--Log3430 -- Méthodes de test et de validation du logiciel

Noureddine Kerzazi |
Noureddine.Kerzazi@polymtl.ca

Bram Adams |
bram.adams@polymtl.ca

Tests basés sur les états

Lab #4

Objectifs : **Tests des états**

- Représenter le fonctionnement d'un parseur avec un diagramme d'états
- Générer l'arbre des transitions à partir d'un diagramme d'états.
- Identifier les cas de test et les implémenter à l'aide de Unittest
- .

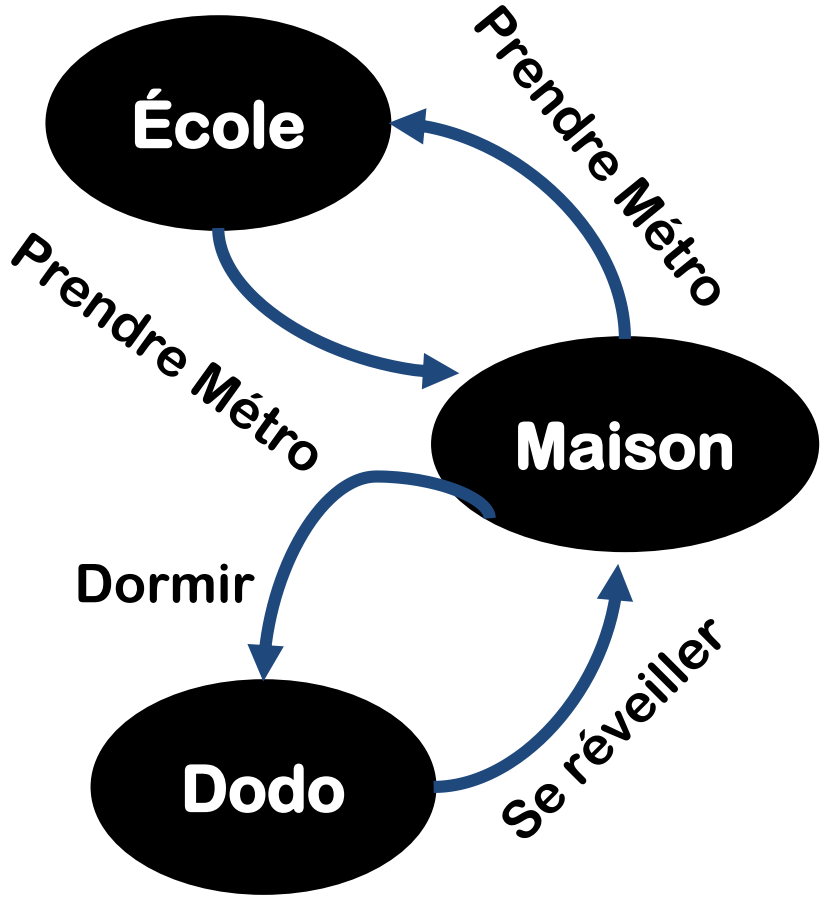
Agenda

- Retour sur le Lab# 3
- Contexte: Machine à états finis
- Pre/post conditions
- Démo du code source (Python)
- Lab #4
- À vous !!!
- Consignes de remise.

Machine à états finis

- Cet exemple montre comment fonctionnent les transitions.
- Chaque transition change l'état.
- Cependant, pas tous les états sont accessibles à partir de tous les états.
- Par exemple, vous ne pouvez pas dormir à l'école.

RÉVEILLEZ-VOUS!



Exemple

```
a = 3 + 4 * 5
print(a)
```

lexing

parsing

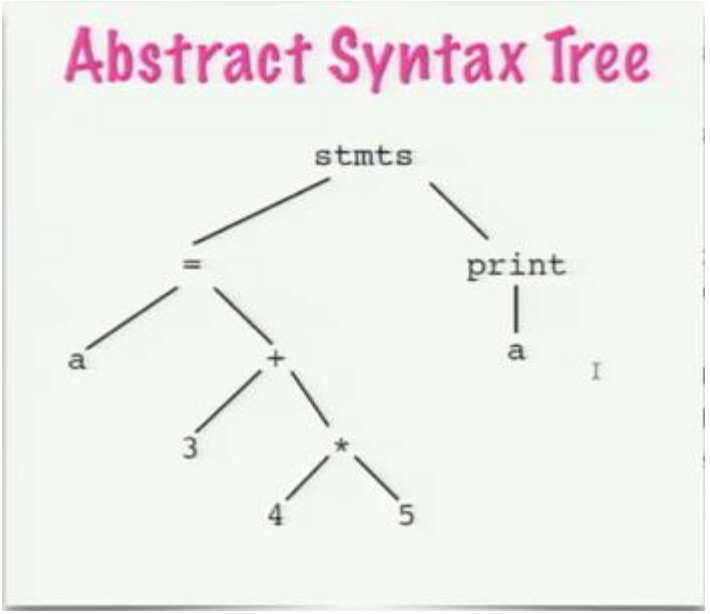
```
ID : '[a-zA-Z_][a-zA-Z0-9_]+'
NUM : '[0-9]+'
PLUS : '\+'
TIMES : '\*'
EQ : '='
LPAREN : '\('
RPAREN : '\)'
PRINT : 'print'
```

```
program : stmts

stmts : stmts stmt
      | stmt

stmt : ID EQ expr
     | PRINT LPAREN expr RPAREN

expr : expr PLUS expr
     | expr TIMES expr
     | LPAREN expr RPAREN
     | ID
     | NUM
```



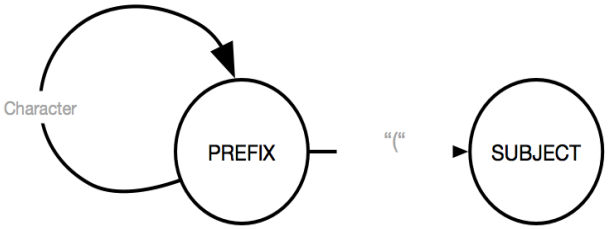
Du parseur vers le compilateur

parsery

```
%union {
   int ival;
   char *name;
}
%token <ival> NUM
%token <name> ID
%token PRINT
%left '+'
%right '*'
%type <ival> expr

statements: statement '\n'
          | statements statement '\n'
          ;
statement:  ID '=' expr { store($1, $3); }
         |  PRINT '(' expr ')' { printf("%d\n", $3); }
         ;

expr: expr '+' expr { $$ = $1 + $3; }
    | expr '*' expr { $$ = $1 * $3; }
    | '(' expr ')' { $$ = $2; }
    | ID { $$ = lookup($1); }
    | NUM { $$ = $1; }
    ;
```



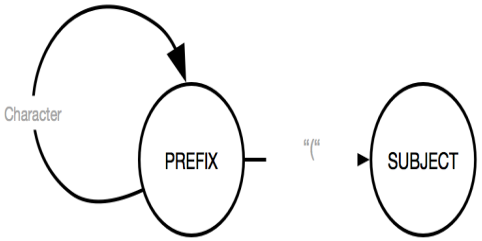
- Il existe plusieurs parseurs solides comme Lex/yacc et PLY
- Il s'agit ici de tester un nouveau micro parseur

Comment ça marche ?

- **Soit la ligne : print(a)**
- **Il y a un préfixe comme "print" puis un jeu de parenthèses.**
- **À l'intérieur de la parenthèse est un sujet**
- **Chacun de ces préfixes + (+ sujet +) peut appeler une règle**
- **Il peut y avoir un opérateur qui enchaîne des règles comme «&» ou «|»**
- **Parfois, il y a des groupes de choses également encapsulées par une parenthèse.**
- **si on lit de gauche à droite, caractère par caractère, on peut observer d'un caractère au suivant lesquels vont modifier l'état**

Dessinez un diagramme d'état

- Si on commence simplement par un état et qu'on se demande où est ce qu'on peut transiter à partir d'ici?
- Par exemple, si on commence par l'état PREFIX en regardant le tout premier caractère du premier exemple, le «p» (dans «print(a)»), on vois qu'on ne peux aller que vers deux endroits :
  - 1) vers un autre caractère, dans ce cas, le 'r'
  - 2) vers le premier '(' parenthèse.



Exploration du code

Exemple

```
example = "print(bonjour log3430)"
```

```
p -> STATE: NEW_GROUP : STATE: PREFIX
r -> STATE: PREFIX : STATE: PREFIX
i -> STATE: PREFIX : STATE: PREFIX
n -> STATE: PREFIX : STATE: PREFIX
t -> STATE: PREFIX : STATE: PREFIX
( -> STATE: PREFIX : STATE: SUBJECT
b -> STATE: SUBJECT : STATE: SUBJECT
o -> STATE: SUBJECT : STATE: SUBJECT
n -> STATE: SUBJECT : STATE: SUBJECT
j -> STATE: SUBJECT : STATE: SUBJECT
o -> STATE: SUBJECT : STATE: SUBJECT
u -> STATE: SUBJECT : STATE: SUBJECT
r -> STATE: SUBJECT : STATE: SUBJECT
  -> STATE: SUBJECT : STATE: SUBJECT
l -> STATE: SUBJECT : STATE: SUBJECT
o -> STATE: SUBJECT : STATE: SUBJECT
g -> STATE: SUBJECT : STATE: SUBJECT
3 -> STATE: SUBJECT : STATE: SUBJECT
4 -> STATE: SUBJECT : STATE: SUBJECT
3 -> STATE: SUBJECT : STATE: SUBJECT
0 -> STATE: SUBJECT : STATE: SUBJECT
) -> STATE: SUBJECT : STATE: END_RULE
----------------------
<RuleGroup: {'op': None, 'parent': None, 'level': 0, 'rule_count': 1, 'rules': [<Rule:  print(bonjour log3430)>]}>
```

Exemple 2

```
str = "print(bonjour (log3430)) !"
```

```
( -> STATE: SUBJECT : STATE: SUBJECT
l -> STATE: SUBJECT : STATE: SUBJECT
o -> STATE: SUBJECT : STATE: SUBJECT
g -> STATE: SUBJECT : STATE: SUBJECT
3 -> STATE: SUBJECT : STATE: SUBJECT
4 -> STATE: SUBJECT : STATE: SUBJECT
3 -> STATE: SUBJECT : STATE: SUBJECT
0 -> STATE: SUBJECT : STATE: SUBJECT
) -> STATE: SUBJECT : STATE: END_RULE
) -> STATE: END_RULE : STATE: END_GROUP
skip ' ' in STATE: END_GROUP
skip '!' in STATE: END_GROUP
----------------------
<RuleGroup: {'op': None, 'parent': <RuleGroup: {'op': None, 'parent': None, 'level': 0, 'rule_count': 1, 'rules': [<Rule:  print(bonjour (log3430)>]}>, 'level': 1, 'rule_count': 1,
```

À vous !!!

C’est fini !!!

Merci !

Questions?

- noureddine.kerzazi@polymtl.ca

Remise .Zip bien commenté d’ici 2 semaines