Compilation

Didier Mailliet

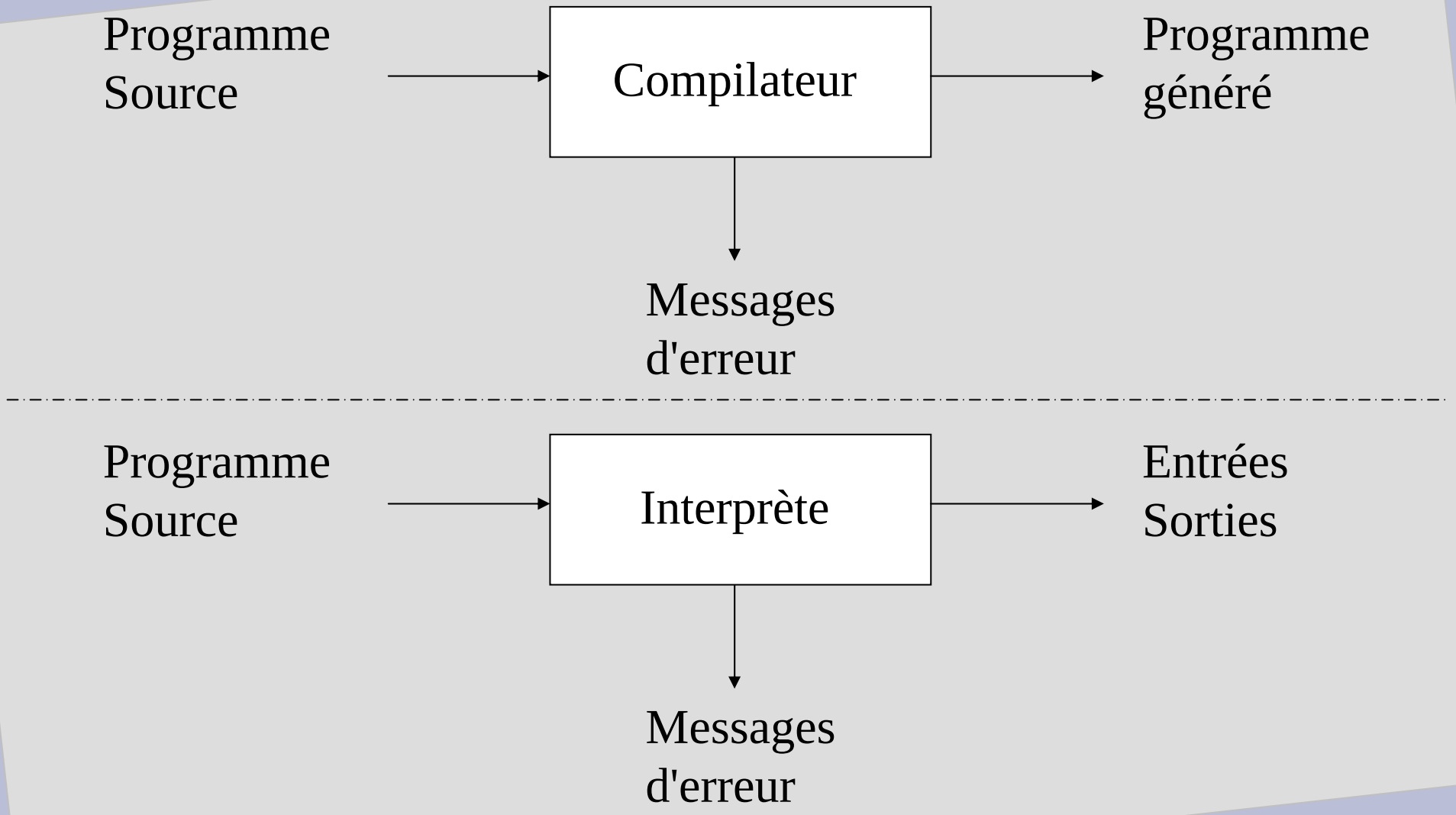
IEEA
Université Lille1

Année 2009

Domaines lies a la compilation

- Langages de programmation
- Matériel informatique
- Théorie des langages
- Algorithmique
- Génie logiciel

Fonctionnement d'un compilateur



Variétés de compilateurs

- De tres nombreux langages:
  - des langages généraux: C, Java, etc.
  - aux langages spécialises: awk, interprètes de commandes, etc.
- Plusieurs sortes de langages cibles:
  - langages machines ou assembleurs
  - langages de machines virtuelles
  - autres langages de haut niveau
- Compilateurs à une ou plusieurs passes
- Compilateurs permettant le déverminage
- Compilateurs optimisants

Outils utilisant la technologie de compilation

- Éditeurs structurés
- Enjoliveurs de texte et de programmes (pretty-printers)
- Vérificateurs statiques de programmes
- Traitement de texte
- Compilateurs vers le matériel
- Interprètes de requêtes
- Convertisseurs d'images, de vidéo, de sons…

Historique

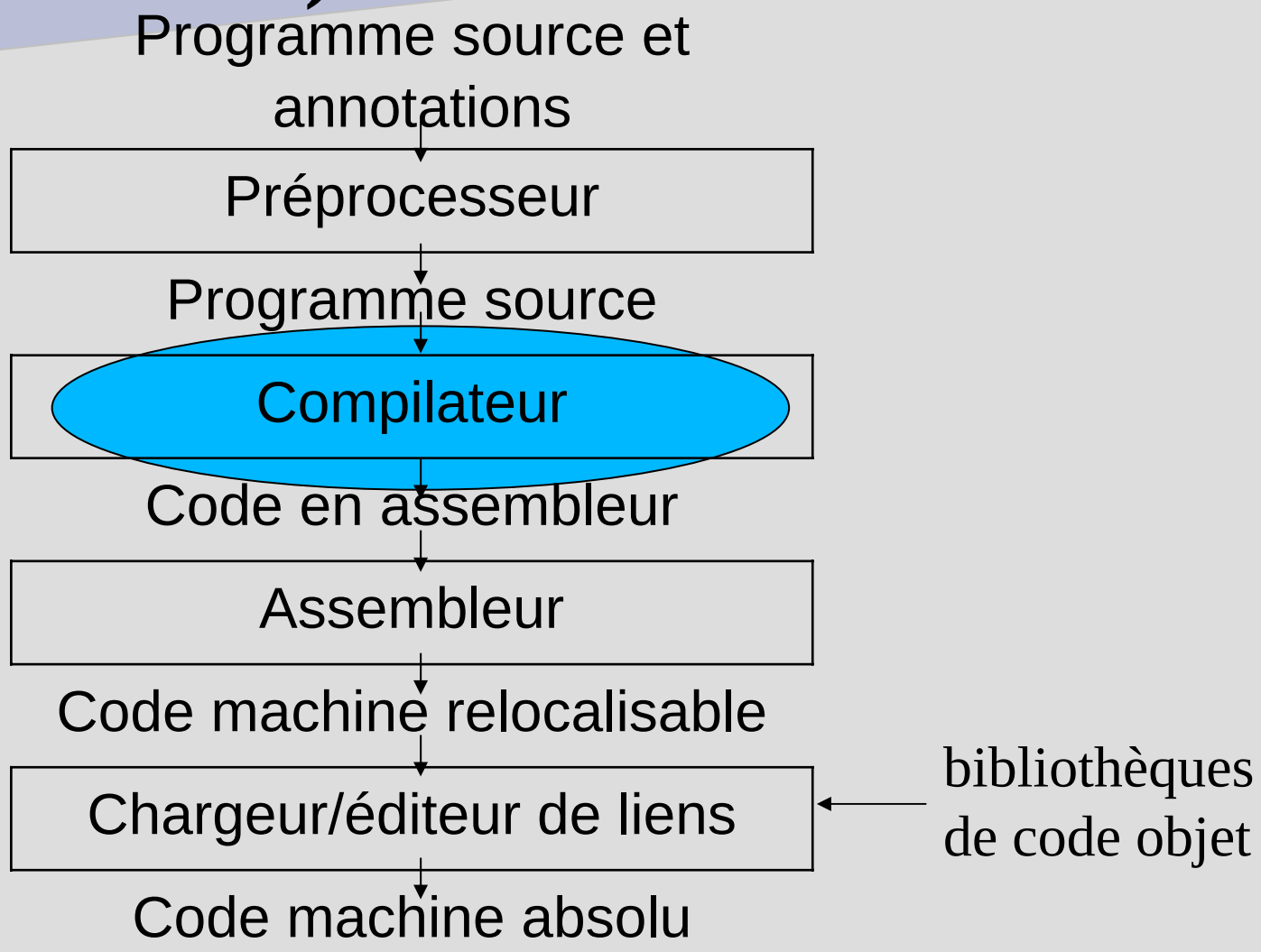
3 périodes

- 1945 – 1960 : Génération de code
- 1960 – 1975 : Analyse syntaxique
- 1975 – actuellement : génération & optimisation de code ; paradigmes (fonctionnelles, logique, distribuées …)

Qu'est-ce qu'un compilateur

- C'est un logiciel qui transforme une entrée en sortie équivalente :
  - Sources et cible ne sont pas nécessairement des programmes
  - Cible n'est pas nécessairement exécutable
- Il vérifie (l'analyse) que la source est correcte
- Émet éventuellement un message d'erreur
- Calcule une donnée de sortie

Organisation de l'implantation d'un langage de programmation (contexte) :



Des critères importants pour faire un « bon » compilateur

- Correction:
  - Entrée valide? Sortie conforme ?

  outil de prédilection = théorie du langage
- Efficacité
  - Rapidité, optimisation, précision du résultat…

  outil de prédilection = algorithmique
- Conception
  - Maintenance, modification, extension…

  outil de prédilection = génie logiciel

C'est ce qui fait de la compilation un sujet varié et passionnant

À quoi sert la théorie du langage (en compilation)

Permet de définir rigoureusement et reconnaître algorithmiquement (pour les langages source et cible)

- leur vocabulaire ou lexique : les mots autorisés
  - automates à nombre fini d'états, expressions régulières
  - analyse lexicale
- leur syntaxe: la structure des phrases autorisées
  - automates à pile, grammaires algébriques
  - analyse syntaxique
- leur sémantique : la signification des phrases autorisées
  - grammaires attribuées
  - analyse sémantique.

À quoi sert le génie logiciel (en compilation)

Notions bien identifiées et couramment admises:

- structuration d'un compilateur en modules
- conception objet, structures de données, algorithmes
- génération automatique de code pour les analyseurs lexical et syntaxique

Les phases de l'analyse d'un programme

Il y a normalement trois phases dans l'analyse:

- Analyse lexicale: Découper et regrouper les caractères constituant un programme en des jetons.
- Analyse syntaxique: Organiser de façon hiérarchique les jetons en une structure arborescente qui reflète la structure syntaxique du programme.
- Analyse sémantique: Vérifier si certaines règles liées à la signification des programmes en un langage donné sont respectées.

Analyse lexicale

À titre d'exemple, l'analyse lexicale de l'énonce:

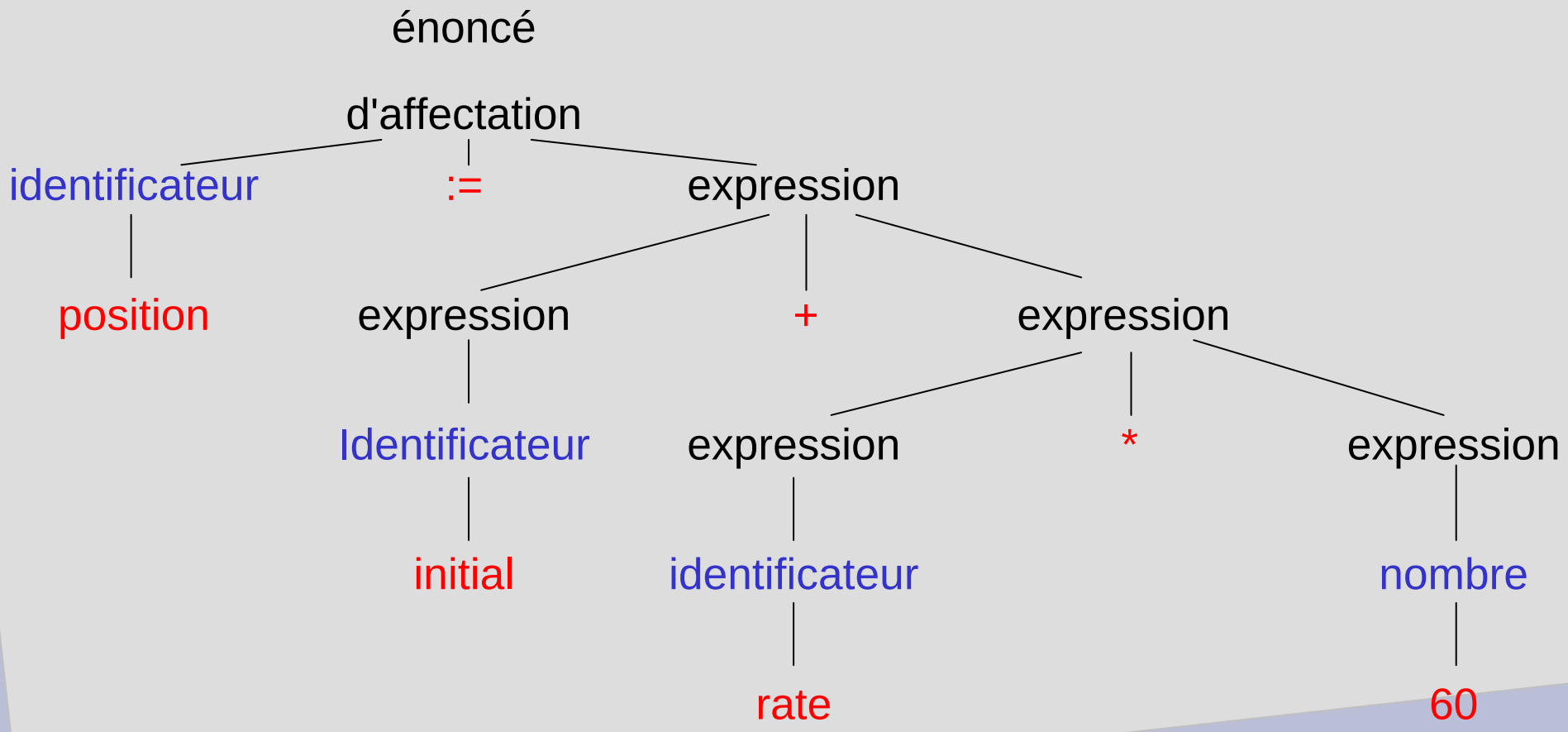
position := initial + rate * 60

devrait donner la suite de jetons suivante:

1. l'identificateur position;
2. le symbole d'affectation;
3. l'identificateur initial;
4. le symbole d'addition;
5. l'identificateur rate;
6. le symbole de multiplication; et
7. le nombre 60.

Analyse syntaxique

L'arbre de syntaxe concrète pour notre énonce est quelque chose comme:



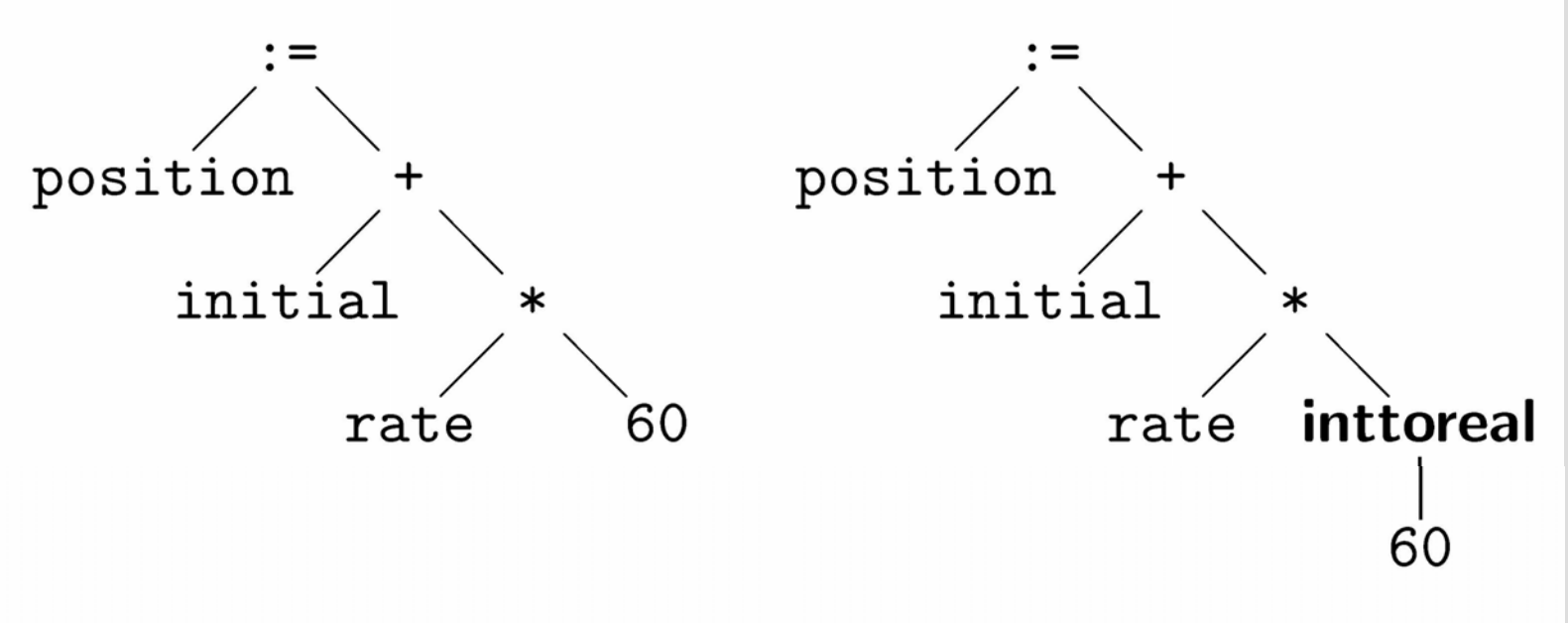
Analyse syntaxique

Dans le cas de notre exemple d'affectation, la syntaxe des expressions pourraient être données a l'aide de règles telles que:

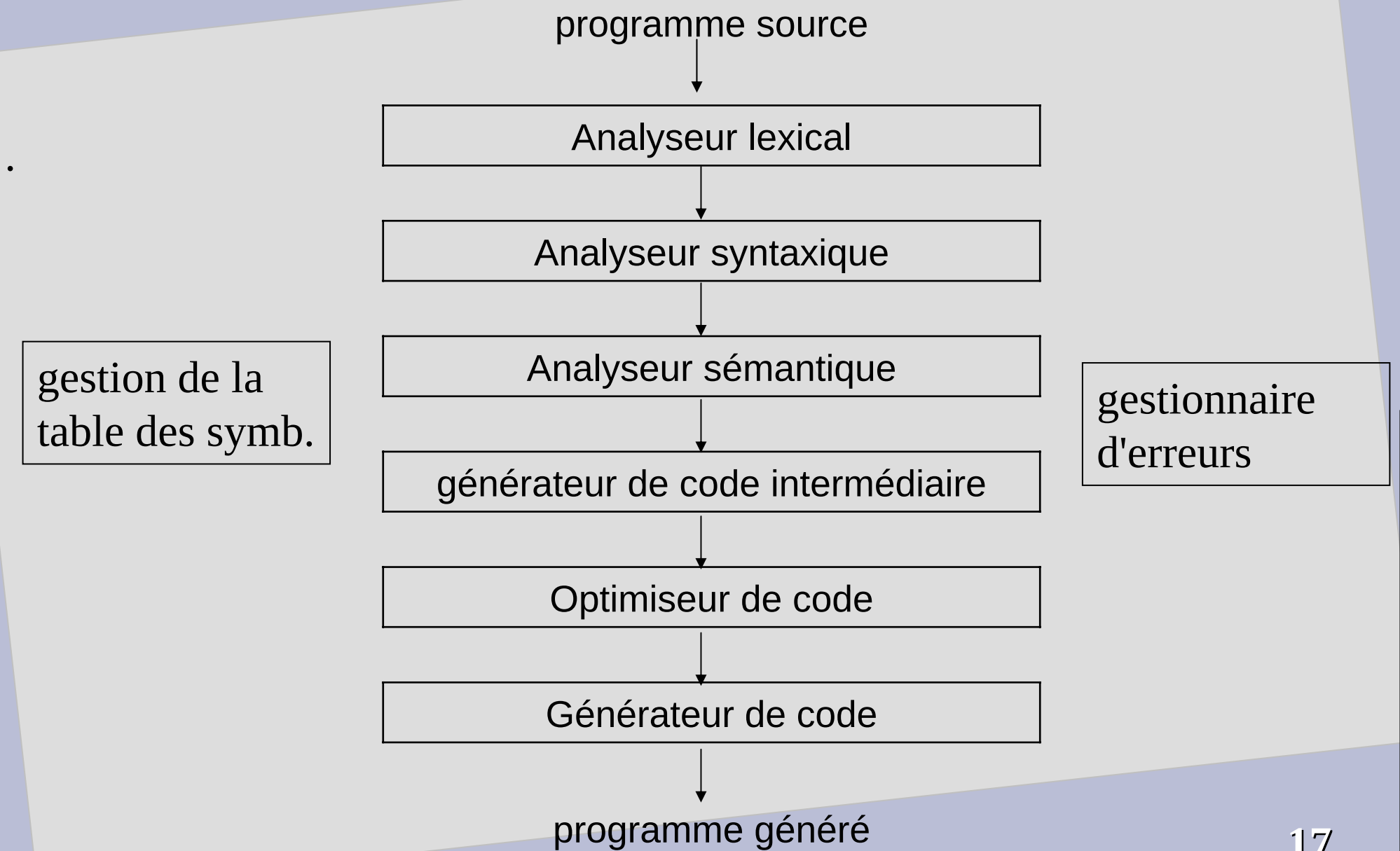
1. Tout identificateur est une expression.
2. Tout nombre est une expression.
3. Si e1 et e2 sont des expressions, alors les constructions suivantes en sont aussi:
   - `e1 + e2`
   - `e1 * e2`
   - `( e1 )`

Analyse sémantique

L'analyse sémantique sert entre autres a faire respecter les règles de typage. En particulier, des conversions automatiques entre types peuvent être ajoutées.



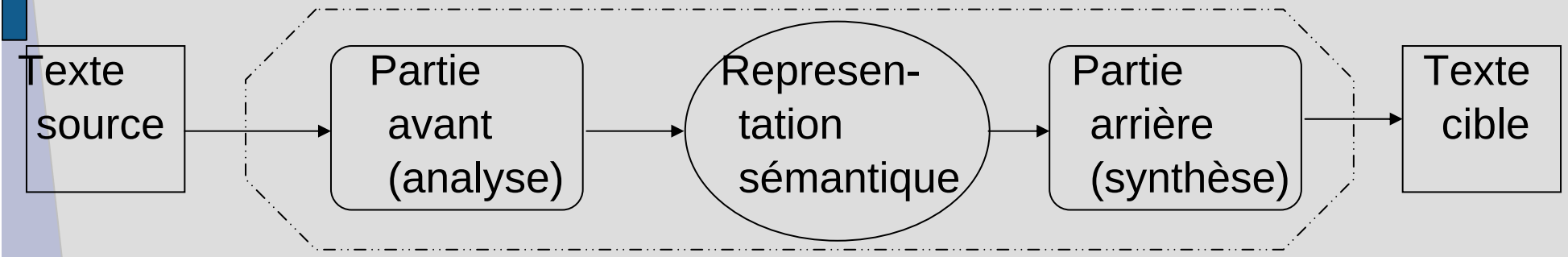
Les phases d'un compilateur



Structure globale

En 2 parties :

- Analyse/reconnaissance
- Synthèse /transformation



Structure classique d'une application de traitement de données

Analyse lexicale

Seul module au contact avec le texte source :

- lit le texte source sous la forme d'une suite de caractères
- Décompose cette suite en une suite d'unités lexicales appelées symboles ou tokens
- En pratique souvent sous-programme appelé par l'analyseur syntaxique
- Génération automatique de code courante (à partir d'une description formelle des unités lexicales, cf TP).

Crible

- Détermine quels symboles sont importants pour la suite de l'analyse et sont envoyés à l'analyseur syntaxique
- Supprime tous les symboles qui ne sont pas significatifs de la structure du texte
- Numérotation éventuelle des identificateurs
- souvent confondu avec l'analyseur lexical.

Analyse syntaxique

- Reçoit les symboles issus de l'analyse lexicale
- En pratique appelle l'analyseur lexical
- Sait comment est structuré un texte correct (expressions, instructions, déclarations, etc
- Tente de reconnaître dans le flot des symboles la structure d'un texte correct
- Structure sous-jacente : arbre syntaxique
- Différentes approches, génération automatique de code courante (cf TP).

Analyse sémantique

Vérifie certaines propriétés dites statiques (= à la compilation, par opposition à dynamique = à l'exécution).

- Vérification de typage
- Vérification des déclarations
- En pratique, parfois intégré à l'analyse syntaxique

Produit une représentation interne du source, selon les cas :

- Un arbre syntaxique décoré
- Un arbre abstrait décoré
- Un code intermédiaire
- ...

Optimisations indépendantes de la machine cible

Analyses plus ou moins poussées pour signaler :

- Risques d'erreur à l'exécution
- Opportunités d'optimisation.

Analyse de flot de données :

- Propagation de constantes
- Indication des variables non initialisées/non utilisées
- Élimination du code mort
- Factorisation d'invariants de boucle
- Élimination de calcul redondants…

Allocation mémoire

Début de la phase de synthèse, dépend de la machine cible :

- Longueur d'un mot ou d'une adresse ?
- Entités directement adressables de la machine ?
- Contraintes d'alignement (frontière de mots) ?

Ex : x adresse 0, y adresse 1.

Génération du programme cible

- Utilise les adresses calculées par l'allocation mémoire
- Accès aux registres plus rapide qu'accès à la mémoire
- Nombre de registres limité : allocation des registres
- Sélection de code.

Ex : on suppose une machine cible avec les instructions suivantes :

LOAD adr, reg		ADDI int,reg
STORE reg, adr		MUL adr, reg
LOADI int, reg		

et un registre R. Pour x := 2 ; y := x*x+1;

1	LOADI 2,R	2	STORE R,0
3	LOAD 0,R	4	MUL 0,R
5	ADDI 1,R	6	STORE R,1

Optimisations dépendantes de la cible

Optimisation à lucarne : déplacement d'une fenêtre sur le programme cible, pour améliorer les parties visibles.

- Éliminer les instructions inutiles
- Remplacer les instructions générales par des instructions plus efficaces.

Ex

- STORE R,0 LOAD 0,R : LOAD inutile
- si la machine possède une instruction INC reg remplacer ADDI 1,R par INC R.