Mots et langages

Jean-Pierre Becirspahic
Lycée Louis-Le-Grand

Mots et langages

Exemples d'applications

La compilation désigne la tâche consistant à transformer un ensemble de commandes écrites dans un langage de programmation de haut niveau en une série d'instructions exécutables par l'ordinateur.

Mots et langages

Exemples d'applications

La compilation désigne la tâche consistant à transformer un ensemble de commandes écrites dans un langage de programmation de haut niveau en une série d'instructions exécutables par l'ordinateur.

- l'analyse lexicale a pour objet d'identifier les mots-clés du langage ;

Mots et langages

Exemples d'applications

La compilation désigne la tâche consistant à transformer un ensemble de commandes écrites dans un langage de programmation de haut niveau en une série d'instructions exécutables par l'ordinateur.

- l'analyse lexicale a pour objet d'identifier les mots-clés du langage ;
- l'analyse syntaxique détecte les erreurs de syntaxe et met en évidence la structure interne du programme ;

Mots et langages

Exemples d'applications

La compilation désigne la tâche consistant à transformer un ensemble de commandes écrites dans un langage de programmation de haut niveau en une série d'instructions exécutables par l'ordinateur.

- l'analyse lexicale a pour objet d'identifier les mots-clés du langage ;
- l'analyse syntaxique détecte les erreurs de syntaxe et met en évidence la structure interne du programme ;
- l'analyse sémantique vise à donner un sens à la phrase étudiée.

Mots et langages

Exemples d'applications

La compilation désigne la tâche consistant à transformer un ensemble de commandes écrites dans un langage de programmation de haut niveau en une série d'instructions exécutables par l'ordinateur.

- l'analyse lexicale a pour objet d'identifier les mots-clés du langage ;
- l'analyse syntaxique détecte les erreurs de syntaxe et met en évidence la structure interne du programme ;
- l'analyse sémantique vise à donner un sens à la phrase étudiée.

En génétique, la recherche de séquences particulières de nucléotides dans un chromosome ou la détection de ressemblances entre plusieurs fragments d'ADN constituent des problèmes de base de la bio-informatique.

Mots et langages

Exemples d'applications

La compilation désigne la tâche consistant à transformer un ensemble de commandes écrites dans un langage de programmation de haut niveau en une série d'instructions exécutables par l'ordinateur.

- l'analyse lexicale a pour objet d'identifier les mots-clés du langage ;
- l'analyse syntaxique détecte les erreurs de syntaxe et met en évidence la structure interne du programme ;
- l'analyse sémantique vise à donner un sens à la phrase étudiée.

En génétique, la recherche de séquences particulières de nucléotides dans un chromosome ou la détection de ressemblances entre plusieurs fragments d'ADN constituent des problèmes de base de la bio-informatique.

Enfin, la notion d'expression régulière permet de rechercher un ensemble de mots qui vérifient certaines propriétés communes. Par exemple :

- **`[Cc]h(at|ien)`** décrit les quatre motifs `Chat`, `chat`, `Chien`, `chien` ;

Mots et langages

Exemples d'applications

La compilation désigne la tâche consistant à transformer un ensemble de commandes écrites dans un langage de programmation de haut niveau en une série d'instructions exécutables par l'ordinateur.

- l'analyse lexicale a pour objet d'identifier les mots-clés du langage ;
- l'analyse syntaxique détecte les erreurs de syntaxe et met en évidence la structure interne du programme ;
- l'analyse sémantique vise à donner un sens à la phrase étudiée.

En génétique, la recherche de séquences particulières de nucléotides dans un chromosome ou la détection de ressemblances entre plusieurs fragments d'ADN constituent des problèmes de base de la bio-informatique.

Enfin, la notion d'expression régulière permet de rechercher un ensemble de mots qui vérifient certaines propriétés communes. Par exemple :

- **`(19|20)[0-9]{2}`** décrit toute date comprise entre 1900 et 2099 ;

Mots et langages

Exemples d'applications

La compilation désigne la tâche consistant à transformer un ensemble de commandes écrites dans un langage de programmation de haut niveau en une série d'instructions exécutables par l'ordinateur.

- l'analyse lexicale a pour objet d'identifier les mots-clés du langage ;
- l'analyse syntaxique détecte les erreurs de syntaxe et met en évidence la structure interne du programme ;
- l'analyse sémantique vise à donner un sens à la phrase étudiée.

En génétique, la recherche de séquences particulières de nucléotides dans un chromosome ou la détection de ressemblances entre plusieurs fragments d'ADN constituent des problèmes de base de la bio-informatique.

Enfin, la notion d'expression régulière permet de rechercher un ensemble de mots qui vérifient certaines propriétés communes. Par exemple :

- **`[a-z]+(\.[a-z]+)?@[a-z]+\.fr`** reconnait les adresses mails de la forme `xxx@yyy.fr` ou `xxx.yyy@zzz.fr`.

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

```
[a-z]+(\.[a-z]+)?@[a-z]+\.fr
```

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

`[a-z]+(\.[a-z]+)?@[a-z]+\.fr`

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;
`(\.[a-z]+)?` : suivi *éventuellement* d'un point et d'une ou plusieurs lettres minuscules ;

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

`[a-z]+(\.[a-z]+)?@[a-z]+\.fr`

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;
`(\.[a-z]+)?` : suivi *éventuellement* d'un point et d'une ou plusieurs lettres minuscules ;
`@` : le mot se poursuit par une arobase ;

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

`[a-z]+(\.[a-z]+)?@[a-z]+\.fr`

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;
`(\.[a-z]+)?` : suivi *éventuellement* d'un point et d'une ou plusieurs lettres minuscules ;
`@` : le mot se poursuit par une arobase ;
`[a-z]+\.fr` : et enfin une ou plusieurs lettres minuscules suivies de `.fr`.

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

`[a-z]+(\.[a-z]+)?@[a-z]+\.fr`

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;
`(\.[a-z]+)?` : suivi *éventuellement* d'un point et d'une ou plusieurs lettres minuscules ;
`@` : le mot se poursuit par une arobase ;
`[a-z]+\.fr` : et enfin une ou plusieurs lettres minuscules suivies de `.fr`.

Plus généralement, on peut utiliser l'expression régulière ci-dessous pour détecter une adresse mail :

`\w+([-+.']\w+)*@\w+([-.]\w+)*\.[a-zA-Z]{2,6}`

`jp.becir@info-llg.fr`

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

```
[a-z]+(\.[a-z]+)?@[a-z]+\.fr
```

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;
`(\.[a-z]+)?` : suivi *éventuellement* d'un point et d'une ou plusieurs lettres minuscules ;
`@` : le mot se poursuit par une arobase ;
`[a-z]+\.fr` : et enfin une ou plusieurs lettres minuscules suivies de `.fr`.

Plus généralement, on peut utiliser l'expression régulière ci-dessous pour détecter une adresse mail :

```
\w+([-+.']\w+)*@\w+([-.]\w+)*\.[a-zA-Z]{2,6}
```

`\w+` : au moins un caractère alpha-numérique (ou le caractère `_`) ;

```
jp.becir@info-llg.fr
```

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

```
[a-z]+(\.[a-z]+)?@[a-z]+\.fr
```

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;
`(\.[a-z]+)?` : suivi *éventuellement* d'un point et d'une ou plusieurs lettres minuscules ;
`@` : le mot se poursuit par une arobase ;
`[a-z]+\.fr` : et enfin une ou plusieurs lettres minuscules suivies de `.fr`.

Plus généralement, on peut utiliser l'expression régulière ci-dessous pour détecter une adresse mail :

```
\w+([-+.']\w+)*@\w+([-.]\w+)*\.[a-zA-Z]{2,6}
```

`([-+.']\w+)` : un caractère spécial suivi au moins d'un caractère alpha-numérique . . .

```
jp.becir@info-llg.fr
```

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

```
[a-z]+(\.[a-z]+)?@[a-z]+\.fr
```

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;
`(\.[a-z]+)?` : suivi *éventuellement* d'un point et d'une ou plusieurs lettres minuscules ;
`@` : le mot se poursuit par une arobase ;
`[a-z]+\.fr` : et enfin une ou plusieurs lettres minuscules suivies de `.fr`.

Plus généralement, on peut utiliser l'expression régulière ci-dessous pour détecter une adresse mail :

```
\w+([-+.']\w+)*@\w+([-.]\w+)*\.[a-zA-Z]{2,6}
```

`([-+.']\w+)*` : un caractère spécial suivi au moins d'un caractère alpha-numérique ... répété 0 fois ou plus ;

```
jp.becir@info-llg.fr
```

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

```
[a-z]+(\.[a-z]+)?@[a-z]+\.fr
```

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;
`(\.[a-z]+)?` : suivi *éventuellement* d'un point et d'une ou plusieurs lettres minuscules ;
`@` : le mot se poursuit par une arobase ;
`[a-z]+\.fr` : et enfin une ou plusieurs lettres minuscules suivies de `.fr`.

Plus généralement, on peut utiliser l'expression régulière ci-dessous pour détecter une adresse mail :

```
\w+([-+.']\w+)*@\w+([-.]\w+)*\.[a-zA-Z]{2,6}
```

`\w+` : au moins un caractère alpha-numérique ;

```
jp.becir@info-llg.fr
```

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

```
[a-z]+(\.[a-z]+)?@[a-z]+\.fr
```

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;
`(\.[a-z]+)?` : suivi *éventuellement* d'un point et d'une ou plusieurs lettres minuscules ;
`@` : le mot se poursuit par une arobase ;
`[a-z]+\.fr` : et enfin une ou plusieurs lettres minuscules suivies de `.fr`.

Plus généralement, on peut utiliser l'expression régulière ci-dessous pour détecter une adresse mail :

```
\w+([-+.']\w+)*@\w+([-.]\w+)*\.[a-zA-Z]{2,6}
```

`([-.]\w+)*` : un caractère spécial suivi au moins d'un caractère alpha-numérique, répété 0 fois ou plus ;

```
jp.becir@info-llg.fr
```

Mots et langages

Exemples d'applications

Détaillons cette dernière expression régulière :

```
[a-z]+(\.[a-z]+)?@[a-z]+\.fr
```

`[a-z]+` : une adresse débute par une ou plusieurs lettres minuscules ;
`(\.[a-z]+)?` : suivi *éventuellement* d'un point et d'une ou plusieurs lettres minuscules ;
`@` : le mot se poursuit par une arobase ;
`[a-z]+\.fr` : et enfin une ou plusieurs lettres minuscules suivies de `.fr`.

Plus généralement, on peut utiliser l'expression régulière ci-dessous pour détecter une adresse mail :

```
\w+([-+.']\w+)*@\w+([-.]\w+)*\.[a-zA-Z]{2,6}
```

`[a-zA-Z]{2,6}` : entre 2 et 6 caractères alphabétiques.

```
jp.becir@info-llg.fr
```

Mots et alphabets

Alphabet : un ensemble fini Σ dont les éléments sont appelés les lettres.

Mots et alphabets

Alphabet : un ensemble fini Σ dont les éléments sont appelés les lettres.
Mot : une suite finie de n lettres, avec $n \geqslant 1$.

Mots et alphabets

Alphabet : un ensemble fini Σ dont les éléments sont appelés les lettres.
Mot : une suite finie de n lettres, avec $n \geqslant 1$.

On note Σ^+ l'ensemble des mots, les lettres étant identifiées aux mots de longueur 1. On note $|s|$ la longueur du mot s. Enfin, si s est un mot et a une lettre, $|s|_a$ désigne le nombre d'occurrences de a dans s.

Mots et alphabets

Alphabet : un ensemble fini Σ dont les éléments sont appelés les lettres.
Mot : une suite finie de n lettres, avec $n \geqslant 1$.

On note Σ^+ l'ensemble des mots, les lettres étant identifiées aux mots de longueur 1. On note $|s|$ la longueur du mot s. Enfin, si s est un mot et a une lettre, $|s|_a$ désigne le nombre d'occurrences de a dans s.
On adjoint à Σ^+ un mot noté ε et appelé mot vide, de longueur nulle. On pose alors $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

Mots et alphabets

Alphabet : un ensemble fini Σ dont les éléments sont appelés les lettres.
Mot : une suite finie de n lettres, avec $n \geqslant 1$.

On note Σ^+ l'ensemble des mots, les lettres étant identifiées aux mots de longueur 1. On note $|s|$ la longueur du mot s. Enfin, si s est un mot et a une lettre, $|s|_a$ désigne le nombre d'occurrences de a dans s.
On adjoint à Σ^+ un mot noté ε et appelé mot vide, de longueur nulle. On pose alors $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

Étant donné deux mots $r = a_1 \cdots a_p$ et $s = b_1 \cdots b_q$, la concaténation de r et de s est le mot $rs = a_1 \cdots a_p b_1 \cdots b_q$. Il s'agit d'une loi associative possédant un élément neutre ε

Mots et alphabets

Alphabet : un ensemble fini Σ dont les éléments sont appelés les lettres.
Mot : une suite finie de n lettres, avec $n \geqslant 1$.

On note Σ^+ l'ensemble des mots, les lettres étant identifiées aux mots de longueur 1. On note $|s|$ la longueur du mot s. Enfin, si s est un mot et a une lettre, $|s|_a$ désigne le nombre d'occurrences de a dans s.
On adjoint à Σ^+ un mot noté ε et appelé mot vide, de longueur nulle. On pose alors $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

Étant donné deux mots $r = a_1 \cdots a_p$ et $s = b_1 \cdots b_q$, la concaténation de r et de s est le mot $rs = a_1 \cdots a_p b_1 \cdots b_q$. Il s'agit d'une loi associative possédant un élément neutre ε
On définit par récurrence le mot r^n lorsque $r \in \Sigma^*$ et $n \in \mathbb{N}$ en posant :

$$r^0 = \varepsilon \qquad \text{et} \qquad \forall n \in \mathbb{N}^*, \quad r^n = r.r^{n-1} = r^{n-1}.r$$

Facteurs et sous-mots

Si u et v sont deux mots, on dit que u est un :

- préfixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = uw$;

Facteurs et sous-mots

Si u et v sont deux mots, on dit que u est un :

- préfixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = uw$;
- suffixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = wu$;

Facteurs et sous-mots

Si u et v sont deux mots, on dit que u est un :

- préfixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = uw$;
- suffixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = wu$;
- facteur de v lorsqu'il existe deux mots $x \in \Sigma^*$ et $y \in \Sigma^*$ tels que $v = xuy$.

Facteurs et sous-mots

Si u et v sont deux mots, on dit que u est un :

- préfixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = uw$;
- suffixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = wu$;
- facteur de v lorsqu'il existe deux mots $x \in \Sigma^*$ et $y \in \Sigma^*$ tels que $v = xuy$.

Les préfixes et suffixes sont dits propres lorsque $w \neq \varepsilon$; les facteurs sont dits propres lorsque $x \neq \varepsilon$ ou $y \neq \varepsilon$.

Facteurs et sous-mots

Si u et v sont deux mots, on dit que u est un :

- préfixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = uw$;
- suffixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = wu$;
- facteur de v lorsqu'il existe deux mots $x \in \Sigma^*$ et $y \in \Sigma^*$ tels que $v = xuy$.

Les préfixes et suffixes sont dits propres lorsque $w \neq \varepsilon$; les facteurs sont dits propres lorsque $x \neq \varepsilon$ ou $y \neq \varepsilon$.

Un sous-mot d'un mot $u = a_1 \cdots a_n$ de longueur n (les a_i désignant ses lettres) est un mot $v = a_{\varphi(1)} \cdots a_{\varphi(p)}$ de longueur p, où $\varphi : [\![1, p]\!] \to [\![1, n]\!]$ est une application strictement croissante.

Facteurs et sous-mots

Si u et v sont deux mots, on dit que u est un :

- préfixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = uw$;
- suffixe de v s'il existe un mot $w \in \Sigma^*$ tel que $v = wu$;
- facteur de v lorsqu'il existe deux mots $x \in \Sigma^*$ et $y \in \Sigma^*$ tels que $v = xuy$.

Les préfixes et suffixes sont dits propres lorsque $w \neq \varepsilon$; les facteurs sont dits propres lorsque $x \neq \varepsilon$ ou $y \neq \varepsilon$.

Un sous-mot d'un mot $u = a_1 \cdots a_n$ de longueur n (les a_i désignant ses lettres) est un mot $v = a_{\varphi(1)} \cdots a_{\varphi(p)}$ de longueur p, où $\varphi : [\![1,p]\!] \to [\![1,n]\!]$ est une application strictement croissante.

Par exemple, "hippopoto" est un préfixe, "phobie" un suffixe, "monstro" un facteur, et "strophe" un sous-mot de

"hippopotomonstrosesquippedaliophobie".

Lemme de Levi et conséquences

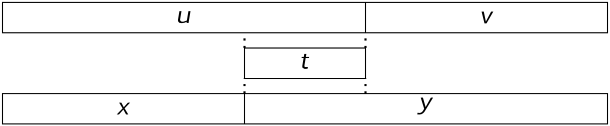
Soient x, y, u et $v \in \Sigma^*$ tels que $uv = xy$. Alors il existe un unique mot $t \in \Sigma^*$ tel que l'une des deux conditions suivantes soit réalisée :

- $u = xt$ et $y = tv$;
- $x = ut$ et $v = ty$.

Lemme de Levi et conséquences

Soient x, y, u et $v \in \Sigma^*$ tels que $uv = xy$. Alors il existe un unique mot $t \in \Sigma^*$ tel que l'une des deux conditions suivantes soit réalisée :

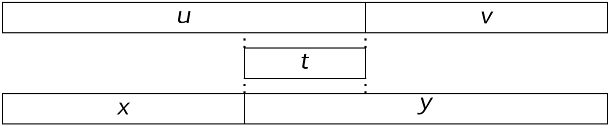
- $u = xt$ et $y = tv$;
- $x = ut$ et $v = ty$.



Lemme de LEVI et conséquences

Soient x, y, u et $v \in \Sigma^*$ tels que $uv = xy$. Alors il existe un unique mot $t \in \Sigma^*$ tel que l'une des deux conditions suivantes soit réalisée :

- $u = xt$ et $y = tv$;
- $x = ut$ et $v = ty$.



Supposons par exemple $|u| \geqslant |x|$. x est un préfixe de uv donc de u, ce qui justifie l'existence d'un mot t tel que $u = xt$. Dans ce cas, l'égalité $uv = xy$ peut encore s'écrire $xtv = xy$ puis en simplifiant $tv = y$.

Lemme de Levi et conséquences

Soient x, y, u et $v \in \Sigma^*$ tels que $uv = xy$. Alors il existe un unique mot $t \in \Sigma^*$ tel que l'une des deux conditions suivantes soit réalisée :

- $u = xt$ et $y = tv$;
- $x = ut$ et $v = ty$.

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Lemme de LEVI et conséquences

Soient x, y, u et $v \in \Sigma^*$ tels que $uv = xy$. Alors il existe un unique mot $t \in \Sigma^*$ tel que l'une des deux conditions suivantes soit réalisée :

- $u = xt$ et $y = tv$;
- $x = ut$ et $v = ty$.

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Si $|x| \geqslant |y|$, on applique le lemme de LEVI : il existe un mot t tel que $x = yt$ et $z = ty$.

Lemme de LEVI et conséquences

Soient x, y, u et $v \in \Sigma^*$ tels que $uv = xy$. Alors il existe un unique mot $t \in \Sigma^*$ tel que l'une des deux conditions suivantes soit réalisée :

- $u = xt$ et $y = tv$;
- $x = ut$ et $v = ty$.

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Si $|x| \leqslant |y|$, on raisonne par induction sur $|y|$.

Lemme de Levi et conséquences

Soient x, y, u et $v \in \Sigma^*$ tels que $uv = xy$. Alors il existe un unique mot $t \in \Sigma^*$ tel que l'une des deux conditions suivantes soit réalisée :

- $u = xt$ et $y = tv$;
- $x = ut$ et $v = ty$.

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Si $|x| \leqslant |y|$, on raisonne par induction sur $|y|$.

- Si $|y| = 1$ on a aussi $|x| = 1$ car $x \neq \varepsilon$ et dans ce cas $x = y = z$.

Lemme de Levi et conséquences

Soient x, y, u et $v \in \Sigma^*$ tels que $uv = xy$. Alors il existe un unique mot $t \in \Sigma^*$ tel que l'une des deux conditions suivantes soit réalisée :

- $u = xt$ et $y = tv$;
- $x = ut$ et $v = ty$.

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Si $|x| \leqslant |y|$, on raisonne par induction sur $|y|$.

- Si $|y| = 1$ on a aussi $|x| = 1$ car $x \neq \varepsilon$ et dans ce cas $x = y = z$.
- Si $|y| > 1$, on applique le lemme de Levi : il existe un mot t tel que $y = xt$ et $y = tz$. On a donc $xt = tz$ et puisque $x \neq \varepsilon$, $|t| < |y|$. On applique l'hypothèse de récurrence à t : il existe deux mots u et v tels que $x = uv$, $t = (uv)^k u = u(vu)^k$ et $z = vu$. Alors $y = xt = tz = (uv)^{k+1} u = u(vu)^{k+1}$.

Lemme de LEVI et conséquences

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Soient x et $y \in \Sigma^*$ tels que $xy = yx$, avec $x \neq \varepsilon$ et $y \neq \varepsilon$. Alors il existe un mot $u \in \Sigma^*$ et deux entiers i et j tels que $x = u^i$ et $y = u^j$.

Lemme de LEVI et conséquences

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Soient x et $y \in \Sigma^*$ tels que $xy = yx$, avec $x \neq \varepsilon$ et $y \neq \varepsilon$. Alors il existe un mot $u \in \Sigma^*$ et deux entiers i et j tels que $x = u^i$ et $y = u^j$.

On raisonne par induction sur $|xy|$.

Lemme de LEVI et conséquences

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Soient x et $y \in \Sigma^*$ tels que $xy = yx$, avec $x \neq \varepsilon$ et $y \neq \varepsilon$. Alors il existe un mot $u \in \Sigma^*$ et deux entiers i et j tels que $x = u^i$ et $y = u^j$.

On raisonne par induction sur $|xy|$.

- Si $|xy| = 2$ alors $|x| = |y| = 1$ et $x = y$. On pose $u = x = y$ et $i = j = 1$.

Lemme de Levi et conséquences

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Soient x et $y \in \Sigma^*$ tels que $xy = yx$, avec $x \neq \varepsilon$ et $y \neq \varepsilon$. Alors il existe un mot $u \in \Sigma^*$ et deux entiers i et j tels que $x = u^i$ et $y = u^j$.

On raisonne par induction sur $|xy|$.

- Si $|xy| = 2$ alors $|x| = |y| = 1$ et $x = y$. On pose $u = x = y$ et $i = j = 1$.
- Si $|xy| > 1$, on applique le théorème précédent : il existe deux mots u et v et un entier k tels que $x = uv = vu$ et $y = (uv)^k u = u(vu)^k$.

Lemme de Levi et conséquences

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Soient x et $y \in \Sigma^*$ tels que $xy = yx$, avec $x \neq \varepsilon$ et $y \neq \varepsilon$. Alors il existe un mot $u \in \Sigma^*$ et deux entiers i et j tels que $x = u^i$ et $y = u^j$.

On raisonne par induction sur $|xy|$.

- Si $|xy| = 2$ alors $|x| = |y| = 1$ et $x = y$. On pose $u = x = y$ et $i = j = 1$.
- Si $|xy| > 1$, on applique le théorème précédent : il existe deux mots u et v et un entier k tels que $x = uv = vu$ et $y = (uv)^k u = u(vu)^k$.

Si $u = \varepsilon$ ou $v = \varepsilon$ alors $y = x^k$ ou $y = x^{k+1}$ et on pose $u = x$, $i = 1$ et $j = k$ ou $j = k + 1$.

Lemme de Levi et conséquences

Soient x, y et $z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u et v et un entier $k \in \mathbb{N}$ tels que :

$$x = uv, \quad y = (uv)^k u = u(vu)^k, \quad z = vu.$$

Soient x et $y \in \Sigma^*$ tels que $xy = yx$, avec $x \neq \varepsilon$ et $y \neq \varepsilon$. Alors il existe un mot $u \in \Sigma^*$ et deux entiers i et j tels que $x = u^i$ et $y = u^j$.

On raisonne par induction sur $|xy|$.

- Si $|xy| = 2$ alors $|x| = |y| = 1$ et $x = y$. On pose $u = x = y$ et $i = j = 1$.
- Si $|xy| > 1$, on applique le théorème précédent : il existe deux mots u et v et un entier k tels que $x = uv = vu$ et $y = (uv)^k u = u(vu)^k$.

Si $u \neq \varepsilon$ et $v \neq \varepsilon$, puisque $y \neq \varepsilon$ on a $|uv| = |x| < |xy|$ donc on peut appliquer l'hypothèse de récurrence : il existe un mot w et deux entiers i et j tels que $u = w^i$ et $v = w^j$. Dans ce cas, $x = w^{i+j}$ et $y = w^{(k+1)i+kj}$.

Distance entre mots

Distance préfixe

On note $\text{plpc}(u,v)$ le *plus long préfixe commun* à deux mots u et v, et on pose : $d(u,v) = |uv| - 2|\text{plpc}(u,v)|$.

Distance entre mots

Distance préfixe

On note $\text{plpc}(u,v)$ le *plus long préfixe commun* à deux mots u et v, et on pose : $d(u,v) = |uv| - 2|\text{plpc}(u,v)|$.

- $d(u,v) \geqslant 0$;
- $d(u,v) = 0 \iff u = v$;
- $d(u,w) \leqslant d(u,v) + d(v,w)$.

Distance entre mots

Distance préfixe

On note $\text{plpc}(u,v)$ le *plus long préfixe commun* à deux mots u et v, et on pose : $d(u,v) = |uv| - 2|\text{plpc}(u,v)|$.

- $d(u,v) \geqslant 0$;
- $d(u,v) = 0 \iff u = v$;
- $d(u,w) \leqslant d(u,v) + d(v,w)$.

La troisième propriété revient à prouver que :

$$|\text{plpc}(u,v)| + |\text{plpc}(v,w)| \leqslant |v| + |\text{plpc}(u,w)|.$$

Distance entre mots

Distance préfixe

On note $\text{plpc}(u,v)$ le *plus long préfixe commun* à deux mots u et v, et on pose : $d(u,v) = |uv| - 2|\text{plpc}(u,v)|$.

- $d(u,v) \geqslant 0$;
- $d(u,v) = 0 \iff u = v$;
- $d(u,w) \leqslant d(u,v) + d(v,w)$.

La troisième propriété revient à prouver que :

$$|\text{plpc}(u,v)| + |\text{plpc}(v,w)| \leqslant |v| + |\text{plpc}(u,w)|.$$

Le plus court des deux préfixes $\text{plpc}(u,v)$ et $\text{plpc}(v,w)$ est commun à u, v et w donc

$$\min(|\text{plpc}(u,v)|, |\text{plpc}(v,w)|) \leqslant |\text{plpc}(u,w)|.$$

Les deux préfixes $\text{plpc}(u,v)$ et $\text{plpc}(v,w)$ sont préfixes de $|v|$ donc

$$\max(|\text{plpc}(u,v)|, |\text{plpc}(v,w)|) \leqslant |v|.$$

Distance entre mots

distance suffixe, distance des facteurs, distance des sous-mots

On obtient d'autres distances en remplaçant le plus long préfixe commun par :

- le plus long suffixe commun ;
- le plus long facteur commun ;
- le plus long sous-mot commun.

Distance entre mots

distance suffixe, distance des facteurs, distance des sous-mots

On obtient d'autres distances en remplaçant le plus long préfixe commun par :

- le plus long suffixe commun ;
- le plus long facteur commun ;
- le plus long sous-mot commun.

Inégalité triangulaire avec la distance des sous-mots :
On note I et J les indices des lettres de v qui correspondent à $\operatorname{plsmc}(u,v)$ et à $\operatorname{plsmc}(v,w)$. Alors

$$|\operatorname{plsmc}(u,v)| + |\operatorname{plsmc}(v,w)| = |I| + |J| = |I \cup J| + |I \cap J|.$$

Distance entre mots

distance suffixe, distance des facteurs, distance des sous-mots

On obtient d'autres distances en remplaçant le plus long préfixe commun par :

- le plus long suffixe commun ;
- le plus long facteur commun ;
- le plus long sous-mot commun.

Inégalité triangulaire avec la distance des sous-mots :
On note I et J les indices des lettres de v qui correspondent à $\text{plsmc}(u, v)$ et à $\text{plsmc}(v, w)$. Alors

$$|\text{plsmc}(u, v)| + |\text{plsmc}(v, w)| = |I| + |J| = |I \cup J| + |I \cap J|.$$

Le sous-mot de v constitué des lettres dont les indices appartiennent à $I \cap J$ est un sous-mot commun à u, v et w donc $|I \cap J| \leqslant |\text{plsmc}(u, w)|$.
Par ailleurs $|I \cup J| \leqslant |v|$, donc :

$$|\text{plsmc}(u, v)| + |\text{plsmc}(v, w)| \leqslant |v| + |\text{plsmc}(u, w)|.$$

Distance entre mots

Distance d'édition

La distance de Levenshtein est le nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour transformer un mot u en un mot v. Par exemple, la distance entre **`polynomial`** et **`polygonal`** est égale à 3 :

- suppression de la lettre **`'i'`** : **`polynomial`** → **`polynomal`** ;
- remplacement du **`'n'`** par un **`'g'`** : **`polynomal`** → **`polygomal`** ;
- remplacement du **`'m'`** par un **`'n'`** : **`polygomal`** → **`polygonal`** ;

Le calcul de la distance d'édition est un problème classique de la programmation dynamique.

Distance entre mots

Distance d'édition

La distance de LEVENSHTEIN est le nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour transformer un mot u en un mot v. Par exemple, la distance entre **`polynomial`** et **`polygonal`** est égale à 3 :

- suppression de la lettre **`'i'`** : **`polynomial`** → **`polynomal`** ;
- remplacement du **`'n'`** par un **`'g'`** : **`polynomal`** → **`polygomal`** ;
- remplacement du **`'m'`** par un **`'n'`** : **`polygomal`** → **`polygonal`** ;

Le calcul de la distance d'édition est un problème classique de la programmation dynamique.

Exemple. Les deux séquences génétiques ci-dessous sont à une distance égale à 3 :

```
A C C T C T - A A T C T A T T C G T A C T G C T A T T
A C C T C T G A A T C C A T T C G T - C T G C T A T T
```

Distance entre mots

Distance d'édition

La distance de LEVENSHTEIN est le nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour transformer un mot u en un mot v. Par exemple, la distance entre **`polynomial`** et **`polygonal`** est égale à 3 :

- suppression de la lettre **`'i'`** : **`polynomial`** → **`polynomal`** ;
- remplacement du **`'n'`** par un **`'g'`** : **`polynomal`** → **`polygomal`** ;
- remplacement du **`'m'`** par un **`'n'`** : **`polygomal`** → **`polygonal`** ;

Le calcul de la distance d'édition est un problème classique de la programmation dynamique.

Exemple. Les deux séquences génétiques ci-dessous sont à une distance égale à 3 :

```
A C C T C T - A A T C T A T T C G T A C T G C T A T T
A C C T C T G A A T C C A T T C G T - C T G C T A T T
```

ou à $2,5$ si on attribue le poids $0,5$ aux substitutions `A` ↔ `G` et `T` ↔ `C`.

Mots de Dick

Nous nous intéressons aux mots sur l'alphabet constitué des deux lettres (et) qui interviennent dans les expressions mathématiques syntaxiquement correctes. Pour faciliter la lecture, on pose $\Sigma = \{a, b\}$, où a désigne la parenthèse ouvrante et b la parenthèse fermante.

Mots de Dick

Nous nous intéressons aux mots sur l'alphabet constitué des deux lettres (et) qui interviennent dans les expressions mathématiques syntaxiquement correctes. Pour faciliter la lecture, on pose $\Sigma = \{a, b\}$, où a désigne la parenthèse ouvrante et b la parenthèse fermante.
L'ensemble $\mathscr{D}$ des expressions bien parenthésées (les *mots de* Dyck) peut être défini à l'aide des règles de construction suivantes :

$$\begin{cases} \varepsilon \in \mathscr{D} \\ (r, s) \in \mathscr{D}^2 \Longrightarrow arbs \in \mathscr{D} \end{cases}$$

Mots de Dick

Nous nous intéressons aux mots sur l'alphabet constitué des deux lettres (et) qui interviennent dans les expressions mathématiques syntaxiquement correctes. Pour faciliter la lecture, on pose $\Sigma = \{a, b\}$, où a désigne la parenthèse ouvrante et b la parenthèse fermante.
L'ensemble $\mathscr{D}$ des expressions bien parenthésées (les *mots de* Dyck) peut être défini à l'aide des règles de construction suivantes :

$$\begin{cases} \varepsilon \in \mathscr{D} \\ (r, s) \in \mathscr{D}^2 \Longrightarrow arbs \in \mathscr{D} \end{cases}$$

On définit un morphisme $\sigma : \Sigma^* \to \mathbb{Z}$ en posant : $\sigma(a) = 1$ et $\sigma(b) = -1$.

Un mot m de Σ^* appartient à $\mathscr{D}$ si et seulement si :

- $\sigma(m) = 0$;
- pour tout préfixe m' de m, on a $\sigma(m') \geqslant 0$.

Mots de DICK

Un mot m de Σ^* appartient à $\mathscr{D}$ si et seulement si :

- $\sigma(m) = 0$;
- pour tout préfixe m' de m, on a $\sigma(m') \geqslant 0$.

On note $\mathscr{D}'$ l'ensemble des mots vérifiant les propriétés ci-dessus.

Mots de Dick

Un mot m de Σ^* appartient à $\mathscr{D}$ si et seulement si :

- $\sigma(m) = 0$;
- pour tout préfixe m' de m, on a $\sigma(m') \geqslant 0$.

On note $\mathscr{D}'$ l'ensemble des mots vérifiant les propriétés ci-dessus.

Soit $m \in \mathscr{D}$. Montrons par induction que m appartient à $\mathscr{D}'$.

- Si $|m| = 0$, c'est clair puisque $m = \varepsilon$.

Mots de Dick

Un mot m de Σ^* appartient à $\mathscr{D}$ si et seulement si :

- $\sigma(m) = 0$;
- pour tout préfixe m' de m, on a $\sigma(m') \geqslant 0$.

On note $\mathscr{D}'$ l'ensemble des mots vérifiant les propriétés ci-dessus.

Soit $m \in \mathscr{D}$. Montrons par induction que m appartient à $\mathscr{D}'$.

- Si $|m| = 0$, c'est clair puisque $m = \varepsilon$.
- Si $|m| > 0$ on pose $m = arbs$ avec $(r, s) \in \mathscr{D}^2$. Par hypothèse d'induction, r et s appartiennent à $\mathscr{D}'$.
  $\sigma(m) = 1 + \sigma(r) - 1 + \sigma(s) = 0$, et les préfixes de m sont :
  - a avec $\sigma(a) = 1$;
  - ar' où r' est préfixe de r, et $\sigma(ar') = 1 + \sigma(r') \geqslant 1$;
  - arb avec $\sigma(arb) = 1 + 0 - 1 = 0$;
  - $arbs'$ où s' est préfixe de s, et $\sigma(arbs') = 1 + 0 - 1 + \sigma(s') \geqslant 0$.

  On en déduit que m est élément de $\mathscr{D}'$.

Mots de Dick

Un mot m de Σ^* appartient à $\mathscr{D}$ si et seulement si :

- $\sigma(m) = 0$;
- pour tout préfixe m' de m, on a $\sigma(m') \geqslant 0$.

On note $\mathscr{D}'$ l'ensemble des mots vérifiant les propriétés ci-dessus.

Soit $m \in \mathscr{D}'$. Montrons par induction que m appartient à $\mathscr{D}$.

- Si $|m| = 0$ c'est clair puisque $m = \varepsilon$.

Mots de Dick

Un mot m de Σ^* appartient à $\mathscr{D}$ si et seulement si :

- $\sigma(m) = 0$;
- pour tout préfixe m' de m, on a $\sigma(m') \geqslant 0$.

On note $\mathscr{D}'$ l'ensemble des mots vérifiant les propriétés ci-dessus.

Soit $m \in \mathscr{D}'$. Montrons par induction que m appartient à $\mathscr{D}$.

- Si $|m| = 0$ c'est clair puisque $m = \varepsilon$.
- Si $|m| > 0$, notons $m = m_1 \cdots m_p$, avec $m_i \in \{a, b\}$.
  m_1 est préfixe de m donc $\sigma(m_1) \geqslant 0$ et $m_1 = a$. On considère

  $$I = \left\{k \in [\![2, p]\!] \;\middle|\; \sigma(m_1 \cdots m_k) = 0\right\}$$

  $I \neq \emptyset$ car $p \in I$ donc possède un plus petit élément $k \geqslant 2$.
  Puisque $k - 1 \notin I$, $\sigma(m_1 \cdots m_{k-1}) \geqslant 1$ donc $m_k = b$. Posons alors $r = m_2 \cdots m_{k-1}$ et $s = m_{k+1} \cdots m_p$; ainsi, $m = arbs$.

Mots de Dick

Un mot m de Σ^* appartient à $\mathscr{D}$ si et seulement si :

- $\sigma(m) = 0$;
- pour tout préfixe m' de m, on a $\sigma(m') \geqslant 0$.

On note $\mathscr{D}'$ l'ensemble des mots vérifiant les propriétés ci-dessus.

Puisque $\sigma(arb) = 0$, tout préfixe s' de s vérifie : $\sigma(s') \geqslant 0$, et $\sigma(s) = 0$. Ainsi, $s \in \mathscr{D}'$.
Puisque $\sigma(arb) = 0$, on a aussi $\sigma(r) = 0$. Enfin, si r' est un préfixe de r, $\sigma(ar') \geqslant 1$ (de par le caractère minimal de k) et donc $\sigma(r') \geqslant 0$. Ainsi, $r \in \mathscr{D}'$.

Mots de Dick

Un mot m de Σ^* appartient à $\mathscr{D}$ si et seulement si :

- $\sigma(m) = 0$;
- pour tout préfixe m' de m, on a $\sigma(m') \geqslant 0$.

On note $\mathscr{D}'$ l'ensemble des mots vérifiant les propriétés ci-dessus.

Puisque $\sigma(arb) = 0$, tout préfixe s' de s vérifie : $\sigma(s') \geqslant 0$, et $\sigma(s) = 0$. Ainsi, $s \in \mathscr{D}'$.
Puisque $\sigma(arb) = 0$, on a aussi $\sigma(r) = 0$. Enfin, si r' est un préfixe de r, $\sigma(ar') \geqslant 1$ (de par le caractère minimal de k) et donc $\sigma(r') \geqslant 0$. Ainsi, $r \in \mathscr{D}'$.

Par hypothèse d'induction, r et s appartiennent à $\mathscr{D}$, donc m aussi.

Mots de Dick

Nombres de Catalan

On note c_n le nombre de mots de $\mathscr{D}$ de longueur $2n$. La suite $(c_n)_{n\in\mathbb{N}}$ est définie par : $c_0 = 1$ et la relation : $\forall n \in \mathbb{N}$, $c_{n+1} = \sum_{p+q=n} c_p c_q$.

(Un mot de Dick de longueur $2n+2$ s'écrit de manière unique sous la forme $arbs$, ou r et s sont des mots de L de longueurs respectives $2p$ et $2q$ vérifiant $p+q=n$.)

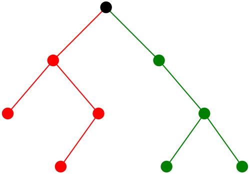
Mots de DICK

Nombres de CATALAN

On note c_n le nombre de mots de $\mathscr{D}$ de longueur $2n$. La suite $(c_n)_{n\in\mathbb{N}}$ est définie par : $c_0 = 1$ et la relation : $\forall n \in \mathbb{N}$, $c_{n+1} = \sum_{p+q=n} c_p c_q$.

c_n est aussi le nombre d'arbres binaires à n nœuds :

- au mot vide est associé l'arbre nil ;
- à un mot bien parenthésé de la forme *arbs* est associé l'arbre dont la racine a pour fils gauche l'arbre associé à r et pour fils droit l'arbre associé à s.



a aabbaabb b abaabbab

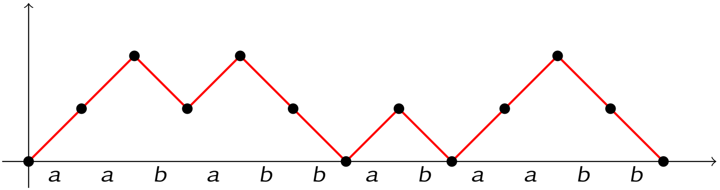
Lors d'un parcours en profondeur d'un arbre binaire, on note par un a le passage à gauche d'un nœud et par un b le passage sous un nœud.

Mots de Dick

Nombres de Catalan

On note c_n le nombre de mots de $\mathscr{D}$ de longueur $2n$. La suite $(c_n)_{n\in\mathbb{N}}$ est définie par : $c_0 = 1$ et la relation : $\forall n \in \mathbb{N}$, $c_{n+1} = \sum_{p+q=n} c_p c_q$.

c_n est aussi le nombre de chemins du plan situés dans le demi plan des ordonnées positives, débutant en $(0,0)$ et se terminant en $(2n,0)$, constitués d'une suite de segments de coordonnées vectorielles $(1,1)$ ou $(1,-1)$.



Mots de Dick

Calcul des nombres de Catalan

Calcul analytique.

On pose $f(x) = \sum_{n=0}^{+\infty} c_n x^n$ et on suppose le rayon de cv strictement positif.

On effectue le produit de Cauchy :

$$f(x)^2 = \sum_{n=0}^{+\infty}\Big(\sum_{p+q=n} c_p c_q\Big)x^n = \sum_{n=0}^{+\infty} c_{n+1}x^n$$

qui conduit à la relation : $xf(x)^2 = f(x) - 1$.

On en déduit : $f(x) = \dfrac{1-\sqrt{1+4x}}{2x}$, avec un rayon de cv égal à 1/4.

Le calcul effectif du développement en série entière donne l'expression :

$$c_n = \frac{1}{n+1}\binom{2n}{n}.$$

Mots de Dick

Calcul des nombres de Catalan

Calcul par dénombrement.
On considère un mot de longueur $2n$ comportant autant de a que de b mais mal parenthésé : $m = m_1 \cdots m_k \cdots m_{2n}$.
Soit k le plus petit entier pour lequel $\sigma(m_1 \cdots m_k) < 0$. Alors $m_k = b$ et $\sigma(m_1 \cdots m_{k-1}) = 0$, donc il y a autant de a que de b dans $m_1 \cdots m_{k-1}$.

Il y a donc un a de plus que de b dans $m_{k+1} \cdots m_{2n}$. On associe au mot m le mot $m' = m_1 \cdots m_k \overline{m}_{k+1} \cdots \overline{m}_{2n}$ (avec $\overline{a} = b$ et $\overline{b} = a$).

Mots de Dick

Calcul des nombres de Catalan

Calcul par dénombrement.

On considère un mot de longueur $2n$ comportant autant de a que de b mais mal parenthésé : $m = m_1 \cdots m_k \cdots m_{2n}$.

Soit k le plus petit entier pour lequel $\sigma(m_1 \cdots m_k) < 0$. Alors $m_k = b$ et $\sigma(m_1 \cdots m_{k-1}) = 0$, donc il y a autant de a que de b dans $m_1 \cdots m_{k-1}$.

Il y a donc un a de plus que de b dans $m_{k+1} \cdots m_{2n}$. On associe au mot m le mot $m' = m_1 \cdots m_k \overline{m}_{k+1} \cdots \overline{m}_{2n}$ (avec $\overline{a} = b$ et $\overline{b} = a$).

$$\text{Soit} \quad B_1 = \left\{ m \in \Sigma^{2n} \;\middle|\; |m|_a = n \text{ et } |m|_b = n \right\} \quad \text{et}$$
$$B_2 = \left\{ m \in \Sigma^{2n} \;\middle|\; |m|_a = n-1 \text{ et } |m|_b = n+1 \right\}.$$

L'association entre m et m' établit une bijection entre $B_1 \setminus \mathscr{D}$ et B_2.

Mots de Dick

Calcul des nombres de Catalan

Calcul par dénombrement.
On considère un mot de longueur $2n$ comportant autant de a que de b mais mal parenthésé : $m = m_1 \cdots m_k \cdots m_{2n}$.
Soit k le plus petit entier pour lequel $\sigma(m_1 \cdots m_k) < 0$. Alors $m_k = b$ et $\sigma(m_1 \cdots m_{k-1}) = 0$, donc il y a autant de a que de b dans $m_1 \cdots m_{k-1}$.

Il y a donc un a de plus que de b dans $m_{k+1} \cdots m_{2n}$. On associe au mot m le mot $m' = m_1 \cdots m_k \overline{m}_{k+1} \cdots \overline{m}_{2n}$ (avec $\overline{a} = b$ et $\overline{b} = a$).

$$\text{Soit} \quad B_1 = \left\{ m \in \Sigma^{2n} \,\middle|\, |m|_a = n \text{ et } |m|_b = n \right\} \quad \text{et}$$
$$B_2 = \left\{ m \in \Sigma^{2n} \,\middle|\, |m|_a = n-1 \text{ et } |m|_b = n+1 \right\}.$$

L'association entre m et m' établit une bijection entre $B_1 \setminus \mathscr{D}$ et B_2.

D'où : $\binom{2n}{n} - c_n = \binom{2n}{n+1}$, soit $c_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\binom{2n}{n}$.

Langages

Un langage L sur un alphabet Σ est une partie de Σ^*, autrement dit un ensemble de mots. Un langage peut être défini :

- en énumérant ses éléments : $L = \left\{a^n b^n \;\middle|\; n \in \mathbb{N}\right\}$;
- à l'aide d'une propriété (langage des palindromes) ;
- à l'aide d'une *grammaire formelle*.

$\mathscr{D}$ est défini par la grammaire : $\varepsilon \in L$ et $(r, s) \in L^2 \Longrightarrow arbs \in L$.

Langages

Un langage L sur un alphabet Σ est une partie de Σ^*, autrement dit un ensemble de mots. Un langage peut être défini :

- en énumérant ses éléments : $L = \left\{a^n b^n \;\middle|\; n \in \mathbb{N}\right\}$;
- à l'aide d'une propriété (langage des palindromes) ;
- à l'aide d'une *grammaire formelle*.

$\mathscr{D}$ est défini par la grammaire : $\varepsilon \in L$ et $(r, s) \in L^2 \Longrightarrow arbs \in L$.

Un langage L est dit :

- récursivement énumérable s'il existe un algorithme qui énumère tous les mots de L ;
- récursif s'il existe un algorithme qui, prenant un mot u de Σ^*, retourne **`true`** si u et dans L et **`false`** sinon.

Tout langage récursif est récursivement énumérable : il suffit de parcourir les mots de Σ^* et de soumettre chacun d'eux à l'algorithme qui détermine s'il appartient ou pas à L. La réciproque est fausse.

Seuls les langages récursifs vont nous intéresser.

Opérations sur les langages

Toutes les opérations ensemblistes sont applicables aux langages : réunion (souvent notée $+$ au lieu de $\cup$), intersection, complémentation …

Opérations sur les langages

Toutes les opérations ensemblistes sont applicables aux langages : réunion (souvent notée $+$ au lieu de $\cup$), intersection, complémentation ...
La concaténation de deux langages L_1 et L_2 se définit par :

$$L_1L_2 = \left\{u \in \Sigma^* \,\middle|\, \exists(x,y) \in L_1 \times L_2 \text{ tel que } u = xy\right\}$$

On définit le langage L^n en posant $L^0 = \{\varepsilon\}$ et $L^{n+1} = L^nL = LL^n$.

Opérations sur les langages

Toutes les opérations ensemblistes sont applicables aux langages : réunion (souvent notée $+$ au lieu de $\cup$), intersection, complémentation ...
La concaténation de deux langages L_1 et L_2 se définit par :

$$L_1L_2 = \left\{u \in \Sigma^* \;\middle|\; \exists(x,y) \in L_1 \times L_2 \text{ tel que } u = xy\right\}$$

On définit le langage L^n en posant $L^0 = \{\varepsilon\}$ et $L^{n+1} = L^nL = LL^n$.

Attention : ne pas confondre $L^2 = \left\{uv \;\middle|\; u, v \in L\right\}$ avec le langage des carrés de L : $\left\{u^2 \;\middle|\; u \in L\right\}$.

Opérations sur les langages

Toutes les opérations ensemblistes sont applicables aux langages : réunion (souvent notée $+$ au lieu de $\cup$), intersection, complémentation ...
La concaténation de deux langages L_1 et L_2 se définit par :

$$L_1L_2 = \left\{u \in \Sigma^* \;\middle|\; \exists (x,y) \in L_1 \times L_2 \text{ tel que } u = xy\right\}$$

On définit le langage L^n en posant $L^0 = \{\varepsilon\}$ et $L^{n+1} = L^nL = LL^n$.

La *fermeture de* Kleene de L se définit par : $L^* = \bigcup_{n\in\mathbb{N}} L^n$.

L^* est l'ensemble des mots que l'on peut construire en concaténant un nombre fini (éventuellement réduit à zéro) d'éléments de L.

Opérations sur les langages

Toutes les opérations ensemblistes sont applicables aux langages : réunion (souvent notée $+$ au lieu de $\cup$), intersection, complémentation …
La concaténation de deux langages L_1 et L_2 se définit par :

$$L_1L_2 = \left\{u \in \Sigma^* \,\middle|\, \exists (x,y) \in L_1 \times L_2 \text{ tel que } u = xy\right\}$$

On définit le langage L^n en posant $L^0 = \{\varepsilon\}$ et $L^{n+1} = L^nL = LL^n$.

La *fermeture de* Kleene de L se définit par : $L^* = \bigcup_{n\in\mathbb{N}} L^n$.

L^* est l'ensemble des mots que l'on peut construire en concaténant un nombre fini (éventuellement réduit à zéro) d'éléments de L.
On peut aussi définir $L^+ = \bigcup_{n\in\mathbb{N}^*} L^n$. À la différence de L^* qui contient toujours le mot vide ε, L^+ ne le contient que si L le contient. On a $L^+ = LL^*$.

Opérations sur les langages

Toutes les opérations ensemblistes sont applicables aux langages : réunion (souvent notée $+$ au lieu de $\cup$), intersection, complémentation ...
La concaténation de deux langages L_1 et L_2 se définit par :

$$L_1L_2 = \left\{ u \in \Sigma^* \;\middle|\; \exists(x,y) \in L_1 \times L_2 \text{ tel que } u = xy \right\}$$

On définit le langage L^n en posant $L^0 = \{\varepsilon\}$ et $L^{n+1} = L^nL = LL^n$.

La *fermeture de* KLEENE de L se définit par : $L^* = \bigcup_{n\in\mathbb{N}} L^n$.

L^* est l'ensemble des mots que l'on peut construire en concaténant un nombre fini (éventuellement réduit à zéro) d'éléments de L.

D'autres opérations sur les langages existent :

- $\sqrt{L} = \left\{ u \in \Sigma^* \;\middle|\; u^2 \in L \right\}$ (racine carrée de L) ;

Opérations sur les langages

Toutes les opérations ensemblistes sont applicables aux langages : réunion (souvent notée $+$ au lieu de $\cup$), intersection, complémentation ...
La concaténation de deux langages L_1 et L_2 se définit par :

$$L_1L_2 = \left\{u \in \Sigma^* \,\middle|\, \exists (x,y) \in L_1 \times L_2 \text{ tel que } u = xy\right\}$$

On définit le langage L^n en posant $L^0 = \{\varepsilon\}$ et $L^{n+1} = L^nL = LL^n$.

La *fermeture de* Kleene de L se définit par : $L^* = \bigcup_{n \in \mathbb{N}} L^n$.

L^* est l'ensemble des mots que l'on peut construire en concaténant un nombre fini (éventuellement réduit à zéro) d'éléments de L.

D'autres opérations sur les langages existent :

- $\sqrt{L} = \left\{u \in \Sigma^* \,\middle|\, u^2 \in L\right\}$ (racine carrée de L) ;
- $K^{-1}L = \left\{v \in \Sigma^* \,\middle|\, \exists u \in K \text{ tel que } uv \in L\right\}$ (quotient gauche de L par K).

Expressions rationnelles

Objectif : introduire un formalisme efficace pour décrire certains langages par des motifs.

Expressions rationnelles

Objectif : introduire un formalisme efficace pour décrire certains langages par des motifs.

Soit Σ un alphabet. Les expressions rationnelles sont définies inductivement par :

- $\emptyset$ et ε sont des expressions rationnelles ;
- $\forall a \in \Sigma$, a est une expression rationnelle ;
- Si e_1 et e_2 sont des expressions rationnelles, alors $e_1 + e_2$, $e_1 e_2$ et e^* sont des expressions rationnelles.

Expressions rationnelles

Objectif : introduire un formalisme efficace pour décrire certains langages par des motifs.

Soit Σ un alphabet. Les expressions rationnelles sont définies inductivement par :

- $\emptyset$ et ε sont des expressions rationnelles ;
- $\forall a \in \Sigma$, a est une expression rationnelle ;
- Si e_1 et e_2 sont des expressions rationnelles, alors $e_1 + e_2$, $e_1 e_2$ et e^* sont des expressions rationnelles.

L'interprétation d'une expression rationnelle est définie par :

- $\emptyset$ dénote le langage vide et ε le langage $\{\varepsilon\}$;
- $\forall a \in \Sigma$, a dénote le langage $\{a\}$;
- $e_1 + e_2$ dénote l'union des langages dénotés par e_1 et e_2 ;
- $e_1 e_2$ dénote la concaténation des langages dénotés par e_1 et e_2 ;
- e^* dénote la fermeture de Kleene du langage dénoté par e.

Un langage dénoté par une expression rationnelle sera qualifié de langage rationnel.

Expressions rationnelles

Exemples

On pose $\Sigma = \{a, b\}$.

- $a^* b^*$ dénote le langage $\left\{ a^p b^q \,\middle|\, (p,q) \in \mathbb{N}^2 \right\}$;

Expressions rationnelles

Exemples

On pose $\Sigma = \{a, b\}$.

- a^*b^* dénote le langage $\left\{a^p b^q \;\middle|\; (p,q) \in \mathbb{N}^2\right\}$;
- $(ab)^*$ dénote le langage $\left\{(ab)^n \;\middle|\; n \in \mathbb{N}\right\}$. Notons que l'expression $\varepsilon + a(ba)^*b$ dénote le même langage ;

Expressions rationnelles

Exemples

On pose $\Sigma = \{a, b\}$.

- a^*b^* dénote le langage $\left\{a^p b^q \;\middle|\; (p,q) \in \mathbb{N}^2\right\}$;
- $(ab)^*$ dénote le langage $\left\{(ab)^n \;\middle|\; n \in \mathbb{N}\right\}$. Notons que l'expression $\varepsilon + a(ba)^*b$ dénote le même langage ;
- $(a+b)^*aaa(a+b)^*$ dénote le langage des mots dont aaa est un facteur ;

Expressions rationnelles

Exemples

On pose $\Sigma = \{a, b\}$.

- a^*b^* dénote le langage $\left\{a^p b^q \;\middle|\; (p,q) \in \mathbb{N}^2\right\}$;
- $(ab)^*$ dénote le langage $\left\{(ab)^n \;\middle|\; n \in \mathbb{N}\right\}$. Notons que l'expression $\varepsilon + a(ba)^*b$ dénote le même langage ;
- $(a+b)^*aaa(a+b)^*$ dénote le langage des mots dont aaa est un facteur ;
- $(a+ba)^*$ dénote le langage des mots dans lesquels chaque b est suivi d'un a ;

Expressions rationnelles

Exemples

On pose $\Sigma = \{a, b\}$.

- a^*b^* dénote le langage $\left\{a^p b^q \,\middle|\, (p, q) \in \mathbb{N}^2\right\}$;
- $(ab)^*$ dénote le langage $\left\{(ab)^n \,\middle|\, n \in \mathbb{N}\right\}$. Notons que l'expression $\varepsilon + a(ba)^*b$ dénote le même langage ;
- $(a + b)^*aaa(a + b)^*$ dénote le langage des mots dont aaa est un facteur ;
- $(a + ba)^*$ dénote le langage des mots dans lesquels chaque b est suivi d'un a ;
- $(a^*b)^*$ dénote le langage formé du mot vide et de tous les mots qui se terminent par un b. Il peut aussi être dénoté par $\varepsilon + (a + b)^*b$.

Expressions rationnelles

Exemples

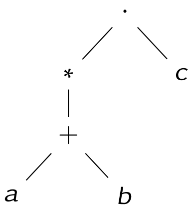
On pose $\Sigma = \{a, b\}$.

- a^*b^* dénote le langage $\left\{a^p b^q \,\middle|\, (p, q) \in \mathbb{N}^2\right\}$;
- $(ab)^*$ dénote le langage $\left\{(ab)^n \,\middle|\, n \in \mathbb{N}\right\}$. Notons que l'expression $\varepsilon + a(ba)^*b$ dénote le même langage ;
- $(a+b)^*aaa(a+b)^*$ dénote le langage des mots dont aaa est un facteur ;
- $(a+ba)^*$ dénote le langage des mots dans lesquels chaque b est suivi d'un a ;
- $(a^*b)^*$ dénote le langage formé du mot vide et de tous les mots qui se terminent par un b. Il peut aussi être dénoté par $\varepsilon + (a+b)^*b$.

L'ensemble $\mathrm{Rat}(\Sigma)$ des langages rationnels est la plus petite partie de Σ^* contenant $\emptyset$, $\{\varepsilon\}$, $\{a\}$ pour tout $a \in \Sigma$ et stable par réunion, concaténation et passage à l'étoile.

Arbre associé à une expression rationnelle

On peut associer à toute expression rationnelle un arbre binaire où les feuilles sont éléments de $\Sigma \cup \{\emptyset, \varepsilon\}$ et les nœuds les opérations $\{+, \cdot, *\}$.



arbre associé à $(a+b)^*c$.

Ceci permet de prouver certaines propriétés par induction sur la profondeur de l'expression rationnelle.

Équivalences et réductions

La correspondance entre expression et langage n'est pas biunivoque : chaque expression dénote un unique langage, mais à un langage donné peuvent correspondre plusieurs expressions différentes. On dit que deux expressions rationnelles sont équivalentes lorsqu'elles dénotent le même langage.

Équivalences et réductions

La correspondance entre expression et langage n'est pas biunivoque : chaque expression dénote un unique langage, mais à un langage donné peuvent correspondre plusieurs expressions différentes. On dit que deux expressions rationnelles sont équivalentes lorsqu'elles dénotent le même langage.

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant pas le symbole $\emptyset$.

Équivalences et réductions

La correspondance entre expression et langage n'est pas biunivoque : chaque expression dénote un unique langage, mais à un langage donné peuvent correspondre plusieurs expressions différentes. On dit que deux expressions rationnelles sont équivalentes lorsqu'elles dénotent le même langage.

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant pas le symbole $\emptyset$.

On procède par induction sur e.

- Si $e = \varepsilon$ ou $e \in \Sigma$, il suffit de poser $e' = e$.

Équivalences et réductions

La correspondance entre expression et langage n'est pas biunivoque : chaque expression dénote un unique langage, mais à un langage donné peuvent correspondre plusieurs expressions différentes. On dit que deux expressions rationnelles sont équivalentes lorsqu'elles dénotent le même langage.

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant pas le symbole $\emptyset$.

On procède par induction sur e.

- Si $e = e_1 + e_2$, alors $L = L_1 \cup L_2$, où L_1 et L_2 sont les langages dénotés par e_1 et e_2.
  - si $L_1 = \emptyset$ alors $L = L_2 \neq \emptyset$ et il existe une expression régulière e_2' sans symbole $\emptyset$ tel que $e \equiv e_2 \equiv e_2'$;
  - si $L_2 = \emptyset$ alors $L = L_1 \neq \emptyset$ et il existe une expression régulière e_1' sans symbole $\emptyset$ tel que $e \equiv e_1 \equiv e_1'$;
  - Si $L_1 \neq \emptyset$ et $L_2 \neq \emptyset$ il existe des expressions e_1' et e_2' équivalentes à e_1 et e_2 et ne contenant pas le symbole $\emptyset$, et $e \equiv e_1' + e_2'$.

Équivalences et réductions

La correspondance entre expression et langage n'est pas biunivoque : chaque expression dénote un unique langage, mais à un langage donné peuvent correspondre plusieurs expressions différentes. On dit que deux expressions rationnelles sont équivalentes lorsqu'elles dénotent le même langage.

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant pas le symbole $\emptyset$.

On procède par induction sur e.

- Si $e = e_1 e_2$ aucune des deux expressions rationnelles e_1 et e_2 ne peut dénoter l'ensemble vide, donc par hypothèse d'induction il existe des expressions rationnelles e_1' et e_2' sans symbole $\emptyset$ équivalentes respectivement à e_1 et e_2 et alors $e \equiv e_1' e_2'$.

Équivalences et réductions

La correspondance entre expression et langage n'est pas biunivoque : chaque expression dénote un unique langage, mais à un langage donné peuvent correspondre plusieurs expressions différentes. On dit que deux expressions rationnelles sont équivalentes lorsqu'elles dénotent le même langage.

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant pas le symbole $\emptyset$.

On procède par induction sur e.

- si $e = e_1^*$ alors ou bien e_1 dénote le langage vide auquel cas $e \equiv \varepsilon$, ou bien e_1 ne dénote pas le langage vide auquel cas il est équivalent à une expression rationnelle e_1' n'utilisant pas le symbole $\emptyset$ et dans ce cas $e \equiv {e_1'}^*$.

Équivalences et réductions

Mise en œuvre

On se restreint désormais aux expressions rationnelles ne contenant pas le symbole $\emptyset$, et on utilisera le type CAML suivant pour représenter en machine une expression rationnelle :

```
type regexp =
  | Epsilon
  | Const of string
  | Sum of regexp * regexp
  | Concat of regexp * regexp
  | Kleene of regexp ;;
```

Par exemple, l'expression rationnelle $(a+b)^*c$ est représentée en CAML par :

```
let e = Concat (Kleene (Sum (Const "a", Const "b")), Const "c") ;;
```

Équivalences et réductions

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant ni le symbole $\emptyset$, ni le symbole ε, telle que e soit équivalente à ε, e' ou à $\varepsilon + e'$.

Équivalences et réductions

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant ni le symbole $\emptyset$, ni le symbole ε, telle que e soit équivalente à ε, e' ou à $\varepsilon + e'$.

On peut déjà supposer que e ne contient pas le symbole $\emptyset$. Raisonnons alors de nouveau par induction sur e.

Équivalences et réductions

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant ni le symbole $\emptyset$, ni le symbole ε, telle que e soit équivalente à ε, e' ou à $\varepsilon + e'$.

On peut déjà supposer que e ne contient pas le symbole $\emptyset$. Raisonnons alors de nouveau par induction sur e.

- Si $e = \varepsilon$ ou $e \in \Sigma$ le résultat est évident.

Équivalences et réductions

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant ni le symbole $\emptyset$, ni le symbole ε, telle que e soit équivalente à ε, e' ou à $\varepsilon + e'$.

On peut déjà supposer que e ne contient pas le symbole $\emptyset$. Raisonnons alors de nouveau par induction sur e.

- Si $e = e_1 + e_2$, on applique l'hypothèse d'induction à e_1 et e_2 et suivant les cas on obtient l'une des formes équivalentes à e suivantes :

$+$	ε	e_2'	$\varepsilon + e_2'$
ε	ε	$\varepsilon + e_2'$	$\varepsilon + e_2'$
e_1'	$\varepsilon + e_1'$	$e_1' + e_2'$	$\varepsilon + (e_1' + e_2')$
$\varepsilon + e_1'$	$\varepsilon + e_1'$	$\varepsilon + (e_1' + e_2')$	$\varepsilon + (e_1' + e_2')$

Équivalences et réductions

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant ni le symbole $\emptyset$, ni le symbole ε, telle que e soit équivalente à ε, e' ou à $\varepsilon + e'$.

On peut déjà supposer que e ne contient pas le symbole $\emptyset$. Raisonnons alors de nouveau par induction sur e.

- Si $e = e_1 e_2$, on applique l'hypothèse d'induction à e_1 et e_2 et suivant les cas on obtient l'une des formes équivalentes à e suivantes :

$\cdot$	ε	e_2'	$\varepsilon + e_2'$
ε	ε	e_2'	$\varepsilon + e_2'$
e_1'	e_1'	$e_1' e_2'$	$e_1' + e_1' e_2'$
$\varepsilon + e_1'$	$\varepsilon + e_1'$	$e_2' + e_1' e_2'$	$\varepsilon + (e_1' + e_2' + e_1' e_2')$

Équivalences et réductions

Si le langage dénoté par l'expression régulière e est non vide, il existe une expression équivalente e' ne contenant ni le symbole $\emptyset$, ni le symbole ε, telle que e soit équivalente à ε, e' ou à $\varepsilon + e'$.

On peut déjà supposer que e ne contient pas le symbole $\emptyset$. Raisonnons alors de nouveau par induction sur e.

- Si $e = e_1^*$ on applique l'hypothèse d'induction à e_1 et on utilise l'une des formules suivantes :

$$\varepsilon^* = \varepsilon \qquad {e_1'}^* = {e_1'}^* \qquad (\varepsilon + e_1')^* = {e_1'}^*.$$

Langages locaux

Si L est un langage sur l'alphabet Σ on pose :

- $P(L) = \left\{a \in \Sigma \;\middle|\; a\Sigma^* \cap L \neq \emptyset\right\}$ (premières lettres des mots de L) ;
- $S(L) = \left\{a \in \Sigma \;\middle|\; \Sigma^* a \cap L \neq \emptyset\right\}$ (dernières lettres des mots de L) ;
- $F(L) = \left\{u \in \Sigma^2 \;\middle|\; \Sigma^* u \Sigma^* \cap L \neq \emptyset\right\}$ (facteurs de long. 2 des mots de L).
- $N(L) = \Sigma^2 \setminus F(L)$.

De manière évidente :

$$L \setminus \{\varepsilon\} \subset \Big(P(L)\Sigma^* \cap \Sigma^* S(L)\Big) \setminus (\Sigma^* N(L) \Sigma^*)$$

Tout mot non vide de L a sa première lettre dans $P(L)$, sa dernière lettre dans $S(L)$, et aucun de ses facteurs de longueur 2 dans $N(L)$.

Langages locaux

Si L est un langage sur l'alphabet Σ on pose :

- $P(L) = \left\{a \in \Sigma \;\middle|\; a\Sigma^* \cap L \neq \emptyset\right\}$ (premières lettres des mots de L) ;
- $S(L) = \left\{a \in \Sigma \;\middle|\; \Sigma^* a \cap L \neq \emptyset\right\}$ (dernières lettres des mots de L) ;
- $F(L) = \left\{u \in \Sigma^2 \;\middle|\; \Sigma^* u \Sigma^* \cap L \neq \emptyset\right\}$ (facteurs de long. 2 des mots de L).
- $N(L) = \Sigma^2 \setminus F(L)$.

De manière évidente :

$$L \setminus \{\varepsilon\} \subset \Big(P(L)\Sigma^* \cap \Sigma^* S(L)\Big) \setminus (\Sigma^* N(L) \Sigma^*)$$

Tout mot non vide de L a sa première lettre dans $P(L)$, sa dernière lettre dans $S(L)$, et aucun de ses facteurs de longueur 2 dans $N(L)$.

Un langage L est dit local lorsqu'il existe deux parties P et S de Σ et une partie N de Σ^2 tels que :

$$L \setminus \{\varepsilon\} = (P\Sigma^* \cap \Sigma^* S) \setminus (\Sigma^* N \Sigma^*).$$

Dans ce cas, $P = P(L)$, $S = S(L)$, $N = N(L)$.

Langages locaux

Exemples

- le langage dénoté par a^* est local : $P = S = \{a\}$, $N = \{ab, ba, bb\}$;

Langages locaux

Exemples

- le langage dénoté par a^* est local : $P = S = \{a\}$, $N = \{ab, ba, bb\}$;
- le langage dénoté par $(ab)^*$ est local : $P = \{a\}$, $S = \{b\}$, $N = \{aa, bb\}$;

Langages locaux

Exemples

- le langage dénoté par a^* est local : $P = S = \{a\}$, $N = \{ab, ba, bb\}$;
- le langage dénoté par $(ab)^*$ est local : $P = \{a\}$, $S = \{b\}$, $N = \{aa, bb\}$;
- le langage dénoté par $L_1 = a^* + (ab)^*$ n'est pas local :

  on calcule $P(L_1) = \{a\}$, $S(L_1) = \{a, b\}$, $F(L_1) = \{aa, ab, ba\}$, $N(L_1) = \{bb\}$ *mais* $aba \notin L_1$;

Langages locaux

Exemples

- le langage dénoté par a^* est local : $P = S = \{a\}$, $N = \{ab, ba, bb\}$;
- le langage dénoté par $(ab)^*$ est local : $P = \{a\}$, $S = \{b\}$, $N = \{aa, bb\}$;
- le langage dénoté par $L_1 = a^* + (ab)^*$ n'est pas local :

  on calcule $P(L_1) = \{a\}$, $S(L_1) = \{a, b\}$, $F(L_1) = \{aa, ab, ba\}$, $N(L_1) = \{bb\}$ *mais* $aba \notin L_1$;

- le langage dénoté par $L_2 = a^*(ab)^*$ n'est pas local :

  on calcule $P(L_2) = \{a\}$, $S(L_2) = \{a, b\}$, $F(L_2) = \{aa, ab, ba\}$, $N(L_2) = \{bb\}$ *mais* $aba \notin L_2$;

Langages locaux

Exemples

- le langage dénoté par a^* est local : $P = S = \{a\}$, $N = \{ab, ba, bb\}$;
- le langage dénoté par $(ab)^*$ est local : $P = \{a\}$, $S = \{b\}$, $N = \{aa, bb\}$;
- le langage dénoté par $L_1 = a^* + (ab)^*$ n'est pas local :

  on calcule $P(L_1) = \{a\}$, $S(L_1) = \{a, b\}$, $F(L_1) = \{aa, ab, ba\}$, $N(L_1) = \{bb\}$ *mais* $aba \notin L_1$;

- le langage dénoté par $L_2 = a^*(ab)^*$ n'est pas local :

  on calcule $P(L_2) = \{a\}$, $S(L_2) = \{a, b\}$, $F(L_2) = \{aa, ab, ba\}$, $N(L_2) = \{bb\}$ *mais* $aba \notin L_2$;

On constate que la réunion et la concaténation de deux langages locaux n'est pas nécessairement local ; en revanche,

L'intersection de deux langages locaux est un langage local.

Langages locaux

L'intersection de deux langages locaux est un langage local.

Considérons deux langages locaux L_1 et L_2 et posons :

$$P = P(L_1) \cap P(L_2), \quad S = S(L_1) \cap S(L_2), \quad N = N(L_1) \cup N(L_2).$$

Langages locaux

L'intersection de deux langages locaux est un langage local.

Considérons deux langages locaux L_1 et L_2 et posons :

$$P = P(L_1) \cap P(L_2), \quad S = S(L_1) \cap S(L_2), \quad N = N(L_1) \cup N(L_2).$$

$$\begin{aligned}
(L_1 \cap L_2) \setminus \{\varepsilon\} &= (L_1 \setminus \{\varepsilon\}) \cap (L_2 \setminus \{\varepsilon\}) \\
&= \Big(P(L_1)\Sigma^* \cap \Sigma^* S(L_1)\Big) \setminus (\Sigma^* N(L_1)\Sigma^*) \cap \Big(P(L_2)\Sigma^* \cap \Sigma^* S(L_2)\Big) \setminus (\Sigma^* N(L_2)\Sigma^*) \\
&= \Big(P(L_1)\Sigma^* \cap \Sigma^* S(L_1) \cap P(L_2)\Sigma^* \cap \Sigma^* S(L_2)\Big) \setminus \Big(\Sigma^* N(L_1)\Sigma^* \cup \Sigma^* N(L_2)\Sigma^*\Big) \\
&= (P\Sigma^* \cap \Sigma^* S) \setminus (\Sigma^* N \Sigma^*)
\end{aligned}$$

donc $L_1 \cap L_2$ est bien local.

Langages locaux

L'intersection de deux langages locaux est un langage local.

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1 \cup L_2$ est local.

Langages locaux

L'intersection de deux langages locaux est un langage local.

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1 \cup L_2$ est local.

Notons Σ_1 et Σ_2 les alphabets sur lesquels sont définis L_1 et L_2, et $\Sigma = \Sigma_1 \cup \Sigma_2$.

On calcule $P(L_1 \cup L_2) = P(L_1) \cup P(L_2)$, $S(L_1 \cup L_2) = S(L_1) \cup S(L_2)$, $F(L_1 \cup L_2) = F(L_1) \cup F(L_2)$.

Langages locaux

L'intersection de deux langages locaux est un langage local.

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1 \cup L_2$ est local.

Notons Σ_1 et Σ_2 les alphabets sur lesquels sont définis L_1 et L_2, et $\Sigma = \Sigma_1 \cup \Sigma_2$.

On calcule $P(L_1 \cup L_2) = P(L_1) \cup P(L_2)$, $S(L_1 \cup L_2) = S(L_1) \cup S(L_2)$, $F(L_1 \cup L_2) = F(L_1) \cup F(L_2)$.

Soit $u = a_1 a_2 \cdots a_p \in (P(L_1 \cup L_2)\Sigma^* \cap \Sigma^* S(L_1 \cup L_2)) \setminus (\Sigma^* N(L_1 \cup L_2)\Sigma^*)$.

Langages locaux

L'intersection de deux langages locaux est un langage local.

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1 \cup L_2$ est local.

Notons Σ_1 et Σ_2 les alphabets sur lesquels sont définis L_1 et L_2, et $\Sigma = \Sigma_1 \cup \Sigma_2$.

On calcule $P(L_1 \cup L_2) = P(L_1) \cup P(L_2)$, $S(L_1 \cup L_2) = S(L_1) \cup S(L_2)$, $F(L_1 \cup L_2) = F(L_1) \cup F(L_2)$.

Soit $u = a_1 a_2 \cdots a_p \in (P(L_1 \cup L_2)\Sigma^* \cap \Sigma^* S(L_1 \cup L_2)) \setminus (\Sigma^* N(L_1 \cup L_2)\Sigma^*)$.

- $a_1 \in P(L_1) \cup P(L_2)$; on suppose par exemple $a_1 \in P(L_1)$. Alors $a_1 \in \Sigma_1$.

Langages locaux

L'intersection de deux langages locaux est un langage local.

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1 \cup L_2$ est local.

Notons Σ_1 et Σ_2 les alphabets sur lesquels sont définis L_1 et L_2, et $\Sigma = \Sigma_1 \cup \Sigma_2$.

On calcule $P(L_1 \cup L_2) = P(L_1) \cup P(L_2)$, $S(L_1 \cup L_2) = S(L_1) \cup S(L_2)$, $F(L_1 \cup L_2) = F(L_1) \cup F(L_2)$.

Soit $u = a_1 a_2 \cdots a_p \in (P(L_1 \cup L_2)\Sigma^* \cap \Sigma^* S(L_1 \cup L_2)) \setminus (\Sigma^* N(L_1 \cup L_2)\Sigma^*)$.

- $a_1 \in P(L_1) \cup P(L_2)$; on suppose par exemple $a_1 \in P(L_1)$. Alors $a_1 \in \Sigma_1$.
- $a_1 a_2 \in F(L_1) \cup F(L_2)$. Mais $F(L_1) \subset \Sigma_1^2$ et $F(L_2) \subset \Sigma_2^2$. Les deux alphabets étant disjoints et $a_1 \in \Sigma_1$ on a $a_1 a_2 \in F(L_1)$ et $a_2 \in \Sigma_1$.

Langages locaux

L'intersection de deux langages locaux est un langage local.

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1 \cup L_2$ est local.

Notons Σ_1 et Σ_2 les alphabets sur lesquels sont définis L_1 et L_2, et $\Sigma = \Sigma_1 \cup \Sigma_2$.

On calcule $P(L_1 \cup L_2) = P(L_1) \cup P(L_2)$, $S(L_1 \cup L_2) = S(L_1) \cup S(L_2)$, $F(L_1 \cup L_2) = F(L_1) \cup F(L_2)$.

Soit $u = a_1 a_2 \cdots a_p \in (P(L_1 \cup L_2)\Sigma^* \cap \Sigma^* S(L_1 \cup L_2)) \setminus (\Sigma^* N(L_1 \cup L_2)\Sigma^*)$.

- $a_1 \in P(L_1) \cup P(L_2)$; on suppose par exemple $a_1 \in P(L_1)$. Alors $a_1 \in \Sigma_1$.
- $a_1 a_2 \in F(L_1) \cup F(L_2)$. Mais $F(L_1) \subset \Sigma_1^2$ et $F(L_2) \subset \Sigma_2^2$. Les deux alphabets étant disjoints et $a_1 \in \Sigma_1$ on a $a_1 a_2 \in F(L_1)$ et $a_2 \in \Sigma_1$.
- De proche en proche on prouve que pour tout $i \in [\![1, p-1]\!]$, $a_i a_{i+1} \in F(L_1)$ et $a_{i+1} \in \Sigma_1$.

Langages locaux

L'intersection de deux langages locaux est un langage local.

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1 \cup L_2$ est local.

Notons Σ_1 et Σ_2 les alphabets sur lesquels sont définis L_1 et L_2, et $\Sigma = \Sigma_1 \cup \Sigma_2$.

On calcule $P(L_1 \cup L_2) = P(L_1) \cup P(L_2)$, $S(L_1 \cup L_2) = S(L_1) \cup S(L_2)$, $F(L_1 \cup L_2) = F(L_1) \cup F(L_2)$.

Soit $u = a_1 a_2 \cdots a_p \in (P(L_1 \cup L_2)\Sigma^* \cap \Sigma^* S(L_1 \cup L_2)) \setminus (\Sigma^* N(L_1 \cup L_2)\Sigma^*)$.

- $a_1 \in P(L_1) \cup P(L_2)$; on suppose par exemple $a_1 \in P(L_1)$. Alors $a_1 \in \Sigma_1$.
- $a_1 a_2 \in F(L_1) \cup F(L_2)$. Mais $F(L_1) \subset \Sigma_1^2$ et $F(L_2) \subset \Sigma_2^2$. Les deux alphabets étant disjoints et $a_1 \in \Sigma_1$ on a $a_1 a_2 \in F(L_1)$ et $a_2 \in \Sigma_1$.
- De proche en proche on prouve que pour tout $i \in [\![1, p-1]\!]$, $a_i a_{i+1} \in F(L_1)$ et $a_{i+1} \in \Sigma_1$.
- Enfin, $a_p \in \Sigma_1$ et $a_p \in S(L_1) \cup S(L_2)$ donc $a_p \in S(L_1)$.

Langages locaux

L'intersection de deux langages locaux est un langage local.

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1 \cup L_2$ est local.

Notons Σ_1 et Σ_2 les alphabets sur lesquels sont définis L_1 et L_2, et $\Sigma = \Sigma_1 \cup \Sigma_2$.

On calcule $P(L_1 \cup L_2) = P(L_1) \cup P(L_2)$, $S(L_1 \cup L_2) = S(L_1) \cup S(L_2)$, $F(L_1 \cup L_2) = F(L_1) \cup F(L_2)$.

Soit $u = a_1 a_2 \cdots a_p \in \big(P(L_1 \cup L_2)\Sigma^* \cap \Sigma^* S(L_1 \cup L_2)\big) \setminus \big(\Sigma^* N(L_1 \cup L_2)\Sigma^*\big)$.

- $a_1 \in P(L_1) \cup P(L_2)$; on suppose par exemple $a_1 \in P(L_1)$. Alors $a_1 \in \Sigma_1$.
- $a_1 a_2 \in F(L_1) \cup F(L_2)$. Mais $F(L_1) \subset \Sigma_1^2$ et $F(L_2) \subset \Sigma_2^2$. Les deux alphabets étant disjoints et $a_1 \in \Sigma_1$ on a $a_1 a_2 \in F(L_1)$ et $a_2 \in \Sigma_1$.
- De proche en proche on prouve que pour tout $i \in [\![1, p-1]\!]$, $a_i a_{i+1} \in F(L_1)$ et $a_{i+1} \in \Sigma_1$.
- Enfin, $a_p \in \Sigma_1$ et $a_p \in S(L_1) \cup S(L_2)$ donc $a_p \in S(L_1)$.

L_1 étant un langage local on en déduit que $u \in L_1 \subset L_1 \cup L_2$.

Langages locaux

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1 L_2$ est encore un langage local.

Langages locaux

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors L_1L_2 est encore un langage local.

Avec les mêmes notations on a cette fois :

$$P(L_1L_2) = \begin{cases} P(L_1) & \text{si } \varepsilon \notin L_1 \\ P(L_1) \cup P(L_2) & \text{sinon} \end{cases} \qquad S(L_1L_2) = \begin{cases} S(L_2) & \text{si } \varepsilon \notin L_2 \\ S(L_1) \cup S(L_2) & \text{sinon} \end{cases}$$

$$F(L_1L_2) = F(L_1) \cup F(L_2) \cup S(L_1)P(L_2).$$

Langages locaux

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors L_1L_2 est encore un langage local.

Avec les mêmes notations on a cette fois :

$$P(L_1L_2) = \begin{cases} P(L_1) & \text{si } \varepsilon \notin L_1 \\ P(L_1) \cup P(L_2) & \text{sinon} \end{cases} \qquad S(L_1L_2) = \begin{cases} S(L_2) & \text{si } \varepsilon \notin L_2 \\ S(L_1) \cup S(L_2) & \text{sinon} \end{cases}$$

$$F(L_1L_2) = F(L_1) \cup F(L_2) \cup S(L_1)P(L_2).$$

Soit $u = a_1a_2\cdots a_p \in (P(L_1L_2)\Sigma^* \cup \Sigma^*S(L_1L_2)) \setminus (\Sigma^*N(L_1L_2)\Sigma^*)$.

Langages locaux

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors L_1L_2 est encore un langage local.

Avec les mêmes notations on a cette fois :

$$P(L_1L_2) = \begin{cases} P(L_1) & \text{si } \varepsilon \notin L_1 \\ P(L_1) \cup P(L_2) & \text{sinon} \end{cases} \qquad S(L_1L_2) = \begin{cases} S(L_2) & \text{si } \varepsilon \notin L_2 \\ S(L_1) \cup S(L_2) & \text{sinon} \end{cases}$$

$$F(L_1L_2) = F(L_1) \cup F(L_2) \cup S(L_1)P(L_2).$$

Soit $u = a_1a_2\cdots a_p \in (P(L_1L_2)\Sigma^* \cup \Sigma^*S(L_1L_2)) \setminus (\Sigma^*N(L_1L_2)\Sigma^*)$.

- Si $a_1 \in \Sigma_2$ alors $\varepsilon \in L_1$ et $a_2 \in P(L_2)$. De proche en proche on prouve que $a_ia_{i+1} \in F(L_2)$ et $a_{i+1} \in \Sigma_2$. En particulier $a_p \in \Sigma_2$ donc $a_p \in S(L_2)$. Puisque L_2 est un langage local $u \in L_2$ et puisque $\varepsilon \in L_1$ on a aussi $u \in L_1L_2$.

Langages locaux

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors L_1L_2 est encore un langage local.

Avec les mêmes notations on a cette fois :

$$P(L_1L_2) = \begin{cases} P(L_1) & \text{si } \varepsilon \notin L_1 \\ P(L_1) \cup P(L_2) & \text{sinon} \end{cases} \qquad S(L_1L_2) = \begin{cases} S(L_2) & \text{si } \varepsilon \notin L_2 \\ S(L_1) \cup S(L_2) & \text{sinon} \end{cases}$$

$$F(L_1L_2) = F(L_1) \cup F(L_2) \cup S(L_1)P(L_2).$$

Soit $u = a_1a_2\cdots a_p \in \left(P(L_1L_2)\Sigma^* \cup \Sigma^*S(L_1L_2)\right) \setminus \left(\Sigma^*N(L_1L_2)\Sigma^*\right)$.

- Si $a_1 \in \Sigma_1$, notons $a_1\cdots a_k$ le plus long préfixe de u qui soit dans Σ_1^*. Alors $a_1 \in P(L_1)$ et $\forall i \in [\![1, k-1]\!]$, $a_ia_{i+1} \in F(L_1)$. Ensuite, de deux choses l'une :

Langages locaux

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors L_1L_2 est encore un langage local.

Avec les mêmes notations on a cette fois :

$$P(L_1L_2) = \begin{cases} P(L_1) & \text{si } \varepsilon \notin L_1 \\ P(L_1) \cup P(L_2) & \text{sinon} \end{cases} \qquad S(L_1L_2) = \begin{cases} S(L_2) & \text{si } \varepsilon \notin L_2 \\ S(L_1) \cup S(L_2) & \text{sinon} \end{cases}$$

$$F(L_1L_2) = F(L_1) \cup F(L_2) \cup S(L_1)P(L_2).$$

Soit $u = a_1a_2 \cdots a_p \in (P(L_1L_2)\Sigma^* \cup \Sigma^* S(L_1L_2)) \setminus (\Sigma^* N(L_1L_2)\Sigma^*)$.

- Si $a_1 \in \Sigma_1$, notons $a_1 \cdots a_k$ le plus long préfixe de u qui soit dans Σ_1^*. Alors $a_1 \in P(L_1)$ et $\forall i \in [\![1, k-1]\!]$, $a_ia_{i+1} \in F(L_1)$. Ensuite, de deux choses l'une :
  - si $k = p$ alors $a_p \in S(L_1)$ et $\varepsilon \in L_2$. Puisque L_1 est local on a $u \in L_1$ et puisque $\varepsilon \in L_2$ on a aussi $u \in L_1L_2$.

Langages locaux

Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors L_1L_2 est encore un langage local.

Avec les mêmes notations on a cette fois :

$$P(L_1L_2) = \begin{cases} P(L_1) & \text{si } \varepsilon \notin L_1 \\ P(L_1) \cup P(L_2) & \text{sinon} \end{cases} \qquad S(L_1L_2) = \begin{cases} S(L_2) & \text{si } \varepsilon \notin L_2 \\ S(L_1) \cup S(L_2) & \text{sinon} \end{cases}$$

$$F(L_1L_2) = F(L_1) \cup F(L_2) \cup S(L_1)P(L_2).$$

Soit $u = a_1a_2\cdots a_p \in (P(L_1L_2)\Sigma^* \cup \Sigma^* S(L_1L_2)) \setminus (\Sigma^* N(L_1L_2)\Sigma^*)$.

- Si $a_1 \in \Sigma_1$, notons $a_1\cdots a_k$ le plus long préfixe de u qui soit dans Σ_1^*. Alors $a_1 \in P(L_1)$ et $\forall i \in [\![1, k-1]\!]$, $a_ia_{i+1} \in F(L_1)$. Ensuite, de deux choses l'une :
  - si $k = p$ alors $a_p \in S(L_1)$ et $\varepsilon \in L_2$. Puisque L_1 est local on a $u \in L_1$ et puisque $\varepsilon \in L_2$ on a aussi $u \in L_1L_2$.
  - si $k < p$ alors $a_ka_{k+1} \in S(L_1)P(L_2)$ donc $a_{k+1} \in \Sigma_2$ et de proche en proche on prouve que $\forall j \in [\![k+1, p-1]\!]$, $a_ja_{j+1} \in F(L_2)$, $a_{j+1} \in \Sigma_2$ et $a_p \in S(L_2)$. Puisque L_1 et L_2 sont locaux on en déduit que $a_1\cdots a_k \in L_1$ et $a_{k+1}\cdots a_p \in L_2$ et donc que $u \in L_1L_2$.

Langages locaux

La fermeture de KLEENE L^* d'un langage local L est aussi local.

Langages locaux

La fermeture de KLEENE L^* d'un langage local L est aussi local.

Soit $u \in L^* \setminus \{\varepsilon\} = L^+$. Alors $u = u_1 u_2 \cdots u_p$ avec $u_i \in L$.
La première lettre de u est aussi celle de u_1 donc est dans $P(L^*) = P(L)$;
La dernière lettre de u est aussi celle de u_p donc est dans $S(L^*) = S(L)$.
Un facteur de longueur 2 de u est :

- un facteur d'un des u_i auquel cas il est dans $F(L)$;
- ou de la forme xy où x est la dernière lettre de u_i et y la première lettre de u_{i+1}, auquel cas il est dans $S(L)P(L)$;

Donc $F(L^*) = F(L) \cup S(L)P(L)$.

Langages locaux

La fermeture de Kleene L^* d'un langage local L est aussi local.

Soit $u \in L^* \setminus \{\varepsilon\} = L^+$. Alors $u = u_1 u_2 \cdots u_p$ avec $u_i \in L$.
La première lettre de u est aussi celle de u_1 donc est dans $P(L^*) = P(L)$;
La dernière lettre de u est aussi celle de u_p donc est dans $S(L^*) = S(L)$.
Un facteur de longueur 2 de u est :

- un facteur d'un des u_i auquel cas il est dans $F(L)$;
- ou de la forme xy où x est la dernière lettre de u_i et y la première lettre de u_{i+1}, auquel cas il est dans $S(L)P(L)$;

Donc $F(L^*) = F(L) \cup S(L)P(L)$.

Soit $u \in (P\Sigma^* \cap \Sigma^* S) \setminus (\Sigma^* N \Sigma^*)$.
Parmi les facteurs de longueur 2 de u, considérons uniquement ceux qui appartiennent à $S(L)P(L)$; ils induisent une factorisation de u en mots : $u = u_1 u_2 \cdots u_p$ avec $u_i \in (P(L)\Sigma^* \cap \Sigma^* S(L)) \setminus (\Sigma^* N(L) \Sigma^*)$.

Par hypothèse L est local, donc chacun des u_i est un mot de L et $u \in L^*$.

Expressions rationnelles linéaires

Une expression rationnelle e est linéaire lorsque tout caractère de Σ apparaît au plus une fois dans e.

Toute expression rationnelle linéaire dénote un langage local.

Expressions rationnelles linéaires

Une expression rationnelle e est linéaire lorsque tout caractère de Σ apparaît au plus une fois dans e.

Toute expression rationnelle linéaire dénote un langage local.

On raisonne par induction structurelle sur e.

Expressions rationnelles linéaires

Une expression rationnelle e est linéaire lorsque tout caractère de Σ apparaît au plus une fois dans e.

Toute expression rationnelle linéaire dénote un langage local.

On raisonne par induction structurelle sur e.

- $\emptyset$ et ε dénotent respectivement les langages $\emptyset$ et $\{\varepsilon\}$ qui sont des langages locaux.

Expressions rationnelles linéaires

Une expression rationnelle e est linéaire lorsque tout caractère de Σ apparaît au plus une fois dans e.

Toute expression rationnelle linéaire dénote un langage local.

On raisonne par induction structurelle sur e.

- $\emptyset$ et ε dénotent respectivement les langages $\emptyset$ et $\{\varepsilon\}$ qui sont des langages locaux.
- si $a \in \Sigma$, a dénote le langage local $\{a\}$ avec $P = \{a\}$, $S = \{a\}$ et $N = \Sigma^2$.

Expressions rationnelles linéaires

Une expression rationnelle e est linéaire lorsque tout caractère de Σ apparaît au plus une fois dans e.

Toute expression rationnelle linéaire dénote un langage local.

On raisonne par induction structurelle sur e.

- $\emptyset$ et ε dénotent respectivement les langages $\emptyset$ et $\{\varepsilon\}$ qui sont des langages locaux.
- si $a \in \Sigma$, a dénote le langage local $\{a\}$ avec $P = \{a\}$, $S = \{a\}$ et $N = \Sigma^2$.
- si $e = e_1 + e_2$ est une expression rationnelle linéaire, il en est de même de e_1 et e_2, et par hypothèse d'induction ces deux expressions dénotent des langages locaux. Mais e_1 et e_2 peuvent être définis sur des alphabets disjoints, donc e dénote aussi un langage local.

Expressions rationnelles linéaires

Une expression rationnelle e est linéaire lorsque tout caractère de Σ apparaît au plus une fois dans e.

Toute expression rationnelle linéaire dénote un langage local.

On raisonne par induction structurelle sur e.

- $\emptyset$ et ε dénotent respectivement les langages $\emptyset$ et $\{\varepsilon\}$ qui sont des langages locaux.
- si $a \in \Sigma$, a dénote le langage local $\{a\}$ avec $P = \{a\}$, $S = \{a\}$ et $N = \Sigma^2$.
- si $e = e_1 + e_2$ est une expression rationnelle linéaire, il en est de même de e_1 et e_2, et par hypothèse d'induction ces deux expressions dénotent des langages locaux. Mais e_1 et e_2 peuvent être définis sur des alphabets disjoints, donc e dénote aussi un langage local.
- si $e = e_1 e_2$ ou $e = e_1^*$ le raisonnement est identique.

Algorithmes de calcul de P, S et F

dans le cas d'une expression rationnelle linéaire

La fonction suivante détermine si ε appartient au langage :

```
let rec motvide = function
  | Epsilon         -> true
  | Const _         -> false
  | Sum (e1, e2)    -> motvide e1 || motvide e2
  | Concat (e1, e2) -> motvide e1 && motvide e2
  | Kleene _        -> true ;;
```

Algorithmes de calcul de P, S et F

dans le cas d'une expression rationnelle linéaire

La fonction suivante détermine si ε appartient au langage :

```
let rec motvide = function
  | Epsilon         -> true
  | Const _         -> false
  | Sum (e1, e2)    -> motvide e1 || motvide e2
  | Concat (e1, e2) -> motvide e1 && motvide e2
  | Kleene _        -> true ;;
```

Calcul de l'ensemble P :

```
let rec prefixe = function
  | Epsilon                         -> []
  | Const a                         -> [a]
  | Sum (e1, e2)                    -> union (prefixe e1) (prefixe e2)
  | Concat (e1, e2) when motvide e1 -> union (prefixe e1) (prefixe e2)
  | Concat (e1, e2)                 -> prefixe e1
  | Kleene e                        -> prefixe e ;;
```

Algorithmes de calcul de P, S et F

dans le cas d'une expression rationnelle linéaire

La fonction suivante détermine si ε appartient au langage :

```
let rec motvide = function
  | Epsilon          -> true
  | Const _          -> false
  | Sum (e1, e2)     -> motvide e1 || motvide e2
  | Concat (e1, e2) -> motvide e1 && motvide e2
  | Kleene _         -> true ;;
```

Calcul de l'ensemble S :

```
let rec suffixe = function
  | Epsilon                          -> []
  | Const a                          -> [a]
  | Sum (e1, e2)                     -> union (suffixe e1) (suffixe e2)
  | Concat (e1, e2) when motvide e2 -> union (suffixe e1) (suffixe e2)
  | Concat (e1, e2)                  -> suffixe e2
  | Kleene e                         -> suffixe e ;;
```

Algorithmes de calcul de P, S et F

dans le cas d'une expression rationnelle linéaire

La fonction suivante détermine si ε appartient au langage :

```
let rec motvide = function
  | Epsilon          -> true
  | Const _          -> false
  | Sum (e1, e2)     -> motvide e1 || motvide e2
  | Concat (e1, e2) -> motvide e1 && motvide e2
  | Kleene _         -> true ;;
```

Calcul de l'ensemble F :

```
let rec produit l1 l2 = match (l1, l2) with
  | [], _      -> []
  | _, []      -> []
  | t1::q1, _ -> union (map (function x -> t1 ^ x) l2) (produit q1 l2) ;;

let rec facteur = function
  | Epsilon          -> []
  | Const a          -> []
  | Sum (e1, e2)     -> union (facteur e1) (facteur e2)
  | Concat (e1, e2) -> let l = union (facteur e1) (facteur e2)
                        in union l (produit (suffixe e1) (prefixe e2))
  | Kleene e       -> union (facteur e) (produit (suffixe e) (prefixe e)) ;;
```

Linéarisation d'une expression rationnelle

Soit e une expression rationnelle quelconque contenant n lettres *non nécessairement distinctes* de Σ.

- On les ordonne par ordre croissant d'apparition dans e ;
- on remplace dans e chaque lettre par le caractère c_k où k désigne son rang d'apparition.

On obtient une nouvelle expression rationnelle linéaire sur $\Sigma' = \{c_1, \ldots, c_n\}$.

Linéarisation d'une expression rationnelle

Soit e une expression rationnelle quelconque contenant n lettres *non nécessairement distinctes* de Σ.

- On les ordonne par ordre croissant d'apparition dans e ;
- on remplace dans e chaque lettre par le caractère c_k où k désigne son rang d'apparition.

On obtient une nouvelle expression rationnelle linéaire sur $\Sigma' = \{c_1, \dots, c_n\}$.

Exemple : $e = (a+b)(a^* + ba^* + b^*)^* \longrightarrow e' = (c_1 + c_2)(c_3^* + c_4 c_5^* + c_6^*)^*$.

Linéarisation d'une expression rationnelle

Soit e une expression rationnelle quelconque contenant n lettres *non nécessairement distinctes* de Σ.

- On les ordonne par ordre croissant d'apparition dans e ;
- on remplace dans e chaque lettre par le caractère c_k où k désigne son rang d'apparition.

On obtient une nouvelle expression rationnelle linéaire sur $\Sigma' = \{c_1, \dots, c_n\}$.

Exemple : $e = (a+b)(a^* + ba^* + b^*)^* \longrightarrow e' = (c_1 + c_2)(c_3^* + c_4 c_5^* + c_6^*)^*$.

Pour retrouver l'expression initiale à partir de l'expression linéarisée on donne la fonction de marquage $\mu : [\![1, n]\!] \to \Sigma$ qui précise par quel caractère de Σ doit être remplacé le caractère $c_{\mu(k)}$.

Linéarisation d'une expression rationnelle

Soit e une expression rationnelle quelconque contenant n lettres *non nécessairement distinctes* de Σ.

- On les ordonne par ordre croissant d'apparition dans e ;
- on remplace dans e chaque lettre par le caractère c_k où k désigne son rang d'apparition.

On obtient une nouvelle expression rationnelle linéaire sur $\Sigma' = \{c_1, \dots, c_n\}$.

Exemple : $e = (a+b)(a^* + ba^* + b^*)^* \longrightarrow e' = (c_1 + c_2)(c_3^* + c_4 c_5^* + c_6^*)^*$.

Pour retrouver l'expression initiale à partir de l'expression linéarisée on donne la fonction de marquage $\mu : [\![1, n]\!] \to \Sigma$ qui précise par quel caractère de Σ doit être remplacé le caractère $c_{\mu(k)}$.

Exemple : $\mu(1) = \mu(3) = \mu(5) = a$ et $\mu(2) = \mu(4) = \mu(6) = b$.