

Chapitre 2 : grammaire et compilation

Notion de grammaire formelle

Analyse lexicale

Analyse syntaxique

Analyse sémantique

Premier programme Java

```
public class Bienvenue{  
    // l'exécution du programme commence avec cette méthode  
    public static void main( String[] args ){  
        System.out.println( "mon premier programme java" );  
    } // fin de main  
} // fin de la classe Bienvenue
```

Ce programme affiche la chaîne de caractères :

```
mon premier programme java  
sur la console
```

- Chaque méthode, chaque classe doit commencer avec un commentaire
- Les identificateurs de classe commencent avec une majuscule (convention)
- Sensibilité à la casse. Minuscules et majuscules sont différenciées

Premier programme Java

```
import java.util.Scanner;
public class Bienvenue2{
    // l'exécution du programme commence avec cette méthode
    public static void main( String[] args ){
        Scanner in = new Scanner( System.in );
        String nom = in.next();
        System.out.println( "Bienvenue Mr "+nom );
    } // fin de main
} // fin de la classe Bienvenue2
```

Le clavier

Premier programme Java

```
(1) import java.util.Scanner;  
(2) public class Exemple{  
(3)   public static void main( String[] args ){  
(4)     Scanner in = new Scanner( System.in );  
(5)     String nom = in.next();  
(6)     System.out.println( "mon nom : " + nom );  
(7)     int maNote = in.nextInt();  
(8)     System.out.println( "ma note :" + maNote );  
(9)     maNote = maNote +2;  
(10)    System.out.println( "Nouvelle note :" + maNote );  
(11)  }  
(12) }
```

Premier exemple de programme Java (2/4)

La structure de ce programme Java fait apparaître 3 parties distinctes :

1. la première permet d'importer des objets externes (`import ...`)
2. la seconde correspond à la création d'une classe principale contenant la méthode `main`, point de départ de l'exécution du programme (`class Exemple{...}`)
3. la troisième est une suite d'instructions formant le corps de la méthode `main({...})`

Commentaires (3/4)

- (1) importation de la classe `Scanner` du package `java.util` donnant accès à la possibilité de saisies au clavier
- (2) déclaration de la classe `Exemple`
- (3) déclaration de la méthode `main` prenant en paramètres un tableau de valeurs de type chaîne de caractères
- (4) déclaration de la variable appelée `in` à laquelle on affecte un objet construit à partir de la classe `Scanner` sur laquelle une saisie de données pourra être réellement effectuée. On note que cet objet est associé, par construction, à l'objet `System.in` qui représente le clavier
- (5) déclaration de la variable `nom` qui ne pourra contenir que du texte (chaîne de caractères) car elle est associée au type (classe) `String`. La valeur initiale de cette variable est produite par le message `in.next()` qui a pour rôle de récupérer la valeur saisie par l'utilisateur.

Commentaires (4/4)

- (6) `System.out` est un objet java qui représente la console sur laquelle on affiche un résultat; ici le texte `"mon nom : "` suivi du contenu de la variable `nom`
- (7) déclaration de la variable `maNote` de type primitif `int` et initialisation de celle-ci avec la valeur entière saisie au clavier
- (1) affichage sur l'objet `System.out` du texte `"ma note :"` suivi de la valeur de la variable `maNote`
- (2) modification de la valeur contenue dans la variable `maNote`. La nouvelle valeur résulte de la somme de l'ancienne valeur augmentée de 2.
- (3) affichage d'un nouveau texte reflétant la nouvelle valeur de `maNote`

Remarques

- Le code à exécuter forme une suite d'ordres séparées par des ;
- L'exécution se fait séquentiellement. On exécute le premier ordre, puis le second et ainsi de suite
- 3 catégories d'ordre :
 - **les déclarations de variables** : permettent de donner un nom à une case de la mémoire dans laquelle stocker une valeur pendant le temps de l'exécution du programme
 - **les instructions d'entrée-sortie** : permettent de faire entrer dans le programme des données extérieures et/ou de fournir des résultats produits par l'exécution du programme
 - **l'instruction d'affectation** : permettent de modifier les variables déclarées

Entités Java manipulées dans un programme

- Mots réservés : `class, public, static, import, ...`
- Identificateurs : `z, _x5, Hello`
- Littéraux : `"Oui", "2005", 0.001, 'x', true`
- Opérateurs : `+, -, *, /, &, ||, <, >, <=, ==, >=, ...`
- Types primitifs : `int, long, char, boolean, float, double, byte, short`

Notion de grammaire formelle

Une *grammaire* est définie comme un quadruplet :

$$\langle A, VT, V, P \rangle$$

A : point d'entrée

VT : vocabulaire terminal (+, -, 0, 1, _, ...)

V : vocabulaire non terminal
(ident, lettre, symbole, ...)

P : règles de productions

V ::= suite de terminaux et non terminaux

Le formalisme BNF étendu

symbole	signification	exemple
[]	partie facultative	[+]
	alternative	+ -
...	intervalle	0...9
{ }	répétition 0 ou n fois	{lettre}
{ } ⁺	répétition stricte (1 à n fois)	{lettre} ⁺
()	mise en facteurs	(+ -)nombre

Analyse lexicale

- Transformation de la suite de caractères en une suite de mots du langage
- Les mots du langage sont :
 - les identificateurs : `x_11`, `VAR`, `toto`
 - les constantes numériques : `11`, `42.21f`
 - les mots clés : `class`, `public`, `import`, `{`, `}`...
 - les opérateurs : `+`, `-`, `*`, `/`, ...
- Cette transformation est basée sur un ensemble de règles lexicales

Règles de construction d'un identificateur Java (1/2)

`<ident> ::= <lettre> { <symbole> }`

`<symbole> ::= <lettre> | <chiffre>`

`<lettre> ::= A...Z | a...z | _ | $ | £ | € | ¤`

`<chiffre> ::= 0...9`

Exemples : valides ou non valides ?

`X_1, A_, _A, 1_JAVA, $TINTIN_é_MILOU, O__K,`
`un$, A+B, €OK, #DO, DO#`

Analyse syntaxique

- Transformation d'une suite de symboles (mots) en une suite de phrases du langage
- Cette transformation est fondée sur un ensemble de règles syntaxiques décrivant les structures syntaxiques du langage et exprimées dans le formalisme BNF (par exemple)
- Pour éviter les ambiguïtés d'interprétation des phrases du langage, à une suite donnée de symboles ne peut correspondre qu'une seule phrase du langage

Structure syntaxique des expressions arithmétiques

Les éléments non terminaux de la grammaire sont notés entre < et >

```
<expression> ::= <facteur> |  
    <expression><opadd><facteur>  
<facteur> ::= <terme> | <facteur><opmult><terme>  
<terme> ::= <ident> | <nombre> | (<expression>)  
<opadd> ::= + | -  
<opmult> ::= * | /  
<nombre> ::= {<chiffre>}+  
<chiffre> ::= 0..9
```

Analyse syntaxique de : **7**+3* (34-6)



<expression>



<facteur>



<terme>



<ident>



7 <ident>

échec =>
 retour vers
 <terme>

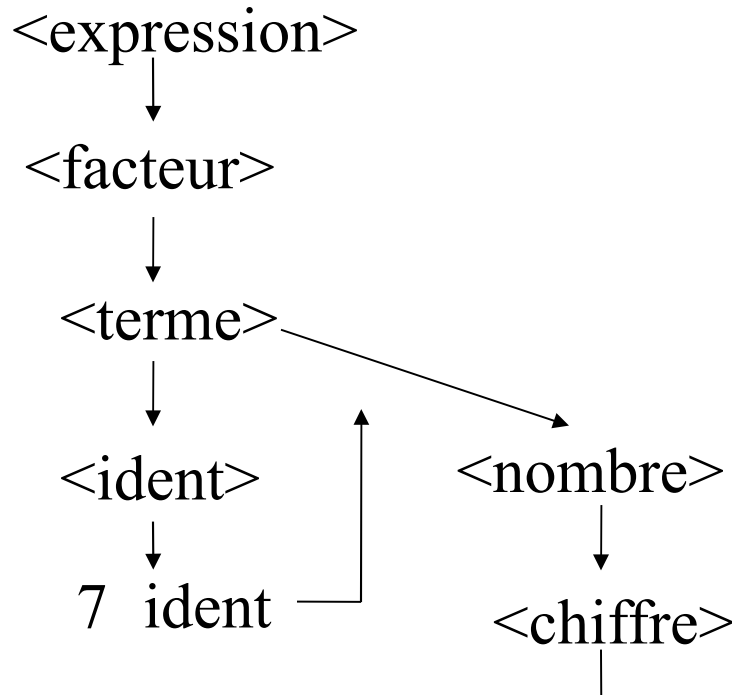
<expression> ::= <facteur> |

<expression><opadd><facteur>

<facteur> ::= <terme> | <facteur><opmult><terme>

<terme> ::= <ident> | <nombre> | (<expression>)

Analyse syntaxique de : **7**+3* (34-6)



échec =>
retour vers
<terme>

7 => analyse terminée
mais expression non reconnue

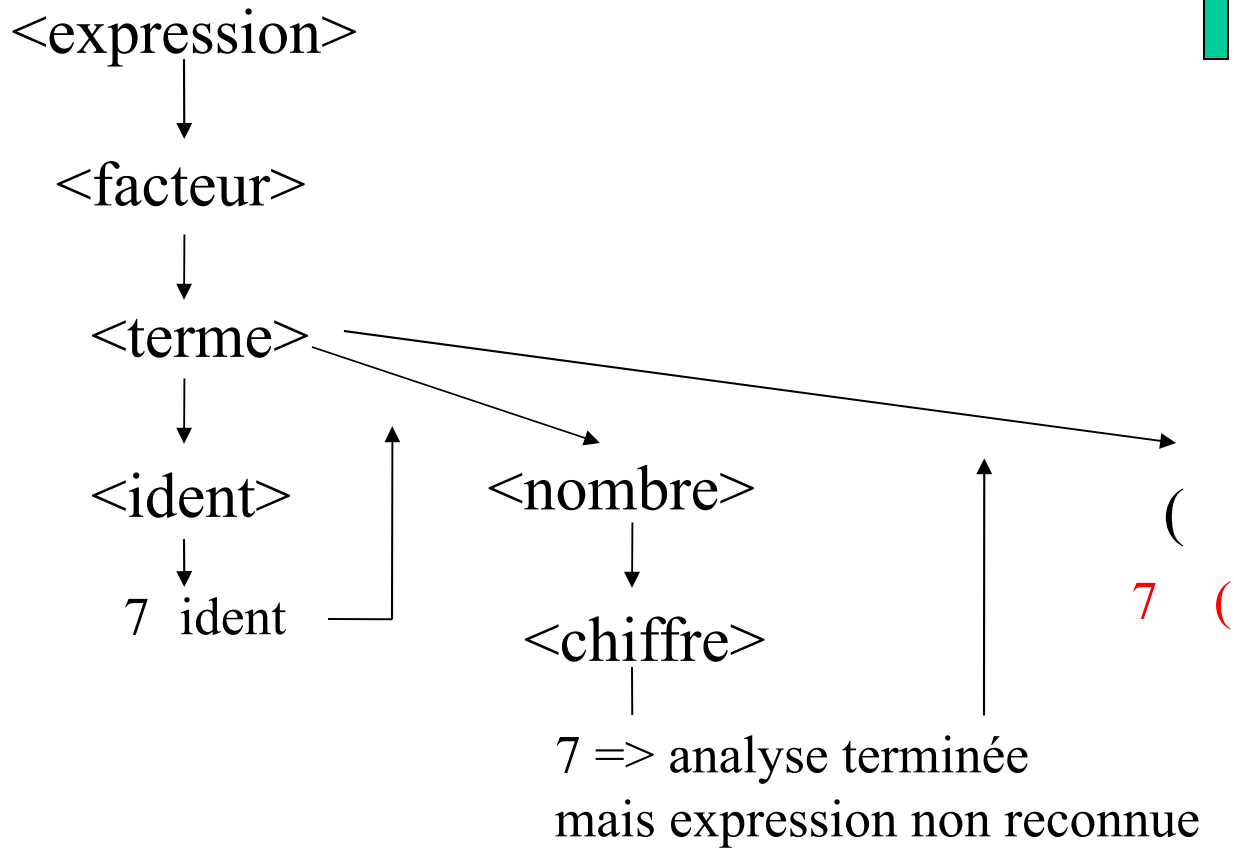
$\langle \text{expression} \rangle ::= \langle \text{facteur} \rangle \mid \langle \text{expression} \rangle \langle \text{opadd} \rangle \langle \text{facteur} \rangle$

$\langle \text{facteur} \rangle ::= \langle \text{terme} \rangle \mid \langle \text{facteur} \rangle \langle \text{opmult} \rangle \langle \text{terme} \rangle$

$\langle \text{terme} \rangle ::= \langle \text{ident} \rangle \mid \langle \text{nombre} \rangle \mid (\langle \text{expression} \rangle)$

$\langle \text{nombre} \rangle ::= \{ \langle \text{chiffre} \rangle \}^+$

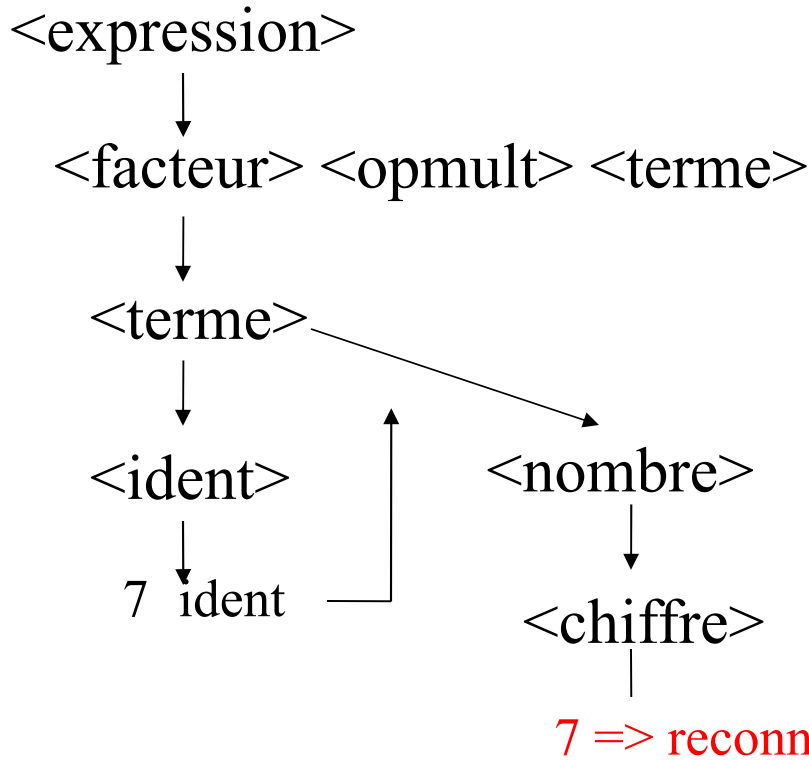
Analyse syntaxique de : $7 + 3 * (34 - 6)$



échec =>
retour vers
<terme>
puis vers
<facteur>

$\langle \text{expression} \rangle ::= \langle \text{facteur} \rangle \mid \langle \text{expression} \rangle \langle \text{opadd} \rangle \langle \text{facteur} \rangle$
 $\langle \text{facteur} \rangle ::= \langle \text{terme} \rangle \mid \langle \text{facteur} \rangle \langle \text{opmult} \rangle \langle \text{terme} \rangle$
 $\langle \text{terme} \rangle ::= \langle \text{ident} \rangle \mid \langle \text{nombre} \rangle \mid (\langle \text{expression} \rangle)$

Analyse syntaxique de : **7**+3* (34-6)



le curseur se positionne sur le symbole suivant

$\langle \text{expression} \rangle ::= \langle \text{facteur} \rangle \mid \langle \text{expression} \rangle \langle \text{opadd} \rangle \langle \text{facteur} \rangle$
 $\langle \text{facteur} \rangle ::= \langle \text{terme} \rangle \mid \langle \text{facteur} \rangle \langle \text{opmult} \rangle \langle \text{terme} \rangle$
 $\langle \text{terme} \rangle ::= \langle \text{ident} \rangle \mid \langle \text{nombre} \rangle \mid (\langle \text{expression} \rangle)$

Analyse syntaxique de : $7 + 3 * (34 - 6)$



<expression>



<facteur> <opmult> <terme>



+ *

puis + /

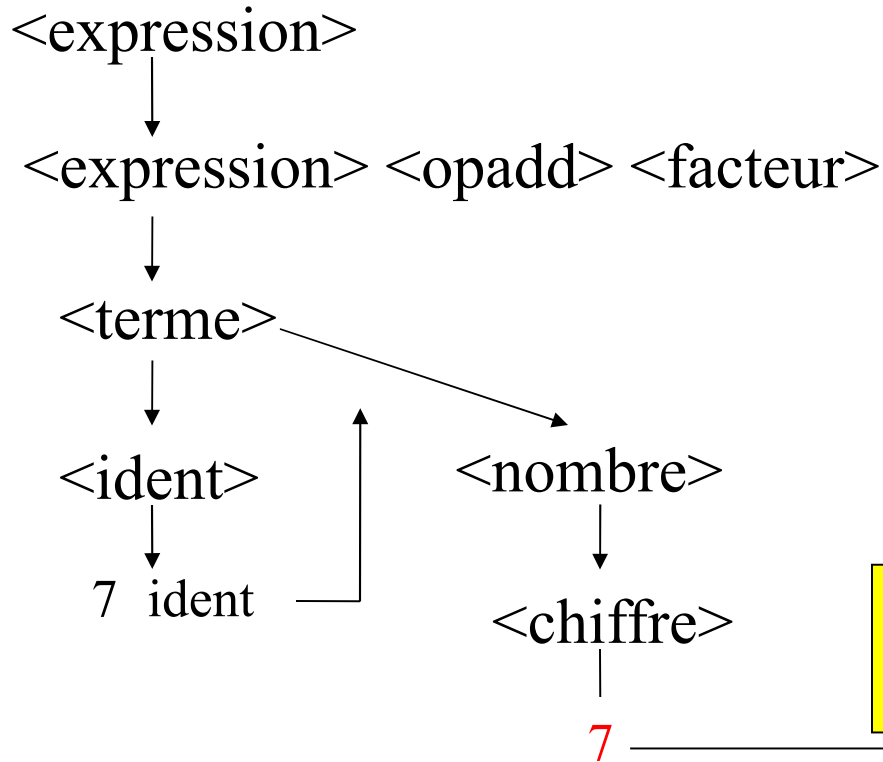
échec => retour
vers
<expression>

<expression> ::= <facteur> | <expression> <opadd> <facteur>

<facteur> ::= <terme> | <facteur> <opmult> <terme>

<opmult> ::= * | /

Analyse syntaxique de : **7**+3*(34-6)



succès => le curseur se déplace

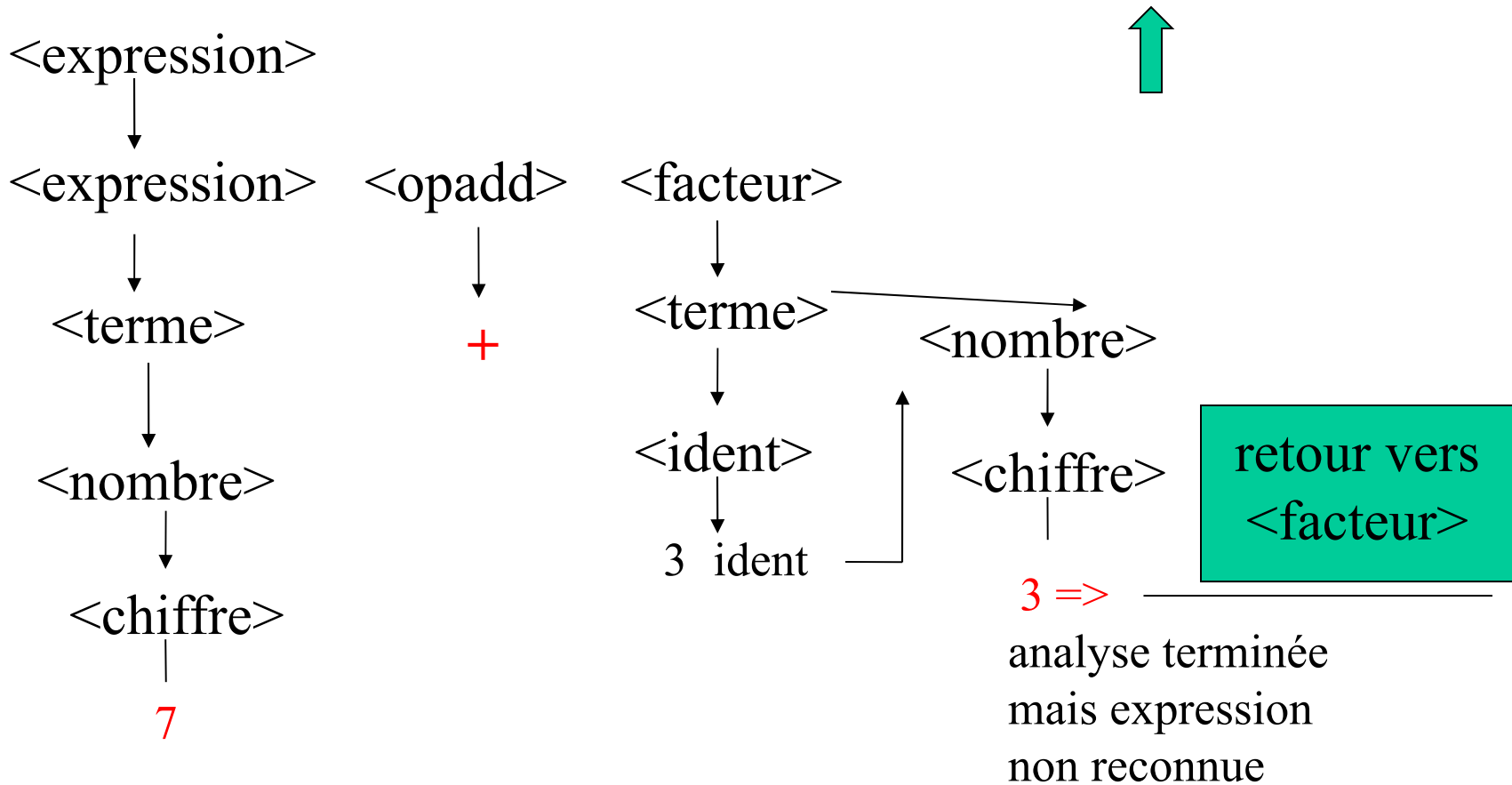
$\langle \text{expression} \rangle ::= \langle \text{facteur} \rangle \mid \langle \text{expression} \rangle \langle \text{opadd} \rangle \langle \text{facteur} \rangle$

$\langle \text{facteur} \rangle ::= \langle \text{terme} \rangle \mid \langle \text{facteur} \rangle \langle \text{opmult} \rangle \langle \text{terme} \rangle$

$\langle \text{terme} \rangle ::= \langle \text{ident} \rangle \mid \langle \text{nombre} \rangle \mid (\langle \text{expression} \rangle)$

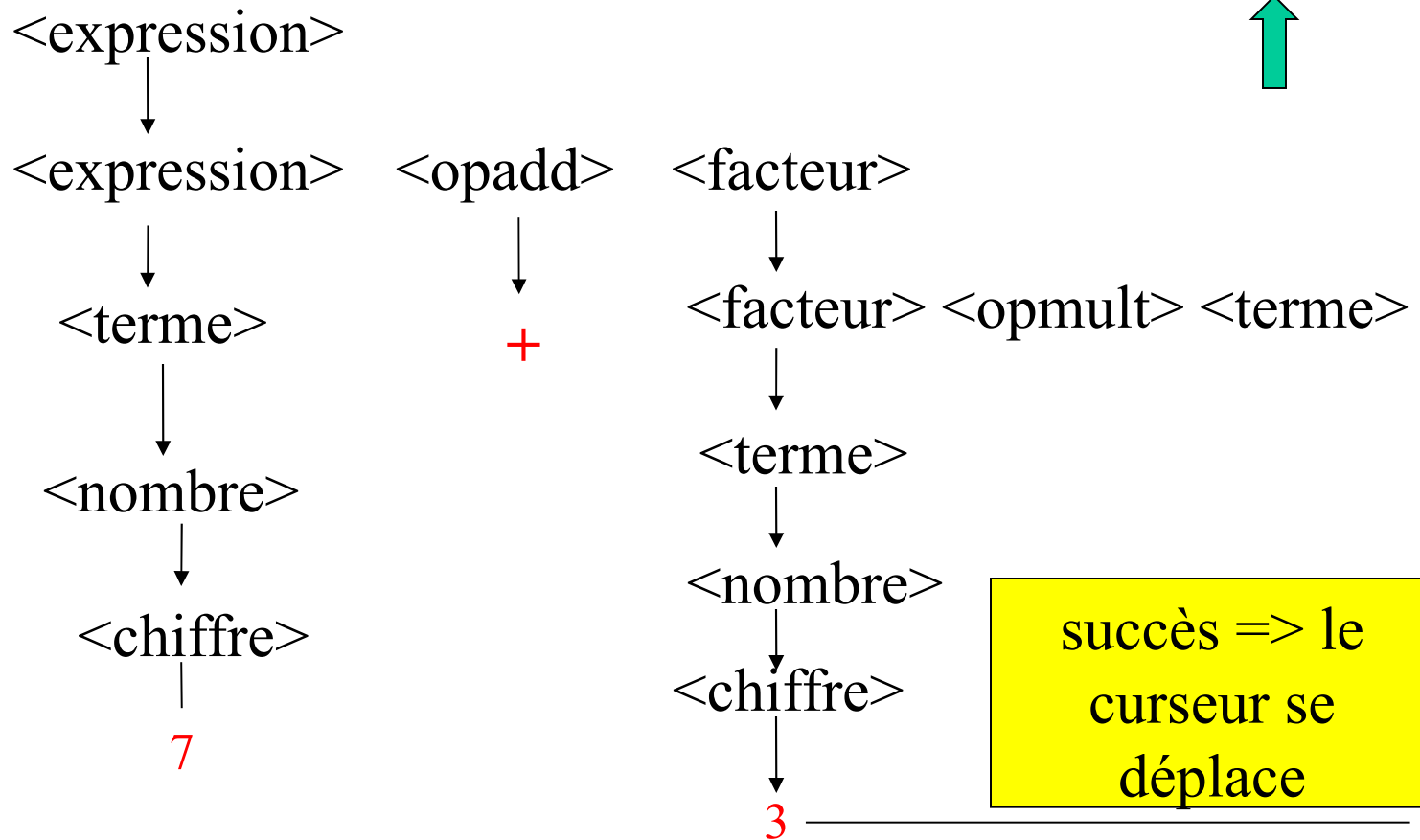
$\langle \text{nombre} \rangle ::= \{ \langle \text{chiffre} \rangle \}^+$

Analyse syntaxique de : $7 + 3 * (34 - 6)$



- <expression> ::= <facteur> | <expression><opadd><facteur>
- <facteur> ::= <terme> | <facteur><opmult><terme>
- <terme> ::= <ident> | <nombre> | (<expression>)
- <nombre> ::= {<chiffre>}⁺

Analyse syntaxique de : 7+3*(34-6)



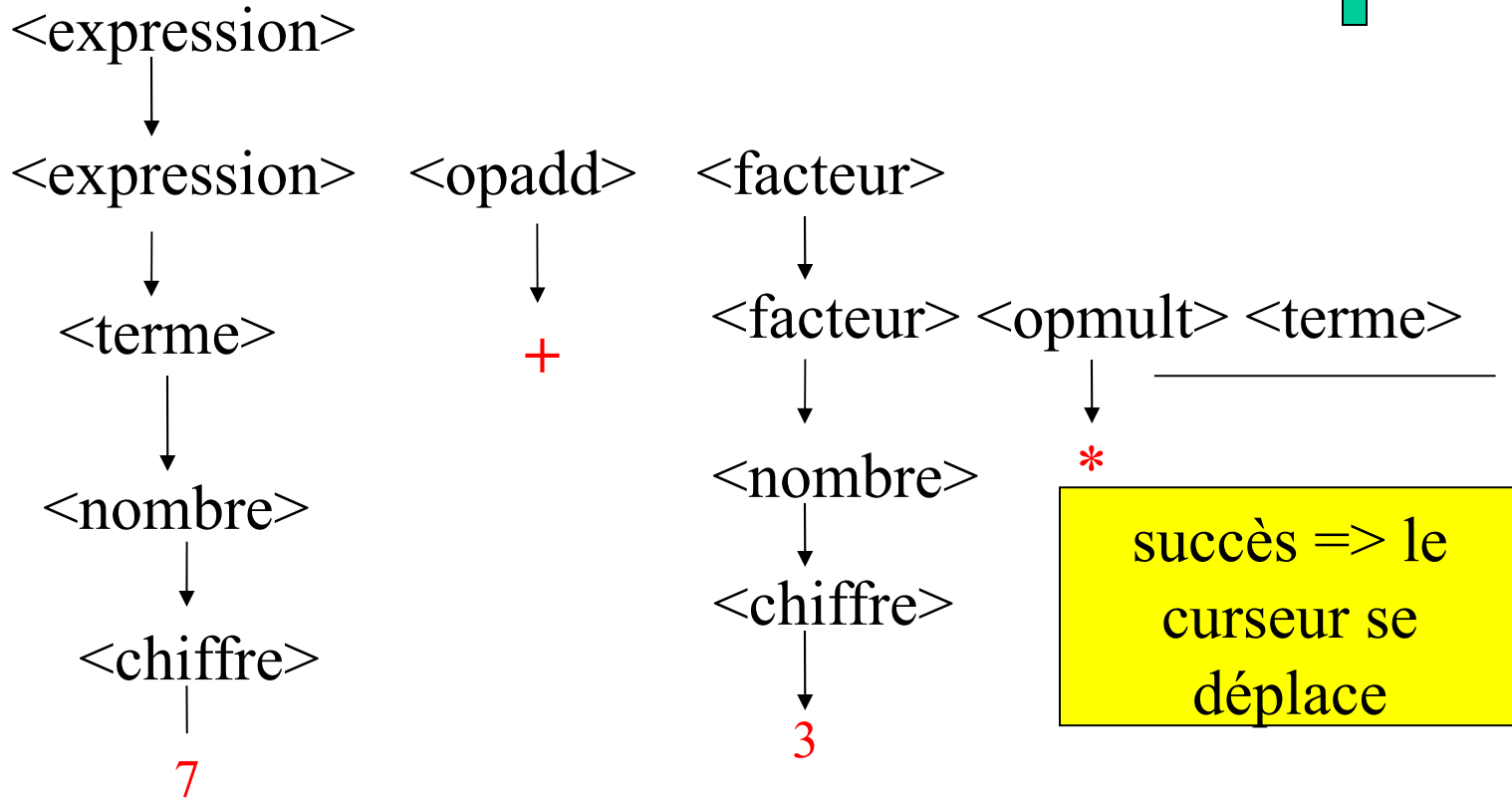
<expression> ::= <facteur> | <expression><opadd><facteur>

<facteur> ::= <terme> | <facteur><opmult><terme>

<terme> ::= <ident> | <nombre> | (<expression>)

<nombre> ::= {<chiffre>}⁺

Analyse syntaxique de : 7+3*(34-6)

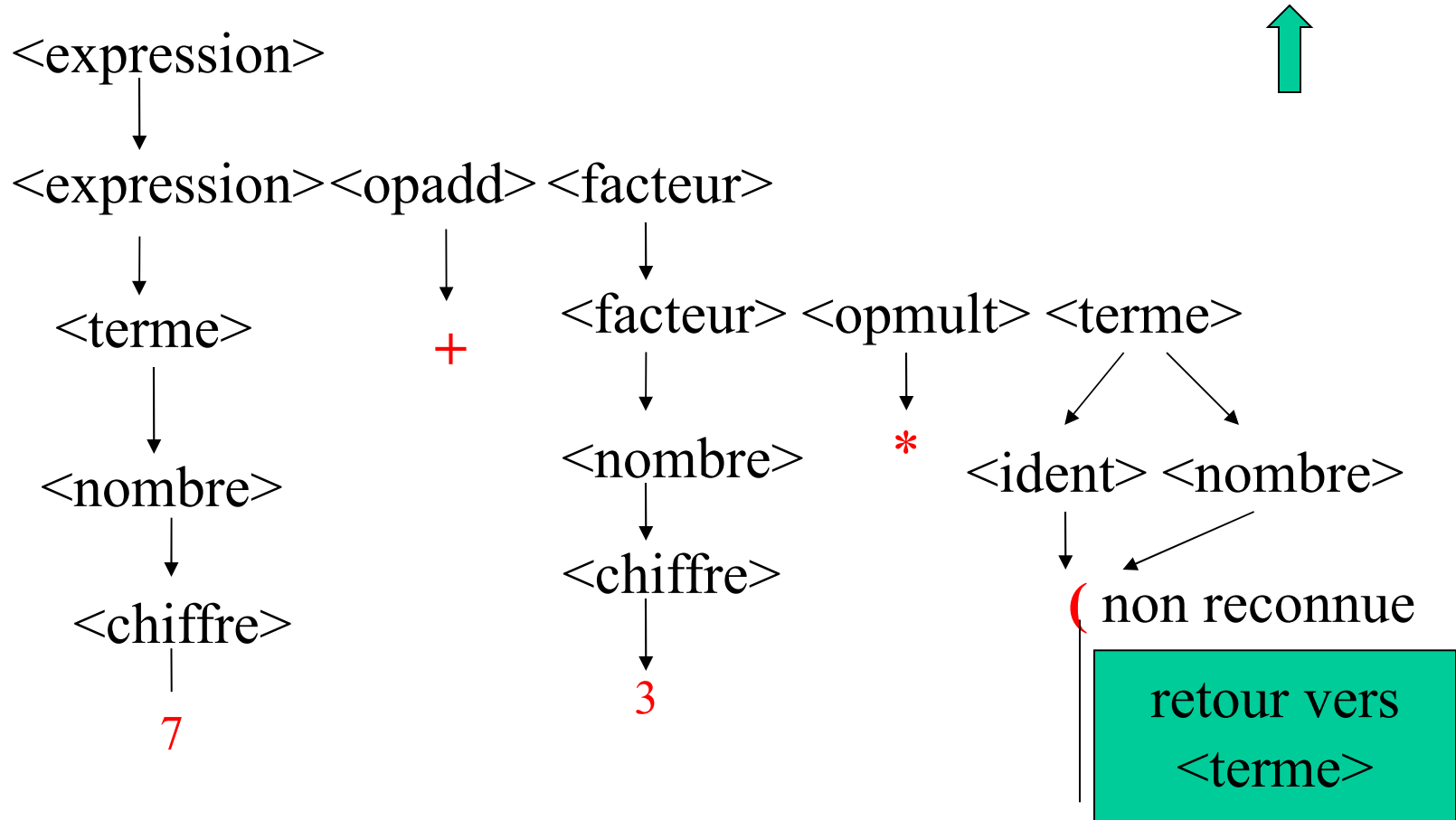


$\langle \text{expression} \rangle ::= \langle \text{facteur} \rangle \mid \langle \text{expression} \rangle \langle \text{opadd} \rangle \langle \text{facteur} \rangle$

$\langle \text{facteur} \rangle ::= \langle \text{terme} \rangle \mid \langle \text{facteur} \rangle \langle \text{opmult} \rangle \langle \text{terme} \rangle$

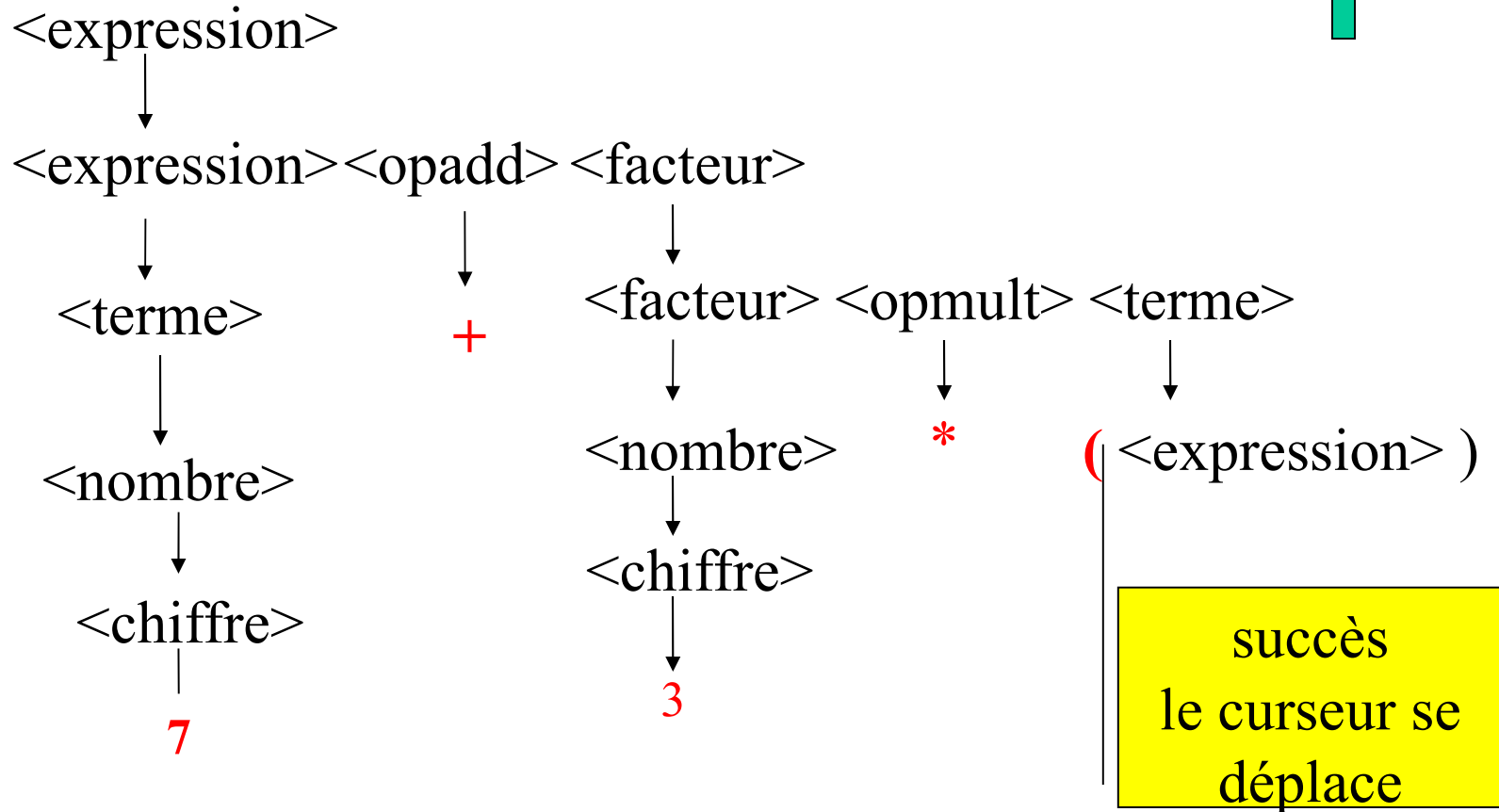
$\langle \text{opmult} \rangle ::= * \mid /$

Analyse syntaxique de : $7+3*(34-6)$



- <expression> ::= <facteur> | <expression><opadd><facteur>
- <facteur> ::= <terme> | <facteur><opmult><terme>
- <terme> ::= <ident> | <nombre> | (<expression>)

Analyse syntaxique de : $7+3*(34-6)$

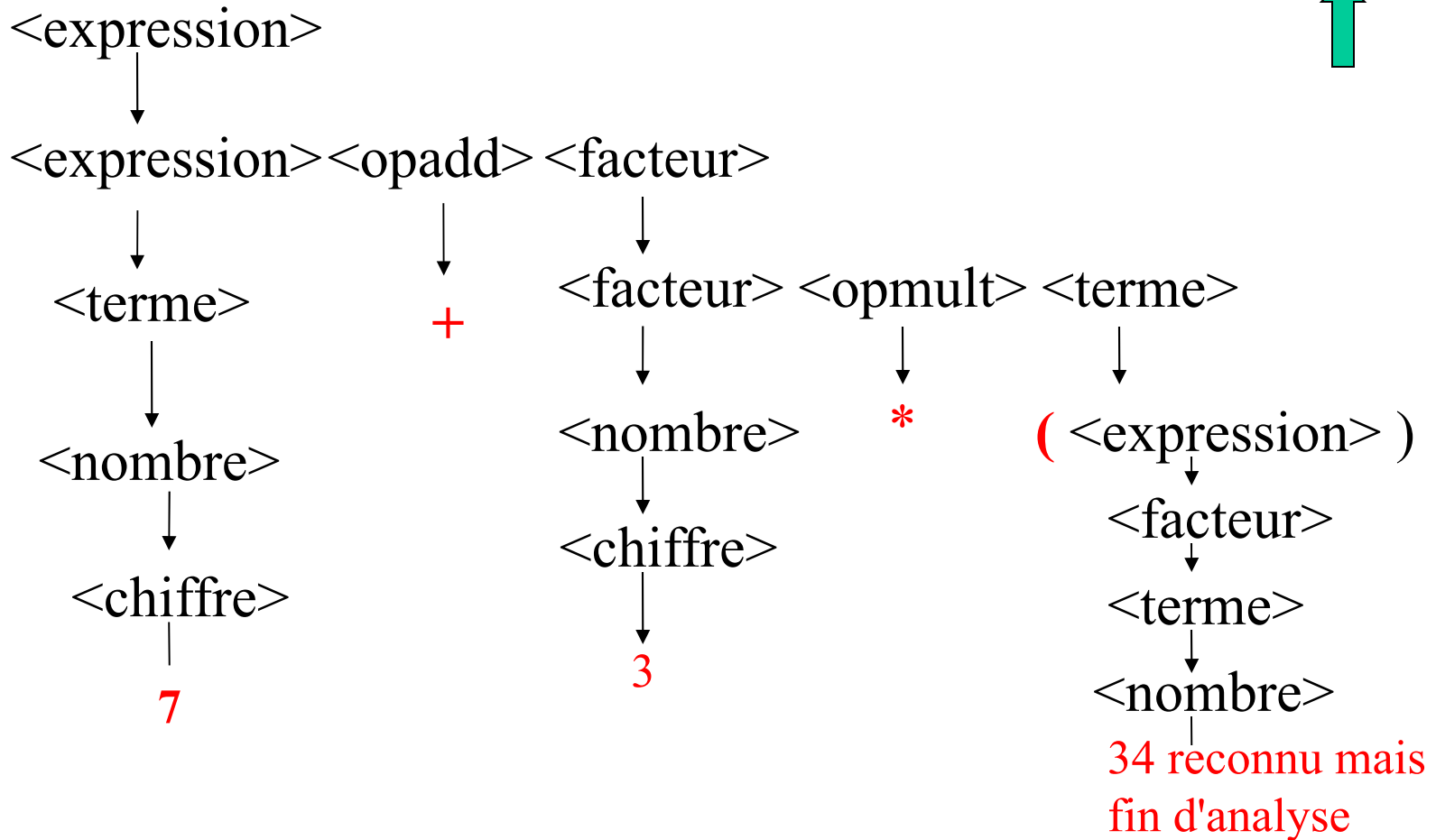


$\langle \text{expression} \rangle ::= \langle \text{facteur} \rangle \mid \langle \text{expression} \rangle \langle \text{opadd} \rangle \langle \text{facteur} \rangle$

$\langle \text{facteur} \rangle ::= \langle \text{terme} \rangle \mid \langle \text{facteur} \rangle \langle \text{opmult} \rangle \langle \text{terme} \rangle$

$\langle \text{terme} \rangle ::= \langle \text{ident} \rangle \mid \langle \text{nombre} \rangle \mid (\langle \text{expression} \rangle)$

Analyse syntaxique de : $7+3*(34-6)$

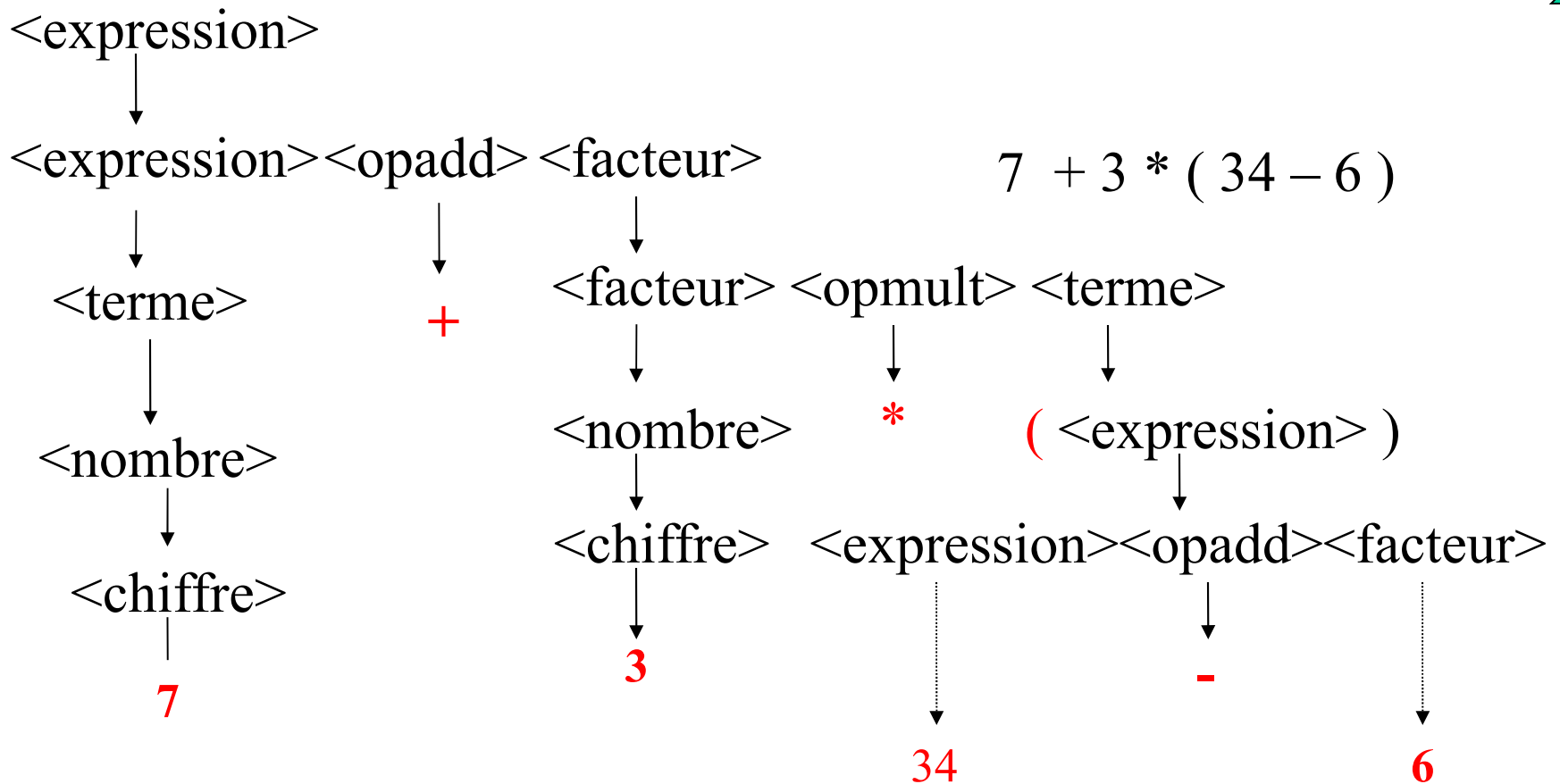


$\langle \text{expression} \rangle ::= \langle \text{facteur} \rangle \mid \langle \text{expression} \rangle \langle \text{opadd} \rangle \langle \text{facteur} \rangle$

$\langle \text{facteur} \rangle ::= \langle \text{terme} \rangle \mid \langle \text{facteur} \rangle \langle \text{opmult} \rangle \langle \text{terme} \rangle$

$\langle \text{terme} \rangle ::= \langle \text{ident} \rangle \mid \langle \text{nombre} \rangle \mid (\langle \text{expression} \rangle)$

Analyse syntaxique de : $7 + 3 * (34 - 6)$

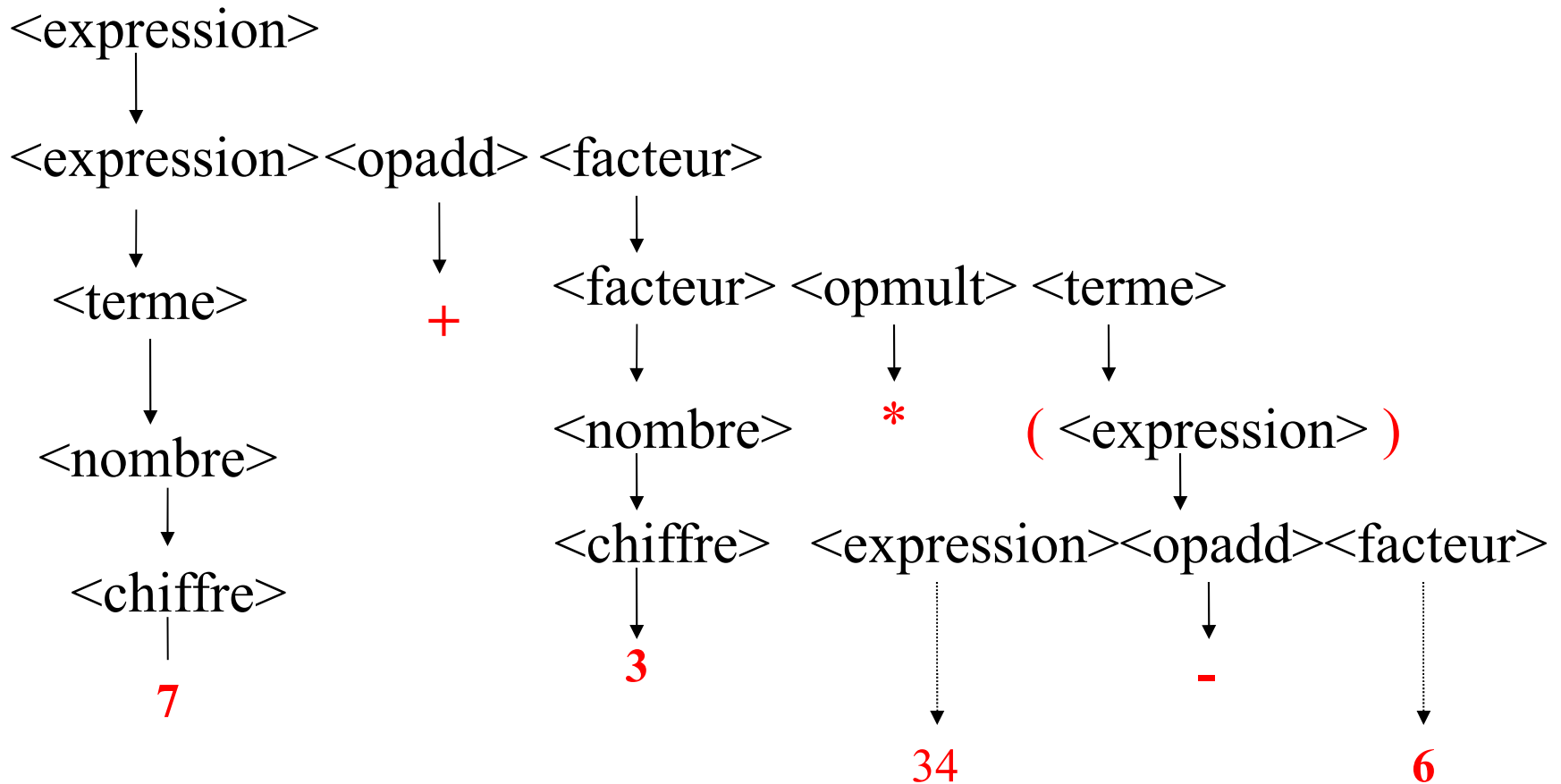


$\langle \text{expression} \rangle ::= \langle \text{facteur} \rangle \mid \langle \text{expression} \rangle \langle \text{opadd} \rangle \langle \text{facteur} \rangle$

$\langle \text{facteur} \rangle ::= \langle \text{terme} \rangle \mid \langle \text{facteur} \rangle \langle \text{opmult} \rangle \langle \text{terme} \rangle$

$\langle \text{terme} \rangle ::= \langle \text{ident} \rangle \mid \langle \text{nombre} \rangle \mid (\langle \text{expression} \rangle)$

Analyse syntaxique de : $7+3*(34-6)$



$\langle \text{expression} \rangle ::= \langle \text{facteur} \rangle \mid \langle \text{expression} \rangle \langle \text{opadd} \rangle \langle \text{facteur} \rangle$

$\langle \text{facteur} \rangle ::= \langle \text{terme} \rangle \mid \langle \text{facteur} \rangle \langle \text{opmult} \rangle \langle \text{terme} \rangle$

$\langle \text{terme} \rangle ::= \langle \text{ident} \rangle \mid \langle \text{nombre} \rangle \mid (\langle \text{expression} \rangle)$

Analyse syntaxique de :

$$7+3*(34-6)$$

L'analyse de <expression> est terminée sans échec

Tous les symboles ont été analysés et reconnus

La lecture de gauche à droite des feuilles de l'arbre correspond à l'expression à analyser

Analyse sémantique

Vérifie que le type d'une construction syntaxique correspond au type attendu

Exemple

L'opérateur `%` est une fonction qui, à 2 entiers, fait correspondre un entier.

Type de `%` : `int*int->int`

Elle nécessite donc 2 opérandes d'un type entier. Le contrôle de type vérifie que c'est bien le cas. Il vérifie aussi que le résultat attendu est lui-même du type entier.

Génération du code objet

- Le bytecode est un code objet produit à la suite des phases antérieures de compilation à partir du code source et de la table des symboles
- Il reviendra à l'interpréteur d'exécuter ce code sur la machine hôte

Edition de liens

- Réunit les différents modules objets (classes) dans un même espace d'adressage (une même image mémoire)
- Permet l'utilisation, dans un programme, de modules précompilés (classes présentes en bibliothèque)

Détection des erreurs (1/3)

```
import java.util.Scanner;
public class Exemple1{
    public static void main( String[] args ){
        Scanner in = new Scanner( System.in );
        String nom& = in.next();
        System.out.println( "mon nom : " + nom& );
        char maNote = in.nextInt();
        System.out.println( "ma note :" + maNote )
        maNote = maNote + 2;
        System.out.println( "Nouvelle note :" + maNote );
    }
}
```

Erreurs lexicale, syntaxique, sémantique ?

Détection des erreurs (2/3)

`nom&` n'est pas un identificateur valide => **erreur lexicale**

```
System.out.println( "ma note :" + maNote )
```

toute instruction doit se terminer par un `;` => **erreur syntaxique**

```
char maNote = in.nextInt();
```

les types des expressions de chaque côté du symbole de l'affectation (`=`) sont différents => **erreur sémantique**

Détection des erreurs (3/3)

```
public class Exemple2{
    public static void main ( String[] args ) {
        char a ;
        int a , bb
        char c = 'bonjour' ;
        boolean tt =
        a = 2;
        bb = x + "x";
        tt=x+4;
        a = bb +1;
        bb= 3 * tt ;
        bb+1 = 3 ;
        bb==2;
    }
}
```