Bases de données et web – CM4

L3 Informatique & L3 MIASHS

Introduction

XPath est un langage de requête permettant de sélectionner des nœuds d'un document XML. XPath est notamment utilisé dans d'autres langages, comme XSLT ou XQuery. Il est également utilisé dans XML Schema.

Ce langage permet la description de **chemins** dans l'arbre XML associé au document et permet donc de *trouver* les nœuds contenus dans ces chemins.

XPath peut aussi être utilisé pour effectuer certaines opérations sur les données XML, par exemple comme des sommes, arrondis, et autres opérations élémentaires . . . sur les nombres, chaînes de caractères et valeurs booléennes.

Exemple

- `2 * 3` est une expression XPath, composée uniquement d'une expression arithmétique.
- `//*[@msg="Hello World"]` est une expression XPath, faisant référence à un chemin dans un arbre XML.

Modèle de données d'XPath

Une expression XPath agit sur un arbre XML, qui contient les types de nœuds suivant :

- ***Document*** : la racine du document XML.
- ***Élément*** : des nœuds éléments.
- ***Attribut*** : des nœuds attributs, représenté comme des enfants d'éléments ou directement intégré dans les nœuds.
- ***Texte*** : des nœuds de données brutes, ce sont les feuilles de l'arbre XML.

XPath est capable de gérer des expression faisant référence à chacun de ces types de nœuds. On peut par exemple souhaiter ne chercher que parmi les nœuds de type attribut ou texte, ou encore chercher une balise précise (on cherche donc un élément).

Modèle de données d'XPath

- Le nœud racine de l'arbre XML est le nœud document.
- L'élément racine est l'unique élément enfant du nœud racine. C'est la balise racine de notre document XML.
- Un nœud possède un nom, ou une valeur, ou les deux :
  - un nœud élément possède un nom mais pas de valeur;
  - un nœud texte possède une valeur (chaîne de caractères) mais pas de nom;
  - un attribut possède un nom et une valeur.
- Les attributs sont spéciaux. Les attributs ne sont pas considérés comme des nœuds standards, il faut y faire référence spécifiquement.

Les contextes XPath

Une étape d'une requête XPath est évaluée dans un contexte spécifique, dénoté par $(< N_1, N_2, \ldots, N_n >, N_c)$ où chaque N_i est un nœud de l'arbre et N_c désigne le nœud actuel.

On peut connaître la longueur du contexte, n, du contexte en utilisant la fonction `last()`.

La position actuel du contexte, $c \in [1, n]$, peut être connue grâce à la fonction `position()`.

Chaque étape est évaluée par rapport aux nœuds du contexte définis par les étapes précédentes. À son tour, une étape sélectionne des nœuds et les ajoute au contexte pour l'étape suivante.

Les étapes XPath

Le composant basique d'une expression XPath est **l'étape**. Une étape est de la forme : `axe::test-nœud`$[P_1][P_2]\ldots[P_n]$ où :

- ***axe*** : le nom de l'axe indiquant la direction de l'étape dans l'arbre XML. L'axe par défaut est `child`, c'est à dire qu'on *explore* les nœuds enfants, on *descend* donc dans l'arbre XML.
- ***test-nœeud*** : le test indiquant quel type de nœuds doit être sélectionner dans la requête.
- $[\mathbf{P_1}][\mathbf{P_2}]\ldots[\mathbf{P_n}]$: le prédicat, qui peut être n'importe quelle expression XPath retournant un booléen, permettant d'affiner la recherche. C'est un prédicat qui permet de filtrer parmi les nœuds trouvés jusqu'à présent. Ce prédicat est optionnel.

Une étape est évaluée par rapport à un contexte et retourne une liste de nœuds.

Exemple

`descendant::C[@att1='1']` est une étape qui *cherche* tous les nœuds éléments nommé `C`, descendant du nœuds contexte actuel, possédant un attribut nommé `att1` ayant pour valeur 1.

Expression XPath

Une expression XPath est de la forme : ***[/]étape$_1$/étape$_2$/.../étape$_n$***. Le premier / est optionnel, si il est présent cela signifie que le chemin est absolue, sinon il est relatif.

Exemple

- `/A/B` est une expression absolue absolue recherchant les nœuds éléments nommés `B`, étant enfants de la racine nommée `A`.
- `./B/descendant::text()` est une expression relative recherchant les nœuds textes descendant de l'élément `B`, lui même étant enfant du nœud contexte actuel.
- `/A/B/@att1[.>2]` recherche tous les nœuds attributs nommés `@att1` d'une valeur strictement supérieure à 2, étant associé à un nœud `B` enfant de la racine `A`.

Évaluation de l'expression

Chaque étape e_i est interprétée par rapport au contexte actuel. Le résultat qu'elle retourne est une liste de nœuds.

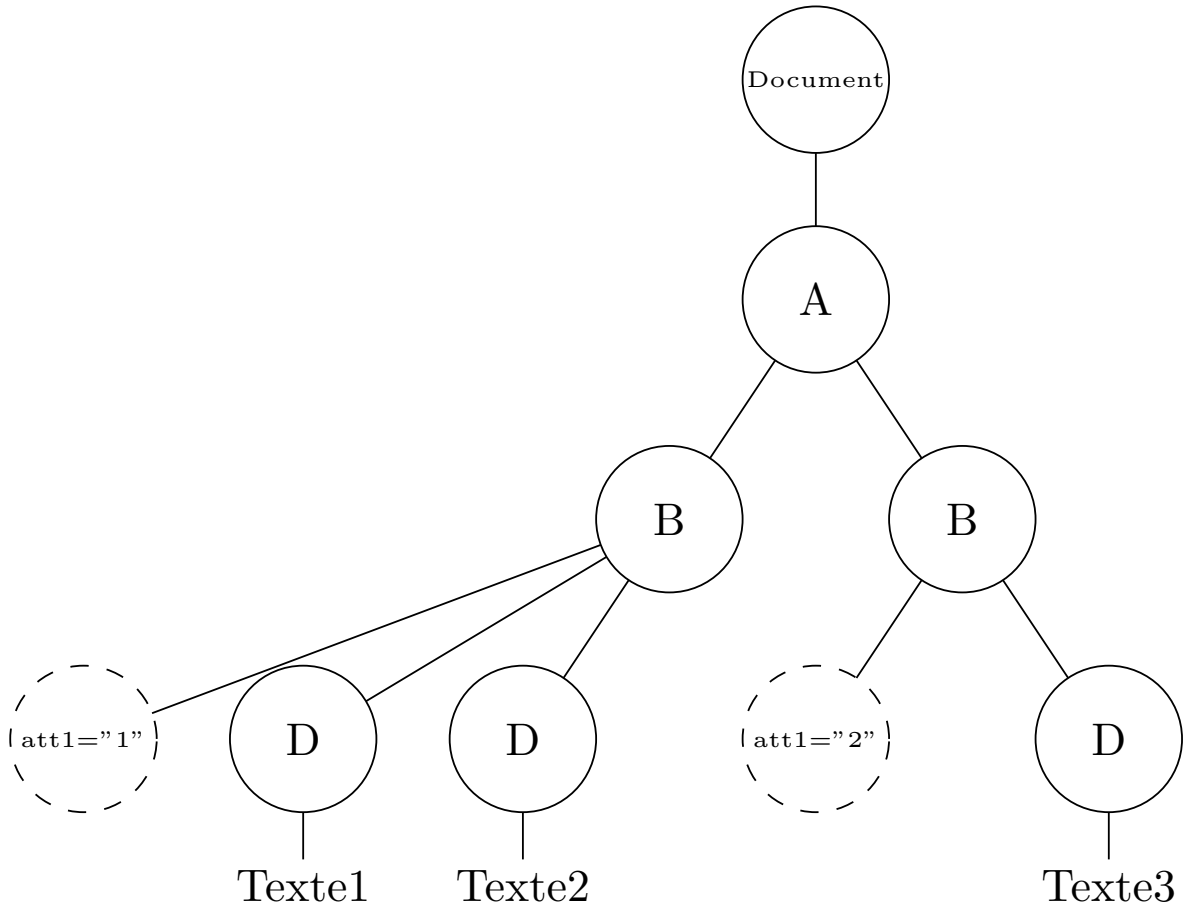
Une étape e_i est évaluée par rapport au contexte de l'étape e_{i-1}. Plus précisément :

- Pour $i = 1$ (la première étape), si la requête est absolue le contexte est simplement la racine du document XML. Sinon, le contexte est déterminée par l'environnement.
- Pour $i > 1$, avec $\mathcal{N} =< N_1, N_2, \ldots, N_n >$ le résultat de l'étape e_{i-1}, alors l'étape e_i est évaluée successivement par rapport au contexte $(\mathcal{N}, N_j)$ pour tout $j \in [1, n]$.

Le résultat de l'expression XPath est l'ensemble de nœuds obtenu après avoir la toute dernière étape.

Évaluation de /A/B/@att1

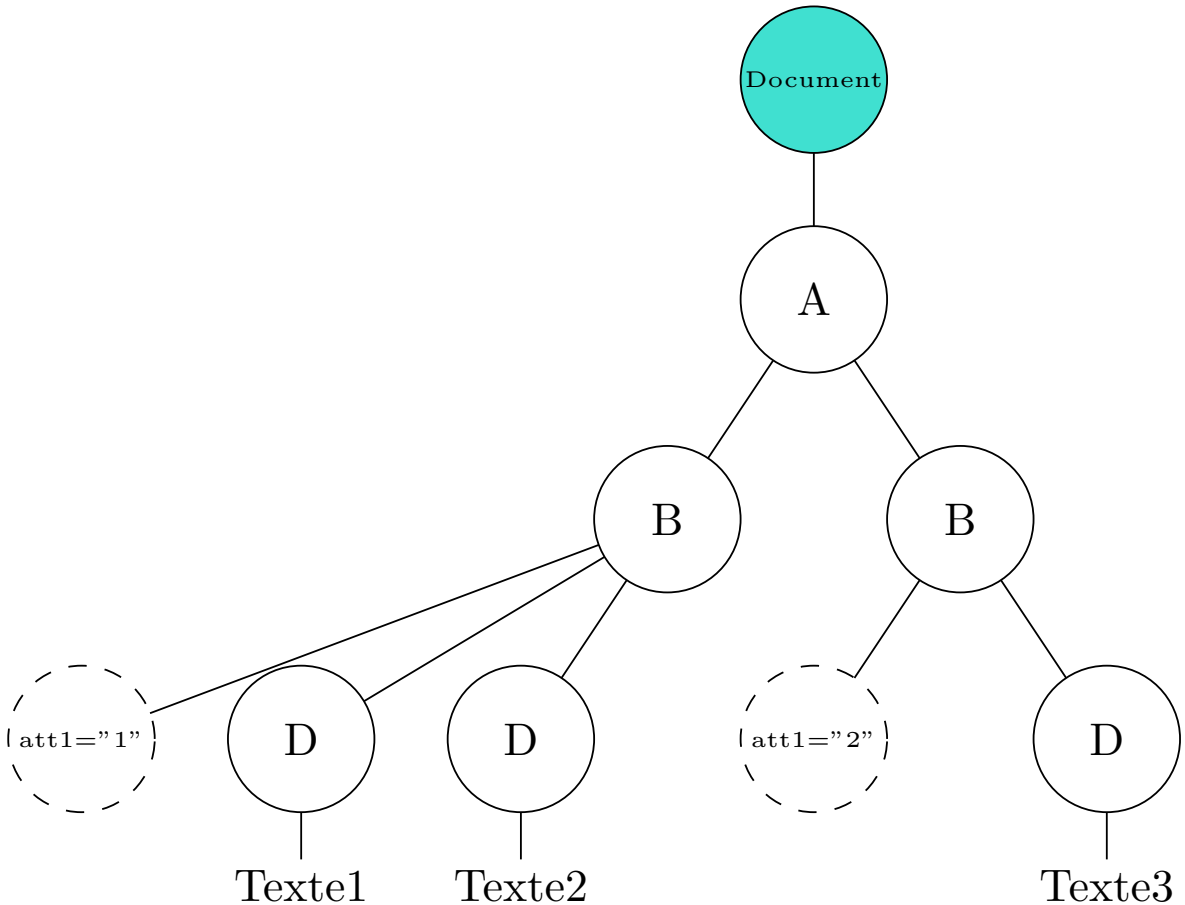
/A/B/@att1



Évaluation de /A/B/@att1

/A/B/@att1

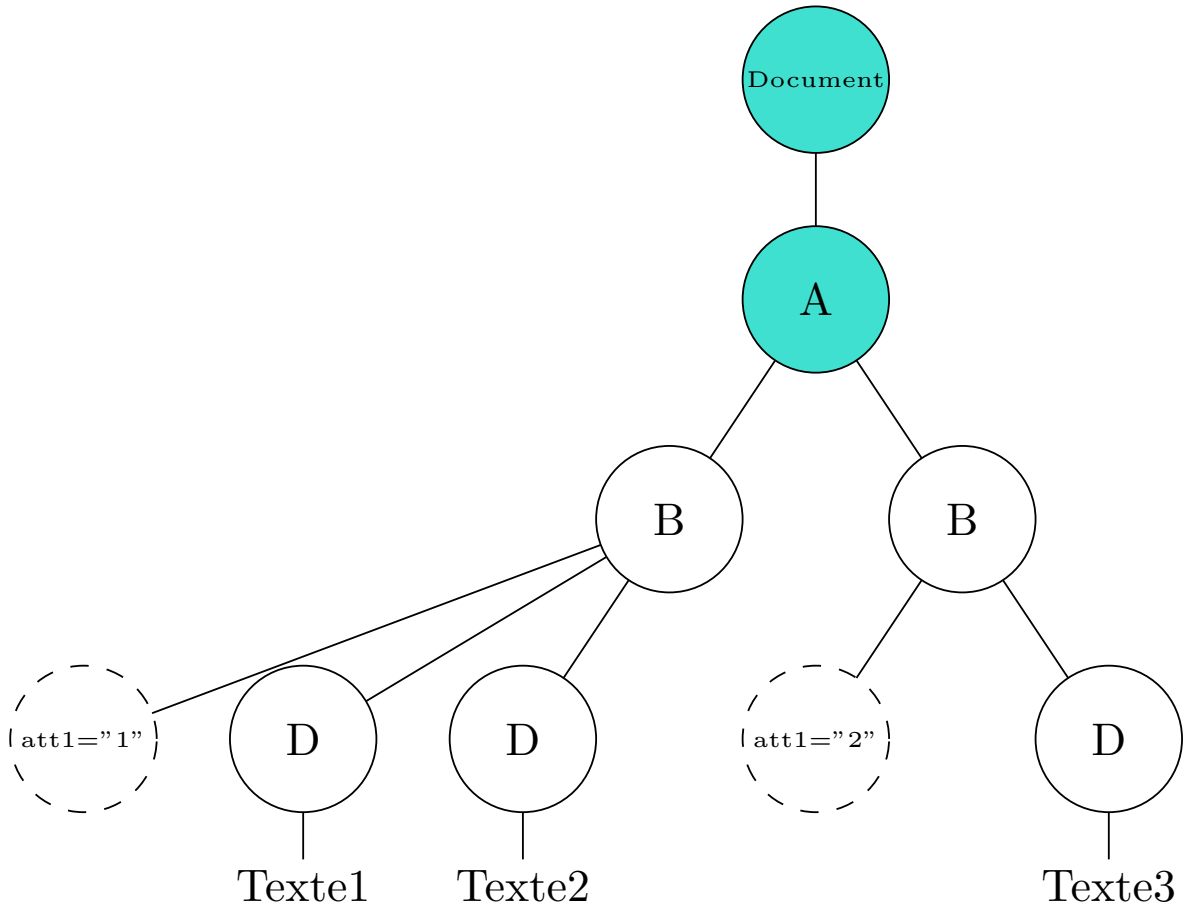
Le / signifie que c'est une requête absolue. Nous sommes donc sur la racine.



Évaluation de /A/B/@att1

/A/B/@att1

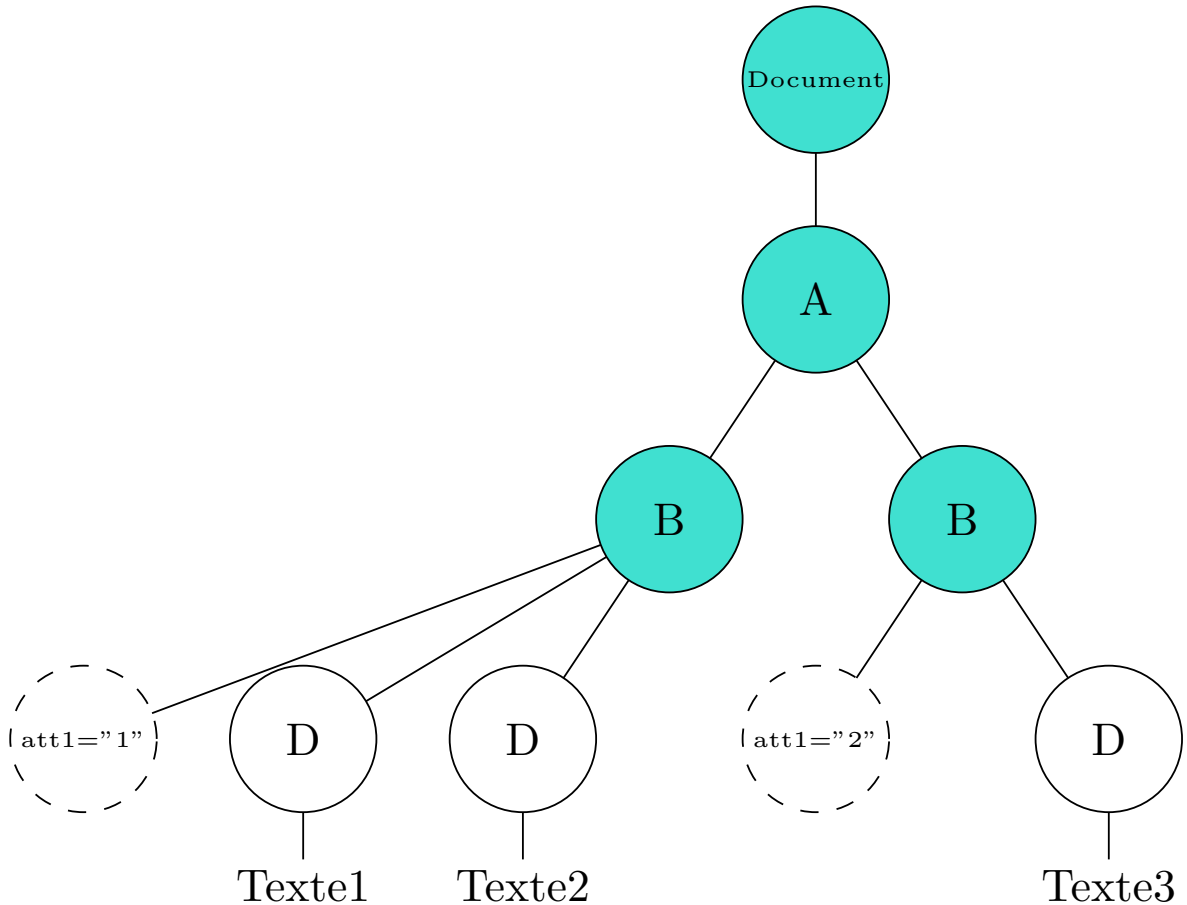
Après la racine du document, on récupère la racine XML.



Évaluation de /A/B/@att1

/A/B/@att1

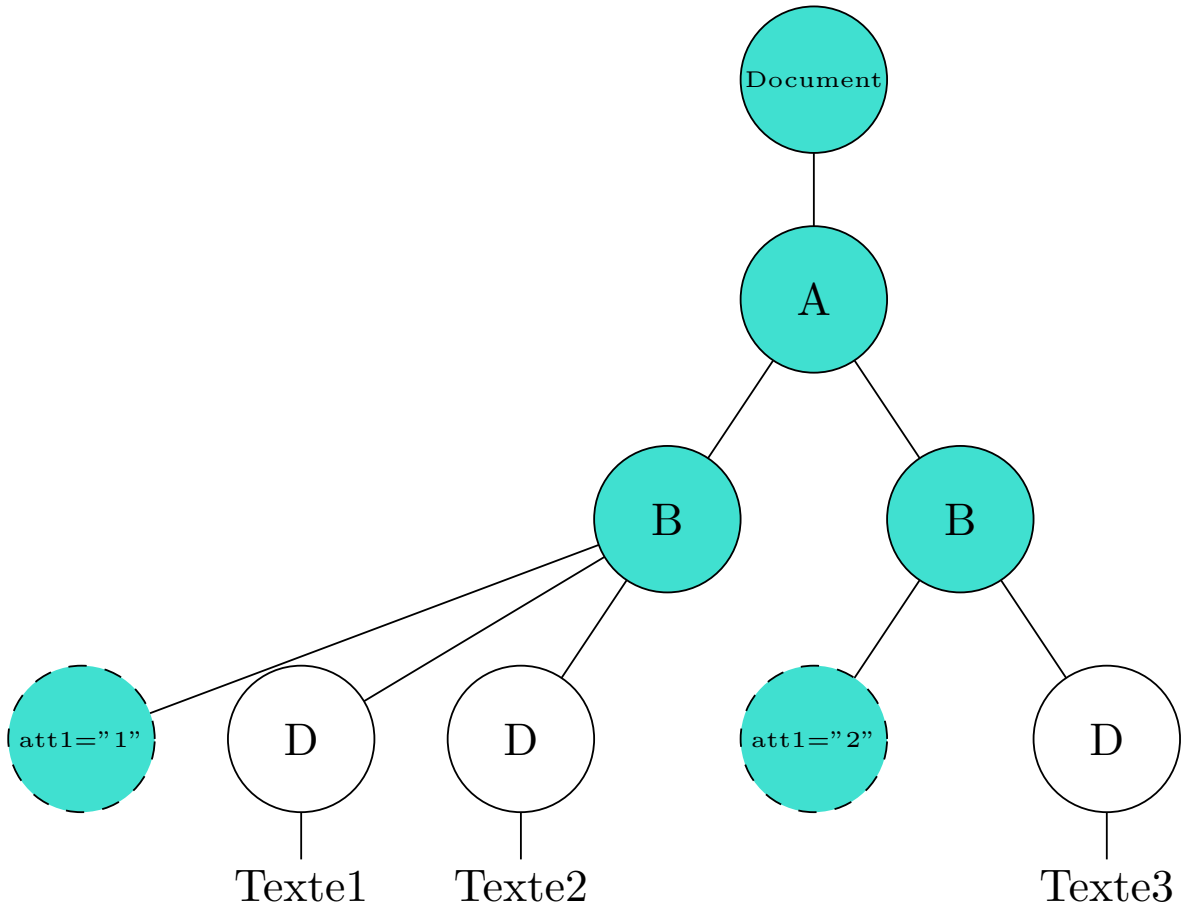
On cherche les nœuds B.
Remarque : on récupère tous les nœuds d'un coup et on les ajoute au contexte.



Évaluation de /A/B/@att1

/A/B/@att1

Enfin, on s'intéresse uniquement à l'attribut att1. Comme dans notre contexte il y avait deux nœuds B, on récupère les attributs att1 de chacun.



Les axes

Un axe permet de récupérer un ensemble de nœuds déterminés grâce au nœud du contexte en cours et à leurs positions dans l'arbre XML.

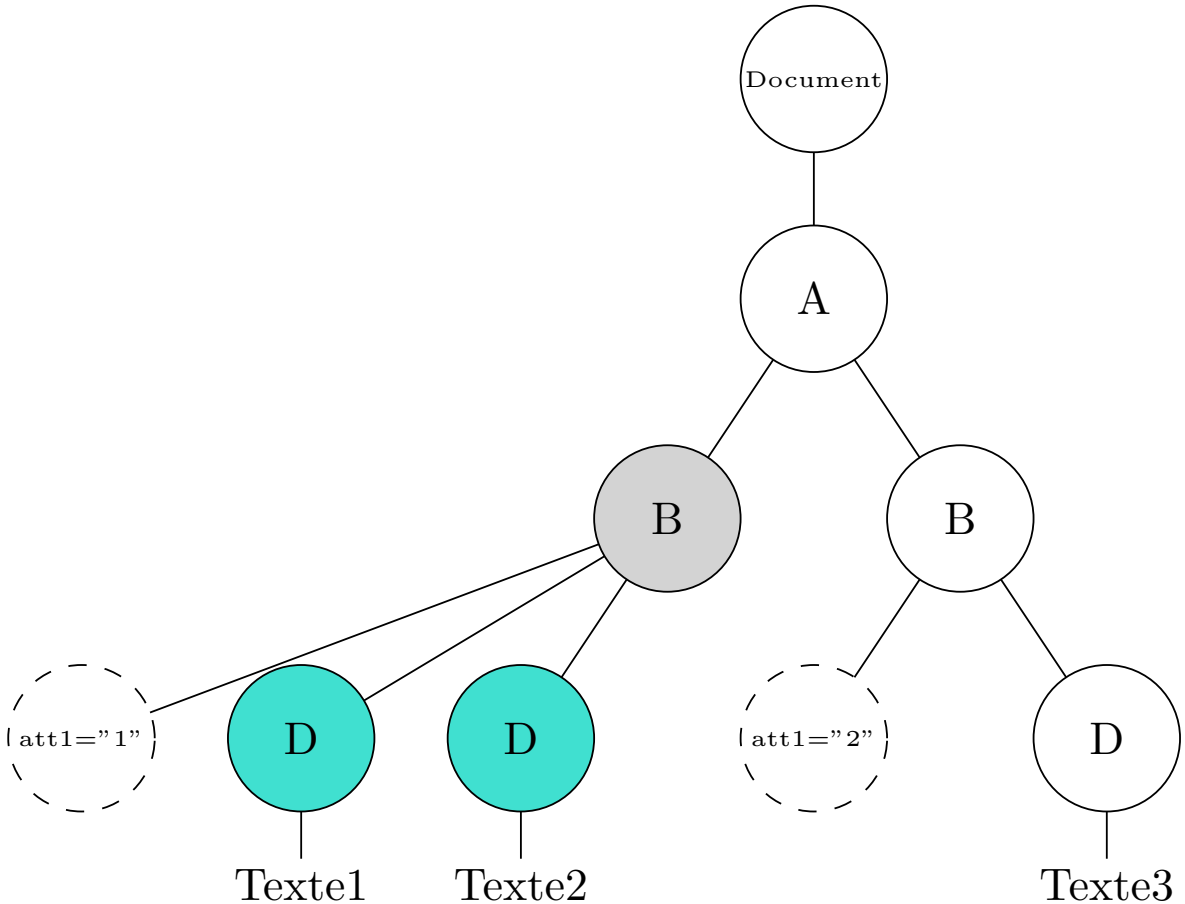
- `child` : l'axe défaut.
- `parent` : le nœud parent.
- `attribute` : le nœud attribut.
- `descendant` : les descendants, à l'exception du nœud lui-même.
- `descendant-or-self` : les descendants, en incluant le nœud.
- `ancestor` : les ancêtres, à l'exception du nœud lui-même.
- `ancestor-or-self` : les ancêtres, en incluant le nœud.
- `following` : les nœuds suivants dans l'ordre du document.
- `following-sibling` : les nœuds frères/sœurs suivants dans l'ordre du document.
- `preceding` : les nœuds précédents dans l'ordre du document.
- `preceding-sibling` : les nœuds frères/sœurs précédents dans l'ordre du document.
- `self` : le nœud du contexte actuel.

Exemples d'interprétation d'axe

`child::D`

L'axe `child` signifie qu'on récupère tous les éléments ou nœuds textes enfants du nœud contexte.

Remarque : un attribut possède un nœud parent (son élément associé), mais l'attribut n'est pas considéré comme un enfant de son parent.

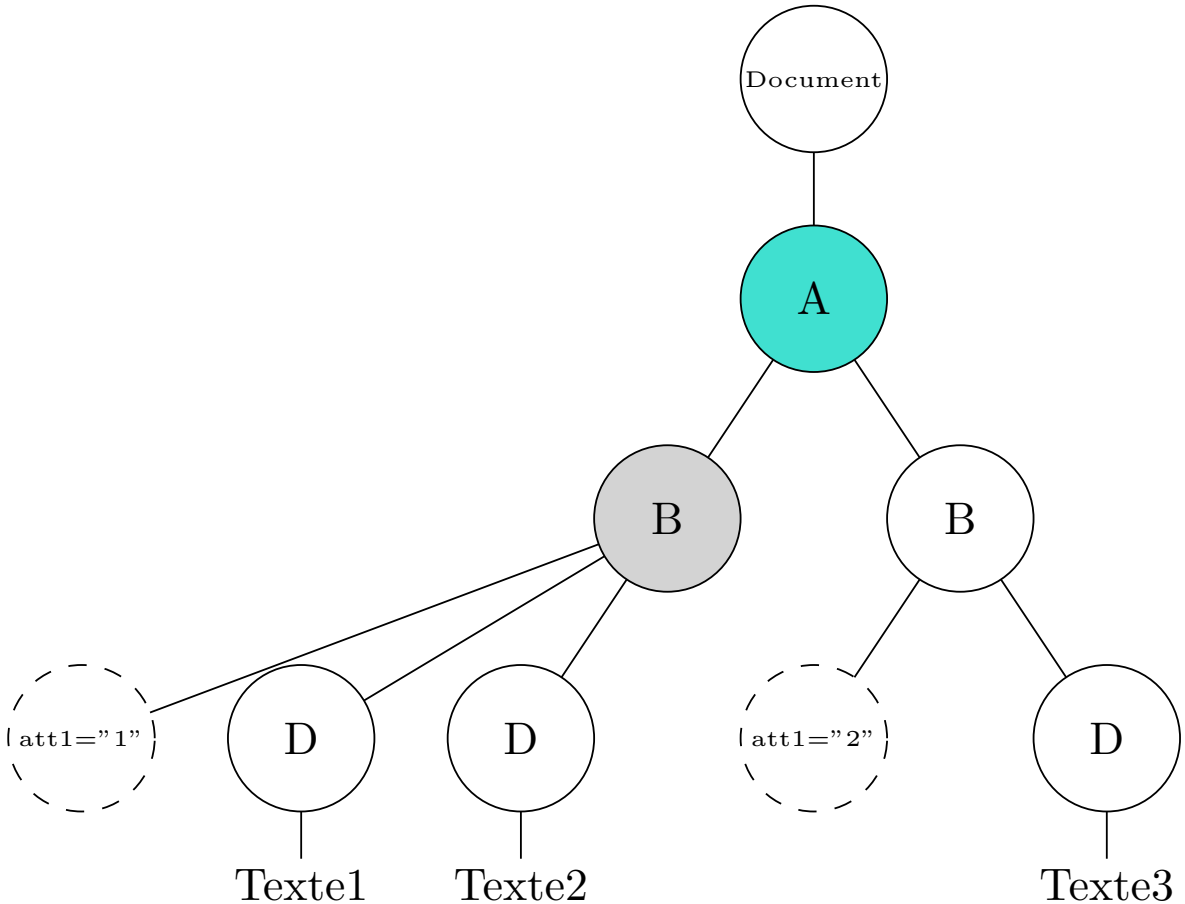


Exemples d'interprétation d'axe

`parent::node()`

L'axe `parent` signifie que l'on récupère le parent du nœud contexte. Le test du nœud est soit un nom d'élément, soit `*` qui récupère tous les noms d'élément, soit `node()` qui récupère tous les types de nœud.

`..` est une abréviation pour `parent::node()`.

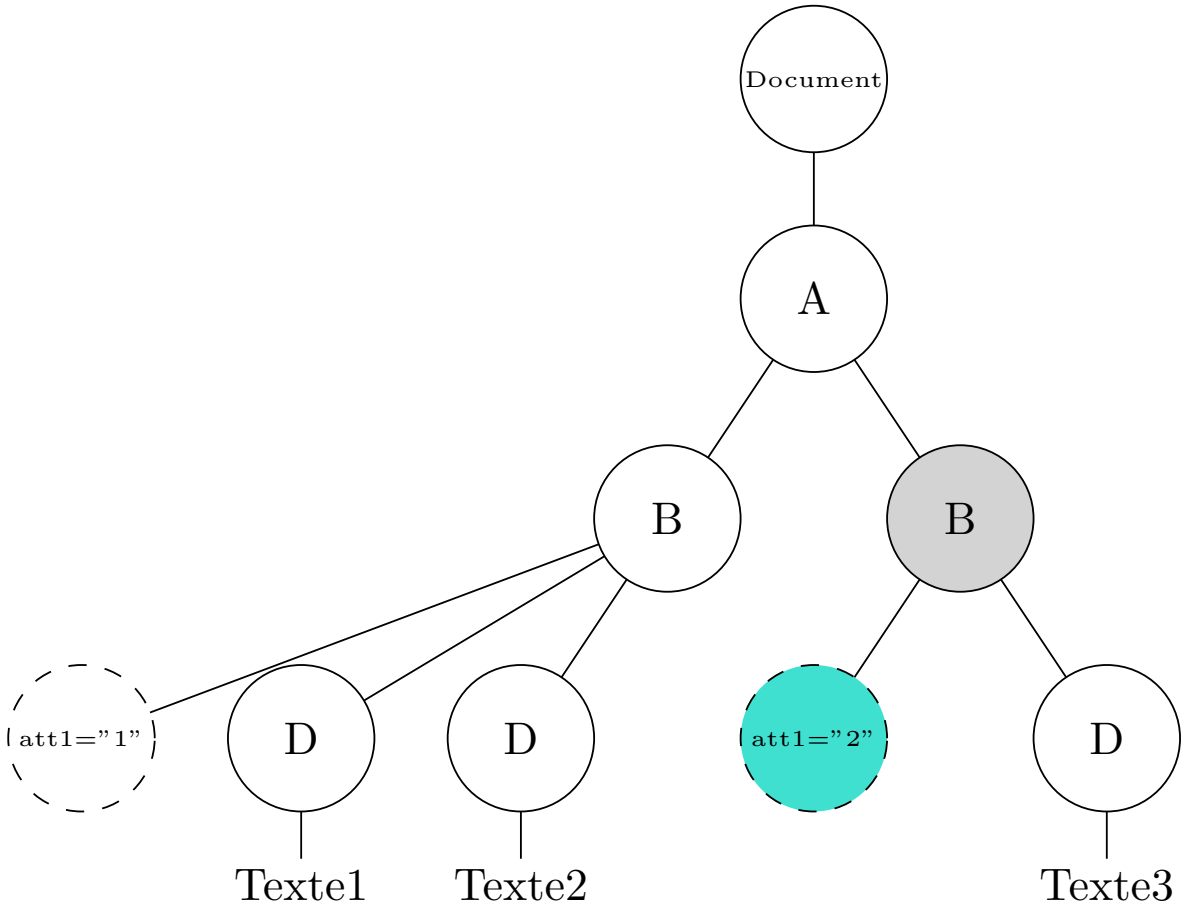


Exemples d'interprétation d'axe

`attribute::*`

L'axe `attribute` signifie que l'on récupère les attributs du nœud contexte. Le test du nœud est alors soit un nom d'attribut, soit `*` qui récupère tous les noms d'attribut.

`@*` est l'abréviation pour `attribute::*`.

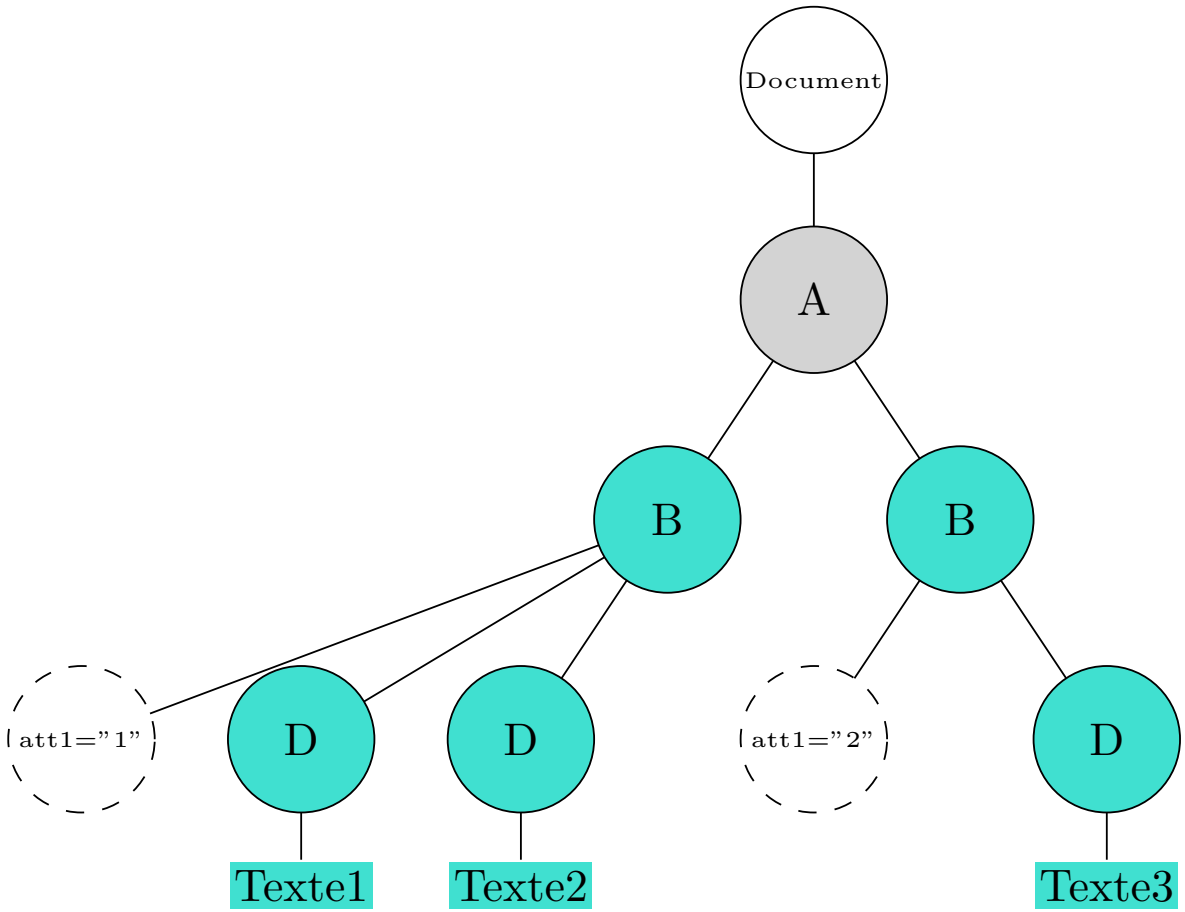


Exemples d'interprétation d'axe

`descendant::node()`

L'axe `descendant` récupère tous les nœuds descendants du nœud actuel, ***à l'exception*** des attributs. Le test de nœud est soit un nom, soit `*`, soit `text()`, soit `node()`.

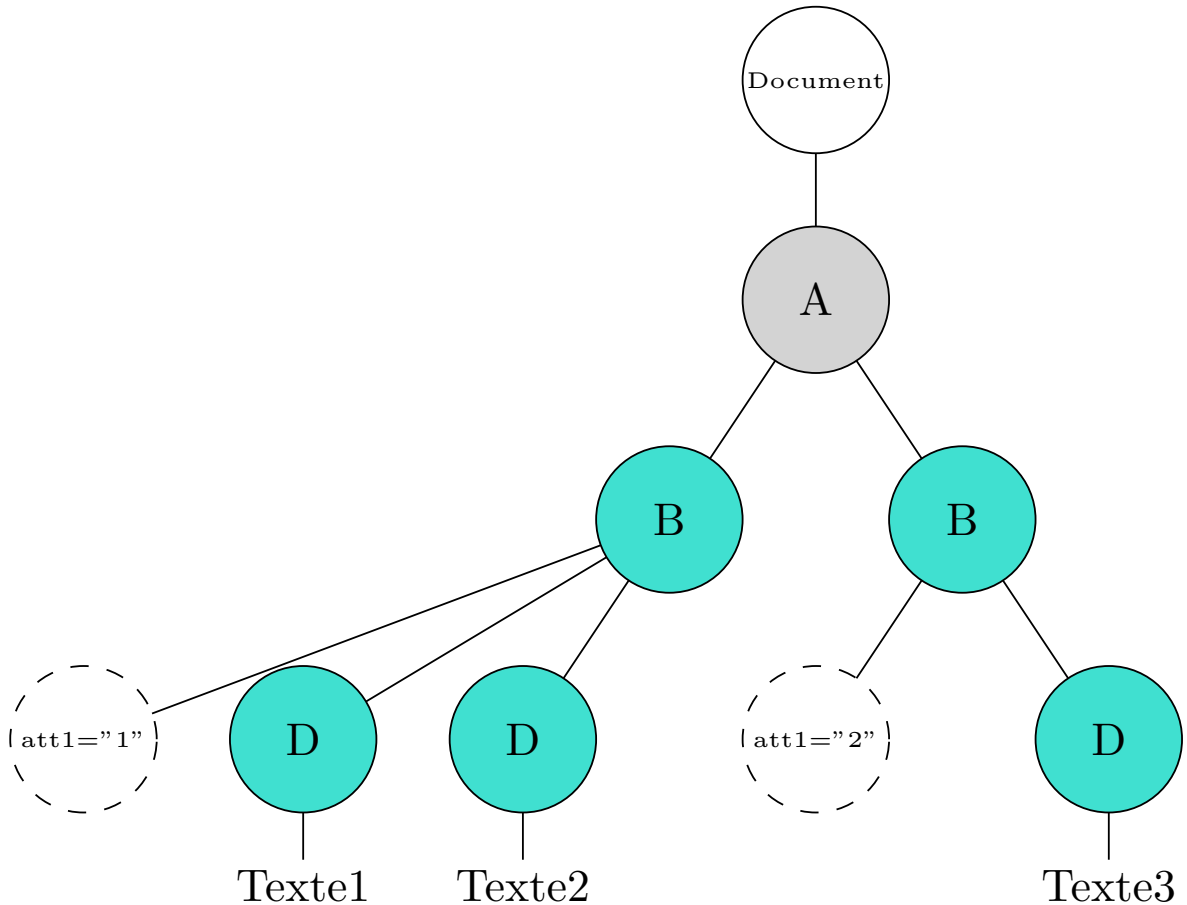
Le nœud actuel n'est pas retourné. Pour cela il faudrait utiliser `descendant-or-self`.



Exemples d'interprétation d'axe

`descendant::*`

Ici tous les nœuds descendants sont retournés à l'exception des attributs ***et*** des nœuds texte, étant donné que nous avons utilisé *, ce qui signifie que l'on cherche n'importe quel élément (et non pas n'importe quel nœud).

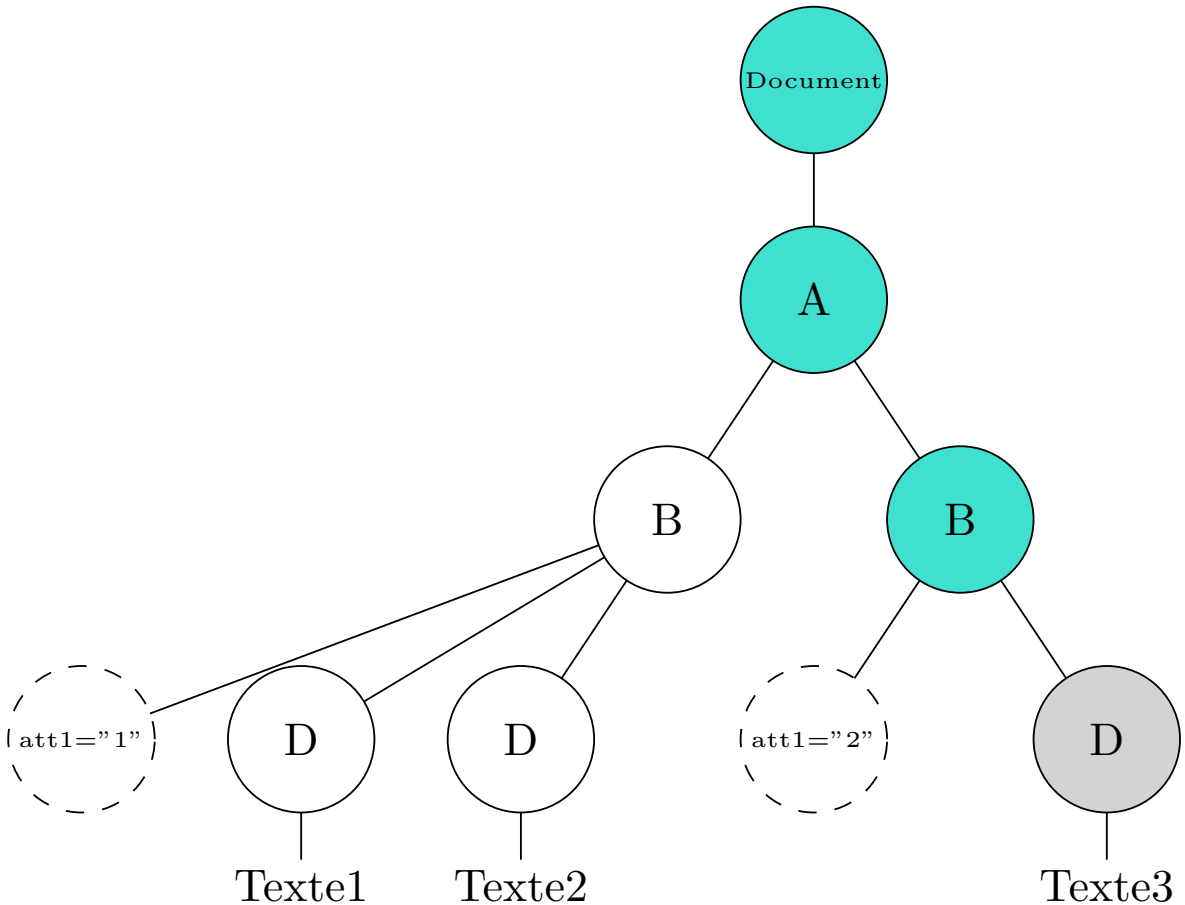


Exemples d'interprétation d'axe

`ancestor::node()`

L'axe `ancestor` retourne tous les nœuds ancêtres, ***y compris*** la racine du document. Le test de nœud est soit un nom d'élément, soit `*`, soit `node()`.

Le nœud actuel n'est pas retourné. Pour cela il faudrait utiliser `ancestor-or-self`.

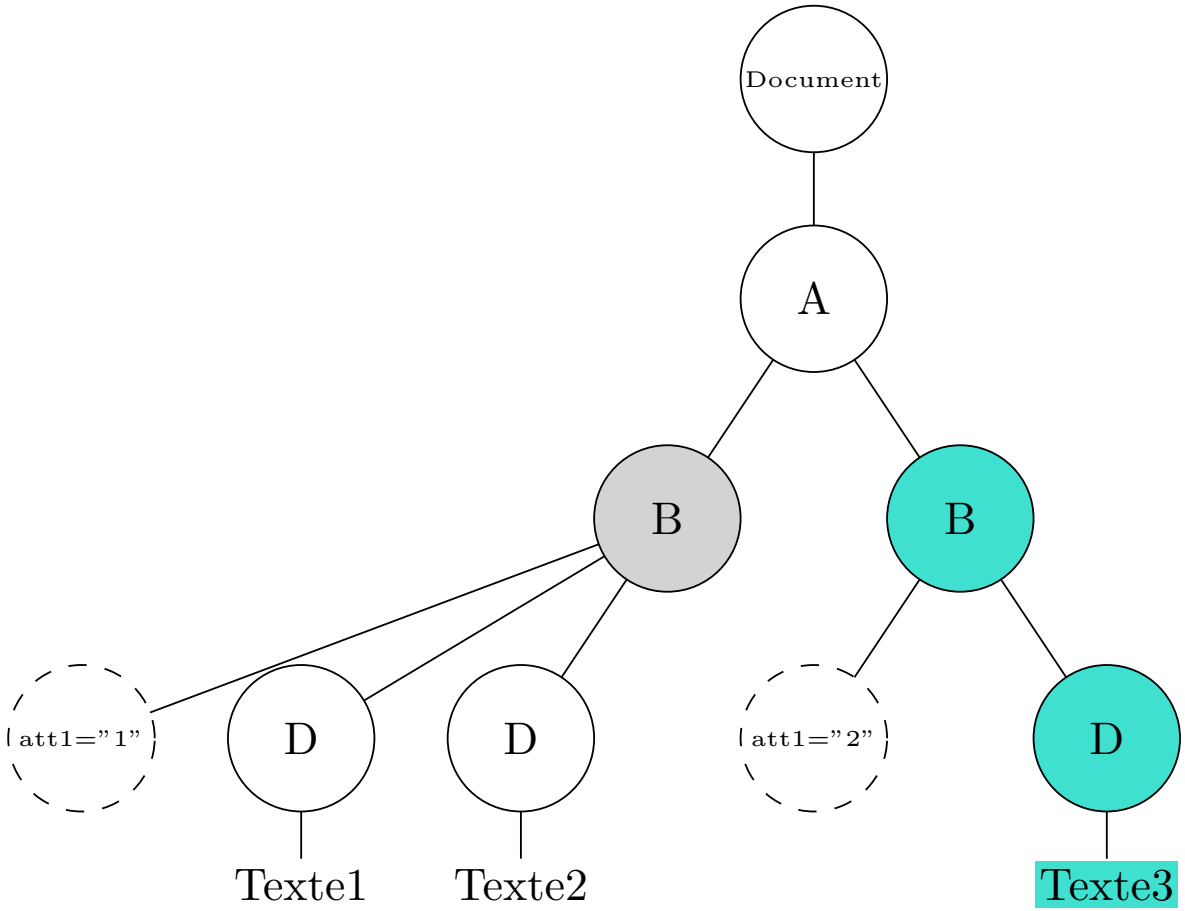


Exemples d'interprétation d'axe

`following::node()`

L'axe `following` récupère les nœuds suivants dans l'ordre de l'arbre XML par rapport au nœud courant. Les attributs ne sont pas sélectionnés. Le test de nœud est soit `*`, soit `text()`, soit `node()`.

L'axe `preceding` récupère les nœuds précédents.

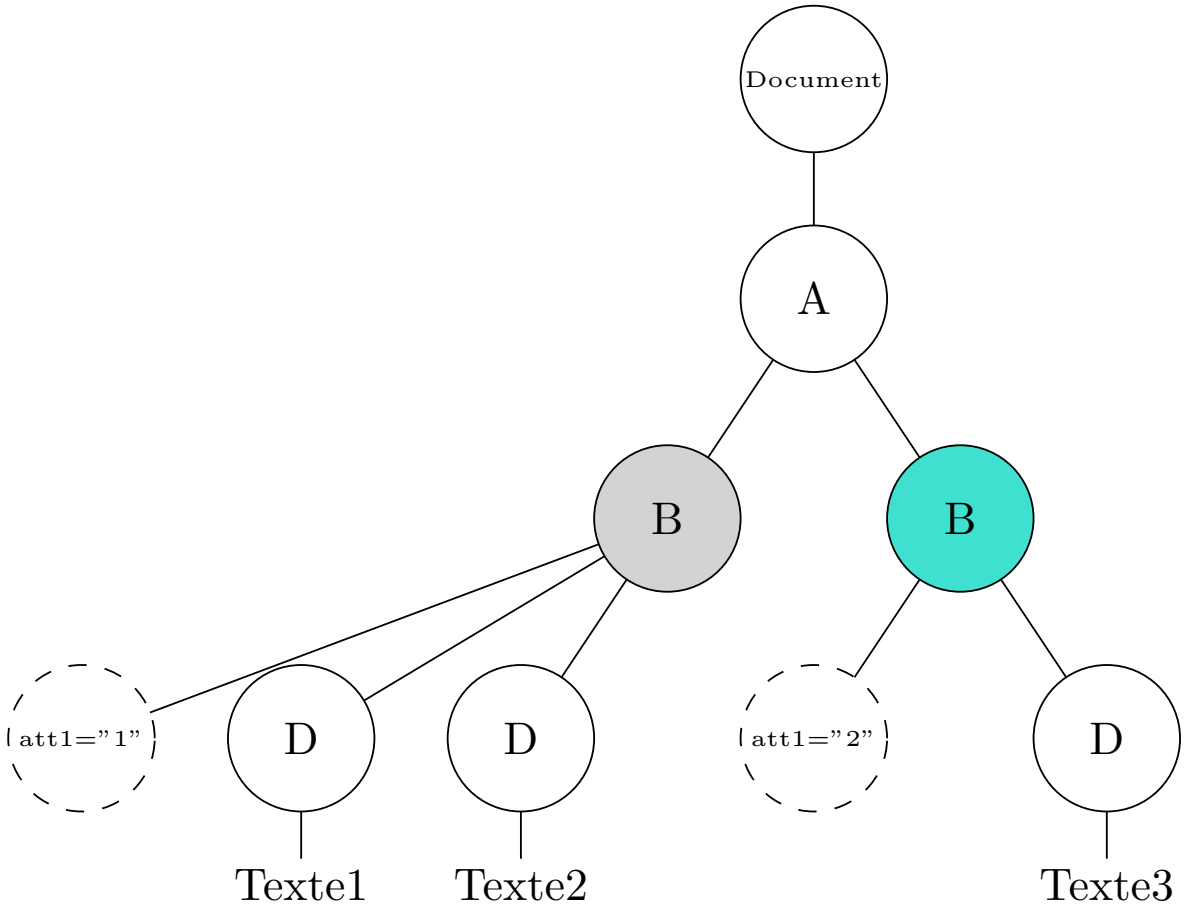


Exemples d'interprétation d'axe

`following-sibling::node()`

L'axe `following-sibling` récupère les nœuds frères/sœurs suivant dans l'ordre de l'arbre XML. En d'autre terme, cela retourne les suivants possédants le même parent que le nœud courant.

`preceding-sibling` permet d'obtenir les nœuds précédents possédant le même parent que le nœud courant.



Abréviations

Sommaire des abréviations :

- `nom` = `child::nom`.
- `.` = `self::node()`.
- `..` = `parent::node()`.
- `@attr1` = `attribute::attr1`.
- `a//b` = `a/descendant-or-self::node()/b`.
- `//a` = `/descendant-or-self::node()/a`.
- `/` = `/self::node()`.

Exemple

- `@b` : sélectionne l'attribut `b` du nœud courant.
- `../*` : sélectionne les éléments frères/sœurs du nœud courant en incluant le nœud actuel si c'est un élément.
- `//@attr1` : sélectionne tous les attributs de l'arbre XML se nommant `attr1`.

Tests de nœud

Un test de nœud est de l'une des formes suivantes :

- `node()` : n'importe que nœud.
- `text()` : n'importe quel nœud texte.
- `*` : n'importe quel élément (ou n'importe quel attribut pour l'axe `attribute`).

Exemple

- `a/node()` : sélectionne les nœuds enfants du nœud a, lui même enfant du nœud courant.
- `/*` : sélectionne la racine du document.

Les prédicats XPath

- Une expression booléenne, construite avec des tests et les connecteurs booléens `and` et `or`. La négation est exprimée à l'aide de la fonction `not()`.
- Un test est :
  - soit une expression XPath dont le résultat est converti en booléen;
  - soit une comparaison ou un appel à une fonction booléenne.

Remarque : l'évaluation des prédicats requière un certain nombre de règles de conversion. Plus précisément, il faut convertir des nœuds ou ensemble de nœuds vers le type approprié.

Exemple

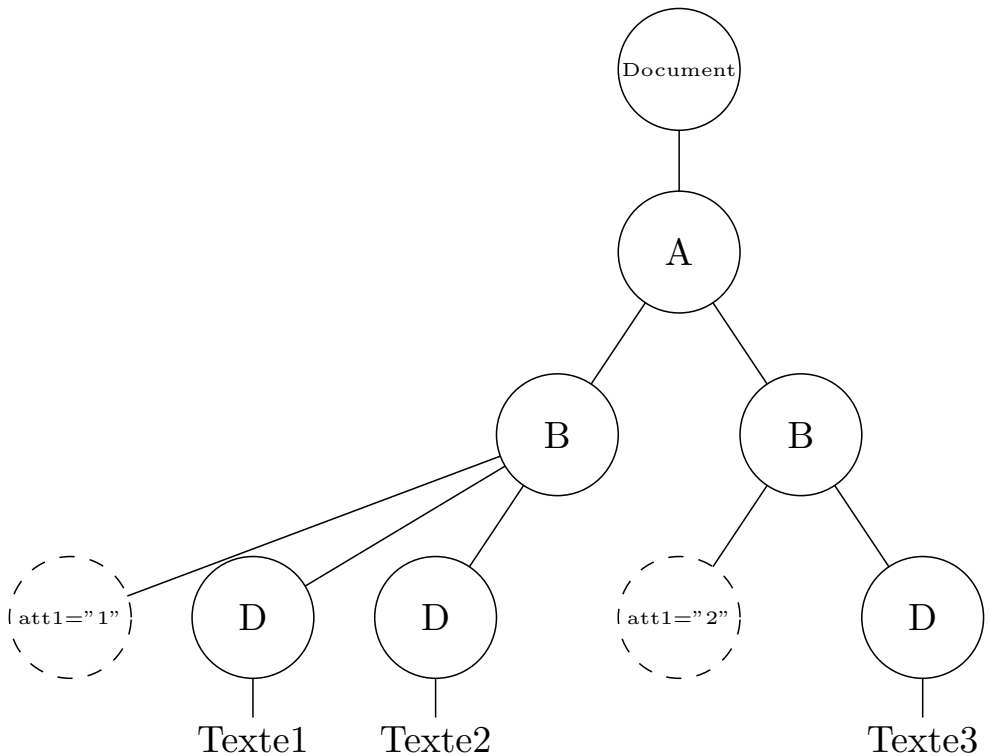
- `//B[@att1=1]` : les nœuds `B` possédant un attribut `att1` égal à 1.
- `//B[@att1]` : les nœuds `B` possédant un attribut nommé `att1`. `@att1` est ici une expression XPath convertie en booléen.
- `//B/descendant::text()[position()=1]` : récupère pour chacun des nœuds `B`, son premier descendant de type texte. Ceci peut être abrégé en `//B/descendant::text()[1]`.

Évaluation de prédicat

Une étape étant de la forme `axe::test-nœud[P]`. L'ordre d'évaluation avec prédicat est le suivant :

1. `axe::test-nœud` est évalué en premier. Ceci nous donne un résultat intermédiaire $\mathcal{N}$.
2. Ensuite, pour nœud N de $\mathcal{N}$, on évalue le prédicat P par rapport à N. Le résultat final est l'ensemble des nœuds de $\mathcal{N}$ pour lesquels le prédicat P est vrai.

`/A/B/descendant::text()[1]`

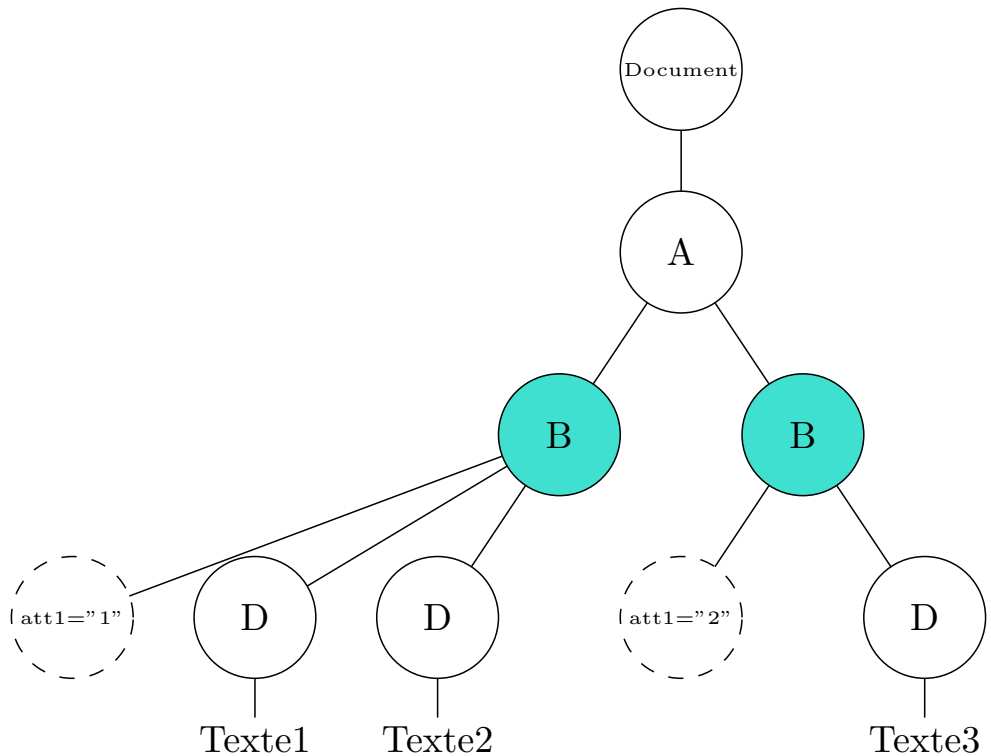


Évaluation de prédicat

Une étape étant de la forme `axe::test-nœud[P]`. L'ordre d'évaluation avec prédicat est le suivant :

1. `axe::test-nœud` est évalué en premier. Ceci nous donne un résultat intermédiaire $\mathcal{N}$.
2. Ensuite, pour nœud N de $\mathcal{N}$, on évalue le prédicat P par rapport à N. Le résultat final est l'ensemble des nœuds de $\mathcal{N}$ pour lesquels le prédicat P est vrai.

`/A/B/descendant::text()[1]`

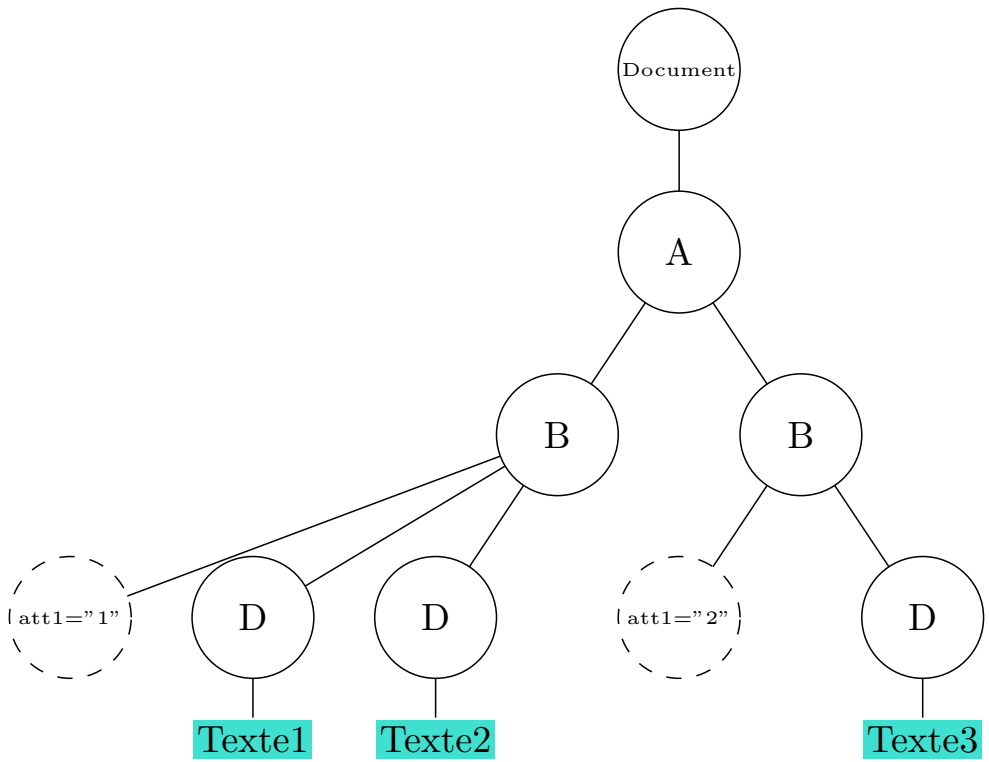


Évaluation de prédicat

Une étape étant de la forme `axe::test-nœud[P]`. L'ordre d'évaluation avec prédicat est le suivant :

1. `axe::test-nœud` est évalué en premier. Ceci nous donne un résultat intermédiaire $\mathcal{N}$.
2. Ensuite, pour nœud N de $\mathcal{N}$, on évalue le prédicat P par rapport à N. Le résultat final est l'ensemble des nœuds de $\mathcal{N}$ pour lesquels le prédicat P est vrai.

`/A/B/descendant::text()[1]`

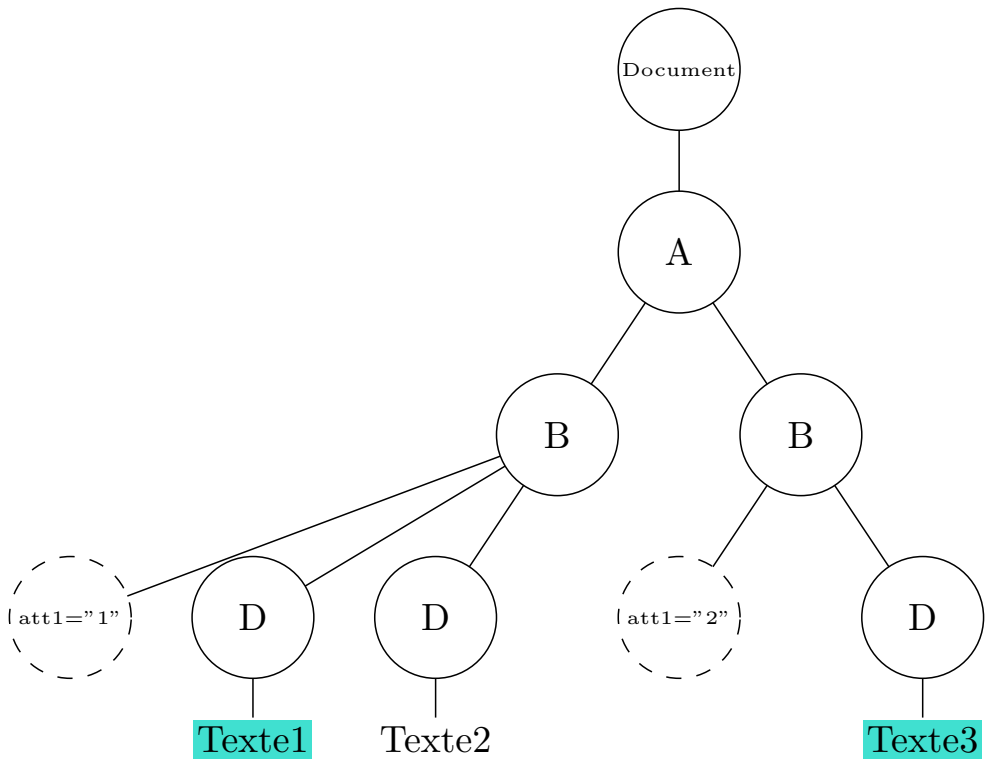


Évaluation de prédicat

Une étape étant de la forme `axe::test-nœud[P]`. L'ordre d'évaluation avec prédicat est le suivant :

1. `axe::test-nœud` est évalué en premier. Ceci nous donne un résultat intermédiaire $\mathcal{N}$.
2. Ensuite, pour nœud N de $\mathcal{N}$, on évalue le prédicat P par rapport à N. Le résultat final est l'ensemble des nœuds de $\mathcal{N}$ pour lesquels le prédicat P est vrai.

`/A/B/descendant::text()[1]`



Le système de types de XPath 1.0

Il existe quatre type primitifs :

Type	Description	Littérale	Exemples
boolean	Valeur booléenne	Aucun	`true()`, `not($a=3)`
number	Nombre à virgule	12, 12.5	`1 div 33`
string	Chaîne de caractères	"to", "ti"	`concat("Hello", "!")`
nodeset	Ensemble de nœuds	Aucun	`/a/b[c=1 or @e]/d`

Les fonctions `boolean()`, `number()` et `string()` sont des fonctions permettant de convertir les autres types en celui demandé. Cependant, cette conversion est réalisée de manière implicite la plus part du temps.

- `boolean()` convertit vers un booléen.
- `number()` convertit vers un nombre à virgule flottante.
- `string()` convertit vers une chaîne de caractères.

Règles de conversion

Conversion vers un booléen :

- Un nombre est **vrai** si il est différent de 0 et *NaN* (Not a Number).
- Une chaîne de caractères des **vrai** si sa longueur n'est pas 0.
- Un ensemble de nœuds est **vrai** si il est non vide.

Conversion d'un ensemble de nœuds vers une chaîne de caractères :

- La conversion en chaîne de caractères d'un ensemble de nœuds est égale à la conversion en chaîne de caractères du premier objet de l'ensemble dans par rapport à l'ordre de l'arbre XML.
- La conversion en chaîne de caractère d'un élément ou du nœud racine document est la concaténation des caractères de type données brutes de tous les nœuds *en dessous*.
- La conversion en chaîne de caractères d'un nœud texte est égale au texte lui même.
- La conversion en chaîne de caractères d'un nœud attribut est la valeur de l'attribut.

Exemples de conversion

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <a toto="3">
4     <b titi="tutu"><c/></b>
5     <d>tata</d>
6  </a>
```

- `string(/)` : a pour résultat ***"tata"***.
- `string(/a/@toto)` : a pour résultat ***"3"***.
- `boolean(/a/b)` : a pour résultat `true()`.
- `boolean(/a/e)` : a pour résultat `false()`.

Pour plus de détails :

`https://developer.mozilla.org/fr/docs/Web/XPath/Fonctions/boolean`
`https://developer.mozilla.org/fr/docs/Web/XPath/Fonctions/number`
`https://developer.mozilla.org/fr/docs/Web/XPath/Fonctions/string`

Les opérateurs

Les opérateurs suivants peuvent être utilisés en XPath :

- **+, -, *, div, mod** : les opérateurs standards d'arithmétique. Attention, l'opérateur **div** est utilisé à la place de **/**.
- **or, and** : les connecteurs booléens standards.
- **=, !=** : les opérateurs d'égalité. Ils peuvent être utilisés sur les chaînes de caractères, les booléens et les nombres. Attention, l'expression `//a!=3` signifie que l'on cherche un élément a dont la valeur des données brutes est différente de 3.
- **<, <=, >=, >** : les opérateurs de comparaisons. Ils peuvent être uniquement utilisés pour comparer des nombres et non pas des chaînes de caractères.
- **|** : opérateur d'union. Il permet de faire l'union entre différents ensembles de nœuds. Par exemple : `node()|@*`.

Les fonctions de nœuds

Ici, **$s** fait référence à une variable XPath représentant un ensemble de nœuds.

- `count($s)` : compte le nombre d'objets contenu dans l'ensemble de nœud $s.
- `local-name($s)` : retourne le nom du premier objet de l'ensemble de nœud $s dans l'ordre de l'arbre XML, sans le préfixe de l'espace de nom. Si aucun paramètre n'est donné c'est l'objet du contexte actuel qui est utilisé.
- `namespace-uri($s)` : retourne l'URI (Uniform Resource Identifier) de l'espace de nom attaché au premier objet de l'ensemble de nœuds $s dans l'ordre de l'arbre XML. Si aucun paramètre n'est donné c'est l'objet du contexte actuel qui est utilisé.
- `name($s)` : retourne le nom du premier objet de l'ensemble de nœuds dans l'ordre de l'arbre XML, en incluant son espace de nom. Si aucun paramètre n'est donné c'est l'objet du contexte actuel qui est utilisé.

Les fonctions de chaînes de caractères

- `concat($s1, ..., $s2)` : concatène les chaînes de caractères $s1, . . . , $s2.
- `starts-with($a, $b)` : retourne `true()` si la chaîne $a commence par $b.
- `contains($a,$b)` : retourne `true()` si la chaîne $a contient $b.
- `substring-before($a, $b)` : retourne la sous-chaîne de $a avant la première occurrence de $b.
- `substring-after($a, $b)` : retourne la sous-chaîne de $a après la première occurrence de $b.
- `substring($a, $n, $l)` : retourne la sous-chaîne de $a de longueur $l commençant à l'indice $n (le premier indice étant 1). $l est optionnel.
- `string-length($a)` : retourne la longueur de la chaîne $a.
- `normalize-space($a)` : supprime tous les espaces superflues de la chaîne $a (espaces avant/après la chaîne et doubles espaces).
- `translate($a,$b,$c)` : retourne la chaîne $a où toutes les occurrences d'un caractère de $b ont été remplacés par le caractère à la même position dans $c.

Les fonctions de nombre et booléens

- `not($b)` : retourne la négation logique du booléen $b.
- `sum($s)` : retourne la somme des valeurs des nœuds dans l'ensemble de nœuds $s.
- `floor($n)` : retourne la valeur arrondie inférieure du nombre $n.
- `ceiling($n)` : retourne la valeur arrondie supérieure du nombre $n.
- `round($n)` : retourne la valeur arrondie (le plus proche entier) du nombre $n.

Exemple

- `count(//*)` : retourne le nombre d'éléments dans le document.
- `normalize-space("  titi   toto ")` : retourne la chaîne `"titi toto"`.
- `translate("baba","abcdef", "ABCDEF")` : retourne la chaîne `"BABA"`.
- `round(3.457)` : retourne le nombre 3.

XPath 2.0

XPath 2.0 est une extension de XPath 1.0, compatible avec les requêtes XPath 1.0. Voici les principales différences :

- **Modèle de données amélioré** : étroitement lié à XML Schema.
- **Plus puissant** : des nouveaux opérateurs, comme les boucles et un meilleur contrôle de la sortie.
- **Extensible** : il est possible de créer des fonctions définies par l'utilisateur.

Il existe des nouveaux test de nœuds en XPath 2.0 :

- `item()` : n'importe quel nœud ou valeur atomique.
- `element()` : n'importe quel élément.
- `element(auteur)` : n'importe quel élément nommé `auteur`.
- `element(*,xs:personne)` : n'importe quel élément de type `xs:personne`.
- `attribute()` : n'importe quel attribut.

Il est également possible de faire des expressions imbriquées. N'importe quel expression retournant une liste de nœuds peut être utilisé comme étape :
`/livre/ (auteur | editeur) /name`

/descendant::femme/child::age/child::text()

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

/descendant::femme/child::age/child::text()

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

```
/descendant::prenom[child::text()="Romeo"]/..::*/attribute::id
```

```
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>63</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>65</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>25</age>
      <poids>73</poids>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>18</age>
      <poids>54</poids>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
   <homme id="5">
      <prenom>Paul</prenom>
      <age>5</age>
      <poids>10</poids>
      <pere>3</pere>
   </homme>
</famille>
```

/descendant::prenom[child::text()="Romeo"]/..::*/attribute::id

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

Les prénoms des personnes plus lourdes.

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

Les prénoms des personnes plus lourdes.

```
/famille/*[not(poids < //poids)]/prenom/text()
```

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

Les prénoms des personnes plus lourdes. ***Attention*** :

```
/famille/*[not(poids >= //poids)]/prenom/text()
```

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

Les ids des hommes qui ne sont père d'aucune personne.

```xml
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>63</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>65</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>25</age>
      <poids>73</poids>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>18</age>
      <poids>54</poids>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
   <homme id="5">
      <prenom>Paul</prenom>
      <age>5</age>
      <poids>10</poids>
      <pere>3</pere>
   </homme>
</famille>
```

Les ids des hommes qui ne sont père d'aucune personne.

```
//homme[not(@id=//pere)]/@id
```

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

Les ids des hommes qui ne sont père d'aucune personne. ***Attention*** :

`//homme[@id!=//pere]/@id`

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

```
//pere[not(text() = following::text())]/../@id
```

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

```
//pere[not(text() = following::text())]/../@id
```

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

```
//pere[not(text() = following::text())]/../@id
```

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```

```
//pere[not(text() = following::text())]/../@id
```

```
1  <famille id="MARTIN">
2     <femme id="1">
3        <prenom>Juliette</prenom>
4        <age>63</age>
5        <poids>58</poids>
6     </femme>
7     <homme id="2">
8        <prenom>Romeo</prenom>
9        <age>65</age>
10       <poids>97</poids>
11    </homme>
12    <homme id="3">
13       <prenom>Max</prenom>
14       <age>25</age>
15       <poids>73</poids>
16       <pere>2</pere>
17       <mere>1</mere>
18    </homme>
19    <femme id="4">
20       <prenom>Marie</prenom>
21       <age>18</age>
22       <poids>54</poids>
23       <pere>2</pere>
24       <mere>1</mere>
25    </femme>
26    <homme id="5">
27       <prenom>Paul</prenom>
28       <age>5</age>
29       <poids>10</poids>
30       <pere>3</pere>
31    </homme>
32 </famille>
```