# Aide Mémoire Ocamllex & Ocamlyacc

Armelle Bonenfant

9 novembre 2010

**Résumé**

C  docum nt  st un  traduction/résumé d s docum ntations d  Ocamll x  t Ocamlyacc  n lign  :
http ://plus.kaist.ac.kr/ shoh/ocaml/ocamll x-ocamlyacc/ocamll x-tutorial  t
http ://plus.kaist.ac.kr/ shoh/ocaml/ocamll x-ocamlyacc/ocamlyacc-tutorial
(Copyright (C) 2004 SooHyoung Oh.).

Il prés nt  l s fonctionnalités d  Ocamll x  t d  Ocamlyacc.

# Table des matières

# Liste des tableaux

# Table des figures

# Chapitre 1

# Ocamllex

Ocamll x  st un outil d'OCaml pour génér r d s *scanners* : programm s qui r connaiss nt d s motifs l xicaux à l'aid  d  *descriptions* sous form  d  pair  (*expression régulière, règle*). Ocamll x génèr  un  xécutabl  qui  ff ctu  l'analys  l xical  t égal m nt l  cod  Ocaml corr spondant.

## 1.1 Commandes et structure

La command  pour lanc r l  l x r :

```
ocamll x program.mll
```

Fig. 1.1 – Lign  d  command

## 1.2 Input

L  fichi r d' ntré  (fig 1.2) d'Ocamll x conti nt 4 s ctions : *header, definitions, rules*  t *trailer*. Son  xt nsion  st `".mll"`.

Il faut r t nir qu  :

- *header*  t *rules* sont obligatoir s
- *header*  t *trailer*  ntr  accolad s, cod  Ocaml qui s ra r mis "t l qu l" dans l  fichi r ocaml d  sorti .
- *definitions* conti nt l s déclarations d s  xpr ssions régulièr s  t l urs id ntifiant.
- *rules* conti nt d s points d' ntré  pour définir l' ff t d  l'analys  l xical .

L s patt rns sont décrit dans la s ction suivant  1.3.

## 1.3 Patterns

L s motifs ou *patterns* ont l  format d'  xpr ssions régulièr s dans l  styl  d  l x à l'aid  d'un syntax  Caml-lik . La tabl  1.1 r pr nd c s  xpr ssions régulièr s.

```
(* header section *)
{ h ad r }

(* definitions section *)
let id nt = r g xp
let ...

(* rules section *)
rul   ntrypoint [arg1 ... argn] = pars
  | patt rn { action }
  | ...
  | patt rn { action }
and  ntrypoint [arg1 ... argn] = pars
  ...
and ...

(* trailer section *)
{ trail r }
```

FIG. 1.2 – Structur  du fichi r d' ntré  ".mll"

| | |
|---|---|
| 'c' | match th  charact r 'c' |
| _ | match any charact r |
| eof | match an  nd-of-fil |
| "foo" | match th  lit ral string "foo" |
| ['x' 'y' 'z'] | match s  ith r an 'x', a 'y', or a 'z' |
| ['a' 'b' 'j'–'o' 'Z'] | match s an 'a', a 'b', any l tt r from 'j' through 'o', or a 'Z' |
| [^ 'A'–'Z'] | a "n gat d charact r s t" |
| [^ 'A'–'Z' 'n'] | any charact r EXCEPT an upp rcas  l tt r or a n wlin |
| r* | z ro or mor  r's, wh r  r is any r gular  xpr ssion |
| r+ | on  or mor  r's, wh r  r is any r gular  xpr ssion |
| r? | "an optional r" |
| id nt | th  xpansion of th  "id nt" |
| (r) | par nth s s ar  us d to ov rrid  pr c d nc  (s  b low) |
| rs | "concat nation" |
| r\|s |  ith r an r or an s |
| r#s | match th  diff r nc  of th  two sp cifi d charact r s ts |
| r as id nt | bind th  string match d by r to id ntifi r id nt |

TAB. 1.1 – Tabl au d s motifs

L s  xpr ssions régulièr s cité s dans la tabl  1.1 sont trié s s lon l s priorités d  précéd nc  du plus fort au plus faibl  : '*'  t '+' ont la plus fort  précéd nc , puis '?', 'concat nation', '|',  t  nfin 'as'.

Par  x mpl , "foo" | "bar"* = ("foo")|("bar"*) = "foo" ou z ro-ou-plusi urs "bar",  t ("foo"|"bar")* = z ro-ou-plusi urs "foo"-ou-"bar" ( x : "foobarbarbarfoobar", "", "barbar"...)

R marqu  : l' xpr ssion n wlin  p ut êtr  r connu  par `"[^␣'A'-'Z']"` à moins qu  `"n"` soit  xplicit m nt prés nt dans la négation : `"[^␣'A'-'Z'␣'\n']"` (non standard).

## 1.4 Match

L  princip  st l  suivant : si il y a plusi urs patt rns qui corr spond nt à l' xpr ssion saisi , l  princip  du "plus long match"  st appliqué. C' st-à-dir  qu' ntr  l s patt rns, "ding", "dong"  t "dingdong", l' xpr ssion saisi  s ra associé  au patt rn "dingdong". S'il y a plusi urs patt rns d  longu ur égal , on pr nd l  pr mi r d  la list .

Lorsqu  l  motif corr spondant  st trouvé, l  t xt  corr spondant, app lé *token*  st considéré comm  disponibl  sous la form  d'un  chaîn  d  caractèr s. L'action corr spondant  au motif filtré  st alors  xécuté (voir d scription d s actions dans la s ction 1.5)  t l  r st  d  l'input  st scanné pour un motif suivant.

Si aucun  corr spondanc  n' st trouvé , l  scann r lèv  l' xc ption `"lexing : empty token"`.

Voici qu lqu s  x mpl s qui montr nt l  fonctionn m nt du filtrag .

L s d ux pr mi rs cas (1.3,1.4) illustr nt qu  lorsqu  d ux motifs p uv nt êtr  r connus, Ocaml x choisit l  pr mi r d  la list . Ainsi, dans l  d uxièm  cas (1.4), l  patt rn "ding"  st donc inutil .

```
(* rules section *)
rul  tok n = pars
  | "ding"      { print_ ndlin  "Ding" }                (* "ding"
    pattern *)
  | ['a'-'z']+ as word                                  (* "word"
    pattern *)
                { print_ ndlin  ("Word:␣" ^ word) }
  ...
```

Fig. 1.3 – Importanc  d  l'ordr  (1)

```
(* rules section *)
rul  tok n = pars
  | ['a'-'z']+ as word                          (* "word" pattern *)
                { print_ ndlin  ("Word:␣" ^ word) }
  | "ding"      { print_ ndlin  "Ding" }                (* "ding"
    pattern *)
  | ...
```

Fig. 1.4 – Importanc  d  l'ordr  (2)

Dans c  troisièm  x mpl  (1.5), on choisit l  plus long motif r connu. Il y a trois patt rns : ding, dong  t dingdong.

```
(* rules section *)
rul  tok n = pars
  | "ding"      { print_ ndlin  "Ding" }                  (* "ding"
     pattern *)
  | "dong"      { print_ ndlin  "Dong" }                  (* "dong"
     pattern *)
  | "dingdong"  { print_ ndlin  "Ding-Dong" }   (* "dingdong" pattern
     *)
  ...
```

FIG. 1.5 – Plus long motif r connu

Lorsqu "dingdong" st donné n input, d ux choix sont possibl s : soit l s motifs ding + dong soit l motif dingdong. Par l princip du plus long motif r connu, l motif dingdong st choisi.

Il st possibl d'appliqu r l princip du plus court motif r connu n r mplaçant l mot clé pars par shorst st. L princip du pr mi r d la list st cons rvé.

## 1.5 Action

Chaqu motif dans un règl a un action corr spondant , qui st un xpr ssion Ocaml. Par x mpl , voici un programm qui ffac tout s l s occurr nc s d "zap m " :

```
(* rules section *)
rul  tok n = pars
  | "zap_m "    { tok n  l xbuf }          (* ignore this token: no
     processing and continue *)
  | _ as c      { print_char c; tok n  l xbuf }
```

FIG. 1.6 – Suppr ssion d "zap m "

Ex mpl d'un programm qui compr ss l s spac s t tabulations multipl s n un s ul spac , t qui supprim un spac n fin d lign :

```
{}
rul  tok n = pars
  | [' ' '\t']+          { print_char ' '; tok n  l xbuf }
  | [' ' '\t']+ '\n'     { tok n  l xbuf }       (* ignore this token *)
```

FIG. 1.7 – Compr ssion

L s actions p uv nt inclur du cod Ocaml qui r nvoi un val ur. Chaqu fois qu la fonction d'analys ur l xical st app lé , ll continu d s'appliqu r sur l s *tokens* d puis l squ ls il a analysé n d rni r jusqu'à att indr la fin du fichi r.

R marqu : c qu fait l'analys ur, c' st simpl m nt d vérifi r qu l'input vérifi l s règl s l xical s, il n'y a pas d "résultat" à propr m nt parlé, mais simpl m nt, n cas d succès, l'assuranc qu l'analys ur r connaît l'input. L s actions p rm tt nt d'agir sur l s mots r connus au cours d l'analys .

L s actions sont évalué s après qu l *lexbuf* soit associé au buff r l xical courant t à l'id nti-fiant qui suit l mot clé (ou à la chaîn d caractèr s corr spondant ). L'utilisation du l xbuf st fourni par la librairi standard du modul L xing dont la tabl 1.2 prés nt qu lqu s xtraits.

| | |
|---|---|
| L xing.l x m l xbuf | Val ur d la chaîn r connu |
| L xing.l x m _char l xbuf n | nièm caractèr d la chaîn r connu (début à 0) |
| L xing.l x m _start l xbuf | Position d la chaîn d puis l début d l'input (début à 0) |
| L xing.l x m _ nd l xbuf | Position d la chaîn d puis la fin d l'input |
| L xing.l x m _start_p l xbuf | Position d typ position (voir doc v rsion Ocaml 3.08) |
| ntrypoint [ xp1 ... xpn] l xbuf | App ll un autr l x r sur l point d' ntré donné. l xbuf doit êtr l d rni r argum nt |

TAB. 1.2 – Qu lqu s fonctions du modul L xing

## 1.6 Output : le scanner généré

La sorti *output* généré st un fichi r `".ml"` du mêm nom qu c lui invoqué par Ocamll x . L fichi r généré conti nt l s fonctions d scann r, d s tabl s utilisé s par l scann r pour fair corr spondr l s tok ns t d s p tit s fonctions auxiliair s. L s fonctions d scann r sont déclaré s d la façon suivant (1.8) :

```
let ntrypoint [arg1 ... argn] l xbuf =
    ...
and ...
```

FIG. 1.8 – L (s) point(s) d' ntré

où la fonction a n + 1 argum nts. L s n argum nts provi nn nt d la définition d s règl s. La fonction scann r résultant néc ssit un argum nt supplém ntair , app lé l xbuf d typ L xing.l xbuf qui doit êtr l d rni r argum nt.

Lorsqu'un point d' ntré st app lé, il scann l s tok ns à partir d l'argum nt l xbuf. Quand il trouv un motif corr spondant, il xécut l s actions corr spondant s t r tourn l l xbuf (amputé du mot r connu). D façon à continu r l'analys l xical après l'évaluation d'un action, il faut app l r la fonction d scann r récursiv m nt.

## 1.7 Démarrage

Il st possibl d'activ r l s règl s d façon conditionn ll . Quand on v ut activ r l'autr règl , il suffit d'app l r l point d' ntré d l'autr fonction. Par x mpl , l'input suivant st constitué d d ux règl s, l'un vérifiant l s tok ns t l'autr "sautant" l s comm ntair s (1.9).

```
{}
rul   tok n = pars
  | [' ' '\t' '\n']+
        (* skip spaces *)
        { tok n l xbuf }
  | "(*"
        (* activate "comment" rule *)
        { comm nt l xbuf }
  ...
and comm nt = pars
  | "*)"
        (* go to the "token" rule *)
        { tok n l xbuf }
  | _
        (* skip comments *)
        { comm nt l xbuf }
  ...
```

FIG. 1.9 – Règl s conditionn ll s

Lorsqu l scann r généré r ncontr un comm ntair ouvrant "(*" à la règl tok n, il activ l'autr règl d comm ntair . Lorsqu'il r ncontr un comm ntair f rmant "*)" dans la règl comm nt, il r tourn à la règl tok n.

## 1.8 Interface a ec Ocamlyacc

L'un d s principaux usag s d Ocamll x st l'association av c son compagnon Ocamlyacc pars ur-générat ur. L s pars urs d'Ocamlyacc app ll nt l'un d s fonctions d scann r pour trouv r l tok n d' ntré suivant. La fonction st c nsé r nvoy r l typ du tok n suivant av c sa val ur associé . Pour utilis r Ocamll x av c Ocamlyacc , l s fonctions d scann r doit utilis r un modul d pars ur pour définir l s typ s d tok n, qui sont défini dans l s attributs '%tok n' apparaissant dans l'input d Ocamlyacc . Par x mpl , si l fichi r d' ntré d Ocamlyacc st pars .mly t qu l'un d s tok n st "NUMBER",un parti du scann r d vrait r ss mbl r à :

```
{
  op n Pars
}

rul   tok n = pars
  ...
  | ['0'-'9']+ as num { NUMBER (int_of_string num) }
  ...
```

FIG. 1.10 – Utilisation tok n l x - yacc

## 1.9 Options

Voici qu lqu s options pour Ocamll x (tabl 1.3) :

| | |
|---|---|
| −o output−fil | Chang l nom d la sorti |
| −ml | L'automat st codé av c d s fonctions ocaml plutôt qu'un automat built-in. Pour l débuggag . |
| −q | Pour supprim r l s m ssag s d'information. |

Tab. 1.3 – Options d'Ocamll x

# Chapitre 2

# Ocamlyacc

## 2.1 Introduction

Ocamlyacc st un générat ur d pars ur qui conv rti un grammair (typ LALR(1)) n un programm Ocaml qui pars c tt grammair . Quand on maîtris Ocamlyacc , on p ut l'utilis r pour construir d s pars rs d'un larg gamm d langag s, d la simpl calculatric au langag d programmation compl x .

Ocamlyacc st très proch d yacc (bison) qui sont répandus dans l s nvironn m nts d programmation C. La maîtris d'ocaml st néc ssair pour l'utilis r.

C tutori l st fait d chapitr s simpl s qui xpliqu nt l s conc pts d bas t prés nt nt qu lqu s x mpl s.

## 2.2 Concepts

### 2.2.1 Langages et grammaires hors-contexte (type 2)

Pour qu Ocamlyacc pars un langag , l langag doit êtr décrit par un grammair hors-cont xt (typ 2). C qui v ut dir qu'on doit spécifi r un ou plusi urs group s syntaxiqu s t donn r l s règl s d constructions corr spondant s. Par x mpl , l langag C, un sort d group syntaxiqu st app lé " xpr ssion". Un règl pour construir un xpr ssion p ut êtr "un xpr ssion p ut êtr construit à l'aid du sign - t d'un autr xpr ssion". Un autr p ut êtr "un xpr ssion p ut êtr un nti r". Comm on p ut l voir dans c s x mpl s, l s règl s sont souv nt récursiv s, mais il doit y avoir au moins un règl qui mèn hors d la récursion.

L systèm form l l plus commun pour prés nt r l s règl s d façon lisibl st la norm Backus-Naur Form ou "BNF". Tout grammair xprimé n BNF st un grammair hors-cont xt . L' ntré d'Ocamlyacc st ss nti ll m nt BNF.

S ul s l s grammair s LALR(1) p uv nt êtr traité s par Ocamlyacc . Pour simplifi r, on doit pouvoir pars r n'import qu l mot d l' ntré av c un s ul tok n d'avanc , c qui st un d scription d'un grammair LR(1), l s grammair s LALR(1) rajout nt d s r strictions plus compl x s à xpliqu r, mais il st rar qu'un grammair LR(1) n soit pas LALR(1). Pour plus d'information à c propos, il faut s référ r à la sous-s ction 2.5.7.

Dans un règl d grammair form ll pour un langag , chaqu unité ou group syntaxiqu st désigné par un symbol . L s symbol s non-t rminaux désign nt c ux qui sont construits par d s unités ou group s plus p tits, l s symbol s t rminaux (ou tok n) n p uv nt pas êtr subdivisés. Un morc au/mot d l' ntré qui corr spond à un symbol t rminal st un tok n, tandis qu s'il corr spond à un non-t rminal, il s dénomm un group /group m nt.

En utilisant l langag C comm x mpl , voici c qu symbol s t rminaux t non-t rminaux v ul nt dir . L s tok n d C sont l s id ntifiants, l s constant s (numériqu t chaîn d caractèr s), l s différ nts mots clés, l s opérat urs arithmétiqu s t l s ponctuations. La grammair d C inclu l s symbol s t rminaux suivants : "id ntifi r", "numb r", "string", un symbol par mot clé, opérat ur ou ponctuation ("if", "r turn", "const", "static", "int", "char", "plus-sign", "op n-brac ", "comma"...).

La figur 2.1 prés nt un simpl fonction C subdivisé n tok ns :

```
int             /* mot clé "int" */
squar (int x)   /* id ntifi r , op n-par n , id ntifi r , id ntifi r ,
   clos -par n */
{               /* op n-brac */
  r turn x * x; /* mot clé "r turn", id ntifi r , ast risk , id ntifi r
     , s micolon */
}               /* clos -brac */
```

FIG. 2.1 – Simpl fonction C traduit n tok ns

L s group s syntaxiqu s d C inclu nt xpr ssion, stat m nt, d claration t définition d fonction. Ils p uv nt êtr r prés ntés dans la grammair d C par l s symbol s non-t rminaux " xpr ssion", "stat m nt", "d claration" t "function d finition". La grammair complèt utilis nt d nombr ux construct urs supplém ntair , chacun ayant son symbol non-t rminal d façon à xprim r c s quatr s group s syntaxiqu s. L' x mpl précéd nt (figur 2.1) st un "function d finition", composé d'un "d claration" t d'un "stat m nt". Dans c "stat m nt", chaqu $x$ st un " xpr ssion" ainsi qu $x * x$.

Chaqu symbol non-t rminal doit avoir un règl grammatical montrant comm nt il st construit à partir d plusi urs règl s plus simpl s. Par x mpl , un stat m nt n C st l r turn ; il s rait décrit av c un règl d grammair qui pourrait s'écrir "un 'stat m nt' p ut êtr construit d'un 'r turn', un ' xpr ssion' t un point virgul ('s micolon')". Et ainsi pour tous l s stat m nts possibl s n C.

Il faut distingu r un symbol non-t rminal : c lui qui énonc l langag compl t. Il s'app ll l symbol d départ (start symbol). Dans un compilat ur, c la signifi l' ntré complèt du programm . En C, l symbol non-t rminal "séqu nc d définitions t déclarations" jou c rôl .

Par x mpl , `1 + 2` st un xpr ssion C valid (un parti valid d'un programm C), mais n' st pas valid n tant qu programm C nti r. Dans la grammair hors-cont xt d C, `1 + 2` qui st un xpr ssion n' st pas l symbol d départ.

L pars r Ocamlyacc , lit n séqu nc l s tok ns comm l s ntré s t r group l s tok ns n utilisant l s règl s d grammair s. Si l' ntré st valid , la totalité d s tok ns st réduit à un s ul group dont l symbol st l symbol d départ d la grammair . Si l' ntré st invalid , l pars ur r nvoi un rr ur. En C, l' ntré doit êtr un "séqu nc d définitions t déclarations".

### 2.2.2 Des règles formelles à l'entrée Ocamlyacc

Un grammair form ll st un construction mathématiqu . Pour définir l langag pour Ocamlyacc , il faut cré r un fichi r xprimant la grammair dans la syntax Ocamlyacc : un fichi r d grammair Ocamlyacc (voir la s ction sur l format d s fichi rs Ocamlyacc 2.3)

Un symbol non-t rminal d'un grammair form ll st r prés nté dans l'input Ocamlyacc par un id ntifi r, comm n Ocaml. C' st un symbol Caml, xc pté qu'il n p ut pas finir par

’ t qu’il doit comm nc r par un minuscul ( x : xpr, stmt, d claration ).

La r prés ntation Ocamlyacc d’un symbol t rminal st aussi app lé typ d tok n. L s typ s d tok n doiv nt êtr déclaré dans la s ction déclaration d’Ocamlyacc t ils doiv nt êtr ajoutés comm construct urs pour d s typ s concr ts d tok n. En tant qu construct urs, ils doiv nt comm nc r par un majuscul ( x : Int g r Id ntifi r , IF ou RETURN ). L symbol t rminal d’ rr ur st rés rvé pour la récupération d’ rr urs (voir la s ction sur l s symbol s 2.3.2).

L s règl s d grammair ont égal m nt un xpr ssion n syntax Ocamlyacc . Dans l’ x mpl suivant, un règl Ocamlyacc possibl pour l r turn d C :

```
stmt: RETURN xpr SEMICOLON ;
```

Voir la s ction sur la syntax d règl s d grammair 2.3.3.

### 2.2.3 Valeurs sémantiques

La grammair form ll sél ction l s tok ns s ul m nt d’après l ur classification. Par x mpl , sur un règl m ntionn l symbol t rminal ’int g r constant’, c la signifi qu n’import qu l nti r constant st valid pour c tt position. La val ur précis d la constant st ignoré lors du parsing d l’ ntré : si $x + 4$ st grammatical m nt corr ct, alors $x + 1$ ou $x + 3989$ st égal m nt grammatical m nt corr ct.

Par contr , la val ur précis st très important pour c qu l’ ntré signifi un fois parsé . Un compilat ur st inutil s’il échou à distingu r ntr l s constant s 4, 1 t 3989 dans l programm . C’ st pourquoi chaqu tok n d la grammair Ocamlyacc possèd un typ d tok n t un val ur sémantiqu . Voir la s ction sur la sémantiqu 2.3.5.

Un typ d tok n st un symbol t rminal définit dans la grammair , t l qu INTEGER , IDENTIFIER ou SEMICOLON . Il conti nt l s informations néc ssair pour décid r où la validité du tok n p ut apparaîtr t comm nt l r group r av c l s autr s tok ns. L s règl s d grammair n considèr qu l typ d s tok ns, pas l s tok ns ux-mêm s.

La val ur sémantiqu conti nt l r st d s l’information à propos d s tok ns, par x mpl la val ur d’un nti r, l nom d’un id ntifi r. Att ntion, un tok n comm SEMICOLON n’a pas d val ur sémantiqu .

Par x mpl , un tok n d’ ntré p ut êtr classé comm un tok n d typ INTEGER t avoir un val ur sémantiqu 4. Un autr tok n d’ ntré p ut av c l mêm typ d tok n INTEGER mais la val ur 3989. Lorsqu un règl d grammair indiqu qu’un INTEGER st autorisé, c s d ux tok ns sont acc ptabl parc qu’ils sont tous d ux d s INTEGER . Lorsqu l pars r acc pt l tok n, il gard un trac d la val ur sémantiqu .

Chaqu group p ut aussi av c un val ur sémantiqu comm son symbol non-t rminal. Par x mpl , dans un calculatric , un xpr ssion a typiqu m nt un val ur sémantiqu qui st un nombr . Dans un compilat ur, pour un langag d programmation, un xpr ssion a typiqu m nt un val ur sémantiqu qui st un structur d’arbr décrivant l s ns d l’ xpr ssion.

### 2.2.4 Actions sémantiques

Pour qu’un programm soit util , il doit êtr plus qu corr ct à pars r. Il doit aussi produir d s sorti s basé s sur l s ntré s... Dans un grammair Ocamlyacc , un règl d grammair p ut avoir un action associé écrit n Ocaml. Chaqu fois qu l pars ur r connaît un corr - spondanc pour un règl , un action st xécuté . Voir la sous-s ction sur l s actions 2.3.5.

La plupart du t mps, l’obj ctif d l’action st d calcul r la val ur sémantiqu global d la construction global à partir d s s parti s. Par x mpl , si on a un règl qui dit qu’un xpr ssion p ut êtr la somm d d ux xpr ssion, lorsqu l pars ur r connaît un t ll somm ,

chaqu  sous- xpr ssion possèd  un  val ur sémantiqu  qui décrit comm nt  ll  a été construit . L'action d'un  t ll  règl  d vrait cré r un  val ur du mêm  g nr  pour l' xpr ssion r connu .

Par  x mpl , voici figur  2.2 un  règl  qui dit qu'un  xpr ssion p ut êtr  la somm  d  d ux xpr ssions :

```
xpr:   xpr PLUS  xpr    { $1 + $3 }
       ;
```

FIG. 2.2 – Règl  d  somm

L'action indiqu  comm nt produir  la val ur sémantiqu  d  la somm  d s  xpr ssions à partir d s val urs sémantiqu s d s d ux sous- xpr ssions.

### 2.2.5 Positions

D  nombr us s applications, t ls int rprét urs ou compilat urs, doiv nt produir  d s m s-sag s d' rr urs util s  t détaillés. Afin d  fair  c la, on doit êtr  capabl  d'id ntifi r la position dans un t xt  d  chaqu  construct ur syntaxiqu . Ocamlyacc p rm t d  fair  c la.

Chaqu  tok n a un  val ur sémantiqu  t un  position associé . Mais l  typ  d  position st l  mêm  pour tous l s tok ns  t group s. D  plus, la sorti  du pars r  st fait  d  structur  d  donné s p rm ttant d  stock r l s positions. Voir la sous-s ction position 2.3.6.

Many applications, lik  int rpr t rs or compil rs, hav  to produc  v rbos  and us ful  rror m ssag s. To achi v  this, on  must b  abl  to k  p track of th  t xtual position, or location, of  ach syntactic construct. Ocamlyacc provid s a m chanism for handling th s  locations. On p ut utilis r c s positions pour att indr  l s actions utilisant l s fonctions du modul  Parsing.

### 2.2.6 Sortie d'Ocamlyacc : le fichier du parser

Lorsqu'on  xécut  Ocamlyacc , on donn  un fichi r d  grammair  Ocamlyacc comm  ntré . La sorti  st un fichi r Ocaml qui pars  l  langag  décrit par la grammair . C  fichi r  st app lé un pars ur Ocamlyacc . Att ntion Ocamlyacc  st l'outil dont la sorti  st l  pars ur Ocamlyacc

La fonction du pars ur Ocamlyacc  st d  r group r l s tok ns dans l s group s s lon l s règl s d  grammair . Par  x mpl , d  construir  id ntifi r  t opérat urs  n  xpr ssion. Lorsqu'il fait c  r group  m nt, il  ff ctu  l s actions d s règl s d  grammair s qu'il utilis .

L s tok ns vi nn nt d'un  fonction app lé  l'analys ur l xical qu  l'on doit fournir d'un  façon ou d'un  autr . L  pars ur Ocamlyacc app ll  l'analys ur l xical chaqu  fois qu'il a b soin d'un nouv au tok n. Il n sait pas c  qu  conti nt l s tok ns, (bi n qu  l ur val ur sémantiqu  p ut êtr  récupéré ). Typiqu m nt, l'analys ur l xical fabriqu nt l s tok ns  n parsant l s caractèr s d'un t xt , tandis qu'Ocamlyacc n  dép nd pas d  ça. Voir la s ction 2.4.2 sur la fonction d'analys l xical .

L  fichi r pars ur d  Ocamlyacc  st  n Ocaml. Il définit l s fonctions qui implém nt nt la grammair . Chaqu  fonction d' ntré  du cod  Ocaml généré  st nommé d'après l s symbol s d départ du fichi r d  grammair . L s fonctions n  constitu nt pas un programm  Ocaml compl t : il faut fournir qu lqu s fonctions supplém ntair s. La pr mièr  st l'analys ur l xical qui doit êtr  donné comm  argum nt d  la fonction d' ntré  du pars ur. Un  autr  fonction  st la fonction d  rapport d' rr ur app lé  par l  pars ur afin d  rapport r un  rr ur. Et  nfin, un programm  compl t  Ocaml doit pouvoir app l r un (ou plusi urs) fonction d' ntré  généré  ou alors l pars ur n  pourra pas d' xécut r. Voir la s ction 2.4 sur l'int rfac  du pars ur.

### 2.2.7 Les étapes pour utiliser Ocamlyacc

Voici l s étap s d construction d'un langag utilisant Ocamlyacc , partant d la spécification d la grammair allant jusqu'au compilat ur ou int rprét ur :

- Form ll m nt spécifi r la grammair sous un form r connu par Ocamlyacc (voir la s ction sur l s fichi rs d grammair 2.3). Pour chaqu règl dans l langag , décrir l'action qui doit êtr ff ctué lorsqu'un instanc d c tt règl s ra r connu . L'action st décrit par un séqu nc d stat m nt Ocaml.
- Ecrir un analys ur l xical pour analys r l' ntré t transm ttr l s tok ns au pars ur. L'analys ur l xical p ut êtr écrit à la main n Ocaml (voir la s ction 2.4.2). Il p ut aussi êtr produit par Ocamll x , voir la parti 1.
- Ecrir un fonction d contrôl qui app ll l pars ur produit par Ocamlyacc
- Ecrir l s fonctions d rapport d' rr urs.

Pour transform r c cod n xécutabl , il faut suivr l s étap s suivant s :

- Ex cut r Ocamlyacc sur la grammair pour produir l pars ur.
- Compil r l cod d sorti d'Ocamlyacc ainsi qu l s autr s fichi rs sourc s.
- Li r l s fichi rs obj ts pour produir l produit fini.

## 2.3 Le fichier de grammaire Ocamlyacc

Ocamlyacc pr nd n ntré un spécification d grammair hors-cont xt t produit un fonction Ocaml qui r connaît l s instanc s corr ct s d c tt grammair . L fichi r d' ntré d grammair Ocamlyacc a par conv ntion un xt nsion n ".mly". Voir la s ction Invoqu r Ocamlyacc 2.8.

### 2.3.1 Structure du fichier

L fichi r d grammair Ocamlyacc st constitué d quatr s ctions principal s, prés nté s ici (figur 2.3) av c l urs délimit urs :

```
%{
        H ad r – Ocaml d clarations (Ocaml cod )
%}
        Ocamlyacc d clarations
%%
        Grammar rul s
%%
        Trail r – Additional Ocaml cod (Ocaml cod )
```

FIG. 2.3 – Structur du fichi r d grammair Ocamlyacc

- L s %%, %{ t %} sont d s ponctuations qui p rm tt nt d sépar r l s s ctions.
- Dans la s ction *header* on définit l s typ s, variabl s t fonctions utilisé s dans l s actions.
- Dans la s ction *Ocamlyacc declarations*, on déclar l s noms d s symbol s t rminaux t non-t rminaux. On p ut aussi décrir l s précéd nc s d s opérat urs t l s typ s d s donné s d s val urs sémantiqu s d s différ nts symbol s.
- La s ction *grammar rules* définit comm nt construir chaqu non-t rminal à partir d s s morc aux
- La s ction *Trailer* p ut cont nir du cod Ocaml

Par défaut, l s comm ntair s sont ncadrés ntr "/*" t "*/" (comm n C) xc pté dans

La valeur renvoyée par la fonction de lexer est toujours l'un des symboles terminaux. Chaque type de token ne devient une valeur de type variant dans le fichier parser, afin que la fonction de lexer puisse ner tourner une.

Parce que la fonction de lexer est définie dans un fichier séparé, il faut faire ensorte que les définitions de types et de tokens soient disponibles à ce niveau. Après avoir invoqué `ocamlyacc file name.mly`, le fichier "file name.mli" qui est généré contient les définitions de type et de token. Il est utilisé dans la fonction de lexer.

Le symbole d'erreur est un symbole terminal réservé pour récupérer les erreurs (voir la section 2.6 sur le rattrapage d'erreur), il doit rester réservé à cet usage.

### 2.3.3 Syntaxe des règles de grammaire

Une règle de grammaire Ocamlyacc a la forme général suivant (figure 2.4) :

```
result:
          symbol ... symbol { semantic-action }
        | ...
        | symbol ... symbol { semantic-action }
        ;
```

FIG. 2.4 – Structure de règle de grammaire Ocamlyacc

où `result` est le symbole non-terminal que cette règle décrit, `symbol` sont des symboles terminaux ou non-terminaux qui sont assemblés par cette règle (voir la section 2.3.2 sur les symboles).

Par exemple la règle suivant (figure 2.5) qui deux groupes de type `exp` avec un token `PLUS` au milieu peuvent être combiné en un groupe plus large de type `exp`.

```
exp:      exp PLUS exp          {}
        ;
```

FIG. 2.5 – Exemple de la combinaison par PLUS

Les espaces dans les règles sont nécessaires seulement pour séparer les symboles, des espaces supplémentaires peuvent être ajoutés.

A la suite des composants de la règle, il doit y avoir l'action qui détermine la sémantique de la règle. Un action doit être défini de cette façon `{ Ocaml code }`, voir la section 2.3.5 sur les actions.

Il est possible d'écrire des règles séparés pour le même résultat, mais on peut aussi les rejoindre à l'aide d'un barre verticale "|" comme dans la figure 2.6. Les règles sont considérés

```
result:
          rule1-symbol ... rule1-symbol { rule1-semantic-action }
        | rule2-symbol ... rule2-symbol { rule2-semantic-action }
        | ...
        ;
```

FIG. 2.6 – Plusieurs règles

comm distinct s, mêm quand ll s sont écrit s d c tt manièr .

Si la composition d'un règl st vid , c la v ut dir qu l résultat p ut corr spondr à un chaîn d caractèr vid . Par x mpl , voici comm nt définir un séqu nc d zéro séparé d virgul s ou d plus d group d' xp (figur 2.7) :

```
xps q:    /* mpty */    {}
        |  xps q1                {}
        ;

xps q1:   xp                     {}
        |  xps q1 COMMA  xp               {}
        ;
```

FIG. 2.7 – Règl vid

L'usag v ut qu'on écriv l comm ntair `/* mpty */` pour chaqu règl sans composant.

### 2.3.4 Règles récursives

Un règl st app lé récursiv si son résultat non-t rminal apparaît aussi dans l m mbr d droit d la règl . Pratiqu m nt tout s l s grammair s Ocamlyacc utilis nt la récursion, parc qu c' st l s ul moy n d définir un séqu nc d'un nombr indét rminé d'un chaîn d caractèr . Voici l' x mpl (figur 2.8) d la définition récursiv d'un séqu nc séparé d virgul d'un ou plusi urs xpr ssions :

```
xps q1:   xp                     {}
        |  xps q1 COMMA  xp               {}
        ;
```

FIG. 2.8 – Règl récursiv - virgul s

C tt récursion st app lé récursion gauch car l'utilisation du symbol `xps q1` st l symbol l plus à gauch dans l m mbr d droit d la règl . Voici la mêm construction av c un récursion droit (figur 2.9) :

```
xps q1:   xp                     {}
        |  xp COMMA  xps q1               {}
        ;
```

FIG. 2.9 – Règl récursiv droit

Tout sort d séqu nc p ut êtr défini n utilisant la récursion gauch ou droit , mais il st préférabl d toujours utilis r la récursion gauch car c la utilis un nombr limité d' spac dans la pil . La récursion droit utilis un spac dans la pil Ocamlyacc proportionn l au nombr d'élém nt d la séqu nc , parc qu tous l s élém nts doiv nt êtr mpilés avant qu la règl puiss êtr appliqué au moins un fois. Voir la s ction 2.5 sur l'algorithm du pars r.

On p ut aussi trouv r d la récursion mutu ll ou indir ct lorsqu l résultat d la règl n'apparaît pas dir ct m nt dans l m mbr droit, mais qu'il apparaît dans d s règl s d'autr s non-t rminaux qui apparaiss nt ux dans l m mbr droit d c tt règl .

Par exemple dans la figure 2.10 suivante, on définit deux non-terminaux mutuellement récursifs, chacun faisant référence à l'autre.

```
xpr:      primary                          {}
        | primary PLUS primary   {}
        ;

primary:  constant                         {}
        | LPAREN xpr RPAREN      {}
        ;
```

FIG. 2.10 – Règles mutuellement récursives

### 2.3.5 Définir la sémantique d'un langage

Les règles de grammaire d'un langage ne définissent que la syntaxe. La sémantique est définie par les valeurs sémantiques associées aux différents tokens et groupes, et par les actions à effectuer lorsque les différents groupes sont reconnus.

Par exemple, un calculateur calculette correct mémorise la valeur associée à chaque expression est le nombre correct. Il ajoute correctement mémorise qu'l'action du groupe $x + y$ est d'ajouter les nombres associés à $x$ et $y$.

#### Type de données des valeurs sémantiques

Dans un simple programme, il peut être suffisant d'utiliser le même type de donnée pour les valeurs sémantiques de tous les constructeurs du langage. Dans la plupart des programmes, il est cependant nécessaire d'avoir des types de données différents pour des types de tokens ou de groupes différents. Par exemple, un constant numérique peut nécessiter le type int ou float alors qu'un constant chaîne de caractère ou un identifiant peut nécessiter le type string.

Afin d'utiliser plus d'un type de donnée pour les valeurs sémantiques dans un parseur, Ocamlyacc impose de choisir un type de donnée pour chaque symbole (terminal ou non-terminal) pour lequel la valeur sémantique est utilisée. Pour les tokens, cela se fait à l'aide de %**token** Ocamlyacc déclaration (voir la sous-sous-section 2.3.7 sur les noms de type de token) et pour les groupes avec [redacted] (voir la sous-sous-section 2.3.7 sur les symboles non-terminaux).

#### Actions

Une action est associée à une règle syntaxique est contenu du code Ocaml qui est exécuté chaque fois qu'une instance de la règle est reconnu. La tâche de la plupart des actions est de calculer une valeur sémantique pour le groupe construit par la règle à partir des valeurs sémantiques des tokens et les plus petits groupes qui constituent la règle.

Une action est construit de statements Ocaml encadré par des accolades. Les règles n'ont qu'une action à la fin de la règle suivi par tous les composants.

```
xp:     ...
      | xp PLUS xp { $1 +. $3 }
```

FIG. 2.11 – Action

C tt règl construit un `xp` à partir d d ux group s `xp` plus p tits conn ctés av c l tok n du sign "+". Dans l'action `$1` t `$3` font référ nc aux val urs sémantiqu s d s d ux composants d s group s `xp`, qui sont l pr mi r t l troisièm symbol s du m mbr droit d la règl . La somm st r nvoyé afin qu la val ur sémantiqu d s l' xpr ssion d'addition soit r connu par la règl . S'il y avait un val ur sémantiqu utilisabl av c l tok n PLUS, on pourrait y référ r av c `$2`.

### 2.3.6 Suivre les positions

Bi n qu l s règl s d grammair t l s actions sémantiqu s soi nt suffisant s pour écrir un pars ur ntièr m nt fonctionn l, il p ut êtr util d pouvoir rajout r d s informations complém ntair s, n particuli r l s symbol s d positions.

La façon dont l s positions sont traité s st défini n fournissant l typ d donné s t l s actions à ff ctu r quand l s règl s sont appliqué s.

#### Type de données des positions

L cont nu d c tt s ction st valabl d puis Ocaml 3.08.

L typ d donné s pour l s positions c construit comm c ci (figur 2.12) :

```
type position = {
    pos_fnam  : string;          (* file name *)
    pos_lnum  : int;             (* line number *)
    pos_bol   : int;             (* the offset of the beginning of the
                                    line *)
    pos_cnum  : int;             (* the offset of the position *)
}
```

FIG. 2.12 – Typ d positions

La val ur du champ `pos_bol` st l nombr d caractèr s ntr l début du fichi r t l début d la lign , la val ur du champ `pos_cnum` st l nombr d caractèr s ntr l début du fichi r t la position.

L mot ur l xical parvi nt s ul m nt au champ d `pos_cnum` du `l xbuf.l x_curr_p` av c l nombr d caractèr s lus à partir du début du l xbuf. L programm ur st donc r sponsabl d l' xactitud d s autr s champs. Avant d'utilis r la position dans l pars ur, il faut initialis r `L xing.l xbuf.l x_curr_p` corr ct m nt dans l l x r, n utilisant un fonction comm c ll ci (figur 2.13) :

#### Actions et positions

L s actions sont util s non s ul m nt pour définir la sémantiqu du langag mais aussi (à l'aid d s positions) pour décrir l comport m nt du pars ur. L moy n l plus simpl d construir d s positions d group s syntaxiqu st très similair à la façon dont l s val urs

```
let incr_lin num l xbuf =
  let pos = l xbuf.L xing.l x_curr_p in
  l xbuf.L xing.l x_curr_p <- { pos with
    L xing.pos_lnum = pos.L xing.pos_lnum + 1;
    L xing.pos_bol = pos.L xing.pos_cnum;
  }
;;
```

FIG. 2.13 – Initialis r l s positions du l x r/pars r

sémantiqu s sont calculé s. Pour un règl donné , plusi urs construct urs p uv nt êtr utilisés pour accéd r aux positions d s élém nts corr spondants. La position du nièm composant du m mbr droit p ut êtr obt nu à l'aid d (figur 2.14) :

```
val Parsing.rhs_start : int -> int
val Parsing.rhs_ nd : int -> int
```

FIG. 2.14 – Position

Parsing.rhs_start n r nvoi l'écart av c l pr mi r caractèr du nièm it m du m mbr droit d la règl . Parsing.rhs_ nd n r nvoi l'écart av c l d rni r caractèr d l'it m. C s fonctions doit êtr app lé s par l s actions. n vaut 1 pour l'it m l plus à gauch t l s pr mi r caractèr dans un fichi r a un écart d 0.

On p ut aussi utilis r l s fonctions suivant s (figur 2.15) :

```
val Parsing.rhs_start_pos : int -> L xing.position
val Parsing.rhs_ nd_pos : int -> L xing.position
```

FIG. 2.15 – Fonctions d position

(D puis Ocaml 3.08) L s fonctions suivant s r tourn nt un position plutôt qu'un écart (voir sous-sous-s ction 2.3.6).

La position du group du m mbr d gauch p ut êtr obt nu par (figur 2.16) :

```
val Parsing.symbol_start : unit -> int
val Parsing.symbol_ nd : unit -> int
```

FIG. 2.16 – Fonctions pour m mbr d gauch

symbol_start () r nvoi l'écart av c l pr mi r caractèr du m mbr gauch t symbol_ nd () l'écart av c l d rni r caractèr .

(D puis Ocaml 3.08) L s fonctions suivant s sont comm symbol_start and symbol_ nd , xc pté qu' ll s r nvoi nt un position plutôt qu'un écart (voir sous-sous-s ction 2.3.6).

```
val Parsing.symbol_start_pos : unit -> L xing.position
val Parsing.symbol_ nd_pos : unit -> L xing.position
```

Voici un exemple basique qui utilise le type donné par défaut des positions (figure 2.17) :

```
expr:    ...
       | expr DIVIDE expr
            { if $3 <> 0.0 then $1 /. $3
              else (
                let start_pos = Parsing.rhs_start_pos 3 in
                let end_pos = Parsing.rhs_end_pos 3 in
                printf "%d.%d-%d.%d: division by zero"
                  start_pos.pos_lnum (start_pos.pos_cnum -
                     start_pos.pos_bol)
                   end_pos.pos_lnum (end_pos.pos_cnum - end_pos.
                     pos_bol);
                1.0
              )
            }$
```

FIG. 2.17 – Exemple d'utilisation des positions

### 2.3.7 Déclarations Ocamlyacc

La section "déclaration Ocamlyacc" de la grammaire Ocamlyacc définit les symboles utilisés pour formuler la grammaire et les types donnés des valeurs sémantiques. Voir la sous-section 2.3.2 sur les symboles.

Tous les types de tokens doivent être déclarés. Les symboles non-terminaux doivent être déclarés s'il est nécessaire de spécifier qu'le type donné sera utilisé pour la valeur sémantique (voir la sous-sous-section 2.3.5 sur les types donnés des valeurs sémantiques).

La première règle dans le fichier spécifie aussi le symbole de départ, par défaut. Si on veut un autre symbole comme départ, on doit le déclarer explicitement (voir la sous-section 2.2.1 sur les grammaires hors-contexte).

#### Noms des types de token

Le moyen classique pour déclarer un nom de type de token (symbole terminal) est le suivant (figure 2.18) :

```
%token name ... name
%token <type> name ... name
```

FIG. 2.18 – Déclaration de type de token

Ocamlyacc convertit la sous forme d'un type dans le parseur, de façon à ce que la fonction lexer puisse utiliser le nom correspondant au code du type de ce token.

Si le token a une valeur, l'argument de la déclaration **%token** doit inclure le type donné encadré par les "<>" (voir la sous-sous-section 2.3.5 sur les types donnés des valeurs sémantiques).

Par exemple : **%token** <float> NUM /* définit **token** NUM **and its type** */

La parti "typ " doit êtr un xpr ssion d typ Caml. La parti <type> st copié dans l fichi r sorti n ".mli", tous l s noms d construct ur d typ doiv nt êtr valid s (par x mpl . Modul _nam .typ _nam ).

### Opérateurs de précédences

On utilis l s déclarations %left, %right ou %nonassoc pour spécifi r l s précéd nc s t associativités. Voir la s ction 2.5.3 sur l s conflits résolus par l s opérat urs d précéd nc .

La syntax d déclaration d précéd nc st la suivant (figur 2.19) :

```
%right symbols ... symbols
%nonassoc symbols ... symbols
```

FIG. 2.19 – Syntax d s déclarations d précéd nc

L s déclarations d précéd nc précis nt l'associativité t la précéd nc r lativ pour tous l s symbol s :

- L'associativité d'un opérat ur "op" dét rmin comm nt int rprét r la répétition d l'opérat ur : si "x op y op z" st considéré comm "(x op y) op z" ou "x op (y op z)". %left spécifi l'associativité gauch "(x op y) op z)" t %right spécifi l'associativité droit "x op (y op z)". %noassoc n spécifi pas d'association : "x op y op z" s ra considéré comm un rr ur d syntax .
- La précéd nc d'un opérat ur dét rmin son comport m nt lorsqu'il st ncapsulé av c d'autr s opérat urs. Tous l s tok ns déclarés dans la mêm déclaration d précéd nc ont un précéd nc équival nt , c qui signifi qu'il s ront ncapsulé s lon l ur associativité. Lorsqu d ux tok ns déclarés dans d ux déclarations d précéd nc différ nt sont associés, c lui déclaré l plus tard possèd la précéd nc la plus fort , il st donc groupé (au s ns syntaxiqu ) n pr mi r.

### Symboles non-terminaux

On p ut déclar r l typ d chaqu symbol non-t rminal pour l squ ls l s val urs sont utilisé s. C la s fait à l'aid d la déclaration %type comm ça :

%type <type> nont rminal ... nont rminal. "nont rminal" st l nom d'un symbol non-t rminal t "typ " st l nom du typ voulu. On p ut donn r autant d symbol s non-t rminaux d départ qu l'on souhait dans la mêm déclaration %type s'ils ont l mêm typ . On utilis l s spac s pour sépar r l s noms d symbol s.

C tt déclaration st néc ssair pour l s symbol s d départ. Voir la sous-sous-s ction 2.3.7 sur l s noms d typ s d tok n.

### Le symbole de départ

On doit déclar r un symbol d départ/démarrag n utilisant la déclaration %start d la façon suivant :

%start symbol ... symbol

Chaqu symbol d départ a un fonction d parsing du mêm nom dans l fichi r d sorti . On s' n s rt comm point d' ntré pour la grammair . Rapp l : chaqu symbol d départ doit avoir un typ associé, on utilis alors la déclaration %type, voir sous-sous-s ction précéd nt 2.3.7 sur l s symbol s non-t rminaux.

**Récapitulatif des déclarations Ocamlyacc**

Voici l  récapitulatif d  s déclarations néc ssair s pour définir un  grammair  :
H r  is a summary of th  d clarations us d to d fin  a grammar :

- **%token** Déclaration d'un symbol  t rminal (nom d  typ  d  tok n) sans précéd nc  ni associativité (voir 2.3.7).
- **%right** Déclaration d'un symbol  t rminal (nom d  typ  d  tok n) qui a un  associativité droit  (voir 2.3.7).
- **%left** Déclaration d'un symbol  t rminal (nom d  typ  d  tok n) qui a un  associativité gauch  (voir 2.3.7).
- **%nonassoc** Déclaration d'un symbol  t rminal (nom d  typ  d  tok n) qui n' st pas associatif ( rr ur d  syntax  si c'était l  cas, voir 2.3.7).
- **%type** Déclaration du typ  d  la val ur sémantiqu  d'un symbol  non-t rminal (voir 2.3.7).
- **%start** Précis  l  symbol  d  départ d  la grammair  (voir 2.3.7).

## 2.4 L'interface du parser

L  pars r Ocamlyacc  st  n fait un  ns mbl  d  fonction Ocaml nommé  s à partir d  s symbol s d  départ d  la grammair  (voir sous-sous-s ction 2.3.7 sur l s symbol s d  départ). C tt s ction décrit l s conv ntions d  s fonctions du pars r  t l s autr s fonctions néc ssair s.

### 2.4.1 La fonction Parser

Afin d  fair  un parsing, il faut app l r la fonction d  pars r av c d ux paramètr s. L pr mi r paramètr  st la fonction d'analys ur l xical d  typ  `L xing.l xbuf -> token`  t l s cond  st un  val ur d  typ  `L xing.l xbuf`.

Soit `p rse` l  symbol  d  départ dans l  fichi r `p rser.mly`  t `token` la fonction l x r dans l  fichi r `lexer.mll`, on procèd  d  la manièr  suivant  (figur  2.20) :

```
let l xbuf = L xing.from_chann l stdin in
...
let r sult = Pars r.pars  L x r.token l xbuf in
...
```

FIG. 2.20 – App l d s fonctions

La fonction pars r lit l s tok ns,  xécut  l s actions  t s  t rmin  soit quand  ll  r ncontr un fin d'  ntré , soit un  rr ur d  syntax  non récupéré .

### 2.4.2 La fonction d'analyse lexicale

La fonction d'analys  l xical  st nommé  d'après l s déclarations d  règl s. Ell  r connaît l s tok ns d  puis l  flux d'  ntré  t l s r nvoi  au pars r. Ocamlyacc n  cré  pas c tt  fonction automatiqu  m nt : il faut l'écrir  afin qu  la fonction d  pars r puiss  l'app l r. On p ut assimil r c tt  fonction à un scann r l xical. Ell  st général m nt généré  par Ocamll x (voir parti  1 ou l chapitr  12 du manu l Ocaml `http ://c ml.inri .fr/pub/docs/m nu l-oc ml/m nu l026.html`).

### 2.4.3 La fonction de rapport d'erreur

L pars r Ocamlyacc dét ct l s rr urs d parsing ou d syntax lorsqu'il lit un tok n qui n satisfait aucun d s règl s d syntax . Un action d la grammair p ut égal m nt provoqu r un rr ur, utilisant la l vé Parsing.Pars _ rror .

L pars r Ocamlyacc va rapport r l' rr ur n app lant un fonction d rapport d' rr ur app lé `p rse_error` qui st optionn ll . Par défaut, la fonction `p rse_error` n fait aucun action. Ell st app lé par la fonction d pars r lorsqu'un rr ur d syntax st trouvé t r çoit un argum nt. Pour un rr ur d parsing, l' rr ur st général m nt `synt xe error`.

La définition suivant st général m nt suffisant pour l s programm s simpl s :

```
let pars _ rror s = print_ ndlin s
```

Si la fonction `p rse_error` r nvoi qu lqu chos à la fonction d parsing, ll ssay ra d rattrap r l' rr ur s'il xist d s règl s (valid s) d grammair qui récupèr nt l s rr urs (voir la s ction 2.6 sur l rattrapag d' rr urs). Si la récupération st impossibl , la fonction d parsing lèv ra l' xc ption Parsing.Pars _ rror .

## 2.5 L'algorithme du parseur Ocamlyacc

Au fur t à m sur qu Ocamlyacc lit l s tok ns, ils sont mpilés av c l ur val ur sémantiqu . La pil st app lé *pile du parser*. On app ll ça du *shifting*.

Par x mpl , supposons qu la calculatric infix a lu 1 + 5∗ t qu 3 soit à lir . La pil aura 4 élém nts, un par tok n shifté.

La pil n'a pas toujours un élém nt pour chaqu tok n. Lorsqu l s d rni rs n tok ns t group s shiftés corr spond nt aux composants d'un règl d grammair , ils p uv nt êtr combinés suivant c tt règl . C' st c qu'on app ll la *réduction*. C s tok ns t group s sont r mplacés dans la pil par un simpl group dont l symbol st l résultat (m mbr d gauch ) d la règl . Exécut r l'action d la règl fait parti du proc ssus d réduction, c la calcul la val ur sémantiqu du group résultant.

Par x mpl , si la pil d la calculatric infix conti nt 1 + 5 ∗ 3 t qu l prochain tok n st un saut d lign , alors l s trois (d rni rs) élém nts p uv nt êtr réduits à 15 n appliquant la règl xpr: xpr MULTIPLY xpr;. La pil conti ndra alors l s trois élém nts suivants : 1 + 15. Un réduction supplém ntair p ut êtr ff ctué , qui produit la val ur 16. Et dès lors l saut d lign p ut êtr shifté.

L pars r ssai , n shiftant ou réduisant, d réduir l'input complèt m nt à un group simpl dont l symbol st l symbol d départ d la grammair (voir la sous-s ction 2.2.1 sur l s grammair s hors-cont xt ).

C g nr d pars r st connu dans la littératur comm un pars r asc ndant.

### 2.5.1 Les tokens en avant

L pars ur Ocamlyacc n réduit pas toujours immédiat m nt, aussitôt qu l s n tok ns t group s corr spond nt à un règl . Un stratégi aussi simpl d couvr pas la plupart d s langag s. En fait, lorsqu'un réduction st possibl , l pars ur r gard " n avant" l tok n suivant d façon à décid r quoi fair .

Quand un tok n st lu, il n' st pas immédiat m nt shifté. Il d vi nt d'abord un *token en avant*, qui n' st pas sur la pil . L pars ur p ut alors ff ctué un ou plusi urs réductions d tok ns t group s sur la pil , tandis qu l tok n n avant r st d côté. Quand plus aucun réduction d p ut êtr ff ctué , l tok n n avant st shifté sur la pil . Il p ut donc r st r d s réductions possibl s, t s lon l typ du tok n n avant, c rtain s règl s p uv nt choisir d r tard r l ur application.

Voici figur  2.21 un  x mpl  d  cas simpl  d'utilisation du tok n  n avant. C s d ux règl s définiss nt d s  xpr ssions cont nant l'opérat ur d'addition  t l'opérat ur postfix  d  factori ll (FACTORIAL pour "!")  t l' ncadr m nt par par nthès s.

```
xpr :      t rm PLUS  xpr
         | t rm
         ;

t rm :     LPAREN  xpr RPAREN
         | t rm FACTORIAL
         | NUMBER
         ;
```

FIG. 2.21 – Tok n  n avant

Supposons qu  l s tok ns 1 + 2 sont lus  t shifté, qu  fair  nsuit  ?

- Si l  tok n suivant  st RPAREN alors l s trois pr mi rs tok ns doiv nt êtr  réduit comm  xpr. C  s ra alors la s ul  possibilité, parc  qu  shift r RPAREN produira un  séqu nc  t rm RPAREN  t aucun règl  n  l  p rm t.
- Si l  tok n suivant  st FACTORIAL, il doit êtr  shifté dir ct m nt afin d  réduir 2 FACTORIAL  n t rm. Si l  pars r voulait réduir  avant d  shift r 1 + 2 d vi ndrait un  xpr. On n  pourrait donc plus shift r FACTORIAL parc  qu'on aurait sur la pil un  séqu nc  d  symbol s  xpr FACTORIAL, qui n' st pas autorisé dans l s règl s.

### 2.5.2 Conflits shift/reduce

Voici un  x mpl  figur  2.22 d  règl s pour un langag  ayant d s if-th n  t if-th n- ls  qu l'on souhait  pars r :

```
if_stmt :
          IF  xpr THEN stmt
        | IF  xpr THEN stmt ELSE stmt
        ;
```

FIG. 2.22 – Conflits shift/r duc

IF, THEN,  t ELSE sont d s symbol s t rminaux pour l s mots clés corr spondants. Quand l  tok n ELSE  st lu  t d vi nt un tok n  n avant, l  cont nu d  la pil  (supposant l' ntré corr ct )  st valid  pour un  réduction par la pr mièr  règl . Mais il  st aussi légitim  d  shift r l  ELSE parc  qu  c la pourrait conduir  à un  év ntu ll  réduction par la d uxièm  règl .

C tt  situation, lorsqu  shift r ou réduir  p uv nt êtr  tout d ux valid s,  st app lé  un conflit shift/r duc . Ocamlyacc  st conçu pour résoudr  c  problèm  n choisissant d  shift r, sauf s'il y a d'autr s précisions dans l s déclarations d  précéd nc  sur l s opérat urs.

La raison pour laqu ll  on choisit plutôt d  shift r qu  d  réduir  st xposé ci-après. Parc qu  l  pars r préfèr  shift r l  ELSE l  résultat  st d'associ r la claus  " ls " au stat m nt "if" l  plus imbriqué,  n r ndant l s d ux  ntré s suivant s équival nt s (figur  2.23) :

Il y a un conflit car la grammair  t ll  qu' ll  st écrit  st  ll -mêm  ambiguë : on p ut pars r un stat m nt "if" simpl  imbriqué. L s conv ntions établi s sont qu  l s ambiguïtés sont

```
if x then if y then win (); else los ;

if x then do; if y then win (); else los ; end;
```

FIG. 2.23 – Entré s équival nt s si shift

résolu s n associant l s claus s " ls " au stat m nt "if" l plus imbriqué t c' st c qu fait Ocamlyacc n choisissant d shift r plutôt qu d réduir .

R marqu : il s rait idéal m nt plus propr d'écrir un grammair non ambiguë, mais c n' st pas aisé dans c cas. C tt ambiguïté particulièr a été r ncontré dans l s spécifications d'*Algol 60* t st app lé l'ambiguïté "dandling ls ".

La définition précéd nt (figur 2.22) du `if_stmt` st la s ul à blâm r pour l conflit, mais l conflit n'apparaît n fait qu'av c l'ajout d'autr s règl s. Voici un fichi r figur 2.24 compl t d grammair Ocamlyacc qui provoqu un conflit.

```
%token IF THEN ELSE variabl
%%
stmt:      xpr
         | if_stmt
         ;

if_stmt:
           IF  xpr THEN stmt
         | IF  xpr THEN stmt ELSE stmt
         ;

 xpr:      variabl
         ;
```

FIG. 2.24 – Grammair à conflit

### 2.5.3 Précédence des opérateurs

D'autr s conflits shift/r duc apparaiss nt dans l s xpr ssions arithmétiqu s. Shift r n' st alors pas la m ill ur solution. L s déclarations d précéd nc Ocamlyacc p rm tt nt d spécifi r quand shift r ou quand réduir .

#### Cas de précédence nécessaire

Voici figur 2.25 un grammair ambiguë (car l' ntré `1 − 2 ∗ 3` p ut êtr parsé d d ux façons) :

Si l pars r voit l s tok ns `1 − 2` d vra-t-il réduir av c la règl d soustraction ? C la dép nd du tok n suivant.

- `)` on doit réduir , shift r n' st pas valid car aucun règl n p ut réduir la séqu nc `MINUS xpr` (c' st-à-dir -2) ou n'import qu l group comm nçant par ça.
- `∗ ou ∗∗` on doit choisir : shift r ou réduir p uv nt êtr ff ctu r mais av c d s résultats différ nts.

```
xpr :       xpr MINUS xpr
          | xpr MULTIPLY xpr
          | xpr LT xpr
          | LPAREN xpr RPAREN
          ...
          ;
```

FIG. 2.25 – Grammair à conflit solubl par précéd nc

Pour décid r c qu Ocamlyacc doit fair , il faut r gard r l s résultats. Si l'opérat ur suivant "op" st shifté, alors il doit êtr réduit d'abord pour p rm ttr un autr possibilité d réduction pour l -. L résultat s rait 1 - (2 op 3). Sinon, si la soustraction st réduit avant d shift r l'opérat ur, l résultat st (1 - 2) op 3. Ainsi, l choix ntr shift t r duc doit dép ndr d la r lativ précéd nc ntr l s opérat urs (ici "-" t "op").

Dans l cas où l' ntré st $1 - 2 - 5$, st-c $(1 - 2) - 5$ ou $1 - (2 - 5)$ qui doit êtr r connu ? Pour la plupart d s opérat urs, on préfèr la pr mièr possibilité, qu'on app ll associativité gauch . La d uxièm , association droit , st préféré pour l s opérat urs d'aff ctations. L choix d'association gauch ou droit dép nd si l pars r choisit shift ou r duc lorsqu la pil conti nt $1 - 2$ t qu l tok n n avant st $-$.

### Spécification des précédences d'opérateurs

Ocamlyacc p rm t d spécifi r d s choix à l'aid d s déclarations d précéd nc **%left** t **%right** (voir la sous-sous-s ction 2.3.7 sur la précéd nc d s opérat urs). Chaqu déclaration conti nt un list d tok n, qui sont d s opérat urs dont on déclar la précéd nc t l'associativité. La déclaration **%left** r nd l s opérat urs associatifs gauch t **%right** associatifs droit. La troisièm alt rnativ %noassoc r nd incorr ct l'association, c' st-à-dir provoqu un rr ur d syntax si l mêm opérat ur st trouvé d ux fois "d suit ".

La précéd nc r lativ ntr l s différ nts opérat urs st défini pas l'ordr dans l qu l ils sont déclarés. La pr mièr déclaration **%left** ou **%right** précis l s opérat urs pour l squ ls la précéd nc st la plus faibl t on p ut ordonn r comm ça l s précéd nc s ntr s opérat urs par déclaration d précéd nc s succ ssiv s.

### Exemples de précédence

Pour la règl d grammair d la figur 2.26 suivant :

```
xpr :       xpr MINUS xpr
          | xpr MULTIPLY xpr
          | xpr LT xpr
          | LPAREN xpr RPAREN
          ...
          ;
```

FIG. 2.26 – Ex mpl d grammair

on va définir l s règl s d précéd nc suivant s :

Dans un x mpl plus compl t, qui élargirait à plusi urs autr s opérat urs, on déclar rait l s opérat urs par group d précéd nc . Par x mpl figur 2.28 "+" st déclaré av c "-".

```
%left LT
%left MINUS
%left MULTIPLY
```

FIG. 2.27 – Définition d  plusi urs précéd nc s gauch

```
%left LT GT EQ NE LE GE
%left PLUS MINUS
%left MULTIPLY DIVIDE
```

FIG. 2.28 – Plusi urs précéd nc s groupé s

On not  ici NE pour l'opérat ur "not  qual"  t ainsi d  suit  pour l s autr s opérat urs.

**Fonctionnement de la précédence**

L  pr mi r  ff t d s déclarations d  précéd nc  st d'aff ct r d s niv aux d  précéd nc  aux symbol s t rminaux déclarés. L  d uxièm  ff t d s d'aff ct r d s niv aux d  précéd nc s à c rtain s règl s : chaqu  règl  obti nt sa précéd nc  du d rni r symbol  t rminal m ntionné dans s s composants.

On p ut aussi spécifi r  xplicit m nt la précéd nc  d'un  règl , voir la sous-s ction 2.5.4 sur l s précéd nc s dép ndant s du cont xt .

En définitiv , la résolution d s conflits fonctionn  n comparant l s précéd nc s d s règl s considéré s av c l  tok n à v nir. Si la précéd nc  du tok n  st plus él vé , l  choix  st d  shift r. Si la précéd nc  d  la règl  st plus él vé , l  choix  st d  réduir . S'ils ont un  précéd nc  égal , l  choix s  fait à partir d  l'associativité du niv au d  précéd nc . L  fichi r d  sorti " xplicit " obt nu à l'aid  d  l'option -v (voir la s ction Invoqu r Ocamlyacc 2.8) comm nt  st résolu chaqu  conflit.

Ni tout s l s règl s, ni tous l s tok ns n'ont d  règl s d  précéd nc . Si aucun  d s règl s ou tok n à v nir n'ont d  précéd nc , on shift par défaut.

### 2.5.4 6.4. Context-Dependent Precedence

Souv nt, la précéd nc  d'un opérat ur dép nd du cont xt . Par  x mpl , un sign  moins a typiqu m nt un  très fort  précéd nc  n tant qu'opérat ur unair , mais on dirait un  précéd nc  un p u moindr  (moindr  qu  la multiplication)  n tant qu'opérat ur binair .

L s déclarations d  précéd nc s, **%left**, **%right**  t **%nonassoc** n  p uv nt êtr  utilisé s qu'un  fois pour un tok n donné. Ainsi un tok n n'a qu'un  s ul  déclaration d  précéd nc . Afin d  déclar r d s précéd nc s dép ndant s du cont xt , il faut utilis r un autr  mécanism , l  modificat ur **%prec** pour l s règl s.

L  modificat ur **%prec** déclar  la précéd nc  d'un  règl  n particuli r  n spécifiant l  symbol  t rminal pour l qu l la précéd nc  doit êtr  utilisé  dans c tt  règl . Il n' st pas néc ssair  qu  c  symbol  apparaiss  aill urs dans l  règl . La syntax  du modificat ur  st : **%prec** t rminal−symbol  t  st écrit  à la suit  d s composants d  la règl . Son  ff t  st d'af- f ct r à la règl  la précéd nc  du symbol  t rminal,  n écrasant la précéd nc  qui aurait pu êtr  déduit  classiqu m nt. La précéd nc  d  la règl  altéré  st utilisé  pour résoudr  l s conflits (voir la sous-sous-s ction 2.3.7 sur l s opérat urs d  précéd nc ).

Voici figur  2.29 comm nt **%prec** résoudr  l  problèm  du moins unair . On déclar  d'abord

un précéd nc pour l symbol t rminal fictif UMINUS. Il n'y a pas d tok n d c typ mais l symbol s rt pour marqu r sa précéd nc :

```
%...
%left PLUS MINUS
%left MULTIPLY
%left UMINUS
....
%  xp:      ...
%           |  xp MINUS xp
%           ...
%           | MINUS xp %prec UMINUS
```

FIG. 2.29 – Définition d s précéd nc s d cont xt

### 2.5.5 Etats du parser

La fonction *yyparse* st implém nté n utilisant un machin à états finis. L s val urs "push d" sur la pil du pars ur n sont pas simpl m nt d s cod s d typ d tok n, ll s r prés nt nt d s séqu nc s ntièr s d symbol s t rminaux t non-t rminaux sur ou près du somm t d la pil . L'état courant accumul tout s l s informations conc rnant l s ntré s précéd nt s qui sont significativ s pour décid r d c qui doit êtr fait nsuit .

Chaqu fois qu'un tok n à v nir st lu, l'état courant du pars r, av c l typ du tok n à v nir sont associés dans un tabl . La tabl d s ntré s p ut indiqu r "shift r l tok n à v nir". Dans c cas, il p ut aussi spécifi r l nouv l état du pars ur, qui st "push d" au somm t d la pil du pars r. La tabl d s ntré s p ut aussi indiqu r "réduir n utilisant la règl r". C qui signifi qu'un c rtain nombr d tok n ou group s sont dépilés t r mplacés par un group . En d'autr s t rm s, un nombr d'états st dépilé d la pil t un nouv l état st "push d".

Il xist un autr alt rnativ : la tabl p ut indiqu r qu l tok n à v nir st rroné dans l'état courant. C la provoqu un rr ur d proc ssing (voir la s ction sur la récupération d' rr ur : 2.6).

### 2.5.6 Les conflits Reduce/Reduce

Un conflit r duc /r duc arriv s'il y a d ux ou plusi urs règl s qui s'appliqu nt à la mêm séqu nc d' ntré . Général m nt, c la indiqu un séri us rr ur dans la grammair .

Par x mpl figur 2.30, voici un t ntativ pour définir un séqu nc d zéro ou plus group s d mots.

L' rr ur st un ambiguïté : il y a plus d'un moy n d pars r un simpl mot dans un séqu nc . Il pourrait êtr réduit comm mayb word puis comm s qu nc à l'aid d la d uxièm règl . Mais égal m nt,"ri n du tout" pourrait êtr réduit n s qu nc à l'aid d la pr mièr règl t êtr combiné av c l mot n utilisant la 3èm règl d s qu nc .

Il y a égal m nt plusi urs moy ns d réduir "ri n du tout" n un s qu nc . C la p ut êtr fait dir ct m nt à l'aid d la pr mièr règl ou indir ct m nt à l'aid d la règl mayb word puis la 2èm règl .

On pourrait p ns r qu c tt distinction n fait pas d différ nc , puisqu ça n chang pas l fait qu' n ntré soit valid ou non. Mais c la aff ct qu ll s actions p uv nt êtr appliqué s. Un ordr d parsing appliqu l s actions d la s cond règl , l'ordr d'un autr parsing appliqu

```
s qu nc : /*  mpty */    { printf " mpty_s qu nc \n" }
        | mayb word      {}
        | s qu nc  word { printf "add d_word_%s\n" $2 }
        ;

mayb word: /*  mpty */  { printf " mpty_mayb word\n" }
        | word           { printf "singl _word_%s\n" $1 }
        ;
```

FIG. 2.30 – Err ur d  grammair  provoquant un conflit r duc /r duc

l s actions d  la pr mièr  puis d  la troisièm  règl . Dans c t  x mpl , la sorti  du programm  chang .

Ocamlyacc résout l s conflits r duc /r duc  n choisissant d'utilis r l s règl s dans l'ordr d'apparition dans la grammair , mais c la p ut êtr  risqué d  s  r pos r s ul m nt là d ssus. Chaqu  conflit r duc /r duc  doit êtr  étudié  t éliminé. Voici figur  2.31 un moy n corr ct d définir `s qu nc` :

```
s qu nc : /*  mpty */    { printf " mpty_s qu nc \n" }
        | s qu nc  word          { printf "add d_word_%s\n" $2 }
        ;
```

FIG. 2.31 – Résolution du conflit r duc /r duc

Voici figur  2.32 un autr  x mpl  d' rr ur commun  qui conduit à un conflit r duc /r duc  :

```
s qu nc : /*  mpty */
        | s qu nc  words
        | s qu nc  r dir cts
        ;

words:    /*  mpty */
        | words word
        ;

r dir cts:/*  mpty */
        | r dir cts r dir ct
        ;
```

FIG. 2.32 – Ex mpl  2 d'un conflit r duc /r duc

L'int ntion ici  st d  définir un  séqu nc  qui p ut cont nir soit d s group s `word` soit d s group s `r dir ct` . L s définitions individu ll s d  `s qu nc` , `words`  t `r dir cts` n'ont pas d' rr urs, mais l'ass mblag  d  tout s apport  un  ambiguïté subtil  : un  ntré  vid  p ut êtr  parsé  par un  infinité d  moy n !

Ici, "ri n du tout" p ut  st un `words` , ou 2 `words` à la suit  ou 3 ou n'import  qu l nombr . "ri n du tout" p ut tout aussi êtr  `r dir cts` un , d ux ou n'import  qu l nombr  d  fois. Ou `words` suivi d  trois `r dir cts`  t d'autr s `words` ...

Voici donc figur  2.33 un pr mi r moy n d  corrig r c s règl s,  n n'ayant qu'un niv au d s qu nc  :

```
s qu nc : /*  mpty */
        | s qu nc  word
        | s qu nc  r dir ct
        ;
```

Fig. 2.33 – Résolution 1 d  l' x mpl  2 d'un conflit r duc /r duc

Figur  2.34, on corrig  n évitant soit word soit r dir cts d'êtr  vid  :

```
s qu nc : /*  mpty */
        | s qu nc  words
        | s qu nc  r dir cts
        ;

words :   word
        | words word
        ;

r dir cts : r dir ct
        | r dir cts r dir ct
        ;
```

Fig. 2.34 – Résolution 2 d  l' x mpl  2 d'un conflit r duc /r duc

### 2.5.7 Mystérieux Reduce/Reduce Conflits

Il arriv  qu  l s conflits r duc /r duc  puiss nt s  produir  sans dét ction garanti . Figur  2.35 montr  un  x mpl  :

Il pourrait s mbl r qu  c tt  grammair  puiss  êtr  parsé  av c s ul m nt un simpl  tok n à v nir : lorsqu  param$_$sp c  st lu, un ID  st un nam  si "," ou " :" ou un **type** si un autr  ID  st à la suit . En d'autr s t rm s, c tt  grammair  st LR(1).

C p ndant, Ocamlyacc , comm  la plupart d s générat urs d  pars urs, n  p ut pas support r tout s l s grammair s LR(1). Dans la grammair  d  c t  x mpl  2.35, d ux cont xt s : après un ID  au début d'un param$_$sp c  t ID  au début d'un r turn$_$sp c , sont suffisamm nt similair s pour qu  Ocamlyacc p ns nt qu'ils soi nt id ntiqu s. Ils apparaiss nt id ntiqu s à Ocamlyacc car l  mêm  ns mbl  d  règl s pourrait êtr  activé (c ll  pour réduir  n un nam  t c ll  pour réduir  n un **type** . Ocamlyacc n  p ut pas dét rmin r à c  mom nt d  la génération du pars ur, qu  l s règl s néc ssit ront un tok n à v nir différ nt dans l s d ux cont xt s, il construit donc un s ul état d  pars r pour  ux. Or, combin r l s d ux cont xt s provoqu  un conflit ultéri ur m nt. En t rm  d  pars r, c  cas illustr  qu'il n  s'agit pas d'un  grammair  LALR(1).

Il  st plutôt compl x  d  corrig r Ocamlyacc pour lui p rm ttr  d  pars r d s grammair s LR(1) qui n  sont pas LALR(1). L s générat urs d  pars ur qui l  font ont t ndanc  à génér r d s pars urs très très gros. En pratiqu , on préfèr  laiss r Ocamlyacc t l qu'il  st.

```
%token ID COMMA COLON

%%
d f:     param_sp c  r turn_sp c COMMA
         ;
param_sp c:
               type
         |     nam _list COLON type
         ;
r turn_sp c:
               type
         |     nam  COLON type
         ;
type:          ID
         ;
nam :          ID
         ;
nam _list:
               nam
         |     nam  COMMA nam _list
         ;
```

FIG. 2.35 – Conflit r duc /r duc non dét ctabl

Lorsqu l problèm LR(1) s prés nt , on p ut souv nt l résoudr n id ntifiant l s d ux états d pars ur qui prêt nt à confusions t ajout r qu lqu chos qui l s f ront s distingu r. Dans l' x mpl précéd nt, ajout r un règl à r turn$_$sp c comm dans l figur 2.36 p rm t d résoudr l problèm :

```
%token BOGUS
...
%%
...
r turn_sp c:
               type
         |     nam  COMMA type
         /* This rul  is n v r us d.  */
         |     ID BOGUS
         ;
```

FIG. 2.36 – Résolution d'un conflit r duc /r duc non dét ctabl

C la corrig l problèm n introduisant la possibilité d'activ r un règl supplém ntair dans l cont xt après ID au début du r turn$_$sp c . C tt règl n' st pas activ dans l cont xt corr spondant d param_sp c , t d c tt façon l s d ux cont xt s r çoiv nt d s états d pars ur différ nts. Du mom nt qu l tok n BOGUS n' st pas généré par Ocamll x , la règl ajouté n modifi pas la façon d pars r l' ntré .

Dans c t x mpl particuli r, il y a un autr moy n d résoudr l problèm : réécrir la

règl d r turn$_$sp c afin d'utilis r ID dir ct m nt, autr m nt qu via nam . C la p rm t qu l s d ux cont xt s ai nt d s ns mbl s d règl s activ s différ nts, puisqu la règl pour r turn$_$sp c activ la règl modifié pour r turn$_$sp c plutôt qu c ll pour nam . C ci st prés nté figur 2.37 :

```
param_sp c:
              type
        |     nam _list COMMA type
        ;
r turn_sp c:
              type
        |     ID COMMA type
        ;
```

Fig. 2.37 – Résolution d'un conflit r duc /r duc non dét ctabl - 2èm possibilité

## 2.6 Error récupération

En général, il n faut pas qu'un programm s'int rrompr sur un rr ur d parsing. Par x mpl , un compilat ur d vrait résist r aux rr urs suffisamm nt pour pars r l r st du fichi r d' ntré à la r ch rch d' rr urs ; un calculatric d vrait acc pt r un autr xpr ssion.

Lorsqu l'on fait du parsing av c un simpl command int ractiv où chaqu ntré st constitué d'un s ul lign , il p ut êtr suffisant d'avoir l'app lant qui attrap l' xc ption t qui ignor l r st d la lign d' ntré lorsqu'un rr ur survi nt ( t on app ll la fonction d parsing à nouv au pour continu r sur un autr lign ). Mais c la n' st pas adapté pour un compilat ur, parc qu'il "oubli " tout l cont xt syntaxiqu qui a am né à l' rr ur. Un rr ur d syntax dans un fonction du compilat ur n d vrait pas provoqu r qu la "lign " suivant soit traité d la mêm manièr qu si c'était l début du fichi r sourc .

On p ut donc définir comm nt résist r à un rr ur d syntax n écrivant d s règl s qui r connaiss nt un tok n spécial d' rr ur. C' st un symbol t rminal qui st rés rvé à la récupération d' rr ur. L pars ur Ocamlyacc génèr un tok n d' rr ur lorsqu s produit un rr ur d syntax ; si on définit un règl qui r connaît c tok n dans l cont xt , l parsing p ut alors continu r.

Par x mpl , figur 2.38 :

```
stmnts:  /* mpty string */                {}
       | stmnts NEWLINE                   {}
       | stmnts xp NEWLINE      {}
       | stmnts rror NEWLINE    {}
```

Fig. 2.38 – Rattrapag d' rr ur

La quatrièm règl dans c t x mpl dit qu' rror suivi d'un nouv ll lign NEWLINE s ra un combinaison valid à n'import qu l stmnts .

Qu s pass -t-il si un rr ur d syntax s produit au mili u d'un xp ? La règl d récupération d' rr ur, int rprété strict m nt, va appliqu r la séqu nc précis d s stmnts , un rror t un NEWLINE . Si un rr ur s produit au mili u d'un xp , il y aura probabl m nt

d s tok ns supplém ntair s  t d s sous- xpr ssions dans la pil  après l  d rni r stmnts, t il y aura d s tok ns à lir  avant la prochain NEWLINE. La règl  n' st donc pas applicabl  normal m nt.

Mais Ocamlyacc p ut forc r la situation d  façon à corr spondr  à la règl , n éliminant un parti  du cont xt  sémantiqu  t un parti  d  l' ntré . Pr mièr m nt, Ocamlyacc élimin  l s états  t l s obj ts d  la pil  jusqu'à r v nir à l'état dans l qu l l tok n rror st acc pté. (C la signifi  qu  l s sous- xpr ssions déjà parsé s sont éliminé s,  n r tournant au d rni r stmnts compl t.) A c  niv au, l tok n rror p ut êtr  shifté. Alors, si l'anci n tok n à v nir n' st pas acc ptabl  pour êtr  shifté  n suivant, l  pars ur lit l s tok ns  t l s élimin nt jusqu'à trouv r un tok n qui  st acc ptabl . Dans c t  x mpl  2.38, Ocamlyacc lit  t élimin  l s  ntré s jusqu'à la prochain NEWLINE afin qu  la quatrièm  règl  puiss  s'appliqu r.

L  choix d s règl s d' rr ur dans un  grammair  st un choix d  stratégi  pour la récupération d' rr ur. Un  stratégi  simpl  t util  st simpl m nt d  skip r l  r st  d  la lign  d' ntré courant  ou d'état courant si un  rr ur  st dét cté , comm  l montr  l' x mpl  d  la figur  2.39 :

```
stmnt:  rror SEMICOLON  {}      /*  n cas d' rr ur, skip r jusqu'au  ;
    */
```

Fig. 2.39 – Stratégi  d  récupération d' rr ur

Il  st aussi util  d  récupér r jusqu'au délimit ur-f rmant corr spondant à un délimit ur-ouvrant qui aurait déjà été parsé. Sinon, l  délimit ur-f rmant apparaîtra probabl m nt isolé  t génér ra un autr  m ssag  d' rr ur ( rroné), voir figur  2.40 :

```
primary:  LPAREN  xpr RPAREN    {}
        | LPAREN  rror RPAREN    {}
        ...
        ;
```

Fig. 2.40 – Err ur causé  ntr  d ux délimit urs

L s stratégi s d  récupération d' rr ur sont néc ssair m nt d s t ntativ s. Lorsqu'un  t ntativ  échou , un  rr ur d  syntax  mèn  souv nt à un autr  ... Dans l' x mpl  précéd nt, la règl  d  récupération d' rr ur suppos  qu'un  rr ur  st du  à un mauvais  ntré  à l'intéri ur d'un stmnts. Supposons qu'un $;$ soit inséré au mili u d'un stmnts valid . Après avoir utilisé l s règl s d  récupération d' rr ur d  la pr mièr  rr ur, un autr  rr ur d  syntax  st trouvé  nsuit  : puisqu  l  t xt  qui suit l  $;$  rroné  st égal m nt un stmnts invalid .

Afin d'évit r l s m ssag s d' rr ur abondants, l  pars ur n  génér ra pas d  m ssag  d' rr ur pour un  rr ur d  syntax  qui s  produit just  après la pr mièr  ; s ul m nt après trois tok n d' ntré  consécutifs corr ct m nt shifté, l s m ssag s d' rr ur sont à nouv au générés l  cas échéant.

## 2.7 Debugger son parser

Pour débugg r l  pars r généré par Ocamlyacc :

– En utilisant la command ocamlyacc −v fil nam .mly, on génèr  l s informations d  parsing dans un fichi r app lé fil nam .output. C s informations consist nt  n un  tabl  d  parsing  t d s indications conc rnant l s conflits.

– M ttr l'option p à la variabl d' nvironn m nt OCAMLRUNPARAM
( xport OCAMLRUNPARAM='p' dans un sh ll bash.

L pars ur affich d s m ssag s à propos d s s actions t ll s qu shift r un tok n, réduir un règl ...

On p ut obt nir l numéros d la règl ou l s numéros d'états m ntionnés dans l s m ssag s dans fil nam .output.

## 2.8 Executer Ocamlyacc

La manièr habitu ll d'invoqu r ocamllyacc : ocamlyacc fil nam .mly . Où fil nam .mly st l nom du fichi r d grammair . L nom du fichi r d parsing st obt nu n r mplaçant l .mly par .ml. Ocamlyacc génèr un .ml à partir du .mly

### 2.8.1 Ocamlyacc Options

Voici un list d'options qui p uv nt êtr utilisé s av c Ocamlyacc :

– −v Par défaut, c tt option génèr un fichi r fil nam .output. Il conti nt d s informations d parsing t ll qu la d scription d s tabl s d parsing t un rapport d s ambiguïtés d la grammair .
– −bfnam P rm t d chang r l nom du fichi r d sorti par fnam .ml fnam .mli t fnam .output

# Chapitre 3

# Astuces

- Quand on fait du parsing " n lign " plutôt qu'à l'aid d'un fichi r, CTRL D pour fin d fichi r
- Utilis r < t > pour ntré t sorti du programm compl t : ./prog < ntr > sorti
- Pour vérifi r la grammair , lir .output
- Fair un Mak fil quand on travaill sur l proc ssus compl t (analys l xical , syntaxiqu t actions sémantiqu s...). Voir l' x mpl figur 3.1.

```
.PRECIOUS: %.mli

all: prog

prog: l x r.cmo pars r.cmo prog.cmo
	ocamlc -o $@ $^

%.cmo: %.ml
	ocamlc -c $<

%.cmi: %.mli
	ocamlc -c $<

%.ml: %.mll
	ocamll x $<

%.ml %.mli: %.mly
	ocamlyacc -v $<


l x r.cmo: pars r.cmi

cl an:
	rm -f *.cmi *.cmo
```

FIG. 3.1 – Ex mpl d Mak fil