Parse Tables with Fix & Foxi

Dr. Edgar Lederer

Fix & Foxi

- directory „FixFoxi“ contains:
  - directory „Grammars“, which contains the following file:
    - Grammar_BasicExpressions.sml (example grammar 1)
    - Grammar_Problems (example grammar 2)
  - directory „src“, which contains the following files (Fix & Foxi):
    - BASIC.sig
    - Basic.fun
    - SET.sig
    - Set.fun
    - FIX_FOXI_CORE.sig
    - FixFoxiCore.fun
    - FIX_FOXI.sig
    - FixFoxi.fun
    - use.sml
  - file „Slides_FixFoxi_V3.pptx“ (these slides)
  - file „Slides_FixFoxi_V3.pdf“ (these slides)
  - file „smlnj.exe“ (Standard ML of New Jersey, Version 110.0.7)

Fix & Foxi

- install Standard ML of New Jersey, Version 110.0.7
  - Standard ML is available
- invoke sml in directory „src“
- call use "use.sml";
  - Fix & Foxi is available
- call OS.FileSys.chDir "..\\Grammars";
  - for more convenient access to directory „Grammars“
- call use "Grammar_BasicExpressions.sml";
  - example grammar 1 is analysed

Grammar_BasicExpressions.sml

```
E ::= T E'              // expr
E' ::= + T E' | ε       // repADDOPRterm3
T ::= F T'              // term3
T' ::= * F T' | ε       // repMULTOPRfactor
F ::= id | ( E )        // factor

datatype term
  = ADDOPR
  | IDENT
  | LPAREN
  | MULTOPR
  | RPAREN

datatype nonterm
  = expr
  | repADDOPRterm3
  | term3
  | repMULTOPRfactor
  | factor
```

```
val productions =
[
(expr,
    [[N term3, N repADDOPRterm3]]),
(repADDOPRterm3,
    [[T ADDOPR, N term3, N repADDOPRterm3],
     []]),
(term3,
    [[N factor, N repMULTOPRfactor]]),
(repMULTOPRfactor,
    [[T MULTOPR, N factor, N repMULTOPRfactor],
     []]),
(factor,
    [[T IDENT],
     [T LPAREN, N expr, T RPAREN]])
]

val S = expr
```

Grammar_BasicExpressions.sml

```
val string_of_term =
  fn ADDOPR                      => "ADDOPR"
   | IDENT                       => "IDENT"
   | LPAREN                      => "LPAREN"
   | MULTOPR                     => "MULTOPR"
   | RPAREN                      => "RPAREN"

val string_of_nonterm =
  fn expr                        => "expr"
   | repADDOPRterm3              => "repADDOPRterm3"
   | term3                       => "term3"
   | repMULTOPRfactor            => "repMULTOPRfactor"
   | factor                      => "factor"

val string_of_gramsym = (string_of_term, string_of_nonterm)

val result = fix_foxi productions S string_of_gramsym
```

Fix & Foxi

- call ?();
  // help command: which information can be displayed
  - dispDiagnosis
  - dispTerms
  - dispNonterms
  - dispProds
  - dispS
  - dispNULLABLE
  - dispFIRST
  - dispFOLLOW
  - dispMM

Fix & Foxi

- call dispDiagnosis result;
  - val it = () : unit // everything is OK!
- call dispFIRST result; // line, entry

```
<expr>
  LPAREN
  IDENT
<repADDOPRterm3>
  ADDOPR
<term3>
  LPAREN
  IDENT
<repMULTOPRfactor>
  MULTOPR
<factor>
  LPAREN
  IDENT
```

Fix & Foxi

- call dispMM result; // line, column, entry

```
<expr>
  terminal LPAREN
    <term3> <repADDOPRterm3>
  terminal IDENT
    <term3> <repADDOPRterm3>
<repADDOPRterm3>
  terminal ADDOPR
    ADDOPR <term3> <repADDOPRterm3>
  $
    // ε
  terminal RPAREN
    // ε
...
```

Grammar_Problems.sml

```
E ::= T              // expr
E ::= E + T          // expr
T ::= F              // term3
T ::= F * T          // term3
F ::= id             // factor
F ::= ( E )          // factor

datatype term
  = ADDOPR
  | IDENT
  | LPAREN
  | MULTOPR
  | RPAREN

datatype nonterm
  = expr
  | term3
  | factor
```

```
val productions =
[
(expr,
    [[N term3],
     [N expr, T ADDOPR, N term3]]),
(term3,
    [[N factor],
     [N factor, T MULTOPR, N term3]]),
(factor,
    [[T IDENT],
     [T LPAREN, N expr, T RPAREN]])
]

val S = expr
```

Fix & Foxi

- call use "Grammar_BasicExpressions.sml";
  - example grammar 1 is analysed
- call dispDiagnosis result;

```
Warning: grammar not LL1:
<expr>
  terminal LPAREN
    <term3>
    <expr> ADDOPR <term3>
  terminal IDENT
    <term3>
    <expr> ADDOPR <term3>
<term3>
  terminal LPAREN
    <factor>
    <factor> MULTOPR <term3>
  terminal IDENT
    <factor>
    <factor> MULTOPR <term3>
val it = () : unit
```