



Licence 3^{ème} année



Analyse syntaxique & Sémantique

P-F. Bonnefoi

Table des matières

1	Contenu et objectifs	6
2	Langage interprété vs compilé: la compilation	7
	Langage interprété vs compilé : l'interprétation	. 8
	Langage interprété vs compilé	9
3	L'analyse lexical : les automates à nombre fini d'états	10
	Expressions régulières ou expressions rationnelles	11
	Analyseur lexical	13
	L'outil Lex	14
	Analyseur lexical	15
	Analyseur lexical: un exemple	16
	Analyseur lexical: un autre exemple	17
	Analyseur lexical: exemple d'un format de fichier	18
	Analyseur lexical: exemples	20
	Analyseur lexical: utilisation	21
	Lex: la gestion des accents	23
4	Rappels sur les grammaires	25
	Rappels sur les grammaires	26
	Les «parsers» LR	31
	Les différentes catégories d'analyseurs LR	32
	Principe de fonctionnement d'un analyseur LR	33

	Exemple de table	34
	Interprétation intuitive d'une table	40
	Conflits	41
	Exemple de conflit	42
5	Analyseur lexical: un avant goût de grammaire?	44
6	Liens entre Lex et un analyseur syntaxique	48
7	L'analyseur syntaxique YACC	56
	Lex & YACC: un exemple de calculette	59
	YACC: gestion des erreurs	63
8	XML: les différents outils	65
	XML vs HTML	68
	Afficher un document XML: XSL	72
	Format XML: DTD, «Document Type Description»	73
	Utilisation de XSL et DTD	74
	DTD: notion de classe de document	75
	Le format XML	76
	XML: les atouts	77
	Exemple de document XML	78
	XML: Utilisation des balises spécifiques	79
	XML: Syntaxe des éléments constitutifs	80
9	DTD : Définition d'une classe de documents	81
	DTD : La notion d'entités	83
	DTD: La notion d'élément	84

DTD : les attributs d'éléments	86
DTD: la notion d'espace de nom	88
10 CSS: présentation rapide	90
XML & CSS	92
11 XSLT: aller plus loin que les CSS	93
XSLT: accès aux données	94
XSLT : opérations avancées	95
XSLT : un exemple de liste de livres	96
XSLT : suite de l'exemple de liste de livres	97
XPath	98
DTD: les limitations	105
12 XSD, «XML Schema Definition»	106
XSD: exemple de carte de visite	107
XSD, un autre exemple de liste de livres	109
XSD: la définition des «éléments» constitutifs du format	110
XSD: les types de données pour un élément1	111
XSD: la définition d'attributs 1	114
XSD: exemples de déclaration d'élément et d'attribut	116
XSD: dérivation de type	117
13 XML et Java	118
SAX: Utilisation	119
SAX: Utilisation 1	120
SAX: un exemple1	121

14 XML DOM & XSLT: utilisation de JavaScript	123
15 XML DOM: JavaScript & AJAX	126
XML AJAX: un exemple	128
XML AJAX: un exemple – suite	129
JQuery: la bibliothèque qui vient au secours d'Ajax	133
16 XML & SOAP	138
17 JQuery, AJAX et XML	140

































Objectifs:

- > retour sur les analyseur lexicaux et syntaxiques : Expression régulières & Grammaires ;
- réalisation d'un «parser».
 - générateur automatique d'analyseurs lexicaux : utilisation de Lex;
 - générateur automatique d'analyseurs sémantiques basés sur les grammaires LR(1): utilisation de Yacc;
- apprentissage et manipulation d'XML:
 - définition d'un format :
 - utilisation d'un parser:

























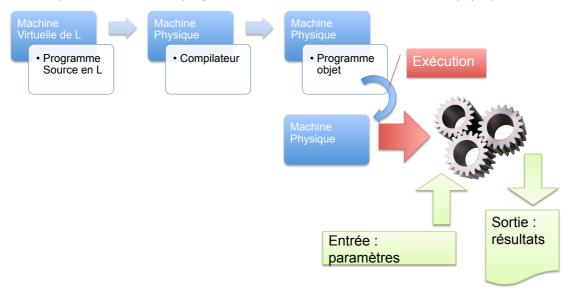




2 Langage interprété vs compilé : la compilation

On écrit un programme en **langage L**, par exemple en C++: ce langage est destiné à une machine «idéalisée», ou virtuelle, qui est plus ou moins proche de la vraie mais où le langage est très bien adapté (en réalité la machine ne gère pas d'objets par exemple).

Enfin, grâce au compilateur, on traduit le programme en vraies instructions de la machine physique.



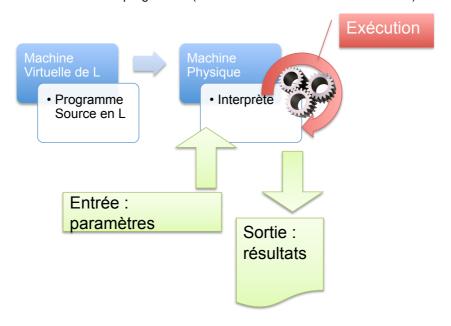
On peut alors **exécuter** le programme sur des entrées et il donne des sorties.



Langage interprété vs compilé : l'interprétation

On écrit un programme en langage L, par exemple en Perl: comme dans le cas d'un langage compilé, ce langage est destiné à une machine «idéalisée», ou virtuelle, plus ou moins proche de la vraie, mais où le langage est très bien adapté (en réalité la machine ne gère pas d'objets par ex.).

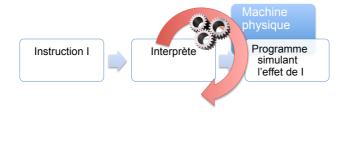
Enfin, grâce à l'interpréteur, on traduit chaque instruction du programme en une suite d'instructions de la machine physique, au fur et à mesure de l'exécution du programme (sur des entrées et il donne des sorties).





Langage interprété vs compilé

Lors de l'**interprétation** d'une instruction, on fait appel à des morceaux de programme réalisant les effets de cette instruction sur la machine :





Lors de l'**exécution** du programme **compilé**, chaque instruction du programme correspond à des instructions de la machine physique.





















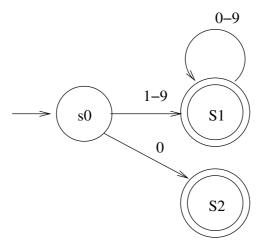




3 L'analyse lexical : les automates à nombre fini d'états

Il est possible de reconnaître un texte à l'aide d'un automate à nombre fini d'états.

Par exemple, voici l'automate permettant de reconnaître des nombres :



Un nombre ne peut commencer par le chiffre zéro que s'il vaut zéro.

Un automate à nombre fini d'états permet de reconnaître des langages dits rationnels.

À un automate, on peut faire correspondre une expression rationnelle (ou régulière, comme en anglais «regular expression»).



Expressions régulières ou expressions rationnelles

Une ER permet de faire de l'appariement de motif, *pattern matching* : il est possible de savoir si un motif est **présent** dans une chaîne, mais également **comment** il est présent dans la chaine (en mémorisant la séquence correspondante).

Une expression régulière est exprimée par une suite de meta-caractères, exprimant :

- * une *position* pour le motif
 - ^ : début de chaîne
 - \$: fin de chaîne

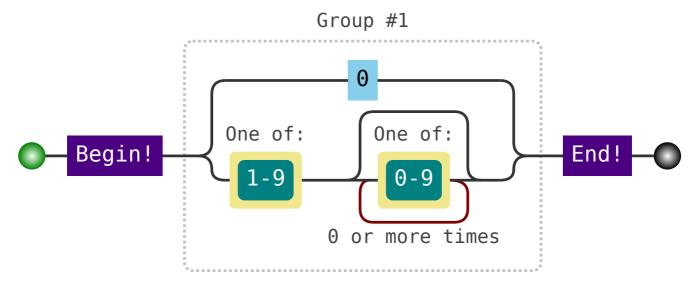
- * un caractère pour lui-même;
 - .: n'importe quel caractère
 - [] : un caractère parmi une liste
 - [^]: tous les caractères sauf...

- * une alternative
 - : ceci ou cela
- * des quantificateurs, qui permettent de répéter le caractère qui les précédent :
 - *: zéro, une ou plusieurs fois
 - +: **une** ou plusieurs fois $\{n\}$: n fois
 - ?: zéro ou une fois $\{n, m\}$: entre n et m fois
- ⋆ des caractères spéciaux : \n : retour à la ligne, \t : tabulation



Un exemple sur la reconnaissance des nombres





http://jex.im/regulex/



Analyseur lexical

Un analyseur lexical ou «lexer»:

- > reçoit un flux de caractères en entrée;
- ▷ lorsqu'il rencontre une suite de caractères qui correspond à un mot clé défini, une unité lexicale ou lexème, un «token», il exécute un ensemble d'actions prédéfinies.

La construction d'un analyseur lexical peut être automatisée à l'aide de l'outil «Lex».

Cet outil peut ensuite être associé à l'outil YACC, «Yet Another Compiler Compiler» ou «bison» :

	lexical		grammar			
	rules		rules			
	V		V			
	+	-+	+	+		
	Lex		Yacc			
	+	-+	+	+		
	V		V			
	+	-+	+	+		
Input ->	> yylex	->	yyparse	->	Parsed	input
	+	-+	+	+		





















L'outil Lex

Il est possible de disposer de l'outil «flex» qui réalise le même travail et accepte le même format de fichier en entrée.

Structure d'un fichier Lex:

```
1 %{
2 Prologue
3 %}
4
5 Déclaration
6
7 %%
8 Expressions régulières de reconnaissance pour chaque token & Actions associées
9 %%
10
11 Epilogue
```

La sortie de l'outil Lex est un programme C :

- \star définissant la fonction ${\tt yylex}$ () permettant l'analyse lexical au sein d'un autre programme ;
- * soit autonome : si rien n'est précisé à la compilation (un appel à la fonction yylex est ajouté automatiquement);
- * soit sous forme d'une bibliothèque à relier, «*linker*» : dans ce cas c'est au programme principal de réaliser l'appel à yylex.



Analyseur lexical

Explications:

- o le prologue délimité par «%{» et «%}» contient :
 - les déclarations de types de données

- des commandes du pré-processeur, des macros;
- les variables globales utilisées par les actions.
- ♦ des «#include» des bibliothèques C.
- o la **déclaration** : contient des définitions de sous expressions régulières :

 les expressions régulières: regroupent les règles de reconnaissance des tokens et associe le code des actions à réaliser;

Il existe un certain nombre de variables C définie par Lex et utilisable dans les actions :

- yytext: qui fournie la valeur reconnue par l'E.R. sous forme de chaîne de caractères;
- yyleng: qui fournie la longueur de la chaîne précédente.
- l'epilogue: il peut contenir des fonctions supplémentaires (bibliothèques de fonction).

En particulier, on peut redéfinir les opérations **prédéfinies** de base de l'analyseur lexical, comme les opérations de gestion des caractères de l'entrée (input (), output (), yysetstr (), etc.)

Le «prologue» et «l'épilogue» peuvent être vides, par exemple pour définir une commande de filtre sur du texte en entrée, c-à-d un programme autonome réalisant des traductions du texte en entrée vers le texte en sortie.



Analyseur lexical: un exemple

Soit le fichier exemple_analyseur.lex:

```
1 %{
2 #include <stdio.h>
3 %}
4
5 %%
6 stop printf("Machine arretee\n");
7 start printf("Machine en marche\n");
8 %%
```

Ensuite, on donne ce fichier à la commande flex qui crée un fichier lex.yy.c qui pourra ensuite être compilé:

```
$ flex exemple_analyseur.lex
$ gcc -o mon_parseur lex.yy.c -lfl
```

Lors de la compilation, on ajoute -lfl pour faire le lien avec la bibliothèque de flex (contenant une fonction main par défaut).

Lors de l'exécution, on obtient :

Un ctrl-d permet de mettre fin à la saisie (fermeture de l'entrée standard).



Analyseur lexical: un autre exemple

```
1 % {
2 #include <stdio.h>
3 8 }
5 88
 [0-9]+
                    printf("NOMBRE\n") ;
 [a-zA-Z][a-zA-Z0-9] printf("MOT\n");
8 [ \t]+
                      /* ignorer les espaces et les tabulations*/
9 %%
En entrée:
                                             En sortie:
$ ./mon_parseur
                                             NOMBRE
                                             MOT
toto
                                             MOT
                                             MOT
t0
                                             NOMBRE
1t0
                                             MOT
                                             NOMBRE
12t
```

Question: pourquoi un t se retrouve en sortie?























Analyseur lexical: exemple d'un format de fichier

```
1 infos {
     categorie sport { basket };
     categorie technologie { smartphone };
 5
   publication "a_la_une" {
     type nouvelles, depeche;
     fichier "/donnees/nouvelles/lundi.txt";
     fichier "/donnees/depeche/national.log"
10|};
 En analysant le fichier ci-dessus, on peut définir les catégories de mots-clés (tokens) suivants :
    MOT: 'infos', 'categorie', 'publication', 'type', etc
    NOM FICHER: '/donnees/depeche/national.log'
    CARACTERE SPECIAL: "/' et "."
   ACCOLADE_GCHE:{
   ACCOLADE DTE:}
   POINT_VIR: ';'
   VIRGULE: ','
    DOUBLEQUOTE: ""
```

Attention

Il faut faire attention à ce que les différents tokens soient suffisamment différentiables.



Analyseur lexical: exemple d'un format de fichier

```
1 8 {
 2 #include <stdio.h>
 3 % }
 5 88
  [a-zA-Z][a-zA-Z0-9]* printf("MOT\n");
  [a-zA-Z0-9/-\]+
                         printf("NOM_FICHIER\n");
                          printf("DOUBLEQUOTE\n");
 9 \ {
                          printf("ACCOLADE GCHE\n");
10 \ }
                          printf("ACCOLADE_DTE\n");
11
                          printf("POINT VIR\n");
12|\n
                          printf("\n"); /* retour à la ligne*/
13|[\t]+
                           /* ignore espaces et tabulation*/
14 88
```

Remarques:

- ▶ La différence entre un MOT et un NOM_FICHIER tient à la présence des caractères '/' dans le texte indiquant un chemin d'accès à un fichier.
- ▶ Pour utiliser un caractère réservé comme « { » on le précède du caractère d'échappement « \backslash».

Analyseur lexical: exemples

```
1 /* -- inversecasse.lex --
2 Ce programme inverse la casse de toutes les lettres */
3 
4 #include <ctype.h>
5 %%
6 [a-z] printf("%c", toupper(yytext[0]));
7 [A-Z] printf("%c", tolower(yytext[0]));
```

Un second exemple

```
/* -- par.lex --
   Ce programme compte le nb de parentheses ouvrantes et
   verifie si le nb de parenthese fermante correspond. */
   int nbParOuv=0; /* Variable globale du nb de par ouvrantes */
 6 88
  " ("
           {nbParOuv++; ECHO;}
          { if (nbParOuv>0)
  \)
10
               nbParOuv--;
11
               ECHO; /* affiche vytext */
12
13
        else
14
            printf ("Votre parenthesage est incorrect\n");
15
```



















Analyseur lexical: utilisation

* Lancement de l'analyseur lexical:

```
1 yylex(); /* retourne 0 si plus de lexème */
```

- * Redéfinition des fichiers d'entrée et de sortie :
 - yyin: désigne le descripteur du fichier d'entrée, par défaut stdin;
 - yyout: désigne le descripteur du fichier de sortie, par défaut stdout.

Il est possible de les redéfinir:

```
1 yyin = fopen("mon_entree", "r");
2 yyout = fopen("ma_sortie", "w");
```

* Recul dans le flux d'entrée :

```
1 yyless(3)
```

Supprime les 3 derniers caractères de yytext. Les caractères supprimées seront fournis pour la reconnaissance du lexème suivant.

* Fusion avec l'unité lexicale suivante :

```
1 yymore()
```

Permet de fusionner yytext avec l'unité lexicale reconnue précédemment.

* Action exécutée lors de l'arrivée à la fin de l'entrée courante :

```
1 yywrap()
```

Cette fonction doit renvoyer 1 si la fin est effective, ou bien zéro dans le cas contraire (ouverture d'un nouveau fichier par exemple pour redéfinir l'entrée courante).



Remarques sur l'utilisation de Lex

La gestion du «\n»

Lex gère le «retour à la ligne» d'une manière spéciale :

- le caractère «\n» ne s'associe pas avec le «.»:
 - il sert, par défaut, de terminaison à l'appariement de motif («pattern matching»);
- il peut être intégré dans une expression régulière uniquement par son expression explicite :

```
1 \n /* ignorer le retour à la ligne */
```

 il faut faire attention à son intégration pour ne pas «traiter» le fichier d'entrée d'un seul coup avec une seule expression régulière!

La gestion de la fin de fichier «End-Of-File»

Ce symbole ne peut pas être utilisé au sein d'une expression régulière : il doit être utilisé seul pour définir une règle associé à son traitement.

Si l'on veut relancer le travail de l'analyseur sur un contenu différent (par exemple, à partir d'un nouveau fichier d'entrée), il faut regarder le travail de la fonction yywrap.

Attention

La gestion explicite de l'EOF ne permet plus de quitter l'analyseur: il faut utiliser la fonction yyterminate pour le terminer.



Lex: la gestion des accents

Gestion des caractères accentués sous Unix: utilisation du codage UTF-8

Le codage UTF-8, «Universal Coded Character Set + Transformation Format – 8-bit» permet de conserver un codage sur un octet pour les caractères latins non accentués et de passer sur un codage sur deux, voire trois ou quatre octets pour des caractères non latins ou des idéogrammes.

Le caractère «e» est codé avec la valeur de l'octet 65.

Le caractère «é» est codé sur deux octets.

L'idéogramme « E » est codé sur 3 octets. Les deux symboles signifient «ordinateur».

Le symbole «€», euro, est également sur 3 octets :

```
      → xterm

      $ echo -n "€" | xxd

      00000000: e282 ac
      ...
```



Lex: la gestion des accents

Solution ancienne : passage à un codage étendu sur 8 bits au lieu de 7bits

- - Il ne permet pas de coder le symbole de l'euro «€», ni le «œ».
- ▷ le codage «Windows-1252»: permet de coder l'ensemble des symboles du français ainsi que le symbole de l'euro.

Solution actuelle : utilisation du codage UTF8 et Consommation mémoire

- un **caractère non accentué** dont la valeur associée suivant le code ASCII ou ANSI est entre 32 et 127 (soient 7 bits) : codé sur 1 octet ;
- □ un caractère accentué : codé sur 2 octets ;
- □ un caractère non latin ou un idéogramme : 2,3 ou 4 octets (6 au maximum).
- ⇒L'utilisation de caractères accentués change la taille des données et complique la tâche de l'analyseur lexical.

Il est possible de convertir le texte d'UTF-8 vers le codage 1252 à l'aide de la commande iconv:

La conversion inverse est également possible (elle est obligatoire pour un affichage correct dans un environnement Unix configuré pour traiter de l'UTF8)



Pour décrire la syntaxe d'un langage de programmation, on utilise une grammaire.

Une grammaire, *hors contexte*, G définie par $(V_T \cup \{\$\}, V_N, S, P)$, où :

- $\circ V_T$ est l'ensemble des symboles terminaux ;
- \circ V_N est l'ensemble des symboles non terminaux;
- \circ S est l'axiome, c-à-d l'élément de départ de la grammaire ;
- \circ P est l'ensemble des règles de production (hors contexte \to un seul non terminal à gauche pour chaque règle);
- \$ désigne la fin de la chaîne à analyser;
- $V = V_T \cup V_N$ est le «vocabulaire» de la grammaire.

Exemple: Soit G définie par:

$$V_T = \{id, +, *, \}, (\};$$

$$\circ$$
 $V_N = \{E, T, F\};$

$$\circ \quad S = E;$$

$$\circ \quad P = \{E \rightarrow E + T, E \rightarrow T, T \rightarrow T * F, T \rightarrow F, F \rightarrow id, F \rightarrow (E)\}$$

Soit p, id * id + id, une séquence de terminaux.

Comment déterminer si p est correcte, c-à-d si p est acceptée par G?



























On peut utiliser la méthode suivante (analyse ascendante):

- 1) on lit les symboles terminaux de p les uns après les autres ;
- 2) lorsqu'on a lu une séquence α de terminaux qui peut constituer la partie droite d'une règle $A \to \alpha$ alors il est possible de remplacer α par A avant de continuer à lire la suite des terminaux de p. Ce remplacement s'appelle une **réduction**;
- 3) durant la lecture de p, après avoir fait d'éventuelles **réductions**, la séquence de terminaux qui a été lue depuis le début a été transformée en une séquence γ de terminaux et de non-terminaux.
- 4) lorsqu'une partie α de γ peut être réduite par un non-terminal A à l'aide d'une règle $A \to \alpha$, alors il est possible de faire une **réduction** avant de continuer à lire les terminaux de p.
- 5) p est correcte si après sa lecture complète, il est possible de la réduire en S.

Sémantique

Analyse syntaxique &

























Rappels sur les grammaires

Sur l'exemple, avec $P = \{$

- 1. $E \rightarrow E + T$
- 2. $E \rightarrow T$
- 3. $T \rightarrow T * F$

- 4. $T \rightarrow F$
- 5. $F \rightarrow id$
- 6. $F \to (E)$

Ce qui donne:

- par (5): $id * id + id \leftarrow F * id + id$
- par (4) $F*id+id \leftarrow T*id+id$
- par (5) $T*id+id \leftarrow T*F+id$
- par (3) $T * F + id \leftarrow T + id$
- par (2) $T + id \leftarrow E + id$
- par (5) $E + id \leftarrow E + F$
- par (4) $E + F \leftarrow E + T$
- $E + T \leftarrow E$ par (1)

On a procédé à des réductions à droite, c-à-d que les règles utilisées correspondent à des dérivations à droite.

On peut représenter les différentes réductions par un tableau.







































28

Rappels sur les grammaires

Explications:

- Colonne de gauche : représente le contenu courant de γ stocké dans une pile ;
- Colonne du milieu: représente la partie de p qui n'a pas été encore traitée;
- Colonne de droite : indique l'opération réalisée pour passer à la ligne suivante :
 - \diamond réduction d'une partie droite de γ , dans ce cas on précise la règle utilisée;
 - lecture du prochain terminal.

Pile	Partie de p non traitée	Règle appliquée (si réduction)
	id*id+id	lecture
id	*id + id	$F \rightarrow id$
F	*id + id	$T \to F$
T	*id + id	lecture
T*	id + id	lecture
T*id	+id	$F \rightarrow id$
T * F	+id	$T \to T * F$
T	+id	$E \to T$
E	+id	lecture
E+	id	lecture
E + id		$F \rightarrow id$
E + F		$T \to F$
E + T		$E \to E + T$
E		

































Rappels sur les grammaires

Utilisation de la pile

Sur le tableau précédent, la chaîne courante γ se trouve dans une pile :

- * l'extrémité gauche de γ est en bas de la pile;
- * l'extrémité droite de γ est en haut de la pile.

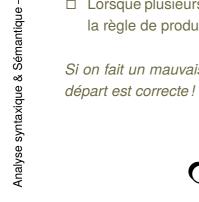
Cette approche est pratique:

- \triangleright une réduction s'effectue sur une partie droite de γ , ce qui se traduit par une modification d'une partie du sommet de la pile;
- \triangleright une lecture ajoute un symbole terminal à droite de γ , ce qui se traduit par l'ajout d'un symbole sur le sommet de la pile.

Problème:

- □ Lorsqu'il est possible de réaliser une réduction, il faut décider s'il faut effectuer cette réduction avant de continuer la lecture.
- □ Lorsque plusieurs réductions sont possibles et que l'on décide de réaliser une réduction, il faut choisir la règle de production à utiliser.

Si on fait un mauvais choix, on risque de ne pas aboutir au symbole de départ, même si la chaîne de départ est correcte!



























Rappels sur les grammaires

30

Exemple de problème :

Sur l'exemple précédent :

Pile	Partie de p non traitée	Règle appliquée (si réduction)
	id*id+id	lecture
id	*id + id	$F \rightarrow id$
F	*id + id	$T \to F$
\overline{T}	*id + id	lecture

La pile contient le symbole non terminal $\mathbb{T},$ on peut :

- □ choisir de continuer la lecture (*lecture* de *), ce qui a été fait ;
- $\hfill\Box$ choisir de réduire par l'utilisation de la règle $E\to T$, ce qui aurait donné :

F	*id + id	$T \to F$
T	*id + id	$E \to T$
E	*id + id	lecture
E*	id + id	lecture
E*id	+id	$F \rightarrow id$
E * F	+id	$T \to T * F$

Bloqué!

Analyse syntaxique & Sémantique

































Ce sont des analyseurs syntaxiques fonctionnant de manière ascendante :

- ▷ R signifie qu'on effectue une séquence de réductions à droite, «right».

Une séquence de réductions est dite «à droite» si elle consiste à parcourir une séquence de dérivations à droite en sens inverse.

Exemple:

* Une séquence de dérivations à droite :

$$E \rightarrow E + T \rightarrow E + F \rightarrow E + id \rightarrow T + id \rightarrow F + id \rightarrow id + id$$

* La séquence de **réductions à droite** correspondante :

$$id + id \leftarrow F + id \leftarrow T + id \leftarrow E + id \leftarrow E + F \leftarrow E + T \leftarrow E$$

























Les différentes catégories d'analyseurs LR

Il existe plusieurs catégories d'analyseurs LR, les plus connus sont :

- SLR(1), LR(1), LALR(1);
 - LR veut dire «Left to right, Rightmost derivation»;
 - ♦ SLR veut dire «Simple LR»
 - ♦ LALR veut dire «Look-Ahead LR» (avec anticipation);
- le paramètre 1 indique qu'à tout moment le parser connaît le premier symbole de la partie non encore traitée du texte à analyser;
- les analyseurs SLR(1) sont les plus restrictifs;
- o un analyseur SLR(1) est un analyseur LALR(1) particulier;
- un analyseur LALR(1) est un analyseur LR(1) particulier (produit par YACC).

Ce qui donne : $LR(0) \subset SLR(1) \subset LALR(1) \subset LR(1)$

Les analyseurs LR(1) sont les **plus généraux** et peuvent s'appliquer à un **plus grand nombre** de grammaires.































Principe de fonctionnement d'un analyseur LR

Le fonctionnement des **différents analyseurs** repose sur l'utilisation d'une **table** composée de deux parties :

- * une partie «action»;
- * une partie «branchement» (à la manière d'un goto).

Cette table de transitions d'états correspond à une écriture particulière d'un automate à nombre fini d'états.

La table contient aussi des informations supplémentaires indiquant à l'analyseur quand il faut effectuer :

- une réduction, «Reduce»;
- une **lecture**, «Shift».

Analyse syntaxique & Sémantique

Les trois types d'analyseurs, SLR(1), LR(1) et LALR(1), sont :

























Soit G définie par :

$$V_T = \{id, +, *, \}, (\}; \quad \circ \quad P = \{0\}$$

$$P = \{$$

3.
$$T \rightarrow T * F$$
 6. $F \rightarrow (E)$

6.
$$F \rightarrow (E)$$

$$\begin{array}{lll} \circ & V_N = \{E,T,F\}\,; & \text{1. } E \to E + T & \text{4. } T \to F \\ \circ & S = E\,; & \text{2. } E \to T & \text{5. } F \to id \end{array}$$

1.
$$E \rightarrow E + T$$

2. $E \rightarrow T$

5.
$$F \rightarrow id$$

La table pour un analyseur SLR(1):

État	+	*	id	()	\$	E	T	F
0			s5	s4			1	2	3
1	s6					OK			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4			s5	s4			8	2	3
5	r5	r5			r5	r5			
6			s5	s4				9	3
7			s5	s4					10
8	s6				s11				
9	r1	s7			r1	r1			
10	r3	r3			r3	r3			
11	r6	r6			r6	r6			































Explications:

- ▷ ligne de haut: symboles terminaux et non-terminaux de la grammaire Le symbole (terminal) \$ est ajouté à la fin du texte à analyser;
- partie centrale, sous les symboles terminaux : Partie action
 - lettre s signifie qu'il faut effectuer une opération «shift» (lecture)
 - ♦ lettre r signifie qu'il faut effectuer une opération «reduce» (réduction)
- ▶ Partie de droite sous les symboles non-terminaux : Partie branchement (goto)





Exemple de table

36

Algorithme LR:

- > Initialisation:
 - symbole \$ mis à la fin du texte à analyser;
 - état 0 (de départ) est empilé;

 prochain_symbole_terminal_non_traité := premier_symbole_du_texte_à_analyser;

> Répéter

- ♦ j := état_au_sommet_de_la_pile; /*état courant de l'analyseur */
- a := prochain_symbole_terminal_non_traité;
- \diamond si (table[j, a] = sm) alors
 - \star symbole a et état m sont empilés;
 - * prochain_symbole_terminal_non_traité := symbole_qui_suit(a);
- \diamond si (table[j, a] = rm) alors
 - * B := partie_gauche_de_la_règle_de_production(m); /* B = symbole non-terminal */
 - * L := longueur_de_la_partie_droite_de_la_règle_de_production(m)
 - * on dépile 2L éléments de la pile; /* L paires (symbole, état) */
 - * k := état_au_sommet_de_la_pile; /* après avoir dépilé */
 - * symbole B et état table[k,B] sont empilés;
- \diamond si (table[j, a] = OK)
 - * alors texte_à_analyser est acceptée
 - * sinon erreur détectée
- ▷ Jusqu'à ce que texte_à_analyser est acceptée ou erreur_détectée.































COMPLÉMENTNT



37

Exemple de table

Si on utilise l'algorithme sur le texte à analyser id * id + id:

	_	-
Pile	texte restant à traiter	Commentaires
vide	id*id+id	symbole \$ rajouté à la fin du texte;0 est empilé;
0	id * id + id\$	table[0,id] = s5 donc: - on empile id et 5; - on avance au symbole suivant (shift);
0 <i>id</i> 5	*id + id\$	$table[5,*]=r5$ et règle(5) $F \rightarrow id$ donc: - on dépile 2 symboles (5 et id); - on empile F et 3 (car $table[0,F]=3$);
0 <i>F</i> 3	*id + id\$	$table[3,*] = r4$ et règle(4) $T \to F$ donc : - on dépile 2 symboles (3 et F); - on empile T et 2 (car $table[0,T]=2$);
0 T 2	*id + id\$	<pre>table[2,*] = s7 donc:</pre>
0 T 2 * 7	<i>id</i> + <i>id</i> \$	table[7,id] = s5 donc: - on empile id et 5; - on avance au symbole suivant (shift);

























CCOMPLÉMENTINT



38

Exemple de table

Pile	texte restant à traiter	Commentaires
0 T 2 * 7 id 5	+id\$	$table[5,+]=r5$ et règle(5) $F \rightarrow id$ donc: - on dépile 2 symboles (5 et id); - on empile F et 10 (car $table[7,F]=10$);
0 T 2 * 7 F 10	+id\$	$table[10,+] = r3 \text{ et règle(3) } T \rightarrow T*F \text{ donc:} \\ - \text{ on dépile 6 symboles (10, } F,7,*,2 \text{ et } T); \\ - \text{ on empile } T \text{ et 2 (car } table[0,T]=2); \\$
0 T 2	+id\$	$table[2,+]=r2$ et règle(2) $E \to T$ donc: - on dépile 2 symboles (2 et T); - on empile E et 1 (car $table[0,E]=1$);
0 <i>E</i> 1	+id\$	<pre>table[1,+] = s6 donc:</pre>
0 E 1 + 6	id\$	table[6, id] = s5 donc: - on empile id et 5; - on avance au symbole suivant (shift);
0 E 1 + 6 id 5	\$	$table[5,\$] = r5$ et règle(5) $F \rightarrow id$ donc: - on dépile 2 symboles (5 et id); - on empile F et 3 (car $table[6,F]=3$);

























Exemple de table

Pile	texte restant à traiter	Commentaires
0 E 1 + 6 F 3	\$	$table[3,\$] = r4$ et règle(4) $T \to F$ donc: - on dépile 2 symboles (3 et F); - on empile T et 9 (car $table[6,T]=9$);
0 E 1 + 6 T 9	\$	$table[9,\$] = r1$ et règle(1) $E \rightarrow E + T$ donc: - on dépile 6 symboles (9, T , 6, +, 1 et E); - on empile E et 1 (car $table[0,E]=1$);
0 <i>E</i> 1	\$	table[1,\$] = Ok donc texte correct!

Et on a fini!

Interprétation intuitive d'une table

40

Construire une mémoire de l'analyseur :

▶ Passé courant d'un analyseur : séquence de «shifts» et de «reduces» effectués

Exemple: texte à analyser id * id + id

Après lecture de id *, on a affectué:

- shift (lecture de id)
- reduce (règle 5)
- reduce (règle 4)
- shift (lecture de *)

définit la mémoire après lecture de id *

- État courant d'un analyseur:
 - informe sur le passé courant du parseur;
 - n'informe pas forcément sur tout le passé ;

Par exemple, l'état peut nous informer sur la séquence de shifts et reduces , sans préciser les numéros des règles utilisés

On observe que:

- ♦ les états générés par un analyseur LR(1) sont les plus précis et donc les plus nombreux;
- ♦ les **états générés** par un analyseur SLR(1) sont les **moins précis** et donc les **moins nombreux**.

























COMPLÉMENTINT



Conflits 41

Lorsqu'un **analyseur** est **utilisé** pour une grammaire pour laquelle il n'est **pas applicable**, alors des **conflits** peuvent avoir lieu.

C'est par exemple le cas si :

- * un analyseur SLR(1) est utilisé pour une grammaire LR(1) ou LALR(1);
- * un analyseur LALR(1) est utilisé pour une grammaire LR(1);
- * un analyseur LR est utilisé pour une grammaire qui n'est pas LR (par exemple, une grammaire ambiguë n'est pas LR);

Un **conflit** est détecté lors de la construction de la table de l'analyseur, à chaque fois que dans une même position de la partie action de la table, on obtient:

- * deux actions si et rj: on dit qu'il y a **conflit** "**shift**"/(***reduce**") \Rightarrow l'analyseur ne sait pas s'il doit effectuer un "**reduce**" à l'aide de la règle (j) ou bien un "shift";
- \star deux actions ri et rj: on dit qu'il y a **conflit** «**reduce**»/«**reduce**» \Rightarrow l'analyseur sait qu'il doit effectuer un «**reduce**», mais il ne sait pas s'il doit appliquer la règle i ou la règle j



























Exemple de conflit

La gestion du «if-then-else»

Soient les règles :

- 1. $Si_Inst \rightarrow IF\ Expr\ THEN\ Si_Inst$
- 2. $Si_Inst \rightarrow IF\ Expr\ THEN\ Si_Inst\ ELSE\ Si_Inst$

Soit la phrase IF *Exp*1 THEN IF *Exp*2 THEN *Inst*1 ELSE *Inst*2

Lorsque l'analyseur arrive à ELSE, il ne sait pas s'il doit effectuer :

- □ une réduction de IF Exp2 THEN Inst1 ⇒ cela aura pour effet d'associer le ELSE au premier IF
- ightharpoonup un **décalage** sur ELSE \Rightarrow cela aura pour effet d'associer le ELSE au second IF

Bien que tout le passé soit connu, il n'est pas possible de décider.

Ceci est dû au fait que la grammaire est ambigüe.

▶ Première approche :

second IF

- □ En cas de conflit si/rj, on peut par exemple décider de ne garder que si
 C'est l'approche utilisée par YACC.
 Dans le cas du conflit shift/reduce du lf-then-else, YACC décidera donc d'associer le ELSE au
- \square En cas de conflit ri/rj, on peut par exemple décider de garder la réduction ayant le plus petit numéro de règle (Exemple : pour r4/r3, on garde r3) C'est l'approche utilisée par YACC.
- ▶ Seconde approche : elle consiste à modifier les règles de production de la grammaire On reviendra sur ce point lorsqu'on étudiera YACC.

5 Analyseur lexical: un avant goût de grammaire?

Il est possible de modifier le comportement de l'analyseur lexical en fonction du «contexte» situé à gauche.

Utilisation de «commutation entre contextes»

On veut interpréter l'expression régulière [a-zA-Z]+ de deux manières, selon le **contexte**:

- 1. si mot entre guillemet, alors il représente une chaîne de caractères;
- 2. si mot pas entre guillemet, alors c'est un identificateur;

On définit deux états CHAINE et NORMAL, pour les deux contextes.

Ces états sont utilisés comme suit :

- au début, l'état est mis à NORMAL par l'instruction BEGIN NORMAL;
- chaque fois qu'on rencontre un guillemet, on *commute* d'un contexte à l'autre (entre NORMAL et CHAINE).

Dans le source Lex, on **préfixe** les expressions régulières par un **contexte d'activation**.

Ainsi, lorsqu'on reconnaît une expression régulière [a-zA-Z]+:

- → au départ l'état est INITIAL, qui correspond à l'état initial;
- ▷ si l'état courant est NORMAL : alors c'est un identificateur qui est reconnu ;
- □ un état sans contexte correspond à l'état 0.

Toute règle sans condition est active tout le temps.

Une règle préfixée avec le contexte <INITIAL> est active au début du travail de l'analyseur.



Analyseur lexical: un avant goût de grammaire?

Soit le programme suivant :

```
1 %start NORMAL CHAINE
 2 | %%
 3 < NORMAL>[a-zA-Z]+ printf("Reconnaissance d'un identificateur: %s\n", yytext);
 4 < NORMAL>\"
                   BEGIN CHAINE:
 5 < CHAINE > [a-zA-Z] + printf("Reconnaissance d'une chaine : %s\n", yytext);
 6 < CHAINE > \"
              BEGIN NORMAL;
 7 < NORMAL, CHAINE>. /* aucune action */
 8 < NORMAL, CHAINE > \n /* aucune action */
 9 88
10 void main()
11
   yyin = fopen("entree.txt", "r");
13 BEGIN NORMAL; /* On definit le mode par defaut */
   yylex(); /* yylex appelé une seule fois car pas de return */
15
    fclose(yyin);
16|}
```

À la ligne 7 et 8, avec < NORMAL, CHAINE >, on définit une règle Lex valable dans les deux contextes.











Analyseur lexical: un avant goût de grammaire?

Gestion des règles ambigües dans Lex:

```
1 integer /* reconnaissance d'un mot clé */
2 [a-z]+ /* reconnaissance d'un identifiant */
```

Ces deux règles sont **ambigües** car elles peuvent s'appliquer **simultanément** sur un même contenu.

Lex applique les règles de sélection suivantes :

- 1. **sélection** de l'expression régulière qui donne la **plus longue** correspondance ;
- 2. si plusieures expressions régulières donnent la même taille, sélection de la première par ordre de saisie.

Dans le cas où le texte est integers alors c'est la ligne 2 du fichier Lex qui est sélectionnée.

Attention : la règle de sélection 1 peut entraîner des erreurs involontaires :

```
1 '.*' /* reconnait une chaine */
```

Si on donne l'entrée 'premiere' chaine d'abord, 'seconde' ensuite alors Lex va retourner: premiere' chaine d'abord, 'seconde.

Il faut alors réécrire en :

```
1|'[^{'}]^{*} /* reconnait une chaine */
```

Ce qui évite le problème.



Lex: un avant goût de grammaire

Il est possible d'utiliser des règles Lex ambigües, à l'aide de l'opérateur REJECT.

Attention : la commande yyless(0) ne passe pas à la règle suivante comme REJECT et doit s'accompagner d'un changement de contexte.

Cette opérateur permet d'aller vers la prochaine règle possible :

```
1 anticonstitutionnel occurrence++; REJECT;
2 constitutionnel occurrence++;
```

lci, on va compter à la fois le mot constitutionel avec la première règle et avec la seconde.

Pour faire de la **cryptanalyse fréquentielle**, on peut chercher le nombre de **digrammes**, groupe de deux lettres, présents dans un texte :

















6 Liens entre Lex et un analyseur syntaxique

Pour utiliser un **analyseur lexical**, construit par Lex, dans un **analyseur syntaxique**, il faut pouvoir fractionner le travail de Lex à la reconnaissance d'un seul lexème à la fois.

L'analyseur syntaxique, ou parser, va effectuer un appel à la fonction yylex () pour le traitement de chaque lexème.

Dans ce cas d'utilisation, l'action associée à la reconnaissance d'un lexème :

- retourne la nature du lexème reconnu;
- affecte la valeur du lexème à une variable globale commune avec le parser.

Il est nécessaire de :

- a. définir une valeur de retour pour chaque action définie dans le source de Lex;
- b. **partager** cette définition avec l'outil utilisant Lex.

La solution est d'utiliser un fichier d'entête commun, par exemple nommé «mes_lexemes.h», à inclure dans le fichier source Lex et dans l'analyseur syntaxique:

```
1 #define ENTIER 1
2 #define QUOTE 2
3 #define MULTIPLIER 3
4 ...
```

lci, on associe pour chaque lexème une valeur entière que l'on retourne dans l'action.



Liens entre Lex et un analyseur syntaxique

Le fichier d'entête est ensuite inclus dans le source Lex:

```
1 8 {
     #include "mes lexemes.h"
 3 | % }
 4|DIGIT ([0-9])
 5 88
  {DIGIT}+ { mon_lexeme = atoi(yytext);
         #ifdef DEBUG
 8
           fprintf(stderr, "Entier :%d\n", mon lexeme);
 9
         #endif
10
         return ENTIER;
11|}
```

Pour la variable recevant la valeur du lexème on peut utiliser la définition

«symbole mon_lexeme» avec:

```
1 typedef union
                                         1 typedef struct
2 {
  int entier;
                                            int type;
   char *chaine;
                                            type_lexeme val;
   type_lexeme;
                                            symbole;
```

L'union permet de considérer la même valeur soit Le struct permet d'associer un entier et un pointeur comme un entier ou soit comme un pointeur sur sur une chaîne. une chaîne.



YACC, «Yet Another Compiler Compiler», est un programme permettant:

- ▷ d'effectuer des actions sémantiques liées à cette grammaire.

La syntaxe des fichiers, d'extension «. y», qu'il accepte est proche de celle de Lex:

```
1 %}
2 prologue
3 %}
4 Déclaration
5 %%
6 Règles de productions
7 %%
8 Epilogue
```

Le fichier est traité de la façon suivante :

```
$ bison mon_fichier.y
```

Ce qui produit le fichier y . tab . c contenant le source C de l'analyseur syntaxique.

Dans ce fichier est définie la fonction yyparse() qui réalise l'analyse syntaxique en utilisant la fonction yylex() fournie par Lex.



La partie «prologue»:

- * des déclarations et définitions C;
- * des «#include»;
- * des définitions de variables qui seront globales à tout l'analyseur syntaxique.

La partie «Déclaration»:

- * définie le type de la variable yylval qui sera partagée avec Lex pour récupérer la valeur d'un lexème:
 - par défaut il est de type int;
 - peut être redéfini dans la partie «prologue» :

```
1 #define YYSTYPE nom-de-type
```

Dans ce cas là, il faut aussi le mettre dans la partie «Épilogue» du fichier Lex.

peut être redéfini sous forme d'union pour définir le vocabulaire de la grammaire :

♦ ou sous forme de struct:



La partie «Déclaration», suite:

Lorsque YACC reconnaît un symbole non terminal il utilise la variable yyval pour le communiquer au programme l'utilisant.

Cette variable est définie par la même définition que yylval avec le %union.

- * définition des **symboles terminaux** à l'aide de %token; Lorsque le terminal n'a pas de valeur ou possède une valeur entière, il n'est pas nécessaire de spécifier le type, car il est entier par défaut.
- * définition des symboles non terminaux à l'aide de %type.
 Lorsqu'un non terminal n'a pas de valeur ou une valeur entière, il n'est pas nécessaire de le déclarer.
 Les types spécifiés par %token et %type ont été définis par %union.
- * des informations sur l'associativité et la précédence des opérateurs à l'aide de %left %right noassoc et prec:
 - 1 %left PLUS MOINS 2 %left MULT DIV

 - une précédence identique pour MULT et DIV;
 - ⋄ une précédence de MULT et DIV supérieure à celle de PLUS et MOINS (définition sur une seconde ligne).
- * la déclaration du symbole non terminal de départ avec %start.



L'analyseur syntaxique YACC : sur un exemple

Sur l'exemple de la grammaire précédente : Soit G définie par :

- $V_T = \{nb, +, *, \}, \{\}; \quad \circ \quad P = \{\}$
- 3. $T \to T * F$ 6. $F \to (E)$

- les terminaux *, +, (,) n'ont pas besoin de valeur associée;
- \circ le terminal nb correspond à un nombre entier.

On peut utiliser la déclaration suivante pour «yylval»:

```
1 %union
   { int *valeur; /*pointeur sur la valeur d'un terminal */
         int type; /* nature d'un non-terminal */
```

Ce qui va donner:

```
1 %token <valeur> NB MULT ADD PAROUV PARFER
2 %type <type> E T F
3 %left ADD
4 %left MULT
5 %start E
```

Ainsi l'expression c * d + e + f * g est interprétée comme ((c * d) + e) + (f * g)













La partie «Règles de productions»

Soit n règles de production ayant le même non-terminale en partie gauche :

```
Non_terminal : corps_1 { actions_1}
| corps_2 { actions_2}
| ...
| corps_n { actions_n}
| ;
```

Chaque $corps_i$ correspond à la partie droite de la règle R_i .

La partie *actions*; permet de manipuler les valeurs des terminaux et non-terminaux :

- □ soit la règle A : U1 U2 ... UN {actions}, où:
 - ♦ A est un symbole non terminal;
 - \diamond Ui est un symbole terminal ou non-terminal, avec i = 1..n
- ☐ dans la partie actions, on peut utiliser les symboles suivants:
 - ♦ \$\$ pour se référer à la valeur de A;
 - \$i pour se référer à la valeur de Ui.
- □ lorsqu'aucune action n'est précisée, YACC génère l'action \$\$=\$1;

Lorsque la **réduction** correspondante est effectuée, alors la variable globale yyval reçoit implicitement la valeur \$\$.

















L'analyseur syntaxique YACC : sur un exemple

Sur la grammaire précédente :

Explications:

- ▷ la valeur d'un symbole terminal est un pointeur sur la valeur du nombre ;
- ⊳ la valeur d'un non-terminal A est la **valeur** de l'expression qui correspond à A.

La partie «Epilogue» contient du code C:

- ▷ le programme principal qui fait appel à l'analyseur syntaxique (c-à-d à la fonction yyparse ()).

Exemple:

```
void main () {
  if ( yyparse() == 0 ) printf("résultat = %d \n", yyval.nbre);
  else printf("Erreur de syntaxe \n")
4 }
```

Pour réaliser l'association avec Lex

- a. dans la partie «prologue» de lex, on inclus un fichier d'en-tête que va produire YACC pour définir le vocabulaire de la grammaire, les «tokens» (par ex. «defs_a_inclure.h»);
- b. on inclus le fichier «global.h» dans la partie «prologue» de Lex et de YACC:

```
1 #define YYSTYPE mon_type
2 extern YYSTYPE yylval;
```

c. on demande à YACC (ou ici bison) de générer le fichier defs a inclure.h:













Lex & YACC: un exemple de calculette

On va créer un **outil de calcul interactif** permettant de faire des calculs d'expression mathématique contenant des réels et utilisant les opérateurs de puissance, multiplication, division, soustraction et addition.

- Soient les fichiers analyse_lexical.l et analyse_syntaxique.y dont le contenu est donné dans les transparents suivants;
- ▶ le fichier global.h qui va définir le type de yylval pour échanger entre Lex et YACC:

```
1 #define YYSTYPE double
2 extern YYSTYPE yylval;
```

▷ la procédure de compilation et un exemple d'utilisation :

```
$ lex analyseur.l
$ bison -d analyseur.y --graph=desc.txt
analyseur.y: conflits: 10 décalage/réduction
$ cp analyseur.tab.h calc.h
$ gcc -o Mon_analyseur analyseur.tab.c lex.yy.c -lm -lfl
$ ./Mon_analyseur
7*4
Resultat: 28.00000
```



Lex & YACC: un exemple de calculette

```
1 | % {
                                               Les différents «tokens» sont PLUS, MOINS, PARGAUCHE,
 2 #include "global.h"
                                               etc
 3 #include "calc.h"
 4 #include <stdlib.h>
 5 | % }
 6 blancs
              [\t]+
 7 chiffre [0-9]
 8 entier
           {chiffre}+
 9|exposant [eE][+-]?{entier}
10 reel
           {entier}("."{entier})?{exposant}?
11 | %%
12 {blancs} /* On ignore */
13 {reel}
              { yylval=atof(yytext);
14
           return(NOMBRE); }
15 "+"
         return (PLUS);
16 "-"
         return (MOINS);
17 " * "
         return (FOIS);
18 "/"
         return (DIVISE);
19 "^"
         return (PUISSANCE);
20 " ("
         return (PARGAUCHE);
21 ") "
         return (PARDROITE);
22 "\n"
         return (FIN);
```

Ces tokens sont définis dans le fichier «analyse_syntaxique.tab.h» qui est généré automatiquement par yacc lors du traitement du fichier de l'analyse sémantique et qui est ensuite renommé en «calc.h».

Lex & YACC: un exemple de calculette

```
1
  응 {
                                             15 Input: /* Vide */
    #include "global.h"
                                             16
                                                  | Input Ligne
    #include <stdio.h>
  응 }
                                             18
                                                Ligne:FIN
  %token NOMBRE
                                                     Expression FIN {printf("Resultat:%f\n", $1);}
                                             20
   %token PLUS
                 MOINS FOIS
                                     PUISS
                                             21 Expression: NOMBRE
   %token PARGAUCHE PARDROITE
                                                                         {$$=$1;}
  %token FIN
                                                    Expression PLUS Expression {$$=$1+$3;}
   %left PLUS
                                             23
                                                    Expression MOINS Expression {$$=$1-$3;}
               MOINS
10 %left FOIS
                                             24
                                                  | Expression FOIS Expression {$$=$1*$3;}
               DIVISE
11 %left NEG
                                             25
                                                   | Expression DIVISE Expression {$$=$1/$3;}
                                             26
12 %right PUISSANCE
                                                   | MOINS Expression %prec NEG
                                                                                {$$=-$2;}
13 %start Input
                                             27
                                                    Expression PUISS Expression {$\$=pow(\$1,\$3);}
                                             28
14 88
                                                   | PARGAUCHE Expression PARDROITE { $$=$2;}
```

Vous noterez que lors de la génération de l'analyseur syn-29 ; taxique, il y a un conflit «décalage/réduction» sur la règle 10.31 int yyerror(char *s) { printf("%s\n",s); } desc.txt.

Attention

Ne pas oublier d'appeler la fonction yyparse() pour déclencher l'analyseur syntaxique (dans la fonction main).

Lex & Yacc: comment compiler un analyseur complet?

Création d'un Makefile pour automatiser déclarativement la construction de l'analyseur :

```
1 CC=qcc
 2 LDFLAGS=-lfl -lm
 3 EXEC NAME=mon analyseur
 4 OBJETS=syntaxique.o lexical.o
 6 .y.c:
       bison -d $<
       mv $*.tab.c $*.c
       mv $*.tab.h $*.h
10
11.1.c:
12
       flex $<
13
      mv lex.vv.c $*.c
14
15 .C.O:
16
       $(CC) -c $<
18 all: $ (EXEC_NAME)
19
20 $ (EXEC NAME): $ (OBJETS)
21
       $(CC) -o $@ $^ $(LDFLAGS)
2.2
23 clean:
24
         $(OBJETS) $(EXEC NAME)
tabulation
```

Explications:

- ligne 2: on indique les bibliothèques à «linker» pour construire l'exécutable;
- ligne 4 : les objets composant l'exécutable ;
 Remarque : l'objet correspondant à l'analyseur sémantique produit par yacc doit être placé avant celui produit par lex pour la définition des tokens.
- ligne 6: on indique à «make», comment générer un source «C» à partir du fichier yacc «.y»;
- ligne 11: pareil pour un fichier lex «.l»;
- ♦ ligne 15: pour créer un objet, il faut compiler le «.c» associé;
- ligne 18: la règle «all» construit l'exécutable;
- ligne 20 : l'exécutable, «EXEC_NAME» est composé de tous les objets obtenu à partir de leur compilation individuelle.

Attention

Respecter les **décalages** dans le Makefile : vous devez utiliser des **tabulations** sinon votre Makefile ne fonctionnera pas.



Rapport entre Lex et Yacc: la variable yylval

Exemple: reconnaissance de la date où le mois est donné sous forme de texte

```
#include <stdlib.h>
   #include "analyseur_syntaxique.tab.h"
   char *m;
   응 }
            Janvier|Fevrier|Mars|Avril|Mai|Juin|
   mois
   Juillet | Aout | Septembre | Octobre | Novembre | Decembre
 8 %%
   {mois}
                   m=(char *)calloc(yyleng+1,
10
                sizeof(char));
11
                   strcpy(m, yytext);
12
                   yylval.texte=m;
13
                   return(token_MOIS);
14
   [0-9]{1,2} {
                   yylval.valint=atoi(yytext);
16
                   return(token_JOUR);
17
   [0-9]{4}
18
                   yylval.valint=atoi(yytext);
19
                   return(token_ANNEE);
20
                   return ',';
22
```

Dans cet exemple, lex va retourner 3 tokens différents de deux types (entier ou chaîne).

Par défaut :

- la variable partagée entre Lex & Yacc est de type «int»;
- la numérotation des tokens commence à 258 pour permettre à Lex de retourner directement la valeur d'un caractère/octet lu (valeur entre 0 et 255).

On ne va pas utiliser de fichier «global.h» pour définir le type de ces tokens mais directement yacc.

La déclaration associée dans le fichier yacc «analyseur_syntaxique.y»:

```
1 %union { int valint ; char *texte ;}
2 %token <valint> token_JOUR
3 %token <valint> token_ANNEE
4 %token <texte> token_MOIS
```

Dans le cas où un «terminal» peut prendre plusieurs types, il faudra également définir le type des pour les «non terminaux» de la grammaire :

Dans cet exemple, YACC défini automatiquement le fichier «analyseur_syntaxique.tab.h» qui contient:

- la liste des tokens et les valeurs numérique associées que devra retourner Lex;
- la déclaration et le type de yylval.



Un exemple complet et fait automatiquement avec le Makefile

Le fichier «syntaxique.y»:

```
1 %{
     #include Psyntaxique.h"
     #include <stdio.h>
     #include <math.h>
5%}
6%union{ float valeur; }
7 %token | <valeur> NOMBRE
8 %tvpe
          <valeur> Expression
 9 %token PLUS MOINS FOIS DIVISE
10 %token PARGAUCHE PARDROITE
11%token FIN
12 %left PLUS MOINS
13 %left FOIS DIVISE
14%left NEG
15 %right PUISSANCE
16%start Input
17 %%
18 Input: /* Vide */
   | Input Ligne
20
21 Ligne: FIN
   | Expression FIN {printf("Resultat:%f\n",$1);}
23
24 Expression: NOMBRE
                          {$$=$1;}
   | Expression PLUS Expression {$$=$1+$3;}
   | Expression MOINS Expression {$$=$1-$3;}
26
   | Expression FOIS Expression {$$=$1*$3;}
   | Expression DIVISE Expression {$$=$1/$3;}
   | MOINS Expression %prec NEG {$$=-$2;}
   | Expression PUISS Expression {$$=pow($1,$3);}
31
    | PARGAUCHE Expression PARDROITE { $$=$2;}
32
33 88
34 int yyerror(char *s) { printf("%s\n",s); }
35 int main(void) { yyparse(); }
```

Le fichier «lexical.l»:

```
1 % {
     #include syntaxique.h"
     #include <stdlib.h>
 4 %}
 5 blancs
             [\t]+
 6 chiffre
             [0-9]
 7 entier
             {chiffre}+
 8 exposant [eE][+-]?{entier}
           {entier}("."{entier})?{exposant}?
 9 reel
10 %%
11 {blancs} /* Qn ignore */
                Tyylval.valeur = atof(yytext);
12 {reel}
13
            return (NOMBRE); }
14"+"
         return (PLUS);
         return (MOINS);
         return (FOIS);
17 "/"
         return(DIVISE);
18 "^"
         return (PUISSANCE) :
19"("
        return (PARGAUCHE):
20")"
        return (PARDROITE);
21 "\n" return(FIN);
```

- f)⇒on inclus le fichier renommé par le «Makefile» vu précédemment;
- 2⇒on définit l'union;
- 3⇒qui contient le type flottant, que l'on associe aux terminaux (token) et non terminaux (type);
- 4 ⇒ on utilise l'union dans le fichier lex.

Les fichiers sont disponibles à https://git.p-fb.net/PeFClic/lex_et_yacc.git















YACC: gestion des erreurs

Lorsqu'une erreur est rencontrée lors de l'analyse syntaxique, la fonction yyerror () est appelée. Cette fonction doit être définie dans la partie «épilogue» du fichier YACC.

Vous pouvez par exemple affichez un message qui spécifie :

- le numéro de la ligne;
- ▷ le dernier terminal lu.

Si la fonction yyerror () n'a pas été défini alors l'analyse s'arrête simplement.

Poursuite de l'analyse syntaxique au-delà d'une erreur

Le terminal error peut être utilisé dans la grammaire pour **permettre de dépiler** le contenu de la pile de l'analyseur syntaxique jusqu'à ce terminal.

Exemple: $A \rightarrow error\alpha$, où:

 \square A est un non terminal qui correspond à une structure formant un «tout» cohérent dans le langage.

Exemple:

une expression;

⋄ une définition de procédure;

une instruction;

♦ une déclaration de variable, etc.

Il est sûr que si une erreur se produit, cette erreur va entrainer une erreur complète de la structure où elle se produit.

 \square a est une séquence, éventuellement vide, de symboles terminaux ou non terminaux.















Analyse syntaxique & Sémantique — P-FB

Exemple de gestion d'erreur :

```
Programme: error POINTVIRGULE
Instruction POINTVIRGULE
Programme Instruction POINTVIRGULE

...
;
```

Ici, l'erreur entraine le passage à l'instruction suivante, α est POINTVIRGULE ce qui fait que l'on force l'analyse à reprendre après le «;» suivant..

Lorsqu'une erreur se produit l'analyseur est dans un mode spécial, dont il faut sortir pour reprendre le mode normal:

On utilise la macro spéciale yyerrok.

Il faut également prévoir de défaire ce qui avait été fait pendant l'analyse qui a abouti à une erreur (désallocation de mémoire, ré-initialisation, etc.).

8 XML: les différents outils

□ **SGML**, «*Standard Generalized Markup Language*» : développé dans les années 70, chez IBM qui deviendra un standard ISO 8879 en 1986.

But: gestion de documents techniques de plusieurs milliers de pages.

☐ **HTML**: une application de XML (la plus populaire).

But : spécialisé dans l'écriture de pages Web et uniquement, il n'est pas extensible ou adaptable à d'autres utilisations.

☐ XML: février 1998, XML v1.0

But : Bénéficier des avantages de SGML en le simplifiant et en enlevant ce qui ne marchait pas (pas utilisé).

□ XSL, «eXtensible Stylesheet Language»: une application d'XML.

But: Permettre la visualisation d'un document XML dans un navigateur.

- ♦ **XSLT**, «XSL Transformation»: permet de transformer un document XML pour la représentation en Web ou bien dans d'autres contextes.
- XSL-FO, «XSL Formatting Object»: permet de décrire la composition des pages pour l'affichage des pages en Web ou à l'impression.
- □ **CSS**, «Cascading Style Sheet»: utilisé pour la représentation des documents HTML But: Permettre la représentation de documents XML comme HTML à partir de la v2.



XML: les différents outils

□ XLL, «eXtensible Link Language»:

But : permettre de définir des modèles de liaisons pour relier des documents XML dans un réseau HyperTexte.

- XLink: pour décrire la relation
- XPointer: pour identifier une partie du document XML

Mais...

- ♦ **XPath**: normaliser les définitions de XPointer et celles utilisées dans XSLT pour identifier une partie du document XML.
- XInclude: évolution de XLink pour la définition de liens entre documents et fragments de documents.
- □ **DOM**, «Document Object Model»: arborescence objet

But : Définir une interface standardisée pour l'accès à un contenu XML depuis un environnement de programmation (Java, JavaScript, C++).

☐ SAX, «Simple API for XML»:

But: disposer d'une API commune pour la commande de parseur XML.

☐ XML Schema: Le DTD n'exprime pas de typage de données ce qui est un inconvénient pour la gestion de données structurées.

But : permettre de décrire un modèle de document XML en XML, très complexe d'utilisation



☐ XML Encryption : l'échange d'un document XML peut être sécurisé au travers du protocole d'échange utilisé.

But: Sécuriser le contenu du document: confidentialité & signature numérique. Le format XML Canonical permet de diminuer les différences entre documents (suppression des espaces inutiles, normalisation des guillemets).

- □ XML 1.1 : XML 1.0 est déjà basé sur unicode 2.0

 But : Tenir compte des ajouts dans Unicode pour le support des langages : mongol, birman, cambodgien
- des formats XML dédiés: SOAP, SVG, XHTML, MathML, XForms, etc.











XML vs HTML 68

Dans un document HTML, on trouve mélangés le contenu et sa présentation :

polices de caractères ;

titres:

images, liens hypertextes,

tableaux:

paragraphes;

etc.

L'ensemble est orienté homme-machine, c-à-d qu'une personne visualise ce document sur un écran d'ordinateur.

Un fichier XML parait identique à un fichier HTML, mais il est plus exigeant car il contient les données et leur structure logique.

```
<html> <?
<p> <!
<br/>
<
```

HTML n'offre qu'un jeu limité de balises (tag) auxquelles il ne sera possible que d'affecter des effets de mise en forme, par exemple grâce à une feuille de style CSS.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Test SYSTEM "Test.dtd"> <Test>
<nom>Campus de la Borie</nom>
<adresse>
<rue>123 avenue Albert Thomas</rue>
<code-postal>87060</code-postal>
<localite>Limoges CEDEX</localite> </adresse>
</Test>
```

XML permet de définir ses propres balises (en gras) et donc de leur donner du sens. Il sera ainsi possible de leur affecter non seulement des effets de mise en forme, mais aussi de leur appliquer des traitements logiques complexes.













Structure d'un document XML

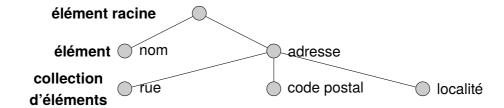
Un document XML est représenté comme un arbre d'éléments.

Il intègre la notion de «lien» entre documents.

Les éléments XML peuvent posséder des attributs :

▶ l'attribut est un couple (nom, valeur) associé à un élément et précisant ses caractéristiques.

Structure logique d'un document XML



Intérêt des langages utilisant des balises

L'introduction du format HTML et sa très grande diffusion ont relancé l'intérêt pour les documents structurés à l'aide de balises :

- o indiquées de manière textuelle;
- intégrées au document lui-même.

Elles visent à séparer la structure du contenu du document.

Constat sur HTML

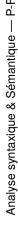
- * très grand succès;
- bien adapté à la diffusion d'informations;

mais

- pas extensible;
- ⊳ peu structuré;
- > peu d'outils de validation des données d'un document.

Les limitations d'HTML se traduisent par :

- les utilisateurs ne peuvent pas définir leurs propres balises pour enrichir leurs documents;
- les documents sont «plats», les enrichissements typographiques sont restreints ;
- il existe des problèmes lors d'échanges d'informations devant respecter une certaine organisation, c-à-d que l'information transmise ne peut être suffisamment structurée.













Buts de XML

XML permet de définir à la fois la structure logique d'un document et la façon dont il va être affiché :

□ gérer des documents **mieux et plus** structurés qu'avec HTLM;

permettre à ces documents d'être traités, indexés, fragmentés, manipulés plus facilement qu'avec
 HTML

XML est une «application» de SGML, «Standard Generalized Markup Language», plus simple, qui peut être mise en œuvre pour le Web et dans des applications utilisant peu de ressources.

Afficher un document XML: XSL

Présentation d'un document XML

Pour présenter un document XML sur un support quelconque, il doit être associé à une feuille de style XSL, par type de format en sortie (XML, HTML, WML, etc): offrir à l'utilisateur plusieurs types d'accès: papier, écran, CDROM, SmartPhone ou même en braille, alors que la présentation d'un document HTML ne peut être visualisé que par un navigateur Web.

Exemple: la feuille de style «test.xsl»:

```
1 <xsl:stylesheet version="1.0">
2 <xsl:output method="xml">
3 <xsl:template match="/Test">
4 <font face="times" size="12"> <xsl:apply-templates/> </font> </xsl:template>
5 <xsl:template match="nom"> <font size="+2" style="bold"> </font> </xsl:template>
6 </xsl:stylesheet>
```

Dans cette feuille de style, on associe un style de caractère à chaque balise définissant un type de données :

- la police de caractères par défaut est "Times 12"
- le nom doit apparaître en "Times 14" et en gras.













Format XML: DTD, «Document Type Description»

L'interêt de XML, par rapport à HTML, est la séparation du contenu de la structure des données. La description des données peut être placée dans un fichier séparé qui donne le DTD, «Document Type Description».

Le DTD sert:

- à décrire toutes les balises du document XML,
- * à définir les relations entre les éléments,
- * à valider un document XML car il contient les règles à suivre pour respecter sa structure.

Le DTD sert à définir des classes de documents.

Exemple «test.dtd»

L'élément adresse comprend trois éléments distincts: la rue, le code postal et la localité.

La syntaxe utilisée pour la DTD n'est pas la même que celle utilisée pour les données et les feuilles de style.

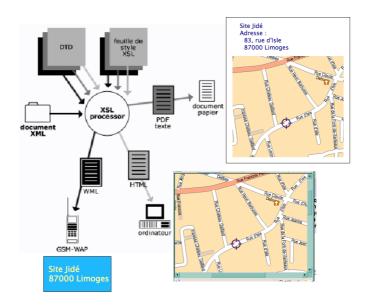


Utilisation de XSL et DTD

Génération des documents cibles

On utilise:

- une feuille de style spécifique pour chaque support (à chaque type de transformation effectuée à partir du document source XML)
- éventuellement, en tenant compte d'une DTD spécifique à chaque support.















DTD: notion de classe de document

XML utilise la notion de classes de documents ou de type de document au travers d'un DTD.

Ce DTD est un fichier qui décrit une structure :

- description des différentes balises utilisables;
- description des imbrications possibles de ces balises.

Ainsi, un document :

- doit être construit suivant les directives de XML;
- peut suivre le format décrit dans un DTD particulier.

Un document est:

- ▷ «bien formé», c-à-d syntaxiquement correct s'il suit les règles de XML
- «valide», s'il se conforme à la DTD associée.

Il est plus proche de SGML que de HTML:

- HTML est une application de SGML;
- XML est un sous-ensemble des fonctionalités de SGML.

La puissance de XML réside dans sa capacité à pouvoir décrire n'importe quel domaine de données grâce à son extensibilité.

Il suffit dans ce cas là de définir:

- le vocabulaire (les balises et leurs attributs);
- la structure des imbrications de ce vocabulaire;
- la syntaxe de ces données;
- un DTD particulier pour le domaine de données voulu.



Analyse syntaxique & Sémantique — P-FE

Les règles pour obtenir un document valide

Si le document suit ces règles on dit qu'il est «bien formé» :

- * respecter la casse des balises <societe> ≠ <Societe>
- * toujours mettre une balise de fin ...
- * mettre les valeurs des attributs entre guillemets <balise attr="valeur">
- * ne pas entrelacer les ouvertures et fins de balises différentes
 - ♦ ... <I> </I> est interdit
 - ♦ ... <I> ... est autorisé

Il doit correspondre à un arbre correct.

Il est possible de vérifier le caractère bien formé avec la commande shell xmllint:

```
$ xmllint document_annuaire.xml
document_annuaire.xml:5: parser error : Opening and ending tag mismatch: companyname line
4 and response
</response></companyname>
```

et la validité d'un document XML:

```
$ xmllint --valid --noout mon_fichier.xml --dtdvalid ma_dtd.dtd
```



XML: les atouts

 acceptation en tant que standard pour la description de données par les principaux acteurs du marché (Adobe, Microsoft, Apple...).

- format utilisé aussi bien dans des logiciels de mise en page comme «Quark Xpress» ou «Indesign» d'Adobe, et qui permet, par exemple, dans ces logiciels de mettre en rapport un document avec le contenu d'une base de donnée :
 - les données sont décrites au format XML;
 - la mise en page de ces données s'appuie sur XML.

On obtient alors un document mis en page dont le contenu peut être dynamique!

- lisible: aucune connaissance ne doit théoriquement être nécessaire pour comprendre un contenu d'un document XML;
- o structure arborescente : ce qui permet de modéliser une majorité de problèmes informatiques ;
- universel et portable : les différents jeux de caractères sont pris en compte ;
- facilement échangé: facilement distribué par n'importe quel protocole à même de transporter du texte, comme HTTP;
- facilement utilisable dans une application: un document XML est utilisable par toute application pourvue d'un analyseur syntaxique, un «parser» de code XML;
- extensible : un document XML doit pouvoir être conçu pour organiser les données de tous domaines d'applications



Exemple de document XML

Le document XML contient différentes parties :

- ☐ déclaration de la version XML utilisée;
- □ type du document utilisé;
- □ corps du document :

Sur l'exemple, on voit qu'il y a des balises :

- b habituelles dans HTML (<h1>, <imq>, ...)
- > spécifiques à l'application (ici <individu>, <nom>, , prenom>)

Ces balises spécifiques fournissent la structure d'une donnée où, chaque individu est composé d'un nom et d'un prénom.

Les balises peuvent avoir des attributs: <individu id="125">



XML: Utilisation des balises spécifiques

Ces balises permettent de stocker les informations comme dans une table de BD:

est équivalent à la table suivante :

Nom	Prénom
Padme	Amidala
Ohi Wan	Kenobi

Les données en XML peuvent également se représenter sous la forme d'attributs :

```
<individu nom="Padme" prenom="Amidala"></individu>
```

En général, les attributs sont utilisés pour les informations non affichables dans les navigateurs (exemple : la valeur i d qui est la clé d'enregistrement de l'individu dans la base de donnée).



XML: Syntaxe des éléments constitutifs

Pour les identifiants des balises :

- les noms peuvent contenir des lettres, des chiffres;
- ils peuvent contenir des caractères accentués;
- ils peuvent contenir les caractères «_», «.», «-»;
- ils ne doivent pas contenir: «?», «'», «\$», «^», «;», «%»;
- o ils peuvent contenir le «:» pour l'utilisation d'espace de nom;
- les noms ne peuvent débuter par un nombre ou un signe de ponctuation;
- les noms ne peuvent commencer par les lettres xml (ou XML ou Xml...);
- les noms ne peuvent contenir des espaces;
- la longueur des noms est libre mais on conseille de rester raisonnable;

Pour les attributs:

- ▷ il n'existe qu'un seul attribut avec un nom donné (si il y a un besoin, il faut utiliser des éléments et non des attributs).

Pour I'«encoding»:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Il doit se trouver en première ligne du document.

Actuellement, on préfère utiliser l'unicode, ce qui permet d'être dans l'encodage naturel d'XML.



9 DTD: Définition d'une classe de documents

Les définitions de classes de document peuvent être :

- □ **internes**, c-à-d intégrées au document lui-même;
- externes, c-à-d chargées à partir d'un fichier auquel on fait référence à l'aide d'une URL.

Définition interne

La balise < ! DOCTYPE > fournit la structure du document :

```
<!DOCTYPE nom [ ... ] >
```

Elle crée un type de document et se compose de deux parties :

- 1. une déclaration de constantes (ou entités);
- 2. une déclaration des éléments du document.

Exemple:

Définition d'une classe de documents

Définition externe

Elle se fait à l'aide de la balise : <!DOCTYPE nom statut url > où:

- ▷ nom: du type de document
- > statut:
 - PUBLIC correspond à un type de document public;
 - ♦ SYSTEM correspond à un type de document local à une organisation.

Exemple:

```
<!DOCTYPE manual PUBLIC "-//DTD manual//EN" "http://www.unilim.fr/manual.dtd">
<!DOCTYPE individu SYSTEM "individu.dtd">
```









DTD: La notion d'entités

La balise <!ENTITY nom valeur > définit une constante utilisable dans le document. Lorsque l'analyseur du document XML lit une entité, il la remplace par sa substitution.

```
Exemple: <!ENTITY euro "6.55957 F">,où:
    nom_de_l'entité = euro
    substitution_de_l'entité = 6.55957 F
Une entité est utilisée sous la forme «&nom_de_l'entité;».
Exemple: la valeur d'un euro est &euro;.
```

Les entités paramétrées

Il est possible de définir également des entités «paramétrées» :

Avantage : on regroupe toutes les entités «similaires» dans un même fichier.

```
<!ENTITY % nom url >
Exemple:<! ENTITY %compagnies "http://localhost/compagnies.den">
L'emploi de %compagnies; avec dans le fichier compagnies.den les définitions suivantes:
<!ENTITY IBM "International Business Machines" >
<!ENTITY ATT "American Telephone and Telegraph" >
permet d'utiliser les constantes &IBM; et &ATT; dans un document
```

DTD: La notion d'élément

```
<!ELEMENT nom (type) >
Exemple: <!ELEMENT mondocument (#PCDATA) >, où:
o nom: définition d'un élément, c-à-d le nom de la balise (ici mondocument);
o type: type de l'élément.
```

Types de données disponibles dans XML

- #PCDATA: «Parsed Character DATA», ce sont des données interprétées qui peuvent être tout caractère sauf < > & encodés respectivement < > et &;
 Il est possible d'utilisé dans ces caractères des entités.
- ANY: tout ce que l'on veut comme élément déjà déclarés;
- ∘ EMPTY: rien!

Exemple : < ! ELEMENT BR EMPTY > veut dire que la balise
 doit s'utiliser de la façon suivante :
 < /BR> ou

Les types de données définis par l'utilisateur

Exemple:<!ELEMENT individu nom >

Cela veut dire que la balise <individu> ne peut contenir qu'une définition de nom.

L'utilisation:

- <individu><nom>Padme</nom></individu> est correcte:
- o <individu>hello</individu> est mauvaise.



Il est possible d'utiliser plusieurs types à la suite les uns des autres en utilisant des opérateurs pour spécifier des «enchaînements» de type :

```
| alternative
, séquence
? optionnel
* 0 ou n
+ 1 ou n
( ) regroupement
```

Exemple:

```
<!ELEMENT nom (#PCDATA) >
<!ELEMENT prenom (#PCDATA) >
<!ELEMENT individu (nom, prenom*) >
<!ELEMENT livre (titre, auteur, remerciements?, chapitre+) >
```

Type mixte

Il est possible également de définir un type mixte :

```
<!ELEMENT définition (#PCDATA | terme) *>
```

Ce qui permet d'avoir du texte, et dans ce texte une ou plusieurs balises <terme>.



DTD: les attributs d'éléments

```
Le lien se fait par le nom de l'élément avec la liste
<!ATTLIST nom de l'element
                                          d'attributs.
nom_de_l'attribut type propriété
               >
Exemple:
        <!ELEMENT individu (nom, prenom+) >
        <!ATTLIST individu
                 noss ID #REQUIRED
                 adresse CDATA #IMPLIED
                 situation (celibataire|marie|divorce) "celibataire"
                 pays CDATA #FIXED "France">
Types reconnus
▷ CDATA: chaîne de caractères quelconques non interprétée;
▷ NMTOKEN: chaîne de caractères composée de lettres, chiffres, et de « . _ - : »;
NMTOKENS : liste de NMTOKEN séparée par des espaces ;
  <! ATTLIST concert dates NMTOKENS #REQUIRED>
  <concert dates="08-27-2010 09-15-2010">
```

Types reconnus - Suite

- ▷ ID: la valeur de l'attribut doit être unique au sein du document, sinon l'analyseur syntaxique XML signale une erreur;
- ▷ IDREF: la valeur de l'attribut doit faire référence à une valeur d'ID existante dans le document, sinon l'analyseur syntaxique XML signale une erreur;
- ▷ IDEREFS: une liste de ID séparés par des espaces;

```
<!ELEMENT date EMPTY>
```

```
<!ATTLIST date mois (Janvier | Février ... | Décembre) #REQUIRED>
```

```
<date mois="Janvier"/>
```

Propriétés reconnues

- > #REQUIRED: attribut obligatoire, sinon l'analyseur syntaxique XML signale une erreur;
- ⇒ #IMPLIED: attribut facultatif;
- > #FIXED: l'attribut a une valeur fixe (donnée par défaut);

```
<!ATTLIST projet version FIXED "1.0">
```

- > valeur par défaut donnée entre guillemets
 - <!ATTLIST page_web protocole NMTOKEN "http">

DTD: la notion d'espace de nom

Elle permet de grouper des éléments et des attributs ensembles.

Exemple:

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <document>
3 <nom>Rapport d'activité</nom>
4 <graph:graphique xmlns:graph= 'http://maths.com/dtds/graphml'>
5 <graph:nom>Courbe d'amortissement</graph:nom>
6 ...
7 </graph:graphique>
8 </document>
```

Un espace de nom est défini par xmlns:préfixe=URL, où URL est une valeur unique garantissant l'unicité de l'espace de nom.

lci, le préfixe graph a été défini comme accès à l'espace de nom défini dans le DTD graphml.

L'intérêt des espaces de noms est de faire cohabiter dans un même document des parties différentes pouvant être traitées séparèments de manière automatique.



DTD: autre

Les commentaires

```
Ils sont compris entre les balises <!-- et -->.
```

```
Exemple: <!-- Ceci est un commentaire XML -->
```

Les sections CDATA

Permettent de définir des zones de textes non interprétées par les navigateurs XML

```
Comprises entre les balises < ! [CDATA [ et ] ] >
```

```
Exemple:<![CDATA[ ici <h1> n'est pas une balise mais du texte ]]>
```

Instructions de traitement

```
< ?Programme et ?> désigne le programme chargé d'interprété les instructions.
```

```
Exemple: <?php ... ?>
```

Résumé XML

Il y a seulement 4 balises principales :

```
    <?xml >: version XML utilisée;
```

- <!DOCTYPE >: classe de documents;
- <!ELEMENT >: les balises du document;
- <!ATTLIST >: leurs attributs.

Ces 4 balises sont suffisantes pour construire «n'importe quelle» classe de documents!



10 CSS: présentation rapide

Les «Cascading Style Sheets» sépare la structure de sa mise en page.

Il existe deux concepts essentiels : les sélecteurs et les propriétés.

Les propriétés possèdent différentes valeurs :

color	red yellow rgb(212 120 20)
font-style	normal italics oblique
font-size	12pt larger 150% 1.5em
text-align left right center justify	
line-height normal 1.2em 120%	
display	block inline list-item none

Un **sélecteur** est une liste d'étiquette.

Pour chaque sélecteur, on associe des valeurs à certaines propriétés :

```
b {color: red; font-size: 12pt}
i {color: green}
```

Des sélecteurs plus long permettent de s'adapter au contexte :

```
table b {color: red; font-size: 12pt}
form b {color: yellow; font-size: 12pt}
i {color: green}
```

C'est le sélecteur le plus «spécifique» dont la valeur s'applique.



Exemples:

CSS: exemples

```
<head>
                                          <body>
<style type="text/css">
                                          <br/><br/>b class=foo>Hey!</b>
b {color: red;}
                                          <b>Wow!
b b {color: blue;}
                                          <br/><b>Amazing!</b>
b.foo {color: green;}
                                          <b class="foo">Impressive!</b>
b b.foo {color: yellow;}
                                          b.bar {color: maroon;}
                                          <i>Fantastic!</i>
</style>
                                          </b>
<title>CSS Test</title> </head>
                                          </body>
```

Hey! Wow! Amazing! Impressive! k00l! Fantastic!

```
h1 { color: #888; font: 50px/50px "Impact"; text-align: center; }
ul { list-style-type: square; } em { font-style: italic; font-weight: bold; }
<html> <head><title>Phone Numbers</title>
k href="style.css" rel="stylesheet" type="text/css"> </head>
<body> ... </body>
</html>
```



Analyse syntaxique & Sémantique — P-FB

Le fichier «annuaire.xml»

```
1 <?xml version="1.0" encoding="UTF-8"
2    standalone="yes"?>
3    <?xml-stylesheet href="annuaire.css"
4    type="text/css"?>
5    <response>
6    <companyname>Ma compagnie à moi</companyname>
7    <telephone>05 55 43 69 83</telephone>
8    </response>
```

Le résultat dans un navigateur :

Ma compagnie à moi

05 55 43 69 83

Le fichier «annuai re.css» 1 < style type="text/css"> 2 response { } 3 telephone { 4 display: block; 5 font-size: 11pt; 6 font-style: italic; 7 font-family: arial; 8 padding-left: 10px; 9 color: red; 10 } 11 companyname { 12 display: block; 13 width: 250px; 14 font-size: 16pt; 15 font-family: arial; 16 font-weight: bold; 17 background-color: teal; 18 color: white; 19 padding-left: 10px; 20 }



21 </style>

11 XSLT: aller plus loin que les CSS...

Ce langage permet de définir des «transformations» à appliquer sur un document XML auquel il est appliqué.

Une feuille de style XSLT:

* est définie par:

```
1 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
2 ...
3 </xsl:stylesheet>
```

- * contient des règles de «templates»;
- * commence le traitement sur le noeud racine du fichier XML.

Une règle de template :

♦ est définie par :

```
1 | <xsl:template match="..."> ...
2 | </xsl:template>
```

- ♦ XLST trouve la règle de template qui correspond au noeud courant en sélectionnant le plus spécifique ;
- évalue le corps du template.

On utilise XPath pour:

- spécifier les motifs pour la sélection des règles de template;
- o sélectionner les noeuds pour les traiter syntaxe proche des chemins fichier Unix.;
- o créer des conditions booléennes;
- générer du texte pour le document en sortie.



XSLT: accès aux données

Il est possible de:

- * **appliquer** un template: <xsl:apply-templates/> (pour le noeud racine ce n'est pas la peine, du moment qu'un template lui correspond indiqué par «/» ou bien simplement son nom);
- * à l'intérieur d'un template:

```
récupérer la valeur d'une balise: <xsl:value-of select="nom balise"/>
                     récupérer la valeur d'un attribut: <xsl:value-of select="@nom attribut"/>
                     parcourir les sous-noeuds: <xsl:for-each select="nom sous noeud">
                     parcourir les attributs: <xsl:for-each select="@nom attribut">
      1 < 2 \times 1 < 
      2 < xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      3 <xsl:template match="/">
      4 < html > < head > < title > Livres < / title > < / head >
      5 <body><xsl:for-each select="booklist/book">
      6 <b><xsl:value-of select="title"/></b>
      7 < i > < x s l: value - of select = "author" / > < / i >
      8 <a href="http://www.amazon.com/exec/obidos/ASIN/{isbn}/">Details</a>
      9 </xsl:for-each>
10 < /body> < /html>
11 </xsl:template>
12 </xsl:stylesheet>
```

On peut utiliser la notation «{isbn}» permet de récupérer la valeur d'une balise directement dans une chaîne de caractère (ici, pour définir un lien).



XSLT: opérations avancées

Toujours à l'intérieur d'un template et dans le cadre d'un parcours avec <xsl:for-each ...>:

- * faire des tests de conditions:
 - directement après le for-each:

```
<xsl:if test="not(nom='Mon livre secret')">.
```

Cela permet de traiter ou non un noeud parcouru par le for-each

pour tester la présence d'un élément (balise ou attribut) :

```
<xsl:if test="dédicace">dédicace: <xsl:value-of select="dédicace"/>
</xsl:if>
```

* des tris:

```
<xsl:sort select="critere" data-type="number"/>
À mettre juste après le for-each
```

* des sélections:

```
1 <xsl:choose>
2 <xsl:when test="@type='essence'">Moteur essence</xsl:when>
3 <xsl:when test="@type='eau'">Vous vous moquez !</xsl:when>
4 <xsl:otherwise>À pédales ?</xsl:otherwise>
5 </xsl:choose>
```



XSLT: un exemple de liste de livres

Exemple de fichier XML:

```
1 < ?xml version="1.0"?>
 2 < ?xml-stylesheet href="test_xsl.xsl"
    type="text/xsl" ?>
 4 < booklist title="Science Fiction">
 5 < book >
    <title>The Naked Sun</title>
    <author>Isaac Asimov</author>
    <isbn>0553293397</isbn>
 9 < /book>
10 <book>
    <title>Foundation's Triumph</title>
    <author>David Brin</author>
    <isbn>0061056391</isbn>
14 </book>
15 <book>
   <title>Snow Crash</title>
   <author>Neal Stephenson</author>
    <isbn>0553380958</isbn>
19 </book>
20 </booklist>
```

En ligne 2, on indique dans l'entête du document XML, la feuille de style à appliquer.

Le résultat obtenu avec la feuille de style du transparent suivant :

The Naked SunIsaac Asimov Details

Foundation's TriumphDavid BrinDetails

Snow CrashNeal Stephenson Details



XSLT : suite de l'exemple de liste de livres

```
Le fichier xslt associé:
 1 < xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 2 < xsl:output method="html" encoding="UTF-8"/>
 3 < xsl:template match="booklist">
 4 < h + m 1 >
 5 < head > < title > < xsl: value - of select = "@title" / > < / title > < / head >
 6 < body > < xsl:apply-templates / > < /body >
 7 < /html>
 8 </xsl:template>
 9
10 < xsl:template match="book">
11 
12 <b><xsl:value-of select="title"/></b>
13 <i><xsl:value-of select="author"/></i>
14 < a href="http://www.amazon.com/exec/obidos/ASIN/{isbn}/">Details</a> 
15 </xsl:template>
16 </xsl:stylesheet>
```

Explications

- ⊳ ligne 6, on applique les «templates» définis plus bas;
- ▷ ligne 14, on obtient dans la chaine du lien, le contenu de la balise «isbn».



Analyse syntaxique & Sémantique — P-FB

XPath permet d'accéder à un élément particulier de l'arborescence d'un document XML, il permet de «naviguer» dans un «chemin» à travers l'arborescence.

XPath reconnait différents types de nœuds :

- le document XML entier;
- la racine, ou «root» et tous les éléments contenus dedans ;
- les éléments, attributs, et le texte;

Il est utilisé en particulier dans XSL.

- / root/one: sélectionner l'élément <one>
- one/two: saut relatif d'un élement <one> à un élément <two>
- ♦ /root/one/two:sélectionnerl'élém.enfant < two>
- root/one/three/lundi/@meteo:un attribut est décrit en utilisant le symbole @
- root/one/three/lundi[@meteo='pluie']:
 un attribut peut être trouver à l'aide d'une comparaison portant sur sa valeur indiquée entre «[]»

Il est possible de sélectionner l'ensemble des éléments dans un document sans tenir compte de leur localisation dans l'arborescence :

```
▷ //@meteo: sélectionner tous les attributs «meteo»;
```

> //two: sélectionner tous les éléments «<two>».



Il est possible de sélectionner un ensemble d'éléments en utilisant le wildcard «*»:

- > /root/one/three/*: sélectionne les éléments à l'intérieur de <three>;
- > //* :sélectionne tous les éléments du document.

Le caractère «.» permet de se référer à l'élément courant, les caractères «..» permet de se référer à l'élément parent:

Si un document possède une série d'éléments identique dans une liste, on peut utiliser une valeur d'index entre « [] » pour sélectionner un élément particulier:

- ▷ /root/one/two[1]: le premier élement «<two>»;
- ▷ /root/one/two[2]:le second;
- ▷ /root/one/two[last()]:le dernier.

Attention

La numérotation des éléments commence à 1.

XPath 100

Exemple

```
1 < ?xml version="1.0"?>
                                         1 < ?xml version="1.0"?>
   <?xml-stylesheet type="text/xsl"</pre>
                                         2 < xsl:stylesheet version="1.0"
 3 href="xsltext.xsl"?>
                                         3 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
                                           <xsl:template match="/">
 4 <t.est.>
 5 < title size="7">Exemple XSLT</title>5 < html><body><center>
 6 1 ine>50%</line>
                                         6 < font color="ff0000">
 7 | <para>ceci est un paragraphe</para> 7 | <xsl:attribute name="size">
 8 < text>Texte 1 < /text>
                                         8 < xsl:value-of select="test/title/@size" />
 9 <text>Texte 2 </text>
                                         9 </xsl:attribute>
10 < text > Text = 3 < / text >
                                        10 < xsl:value-of select="test/title"/>
                                        11 </font> <br/> <font size="2" color="0000ff">
11 < text > Text = 4 < / text >
12 </test>
                                        12 < xsl: value-of select="test/text[1]"/> </font>
                                        13 <br/> <font size="3" color="008800">
 Le résultat :
                                        14 < xsl: value-of select="test/text[2]"/> </font>
           Exemple XSLT
                                        15 <br/> <font size="4" color="ff00ff">
                                        16 < xsl: value-of select="test/text[3]"/> </font>
```

19 < br/> < hr>

17 |

20 < xsl:attribute name="width">

22 </hr> </center> </body> </html> 23 </xsl:template> </xsl:stylesheet>

18 < xsl: value-of select="test/text[last()]"/>

21 < xsl:value-of select="test/line" /> </xsl:attribute>

Texte 1
Texte 2
Texte 3
Texte 4



Lien entre éléments XML : exemple d'une liste de livres et d'auteurs

On a définit deux parties distinctes :

- \square <book>...</book>: où l'on donne le nom du livre et son année d'édition;
- □ <author>...</author>:où l'on détaille le nom complet de l'auteur.

Le fichier XSLT utilise la notion de key pour faire un lien entre ces deux parties à la manière d'une base de données relationnelles: c'est la valeur unique de <author-ref> qui fait le lien entre le livre et son auteur.

```
<xsl:stylesheet version="1.0"</pre>
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:key name="lien" match="author" use="@id" />
<xsl:template match="/library">
<ht.ml>
  <head>
  </head>
  <body>
   <h2>Bibliothèque</h2>
    <t.head>
         Titre
         Année
         Auteur(s)
       </thead>
      <xsl:for-each select="book">
       <xsl:value-of select="title"/>
       <xsl:value-of select="year"/>
       <+d>>
         <xsl:value-of select="key('lien', author-</pre>
ref)/first-name"/>
         <xsl:text> </xsl:text>
         <xsl:value-of select="key('lien', author-</pre>
ref)/last-name"/>
       </t.d>
     </t.r>
     </xsl:for-each>
   </body>
</html>
</xsl:template>
</xsl:stylesheet>
```



COMPLÉMENTINT



102

Pour aller plus loin : un guide de voyage avec affichages optionnels

On voudrait:

13 </name>
14 </city>

- tenir compte de la localisation du document indiquée par «location»;
- afficher la description «<bri>brief>», «<general>» et «<fun>»;
- afficher chaque commentaire indiqué par des «<comments>»;
- en fonction du nombre de commentaires, afficher si la destination est populaire ou non.



COMPLÉMENTNT



103

Pour aller plus loin : un guide de voyage avec affichages optionnels

Début du fichier XSL

```
<?xml version="1.0"?>
   <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
   <html><body>
   <xsl;if test="/name[@location='uk']">
                                                               1 ⇒ teste si cela s'applique sur la version an-
     <font size="2"><i>UK version </i></font>
                                                                   glaise «uk»
   </xsl:if>
                                                               2⇒sélection du contenu de la balise brief
   <hr width="60%" align="left"/>
                                                               3⇒sélection du contenu de la balise general
 9 < ?-- HTML table -->
                                                               4⇒sélection du contenu de la balise fun
10 
11 CategoryInfo
                                                               Le résultat :
12 <font color="0000ff">Brief Description:</font>
     UK version
14 < /t.r >
                                                                  Category
                                                                                   Info
15 <font color="0000ff">General info:</font>
                                                                Brief Description: Liverpool is in the North of England.
                                                                General info:
                                                                         Three universities, two cathedrals, one airport
     <xsl:value-of select="/general"/>
                                                                Fun bits:
                                                                         Bars, clubs, the FACT center
17 
                                                                Places to Visit:
                                                                         Albert Dock
18 <font color="0000ff">Fun bits:</font>
                                                                         Speke Hall
     <xsl:value-of select="/fun"/>
                                                                         The Cathedrals
                                                                Your Comments
20 
                                                                         An interesting place - Jim
21 <font color="0000ff">Places to Visit:</font>
                                                                         Scary! - Beaker
                                                                         Good fun if you like to party - Sally
22 < /t.d>
23 
                                                                Page popularity: High
```



COMPLÉMENTNT



Pour aller plus loin : un guide de voyage avec affichages optionnels

104

La suite du fichier XSL

```
22 <xsl for-each select="city/name/places">
                                                        1⇒parcours des différentes «places»
23 <xsl:sort select="places"
                                                        2⇒récupération de la valeur de la balise cou-
24
        order="descending" data-type="text"/>
                                                            rante
    < t.r > < t.d > < /t.d >
                                                        3⇒choix entre différentes conditions
      <xsl:value-of select="."/> 

    ⊕ permet de compter les éléments et de prendre

    </t.r>
                                                            une décision booléenne
28 </xsl:for-each>
29 <font color="0000ff">Your Comments:</font>
30 
31 < xsl:for-each select="city/name/comments">
32 <xsl:sort select="comments" order="ascending"/>
33
    34 </xsl:for-each>  <hr width="60%" align="left"/>
35 <xsl; choose>
    <xsl when test="count(/city/name/comments)>2">
36
37
      <font size="2"> Page popularity: High </font>
38
    </xsl:when>
    <xsl:otherwise> <font size="2"> Page popularity: Average
40 </font>
    </xsl:otherwise>
42 </xsl:choose>
43 </body></html>
44 </xsl:template>
45 </xsl:stylesheet>
```









DTD: les limitations

```
Soit un DTD pour définir des recettes :
```

```
1 < !ELEMENT collection (description, recette*) >
 2 < !ELEMENT description (#PCDATA) >
 3 <! ELEMENT recette (titre, date, ingredient*, preparation, remarque?,
 4 nutrition, relatif*) >
 5 < !ATTLIST recette id ID #IMPLIED>
 6 < !ELEMENT titre (#PCDATA) >
 7 < !ELEMENT date (#PCDATA) >
 8 < !ELEMENT ingredient (ingredient*, preparation)?>
 9 < !ATTLIST ingredient name CDATA #REQUIRED quantité CDATA #IMPLIED
   unité CDATA #TMPLTED>
101
11 < !ELEMENT preparation (étape*) >
12 < !ELEMENT étape (#PCDATA) > < !ELEMENT comment (#PCDATA) >
13 < !ELEMENT nutrition EMPTY>
14 < !ATTLIST nutrition calories CDATA #REQUIRED graisse CDATA #REQUIRED
   protéine CDATA #REQUIRED>
16 < !ELEMENT relatif EMPTY>
17 < !ATTLIST relatif ref IDREF #REQUIRED>
```

Problèmes:

- o calories: doit être un nombre positif;
- o protéine : est un pourcentage compris entre 1 et 100 ;
- o unité ou quantité : un seul à la fois parmi les deux ;

Comment faire ? Remplacer les DTDs...



12 XSD, «XML Schema Definition»

Buts:

- * plus expressif que les DTDs;
- utilise une notation XML;
- plus de simplicité;
- * auto-documenté;

Les contraintes techniques :

- * doit gérer les espaces de noms;
- permet de définir des types de données utilisateurs;
- supporte de l'héritage à la manière de l'objet;
- * possède une documentation interne;
- * ...

Les propositions:

- ▷ Définition simple de type, simple Type : utilisation de chaîne unicode ;
- ▷ Définition complexe de type, complexType : définit un contenu et un modèle d'attribut ;
- ▷ Déclaration Élement : associer un nom d'élément avec un type simple ou complexe ;
- ▷ Déclaration Attribut : associer un nom d'attribut avec un type simple ;
- ▶ Possibilité de définir le nombre d'occurrence d'un élément/attribut.



XSD: exemple de carte de visite

Exemple: Définition d'une carte de visite

Remarques:

- ▷ L'espace de nom est «b».
- ▶ Le schemaLocation assure l'unicité du XSD pour éviter les conflits.
- ▷ Il est possible d'utiliser :

```
<card xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="schema.xsd">
```

Pour faire référence à un fichier XSD local, à l'aide du noNamespaceSchemaLocation.

Schémas XML : suite de l'exemple de carte de visite

```
1 < schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:b="http://businesscard.org"
    targetNamespace="http://businesscard.org">
 4 < element name="card" type="b:card type"/>
 5 < element name="name" type="string"/>
 6 < element name="title" type="string"/>
 7 < element name="email" type="string"/>
 8 < element name="phone" type="string"/>
 9 < element name="logo" type="b:logo type"/>
10 <attribute name="uri" type="anyURI"/>
11 < complexType name="card_type" > < sequence >
12
    <element ref="b:name"/>
   <element ref="b:title"/>
1.3
14
    <element ref="b:email"/>
15
    <element ref="b:phone" minOccurs="0"/>
16
    <element ref="b:logo" minOccurs="0"/>
17
    </sequence> </complexType>
18 < complexType name="logo_type">
19 <attribute ref="b:uri" use="required"/>
20 </complexType>
21 </schema>
       xterm
   pef@darkstar:~/XML$ xmllint bcard.xml --schema bcard.xsl --noout
    bcard.xml validates
```



XSD, un autre exemple de liste de livres

```
Une description de livre:
 1 < ?xml version="1.0" encoding="UTF-8"?>
 2 < book xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 3 xsi:noNamespaceSchemaLocation="schema.xsd">
    <title>Cryptography Engineering</title>
    <author>Bruce Schneier</author>
    <type>Non-fiction</type>
 7 </book>
 et le fichier schema.xsd:
 1 < ?xml version="1.0" encoding="UTF-8"?>
 2 < xs:schema xmlns:xs= http://www.w3.org/2001/XMLSchema
       elementFormDefault="qualified">
 4 < xs:element name="book">
 5 < xs:complexType > < xs:sequence >
 6 < xs:element name="title" type="xs:string"/>
 7 < xs:element name="author" type="xs:string"/>
 8 < xs:element name="type" type="xs:string"/>
 9 </xs:sequence </xs:complexType </xs:element>
10 <!--end the schema-->
11 </xs:schema>
```

lci, l'espace de nom est xs, très souvent utilisé par défaut.

Analyse syntaxique & Sémantique — P-FB

Définition d'un élément

Un élément en XSD est soit :

- * **simple**: il contient du texte mais ne contient pas de sous-éléments ni d'attributs;
- * **complexe**: il peut avoir des sous-éléments, des attributs, du texte ou bien être vide.

Les différences entre «simple» et «complexe»

```
▷ <separateur/>: pas un type complexe;
```

- > <auteur> Isaac Asimov</auteur>:pas un type complexe;
- > <parking id="456"/>:type complexe;
- > <parking id="456">M. Durand</parking>:type complexe;
- > <individu><nom>James</nom><prenom>Kirk</prenom></individu>:type complexe.

Le type «complexe»

Exemple de type complexe

```
1 <xs:element name="optional" type="xs:string" minOccurs="0"/>
2 <xs:element name="nm" type="xs:string" minOccurs="1" maxOccurs="40"/>
3 <xs:element name="maxfive" type="xs:string" maxOccurs="5"/>
4 <xs:element name="infinite" type="xs:string" maxOccurs="unbounded"/>
```



Les types simples qui s'appliquent aux contenus

Les types de données :

11 type="xs:boolean"

1 type="xs:string" 2 type="xs:decimal" 3 type="xs:integer" 4 type="xs:anyURI" 5 type="xs:date" 6 type="xs:time" 7 type="xs:duration" 8 type="xs:byte" 9 type="xs:negativeInteger" 10 type="xs:nonNegativeInteger"

Les valeurs possibles :

```
1 Chaîne de caractères, ex. "Salut !"
2 Nombre à virgule, ex. 11.6, 5.3
3 Nombre entier, ex. 7, 8, 42
4 Un lien, ex. libpfb.so
5 Année/Mois/Jour, ex. 2003-08-15
6 Heure/Min/Sec, ex. 10:22:00
7 PeriodYrMnthHrMinSec, ex. P4Y3M2D
8 Une valeur sur 8bits de -128 à 127
9 Nombre entier négatif, ex. -256
10 Nombre entier positif, ex. 100, 999
11 TRUE, FALSE, 1 or 0
```

Il est possible de choisir une valeur par défaut :

```
1 <xs:element name="month" type="xs:string" default="Jan"/>
2 <xs:element name="pi" type="xs:decimal" fixed="3.14159"/>
```

Pour définir un élément avec un **contenu vide**, on le définit sans type :

```
1 < xs:element name="separateur"/>
```

XSD: les types de données simple obtenus par «restriction»

On définie un nouveau type de donnée qui va s'appliquer sur un «simple Type» :

```
1 <xs:simpleType name="restreint">
2 <xs:restriction base="xs:string">
3 <xs:maxLength value="4"/>
4 </xs:restriction>
5 </xs:simpleType>
Pour l'utiliser:
1 <xs:element name="type" type="restreint"/>
```

D'autres définitions

- ♦ Pour un type «*string*», il est possible d'utiliser maxLength et minLength.
- ♦ Pour un choix parmi différentes valeurs :

```
1 <xs:enumeration value="News"/>
2 <xs:enumeration value="Sport"/>
3 <xs:enumeration value="Musique"/>
```

♦ Pour un choix d'une valeur numérique :

```
1 <xs:minInclusive value="50"/>
2 <xs:maxInclusive value="200"/>
```

> Pour une taille exprimée en nombre de chiffres :

```
1 <xs:totalDigits value="7"/> un nombre jusqu'à 7 chiffres
2 <xs:fractionDigits value="2"/> le nombre 1.55 est valide, 1.552 ne l'est pas.
```



XSD: les types de données simple obtenus par «restriction»

Il est possible de définir des types de données à l'aide :

o d'expressions régulières :

- d'union pour permettre des alternatives :
- o de liste :

d'énumération :

On peut aussi traiter les espaces :

```
1 <xs:whiteSpace value="preserve"/> "Space text" reste inchangé  
2 <xs:whiteSpace value="collapse"/> "Space text" se transforme en "Space text"
```

«collapse» réduit des espaces multiples en un seul espace comme en HTML.

«preserve» conserve le formatage.

«replace» change les retours à la ligne en espace.

Pour la signature électronique, il est important de gérer les espaces de manière à supprimer toute ambigüité.









XSD: la définition d'attributs

Un attribut est définit avec «attribute» au sein d'un type complexe uniquement.

Chaque définition d'attribut doit être :

- o accompagnée d'un nom et d'un type simple uniquement;
- o déclaré comme optionnel ou requis ;
- o associé à une valeur par défaut.

```
1 <xs:attribute name="code" type="xs:number" use="optional"/>
2 <xs:attribute name="colour" type="xs:string" use="optional"/>
3 <xs:attribute name="id" type="xs:string" use="required"/>
```

On peut définir des groupes pour lier des attributs et les utiliser dans un élément :



On fait le lien entre la définition et l'utilisation avec le ref.

Analyse syntaxique & Sémantique — P-FB

La définition d'un élément vide avec attributs

Cette définition s'utilise comme la balise «img» du HTML.

Un élément avec contenu simple et attributs

Un élément avec contenu complexe et attributs

On ne peut inclure dans la balise «individu» que les deux balises «nom» et «prenom».

Un élément avec contenu mixte

On peut mélanger texte et balises dans le contenu de la balise «employe».



XSD: exemples de déclaration d'élément et d'attribut

Exemples:

- <element name="serialnumber" type="nonNegativeInteger"/>
- <attribute name="alcool" type="pourcentage"/>

Type simples:

string, boolean, decimal, float, double, dateTime, time, date, hexBinary, base64Binary, anyURI etc.

Types dérivés contraints

length, minLength, maxLength, pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, totalDigits, fractionDigits

Exemple

```
1 <simpleType name="score_from_0_to_100">
2 <restriction base="integer">
3 <minInclusive value="0"/> <maxInclusive value="100"/>
4 </restriction>
5 </simpleType>
6
7 <simpleType name="pourcentage"> <restriction base="string">
8 <pattern value="([0-9]|[1-9][0-9]|100)\%"/> </restriction>
9 </simpleType>
```

À la ligne 6, on utilise une expression régulière.



XSD: dérivation de type

La «restriction» est une forme de «dérivation» à partir d'un type existant :

Ici, la dérivation est réalisée sur un type simple. Sur un type complexe, on peut utiliser l'«extension» qui ressemble à de l'héritage...

Conclusion

On appelle «schéma», une description formelle de la syntaxe d'un langage basé sur XML:

- * DTD: language simple de schéma: éléments, attributs, entitées, ...
- * Schéma XML: langage plus avancé:
 - déclaration d'élément/attribut;
 - type simple/complexe/dérivé;
 - ♦ contrôle de la valeur des contenus ;
 - etc.



COMPLÉMENTNT



13 XML et Java

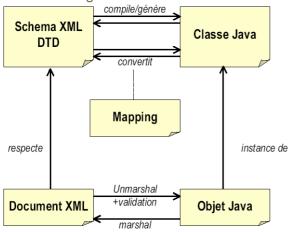
Avec SAX, «Simple API for XML»:

- * qui est une API pour la lecture d'un flux XML :
- * interface événementielle : une balise ouvrante est un événement, une balise fermante est un événement, etc

Avec DOM, «Document Object Model»:

- * génére une arborescence d'objet correspondant à la structure du document XML;
- * possibilité de naviguer dans cette arborescence ;
- * lourd!

XML Data Binding:



Permet de faire de la sérialisation d'objet, mais il faut faire attention, l'ordre entre les noeuds peut être perdu et cela peut poser des problèmes lors de la dé-sérialisation : mauvaise valeur d'index, *etc* Il faut bien définir le format!







COMPLÉMENTNT



SAX: Utilisation 119

- Utilisation du package org.xml.sax;
- 2. Implémentation de l'interface Content Handler;
- 3. instanciation d'un parser en lui indiquant quel gestionnaire de contenu à utiliser;
- 4. fini!

API du ContentHandler

- setDocumentLocator: connaitre la position du curseur de lecture (numéro ligne et de colonne) dans le document; utile pour déboger mais à éviter de manipuler.
- startDocument: à appeler une seule fois au démarrage de l'analyse avant toutes les autres (à part setDocumentLocator) et sert à initialiser le parser.
- o endDocument : à appeler à la fin, permet de nettoyer et de finaliser le parser.
- processingInstruction: intervient à chaque instruction de fonctionnement, c-à-d en dehors de ceux concernant la structure du document.

```
Exemple: <?xml version= '1.0'>
```

- startPrefixMapping: intervient à chaque rencontre de l'indication d'un espace de nom;
- endPrefixMapping: pour la fin de l'utilisation d'un espace de nom;







COMPLÉMENTINT



SAX: Utilisation 120

API du ContentHandler - Suite

- startElement: début de la lecture d'un élément XML (la méthode la plus importante): startElement (String namespaceUri, String localName, String rawName, Attributs atts)
 - nameSpaceUri: la chaîne de caractères contenant l'URI complète de l'espace de nommage de l'élément ou une chaîne vide si l'élément n'est pas compris dans un espace de nommage;
 - localName: le nom de l'élément sans le préfixe s'il y en avait un;
 - rawName: est le nom de l'élément version xml 1.0 c-à-d \$prefix:\$localname;
 - attributs: la liste des attributs de l'élément que l'on étudiera un peu plus loin.
 - ♦ endElement: fin de lecture d'un élément.
 - characters: correspond au texte entre les balises : <maBalise>un peu de texte</maBalise>







16

17

18

19

20

21

22

23

2425

COMPLÉMENTNT



121

SAX: un exemple

1 // Exemple d'utilisation : récupération de la quantité de farine dans une recette 2 import java.io.*; 3 import org.xml.sax.*; 4 import org.xml.sax.helpers.*; 5 import org.apache.xerces.parsers.SAXParser; public class Flour extends DefaultHandler { float amount = 0;public void startElement (String namespaceURI, String localName, 10 String gName, Attributes atts) { 11 if (namespaceURI.equals("http://recipes.org") && localName.equals("ingredient")) 12 13 String n = atts.getValue("", "name"); 14 if (n.equals("flour")) 15 { String a = atts.getValue("", "amount"); // assume 'amount' exists

amount = amount + Float.valueOf(a).floatValue();}}

public static void main(String[] args) {

catch (Exception e) {e.printStackTrace();}

SAXParser p = new SAXParser();

System.out.println(f.amount);

Flour f = new Flour();

p.setContentHandler(f);

try { p.parse(args[0]); }





COMPLÉMENTINT



122

SAX : suite de l'exemple

```
1 < ?xml version="1.0" encoding="UTF-8" ?>
 2 <?dsd href="recipes.dsd"?>
 3 < collection xmlns="http://recipes.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4
    xsi:schemaLocation="http://recipes.org recipes.xsd">
    <description>Some recipes used in the XML tutorial.</description>
 7 < recipe>
    <title>Ricotta Pie</title>
 8
    <ingredient name="ricotta cheese" amount="3" unit="pound" />
10|...
11
    preparation>
12
    <step>Beat the 12 eggs, 2 cups sugar and vanilla extract together.
13
    </preparation>
14
    </ingredient>
    <ingredient name="flour" amount="4" unit="cup" />
15
16|...
17
    <ingredient name="vanilla extract" amount="1" unit="teaspoon" />
18
   preparation>
19
    <step>Combine the flour, baking powder, and 1 cup of the sugar together.
    </step>
20
21 ...
22
    </recipe>
23 </collection>
```



14 XML DOM & XSLT: utilisation de JavaScript

Dans un fichier XSLT, on peut utiliser du JavaScript, pour manipuler XML & XSL. On va définir du code JavaScript dans le document HTML produit par le fichier XSLT:

```
1 < xsl:template match="/">
                                                                          Partie
 2 < h + m1 >
                                                                          «invisible»
 3 < head>
    <title>Collection de nuages</title>
    <script type="text/javascript"><! [CDATA[</pre>
    var XMLsource = new Object;
    var XSLsource = new Object;
    XMLsource = document.XMLDocument;
    XSLsource = document.XSLDocument;
10
       function changecouleur()
11
         balisefont=XSLsource.documentElement.selectNodes("//font");
12
       couleur=balisefont[0].getAttribute('color');
13
14
       document.body.innerHTML = XMLsource.transformNode(XSLsource); }
    function affichealtitude(altmax, altmin)
15
     { alert("Altitude maximale de ce nuage : "+altmax+"m, minimale : "
16
17
           +altmin+"m."); }
18
    |||></script>
19 </head>
```



XML, DOM & XSLT: utilisation de JavaScript

```
20 <!--
          <body onClick="changecouleur()">-->
21
    <body>
22
      <h1>Les nuages</h1>
23
      <xsl:for-each select="nuages/nuage">
24
      <h2 onClick="affichealtitude({altitude/@max}, {altitude/@min})">
25
      <xsl:value-of select="nom/text()"/></h2>
26
        <e type de nuage possède les <font color="red">espèces</font>
27
        suivantes :
28
        <111>
29
          <xsl:for-each select="nom/espece">
30
            <xsl:value-of select="."/>
31
          </xsl:for-each>
32
        33
      </xsl:for-each>
34
    </body>
35 </html>
36 </xsl:template>
```

Explications:

- o ligne 5 : on ajoute du code JavaScript dans la partie < head > de la page résultat de XSLT, et pour ne pas que ce contenu soit interprété on le met entre < ! [CDATA [et]].
- ligne 8 et 9: on accède au DOM pour la partie XML et XSL;
- ligne 15: la modification d'un élement de XSL est propagé sur la transformation de l'élément XML, ce qui force l'apparence à changer dans le navigateur;
- o ligne 24: pour chaque «nuage» du document XML, on attache un événement sur le click de la souris qui appelle la fonction en passant en paramètre la valeur des attributs altitude/@min et altitude/@max.







XML, DOM & XSLT: utilisation de JavaScript

Le fichier «nuages.xml» associé à l'exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
  <nuages>
  <nuage>
    <nom>altocumulus
      <espece>lenticularis</espece>
      <espece>stratiformis</espece>
 6
      <espece>castellanus</espece>
 8
      <espece>flocus</espece>
 9
    </nom>
10
    <altitude max="6000" min="2000"/>
11
    <hydrometeores>Aucun.
12 </nuage>
13 </nuages>
```



COMPLÉMENTINT



126

15 XML DOM: JavaScript & AJAX

Ajax, «Asynchronous JavaScript + XML» permet à un document HTML chargé dans un navigateur de faire des requêtes asynchrones, c-à-d sans avoir à recharger le contenu du document, sur un serveur pour récupérer un contenu au format XML.

L'objet JavaScript permettant de faire cette récupération est XMLHttpRequest:

- abort (): arrête la requête courante;
- o getAllResponseHeaders (): retourne la chaîne contenant toutes les en-têtes de la réponse HTTP du serveur;
- o getResponseHeader ("name"): retourne la valeur sous forme chaîne d'une en-tête donnée par son nom;
- open ("method", "url", asyncflag, "username", "password"): réalise la configuration de la future requête où l'on peut donner l'URL, la méthode HTTP employée (en général «GET» ou «POST») ainsi que des options supplémentaires comme un «login/mot de passe» et un drapeau si l'on désire que la requête se fasse de manière asynchrone;
- send (content): transmet la requête, avec un paramètre optionnel pouvant être une chaîne ou bien des données du DOM;
- setRequestHeader ("label", "value"): permet de créer des associations «nom, valeur» qui pourront être transmises avec la requête et suivant la méthode sélectionnée («GET» ou «POST»).







COMPLÉMENTINT



XML DOM: JavaScript & AJAX

127

Les propriétés de l'objet XMLHttpRequest:

- onreadystatechange: gestionnaire d'événement, «Event handler», qui est appelé à chaque changement d'état;
- readyState: fournit le statut de l'objet: (0 = unitialized, 1 = loading, 2 = loaded, 3 = interactive,
 4 = complete);
- o responseText: une chaîne correspondant au contenu des données retournées par le serveur;
- responseXML: la représentation DOM des données retournées par le serveur;
- o status: le code numérique de retour de la requête HTTP au serveur le «HTTP status code»;
- o statusText: la chaîne de caractères de description du «code de status».





17



128

XML AJAX : un exemple

```
1 < html > < head > < title > Exemple < / title >
 2
 3 < script type="text/javascript" language="JavaScript">
 4 var obi;
 5 function getData() {
 6 document.body.style.cursor='wait';
 7 if (window.XMLHttpRequest)
    { obj = new XMLHttpRequest();}
 9 else if (window.ActiveXObject)
    { obj = new ActiveXObject("Microsoft.XMLHTTP");}
10
11
12 var goUrl = "http://localhost/document annuaire.xml";
13 obj.onreadystatechange = xmlReady;
14 obj.open("GET", goUrl, false);
15 obj.send();
16 document.body.style.cursor='auto';
```

L'obtention de cet objet dépend du navigateur employé.



XML AJAX: un exemple – suite

```
18 function xmlReady() {
    if (obj.readyState == 4) { // On verifie le succes de la requete AJAX
19
      if (obj.status == 200) { // On verifie que le document a bien ete trouve
20
21
      var xmldoc = obj.responseXML;
22
      var node1 = xmldoc.selectSingleNode('response/telephone');
23
      var node2 = xmldoc.selectSingleNode('response/companyname');
24
      document.forms[0].telephone.value = node1.firstChild.data;
25
      document.forms[0].cname.value = node2.firstChild.data;
26 } }
27 </script></head><body>
28
29 < form id="frmTest">
30 Compagnie: <input type="text" name="cname" /><br />
31 Té Lé phone : <input type="text" name="telephone" /><br />
32 < input type="button" value="Click" onclick="getData();" />
33 </form></body></html>
```

Explications:

- ligne 19 et 20 : on vérifie que la récupération du document XML s'est bien réalisée sans erreur

XML DOM & JavaScript

Les instructions de manipulation de l'arborescence du DOM:

- basename: retourne le nom d'un noeud (mais sans le «namespace»);
- childNodes: retourne la liste de tous les noeuds enfant du noeud sélectionné;
- lastChild: retourne le dernier fils du noeud courant;
- nextSibling: retourne le prochain noeud qui suit immédiatement le noeud courant.
 Deux noeuds sont «siblings», ou cousin, s'ils ont le même parent;
- parentNode: retourne le noeud parent d'un noeud (pour le noeud racine, renvoie Null);
- text: retourne le contenu texte d'un noeud et de tous ses enfants
- xml: retourne le contenu XML d'un noeud et de tous ses enfants;
- hasChildNodes: retourne vrai ou faux suivant que le noeud possède ou non des enfants;
- nextNode: retourne le prochain noeud d'une collection (cette méthode peut être utilisée avec d'autres pour le parcours du DOM);
- previousNode: retourne le noeud précédent d'une collection;
- selectNodes: retourne un objet «NodeList» qui contient tous les noeuds correspondant un motif donné, «pattern matching»;
- o selectSingleNode: retourne un objet «Node» pour le premier noeud enfant qui correspond à un motif donné.





XML DOM 131

Exemple en JavaScript:

```
for each x in xmlDocument.documentElement.childNodes
document.write("<b>" & x.nodename & ":</b> ");
document.write(x.text);
document.write("<br />");
next
```

Sur le document XML suivant :

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <response>
3 <companyname>Ma compagnie à moi</companyname>
4 <telephone>05 55 43 69 83</telephone>
5 </response>
```



Le point sur XML, AJAX et les navigateurs

Tout dépend du navigateur employé!

- ▷ Pour la requête Ajax :
 - ♦ la réalisation de la requête AJAX : un objet différent dans Explorer et Firefox ;
 - la gestion des événements asynchrones ou non liès à la requête;
 - la détection du succès de la requête;
- ▶ Pour l'exploitation des données XML reçues :
 - une API de parcours du DOM qui ne fonctionne bien que sous... et mal sous... (mettre le nom de son navigateur favori et celui de son concurrent);
 - complexité d'accès aux éléments de l'arborescence XML;
 - Que faire ensuite de ces éléments récupérés ? Comment les intégrer dans la page HTML ?

La solution

- ne pas dépendre, et ne pas tenir compte, des spécificités des différents navigateurs;
- manipuler tous les éléments constitutifs de la page HTML;
- simplifier la réalisation de requête AJAX et faciliter l'accès aux données reçues (XML, JSON, HTML, text, javascript etc.)
- s'intégrer naturellement dans Javascript;
- que tout soit simple d'utilisation...
- ⇒ Utiliser JQuery!



JQuery: la bibliothèque qui vient au secours d'Ajax

Qu'est-ce que JQuery?

Une bibliothèque Javascript d'environ 19Ko compressé (120ko décompressé) à ajouter à une page HTML:

- qui normalise l'utilisation des différents navigateurs;
- qui bénéficie d'une communauté et d'une forte adoption : Amazon, IBM, Nokia, Google, Microsoft, etc.
- o qui permet de disposer d'une interface évoluée, «JQuery UI», dans le navigateur;
- o qui utilise les sélecteurs de CSS v3 pour intervenir sur les éléments du DOM;
- qui possède de nombreux plugins;
- qui est simple, élégante, bien documentée : elleplait énormément...
- qui permet d'ajouter de nombreux effets dynamiques à une page HTML:
 - des intéractions avec l'utilisateur avec les événements liés à la navigation (c'est du Javascript);
 - des effets d'animations graphiques de qualités : fading, Sliding, etc.
 - l'apparition/masquage et l'ajout/retrait d'éléments dans la page HTML;
 - la manipulation du CSS: ajout de classe à un élément, changement de valeur d'attributs etc.
 - la gestion d'Ajax, d'XML.
- qui repose sur les 3 opérations suivantes :
 - la sélection d'éléments du DOM via le CSS;
 - l'application d'opération sur ces éléments au travers de méthodes proposées par JQuery;
 - ⋄ le chaînage d'opérations multiples sur ces éléments et l'utilisation d'itération automatique sur ces éléments.

	Paramètres	Action	Sélecteur
changer la couleur des paragraphes	('color','red');	.css	JQuery('p')
avec le raccourci «\$»	('color', 'red');	.css	\$('p')

JQuery: la bibliothèque qui vient au secours d'Ajax

L'intégration de JQuery dans la page HTML

On peut inclure JQuery en le téléchargeant depuis le CDN de Google (Content Distribution Network) (avantages : rapidité, et mutualisation avec d'autres pages et ou d'autres sites) :

Sélection des éléments et ajout dynamique de contenu HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"</pre>
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
   <html>
   <head>
   <script type="text/JavaScript"</pre>
     src="http://ajax.qoogleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
   </head>
  <body>
   <div id="contents"></div>
10 <script>
11 var contenu = $('<h1> Un contenu de plus </h1>');
12 \$ (contenu).appendTo('#contents');
13 \$ ('#contents').css({'background-color':'yellow','width':'400px'});
14 </script>
15 </body>
16 </html>
```

- ligne 9 : on crée un contenu HTML à partir de texte.
- o ligne 10: on ajoute ce contenu au div qui a pour id contents
- ligne 11 : on modifie le CSS associé au div.

Un contenu de plus



La requête Ajax

```
En JQuery: $.ajax(url, [paramètres]);
```

	Paramètre	Description	
	url	L'url de la ressource	
	type	La méthode HTTP utilisée pour réaliser la requête (GET ou POST en général)	
	data	Les données à envoyer dans la requête	
	dataType	Le type de données attendues en retour de la requête : XML, HTML, script, JSON, text	
success(data,textStatus,jqXHR)		Une fonction à appeler en cas de succès de la requête	
error (jqXHR, textStatus, errorThrown)		en cas d'échec	
	complete(jqXHR,textStatus)	lorsque la requête est finie	
	timeout	délai pour la réalisation de la requête	
1	\$.ajax({		
2	type: 'GET',		
3	url: 'books.xml',		
4	<pre>dataType: 'xml',</pre>		
5	5 success: function(donnees, textStatus) {		
6	<pre>livre = \$(donnees).find('author').text(); \$('#contents').html(livre); },</pre>		
7	7 error: function(xhr, textStatus, errorThrown) {		
8	alert('An error occurred! ' + (errorThrown ? errorThrown : xhr.status));		
9	}		
10	});		

La requête Ajax

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"</pre>
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
   <html>
   <head><script type="text/JavaScript"</pre>
     src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script></head>
   <body>
   <h2>Contenu fourni par Ajax</h2>
   <div ID="contents"></div>
   <input type="button" id="requete" value="Ajax !">
10
11
   <script>
   function traiter_livre(index, valeur) {
13
     $ ('#liste_livres').append(''+$ (valeur).text()+''); }
14
   function traiter livres (donnees)
16
       $('').appendTo('#contents');
17
       $.each(donnees, traiter livre); }
18
     $('#requete').click(function(){
19
       $.ajax({
20
         type: 'GET',
21
        url: 'books.xml',
22
         dataType: 'xml',
         success: function(donnees, textStatus) {
24
               traiter_livres($(donnees).find('author'));
25
26
         error: function(xhr, textStatus, errorThrown) {
27
           alert('Une erreur s'est produite! ' + ( errorThrown ? errorThrown : xhr.status )); }
28
       });
29
     });
30 </script>
31
   </body>
32 </html>
```



Analyse syntaxique & Sémantique — P-FB

Au chargement de la page :

Contenu fourni par Ajax

Ajax !

Après appui sur le bouton «Ajax!»:

Contenu fourni par Ajax

- · Isaac Asimov
- · David Brin
- Neal Stephenson

Ajax !

Explications du code proposé sur le transparent précédent

- ligne 6: un div est défini avec un identifiant unique pour le sélectionner facilement contents;
- o ligne 7: un bouton est ajouté pour déclencher la récupération du document XML;
- ligne 13: on définie une fonction pour traiter la liste complète des éléments «author» du fichier XML, en créant un contenu HTML dynamiquement basé sur une liste non ordonnée «ul»;
- o ligne 10: on définie une fonction pour traiter un élément author: on l'ajoute à la liste comme nouvel élément;
- o ligne 17: on associe au bouton une fonction qui réalise la récupération du fichier XML:
 - ligne 19: utilisation de la méthode HTTP «GET»;
 - ♦ ligne 20 : url de récupération du fichier XML, en chemin relatif par rapport au serveur d'où on a récupéré le document HTML initial ;
 - ligne 21: on indique attendre un document au format XML;
 - ligne 22: en cas de succès de la requête Ajax, on traite son résultat à l'aide de la fonction définie en ligne 13;
 - ♦ ligne 25 : en cas d'erreur, on affiche un message d'erreur à l'utilisateur dans une fenêtre modale.

Attention à ne pas rappuyer plus d'une fois sur le bouton...



COMPLÉMENTINT



138

16 XML & SOAP

SOAP, «Simple Object Access Protocol»

SOAP est un format de message en XML permettant d'échanger des informations à l'aide du protocole de transport HTTP.

Il est utilisé dans le cadre des applications distibuées avec des échanges de type client/serveur.

Il est constitué:

- b d'une en-tête contenant des infos à destination du serveur, comme un numéro de message, une date d'expiration ou de requête etc.
- ▷ d'un corps où se trouve la requête ou la réponse.

Pour la requête :

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
3 <SOAP-ENV:Header SOAP-ENV:mustUnderstand="1">
4 <requestnum>8312</requestnum>
5 <date>11/05/03</date>
6 </SOAP-ENV:Header>
7
8 <SOAP-ENV:Body>
9 <m:media xmlns: m="http://videoondemandplace.com">
10 <m:name>My documentary</name>
11 <m:length>120m</length>
12 <m:codec>DivX</codec>
13 </m:media>
4 </SOAP-ENV:Body>
</SOAP-ENV:Body>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```







XML & SOAP

La réponse associée:

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
3 <SOAP-ENV:Body>
4 <SOAP-ENV:Fault>
5 <faultcode> SOAP-ENV: MustUnderstand</faultcode>
6 <faultstring> Did not understand Header</faultstring>
7 </SOAP-ENV:Fault>
8 </SOAP-ENV:Body>
9 </SOAP-ENV:Envelope>
```

Par exemple, ici, le serveur renvoie un message d'erreur. Complément

Plan

- □ Disposer d'un serveur Web léger;
- Quelques notions de JavaScript.
- □ JQuery et le DOM :
 - notion de «classe» et d'«id»: sélection d'élément;
 - modification du DOM: ajouter du contenu HTML, modifier des attributs...
 - interactivité;
- les éléments de formulaires : cases à cocher, saisie clavier.



Pour l'environnement de développement

Accès HTTP

Il est très important d'accéder au contenu du site web par réseau, service «http://», et non par le système de fichier, service «file://».

Pour cela, il faut utiliser un serveur Web léger.

Serveur Web léger





On pourra ensuite s'y connecter dans son navigateur avec l'URL suivante :

«http://localhost:8080/»



Programmation Javascript

Déclaration de variable

```
1 var ma_variable = "";
```

La variable n'a pas de type préalable, il dépend de ce quelle contient. Ici, une chaîne de caractères vide.

Un tableau commence à l'indice zéro.

3 var resultat = false;

2|var i = 0;

1 var chaine = "ABCDEFGHIJ";

Manipulation de chaîne de caractères

```
var c1 = "un";
var c2 = "mot";
var c3 = c1 + " " + c2;
var taille = c3.length;
La concaténation et l'obtention de la taille avec l'attribut lend
```

```
La concaténation et l'obtention de la taille avec l'attribut lengt 5 if (chaine[i] == 'E')
6 { resultat = true; }
```

La chaîne de caractères peut être traitée comme un tableau.

4 for (i=0; i < chaine.length; i++)

Déclaration de fonction

```
1 function ma_fonction() {
2 //
3 }
```

```
1 var une_fonction = function () {
2 //
3 }
4 une_fonction();
```

Ici, la fonction est anonyme.



Intégrer JQuery 143

JQuery se présente comme une bibliothèque à intégrer dans le document HTML:

```
1 < script src="js/jquery-3.6.4.min.js">
                                           1 < script src="js/jquery-3.6.4.min.js">
                                             </script>
 </script>
3
     <script>
                                           3 <script>
4
     $ (document).ready(function() {
                                           4|$(function() {
        // Votre programme est ici
                                                  // Votre programme est ici
6
      });
                                                });
 </script>
                                             </script>
```

La méthodologie en deux étapes

1. Sélectionner un ou des éléments : un «tag» en complément d'une «classe» ou par un «id» :

```
a . menu #CONTENU renvoie la liste des «tag anchors» de classe «menu» renvoie l'élément d'«id» «CONTENU»
```

- 2. Modifier l'élément :
 - changer la valeur d'un attribut;
 - ajouter un nouveau contenu;
 - supprimer un élément;
 - récupérer des informations depuis l'élément;
 - ajouter/supprimer une classe;
 - ♦ obtenir un «événement» à partir de l'élément.

Il est possible d'enchaîner les opérations :

```
$('#popUp').width(300).height(300);
```

Le retour d'un appel à JQuery fournit un «objet» capable de supporter des opérations liées à JQuery.



Utilisation de JQuery

□ Sélectionner un élément : Ceci est une introduction \$('.description') Ceci est une introduction \$('#contenu') □ modifier le HTML : \$('#erreurs').html('2 erreurs dans le formulaire.'); <div id="conteneur"> <div id="erreurs"> On aurait aussi pu utiliser . append () pour ajouter le contenu HTML à celui <h2>Erreurs:</h2> déjà existant. </div> </div> □ Supprimer du contenu: \$ ('#popup').remove() ou le remplacer avec replaceWith(); ☐ Ajouter/supprimer une classe: \$('#alertBox').removeClass('highlight'); ou alterner avec toggleClass(); □ modifier le CSS avec .css: \$('p.highlight').css('border', 'lpx solid black'); □ modifier un attribut avec .attr: \\$('#banniere img').attr('src', 'images/deux.png'); pour un champ de formulaire: | <input type="text" id="saisie" > obtenir la valeur du champ: positionner la valeur du champ: var ma_valeur = \$('#saisie').val(); \$('#saisie').val('toto@unilim.fr');

Utilisation de JQuery

- □ gérer des événements:
 - le clic de souris: |\$('#menu').click();
 - \$ le survol par la souris: \$('a').mouseover();
 \$('#menu').mouseover(function() {
 \$('#submenu').show();
 });

Gérer un menu et faire apparaitre un sous menu.

- pour un champ de formulaire :
 - ▷ blur (): lorsque le champ du formulaire est déselectionner;
 - ▶ focus () : lorsque le champ du formulaire est sélectionné ;
 - ▷ change (): lorsque son contenu a changé.

Les opérations avancées

□ la sélection conditionnelle :

lci, on sélectionne l'«input» suivant la condition «est de type "text"» et on change sa valeur.

□ La modification de propriétés :

```
<input type="checkbox"> $ ('input[type="checkbox"]').prop('checked', true);
<input type="text"> $ ('input[type="text"]').prop('disabled', true);
```

lci, on change l'état coché ou non coché avec le checked, la zone d'entrée en désactivée ou non avec di sabled.

□ l'utilisation des événements clavier :

Ici, on active le bouton «Envoyer» uniquement lorsque la longueur de la donnée saisie est supérieure à 2.

