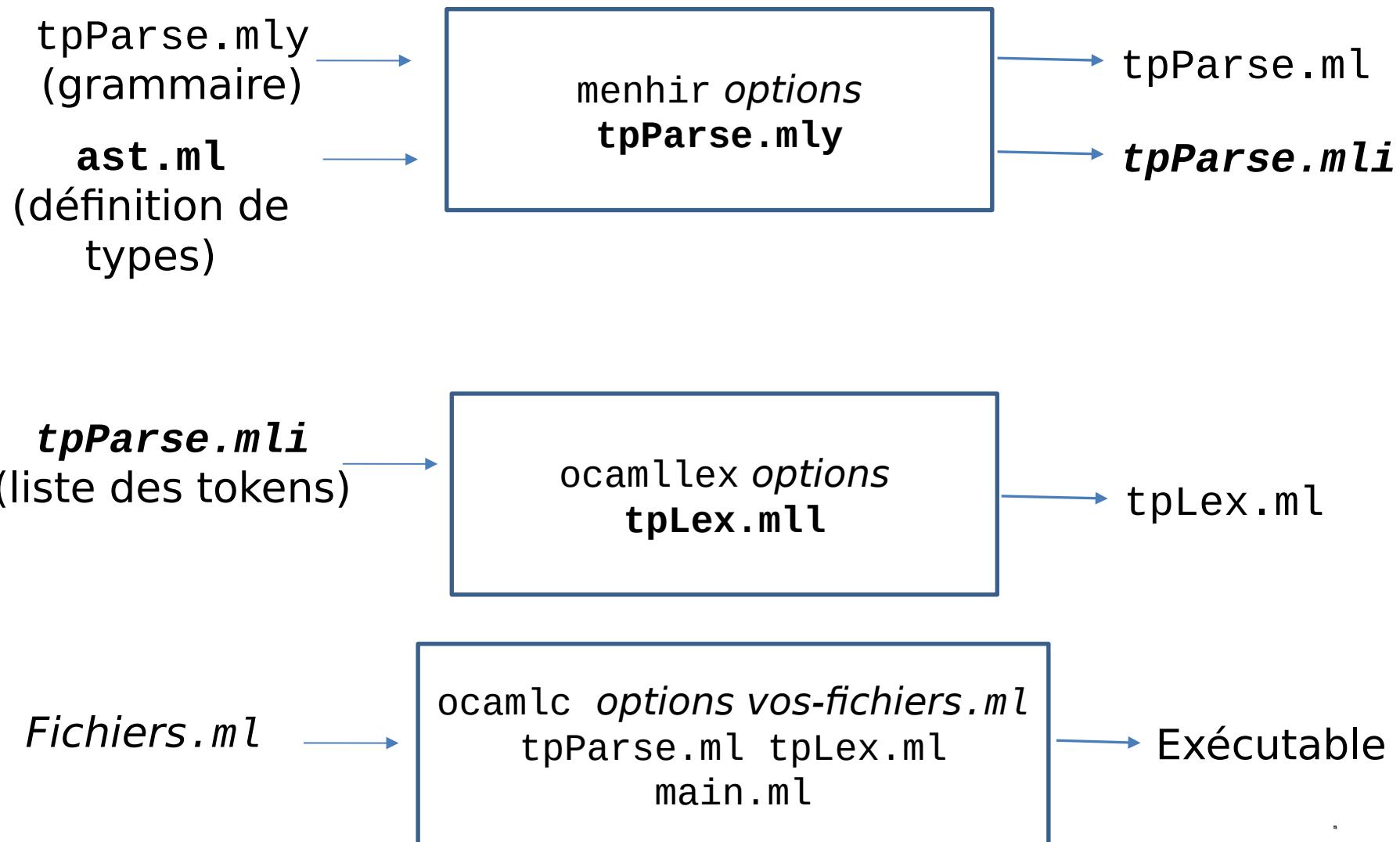


Menhir : un générateur d'analyseur syntaxique

- Semblable à d'autres générateurs d'analyseurs syntaxiques (Yacc/Bison) , avec quelques particularités
- Couplé à ocamllex pour réaliser la partie « front-end » d'un compilateur
- Générateur d'analyse syntaxique de type LR + actions associées aux productions. Les actions sont exécutées au moment des **réductions**.
- Gère une pile de « valeurs » en parallèle de la pile d'analyse syntaxique. Ces valeurs correspondent à un (unique) **attribut synthétisé**.



Schéma général



Première étape : définir les nœuds du futur AST, *i.e* définir des types ocaml (fichier **ast.ml**)

```
type opComp = Eq | Neq | Lt | Le | Gt | Ge
```

```
type expType =
  Id      of string
| Cste   of int
| Plus   of expType*expType
| Minus  of expType*expType
| Times  of expType*expType
| Div    of expType*expType
| Comp   of opComp*expType*expType
```

...

```
type instr =
  Ite of expType*instr*instr
| Assign of string*expType
type decl = ...
```

*if_then_else
affectation*

Structure d'un fichier .mly (début)

```
{  
  (* code CAML copié en tête du fichier produit *)
```

```
open Ast
```

```
...
```

```
}
```

Type ocaml de la valeur lexicale
du token

```
%token <string> ID STRING
```

```
%token <int> CSTE
```

```
%token <Ast.opComp> RELOP
```

```
%token PLUS MINUS TIMES DIV
```

```
%token IF THEN ELSE
```

```
%token EOF
```

```
%nonassoc RELOP /* lowest precedence */
```

```
%left PLUS MINUS
```

```
%left TIMES DIV /* highest precedence */
```

Indications de
préférence
et associativité

Structure d'un fichier .mly (suite)

```
%type <int> axiome  
%type <expType> expr  
%type <instr> instr  
%type <decl> declaration
```

...

```
%start<Ast.progType> axiome
```

```
%%
```

```
axiome: ld = list(declaration) li = list(instr) EOF
```

```
{
```

(* code ocaml associé à cette production.

ld sera de type decl list et li de type instr list

Exemple d'actions: imprimer l'AST du programme,

```
*
```

```
List.iter (fun d -> ...) ld;
```

```
List.iter (fun i -> ...) li
```

```
(ld, li) (* valeur retournée par l'analyseur *)
```

```
}
```



Type ocaml de la valeur associé au non-terminal

list: opérateur fourni par menhir pour simplifier l'écriture des productions.

Structure d'un fichier .mly (suite2)

Le rôle principal des actions des règles est de construire l'AST du programme source

declaration:

{

(* code ocaml associé à la partie déclarations de l'AST *)

}

expr:

x = ID { Id x }

| v = CSTE { Cste v }

| g = expr PLUS d = expr { Plus(g, d) }

| g = expr TIMES d = expr { Times(g, d) }

...

| e = **delimited**(LPAREN, expr, RPAREN) { e }

Prochainement ici : vos productions pour décrire la syntaxe d'une déclaration et construire la partie d'AST associée

Structure d'un fichier .mly (fin)

```
instr: x = ID ASSIGN e = expr      { Assign(x, e) }
| IF si=expr THEN alors=instr ELSE sinon=instr
{ Ite(si, alors, sinon) }
```

D'autres opérateurs de menhir: voir la doc en ligne

list(X)	$type_de_X$ list
nonempty_list(X)	
separated_list(<i>séparateur</i> , X)	
separated_nonempty_list(<i>séparateur</i> , X)	
delimited(<i>opening</i> , X , <i>closing</i>)	
option(X)	Some $type_de_X$ None
boption(X)	renvoie un booléen
loption(X)	renvoie une liste du type de X
append(X , Y)	pour des listes

Ces opérateurs se composent :

A : ... { valeur:monType }

LA : l = delimited(LPAREN,
 separated_nonempty_list(COMMA, A)
 RPAREN)

l sera une liste non vide de type monType list

Exercice: écrire les productions classiques pour LA