : 2 éléments identiques
- list(*,x,*,!x,*) : 2 éléments différents

Listes associatives

- list(*,x,*,x,*) : 2 éléments identiques
- list(*,x,*,!x,*) : 2 éléments différents
- !list(*,x,*,x,*) : tous différents

Listes associatives

- list(*,x,*,x,*) : 2 éléments identiques
- list(*,x,*,!x,*) : 2 éléments différents
- !list(*,x,*,x,*) : tous différents
- !list(*,x,*,!x,*) : tous identiques

Quelle sémantique ?

Quelle sémantique ?

- sol(q<<t) = { σ | t ∈ ⟨σ(q)⟩ avec σ∈GS(q) }

Quelle sémantique ?

- sol(q<<t) = { σ | t ∈ ⟨σ(q)⟩ avec σ∈GS(q) }
- ⟨t⟩ = { σ(t) | σ∈GS(t) }

Quelle sémantique ?

- sol(q<<t) = { σ | t ∈ ⟨σ(q)⟩ avec σ∈GS(q) }
- ⟨t⟩ = { σ(t) | σ∈GS(t) }

- ⟨q[!q']ω⟩ = ⟨q[z]ω⟩ \ ⟨q[q']ω⟩

Quelle sémantique ?

- sol(q<<t) = { σ | t ∈ ⟨σ(q)⟩ avec σ∈GS(q) }
- ⟨t⟩ = { σ(t) | σ∈GS(t) }
- ⟨q[!q']ω⟩ = ⟨q[z]ω⟩ \ ⟨q[q']ω⟩
- ⟨!x⟩ = ⟨z⟩ \ ⟨x⟩ = ∅

Quelle sémantique ?

- sol(q<<t) = { σ | t ∈ ⟨σ(q)⟩ avec σ∈GS(q) }
- ⟨t⟩ = { σ(t) | σ∈GS(t) }

- ⟨q[!q']ω⟩ = ⟨q[z]ω⟩ \ ⟨q[q']ω⟩
- ⟨!x⟩ = ⟨z⟩ \ ⟨x⟩ = ∅
- ⟨f(a,!b)⟩ = ⟨f(a,z)⟩ \ ⟨f(a,b)⟩

Algorithme

Algorithme

filtrage classique

Algorithme

filtrage classique

+ *ElimAnti* :

$q[!q]_\omega << t$ ➠

$\exists z\ q[z]_\omega << t \wedge$

$\forall x \in FV(q')\ not(q[q']_\omega << t)$

Algorithme

filtrage classique

+ *ElimAnti* :

$q[!q]_\omega << t$ ➠

$\exists z\ q[z]_\omega << t \wedge$

$\forall x \in FV(q')\ not(q[q']_\omega << t)$

+ **disunification** [Comon Lescanne 1990]

Propriétés

- *ElimAnti* est correcte et complète
- un problème d'anti-filtrage peut toujours être transformé en un problème équationnel équivalent
- l'anti-filtrage est unitaire, dans le cas syntaxique

Anti-patterns

Spécifier ce qu’on ne veut pas

C. Kirchner

R. Kopetz

Je vous ai parlé

- d'îlots Formels
- de Tom, de certification
- de stratégies
- d'anti-patterns
- *mais pas* : de graphe, d'analyse de programmes, du compilateur Tom, d'inférence d'ancrage, d'ADT, de GC, de smart constructors, d'applications, *etc.*

Perspectives

- Îlots
  - établir des propriétés
  - relier les îlots entre eux
  - identifier de nouveaux îlots

Perspectives

- Stratégies
  - de plus haut niveau
  - montrer des propriétés
  - caractériser les formes normales
  - optimiser

Perspectives

- Intégration
  - noyer les îlots dans des API
  - inférer les ancrages

Perspectives

- Chercher de nouveaux concepts pour
  - analyser des systèmes
  - représenter et transformer des modèles
  - comprendre les combinaisons de langages
  - modéliser des politiques de sécurité

Résumé

- encadrement 3 thèses, 3 masters, 17 stages
- revues (TCS, JFP, IEE, JLAP)
- conférences (AMAST, CC, ESOP, FASE, PPDP, RTA, ...)
- co-président de RULE'05, JFLA'06, JFLA'07
- membre de 20 comités, dont CC'2009
- RNTL Manifico, ANR Modulogic, Ravaj, NoE Rewerse

Résumé

- encadrement 3 thèses, 3 masters, 17 stages
- revues (TCS, JFP, IEE, JLAP)
- conférences (AMAST, CC, ESOP, FASE, PPDP, RTA, ...)
- co-président de RULE'05, JFLA'06, JFLA'07
- membre de 20 comités, dont CC'2009
- RNTL Manifico, ANR Modulogic, Ravaj, NoE Rewerse

Remerciements

Claude Kirchner, Hélène Kirchner
Antoine Reilles (Dassault Systèmes)
Émilie Balland, Radu Kopetz, Paul Brauner, ...
mes collègues et amis, ma famille