Analyse
Première phase : Analyse Léxicale

Reconnaître le "vocabulaire" : les **unités lexicales (*token* ou symbole)**

Tout d'abord disposer :
- de l'alphabet *terminal* (V_T) du langage composé des symboles {le, chat} et
- de l'alphabet *auxiliaire(non-terminal)* (V_N) du langage composé des symboles {article, nom_commun, syntagme_Nom}
- S = *syntagme_Nom* : symbole auxiliaire initial ou axiome

Ensuite :
1. identifie tous les tokens qui appartiennent au langage ;
2. repère et isole toutes les autres parties du texte ;
3. fournit une suite plate de tokens reconnus.

Exercice : définir les alphabets, tokéniser et fournir la liste plate des tokens reconnus

$3 - x + y$ /* toto */
/ toto / $y + x - 3$

```
constante opérateur identificateur commentaire alpha_num
```


Analyse Lexicale

Le rôle d'un analyseur lexicale est de :
- lire un flux de caractères en entrée ;
- découper le flux en unités lexicales conformément à la définition du langage ;
- transmettre à l'analyseur syntaxique les unités élémentaires à plat.

Analyse
Deuxième phase : Analyse syntaxique

Tout d'abord disposer :
- P l'ensemble fini des règles de production du type : $A \rightarrow xB$ ou $A \rightarrow x$
  avec A un symbole de l'alphabet auxiliaire($\in V_N$) et x un mot terminal

  syntagme_Nom $\rightarrow$ *article* *nom_commun*
  article $\rightarrow$ le
  nom_commun $\rightarrow$ chat

Ensuite :
1. respect l'ordre des unités lexicales ;
2. les regroupe en structures grammaticales ;
3. produit un arbre syntaxique.

Exercice, quel serait P et l'arbre syntaxique de :

$3 - x + y$

Analyse
Deuxième phase : Analyse syntaxique

1. respect l'ordre des unités lexicales ;
2. les regroupe en structures grammaticales ;
3. produit un arbre syntaxique.

Exercice, quel serait P et l'arbre syntaxique de :

$3 - x + y$

$$
\begin{array}{rcl}
S & \rightarrow & Expression\ Constante | Identificateur \\
Expression & \rightarrow & Constante | Identificateur\ Operateur\ (Expression) \\
Operateur & \rightarrow & + | - \\
Constante & \rightarrow & /^{\wedge}[0-9][0-9]*((,|.)[0-9]+)*/ \\
Identificateur & \rightarrow & /^{\wedge}[A-Za-z]([A-Za-z])*/
\end{array}
$$

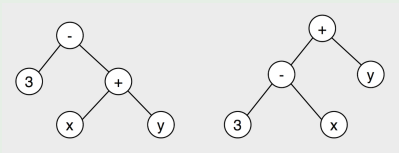
Analyse
Deuxième phase : Analyse syntaxique

1. respect l'ordre des unités lexicales ;
2. les regroupe en structures grammaticales ;
3. produit un arbre syntaxique.

Exercice, quel serait P et l'arbre syntaxique de :

$3 - x + y$

On obtenir ces 2 arbres syntaxiques :



(1) Pourquoi, (2) sont-ils syntaxiquement valident et (3) le sont-ils sémantiquement ?

Analyse
Deuxième phase : Analyse syntaxique

1. respect l'ordre des unités lexicales ;
2. les regroupe en structures grammaticales ;
3. produit un arbre syntaxique.

Exercice, quel serait P et l'arbre syntaxique de :

$3 * x + y$

Analyse
Deuxième phase : Analyse syntaxique

1. respect l'ordre des unités lexicales ;
2. les regroupe en structures grammaticales ;
3. produit un arbre syntaxique.

Exercice, quel serait P et l'arbre syntaxique de :

$3 * x + y$

$$
\begin{array}{rcl}
S & \rightarrow & Expression\ Constante|Identificateur \\
Expression & \rightarrow & Constante|Identificateur\ Operateur\ (Expression) \\
Operateur & \rightarrow & +|-|* \\
Constante & \rightarrow & /\hat{}[0-9][0-9]*((,|.)[0-9]+)*/ \\
Identificateur & \rightarrow & /\hat{}[A-Za-z]([A-Za-z])*/
\end{array}
$$

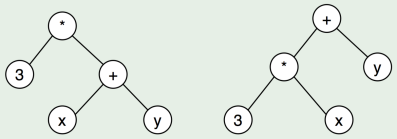
Analyse
Deuxième phase : Analyse syntaxique

1. respect l'ordre des unités lexicales ;
2. les regroupe en structures grammaticales ;
3. produit un arbre syntaxique.

Exercice, quel serait P et l'arbre syntaxique de :

$3 * x + y$

On obtiendra ces 2 arbres syntaxiques



(1) Pourquoi, (2) sont-ils syntaxiquement valident et (3) le sont-ils sémantiquement ?

Analyse Syntaxique

L'analyse syntaxique explicite la structure du code source grâce à une représentation en arbre, chaque nœud de cette arbre correspond à un opérateur et ses fils aux opérandes sur lesquels il agit.

Construction de l'arbre syntaxique :

- un code source est une très longue chaîne de caractères
- dont chaque élément (de la chaîne) est un des symboles (du code source)
- chaque symbole doit être placé sur un feuille d'un arbre conformément à une grammaire

Analyse sémantique

Troisième phase : Analyse contextuelle

Opèrer certains contrôles par rapport à un contexte :

- la résolution des noms et détection des ambiguïtés (ex : nom identique pour deux identificateurs d'un même bloc),
- la vérification les types (ex : compatibilité des types dans les instructions et expressions),
- la priorisation ...

Transformation

Quatrième phase : Transformation

Deux situations différentes

1. Traduction de code, c-à-d passage d'un langage source vers un langage cible
2. Génération ou optimisation de code (ex : remplacer des instructions générales par des instructions plus adaptées). Ne concerne pas cette UE...

Bilan

Synthèse

les 2 étapes de la compilation		Outils théoriques utilisés
Phase d'analyse	analyse lexicale	expressions régulières automates à états finis
	analyse syntaxique *parser*	grammaires algébriques automates à pile
	analyse sémantique	grammaires contextuelles
Phase de production	génération de code	grammaires contextuelles...
	optimisation de code	Ne concerne pas cette UE...

Exercices

Étant le source SVG suivant :

```
<svg xmlns="http://www.w3.org/2000/svg">
<g stroke="red" > <line x2="18" y2="-20" x1="10" y1="15" /> </g>
</svg>
```

Exercice :

1. Déduire l'alphabet *terminal*. Découper en token en expliquant la logique de découpage choisie.
2. Déduire l'alphabet *auxiliaire*. Catégoriser et nommer les différentes unités lexicales.

Exercices

Étant le source SVG suivant :

```
<svg xmlns="http://www.w3.org/2000/svg">
<g stroke="red" > <line x2="18" y2="-20" x1="10" y1="15" /> </g>
</svg>
```

Exemple d'un possible découpage en tokens

```
<
svg
xmlns
=
"
http://www.w3.org/2000/svg
"
>
<
g
stroke
=
"
red
"      .....
</
svg
>
```


Exercices

Exercice : Étant le source SVG suivant :

```
<svg xmlns="http://www.w3.org/2000/svg">
<g stroke="red" > <line x2="18" y2="-20" x1="10" y1="15" /> </g>
</svg>
```

Les mots du langage ?

alphabet terminal : $<$, $=$, ", $>$, $</$, $/>$, ...
alphabet auxiliaire : *Const*, *AlphaNum*, $Balise_O$, $Balise_F$, $Balise_V$ ...

Exercices

Exercice : Étant le source SVG suivant :

```
<svg xmlns="http://www.w3.org/2000/svg">
<g stroke="red" > <line x2="18" y2="-20" x1="10" y1="15" /> </g>
</svg>
```

Les règles de constructions ?

$Balise_V \rightarrow \ldots$
$Balise_F \rightarrow \ldots$
$Balise_O \rightarrow \ldots$
$Const \rightarrow \ldots$
$AlphaNum \rightarrow \ldots$

Exercices

Exercice : Reécrire les 2 sources suivant en pseudo-code puis construire l'arbre de syntaxe abstraite.

Code C

```
int a, b, q, r;
main () {
scanf("% d", & a);
scanf("% d", & b); q = 0; r = a;
while (r >= b) {
q = q+1; r = r-b;
}
printf("% d", q); printf("% d", r);
}
```

Code Ada

```
procedure Main is
a, b, q, r : Integer;
begin get(a); get(b); q := 0; r := a;
while r >= b loop q := q+1; r := r-b; end loop;
put(q); put(r);
end Main;
```

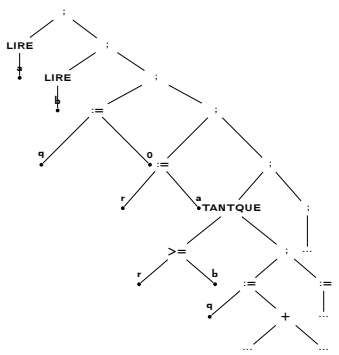

Analyse Syntaxique

En pseudo-code

```
LIRE a; LIRE b;
q := 0; r := a;
TANTQUE r >= b FAIRE
q := q+1; r := r-b
FINTQ;
ECRIRE q; ECRIRE r;
```


Analyse Syntaxique

L'arbre de syntaxe abstraite



Produisons un DSL en syntaxe XML pour le traduire...