Compilation et interprétation

IFT-3101

Danny Dubé
Université Laval

* Crédits *

Matériel monté par:

- Danny Dubé
- Mathieu Marcotte-Gagnon

Aide financière et logistique:

- Département d'informatique et de génie logiciel
- Bureau des services pédagogiques

Introduction

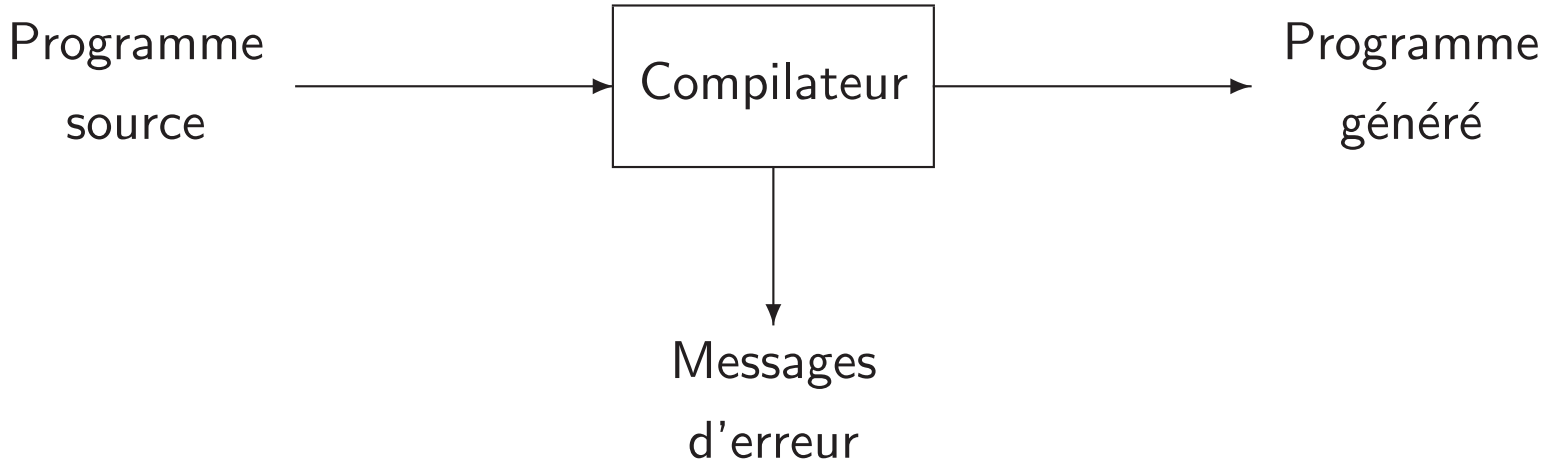
Chapitre 1

Sections 1.1, 1.2, 1.3, 1.4, 1.5

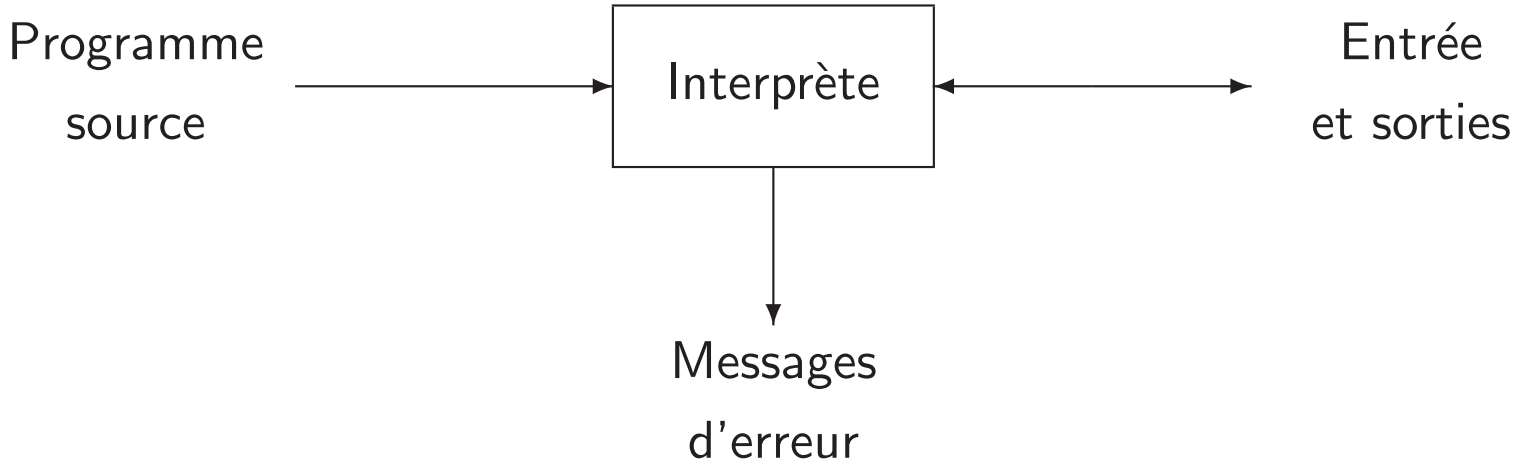
* Contenu *

- Fonctionnement d'un compilateur et d'un interprète
- Domaines liés
- Types de compilateurs et applications de la compilation
- Contexte du compilateur
- Phases de la compilation
- Outils de construction de compilateurs
- Évolution des langages de programmation

Fonctionnement d'un compilateur



Fonctionnement d'un interprète



Domaines liés à la compilation

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

Variétés de compilateurs

- De très nombreux langages:
  - des langages généraux: C, Java, etc.
  - aux langages spécialisés: `awk`, interprètes de commandes, Verilog, 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

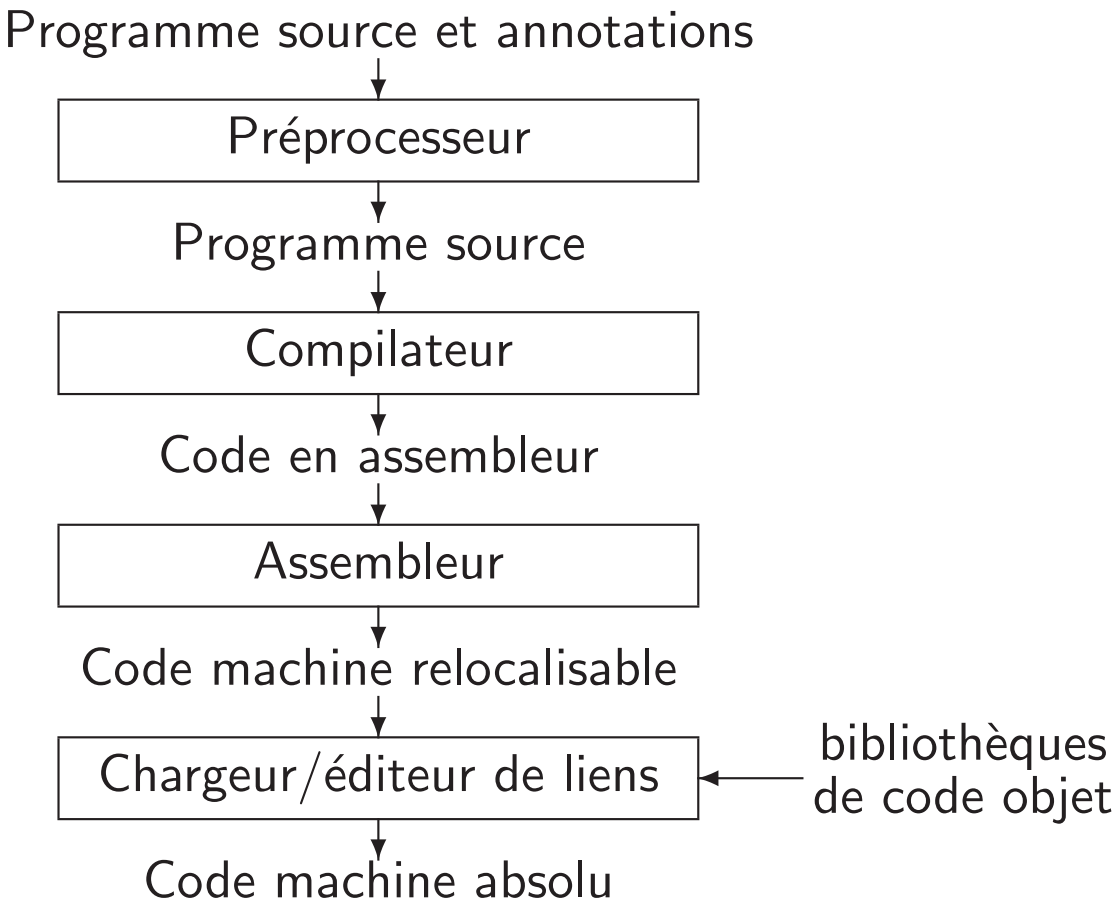
- Éditeurs structurés
- Enjoliveurs de texte et de programmes (*pretty-printers*)
- Vérificateurs statiques de programmes
- Traitement de texte
- Compilateurs vers le matériel
- Interprètes de requêtes

Outils cousins du compilateur

- Préprocesseurs
- Assembleurs
- Chargeurs de programmes et éditeurs de lien

Contexte du compilateur

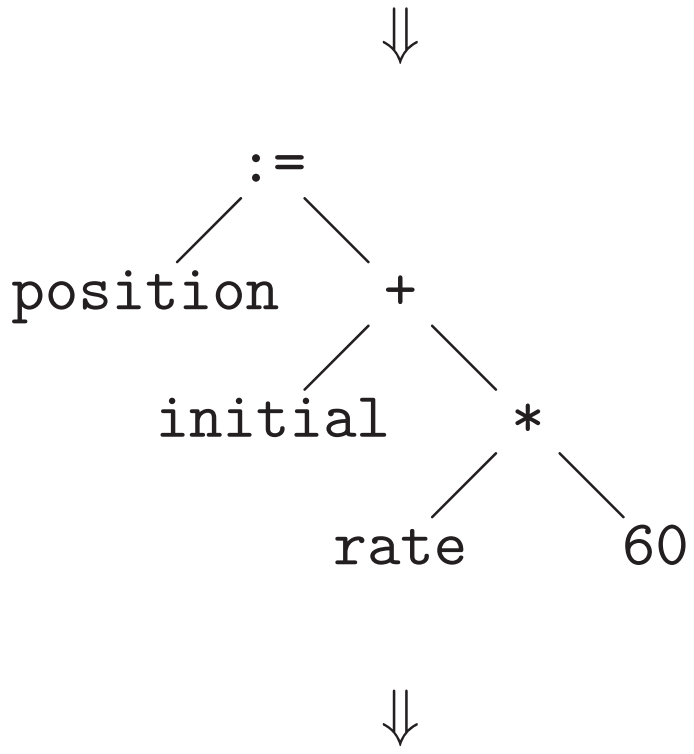
Organisation de l'implantation d'un langage de programmation:



Les phases de l'analyse d'un programme

Modèle analyse-synthèse de la compilation

```
position := initial + rate * 60
```



code généré

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 réflète 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'énoncé:

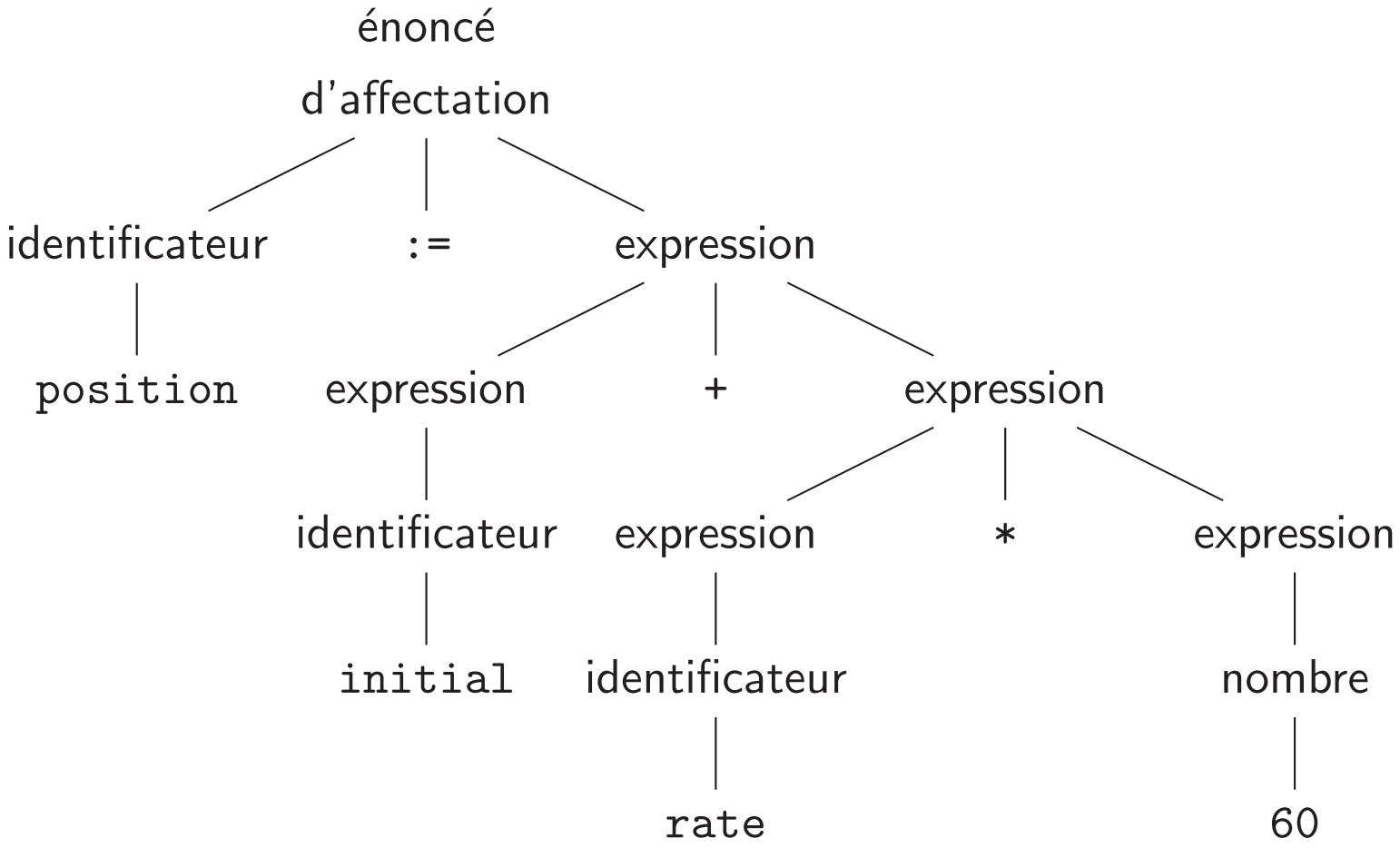
```
position := initial + rate * 60
```

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 énoncé est quelque chose comme:



Analyse syntaxique

Dans le cas de notre exemple d'affectation, la syntaxe des expressions pourrait être donnée à l'aide de règles telles que:

1. Tout identificateur est une expression.
2. Tout nombre est une expression.
3. Si e_1 et e_2 sont des expressions, alors les constructions suivantes en sont aussi:

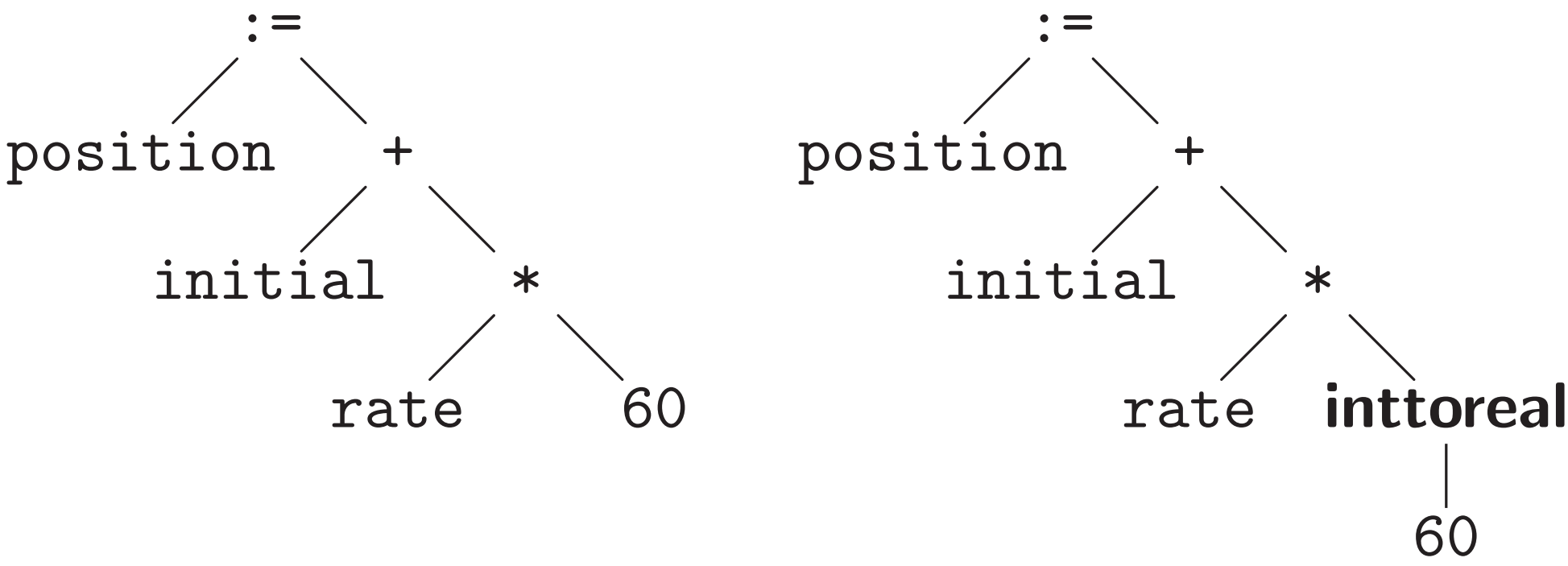
   e_1 + e_2

   e_1 * e_2

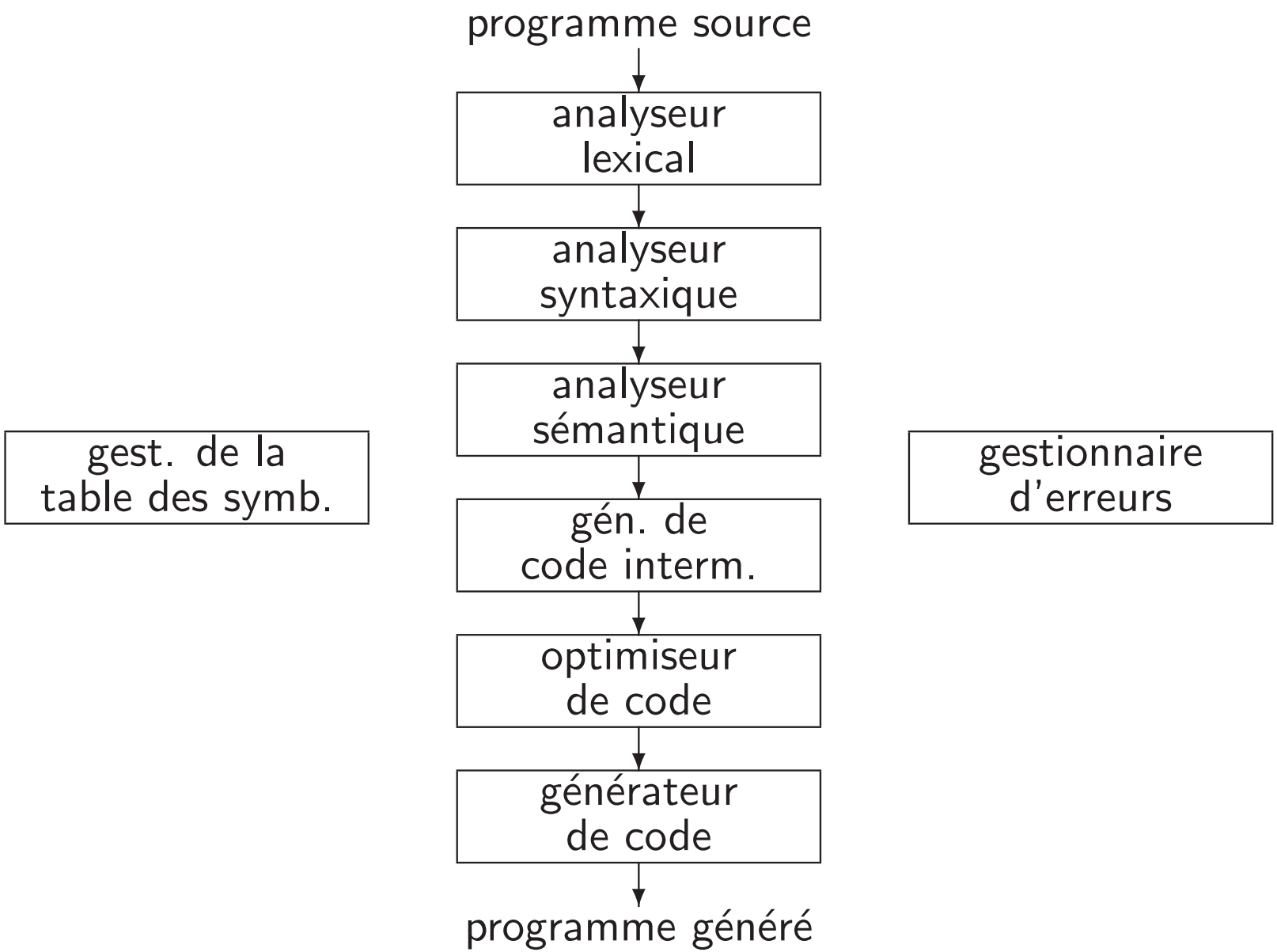
   (e_1)

Analyse sémantique

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

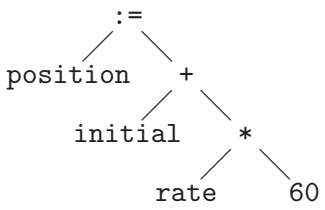


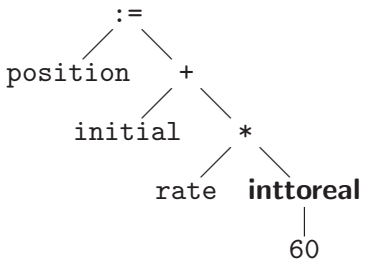
Les phases d'un compilateur



Les phases d'un compilateur

Exemple de traduction d'un énoncé:

1. `position := initial + rate * 60`
2. $\texttt{id}_1$ `:=` $\texttt{id}_2$ `+` $\texttt{id}_3$ `* 60`
3. 

4. 

5.
```
temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```
6.
```
temp1 := id3 * 60.0
id1 := id2 + temp1
```
7.
```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

où la table des symboles indique:

(a) `position` ...

(b) `initial` ...

(c) `rate` ...

(d) ...

Outils de construction de compilateurs

- Générateurs d'analyseurs lexicaux
- Générateurs d'analyseurs syntaxiques
- Outils de traduction orientée-syntaxe
- Engins de flôt de données
- Générateurs de générateurs de code

Évolution des langages de programmation

Les langages peuvent être classés par générations.

- Première génération: langages machine
- Deuxième génération: langages assembleurs
- Troisième génération: langages de plus haut niveau comme Fortran, C ou Lisp
- Quatrième génération: langages à applications spécifiques comme SQL ou Postscript
- Cinquième génération: langages de programmation logique ou par contraintes comme Prolog

Évolution des langages de programmation

Les langages peuvent aussi être classés par genres.

- Impératifs vs. déclaratifs
- De genre von Neumann
- Orientés-objets
- Langages de scriptage