```
%left arrow
%left cart_prod

%left and_s
%left or_s
%left not_s
%nonassoc relop conc rw_then rw_else
%left plus sub
%left prod div mod_op

%with decls.d_tree
 {subtype yystype is decls.d_tree.pnode;}

%%



S:
    PROG {sr_s($1);}
  ;



PROG:
    DECLS E {sr_prog($$, $1, $2);}
  ;



DECLS:
    DECLS DECL {sr_decls($$, $1, $2);}
  |                       {sr_decls($$);}
  ;



DECL:
    TYPEVAR_DECL  {sr_typevar_decl($$, $1);}
  | TYPE_DECL         {sr_type_decl($$, $1);}
  | FUNC_DECL         {sr_func_decl($$, $1);}
  | EQUATION           {sr_eq_decl($$, $1);}
  ;
```

```
TYPEVAR_DECL:
    rw_typevar LID TYPEDESC semicolon {sr_typevar($$, $2, $3);}
  ;

LID:
    LID comma identifier {sr_lid($$, $1, $3);}
  | identifier                {sr_lid($$, $1);}
  ;

TYPEDESC:
    colon DESC {sr_typedef($$, $1);}
  |                 {sr_typedef($$);}
  ;

DESC:
   o_par DESC c_par       {sr_desc($$, $2);}
  | DESC cart_prod DESC {sr_desc($$, $1, $3);}
  | DESC arrow DESC       {sr_desc_func($$, $1, $3);}
  | FCALL                        {sr_desc_id($$, $1);}
  ;


TYPE_DECL:
    rw_type identifier PARAMS colon ALTS semicolon {sr_type($$, $2, $3, $5);}
  ;

PARAMS:
    o_par EL c_par {sr_params($$, $2);}
  |                        {sr_params($$);}
  ;

EL:
    E                 {sr_el($$, $1);}
  | EL comma E {sr_el($$, $1, $3);}
  ;

ALTS:
    FCALL                        {sr_alts($$, $1);}
  | ALTS derivator FCALL {sr_alts($$, $1, $3);}
  ;

FCALL:
    identifier PARAMS {sr_fcall($$, $1, $2);}
  ;
```

```
FUNC_DECL:
    rw_dec identifier colon DESC semicolon {sr_func($$, $2, $4);}
  ;

EQUATION:
    pattern_s identifier PATTERN assig_s E semicolon {sr_equation($$, $2, $3, $5);}
  ;

PATTERN:
    o_par LMODELS c_par {sr_pattern($$, $2);}
  |                     {sr_pattern($$);}
  ;

LMODELS:
    LMODELS comma MODEL {sr_lmodels($$, $1, $3);}
  | MODEL               {sr_lmodels($$, $1);}
  ;

MODEL:
    E            {sr_model($$, $1);}
  | MODEL conc E {sr_model($$, $1, $3);}
  ;

E:
    o_par E c_par   {sr_e($$, $2);}
  | E plus E        {sr_plus($$, $1, $3);}
  | E sub E         {sr_sub($$, $1, $3);}
  | E prod E        {sr_prod($$, $1, $3);}
  | E div E         {sr_div($$, $1, $3);}
  | E mod_op E      {sr_mod($$, $1, $3);}
  | E and_s E       {sr_and($$, $1, $3);}
  | E or_s E        {sr_or($$, $1, $3);}
  | E relop E       {sr_relop($$, $1, $2, $3);}
  | not_s E         {sr_not($$, $2);}
  | sub E           {sr_usub($$, $2);}
  | COND            {sr_econd($$, $1);}
  | LIST_E          {sr_elist($$, $1);}
  | TUPLE           {sr_tuple($$, $1);}
  | LITERAL         {sr_elit($$, $1);}
  | FCALL           {sr_efcall($$, $1);}
  ;

COND:
    rw_if E rw_then E rw_else E {sr_cond($$, $2, $4);}
  ;
```

```
TUPLE:
    o_par LIST c_par {sr_tuple($$, $2);}
  ;

LIST_E:
    o_braq LIST c_braq {sr_list_e($$, $2);}
  ;

LIST:
    LIST comma E {sr_list($$, $1, $3);}
  |  E                   {sr_list($$, $1);}
  ;

LITERAL:
    chr_lit {sr_lit($$, $1);}
  |  int_lit  {sr_lit($$, $1);}
  |  str_lit  {sr_lit($$, $1);}
  ;


%%
package syntactic_a is
  procedure yyparse;
end syntactic_a;
with lexical_a, fun_dfa, fun_io, fun_shift_reduce, fun_goto, fun_Tokens, text_io,
semantic.c_tree;
use lexical_a, fun_dfa, fun_io, fun_shift_reduce, fun_goto, fun_Tokens, text_io,
semantic.c_tree;
package body syntactic_a is
  procedure YYError(S: in string) is
  begin
          Put_Line(S&" around line: "& Yy_Line_Number'Img);
  end YYError;
##
end syntactic_a;
```