Selectores de nós:

/	representa a raíz ou uma relação pai/filho
.	representa o nó corrente
..	representa o nó pai do nó corrente
@	modo de aceder a atributos e.g., @nomedoatributo
*	selecciona todos os elementos filhos do nó actual
@*	seleciona todos os atributos do nó actual
//	todos os nós descendentes de um nó ou no princípio de um caminho todos os descendentes da raíz
:	separador de namespace
::	separador de eixo de navegação
text()	selecciona nós que contêm texto
node()	selecciona nós de qualquer tipo

Exemplos de selecção de nós:

../../title	selecciona o elemento `title` filho do avô (pai do pai) do nó corrente
//author	selecciona todos os nós `author` independentemente da sua posição no documento
/catalog/book/@id	selecciona o atributo `id` do nó `book`, filho de `catalog` debaixo da `raíz`
/catalog/*/title	selecciona todos os elementos `title` descendentes de todos os nós filhos de `catalog`
./author	selecciona todos os nós `author` filhos do nó actual
//@*	selecciona todos os atributos do documento
.//author	selecciona todos os elementos `author` descendentes do nó actual
//processing-instruction('php')	selecciona todos os nós instrução de processamento do tipo `php`

Operadores *XPath*

Operador	Descrição	Exemplo
|	Alternativa (OU)	//book | //cd
+	Adição	6 + 4
-	Subtracção	6 - 4
*	Multiplicação	6 * 4
div	Divisão	8 div 4
mod	Módulo (Resto da divisão Inteira)	5 mod 2
=	Igual	price = 9.80
!=	Diferente	price != 9.80
< (<)†	Menor	price < 9.80
<= (<)†	Menor ou igual a	price <= 9.80
> (>)†	Maior	price > 9.80
>= (>=)†	Maior ou igual a	price >= 9.80
or	Ou lógico	price=9.80 or price=9.70
and	E lógico	price > 9.00 and price < 9.90

† Dentro de um documento XSLT, devem ser usadas as entidades; num query com `javascript` DOM ou numa pesquisa *XPath* (ex. IntelliJ, netbeans, XML CopyEditor, Notepat++, etc.) usa-se o símbolo

```
<xsl:template match="/" >
 <html><body><h3>Lista de livros </h3>
  <xsl:apply-templates select="catalog/book">
    <xsl:sort select="pubdate" data-type="number"
                               order="ascending"/>
    <xsl:sort select="title" data-type="text"
                               order="ascending"/>
  </xsl:apply-templates>
 </body></html>
</xsl:template>
```

de navegação *XPath*

d	Selecciona os filhos do nó corrente, eixo por omissão, qualquer nó excepto atributo ou namespace
ent	Selecciona o nó pai do nó corrente, se existir, equivalente a ..
	Selecciona o nó corrente, self::* equivalente a .
ribute	Selecciona os atributos do nó corrente, equivalente a @
estor	Nós ascendentes até à raiz; pai, avó, etc.
estor-or-self	Ascendentes incluindo o próprio nó
cendant	Descendentes do nó actual: filhos, netos, etc.
cendant-or-self	Descendentes incluindo o próprio nó
ceding-sibling	Irmãos anteriores (todos os filhos do mesmo pai) ao nó actual, se o nó for atributo ou namespace devolve uma lista vazia
following-sibling	Irmãos seguintes (todos os filhos do mesmo pai) ao nó actual, se o nó for atributo ou namespace devolve uma lista vazia
preceding	Selecciona todos os nós que aparecem antes no documento e não são ascendentes
following	Selecciona todos os nós que aparecem depois no documento e não são descendentes
namespace	Selecciona nós namespace do nó actual

Declaração como conteúdo do elemento

```
<xsl:variable name="header">
   <tr>
      <th>Element</th>
      <th>Description</th>
   </tr>
</xsl:variable>
```

Utilização da variável

```
<xsl:value-of select="$header"/>
```

se for usada como conteúdo de um atributo podem-se usar { } para substituir `<xsl:value-of …>` , que não pode ser usado

```
<img src="{$link}" …
```

as { } também são válidas para qualquer expressão XPath

`sum(node-set)`
Soma os valores seleccionados com o query *XPath e.g.*, `sum(price)`

`contains(string, substring)`
devolve true se a substring existe na string *e.g.*, `contains(title, "XML")`

Regioes existentes: `<xsl:apply-templates select="/ns:Catalogo/ns:Garrafa/ns:InfoAdicional/ns:Producao/ns:Regiao[not(.=../../../preceding-sibling::ns:Garrafa/ns:InfoAdicional/ns:Producao/ns:Regiao)]"/>`

Tipos de bebidas: `<xsl:value-of select="count(/ns:Catalogo/ns:Garrafa/ns:Tipo[not(.=../preceding-sibling::ns:Garrafa/ns:Tipo)])"/>`

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <head>
                <title>projetos.xsl</title>
            </head>
            <body>
                <p>Projetos 3+:  <xsl:value-of select="count(/Gestao/Projetos/Projeto[count(.//Elemento) &gt; 3])"/></p>
                <table>
                    <tr>
                        <th>Nome</th>
                        <th>Projectos</th>
                    </tr>
                    <xsl:apply-templates select="//Pessoa[@id = //Elementos/*/@id]">
                        <xsl:sort select="@nome" order="ascending"/>
                    </xsl:apply-templates>
                </table>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="//Pessoa">
        <xsl:variable name="mid" select="@id"/>
        <tr>
            <td>
                <xsl:value-of select="@nome"/>
            </td>
            <td>
                <xsl:value-of select="count(/Gestao/Projetos/Projeto[.//Elemento/@id = $mid])"/>
            </td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

Parsers em descida recursivo

A descida recursiva pode ser usada, para implementar *parsers* preditivos;

Um *parser* preditivo é capaz de escolher a produção a aplicar simplesmente sabendo o **não terminal** actual e o **terminal** a ser processado;

Este tipo de gramáticas são chamadas de LL(1), onde:

- o primeiro "L" indica que a frase é processada da esquerda para a direita;
- o segundo "L" indica que se usa a derivação mais à esquerda;
- o (1) indica o número de terminais de avanço necessários para escolher entre produções alternativas;

Todas as gramáticas do tipo 3 podem ser convertidas em LL(1).

Formalmente, uma gramática é definida pelo tuplo $G = (V, \Sigma, P, S)$, no qual:

- V – um conjunto finito, não vazio, de **variáveis** (símbolos **não terminais**);
- Σ – é um conjunto finito, não vazio, dito **alfabeto** ou conjunto de símbolos **terminais**;
- P – conjunto de **produções**, regras da gramática. A sua forma geral é a seguinte: $\alpha \rightarrow \beta$ que definem a forma como o conjunto de símbolos α podem ser substituídos pelo conjunto de símbolos β;
- S – **símbolo inicial** a partir do qual todas as frases são derivadas.

Gramáticas do tipo 3

Gramáticas regulares. A produções são da forma:

$$A \rightarrow a$$
$$A \rightarrow aB$$
$$A \rightarrow \varepsilon$$

em que ***A*** e ***B*** são dois quaisquer **não terminais** singulares e ***a*** é um qualquer **terminal** singular.

Estas são as formas de gramáticas mais restritas em termos de poder de representação.

Lineares à direita

$$A \rightarrow a$$
$$A \rightarrow aB$$
$$A \rightarrow \varepsilon$$

Lineares à esquerda

$$A \rightarrow a$$
$$A \rightarrow Ba$$
$$A \rightarrow \varepsilon$$

Nota: A produção $A \rightarrow a$ pode ser omitida, pois pode ser derivada a partir das outras duas.

Gramáticas do tipo 2

Gramáticas independentes do contexto. A produções são da forma:

$$A \rightarrow \alpha$$

em que α é uma sequência arbitrária de símbolos terminais e não terminais, sendo ***A*** um qualquer não terminal singular.

Tal significa que qualquer ocorrência de ***A*** pode ser substituída por α independentemente do contexto.

Reconhece frases do tipo $\{a^n b^p d^q c^n : n, p, q \geq 0\}$

Exemplo

$$A \rightarrow aAc$$
$$A \rightarrow BD$$
$$B \rightarrow bB|\varepsilon$$
$$D \rightarrow dD|\varepsilon$$

Gramáticas do tipo 1

Gramáticas dependentes do contexto. A produções são da forma:

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

em que α,β e γ são sequências arbitrárias de símbolos terminais e não cterminais, sendo que γ não é nulo e ***A*** é um qualquer não terminal singular.

Estas gramáticas têm que respeitar a condição $|\alpha A \beta| \leq |\alpha \gamma \beta|$

A única excepção à regra anterior ($|\alpha A \beta| \leq |\alpha \gamma \beta|$) é a produção inicial ser $S \rightarrow \varepsilon$, mas **se e só se** o S não existir do lado direito (para permitir a palavra vazia).

Reconhece frases do tipo $\{a^n b^n c^n : n \geq 1\}$

Exemplo

$$S \rightarrow aSBC|abC$$
$$CB \rightarrow XB$$
$$XB \rightarrow XC$$
$$XC \rightarrow BC$$
$$bB \rightarrow bb$$
$$bC \rightarrow bc$$
$$cC \rightarrow cc$$

Gramáticas do tipo 0

Gramáticas livres ou sem restrições. A produções são da forma:

$$\alpha \rightarrow \beta$$

em que tanto α como β são sequências arbitrárias de símbolos terminais e não terminais.

O lado esquerdo da produção não pode ser vazio.

Reconhece frases do tipo $\{a^n b^n c^n : n \geq 1\}$

Exemplo

$$S \rightarrow aSBC|abC$$
$$CB \rightarrow BC$$
$$bB \rightarrow bb$$
$$bC \rightarrow bc$$
$$cC \rightarrow cc$$

Minimização de AFDs

	0	1
$\rightarrow s_0$	s_0	s_1
s_1	s_2	s_1
$*s_2$	s_0	s_3
s_3	s_2	s_3

		0	1
	$\rightarrow s_0$	s_0	s_1
A	s_1	s_2	s_1
	s_3	s_2	s_3
B	$*s_2$	s_0	s_3

		0	1
	$\rightarrow s_0$	s_0 (A)	s_1 (A)
A	s_1	s_2 (B)	s_1 (A)
	s_3	s_2 (B)	s_3 (A)
B	$*s_2$	s_0	s_3

	0	1
$\rightarrow s_0$	s_0	s_1
s_1	s_2	s_1
s_3	s_2	s_3
$*s_2$	s_0	s_3

		0	1
A	$\rightarrow s_0$	s_0	s_1
	s_1	s_2	s_1
B	s_3	s_2	s_3
C	$*s_2$	s_0	s_3

		0	1
A	$\rightarrow s_0$	s_0	s_1
	s_1	s_2 (C)	s_1 (B)
B	s_3	s_2 (C)	s_3 (B)
C	$*s_2$	s_0	s_3 s_1

Conversão formal de AFNs em AFDs (simplificada)

1. Copiar estado inicial
2. Sempre que aparecer novos conjuntos, criá-los na tabela
4. Criar um nome para cada estado
5. Substituir nomes nas transições
6. Eliminar nomes antigos

Enumeration	Define uma lista de valores válidos
fractionDigits	Especifico o número máximo de casas decimais permitidas. Deve ser maior ou igual que zero.
Length	Especifica o número exacto de caracteres ou itens permitidos. Deve ser maior ou igual que zero.
maxExclusive	Especifica o valor máximo para valores numéricos (o valor deve ser menor que este valor).
maxInclusive	Especifica o valor máximo para valores numéricos (o valor deve ser menor ou i
maxLength	Espe mai
minExclusive	Espe que
minInclusive	Espe ou i
minLength	Espe mai
Pattern	Defi
totalDigits	Espe
whiteSpace	Espe trat

```
<xs:element name="id" type="xs:ID">
```
O tipo `xs:ID` é usado para validar um identificador único e usado em atributos ou elementos simples;
Não podem existir dois elementos/atributos do tipo `xs:ID` com o mesmo identificador.
No código seguinte, o validador de *XSD*, dá erro no segundo `<id>` pois é repetido:
```
<id>abcd</id>
<id>abcd</id>
```
O conteúdo do tipo `xs:ID` é uma palavra e tem que iniciar com uma letra.
```
<xs:element name="idAluno" type="xs:IDREF">
```
O tipo `xs:IDREF` é usado para validar atributos ou elementos que no documento existam como `xs:ID`;
Se não existir no documento um `xs:ID` com o mesmo valor é gerado um erro pelo validador *XML*.
Também existe o tipo `xs:IDREFS` que aceita uma lista de `xs:ID` separados por espaços

eger e maxOccurs=
o valor destes

```
<xsd:element name="
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Audi"/>
      <xsd:enumeration value="Golf"/>
      <xsd:enumeration value="BMW"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

- xsd:integer
- xsd:boolean
- xsd:date
- xsd:time

```
<xsd:element name="idade">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="letra">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[a-z]"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

- "**preserve**" - significa que o processador XML não vai remover nenhum carácter vazio
- "**replace**" - significa que o processador XML vai substituir todos os caracteres vazios (quebras de linha, tabs, espaços) por espaços.
- "**collapse**" - significa que o processador XML vai remover todos os caracteres vazios (quebras de linha, tabs, espaços são substituídos com espaços, espaços iniciais e finais são removidos, espaços múltiplos são reduzidos a um).

exemplo seguinte define um elemento chamado <endereço> com uma restrição:

```
<xsd:element name="endereço">
 <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:whiteSpace value="collapse"/>
    </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="password">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="password">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="5"/>
      <xsd:maxLength value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="identificacao">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="bi" type="BI"/>
      <xsd:element name="cc" type="CC"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:attribute name="lang" type="xsd:string" default="pt">
<xsd:attribute name="lang" type="xsd:string" fixed="pt"/>
<xsd:attribute name="lang" type="xsd:string" use="optional"/>
<xsd:atrribute name="lang" type="xsd:string" use="required"/>
```

.	any character except newline
\w \d \s	word, digit, whitespace
\W \D \S	not word, digit, whitespace
[abc]	any of a, b, or c
[^abc]	not a, b, or c
[a-g]	character between a & g
Anchors	
^abc$	start / end of the string
\b \B	word, not-word boundary
Escaped characters	
\. * \\	escaped special characters
\t \n \r	tab, linefeed, carriage return
\u00A9	unicode escaped ©
Groups & Lookaround	
(abc)	capture group
\1	backreference to group #1
(?:abc)	non-capturing group
(?=abc)	positive lookahead
(?!abc)	negative lookahead
Quantifiers & Alternation	
a* a+ a?	0 or more, 1 or more, 0 or 1
a{5} a{2,}	exactly five, two or more
a{1,3}	between one & three
a+? a{2,}?	match as few as possible
ab|cd	match ab or cd

```
<xsd:element name="funcionário" type="pessoa"/>
<xsd:element name="estudante" type="pessoa"/>
<xsd:element name="membro" type="pessoa"/>
<xsd:complexType name="pessoa">
    <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

```
<xsd:element name="product">
  <xsd:complexType>
    <xsd:attribute name="prodid" type="xsd:positiveInteger"/>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="nome">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="primeiro" type="xsd:string"/>
         <xsd:element name="ultimo" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="pessoa">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="full_name" type="xsd:string"/>
            <xsd:element name="child_name" type="xsd:string"
            maxOccurs="10" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="pessoas">
  <xsd:complexType>
    <xsd:sequence maxOccurs="10" minOccurs="0"/>
      <xsd:element name="full_name" type="xsd:string"/>
      <xsd:element name="child_name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:group name="persongroup">
    <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
        <xsd:element name="birthday" type="xsd:date"/>
    </xsd:sequence>
</xsd:group>
```

```
<xsd:element name="person" type="personinfo"/>
```

```
<xsd:complexType name="personinfo">
  <xsd:sequence>
    <xsd:group ref="persongroup"/>
    <xsd:element name="country" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

AttributeGroup name - usados para definir grupos de atributos.

```
<xsd:attributeGroup name="personattrgroup">
    <xsd:attribute name="firstname" type="xsd:string"/>
    <xsd:attribute name="lastname" type="xsd:string"/>
    <xsd:attribute name="birthday" type="xsd:date"/>
</xsd:attributeGroup>
```

```
<?xml version="1.0"?>
<xs:schema version="1.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" xmlns:zs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Gestao" type="TGestao"/>
    <xs:complexType name="TGestao">
        <xs:sequence>
            <xs:element name="Projetos" type="TProjetos"/>
            <xs:element name="Pessoas" type="TPessoas"/>
        </xs:sequence>
    </xs:complexType>
    <!-- COMPLETAR schema -->
    <xs:complexType name="TProjetos">
        <xs:sequence maxOccurs="unbounded">
            <xs:element name="Projeto" type="TProjeto"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="TPessoas">
        <xs:sequence maxOccurs="unbounded">
            <xs:element name="Pessoa" type="TPessoa"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="TPessoa">
        <xs:attribute name="id" type="TPessoaID"/>
        <xs:attribute name="nome" type="xs:string"/>
        <xs:attribute name="classificacao" type="TPessoaClass" use="optional"/>
    </xs:complexType>

    <xs:complexType name="TProjeto">
        <xs:sequence>
            <xs:element name="Elementos" type="TElementos"/>
            <xs:element name="DataInicio" type="xs:date"/>
        </xs:sequence>
        <xs:attribute name="id" type="TProjetoID"/>
        <xs:attribute name="designacao" type="TProjetoDesignacao"/>
    </xs:complexType>

    <xs:complexType name="TElementos">
        <xs:sequence maxOccurs="unbounded">
            <xs:element name="Elemento" type="TElemento"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="TElemento">
        <xs:attribute name="id" type="xs:IDREF"/>
    </xs:complexType>

    <xs:simpleType name="TProjetoID">
        <xs:restriction base="xs:ID">
            <xs:pattern value="p[0-9]{3}"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="TPessoaID">
        <xs:restriction base="xs:ID">
            <xs:pattern value="ps[0-9]{3}"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="TProjetoDesignacao">
        <xs:restriction base="xs:string">
            <xs:minLength value="5"/>
            <xs:maxLength value="30"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="TPessoaClass">
    <xs:restriction base="xs:string">
        <xs:pattern value="Excelente|Muito Bom|Bom"/>
    </xs:restriction>
    </xs:simpleType>

</xs:schema>
```