Grammaires non contextuelles

Montaigne 2023-2024

– mpi23@arrtes.net –

Vers les grammaires

L'étude des langages réguliers a montré leur force en offrant des solutions à certains problèmes comme celui qui consiste à analyser le code d'un programme se trouvant dans un fichier :

- reconnaître un mot-clé ;
- reconnaître un identificateur ;
- reconnaître une constante flottante ;
- reconnaître un commentaire.

Vers les grammaires

Mais elle a aussi établi leur limite. Certaines questions restent en suspens dont deux principales :

- comment vérifier des **propriétés non régulières** ;
- comment comprendre **la structure** de ce qu'on a lu.

Ce dernier point est particulièrement important. Si des automates permettent de reconnaître un mot, la **façon** dont le mot a été reconnu importe peu. Ce n'est plus le cas avec un texte structuré. Comment lire le texte $\mathtt{x} + \mathtt{y} \times \mathtt{z}$ de façon à comprendre qu'il s'agit de $\mathtt{x} + (\mathtt{y} \times \mathtt{z})$?

Vers les grammaires

De nouveaux objets, appelés **grammaires**, permettent de traiter formellement des problèmes du type précédent ainsi que d'autres, plus complexes, comme notamment reconnaître des langages situés au-delà de l'ensemble des langages reconnaissables.

De manière générale, la grammaire est l'étude des structures et des règles qui régissent une langue. En informatique, ce concept est utilisé dans le domaine de la compilation (analyse syntaxique), en théorie de la calculabilité ou encore dans le traitement des langues naturelles.

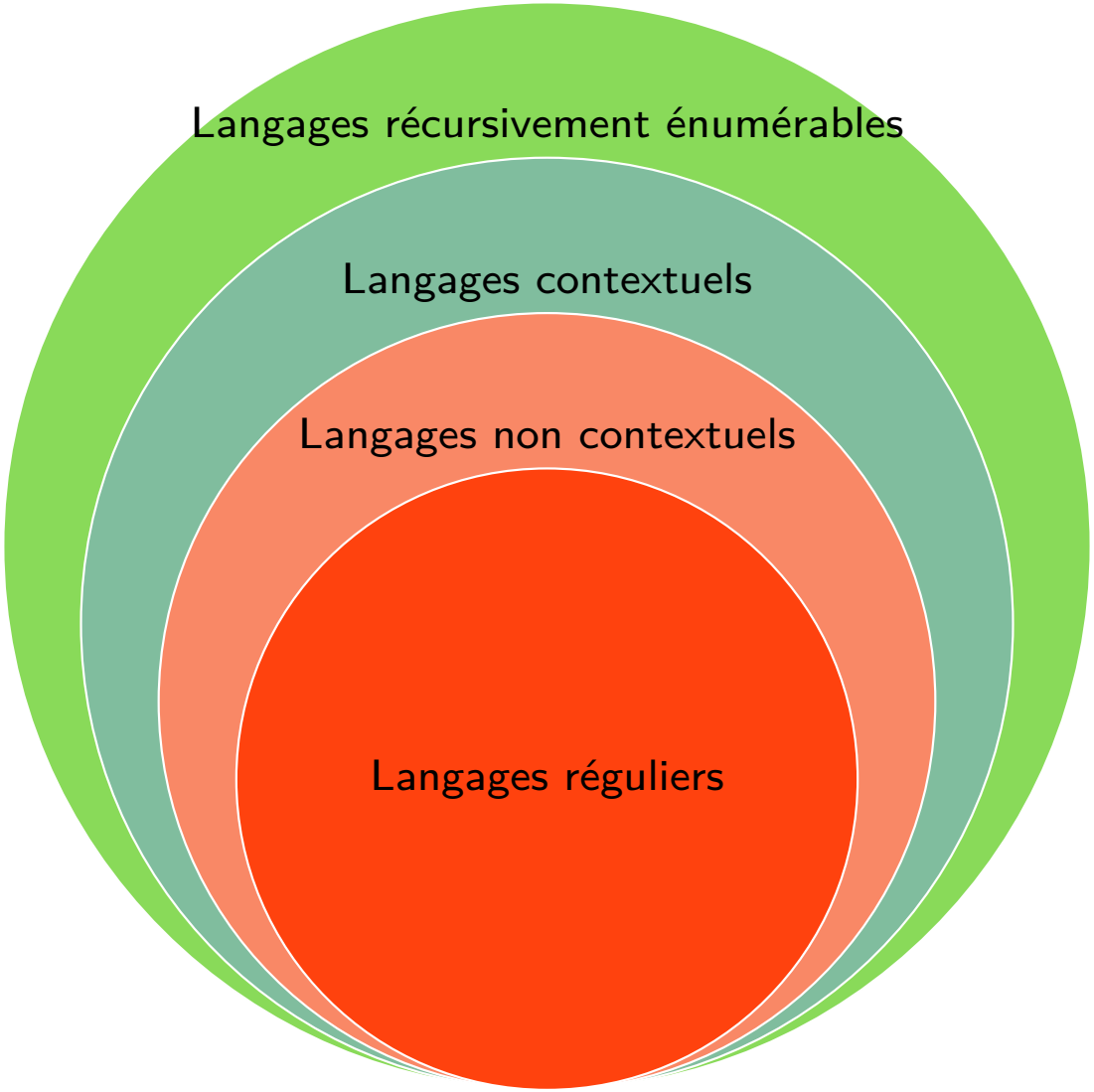
Une **grammaire formelle** définit un ensemble de règles visant à construire un **langage formel**.

Hiérachie de Chomsky

En 1956, le linguiste Noam Chomsky a formellement défini la notion de grammaire formelle et a proposé une classification des langages associés appelée **hiérarchie de Chomsky**.

- Les **langages réguliers** (type 3) sont reconnaissables par des **automates finis** et qu'ils peuvent être dénotés par des **expressions régulières**.
- Les **langages non contextuels** (type 2) sont reconnaissables par des automates à pile non déterministes. (HP)
- Les **langages contextuels** (type 1) sont reconnaissables par des machines de Turing non déterministes à ruban de longueur bornée. (HP)
- Les **langages récursivement énumérables** (type 0) sont reconnaissables par des machines de Turing. (HP)

Hiérachie de Chomsky



Généralités

Grammaire non contextuelle

Conformément au programme, ce chapitre étudie uniquement les **grammaires non contextuelles**, qui permettent de définir les **langages non contextuels** (ou **langages algébriques**), sans aborder le thème des automates à piles.

Pour définir formellement une grammaire non contextuelle, on considère deux premiers ensembles de symboles $\mathcal{V}_T$ et $\mathcal{V}_N$ qui définissent deux **alphabets**.

- $\mathcal{V}_N$ est l'alphabet des **non terminaux**.
- $\mathcal{V}_T$ est l'alphabet des **terminaux**.
- Ces ensembles sont disjoints : $\mathcal{V}_T \cap \mathcal{V}_N = \varnothing$.

Grammaire non contextuelle

L'ensemble $\mathcal{V} = \mathcal{V}_T \cup \mathcal{V}_N$ définit le **vocabulaire** de la grammaire. Les éléments de $\mathcal{V}^*$ peuvent être appelé des **proto-mots**. Ceux de $\mathcal{V}_T^*$ sont appelés **mots**.

Définition 1 (grammaire)

Une **grammaire non contextuelle** $\mathcal{G}$ est définie par la donnée d'un quadruplet $(\mathcal{V}_N, \mathcal{V}_T, \mathcal{R}, S)$ où $S \in \mathcal{V}_N$ est **le symbole initial** de la grammaire et $\mathcal{R} \subset \mathcal{V}_N \times \mathcal{V}^*$ est un ensemble **fini** de couples appelés **règles de production**.

Exemple

Soit :

$$\mathcal{V}_N = \{S\} \qquad \mathcal{V}_T = \{a, b\} \qquad \mathcal{R} = \{(S, \varepsilon), (S, aSb)\}$$

Le quadruplet $(\mathcal{V}_N, \mathcal{V}_T, \mathcal{R}, S)$ définit une grammaire :

- à **deux symboles terminaux** a et b ;
- à **un seul symbole non terminal** S ;
- à **deux règles de production** définies par (S, ε) et (S, aSb) ;
- au **symbole initial** S.

Notations

On adopte souvent des conventions d'écriture (non obligatoires).

- Les **symboles terminaux** sont représentés par des lettres minuscules romanes a, b, c, etc.
- Les **symboles non terminaux** sont représentés par des lettres majuscules romanes A, B, C, etc.
- Les **proto-mots** sont représentés par des lettres grecques minuscules α, β, γ, etc.

Notations

Si $A \in \mathcal{V}_N$ et $\alpha \in \mathcal{V}^*$, une **règle** $(A, \alpha) \in \mathcal{R}$ est souvent notée sous la forme : $A \to \alpha$.

Une **règle** peut être **vide à droite**, ce que l'on écrit : $A \to \varepsilon$.

Si un même symbole non terminal A peut engendrer plusieurs mots par les règles $(A, \alpha_1) \in \mathcal{R}, \ldots, (A, \alpha_n) \in \mathcal{R}$, on note simplement : $A \to \alpha_1 \mid \ldots \mid \alpha_n$.

En pratique, **les règles de production** suffisent pour définir une grammaire. Ainsi la grammaire de l'exemple précédent est définie par : $\mathcal{R} = \{S \to \varepsilon \mid aSb\}$.

Vocabulaire

Les **grammaires non contextuelles** sont également appelées **grammaires hors-contextes** ou **grammaires algébriques** en français ; **context-free grammar (CFG)** en anglais.

Le **qualificatif non contextuel** vient de la forme des règles de production $A \to \alpha$ où $A \in \mathcal{V}_N$ et $\alpha \in \mathcal{V}_T \cup \mathcal{V}_N^*$. Cette règle est appliquée indépendamment du contexte.
Une règle de la forme $aAb \to \alpha$ avec $a, b \in \mathcal{V}_T$, $A \in \mathcal{V}_N$, $\alpha \in \mathcal{V}^*$, est appliquée en fonction d'un contexte : si A est précédé de a et suivi de b alors il peut être remplacé par α.

Exemple

Considérons la grammaire $\mathcal{G} = (\mathcal{V}_N, \mathcal{V}_T, \mathcal{R}, S)$ suivante.

- $\mathcal{V}_N = \{S, I, E, V, N, C\}$
- $\mathcal{V}_T = \{\texttt{x}, \texttt{y}, \texttt{print}, \texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}, \texttt{=}, \texttt{+}, \texttt{;}, \texttt{(}, \texttt{)}\}$
- $\mathcal{R}$:

$$
\begin{array}{lcl}
S & \rightarrow & I\,\texttt{;}\,S \mid \varepsilon \\
I & \rightarrow & V\,\texttt{=}\,E \mid \texttt{print(}E\texttt{)} \\
E & \rightarrow & E\,\texttt{+}\,E \mid N \\
V & \rightarrow & \texttt{x} \mid \texttt{y} \\
N & \rightarrow & CN \mid C \\
C & \rightarrow & \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}
\end{array}
$$

Exemple

Le mot `x = 42;` est une suite de terminaux que l'on peut obtenir en partant de S en :

→ remplaçant S par $I;S$:	$I;S$
→ remplaçant I par $V = E$:	$V = E;S$
→ remplaçant V par `x` :	`x` $= E;S$
→ remplaçant E par N :	`x` $= N;S$
→ remplaçant N par CN :	`x` $= CN;S$
→ remplaçant C par `4` :	`x` $=$ `4`$N;S$
→ remplaçant N par C :	`x` $=$ `4`$C;S$
→ remplaçant C par `2` :	`x` $=$ `42`$;S$
→ remplaçant S par ε :	`x` $=$ `42;`

Dérivation immédiate

Définition 2 (dérivation immédiate)

Soit $\alpha = \alpha_1 A \alpha_2 \in \mathcal{V}^* \mathcal{V}_N \mathcal{V}^*$ et $\beta \in \mathcal{V}^*$.
On dit que α **se dérive immédiatement** en β s'il existe $\gamma \in \mathcal{V}^*$ tel que $A \to \gamma \in \mathcal{R}$ et $\beta = \alpha_1 \gamma \alpha_2$. On note : $\alpha \Rightarrow \beta$.

Dans l'exemple précédent, chaque étape constitue une **dérivation immédiate** d'un proto-mot vers un autre proto-mot.

Dérivation

Définition 3 (dérivation)

Soit $\alpha \in \mathcal{V}^*$ et $\beta \in \mathcal{V}^*$.
On dit que α **se dérive** en β s'il existe une suite de dérivations immédiates qui mène de α à β. On note : $\alpha \Rightarrow^* \beta$.

Dans l'exemple précédent, le mot `x = 42;` est une **dérivation** du symbole initial.

$$S \Rightarrow^* \texttt{x = 42;}$$

Langage engendré par une grammaire

Définition 4 (langage engendré par une grammaire)

Soit $\mathcal{G} = (\mathcal{V}_N, \mathcal{V}_T, \mathcal{R}, S)$ une grammaire.
Le **langage engendré par la grammaire** est défini par

$$\mathcal{L}(\mathcal{G}) = \{w \in \mathcal{V}_T^* \mid S \Rightarrow^* w\}$$

Un langage engendré par une grammaire non contextuelle est un **langage non contextuel**. Il n'est rien d'autres que l'ensemble des **mots** que la grammaire peut engendrée. Ce qui justifie que $\mathcal{V}_T^*$ désigne l'ensemble des mots (terminaux).

Les termes de langage non contextuel, langage hors contexte et langage algébrique sont synonymes. En anglais, l'acronyme **CFL** pour **context-free language** est couramment utilisé.

Non contextualité des langages réguliers

Les langages réguliers sont des cas particuliers des langages non contextuels. Le théorème suivant formalise cette idée.

Théorème 5 (non contextualité des langages réguliers)

L'ensemble des langages réguliers est inclus strictement dans l'ensemble des langages non contextuels.

Non contextualité des langages réguliers

Démonstration

Montrons d'abord l'inclusion.
Si un langage est régulier alors il existe un automate $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ qui le reconnaît. Sans perte de généralité, on peut supposer $\mathcal{A}$ complet et déterministe. On construit une grammaire $\mathcal{G} = (\mathcal{V}_N, \mathcal{V}_T, \mathcal{R}, X_{q_0})$ comme suit.

- $\mathcal{V}_N = \{X_q \mid q \in Q\}$
- $\mathcal{R} = \{X_q \to aX_{\delta(q,a)} \mid q \in Q, a \in \mathcal{V}_T\} \cup \{X_q \to \varepsilon \mid q \in F\}$

On montre l'équivalence $X_q \Rightarrow^* vX_{q'} \Leftrightarrow q \xrightarrow{v}{}^*_{\mathcal{A}} q'$ par récurrence sur $|v|$.

- **Cas de base $|v| = 0$.**
  Quels que soient les automates et les grammaires considérés, on a toujours :
  $$X_q \Rightarrow^* X_q \qquad \text{et} \qquad q \xrightarrow{\varepsilon}{}^*_{\mathcal{A}} q$$
- **Cas $|v| > 0$.**
  Supposons la propriété vraie pour les mots de taille n et montrons-la pour les mots v de taille $n+1$. On pose $v = ua$ avec $|u| = n$. Par hypothèse de récurrence :
  $$X_q \Rightarrow^* uX_{q'} \Leftrightarrow q \xrightarrow{u}{}^*_{\mathcal{A}} q'$$
  L'automate étant complet et déterministe, il existe q'' tel que $q'' = \delta(q', a)$ et par construction, il existe $X_{q''}$ tel que $X'_q \to aX_{q''}$.
  Ainsi $X_q \Rightarrow^* uX_{q'} \Rightarrow uaX_{q''}$ est une dérivation valide.

Démonstration

Montrons maintenant que l'inclusion est stricte en considérant la grammaire :

$$\mathcal{G} = (\{S\}, \{a, b\}, \{S \to aSb \mid \varepsilon\}, S)$$

On a $\mathcal{L}(\mathcal{G}) = \{a^n b^n \mid n \in \mathbb{N}0\}$, langage dont on sait qu'il n'est pas régulier. Pour tout entier naturel n, on montre, par récurrence sur n, qu'il existe une dérivation de longueur $n+1$ telle que $S \Rightarrow^{n+1} a^n b^n$.

- **Cas de base $n = 0$.**
  La dérivation de longueur un $S \Rightarrow \varepsilon$ est bien possible et $\varepsilon = a^0 b^0$.
- **Cas $n > 0$.**
  On considère $S \Rightarrow aSb$. Par hypothèse de récurrence, il existe une dérivation de taille $n+1$, $S \Rightarrow^{n+1} a^n b^n$.
  On peut donc construire la dérivation $S \Rightarrow aSb \Rightarrow^{n+1} aa^n b^n b$ de taille $n+2$, qui engendre le mot $a^{n+1} b^{n+1}$.

Arbre d'analyse

Arbre de dérivation

Les dérivations ne font pas apparaître ce qu'il advient quand on engendre un mot à l'aide d'une grammaire. Un **arbre de dérivation** le fait.

Définition 6 (arbre de dérivation)

Soit $\mathcal{G} = (\mathcal{V}_N, \mathcal{V}_T, \mathcal{R}, S)$ une grammaire. Un **arbre de dérivation** est un arbre étiqueté aux nœuds tel que :

- la racine est étiquetée par S ;
- tout nœud interne est étiqueté par un symbole de $\mathcal{V}_N$;
- toute feuille est étiquetée par un symbole de $\mathcal{V}_T \cup \{\varepsilon\}$;
- si $u_1, \ldots, u_n$ sont les enfants d'un nœud étiqueté X, alors il existe une règle $X \to u_1 \ldots u_n$ dans $\mathcal{R}$.

Arbre de dérivation

Un arbre dérivation dont les feuilles, concaténées de gauche à droite, forment un mot u est appelé **arbre de dérivation de u**.

On emploie également les termes **arbre de dérivation syntaxique de u** ou **arbre syntaxique de u**.

Exemple

Considérons la grammaire $\mathcal{G}_{\text{arith}} = (\mathcal{V}_N, \mathcal{V}_T, \mathcal{R}, S)$ avec :

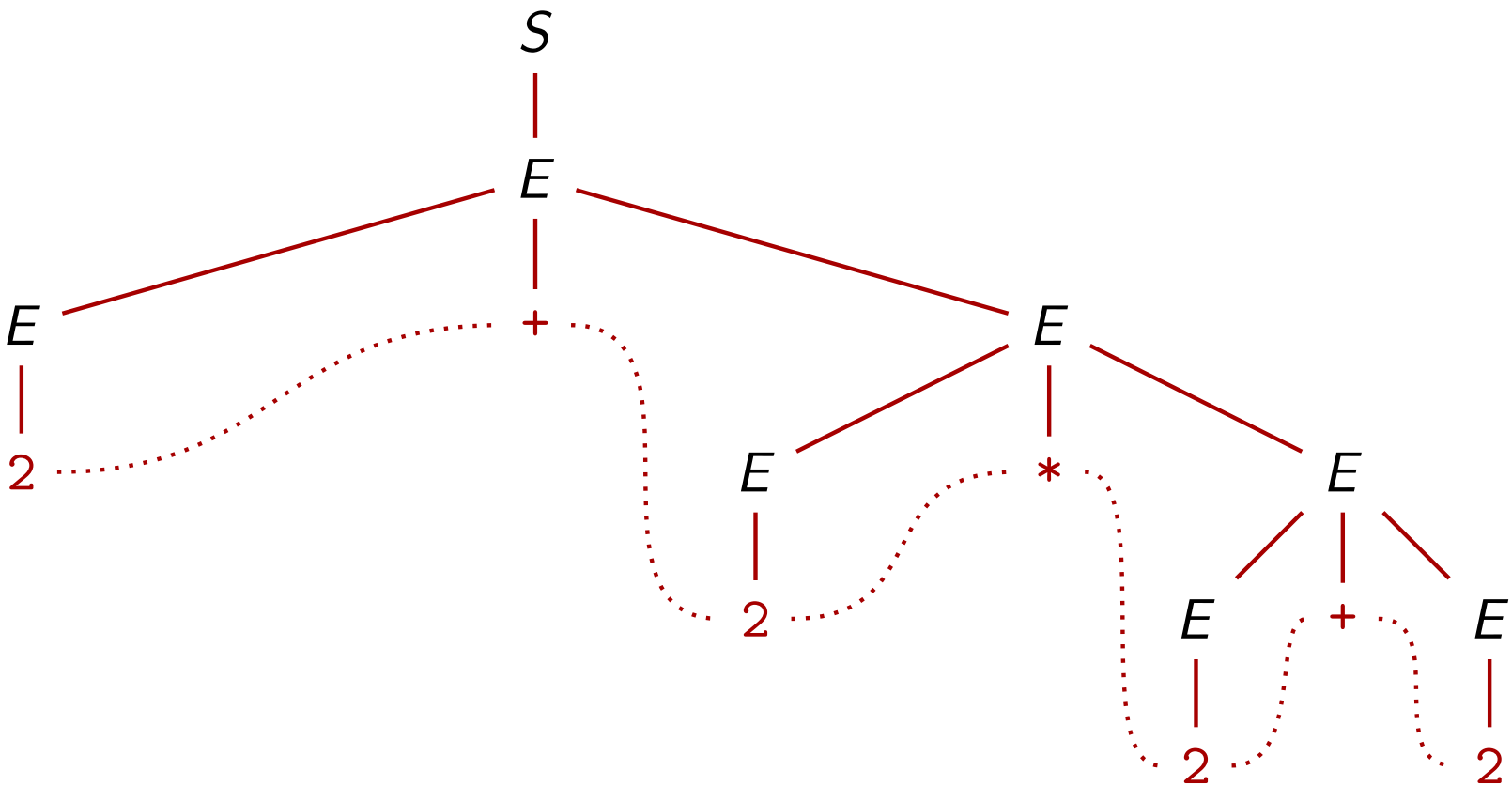
- $\mathcal{V}_N = \{S, I, E, V, N, C\}$
- $\mathcal{V}_T = \{2, +, *, (,)\}$
- $\mathcal{R}$: $S \rightarrow E$
  $E \rightarrow E+E \mid E*E \mid (E) \mid 2$

La diapositive suivante présente un arbre de dérivation pour :

$$2 + 2 * 2 + 2$$

Les arcs en pointillés indiquent le mot constitué des feuilles de l'arbre.

Exemple



Arbre de dérivation pour 2 + 2 * 2 + 2

Levée de non déterminisme (1)

Les arbres de dérivations permettent de se débarasser d'**une première source de non déterminisme** : celle du choix du non terminal à substituer.

Proposition (*arbre de dérivation indépendant de la stratégie*)

Soit $\mathcal{G} = (\mathcal{V}_N, \mathcal{V}_T, \mathcal{R}, S)$ une grammaire et $u \in \mathcal{V}_T^*$ tel que $S \Rightarrow u^*$. Soit t un arbre de dérivation pour u. Il n'existe qu'une seule dérivation gauche $S \Rightarrow_g^* u$ et une seule dérivation droite pour $S \Rightarrow_d^* u$ pour t.

Démonstration

La preuve utilise le fait que la dérivation gauche (resp. droite) correspond à appliquer les règles telles qu'on les rencontre en parcourant l'arbre en profondeur d'abord, de gauche à droite (resp. de droite à gauche).

Ainsi, **quel que soit** le non terminal choisi lors d'une dérivation au sein du membre droit d'une règle, le mot obtenu sera le même **et** il sera obtenu de la même façon.

Levée de non déterminisme (2)

La **seconde source de non déterminisme** ne peut être ignorée. C'est celle qui consiste à choisir entre plusieurs règles possibles pour un même non terminal.

Définition 7 (grammaire ambiguë)

Une grammaire $\mathcal{G}$ est dite **ambiguë** s'il existe un mot $v \in \mathcal{L}(\mathcal{G})$ tel que v possède deux arbres de dérivations distincts.

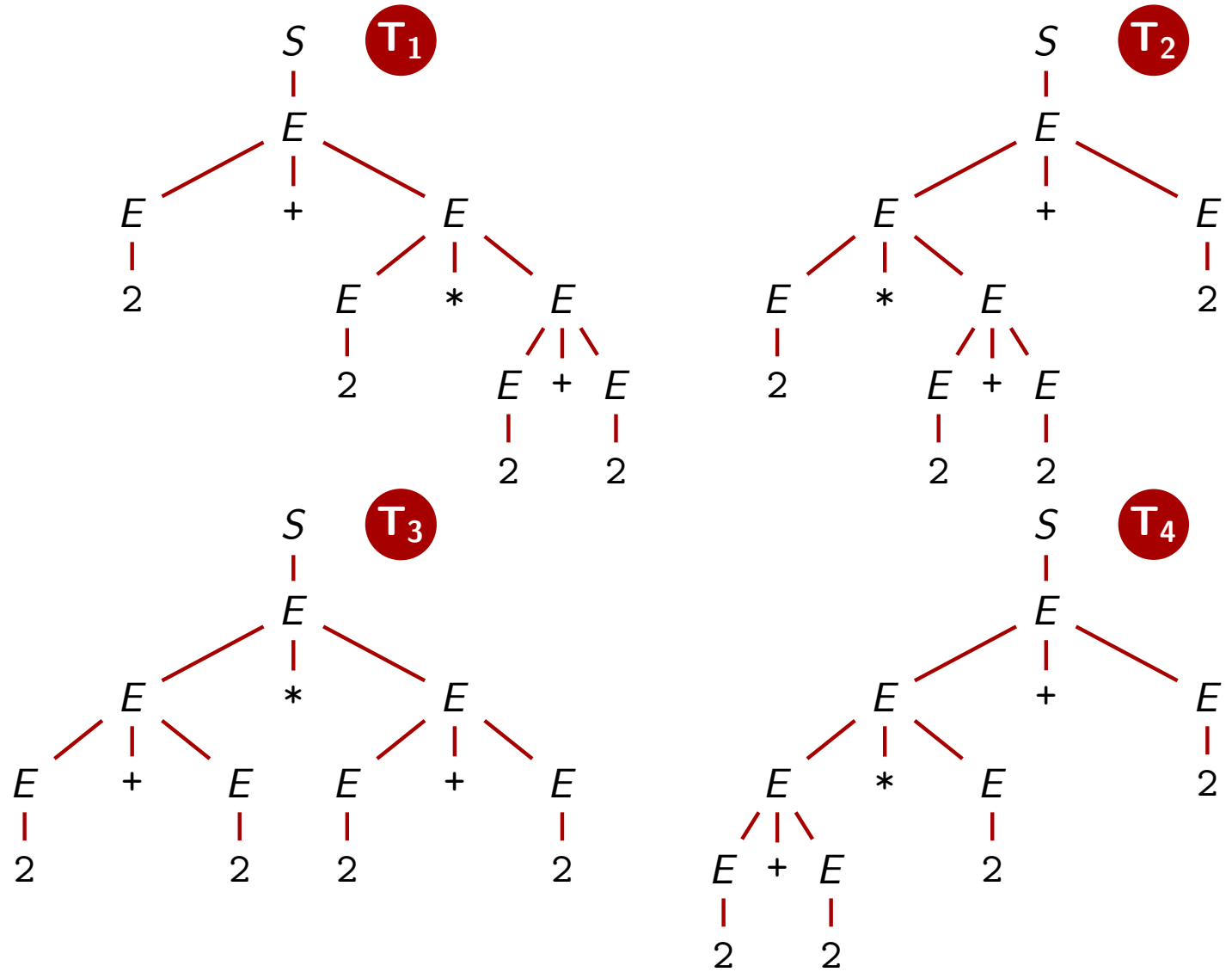
La même ambiguïté syntaxique existe en mathématiques. Lorsque l'on écrit $2 + 2 \times 2 + 2$, il n'y a rien **dans la syntaxe** de l'expression qui nous dit comment la calculer alors que dans l'expression $2 + ((2 \times 2) + 2)$ la syntaxe force un ordre de calcul. Il y a donc en mathématiques une règle supplémentaire qui indique que, en cas de conflit entre deux calculs, la multiplication est prioritaire par rapport à l'addition.

Exemple

La diapositive suivante montre quatre arbres de dérivation pour la même expression, obtenus **indépendamment du type de dérivation** (droite, gauche, arbitraire). Ces quatre arbres ont une dérivation gauche distinctes.

- $T_1 : S \Rightarrow_g E \Rightarrow_g E{+}E \Rightarrow_g 2{+}E \Rightarrow_g \ldots$
- $T_2 : S \Rightarrow_g E \Rightarrow_g E{+}E \Rightarrow_g E{*}E{+}E \Rightarrow_g 2\ {*}E{+}E \Rightarrow_g \ldots$
- $T_3 : S \Rightarrow_g E \Rightarrow_g E{*}E \Rightarrow_g \ldots$
- $T_4 : S \Rightarrow_g E \Rightarrow_g E{+}E \Rightarrow_g E{*}E{+}E \Rightarrow_g E{+}E{*}E{+}E \Rightarrow_g \ldots$

Exemple



Plusieurs arbres distincts pour 2 + 2 * 2 + 2.

dangling else

L'ambiguïté se retrouve aussi sur des constructions « purement informatiques », comme on l'illustre maintenant avec le célèbre exemple du « sinon pendant » (**dangling else**).

Considèrons le langage C et une version simplifiée d'une partie de sa grammaire.

$$
\begin{array}{lcl}
I & \rightarrow & \texttt{if (}E\texttt{)}\ I \\
I & \rightarrow & \texttt{if (}E\texttt{)}\ I\texttt{else} \\
I & \rightarrow & \texttt{E;} \mid \ldots \\
E & \rightarrow & \texttt{x} = E \mid E < E \mid E > E \mid \ldots
\end{array}
$$

Cette grammaire dit qu'une instruction peut être un `if` sans partie `else`, un `if` avec `else` ou d'autres types d'instructions (et parmi ces dernières les expressions suivies d'un « `;` »).
Ainsi, on peut écrire :

```
if (x > 4) if (x < 5) x = 10; else x = 42;
```

dangling else

Cette grammaire est ambiguë : il existe deux arbres de dérivations possibles. On peut choisir d'appliquer en premier la règle du `if` sans `else` donnant un programme équivalent à (les accolades ont été rajoutées pour aider à la compréhension) :

```
if (x > 4) {
   if (x < 5) x = 10;
   else x = 42;
};
```

On peut également choisir la règle du `if` avec `else` dans la dérivation, donnant un programe équivalent à :

```
if (x > 4) {
   if (x < 5) x = 10;
} else x = 42;
```

dangling else

En pratique, cette ambiguïté est résolue par une règle externe consistant à choisir le `if` le plus proche du `else`. Le compilateur C comprendra donc le programme original comme le premier cas ci-dessus. Il est considéré comme une bonne pratique de mettre systématiquement des accolades afin d'augmenter la lisibilité du code dans des cas semblables.

Équivalence faible

Le fait que des grammaires puissent être ambiguës a un impact sur la notion d'équivalence de grammaire.

Définition 8 (équivalence faible)

Soit $\mathcal{G}_1$ et $\mathcal{G}_2$ deux grammaires non contextuelles. Ces grammaires sont dites **faiblement équivalentes** si $\mathcal{L}(\mathcal{G}_1) = \mathcal{L}(\mathcal{G}_2)$

Cette définition ignore donc complètement la façon dont les mots sont engendrés par les grammaires, du moment que les deux engendrent le même langage. Une equivalence plus forte existe mais est hors programme.

Que faire des grammaires ambiguës ?

Les **langages non contextuels** sont plus puissants que les **langages réguliers**. Ils fournissent un cadre plus riche pour spécifier la syntaxe de langages. Mais cette expressivité a un coût.

En premier lieu, les langages non contextuels ont moins de propriétés de stabilité : ils sont stables par union, concaténation et étoile de Kleene, mais pas par intersection ni par complémentaire.

Pire encore, le problème de savoir si une grammaire non contextuelle donnée est ambiguë est **indécidable** : il n'est pas **possible** d'écrire un programme prenant en entrée une grammaire et déterminant si elle est ambiguë.

En pratique, les outils permettant de définir les syntaxes des langages au moyen de grammaires imposent des restrictions de syntaxe ou de formation des règles afin de pouvoir émettre une erreur en cas d'ambiguïté.