Performances et généricité

LRDE Epita

25 février 2009

Pierre-Etienne Moreau
INRIA Nancy - Grand Est

Programmation par règles et stratégies

INRIA Nancy - Grand Est

Equipe PAREO

pareo.loria.fr

Comment rendre une application plus *efficace* ?

efficace

efficace

- en exploitant les multi-core

efficace

- en exploitant les multi-core
- en optimisant le code

efficace

- en exploitant les multi-core
- en optimisant le code
- en utilisant de meilleures structures de données
- en utilisant de meilleurs algorithmes

Comment rendre une application plus *générique* ?

générique

générique

- en évitant les cas particuliers

générique

- en évitant les cas particuliers
- en généralisant

générique

- en évitant les cas particuliers
- en généralisant
- en introduisant des niveaux d’abstraction
- en évitant les effets de bord

Tom

- Termes
- Filtrage
- 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

Business Objects
an SAP company

tom.loria.fr

Quelques succès

- traduction efficace MDX vers SQL
- compilation
- model checker - vérification de protocoles
- simulation de réactions chimiques
- assistant à la preuve

Programmation par règles

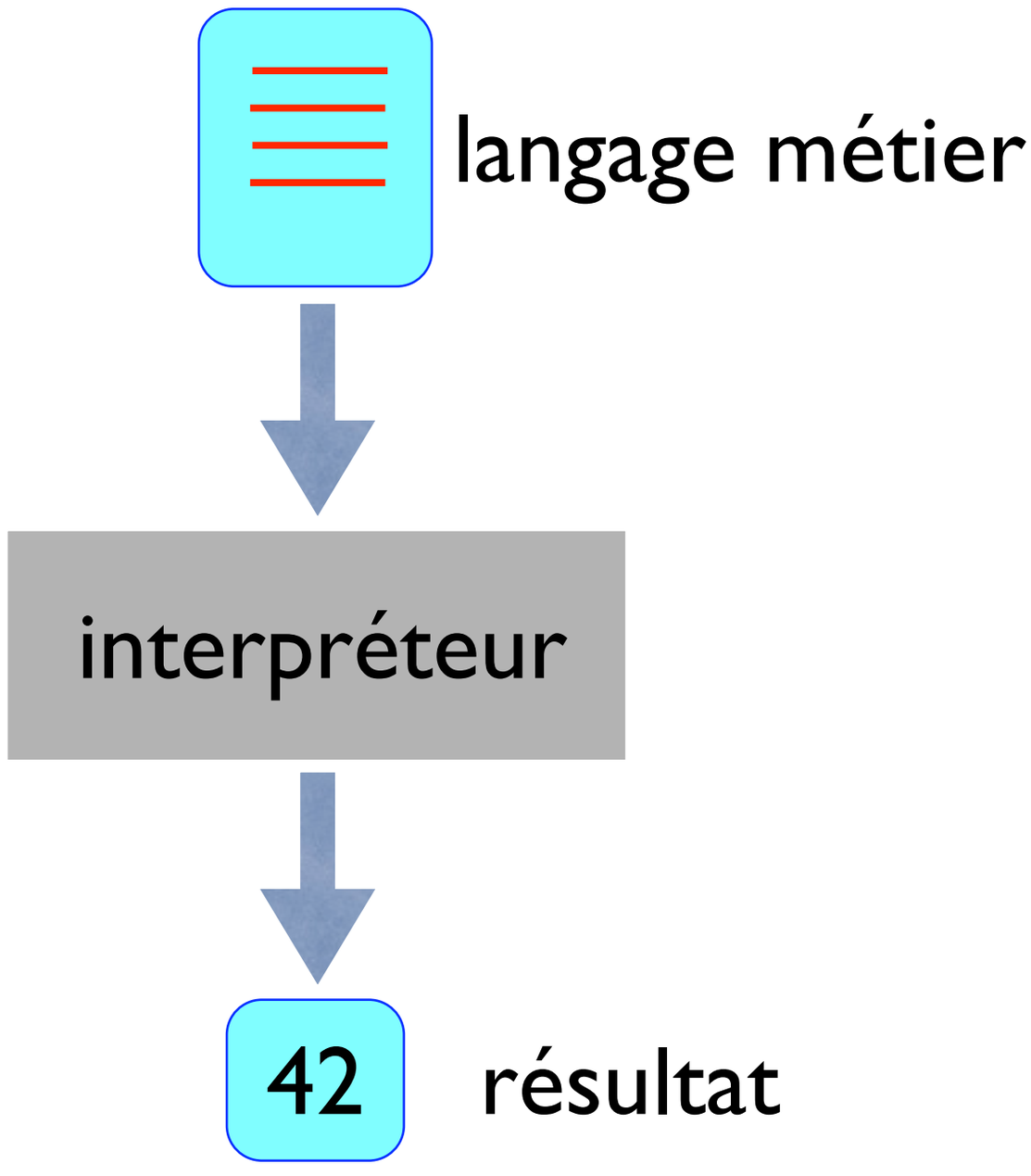
Application aux DSLs

DSL

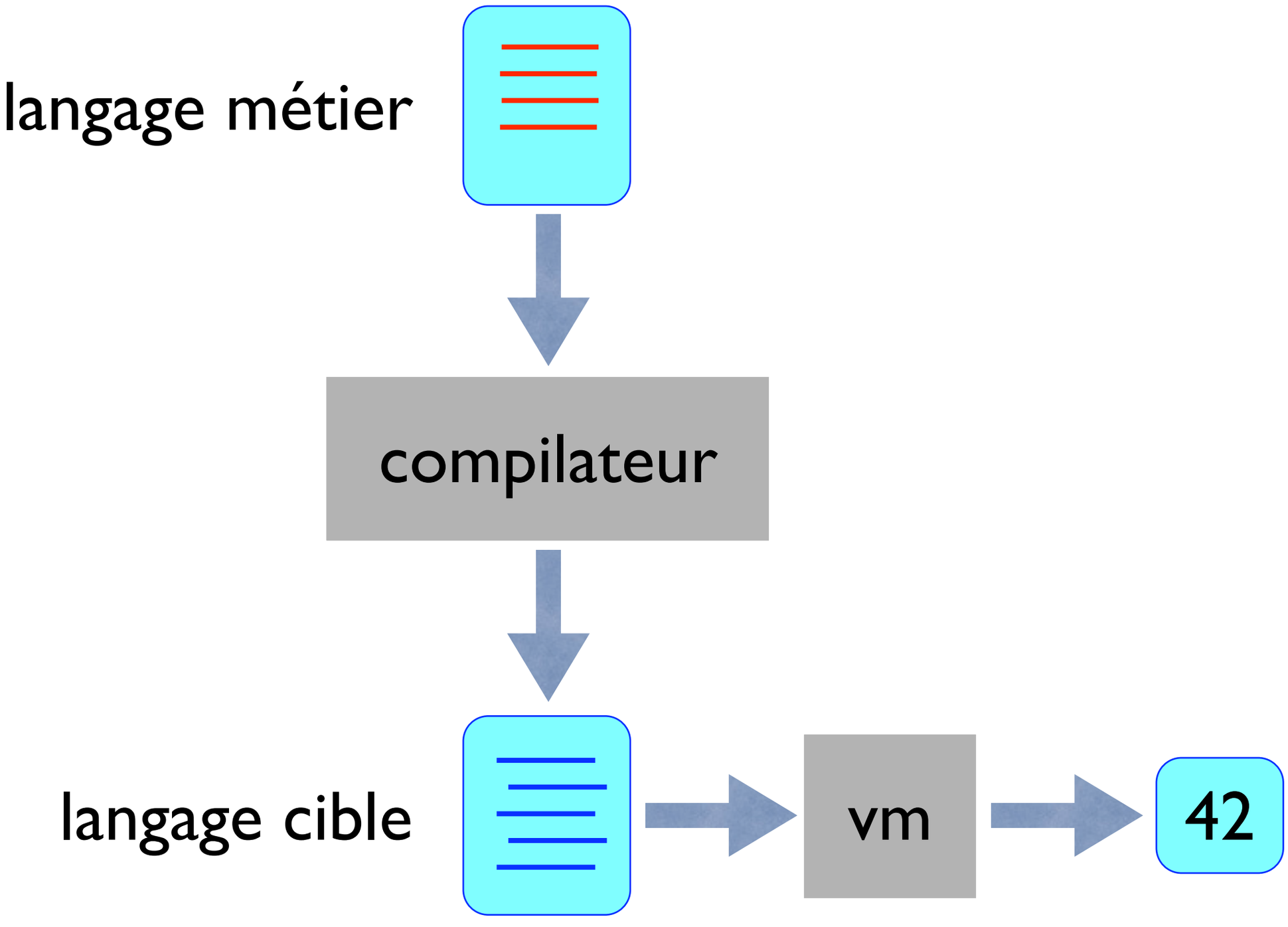
Problèmes liés à la réalisation d'un DSL

- définition du langage métier
- environnement d'édition
- environnement d'exécution
- vérification de propriétés

DSL : exécution



DSL : compilation

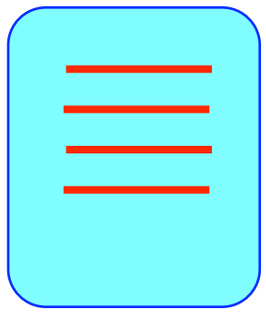


étapes

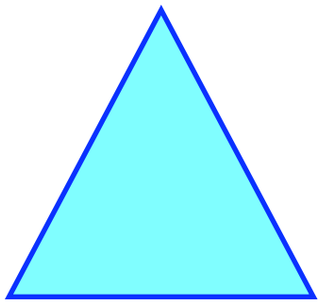
- analyse syntaxique
- typage
- optimization
- compilation
- génération de code

comment programmer ces étapes ?

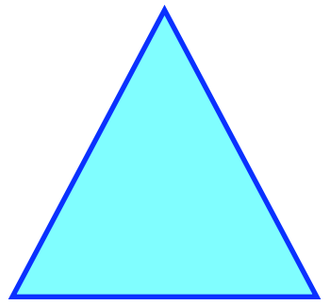
étapes séparées



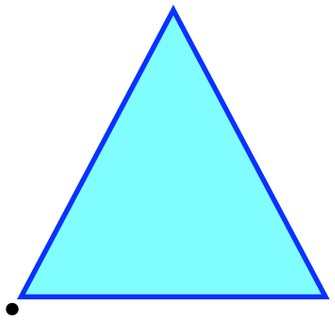
parsing



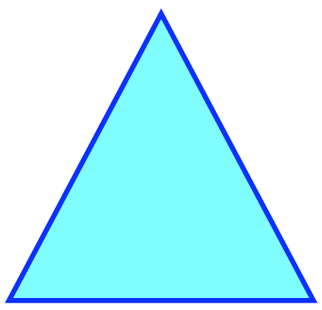
typing



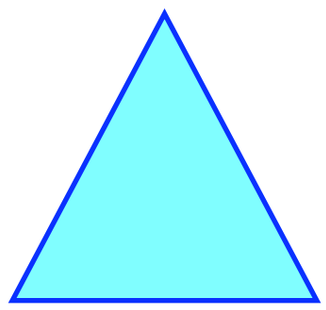
optim.



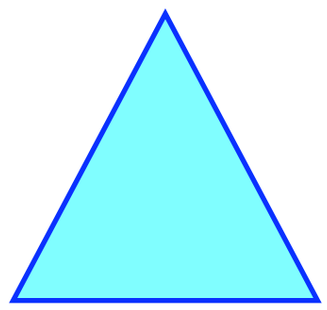
compilation



generation

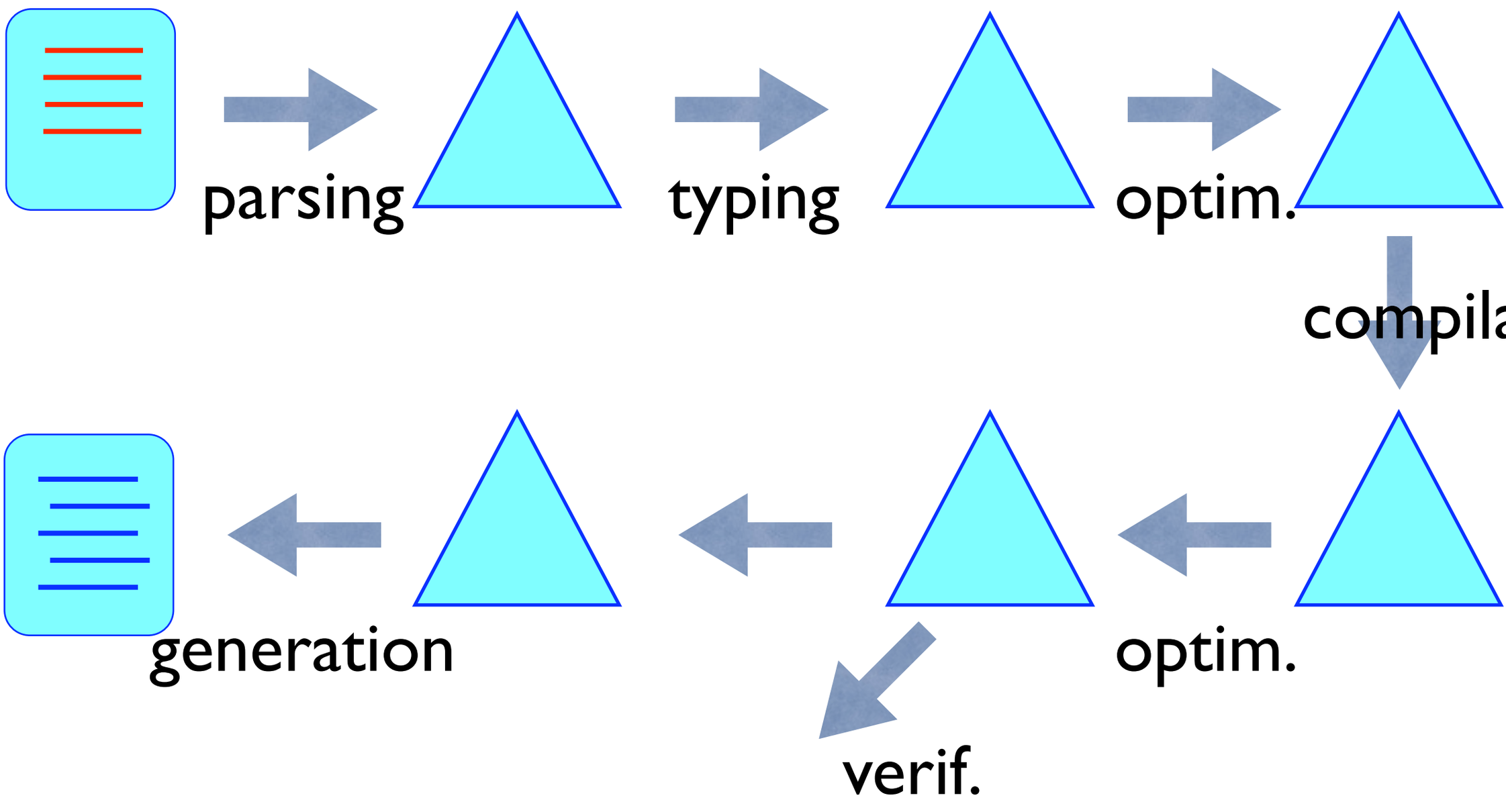


verif.

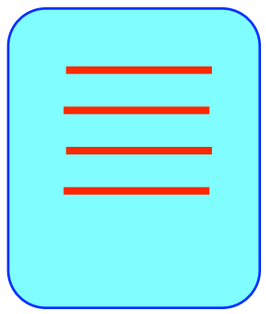


optim.

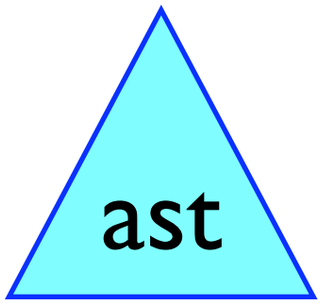
étapes séparées



utilisation d'arbres

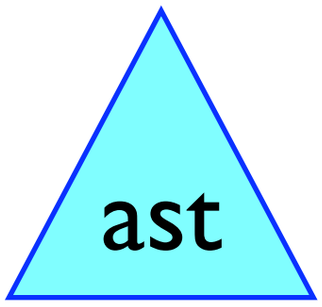


parsing



- ast-1 : abstraction du langage source

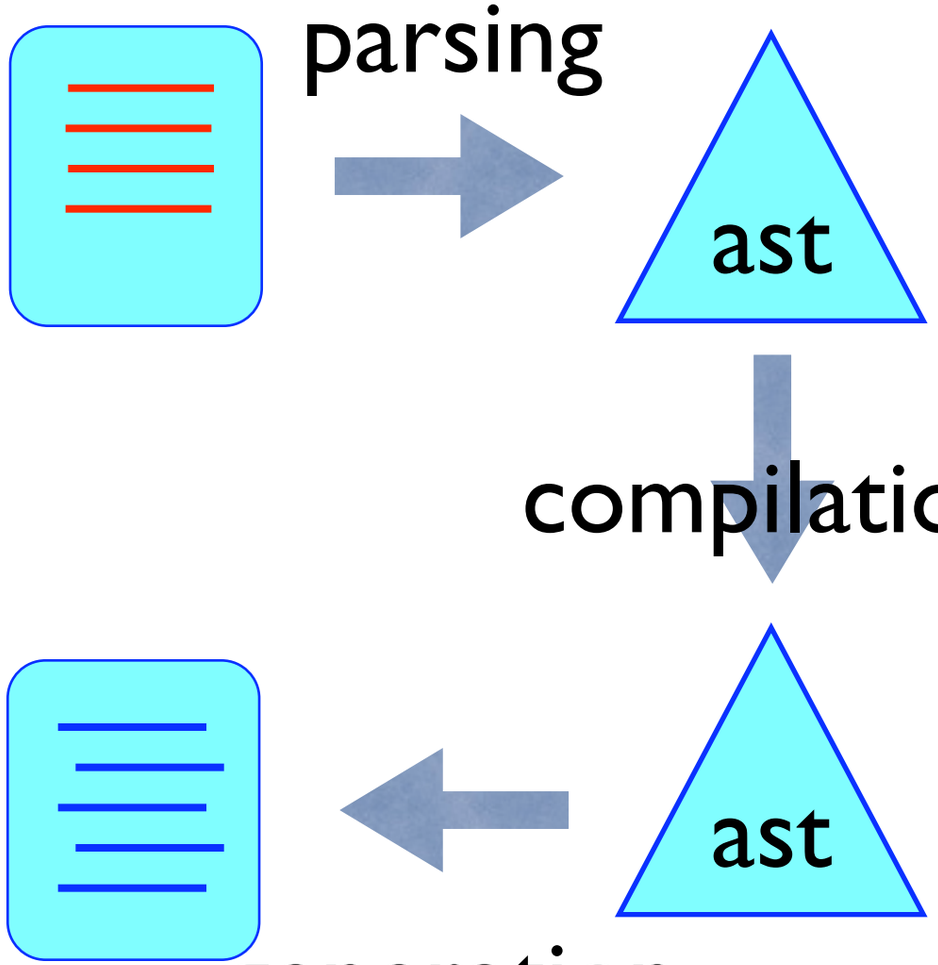
compilation



generation

- ast-2 : langage intermédiaire
- compilation, optimisation par transformation d'arbres

utilisation d'arbres



- ast-1 : abstraction du langage source

- ast-2 : langage intermédiaire

- compilation, optimisation par transformation d'arbres

intérêts ?

avantages

- séparation des phases
- compilateur extensible
- développement collaboratif
- pas d’effet de bord

comment écrire un tel compilateur ?

avec un DSL pour transformer des arbres

réécriture

Concepts principaux

a → b : *règle* décrivant une transformation

S : *stratégie* contrôlant les applications

Réécriture

- expressif
- exécutable
- formel

Permet de raisonner

- terminaison
- confluence
- complexité
- combinaison

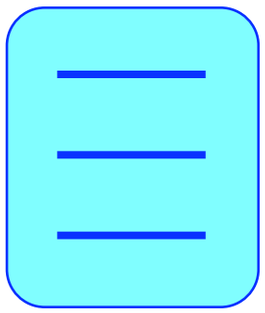
Réécriture

à la base de nombreux langages

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

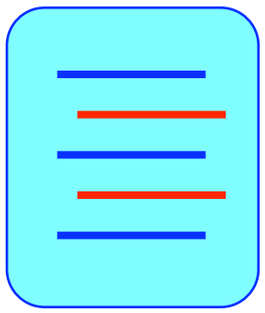
Comment rendre utilisables les concepts liés à la réécriture ?

A piggyback ride



hôte

A piggyback ride

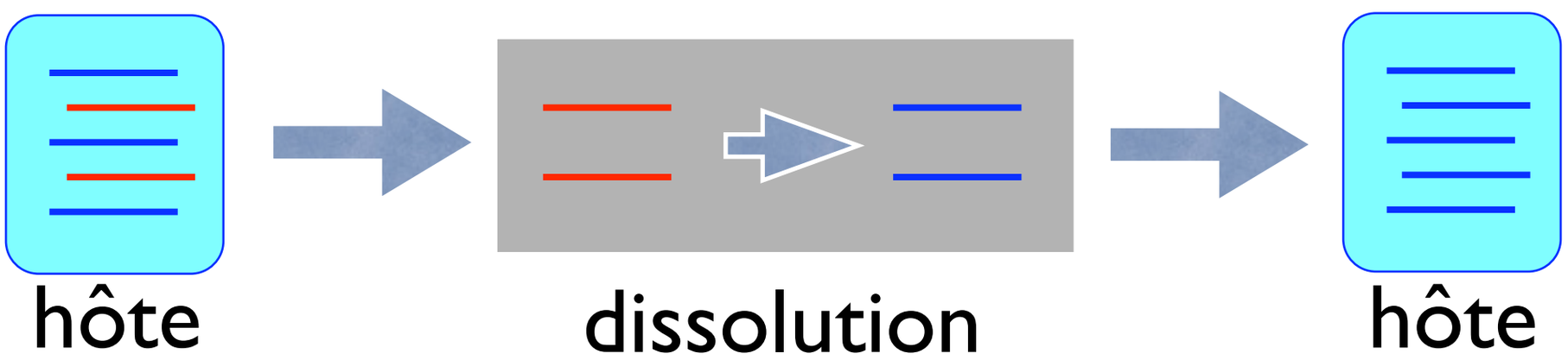


hôte

A piggyback ride

hôte hôte

A piggyback ride



hôte

dissolution

hôte

Tom [2001]

Un moyen de rendre utilisable la réécriture

M. Vittek

C. Ringeissen

Îlot Formel

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

Demo

résumé

- construction de termes
- filtrage
- manipulation de listes
- représentation efficace en mémoire
- connexion aisée avec un parseur

séparer les règles du contrôle

stratégies

- un DSL pour contrôler l'application des règles
- sépare les règles "métier" de leur contexte d'utilisation
- augmente leur ré-utilisation

stratégies

- objet qui s'applique sur un terme
- pour produire un nouveau terme
- peut échouer

stratégies élémentaires

- identité
- échec
- règle de réécriture
  - (a→b)[a] = b
  - (a→b)[c] = échec

combinateurs élémentaires

- S1 ; S2 (séquence)
- S1 <+ S2 (choix de gauche à droite)

stratégie composée

- try(s) = s <+ id

stratégie récursive

- repeat(s) = try(s ; repeat(s))

stratégies de congruence

- all(s), one(s)
- BottomUp(s), TopDown(s)
- OnceBottomUp(s), OnceTopDown(s)
- Innermost(s) = Repeat(OnceBottomUp(s))

Demo

résumé

- sépare règles et contrôle
- permet de traverser un terme
  - collecter de l'information
  - effectuer des transformations

Je vous ai parlé

- de Tom : *http://tom.loria.fr*
- de programmation par filtrage
- de stratégies
- *mais pas* : de graphe, d'anti-pattern, de certification, d'inférence d'ancrage, d'ADT, de GC, de smart constructors, *etc.*

ce qu'il faut retenir

- termes
  - introduit un niveau d'abstraction
  - réduit les effets de bords
- règles et stratégies
  - réduit la maintenance
  - permet de programmer des algorithmes plus complexes