Syntaxe

Vincent Archambault-Bouffard
IFT 2035 - Université de Montréal

Pour obtenir le code source de ce document

- https://github.com/archambaultv/IFT2035-UdeM
- vincent.archambault-bouffard@umontreal.ca

Plan du cours

- Syntaxe et sémantique
- Syntaxe infixe, préfixe, postfixe
- Arbre de syntaxe abstraite
- Grammaire BNF et EBNF

Syntaxe et sémantique

Syntaxe	Représentation du programme
Sémantique	Sens attaché aux programmes
Spécification	Syntaxe + Sémantique

Syntaxe infixe, préfixe, postfixe

Syntaxe infixe

- L'opérateur au centre, les arguments de chaque côté
- Syntaxe familière, vous la connaissez depuis le primaire

```
1 + 2

1 + 2 - 3

1 + (2 - 3)
```

Syntaxe infixe

Syntaxe infixe

- Syntaxe ambiguë lorsqu'on utilise deux ou plusieurs opérateurs de suite

```
a + b - c = (a + b) - c
          ou
a + b - c = a + (b - c)


a + b * c = (a + b) * c
          ou
a + b * c = a + (b * c)
```

Syntaxe infixe

Syntaxe infixe

- Pour éliminer l’ambiguïté
  - Priorité des opérateurs
  - Associativité

Syntaxe infixe : priorité des opérateurs

- Les fonctions ont toujours priorité sur les opérateurs

```
sin a - b = (sin a) - b
```

Syntaxe infixe

Syntaxe infixe : priorité des opérateurs

- Un opérateur de plus haute priorité doit être calculé en premier

  * : priorité 7 en Haskell

  \+ : priorité 6 en Haskell

```
a + b * c = a + (b * c)
```

Syntaxe infixe

Syntaxe infixe : priorité des opérateurs

- Un opérateur de plus haute priorité doit être calculé en premier

  + : priorité 6 en Haskell

  < : priorité 5 en Haskell

```
a + b < c = (a + b) < c
```

Syntaxe infixe

Syntaxe infixe : associativité

- Lorsque deux opérateurs ont la même priorité, on utilise l'associativité pour savoir lequel doit être calculé en premier
- Il ne peut y avoir qu'une seule associativité par niveau

  + : associatif à gauche

```
a + b + c = (a + b) + c
```

Syntaxe infixe

Syntaxe infixe : associativité

- + : priorité 6 en Haskell
  associatif à gauche

  - : priorité 6 en Haskell
  associatif à gauche

```
a - b + c = (a - b) + c

a + b - c = (a + b) - c
```

Syntaxe infixe

Syntaxe infixe : associativité

- Certains opérateurs ne sont pas associatifs

  > : priorité 4 en Haskell
  non associatif en Haskell

```
a > b > c != (a > b) > c
```

Syntaxe infixe

Syntaxe préfixe

- L’opérateur en premier, les arguments ensuite
- Aucune ambiguïté

Syntaxe préfixe

```
1 + 2

1 + 2 - 3

1 + (2 - 3)
```

```
+ 1 2

- + 1 2 3

+ 1 - 2 3
```

Syntaxe infixe

Syntaxe préfixe

Syntaxe postfixe

- Les arguments en premier, l'opérateur ensuite
- Aucune ambiguïté

Syntaxe postfixe

```
1 + 2

1 + 2 - 3

1 + (2 - 3)
```

```
1 2 +

1 2 + 3 -

1 2 3 - +
```

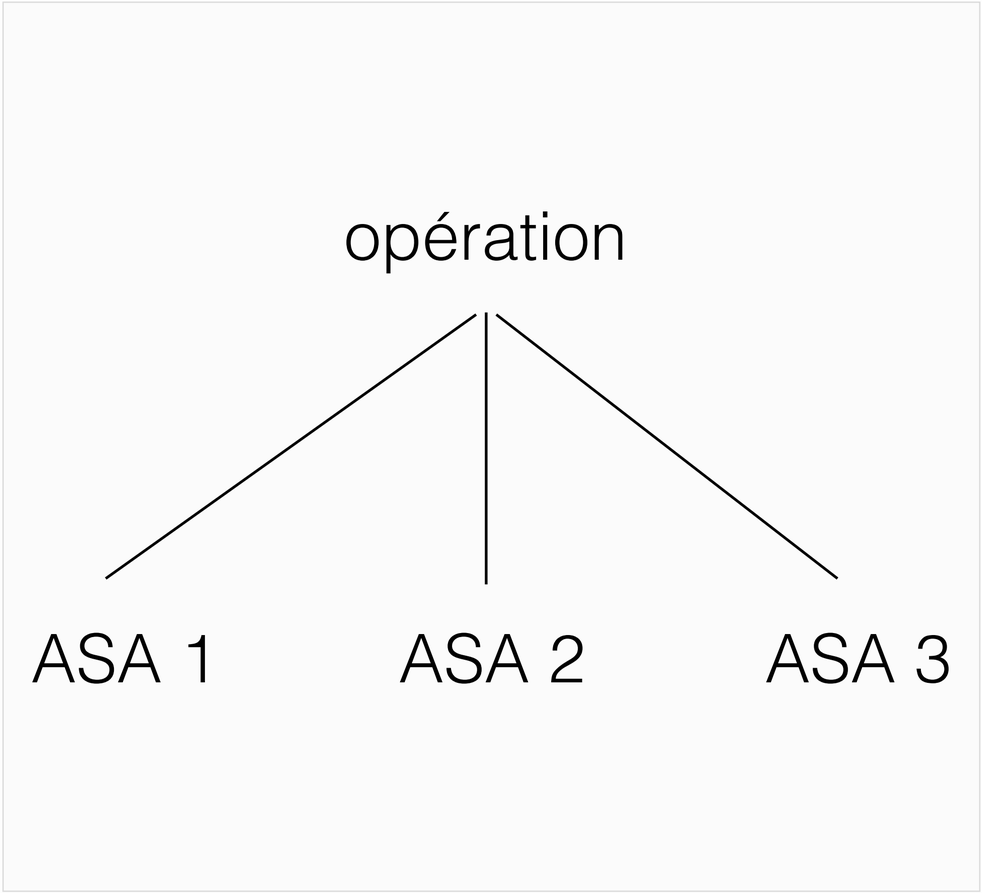
Syntaxe infixe

Syntaxe préfixe

Arbre de syntaxe abstraite

Arbre de syntaxe abstraite (ASA)

- Forme générale
  - L'opération est la racine de l'arbre (ou du sous-arbre qui lui est associé)
  - Les arguments sont les enfants



Forme générale d'une arbre de syntaxe abstraite

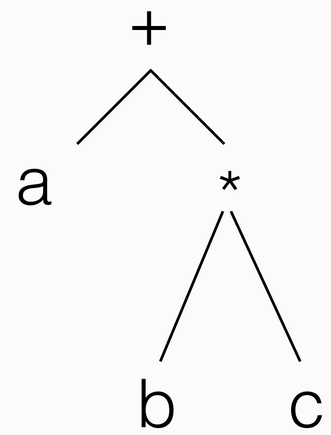
Arbre de syntaxe abstraite (ASA)

- Il est plus facile de représenter et de manipuler la syntaxe sous la forme d'un arbre
  - Pas de parenthèse et autres caractères visuels
  - Point milieu entre la syntaxe et la sémantique
  - Structure de donnée pouvant être représentée en mémoire

Arbre de syntaxe abstraite (ASA)

```
a + b * c
```

Code Haskell

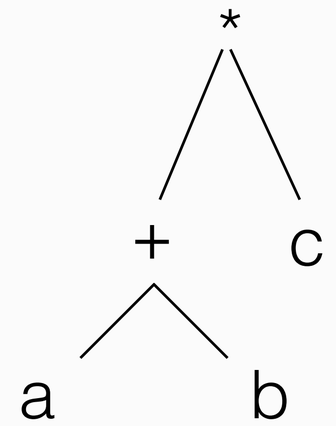


Arbre de syntaxe abstraite

Arbre de syntaxe abstraite (ASA)

```
(a + b) * c
```

Code Haskell

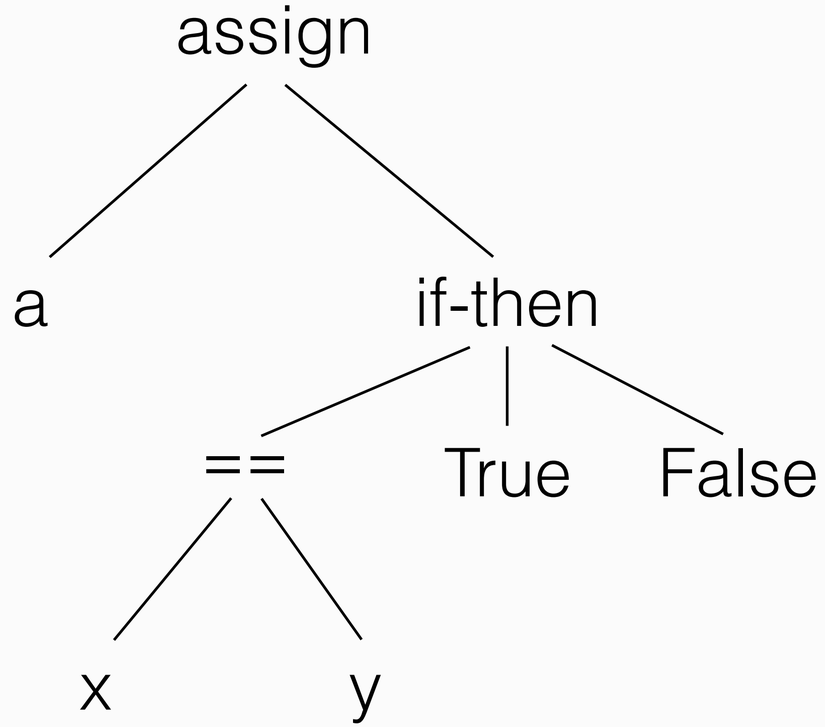


Arbre de syntaxe abstraite

Arbre de syntaxe abstraite (ASA)

```
a = if x == y
    then True
    else False
```

Code Haskell



Arbre de syntaxe abstraite

Grammaire BNF et EBNF

Grammaire

- La grammaire d'un langage de programmation :
  - Spécifie les programmes syntaxiquement valides
  - Permet une représentation des programmes sous forme d'arbre

Grammaire

Vocabulaire	Ensemble des symboles permis
Phrase	Séquence de symboles tirés du vocabulaire
Langage	Ensemble de phrases
Grammaire	Règles décrivant les phrases valides du langage

Grammaire BNF

- La grammaire BNF est une syntaxe pour exprimer des grammaires formelles

Grammaire BNF

BNF Backus-Naur form

Catégorie Nom d'un sous-ensemble de phrase valides
Ex : <Expression>, <Entier>, <Nombre>

Production Règle pour préciser les catégories
Ex: <catégorie> ::= x_1 x_2 x_3
où x_n est une <catégorie> ou un symbole

Grammaire BNF

- Une catégorie doit être désignée catégorie de départ

Grammaire BNF pour un nombre flottant

- <flottant> est la catégorie de départ
- La barre verticale | indique qu'on peut choisir l'un ou l'autre des catégories ou symbols
- Deux ou plusieurs règles de production peuvent définir conjointement une catégorie, exemple <entier>.

⟨flottant⟩ ::= ⟨entier⟩ . ⟨entier⟩
⟨entier⟩ ::= ⟨chiffre⟩
⟨entier⟩ ::= ⟨chiffre⟩ ⟨entier⟩
⟨chiffre⟩ ::= 0|1|2|3|4|5|6|7|8|9

Grammaire BNF pour un nombre flottant

Grammaire BNF et dérivations

Dérivation directe	Application d'une règle de production
Dérivation	Séquence de dérivation directes, commançant par la règle de départ et se terminant par un phrase
Langage	L'ensemble des phrases ayant une dérivation

Grammaire BNF et dérivations

- 1.5 est-il un nombre flottant ? Oui car il existe une dérivation

⟨flottant⟩ ::= ⟨entier⟩ . ⟨entier⟩
⟨entier⟩ ::= ⟨chiffre⟩
⟨entier⟩ ::= ⟨chiffre⟩ ⟨entier⟩
⟨chiffre⟩ ::= 0|1|2|3|4|5|6|7|8|9

Grammaire BNF pour un nombre flottant

```
   ⟨flottant⟩
=> ⟨entier⟩ . ⟨entier⟩
=> ⟨chiffre⟩ . ⟨entier⟩
=> ⟨chiffre⟩ . ⟨chiffre⟩
=> 1 . ⟨chiffre⟩
=> 1 . 5
```

Dérivation pour le nombre 1.5

Grammaire BNF et arbre de dérivation

- Le lien entre une dérivation et un arbre est simple
  - La catégorie à gauche du ::= d'une règle de production devient la racine de l'arbre ou de son sous-arbre
  - Les catégories ou les symbols à droite sont les enfants

Grammaire BNF et arbre de dérivation

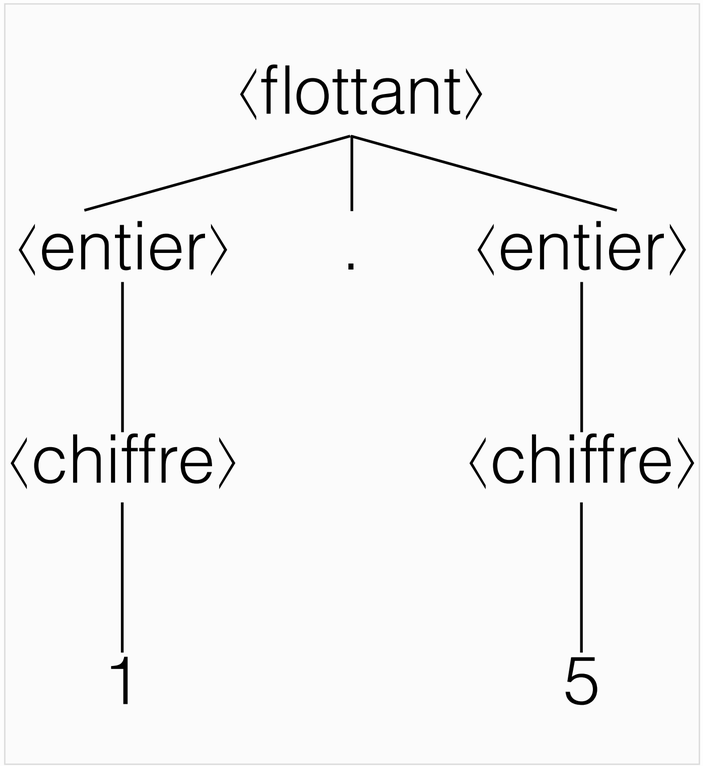
- Arbre et dérivation pour le nombre 1.5

⟨flottant⟩ ::= ⟨entier⟩ . ⟨entier⟩
⟨entier⟩ ::= ⟨chiffre⟩
⟨entier⟩ ::= ⟨chiffre⟩ ⟨entier⟩
⟨chiffre⟩ ::= 0|1|2|3|4|5|6|7|8|9

Grammaire (nombre flottant)

```
   ⟨flottant⟩
=> ⟨entier⟩ . ⟨entier⟩
=> ⟨chiffre⟩ . ⟨entier⟩
=> ⟨chiffre⟩ . ⟨chiffre⟩
=> 1 . ⟨chiffre⟩
=> 1 . 5
```

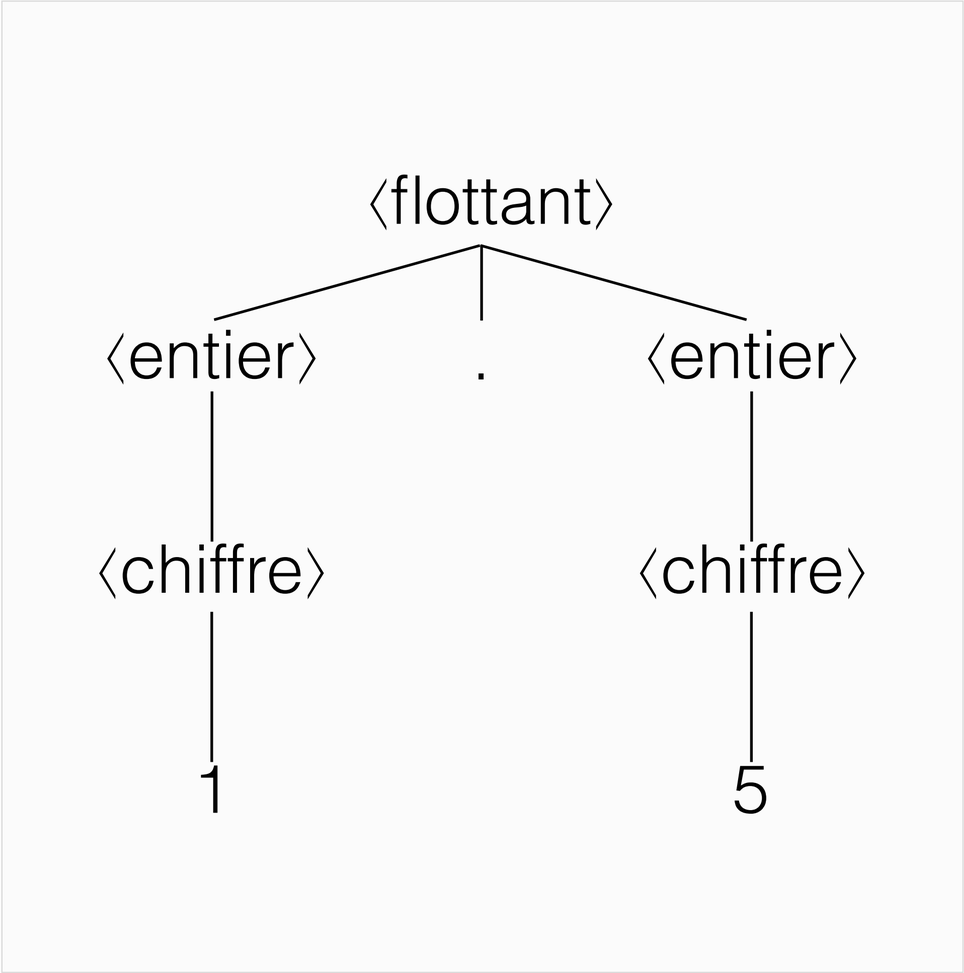
Dérivation de 1.5



Arbre de dérivation pour 1.5

Grammaire BNF et arbre de dérivation

- La racine de l'arbre de dérivation est la catégorie de départ
- Les feuilles de l'arbre de dérivation sont des symboles



Arbre de dérivation pour 1.5

Arbre de dérivation et arbre de syntaxe abstraite

- L’arbre de dérivation est presque un arbre de syntaxe abstraite
- L’arbre de dérivation contient par contre tous les symboles de la phrase
- L’arbre de syntaxe abstraite ne contient pas certains symboles superflux
- L’arbre de syntaxe abstraite peut contenir d’autres informations, comme le numéro de ligne où une variable est déclarée

Grammaire BNF et ambiguïté

- Une grammaire est ambiguë s'il existe pour une même phrase plus qu'un arbre de dérivation

Grammaire BNF et ambiguïté

- Grammaire ambiguë pour l'addition

⟨expr⟩ ::= <expr> + <expr>
⟨expr⟩ ::= ⟨chiffre⟩
⟨chiffre⟩ ::= 0|1|2|3|4|5|6|7|8|9

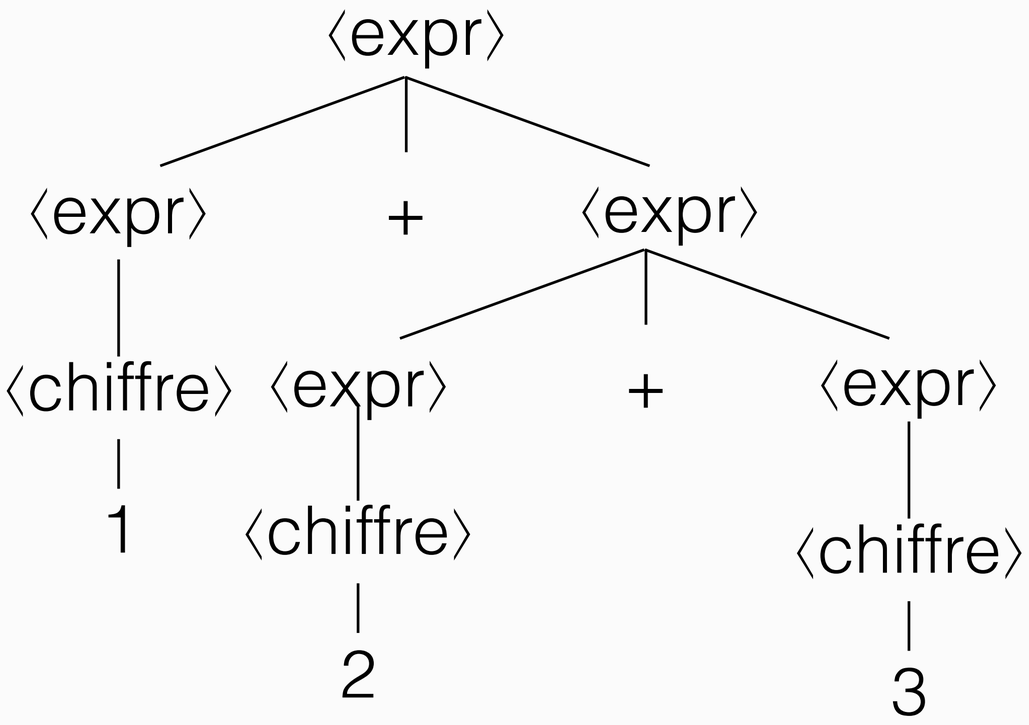
Grammaire ambiguë pour l'addition

Grammaire BNF et ambiguïté

- Arbre et dérivation pour 1 + 2 + 3

```
   ⟨expr⟩
=> ⟨expr⟩ + ⟨expr⟩
=> ⟨chiffre⟩ + ⟨expr⟩
=> ⟨chiffre⟩ + ⟨expr⟩ + ⟨expr⟩
=> ⟨chiffre⟩ + ⟨chiffre⟩ + ⟨expr⟩
=> ⟨chiffre⟩ + ⟨chiffre⟩ + ⟨chiffre⟩
=> 1 + ⟨chiffre⟩ + ⟨chiffre⟩
=> 1 + 2 + ⟨chiffre⟩
=> 1 + 2 + 3
```

Dérivation de 1 + 2 + 3



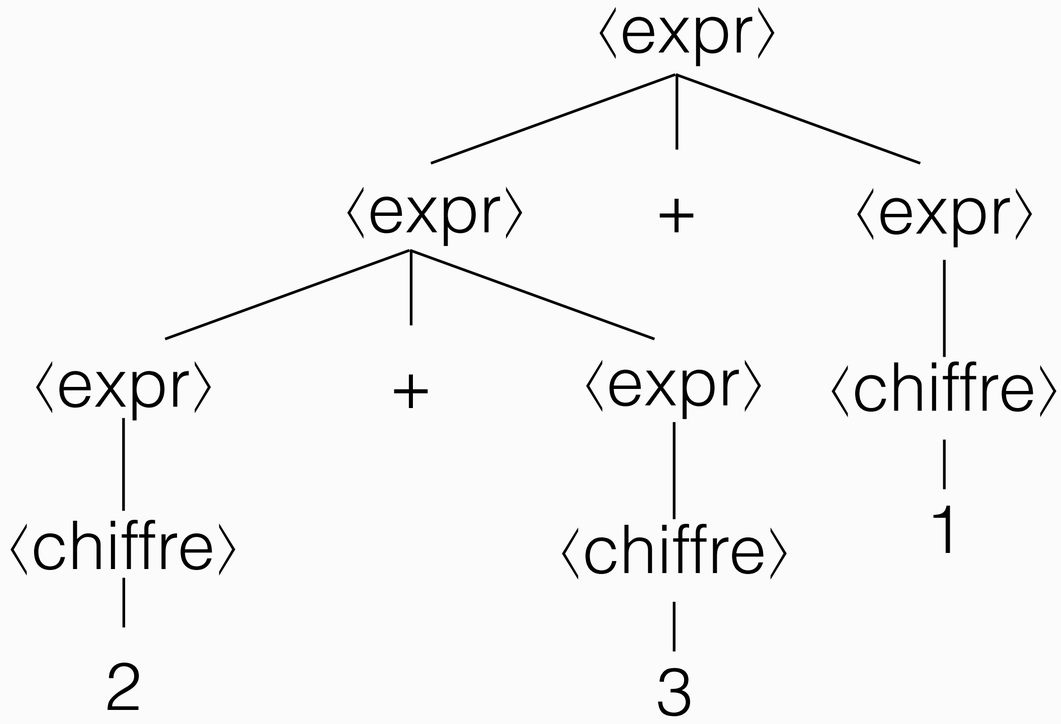
Arbre de dérivation pour 1 + 2 + 3

Grammaire BNF et ambiguïté

- 2e Arbre et dérivation pour 1 + 2 + 3

```
   ⟨expr⟩
=> ⟨expr⟩ + ⟨expr⟩
=> ⟨expr⟩ + ⟨chiffre⟩
=> ⟨expr⟩ + ⟨expr⟩ + ⟨chiffre⟩
=> ⟨chiffre⟩ + ⟨expr⟩ + ⟨chiffre⟩
=> ⟨chiffre⟩ + ⟨chiffre⟩ + ⟨chiffre⟩
=> 1 + ⟨chiffre⟩ + ⟨chiffre⟩
=> 1 + 2 + ⟨chiffre⟩
=> 1 + 2 + 3
```

2e dérivation de 1 + 2 + 3



2e arbre de dérivation pour 1 + 2 + 3

Extended BNF (EBNF)

- Extension de la syntaxe BNF avec des expressions régulières
  - (x) => groupement entre parenthèses
  - [x] => optionnel. Équivalent à ε | x
  - x* => 0 ou plusieurs fois. Parfois noté {x}
  - x+ => 1 ou plusieurs fois

Extended BNF pour un nombre flottant

⟨flottant⟩ ::= ⟨entier⟩ . ⟨entier⟩
⟨entier⟩ ::= ⟨chiffre⟩
⟨entier⟩ ::= ⟨chiffre⟩ ⟨entier⟩
⟨chiffre⟩ ::= 0|1|2|3|4|5|6|7|8|9

Grammaire BNF pour un nombre flottant

flottant = entier . entier
entier = chiffre+
chiffre = 0|1|2|3|4|5|6|7|8|9

Grammaire EBNF pour un nombre flottant