The Scanner and The Parser

Akim Demaille Étienne Renault Roland Levillain
first.last@lrde.epita.fr

EPITA — École Pour l'Informatique et les Techniques Avancées

February 2, 2020

The Scanner and The Parser

Flex & Bison: Recalls

Flex

Flex:

- Lexical analyser
- Generates **scanners**
- Description in the form of regular expressions

Structure

```
%{
    [definitions]
%}
%%
[rules]
%%
[subprograms]
```

Flex – details

- Work on regular expressions **ONLY**
- Define regexps
  - Letter [a-zA-Z]
  - Number [0-9]
  - ...
- **yytext** the recognized text
- **yyleng** the size of the recognized text
- **yylex** starts the scanning
- **yywrap** called when the end of the text to analyze is encountered. Can be refined if needed.
- For each of matched regexps one can return and identifier (a token)
- *Bison will analyze this stream of tokens...*
  - Details later in this lecture for coupling flex and bison

Flex example – wc linux command

```
%{
#include <stdio.h>
static int chars_ = 0, lines_ = 0, words_ = 0;
%}

%%
\n          { ++chars_; ++lines_; }
[^ \t\n]+   { chars_ += yyleng; ++words_; }
.           { ++chars_; }
%%
int yywrap () {
  printf ("%7d %7d %7d\n", lines_, words_, chars_);
  return 1;
}
int main(){  yylex(); return 1; }
```

Bison

Bison[1]:

- Syntactic analyser
- Generates **parser**
- hand-by-hand with flex: read token to analyse the input stream

Structure

```
[definitions]
%%
[rules]
%%
[%%
  subprograms
]
```

[1] One should note that for the project we use a patched version of bison that supports variants `www.lrde.epita.fr/~tiger/download/bison-3.2.1.52-cd4f7.tar.gz`

LALR vs. GLR

LALR-1

- Default for bison
- Default behavior when a conflict occurs:
  - reduce/reduce: reduce to the first rule in conflict
  - shift/reduce: performs the shift
- During a shift/reduce conflict the parser may *miles away from the ball*

GLR

1. During a conflict the parser walks the two branches hoping that one of the two will win.
2. Maintains multiple parse stacks
3. Allows ambigous grammars (when required by the langage)

Ambiguous grammar

1. **Ambiguous grammar**: the parser cannot choose
2. **one branch succeeds**: the parser choose this one
3. **Syntax Error**: easy case, report error!

Ambiguous grammar: example 1

```
%%
exp:
      "if" exp "then" exp
    | "if" exp "then" exp "else" exp
    | "exp"
    ;
%%
```

Problem: Dangling Else

"else" should rattach to which "if"? Inner one or outer one?

if "exp" then if "exp" then "exp" else "exp"

Ambiguous grammar: example 1

```
%%
exp:
      "if" exp "then" exp
    | "if" exp "then" exp "else" exp
    | "exp"
    ;
%%
```

Problem: **Dangling Else**

"else" should rattach to which "if"? Inner one or outer one?

if "exp" then if "exp" then "exp" else "exp"

Ambiguous grammar: example 1 solution

```
%expect 0
%right "else" "then"
%%
exp:
      "if" exp "then" exp
    | "if" exp "then" exp "else" exp
    | "exp"
    ;
%%
```

- %right: choose shift
- %left: choose reduce
- %expect: the number of expected conflicts

Another solution would be to add "fi".

Ambiguous grammar: example 2

```
%%
exp:
        exp "?" exp ":" exp
    | "exp"
    ;
%%
```

Problem: Dangling ":"

":" should rattach to which "?" ? Inner one or outer one?

"exp" ? "exp" ? "exp" : "exp"

Ambiguous grammar: example 2

```
%%
exp:
        exp "?" exp ":" exp
    | "exp"
    ;
%%
```

Problem: **Dangling ":"**

":" should rattach to which "?" ? Inner one or outer one?

"exp" ? "exp" ? "exp" : "exp"

Ambiguous grammar: example 2 solution

```
%expect 0
%right ":" "?"
%%
exp:
      exp "?" exp ":" exp
    | exp
    ;
%%
```

Ambiguous grammar: example 3

```
%%
exp:
      typeid "[" exp "]" "of" exp
    | lvalue
    ;

lvalue:
      "id"
    | lvalue "[" exp "]"
    ;

typeid:
      "id"
    ;
%%
```

Ambiguous grammar: example 3 – problems

Problems

- typeid must be removed and "id" must be propagated

  "id" ["id" ["id" ["id"]]]

  - lvalues can be nested and the decision is taken on the “of” which is too late! There must be no question between "typeid" and "["

Ambiguous grammar: example 3 solution

```
%%
exp:
      "id" "[" exp "]" "of" exp
    | lvalue
    ;

lvalue:
      "id"
    | lvalue_b
    ;

lvalue_b:
      "id" of "[" exp "]"
    | lvalue_b "[" exp "]"
    ;
%%
```

Semantic Values

Coupling Parser and Scanner

Coupling flex and bison

Objectives

How to produce a stream of tokens in the scanner that will be analyzed by the parser?

Steps:

1. define token in the *parser.yy* using *%token TOKENNAME*
2. bison will produce an header file **that should be included into your scanner**
   - Your scanner can now see declared tokens
   - When the scanner match a regexp return to associated token in the flex's rule

Note: *%token* $<$ *XXX* $>$ *TOKENNAME* associates a token to a value (here XXX).

Calculator Example (in C)

Demo Time!

Variants (or how to move to C++)

The parser maintains a stack of types.

- In C, no problem use a **union**
- In C++, ...???

  $\Longrightarrow$ solution: **variants**

Variants

Variants are type safe unions:

- allocated directly within the object representation of the variant
- call destructors
- bison implements such a variants where the stack maintains the type (to call the correct destuctor).

Variants (or how to move to C++)

The parser maintains a stack of types.

- In C, no problem use a **union**
- In C++, ...???

  $\Longrightarrow$ solution: **variants**

Variants

Variants are type safe unions:

- allocated directly within the object representation of the variant
- call destructors
- bison implements such a variants where the stack maintains the type (to call the correct destuctor).

Parser

Reading tokens in the parser

```
// Allow storing object values.
%define api.value.type variant
// Generate functions to build tokens.
%define api.token.constructor
// Prefix all the tokens with TOK_ to avoid colisions.
%define api.token.prefix {TOK_}

%token <misc::symbol>   ID     "identifier"
%token <int>            INT    "integer"
%token <std::string>    STRING "string"

%printer { yyo << $$; } "identifier" "integer" "string"
%%
// ...
exp:
  INT     { $$ = new IntExp($1); }
| STRING  { $$ = new StringExp($1); }
//...
```

Scanner

Generating tokens from the scanner

```
id          [a-zA-Z][a-zA-Z_0-9]*
int         [0-9]+
string      "\""([^\\]|\\.)*"\""

%%

{id}        return parser::make_ID(yytext);
{int}       return parser::make_INT(atoi(yytext));
{string}    return parser::make_STRING(std::string(yytext + 1,
                                                   yyleng - 2));
```

or even (C++ 11)

```
{string}    return parser::make_STRING({yytext+1, yyleng-2});
```

Locations

Location tracking in the Scanner

Location tracking in Flex

What

loc the current location

How

%initial-action

run at the beginning of yylex

YY_USER_ACTION

once per scanner match

%{ ... %}

(after the first %%) pasted into yylex.

When at its top when first in the rule section:

- local variables
- code run once per yylex invocation

Location tracking in Flex

What

loc the current location

How

%initial-action

run at the beginning of yyparse.

YY_USER_ACTION

once per scanner match

%{ ... %}

(after the first %%) pasted into yylex.

When at its top when first in the rule section:

- local variables
- code run once per yylex invocation

Location tracking in Flex

What

loc the current location

How

`%initial-action`

run at the beginning of `yyparse`.

`YY_USER_ACTION`

once per scanner match

`%{ ... %}`

(after the first %%) pasted into `yylex`.

When at its top when first in the rule section:

- local variables
- code run once per `yylex` invocation

Location tracking in Flex

What

loc the current location

How

%initial-action

run at the beginning of `yyparse`.

YY_USER_ACTION

once per scanner match

%{ ... %}

(after the first %%) pasted into `yylex`.

When at its top when first in the rule section:

- local variables
- code run once per `yylex` invocation

Location tracking in Flex

What

loc the current location

How

%initial-action

run at the beginning of `yyparse`.

YY_USER_ACTION

once per scanner match

%{ ... %}

(after the first %%) pasted into `yylex`.

When at its top when first in the rule section:

- local variables
- code run once per `yylex` invocation

Location tracking in Flex

What

`loc` the current location

How

`%initial-action`

run at the beginning of `yyparse`.

`YY_USER_ACTION`

once per scanner match

`%{ ... %}`

(after the first %%) pasted into `yylex`.

When at its top when first in the rule section:

- local variables
- code run once per `yylex` invocation

Location tracking in Flex

What

`loc` the current location

How

`%initial-action`

run at the beginning of `yyparse`.

`YY_USER_ACTION`

once per scanner match

`%{ ... %}`

(after the first %%) pasted into `yylex`.

When at its top when first in the rule section:

- local variables
- code run once per `yylex` invocation

Location tracking in Flex

What

`loc` the current location

How

`%initial-action`

run at the beginning of `yyparse`.

`YY_USER_ACTION`

once per scanner match

`%{ ... %}`

(after the first %%) pasted into `yylex`.

When at its top when first in the rule section:

- local variables
- code run once per `yylex` invocation

Location tracking in Flex

```
%{
  /* At each match, adjust the last column. */
# define YY_USER_ACTION  loc.columns(yyleng);
%}
/* ... */
%%
%{
  /* At each call, bring the tail to the head.  */
  loc.step();
%}
        /* Locations of blanks are ignored. */
[ \t]+  loc.step();

        /* Newlines change the current line number,
           but are ignored too. */
\n+     loc.line(yyleng); loc.step();
```

Location tracking in Flex

```
{id}        return parser::make_ID(yytext, loc);
{int}       return parser::make_INT(atoi(yytext), loc);
{string}    return parser::make_STRING({yytext+1, yyleng-2}, loc);
```

Location tracking in the Parser

Using the Location in the Parser

```
%define filename_type {const std::string}
%locations

%%

lvalue.big:
  ID "[" exp "]"
   { $$ = new SubscriptVar
              (@$, new SimpleVar(@1, $1), $3); }
| lvalue.big "[" exp "]"
   { $$ = new SubscriptVar(@$, $1, $3); }
;
```

Error Messages

```
%error-verbose
%%
// ...
%%
void
yy::parser::error(const location_type& l, const std::string& m)
{
  tp.error_ << misc::Error::parse
            << l << ": " << m << std::endl;
}
```

Improving the Scanner/Parser

Error Recovery

Error Recovery

- The `error` token in Yacc/Bison:
  1. dig in the stack to find a nice place
  2. throw away unpleasant lookaheads
  3. reduce as usual
- "Guard" it, put bounds around
- May introduce new conflicts.
- Do as if there were no error: generate dummy values
- Maybe introduce an `Error` class to prevent cascades of errors.

Error Recovery

- The `error` token in Yacc/Bison:
  1. dig in the stack to find a nice place
  2. throw away unpleasant lookaheads
  3. reduce as usual
- "Guard" it, put bounds around
- May introduce new conflicts.
- Do as if there were no error: generate dummy values
- Maybe introduce an `Error` class to prevent cascades of errors.

Error Recovery

- The `error` token in Yacc/Bison:
  1. dig in the stack to find a nice place
  2. throw away unpleasant lookaheads
  3. reduce as usual
- "Guard" it, put bounds around
- May introduce new conflicts.
- Do as if there were no error: generate dummy values
- Maybe introduce an `Error` class to prevent cascades of errors.

Error Recovery

- The `error` token in Yacc/Bison:
  1. dig in the stack to find a nice place
  2. throw away unpleasant lookaheads
  3. reduce as usual
- "Guard" it, put bounds around
- May introduce new conflicts.
- Do as if there were no error: generate dummy values
- Maybe introduce an `Error` class to prevent cascades of errors.

Error Recovery

- The `error` token in Yacc/Bison:
  1. dig in the stack to find a nice place
  2. throw away unpleasant lookaheads
  3. reduce as usual
- "Guard" it, put bounds around
- May introduce new conflicts.
- Do as if there were no error: generate dummy values
- Maybe introduce an `Error` class to prevent cascades of errors.

Error Recovery

- The `error` token in Yacc/Bison:
  1. dig in the stack to find a nice place
  2. throw away unpleasant lookaheads
  3. reduce as usual
- "Guard" it, put bounds around
- May introduce new conflicts.
- Do as if there were no error: generate dummy values
- Maybe introduce an `Error` class to prevent cascades of errors.

Error Recovery

- The `error` token in Yacc/Bison:
  1. dig in the stack to find a nice place
  2. throw away unpleasant lookaheads
  3. reduce as usual
- "Guard" it, put bounds around
- May introduce new conflicts.
- Do as if there were no error: generate dummy values
- Maybe introduce an `Error` class to prevent cascades of errors.

Error Recovery

- The `error` token in Yacc/Bison:
  1. dig in the stack to find a nice place
  2. throw away unpleasant lookaheads
  3. reduce as usual
- "Guard" it, put bounds around
- May introduce new conflicts.
- Do as if there were no error: generate dummy values
- Maybe introduce an `Error` class to prevent cascades of errors.

Error Recovery

parse/parsetiger.yy

```
// Reclaim the memory.
%destructor { delete $$; } exp
%%
exp:
  "nil"           { $$ = new NilExp(@$); }
| "(" exps ")"    { $$ = new SeqExp(@$, $2); }
| "(" error ")"   { $$ = new SeqExp(@$, new exps_t); }
// ...
```

Pure Parser

The Parsing Driver

- Information exchanged with the parser/scanner
  - Input data
    library path, debugging flags, etc.
  - Output data
    The `ast`, the error messages/status
  - Data maintained during the parsing
    Open files
- Coordination
  - Initialize/open the scanner
  - Parse
  - Close the scanner
- Introduce a parsing driver.

The Parsing Driver

- Information exchanged with the parser/scanner
  - Input data
    library path, debugging flags, etc.
  - Output data
    The `ast`, the error messages/status
  - Data maintained during the parsing
    Open files
- Coordination
  - Initialize/open the scanner
  - Parse
  - Close the scanner
- Introduce a parsing driver.

The Parsing Driver

- Information exchanged with the parser/scanner
  - Input data
    library path, debugging flags, etc.
  - Output data
    The `ast`, the error messages/status
  - Data maintained during the parsing
    Open files
- Coordination
  - Initialize/open the scanner
  - Parse
  - Close the scanner
- Introduce a parsing driver.

The Parsing Driver

- Information exchanged with the parser/scanner
  - Input data
    library path, debugging flags, etc.
  - Output data
    The `ast`, the error messages/status
  - Data maintained during the parsing
    Open files
- Coordination
  - Initialize/open the scanner
  - Parse
  - Close the scanner
- Introduce a parsing driver.

The Parsing Driver

- Information exchanged with the parser/scanner
  - Input data
    library path, debugging flags, etc.
  - Output data
    The `ast`, the error messages/status
  - Data maintained during the parsing
    Open files
- Coordination
  - Initialize/open the scanner
  - Parse
  - Close the scanner
- Introduce a parsing driver.

The Parsing Driver

- Information exchanged with the parser/scanner
  - Input data
    library path, debugging flags, etc.
  - Output data
    The `ast`, the error messages/status
  - Data maintained during the parsing
    Open files
- Coordination
  - Initialize/open the scanner
  - Parse
  - Close the scanner
- Introduce a parsing driver.

The Parsing Driver

- Information exchanged with the parser/scanner
  - Input data
    library path, debugging flags, etc.
  - Output data
    The `ast`, the error messages/status
  - Data maintained during the parsing
    Open files
- Coordination
  - Initialize/open the scanner
  - Parse
  - Close the scanner
- Introduce a parsing driver.

The Parsing Driver

- Information exchanged with the parser/scanner
  - Input data
    library path, debugging flags, etc.
  - Output data
    The `ast`, the error messages/status
  - Data maintained during the parsing
    Open files
- Coordination
  - Initialize/open the scanner
  - Parse
  - Close the scanner
- Introduce a parsing driver.

The Parsing Driver

- Information exchanged with the parser/scanner
  - Input data
    library path, debugging flags, etc.
  - Output data
    The `ast`, the error messages/status
  - Data maintained during the parsing
    Open files
- Coordination
  - Initialize/open the scanner
  - Parse
  - Close the scanner
- Introduce a parsing driver.

The Parsing Driver (parse/tiger-parser.hh)

```
class TigerParser
{
public:
  /// Parse a Tiger program, return its AST.
  ast::Exp* parse_program(...);
  /// Parse a Tiger prelude, return the list of decs.
  ast::decs_list_type* parse_import(...);

private:
  /// The result of the parse.
  ast_type ast_;
  /// Parsing errors handler.
  misc::error error_;
  /// The source to parse.
  input_type input_;
  /// The file library for imports.
  misc::file_library library_;
};
```

The Parsing Driver (parse/tiger-parser.cc)

```
void TigerParser::parse_() {
  std::string* fn = boost::get<std::string>(&input_);
  misc::symbol filename(fn == nullptr ? ""
                        : *fn == "-" ? "standard input" : *fn);
  location_.initialize(&filename.name_get());
  std::shared_ptr<std::istream> in;
  if (fn_ == "-")
    in.reset(&std::cin, [](...){});
  else {
      in = std::make_shared<std::ifstream>(filename);
      // Check for errors...
    }
  scanner_->scan_open(*in);
  parser parser(*this);
  parser.set_debug_level(parse_trace_p_);
  decs_ = nullptr; exp_ = nullptr;
  parser.parse();
  scanner_->scan_close();
}
```

The Parser (parse/parsetiger.yy)

```
%define filename_type {const std::string}
%locations

// The parsing context.
%param { parse::TigerParser& tp }
```

Two Grammars in One

The Parser

parse/parsetiger.yy

```
%token SEED_IMPORT  "seed-import"
%token SEED_SOURCE  "seed-source"
%%
program:
  /* Parsing a source program. */
  "seed-source" exp                  { tp.exp_ = $2; }
| /* Parsing an imported file.  */
  "seed-import" "let" decs "end"     { tp.decs_ = $3; }
;
```

The Scanner: Wrapping yyflex

parse/scantiger.ll

```
int
yylex (yystype *yylval, yy::location *yyloc,
       parse::TigerParser& tp)
{
  if (tp.seed_)
    {
      int res = 0;
      std::swap(res, tp.seed_);
      return res;
    }
  else
    return flex_yylex(yylval, yyloc, tp);
}
```

The Scanner: Using the top of yyflex

parse/scantiger.ll

```
%%
%{
  if (tp.seed_)
    {
      int res = 0;
      std::swap(res, tp.seed_);
      return res;
    }
%}
```

Without Seeds

parse/parsetiger.yy

```
%%
program:
  /* Parsing a source program. */
  exp    { tp.exp_ = $1; }
| /* Parsing an imported file.  */
  decs   { tp.decs_ = $1; }
;
```

Reentrancy

Reentrant Flex Scanner

parse/scantiger.ll

```
void yyFlexLexer::scan_open_(std::istream& f)
{
  yypush_buffer_state(YY_CURRENT_BUFFER);
  yy_switch_to_buffer(yy_create_buffer(&f, YY_BUF_SIZE));
}

void yyFlexLexer::scan_close_()
{
  yypop_buffer_state();
}
```

Recursive Invocation of the Parser

parse/parsetiger.yy

```
importdec: "import" STRING
  {
    $$ = tp.parse_import(take($2), @$);
    // Parsing may have failed.
    if (!$$)
      $$ = new ast::decs_list_type;
  }
;
```

Symbols

cstats

cstats: Counting Symbols

```
g++ -E -P "$@" \
  | tr -cs '[:alnum:]_' '[\n*]' \
  | grep '^[[:alpha:]]' \
  | grep -v -E -w "$cxx_keywords" > $tmp.1
total=$(wc -lc < $tmp.1 \
        | awk '{print $1 " (" $2 " chars)"}')
sort $tmp.1 \
  | uniq -c \
  | sed 's/^   //;s/\t/ /' \
  | sort -rn >$tmp.2
unique=$(sed -s 's/.* //' $tmp.2 | wc -lc \
         | awk '{print $1 " (" $2 " chars)"}')
echo $total occurrences of $unique symbols.
sed 42q $tmp.2 \
  | pr --page-width=60 --column=3 --omit-header
rm -f $tmp.*
```

Lemon (as-of 2019-01-15)

```
15182 (78642 chars) occurrences of 1082 (8875 chars) symbols.
1868 gt                176 lineno              87 rule
 943 quot              155 lt                  87 h
 654 i                 149 cp                  82 np
 458 amp               148 s                   78 filename
 373 lemp              146 name                72 z
 347 rp                139 cfp                 71 fp
 306 n                 116 next                70 array
 297 psp               109 stp                 69 ht
 227 fprintf           108 p                   69 config
 199 sp                107 a                   62 errorcnt
 198 out               101 type                62 action
 187 j                  94 state               61 lem
 182 x                  91 symbol              60 d
 177 ap                 89 c                   56 data
```

GCC's C Parser

```
18958 (198353 chars) occurrences of 5835 (89396 chars) symbols.
2676 tree                89 new_type_flag    38 build_nt
1579 ttype               70 cpp_reader       36 itype
1123 yyvsp               69 build_tree_lis   36 build_x_binary
 909 yyval               67 parse            35 yychar
 358 ftype               65 y                35 frob_opname
 247 t                   61 obstack          35 d
 206 gt_pointer_ope      58 GTY              34 e
 200 common              46 identifier       33 tree_code_type
 192 size_t              43 error            33 operator_name_
 175 code                40 cp_global_tree   33 C
 171 tree_code           39 yyn              32 got_scope
 123 FILE                39 s                31 IDENTIFIER_NOD
  97 rtx                 39 lookups          30 tree_class_che
  95 type                38 TREE_LIST        30 global_trees
```

Tiger Compiler's Driver (as-of 1.70)

```
8544 (83423 chars) occurrences of 1320 (16098 chars) symbols.
 603 std              76 FILE              48 hash
 354 size_t           74 false_type        47 iterator_trait
 351 noexcept         73 declval           47 begin
 334 size_type        71 reverse_iterat    46 compare
 274 basic_string     64 difference_typ    46 char_traits
 268 type             62 pointer           42 integral_const
 202 constexpr        61 pair              41 allocator
 158 char_type        56 int_type          40 C
 153 forward          55 locale_t          39 first
 114 value            53 value_type        37 string
  96 decltype         53 move_iterator     37 replace
  94 true_type        52 move              37 basic_istream
  80 size             50 traits_type       36 exception_ptr
  77 base             48 length            35 wstring
```

Symbols

Save Time and Space

One unique occurrence for each identifier:

In C a simple `const char*`

In C++ an iterator in a `std::set`

"Set has the important property that inserting a new element into a set does not invalidate iterators that point to existing elements."

Save space fewer allocations

Save time fewer allocations, easier comparisons

Save nerves easier memory management

Save Time and Space

One unique occurrence for each identifier:

In C a simple `const char*`

In C++ an iterator in a `std::set`

"Set has the important property that inserting a new element into a set does not invalidate iterators that point to existing elements."

Save space fewer allocations

Save time fewer allocations, easier comparisons

Save nerves easier memory management

Save Time and Space

One unique occurrence for each identifier:

In C a simple `const char*`

In C++ an iterator in a `std::set`

"Set has the important property that inserting a new element into a set does not invalidate iterators that point to existing elements."

Save space fewer allocations

Save time fewer allocations, easier comparisons

Save nerves easier memory management

Save Time and Space

One unique occurrence for each identifier:

In C a simple `const char*`

In C++ an iterator in a `std::set`

"Set has the important property that inserting a new element into a set does not invalidate iterators that point to existing elements."

Save space fewer allocations

Save time fewer allocations, easier comparisons

Save nerves easier memory management

Save Time and Space

One unique occurrence for each identifier:

In C a simple `const char*`

In C++ an iterator in a `std::set`

"Set has the important property that inserting a new element into a set does not invalidate iterators that point to existing elements."

Save space fewer allocations

Save time fewer allocations, easier comparisons

Save nerves easier memory management

Save Time and Space

One unique occurrence for each identifier:

In C a simple `const char*`

In C++ an iterator in a `std::set`

"Set has the important property that inserting a new element into a set does not invalidate iterators that point to existing elements."

Save space	fewer allocations
Save time	fewer allocations, easier comparisons
Save nerves	easier memory management