1. Copyright.

2. *mpost_output* **Grammar.**
Write out the *cweb*'s representation of the grammar's code and *mpost*'s railroad diagrams. These appropriate files have the grammar's name along with extension "w" for *cweave* to compile and extension "mp" for *mpost* to draw from.

Enjoy the art work. All the grammar's vocabulary are cross referenced to accomodate the grammar writer. The railroad diagrams show left recursion and where the subrule has appropriate syntax directed code, it redraws the subrule with each element's position number below it. This aids in the writing of syntax directed code when individual parameters are referenced by use of the *sf* stack frame structure. Each element within the stack frame has a naming convention of *p#__* where the number sign is the element's position within the subrule sentence relative to 1.

Fodder to this grammar is provided by the tree container doing a prefix traverse. Why a prefix traverse? The parent node allowed me to declare the *mpost* variables before their use.

Caveat: Grammar c++ comments must observe *cweave* dictums: underscored items like variable names must be enclosed with | and the number sign "#" must use its literal of backslash #. Without these observances, *pdftex* will honk.

3. Fsm Cmpost_output class.

4. Cmpost_output constructor directive.

⟨ Cmpost_output constructor directive 4 ⟩ ≡
```
MPOST_CWEB_LOAD_XLATE_CHRS(this);
```

5. Cmpost_output op directive.

Due to a MS 2005 Visual Studio defficiency, part of the ctor's initialization is included here instead of all being done by `MPOST_CWEB_EMIT_PREFIX_CODE`. The reason is noted in the documentation of *o2externs*.

```
⟨ Cmpost_output op directive 5 ⟩ ≡
  no_subrules_per_rule_.push_back(0);
  time_t theTime = time(0);
  char *cTime = ctime(&theTime);
  gened_date_time_ += string(cTime);
  int n = gened_date_time_.find('\n');
  gened_date_time_[n] = ' ';
  mp_dimension_ += " abcdefghijklmnopqrstuvwxyz";
  w_fig_no_ = 0;
  rule_def_ = 0;
  subrule_def_ = 0;
  rule_no_ = 0;
  subrule_no_ = 0;
  elem_no_ = 0;
  no_of_rules_ = 0;
  no_of_subrules_ = 0;
  mp_filename_ += grammar_filename_prefix_.c_str( );
  mp_filename_ += ".mp";
  omp_file_.open(mp_filename_.c_str( ), ios_base::out | ios::trunc);
  if (¬omp_file_) {
    CAbs_lr1_sym * sym = new Err_bad_filename(mp_filename_.c_str( ));
    sym→set_who_created(__FILE__, __LINE__);
    parser__→add_token_to_error_queue(*sym);
    parser__→set_stop_parse(true);
    return;
  }
  w_filename_ += grammar_filename_prefix_.c_str( );
  w_filename_ += ".w";
  ow_file_.open(w_filename_.c_str( ), ios_base::out | ios::trunc);
  if (¬ow_file_) {
    CAbs_lr1_sym * sym = new Err_bad_filename(w_filename_.c_str( ));
    sym→set_who_created(__FILE__, __LINE__);
    parser__→add_token_to_error_queue(*sym);
    parser__→set_stop_parse(true);
    return;
  }
  w_index_filename_ += grammar_filename_prefix_.c_str( );
  w_index_filename_ += "_idx.w";
  ow_index_file_.open(w_index_filename_.c_str( ), ios_base::out | ios::trunc);
  if (¬ow_index_file_) {
    CAbs_lr1_sym * sym = new Err_bad_filename(w_index_filename_.c_str( ));
    sym→set_who_created(__FILE__, __LINE__);
    parser__→add_token_to_error_queue(*sym);
    parser__→set_stop_parse(true);
    return;
  }
  KCHARP t_file = "T.w";
  ow_t_file_.open(t_file, ios_base::out | ios::trunc);
```

```
if (¬ow_t_file_) {
  CAbs_lr1_sym * sym = new Err_bad_filename(t_file);
  sym→set_who_created(__FILE__, __LINE__);
  parser__→add_token_to_error_queue(*sym);
  parser__→set_stop_parse(true);
  return;
}
KCHARP err_file = "Err.w";
ow_err_file_.open(err_file, ios_base::out | ios::trunc);
if (¬ow_err_file_) {
  CAbs_lr1_sym * sym = new Err_bad_filename(err_file);
  sym→set_who_created(__FILE__, __LINE__);
  parser__→add_token_to_error_queue(*sym);
  parser__→set_stop_parse(true);
  return;
}
KCHARP lrk_file = "Lrk.w";
ow_lrk_file_.open(lrk_file, ios_base::out | ios::trunc);
if (¬ow_lrk_file_) {
  CAbs_lr1_sym * sym = new Err_bad_filename(lrk_file);
  sym→set_who_created(__FILE__, __LINE__);
  parser__→add_token_to_error_queue(*sym);
  parser__→set_stop_parse(true);
  return;
}
MPOST_CWEB_EMIT_PREFIX_CODE(this);
```

6. Cmpost_output user-declaration directive.

⟨ Cmpost_output user-declaration directive 6 ⟩ ≡

public: **char** *big_buf_*[BIG_BUFFER_32K]; *std* :: *map* < **int** , *std* :: *string* > *xlated_names_*; *std* :: *vector* < **int** > *no_subrules_per_rule_*;

std :: *string gened_date_time_*;
std :: *string mp_filename_*;
std :: *ofstream omp_file_*;
std :: *string w_filename_*;
std :: *string w_index_filename_*;
std :: *ofstream ow_file_*;
std :: *ofstream ow_index_file_*;
std :: *ofstream ow_t_file_*;
std :: *ofstream ow_err_file_*;
std :: *ofstream ow_lrk_file_*;
std :: *string grammar_filename_prefix_*;
std :: *string fq_filename_noext_*;

int *w_fig_no_*;
int *rule_no_*;
int *subrule_no_*;
int *elem_no_*;
int *no_of_rules_*;
int *no_of_subrules_*;

std :: *string rule_name_*;
std :: *string elem_name_*;
rule_def ∗ *rule_def_*;
T_subrule_def ∗ *subrule_def_*;
std :: *string mp_dimension_*;

static void *MPOST_CWEB_gen_dimension_name*(*Cmpost_output* ∗ *Fsm*, *std* :: *string* & *Mp_obj_name*, **int** *Dimension*);
static void *MPOST_CWEB_calc_mp_obj_name*(*Cmpost_output* ∗ *Fsm*, *std* :: *string* & *Mp_obj_name*, **int** *Elem_no*);
static void *MPOST_CWEB_wrt_mp_rhs_elem*(*Cmpost_output* ∗ *Fsm*, *std* :: *string* & *Elem_name*, *std* :: *string* & *Drw_how*);
static void *MPOST_CWEB_gen_sr_elem_xrefs*(*Cmpost_output* ∗ *Fsm*, AST ∗ *Subrule_tree*);
static void *MPOST_CWEB_woutput_sr_sdcode*(*Cmpost_output* ∗ *Fsm*, *T_subrule_def* ∗ *Subrule_def*);
static void *MPOST_CWEB_wrt_fsm*(*Cmpost_output* ∗ *Fsm*, *T_fsm_phrase* ∗ *Fsm_phrase*);
static void *MPOST_CWEB_wrt_rule_s_lhs_sdc*(*Cmpost_output* ∗ *Fsm*, *rule_def* ∗ *Rule_def*);
static bool *MPOST_CWEB_should_subrule_be_printed*(*Cmpost_output* ∗ *Fs*, *T_subrule_def* ∗ *Subrule_def*);
static void *MPOST_CWEB_xref_refered_T*(*Cmpost_output* ∗ *Fsm*, *refered_T* ∗ R_T);
static void *MPOST_CWEB_xref_refered_rule*(*Cmpost_output* ∗ *Fsm*, *refered_rule* ∗ *R_rule*);
static void MPOST_CWEB_LOAD_XLATE_CHRS(*Cmpost_output* ∗ *Fsm*);
static void MPOST_CWEB_EMIT_PREFIX_CODE(*Cmpost_output* ∗ *Fsm*);
static void *MPOST_CWEB_wrt_T*(*Cmpost_output* ∗ *Fsm*, *T_terminals_phrase* ∗ *T_phrase*);
static void *MPOST_CWEB_wrt_Err*(*Cmpost_output* ∗ *Fsm*, *T_error_symbols_phrase* ∗ *Err_phrase*);
static void *MPOST_CWEB_wrt_Lrk*(*Cmpost_output* ∗ *Fsm*, *T_lr1_k_phrase* ∗ *Lrk_phrase*);

std :: *list* < *NS_yacco2_terminals* :: *rule_def* ∗> *rules_for_fs_prt_*;

static void *MPOST_CWEB_crt_rhs_sym_str*(*state_element* ∗ *se*, *std* :: *string* ∗ *Xlated_str*);

7. Cmpost_output user-prefix-declaration directive.

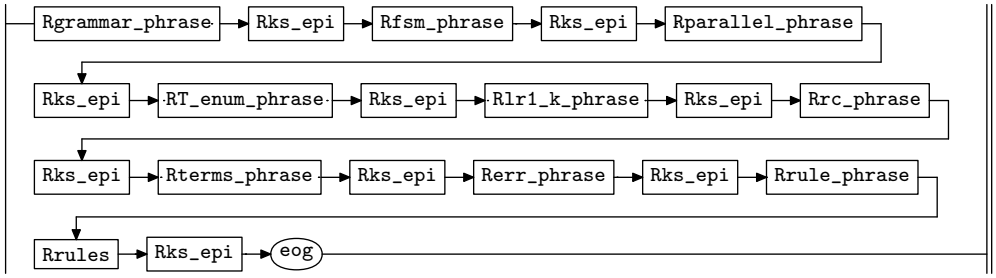
⟨ Cmpost_output user-prefix-declaration directive 7 ⟩ ≡

```
#include "time.h"
#include "o2_externs.h"
  extern void XLATE_SYMBOLS_FOR_cweave(const char *Sym_to_xlate, char *Xlated_sym);
  extern void PRT_RULE_S_FIRST_SET(std::ofstream & Ofile, NS_yacco2_terminals::rule_def * Rule_def);
  extern int MPOST_CWEB_xlated_symbol(AST * Sym, char *Xlated_sym);
  extern STATES_typeLR1_STATES;
  extern COMMON_LA_SETS_typeCOMMON_LA_SETS;
#include "cweave_fsm_sdc.h"
#include "cweave_lhs_sdc.h"
#include "cweave_sdc.h"
#include "cweave_T_sdc.h"
```

8. *Rmpost_output* **rule.**

Note how i use the sentence recognition to complete *mpost* figures. The individual figures are defined before their execution.

Rmpost_output



⟨ Rmpost_output subrule 1 op directive 8 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__->fsm_tbl__;
KCHARP mp_init_vars = "init_variables;";
fsm->omp_file_ << mp_init_vars << endl;
KCHARP mp_figure = "beginfig(%i);\n""␣␣drw_rule(%i);\n""endfig;";
KCHARP sr_figure = "beginfig(%i);\n""␣␣drw_rule_a_rhs(%i,%i);\n""endfig;";
int rule_no = 1;
int fig_no(0);
for ( ; rule_no <= fsm->no_of_rules_; ++rule_no) {
  ++fig_no;
  sprintf(fsm->big_buf_, mp_figure, fig_no, rule_no);
  fsm->omp_file_ << fsm->big_buf_ << endl;
  int no_srs = fsm->no_subrules_per_rule_[rule_no];
  int sr_no(1);
  for ( ; sr_no <= no_srs; ++sr_no) {
    ++fig_no;
    sprintf(fsm->big_buf_, sr_figure, fig_no, rule_no, sr_no);
    fsm->omp_file_ << fsm->big_buf_ << endl;
  }
}
KCHARP mp_end = "end;";
fsm->omp_file_ << mp_end << endl;
fsm->omp_file_.close( );
KCHARP w_fsc_file = "@**␣First␣Set␣Language␣for␣\\Olinker.\\fbreak\n""\\openin␣1␣=␣\"%s.f\
    sc\"\n"
  /* macro file open */
"\\ifeof␣1␣true\n"     /* test the result */
"␣\\closein␣1\n""\\else\n""␣␣␣␣\\closein␣1\n""␣␣␣␣\\listing{\"%s.fsc\"}""\\fi";
sprintf(fsm->big_buf_, w_fsc_file, fsm->fq_filename_noext_.c_str( ), fsm->fq_filename_noext_.c_str( ));
fsm->ow_file_ << fsm->big_buf_ << endl;
KCHARP w_lr1_states = "@**␣Lr1␣State␣Network.\\fbreak\n""\\statecolumnspace␣%s";
sprintf(fsm->big_buf_, w_lr1_states, "␣");
fsm->ow_file_ << fsm->big_buf_ << endl;
KCHARP w_stateno =      /* stateno, vectored into symbol, conflict */
"\\stateno{%i}␣{%s}␣{%s}\n""\\halign{\n""\\span\\statetemplate\n""\\statetitle";
char vectored_into[Max_cweb_item_size];
```

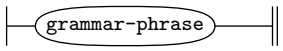
```
char rule_name[Max_cweb_item_size];
char subrule_sym[Max_cweb_item_size];
char cur_sym[Max_cweb_item_size];
char nxtsym_1[Max_cweb_item_size];
char nxtsym_2[Max_cweb_item_size];
char state_type[Max_cweb_item_size];
char la_set_no[Max_cweb_item_size];
string w_possible_ar_code;
KCHARP la_set_no_template = "%i";
KCHARP state_type_template = "%s";
KCHARP three_symbols_string_template = "%s␣%s␣%s";      /* 3 symbols: current and 2 la */
KCHARP w_stateno_subrule = "{%s}&\n"      /* closure or transitive */
"{%s}&\n"      /* rule */
"{%i\\hss}&\n"      /* rule no */
"{%i\\hss}&\n"      /* subrule no */
"{%i\\hss}&\n"      /* element pos */
"{%s}&\n"      /* current symbol */
"{%i\\hss}&\n"      /* born state no */
"{%i\\hss}&\n"      /* goto state no */
"{%i\\hss}&\n"      /* reduced state no */
"{%s\\hss}\\cr";      /* la set if reduced */
int fnd_reduced_stateno(0);
STATES_ITER_type si = LR1_STATES.begin( );
STATES_ITER_type sie = LR1_STATES.end( );
for ( ; si ≠ sie; ++si) {      /* walk the states */
  vectored_into[0] = (char) 0;
  w_possible_ar_code.clear( );
  state_type[0] = (char) 0;
  state * cur_state = *si;
  if (cur_state→state_no_ ≠ 1) {      /* state 1 vector into is empty */
    XLATE_SYMBOLS_FOR_cweave(cur_state→entry_symbol_literal( ), vectored_into);
  }
  switch (cur_state→state_type_) {
  case 0:
    {      /*shift only */
      sprintf(state_type, state_type_template, "\\Shiftonly");
      break;
    }
  case 1:
    {
      sprintf(state_type, state_type_template, "\\Reduceonly");
      break;
    }
  case 2:
    {
      sprintf(state_type, state_type_template, "\\ShiftReduce");
      break;
    }
  case 3:
    {
      sprintf(state_type, state_type_template, "\\MultipleReduces");
```

```
      break;
    }
  case 4:
    {
      sprintf(state_type, state_type_template, "\\ShiftandMultipleReduces");
      break;
    }
  }
  w_possible_ar_code.append(vectored_into);
  if (cur_state→vectored_into_by_elem_ ≡ LR1_PARALLEL_OPERATOR) {
      /* add arbitration code? to vectored_into string */
    if (cur_state→arbitrator_name_ ≠ 0) {
      w_possible_ar_code += "\\␣arbitration-code:␣\\␣|";
      w_possible_ar_code += cur_state→arbitrator_name_→c_str( );
      w_possible_ar_code += "|";
    }
    else {
      w_possible_ar_code += "\\␣arbitration-code:␣\\␣\\epsilon";
    }
  }
  sprintf(fsm→big_buf_, w_stateno, cur_state→state_no_, w_possible_ar_code.c_str( )      /* vectored_into */
  , state_type);
  fsm→ow_file_ ≪ fsm→big_buf_ ≪ endl;
  S_VECTORS_ITER_type svi = cur_state→state_s_vector_.begin( );
  S_VECTORS_ITER_type svie = cur_state→state_s_vector_.end( );
  string rhs_syms_str;
  for ( ; svi ≠ svie; ++svi) {
    S_VECTOR_ELEMS_ITER_type seli = svi→second.begin( );
    S_VECTOR_ELEMS_ITER_type selie = svi→second.end( );
    for ( ; seli ≠ selie; ++seli) {
      rhs_syms_str.clear( );
      state_element * se = *seli;
      rule_def * rd = se→subrule_def_→its_rule_def( );
      T_subrule_def * srd = se→subrule_def_;
      rule_name[0] = (char) 0;
      subrule_sym[0] = (char) 0;
      cur_sym[0] = (char) 0;
      la_set_no[0] = (char) 0;
      XLATE_SYMBOLS_FOR_cweave(rd→rule_name( )→c_str( ), rule_name);
      int elem_pos(0);
      if (se→previous_state_ ≡ 0) elem_pos = 1;
      elem_pos = MPOST_CWEB_xlated_symbol(se→sr_element_, cur_sym);
      if (se→goto_state_ ≡ 0) {      /* reducing subrule */
        sprintf(la_set_no, la_set_no_template, se→common_la_set_idx_ + 1);
      }
      else {      /* no la set */
      }
      if (se→reduced_state_ ≠ 0) {
        fnd_reduced_stateno = se→reduced_state_→state_no_;
      }
      else {      /* find from merged into */
        fnd_reduced_stateno = −1;
```

```
        if (se→next_state_element_→reduced_state_ ≠ 0) {
          fnd_reduced_stateno = se→next_state_element_→reduced_state_→state_no_;
        }
      }
      fsm→MPOST_CWEB_crt_rhs_sym_str(se, &rhs_syms_str);
      sprintf(fsm→big_buf_, w_stateno_subrule, (elem_pos ≡ 1) ? "c" : "t", rule_name, rd→rule_no( ),
          srd→subrule_no_of_rule( ), elem_pos, rhs_syms_str.c_str( ), se→closure_state_→state_no_,
          (se→goto_state_ ≠ 0) ? se→goto_state_→state_no_ : 0, fnd_reduced_stateno, la_set_no);
      fsm→ow_file_ ≪ fsm→big_buf_ ≪ endl;
    }
  }
  fsm→ow_file_ ≪ "}" ≪ endl;
}
KCHARP w_end = "@** Index.";
fsm→ow_file_ ≪ w_end ≪ endl;
fsm→ow_file_.close( );
```

9. *Rgrammar_phrase* **rule.**

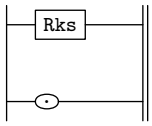
Rgrammar_phrase



10. *Rks_epi* **rule.**

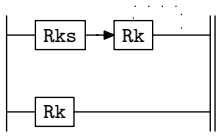
Due to technical problems with the *convertMPtoPDF* macro, the epsilon subrule is drawn with a dot.

Rks_epi



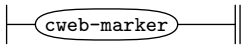
11. *Rks* **rule.**

Rks



12. *Rk* **rule.**

Rk

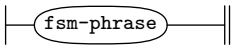


⟨ Rk subrule 1 op directive 12 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__→fsm_tbl__;
WRT_CWEB_MARKER(&fsm→ow_file_, sf→p1__→ast( ));
```

13. *Rfsm_phrase* **rule.**

```
Rfsm_phrase
```

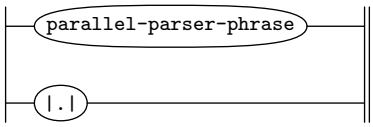


⟨ Rfsm_phrase subrule 1 op directive 13 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__→fsm_tbl__;
fsm→MPOST_CWEB_wrt_fsm(fsm, sf→p1__);
```

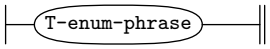
14. *Rparallel_phrase* **rule.**

```
Rparallel_phrase
```



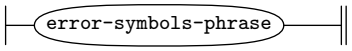
15. *RT_enum_phrase* **rule.**

```
RT_enum_phrase
```



16. *Rerr_phrase* **rule.**

```
Rerr_phrase
```

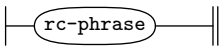


⟨ Rerr_phrase subrule 1 op directive 16 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__→fsm_tbl__;
fsm→MPOST_CWEB_wrt_Err(fsm, sf→p1__);
```

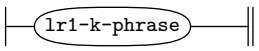
17. *Rrc_phrase* **rule.**

```
Rrc_phrase
```



18. *Rlr1_k_phrase* **rule.**

```
Rlr1_k_phrase
```

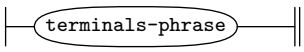


⟨ Rlr1_k_phrase subrule 1 op directive 18 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__→fsm_tbl__;
fsm→MPOST_CWEB_wrt_Lrk(fsm, sf→p1__);
```

19. *Rterms_phrase* **rule.**

```
Rterms_phrase
```



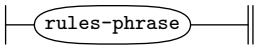
⟨ Rterms_phrase subrule 1 op directive 19 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__→fsm_tbl__;
fsm→MPOST_CWEB_wrt_T(fsm, sf→p1__);
```

20. *Rrule_phrase* **rule.**

Define the number of rules for *mpost*.

`Rrule_phrase`

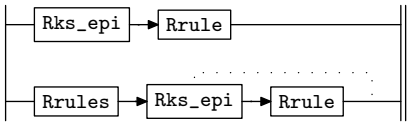


⟨ Rrule_phrase subrule 1 op directive 20 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__->fsm_tbl__;
fsm->no_of_rules_ = sf->p1__->rules_alphabet( )->size( );
KCHARP mp_no_of_rules = "no_of_rules␣:=␣%i;";
sprintf (fsm->big_buf_, mp_no_of_rules, fsm->no_of_rules_);
fsm->omp_file_ << fsm->big_buf_ << endl;
```

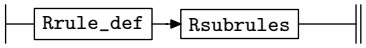
21. *Rrules* **rule.**

`Rrules`



22. *Rrule* **rule.**

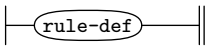
`Rrule`



23. *Rrule_def* **rule.**

Define rule for *mpost* and *cweb*. To help the grammar writer, walk the rule's subrules to generate a *cweb* cross-reference per subrule element. Originally this was done at the subrule level to be specific to the subrule entry instead of at the overall rule level. This meant that each subrule had to be redrawn with a *cweb* directive even when there was no syntax directed code present. Just ugly is my take so i follow Occam's dictum.

Rrule_def

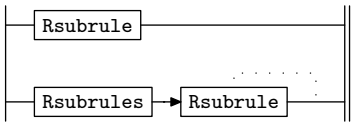


⟨ Rrule_def subrule 1 op directive 23 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__->fsm_tbl__;
fsm->rule_def_ = sf->p1__;
fsm->rules_for_fs_prt_.push_back(fsm->rule_def_);
KCHARP rname = fsm->rule_def_->rule_name( )->c_str( );
++fsm->w_fig_no_;
++fsm->rule_no_;
fsm->subrule_no_ = 0;
KCHARP mp_rule_names = "rule_names[%i].literal␣:=␣\"%s\";""rule_names[%i].vnam\
    e␣:=␣\"%s\";";
sprintf(fsm->big_buf_, mp_rule_names, fsm->rule_no_, rname, fsm->rule_no_, rname);
fsm->omp_file_ << fsm->big_buf_ << endl;
T_subrules_phrase * sr_ph = fsm->rule_def_->subrules( );
fsm->no_of_subrules_ = sr_ph->no_subrules( );
KCHARP mp_rule_s_no_rhs = "rule_s_no_rhs[%i]␣:=␣%i;";
sprintf(fsm->big_buf_, mp_rule_s_no_rhs, fsm->rule_no_, fsm->no_of_subrules_);
fsm->omp_file_ << fsm->big_buf_ << endl;
fsm->no_subrules_per_rule_.push_back(fsm->no_of_subrules_);      /* [rule #] */
KCHARP rule_cweb = "@*2␣|@!%s|␣rule.\\fbreak\n";
sprintf(fsm->big_buf_, rule_cweb, rname);
fsm->ow_file_ << fsm->big_buf_;
if (fsm->rule_def_->cweb_marker( ) ≠ 0) {
  WRT_CWEB_MARKER(&fsm->ow_file_, fsm->rule_def_->cweb_marker( ));
}
KCHARP rule_cweb_diagram = "\\fbreak\n""\\convertMPtoPDF{%s.%i}{1}{1}\n";
sprintf(fsm->big_buf_, rule_cweb_diagram, fsm->grammar_filename_prefix_.c_str( ), fsm->w_fig_no_);
fsm->ow_file_ << fsm->big_buf_;
std::vector < T_subrule_def *> *srules = sr_ph->subrules( );
std::vector < T_subrule_def *> ::iterator i = srules->begin( );
std::vector < T_subrule_def *> ::iterator ie = srules->end( );
for ( ; i ≠ ie; ++i) {
  T_subrule_def * srd = *i;
  fsm->MPOST_CWEB_gen_sr_elem_xrefs(fsm, srd->subrule_s_tree( ));
}
fsm->MPOST_CWEB_wrt_rule_s_lhs_sdc(fsm, fsm->rule_def_);
```

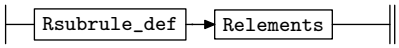
24. *Rsubrules* **rule.**

Rsubrules



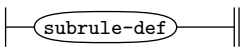
25. *Rsubrule* **rule.**

Rsubrule



26. *Rsubrule_def* **rule.**

`Rsubrule_def`



The *eosubrule* T has no meaning in the drawn subrule. It is just a specific end-of-subrule indicator. So it is removed from the rhs element count of the subrule for *mpost* processing so that it does not get drawn. The balance of the logic relates to *cweb*'s document: comments about the subrule and the appropriate *mpost* rendering.

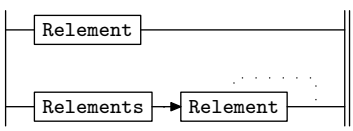
One subtlety is the forced drawing of a single subrule that also has a rule's lhs syntax directed code. Normally i save the redrawing of the subrule as the forest and the tree is the same. But having the rule syntax code requires to have the subrule directive. Without it there are 2 code directives back-to-back that honks.

⟨ Rsubrule_def subrule 1 op directive 26 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__->fsm_tbl__;
++fsm->w_fig_no_;
++fsm->subrule_no_;
fsm->elem_no_ = 0;
fsm->subrule_def_ = sf->p1__;
int no_elements = fsm->subrule_def_->no_of_elems( ) - 1;
if (no_elements ≡ 0) no_elements = 1;      /* epsilon subrule */
KCHARP mp_rule_s_subrule_no_elems = "rule_s_subrule_no_elems[%i][%i]:= %i;";
sprintf (fsm->big_buf_, mp_rule_s_subrule_no_elems, fsm->rule_no_, fsm->subrule_no_, no_elements);
fsm->omp_file_ ≪ fsm->big_buf_ ≪ endl;
if (fsm->MPOST_CWEB_should_subrule_be_printed(fsm, fsm->subrule_def_) ≡ false) return;
KCHARP rname = fsm->rule_def_->rule_name( )->c_str( );
KCHARP subrule_cweb = "@*3 |%s| 's subrule %i.\\fbreak\n";
if ((fsm->rule_def_->rule_lhs( ) ≠ 0) ∨ (fsm->no_of_subrules_ > 1)) {
  sprintf (fsm->big_buf_, subrule_cweb, rname, fsm->subrule_no_);
  fsm->ow_file_ ≪ fsm->big_buf_;
}
if (fsm->subrule_def_->cweb_marker( ) ≠ 0) {
  WRT_CWEB_MARKER(&fsm->ow_file_, fsm->subrule_def_->cweb_marker( ));
  cout ≪ "subrule's cweb marker for rule: " ≪ rname ≪ endl;
}
else {
  cout ≪ "no subrule's cweb marker for rule: " ≪ rname ≪ endl;
}
KCHARP subrule_cweb_diagram = "\\fbreak""\\convertMPtoPDF{%s.%i}{1}{1}\n";
if ((fsm->rule_def_->rule_lhs( ) ≠ 0) ∨ (fsm->no_of_subrules_ > 1)) {
  sprintf (fsm->big_buf_, subrule_cweb_diagram, fsm->grammar_filename_prefix_.c_str( ), fsm->w_fig_no_);
  fsm->ow_file_ ≪ fsm->big_buf_;
}
fsm->MPOST_CWEB_woutput_sr_sdcode(fsm, fsm->subrule_def_);
```

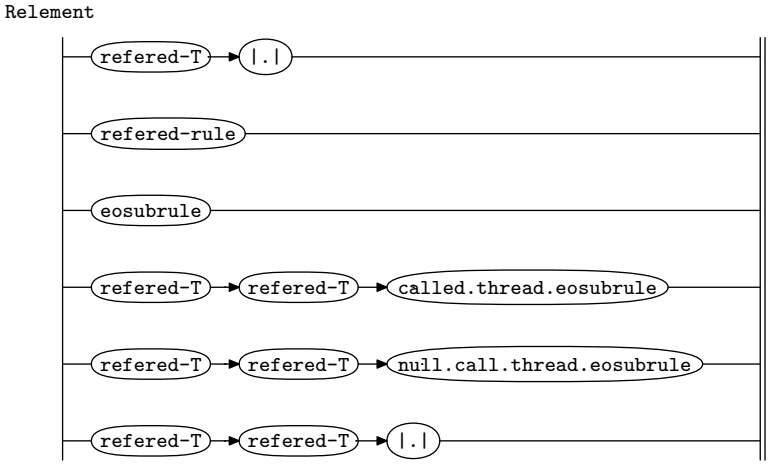
27. *Relements* **rule.**

`Relements`

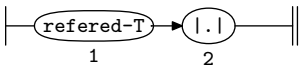


28. *Relement* **rule.**

Use of `|.|` to make grammar lr(1).



29. *Relement*'**s subrule 1.**

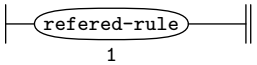


Why is there a test for \" within the T? Glad u asked as *mpost* has problems dealing with this legitimate escape sequence and the double quote sequence within the c string. So this was my 1st attempt to silence the honk. Unfortunately the char(34) *mpost* operator does not convert the 34 into the " character. For the moment i will leave this as is. U will see other attempts like this throughout the grammar.

⟨ Relement subrule 1 op directive 29 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__->fsm_tbl__;
++fsm->elem_no_;
std::string mp_obj_type("Circle_solid");
std::string mp_elem_name(sf->p1__->t_in_stbl()->t_def()->t_name()->c_str());
string::size_type x = mp_elem_name.find("\\\"");
if (x ≠ string::npos) {
  mp_elem_name.clear();
  mp_elem_name += "char(34)";
}
fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
```

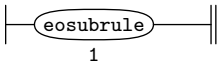
30. *Relement*'**s subrule 2.**



⟨ Relement subrule 2 op directive 30 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__->fsm_tbl__;
++fsm->elem_no_;
std::string mp_obj_type("Box_solid");
std::string mp_elem_name(sf->p1__->Rule_in_stbl()->r_def()->rule_name()->c_str());
fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
```

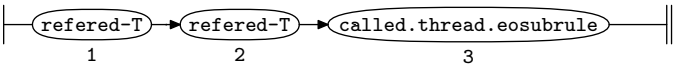
31. *Relement*'s subrule 3.



⟨ Relement subrule 3 op directive 31 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__->fsm_tbl__;
++fsm->elem_no_;
if (fsm->elem_no_ == 1) {       /* epsilon */
  std::string mp_obj_type("Circle_solid");
  std::string mp_elem_name("␣");
  fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
}
```

32. *Relement*'s subrule 4.

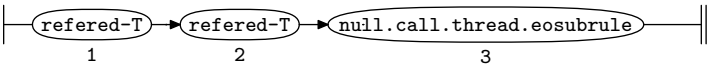


Why is there a re-aligning on the number of *rule_s_subrule_no_elems*? Two reasons: Due to the tree walk being top down, initially the number of items are determined by the *subrule − def* T before processing the actual subrule sentence. Due to the calling of threads, this number gets shrunk as i combine the 3 parts of the called thread into 1 item. Though it looks like a mistake in the emitted *mpost* code, it's a slight inefficiency to manage a nice picture.

This comment hold for the other version of thread call below.

```
⟨ Relement subrule 4 op directive 32 ⟩ ≡
  Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__->fsm_tbl__;
  KCHARP re_align_nos = "rule_s_subrule_no_elems[%i][%i]:= %i;";
  sprintf (fsm->big_buf_, re_align_nos, fsm->rule_no_, fsm->subrule_no_, 3);
  fsm->omp_file_ << fsm->big_buf_ << "% re-align cnt" << endl;
  ++fsm->elem_no_;
  std::string mp_obj_type("Circle_dotted");
  std::string mp_elem_name;
  if (sf->p1__->its_t_def()->enum_id() ≡ T_Enum::T_LR1_parallel_operator_) {
    mp_elem_name += "|||";
  }
  else {
    mp_elem_name += "|t|";
  }
  fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
  ++fsm->elem_no_;
  mp_obj_type.clear();
  mp_obj_type += "Circle_dotted";
  mp_elem_name.clear();
  mp_elem_name += sf->p2__->t_in_stbl()->t_def()->t_name()->c_str();
  string::size_type x = mp_elem_name.find("\\\"");
  if (x ≠ string::npos) {
    mp_elem_name.erase();
    mp_elem_name += "char(34)";
  }
  fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
  ++fsm->elem_no_;
  mp_elem_name.clear();
  mp_obj_type.clear();
  mp_obj_type += "Box_dotted";
  mp_elem_name += sf->p3__->ns()->identifier()->c_str();
  mp_elem_name += "::";
  mp_elem_name += sf->p3__->called_thread_name()->identifier()->c_str();
  fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
```

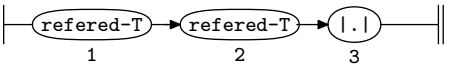
33. *Relement*'s subrule 5.



⟨ Relement subrule 5 op directive 33 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__->fsm_tbl__;
KCHARP re_align_nos = "rule_s_subrule_no_elems[%i][%i]:= %i;";
sprintf(fsm->big_buf_, re_align_nos, fsm->rule_no_, fsm->subrule_no_, 3);
fsm->omp_file_ << fsm->big_buf_ << "% re-align cnt" << endl;
++fsm->elem_no_;
std::string mp_obj_type("Circle_dotted");
std::string mp_elem_name;
if (sf->p1__->its_t_def()->enum_id() == T_Enum::T_LR1_parallel_operator_) {
  mp_elem_name += "|||";
}
else {
  mp_elem_name += "|t|";
}
fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
mp_elem_name.clear();
++fsm->elem_no_;
mp_elem_name += sf->p2__->t_in_stbl()->t_def()->t_name()->c_str();
string::size_type x = mp_elem_name.find("\\\"");
if (x != string::npos) {
  mp_elem_name.clear();
  mp_elem_name += "char(34)";
}
fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
++fsm->elem_no_;
mp_elem_name.clear();
mp_obj_type.clear();
mp_obj_type += "Box_dotted";
mp_elem_name += "NULL";
fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
```

34. *Relement*'**s subrule 6.**



⟨ Relement subrule 6 op directive 34 ⟩ ≡

```
Cmpost_output * fsm = ( Cmpost_output * ) rule_info__.parser__->fsm_tbl__;
++fsm->elem_no_;
std::string mp_obj_type("Circle_solid");
std::string mp_elem_name(sf->p1__->t_in_stbl()->t_def()->t_name()->c_str());
string::size_type x = mp_elem_name.find("\\\"");
if (x != string::npos) {
  mp_elem_name.erase();
  mp_elem_name += "char(34)";
}
fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
++fsm->elem_no_;
mp_obj_type.clear();
mp_obj_type += "Circle_solid";
mp_elem_name.clear();
mp_elem_name += sf->p2__->t_in_stbl()->t_def()->t_name()->c_str();
x = mp_elem_name.find("\\\"");
if (x != string::npos) {
  mp_elem_name.erase();
  mp_elem_name += "char(34)";
}
fsm->MPOST_CWEB_wrt_mp_rhs_elem(fsm, mp_elem_name, mp_obj_type);
```

35. First Set Language for O_2^{linker}.

```
/*
 File: mpost_output.fsc
 Date and Time: Fri Jan  2 15:33:44 2015
*/
transitive   n
grammar-name "mpost_output"
name-space   "NS_mpost_output"
thread-name  "Cmpost_output"
monolithic   y
file-name    "mpost_output.fsc"
no-of-T      569
list-of-native-first-set-terminals 1
   T_grammar_phrase
end-list-of-native-first-set-terminals
list-of-transitive-threads 0
end-list-of-transitive-threads
list-of-used-threads 0
end-list-of-used-threads
fsm-comments
"Output grammar rules railroad diagrams for mpost that cweb program uses."
```

36. Lr1 State Network.

⇒ State: 1 state type: ^{s}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rgrammar_phrase			2	1	1	grammar-phrase			1	2	2	
c	Rmpost_output			1	1	1	Rgrammar_phrase	$\underline{Rks\_epi^{\epsilon}}$ $\underline{Rfsm\_phrase}$		1	3	22	

⇒$^{grammar-phrase}$ State: 2 state type: ^{r}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rgrammar_phrase			2	1	2				1	0	2	1

⇒$^{Rgrammar\_phrase}$ State: 3 state type: $^{s/r}$

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rks_epi			3	2	1	ϵ			3	0	3	2
c	Rk			5	1	1	cweb-marker			3	42	42	
t	Rmpost_output			1	1	2	Rks_epi	$\underline{Rfsm\_phrase}$		1	4	22	
c	Rks			4	1	1	Rks	$\underline{Rk}$		3	43	44	
c	Rks_epi			3	1	1	Rks			3	43	43	
c	Rks			4	2	1	Rk			3	45	45	

⇒$^{Rks\_epi}$ State: 4 state type: ^{s}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rfsm_phrase			6	1	1	fsm-phrase			4	46	46	
t	Rmpost_output			1	1	3	Rfsm_phrase	$\underline{Rks\_epi^{\epsilon}}$ $\underline{Rparallel\_phrase}$		1	5	22	

⇒$^{Rfsm\_phrase}$ State: 5 state type: $^{s/r}$

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rks_epi			3	2	1	ϵ			5	0	5	3
c	Rk			5	1	1	cweb-marker			5	42	42	
t	Rmpost_output			1	1	4	Rks_epi	$\underline{Rparallel\_phrase}$		1	6	22	
c	Rks			4	1	1	Rks	$\underline{Rk}$		5	43	44	
c	Rks_epi			3	1	1	Rks			5	43	43	
c	Rks			4	2	1	Rk			5	45	45	

⇒$^{Rks\_epi}$ State: 6 state type: ^{s}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rparallel_phrase			7	2	1	|.|			6	47	47	
c	Rparallel_phrase			7	1	1	parallel-parser-phrase			6	48	48	
t	Rmpost_output			1	1	5	Rparallel_phrase	$\underline{Rks\_epi^{\epsilon}}$ $\underline{RT\_enum\_phrase}$		1	7	22	

⇒$^{Rparallel\_phrase}$ State: 7 state type: $^{s/r}$

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rks_epi			3	2	1	ϵ			7	0	7	4
c	Rk			5	1	1	cweb-marker			7	42	42	
t	Rmpost_output			1	1	6	Rks_epi	$\underline{RT\_enum\_phrase}$		1	8	22	
c	Rks			4	1	1	Rks	$\underline{Rk}$		7	43	44	
c	Rks_epi			3	1	1	Rks			7	43	43	
c	Rks			4	2	1	Rk			7	45	45	

⇒$^{Rks\_epi}$ State: 8 state type: ^{s}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	RT_enum_phrase			8	1	1	T-enum-phrase			8	49	49	

t	Rmpost_output	1	1	7	RT_enum_phrase *Rks_epi*$^{\epsilon}$ *Rlr1_k_phrase*	1	9	22		

⇒*RT_enum_phrase* State: 9 state type: $^{s/r}$

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rks_epi		3	2	1	ϵ		9	0	9	5
c	Rk		5	1	1	cweb-marker		9	42	42	
t	Rmpost_output		1	1	8	Rks_epi *Rlr1_k_phrase*		1	10	22	
c	Rks		4	1	1	Rks *Rk*		9	43	44	
c	Rks_epi		3	1	1	Rks		9	43	43	
c	Rks		4	2	1	Rk		9	45	45	

⇒*Rks_epi* State: 10 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rlr1_k_phrase		11	1	1	lr1-k-phrase		10	50	50	
t	Rmpost_output		1	1	9	Rlr1_k_phrase *Rks_epi*$^{\epsilon}$ *Rrc_phrase*		1	11	22	

⇒*Rlr1_k_phrase* State: 11 state type: $^{s/r}$

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rks_epi		3	2	1	ϵ		11	0	11	6
c	Rk		5	1	1	cweb-marker		11	42	42	
t	Rmpost_output		1	1	10	Rks_epi *Rrc_phrase*		1	12	22	
c	Rks		4	1	1	Rks *Rk*		11	43	44	
c	Rks_epi		3	1	1	Rks		11	43	43	
c	Rks		4	2	1	Rk		11	45	45	

⇒*Rks_epi* State: 12 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rrc_phrase		10	1	1	rc-phrase		12	51	51	
t	Rmpost_output		1	1	11	Rrc_phrase *Rks_epi*$^{\epsilon}$ *Rterms_phrase*		1	13	22	

⇒*Rrc_phrase* State: 13 state type: $^{s/r}$

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rks_epi		3	2	1	ϵ		13	0	13	7
c	Rk		5	1	1	cweb-marker		13	42	42	
t	Rmpost_output		1	1	12	Rks_epi *Rterms_phrase*		1	14	22	
c	Rks		4	1	1	Rks *Rk*		13	43	44	
c	Rks_epi		3	1	1	Rks		13	43	43	
c	Rks		4	2	1	Rk		13	45	45	

⇒*Rks_epi* State: 14 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rterms_phrase		12	1	1	terminals-phrase		14	52	52	
t	Rmpost_output		1	1	13	Rterms_phrase *Rks_epi*$^{\epsilon}$ *Rerr_phrase*		1	15	22	

⇒*Rterms_phrase* State: 15 state type: $^{s/r}$

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rks_epi		3	2	1	ϵ		15	0	15	8
c	Rk		5	1	1	cweb-marker		15	42	42	
t	Rmpost_output		1	1	14	Rks_epi *Rerr_phrase*		1	16	22	
c	Rks		4	1	1	Rks *Rk*		15	43	44	
c	Rks_epi		3	1	1	Rks		15	43	43	
c	Rks		4	2	1	Rk		15	45	45	

$\Rightarrow^{Rks\_epi}$ State: 16 state type: ^{s}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rerr_phrase			9	1	1	error-symbols-phrase			16	53	53	
t	Rmpost_output			1	1	15	Rerr_phrase	$\underline{Rks\_epi^{\epsilon}}$ $\underline{Rrule\_phrase}$		1	17	22	

$\Rightarrow^{Rerr\_phrase}$ State: 17 state type: $^{s/r}$

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rks_epi			3	2	1	ϵ			17	0	17	9
c	Rk			5	1	1	cweb-marker			17	42	42	
t	Rmpost_output			1	1	16	Rks_epi	$\underline{Rrule\_phrase}$		1	18	22	
c	Rks			4	1	1	Rks	$\underline{Rk}$		17	43	44	
c	Rks_epi			3	1	1	Rks			17	43	43	
c	Rks			4	2	1	Rk			17	45	45	

$\Rightarrow^{Rks\_epi}$ State: 18 state type: ^{s}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rrule_phrase			13	1	1	rules-phrase			18	54	54	
t	Rmpost_output			1	1	17	Rrule_phrase	$\underline{Rrules}$		1	19	22	

$\Rightarrow^{Rrule\_phrase}$ State: 19 state type: $^{s/r}$

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rks_epi			3	2	1	ϵ			19	0	19	10
c	Rk			5	1	1	cweb-marker			19	42	42	
c	Rrules			14	1	1	Rks_epi	$\underline{Rrule}$		19	55	56	
c	Rks			4	1	1	Rks	$\underline{Rk}$		19	43	44	
c	Rks_epi			3	1	1	Rks			19	43	43	
c	Rks			4	2	1	Rk			19	45	45	
c	Rrules			14	2	1	Rrules	$\underline{Rks\_epi^{\epsilon}}$ $\underline{Rrule}$		19	20	24	
t	Rmpost_output			1	1	18	Rrules	$\underline{Rks\_epi^{\epsilon}}$ $\underline{eog}$		1	20	22	

$\Rightarrow^{Rrules}$ State: 20 state type: $^{s/r}$

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rks_epi			3	2	1	ϵ			20	0	20	11
c	Rk			5	1	1	cweb-marker			20	42	42	
t	Rrules			14	2	2	Rks_epi	$\underline{Rrule}$		19	21	24	
t	Rmpost_output			1	1	19	Rks_epi	$\underline{eog}$		1	21	22	
c	Rks			4	1	1	Rks	$\underline{Rk}$		20	43	44	
c	Rks_epi			3	1	1	Rks			20	43	43	
c	Rks			4	2	1	Rk			20	45	45	

$\Rightarrow^{Rks\_epi}$ State: 21 state type: ^{s}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rmpost_output			1	1	20	eog			1	22	22	
c	Rrule_def			16	1	1	rule-def			21	23	23	
t	Rrules			14	2	3	Rrule			19	24	24	
c	Rrule			15	1	1	Rrule_def	$\underline{Rsubrules}$		21	25	27	

$\Rightarrow^{eog}$ State: 22 state type: ^{r}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rmpost_output			1	1	21				1	0	22	12

⇒*rule−def* State: 23 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rrule_def			16	1	2				21	0	23	13

⇒*Rrule* State: 24 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rrules			14	2	4				19	0	24	14

⇒*Rrule_def* State: 25 state type: *s*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rsubrule_def			19	1	1	subrule-def			25	26	26	
t	Rrule			15	1	2	Rsubrules			21	27	27	
c	Rsubrules			17	2	1	Rsubrules *Rsubrule*			25	27	28	
c	Rsubrules			17	1	1	Rsubrule			25	41	41	
c	Rsubrule			18	1	1	Rsubrule_def *Relements*			25	29	38	

⇒*subrule−def* State: 26 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rsubrule_def			19	1	2				25	0	26	15

⇒*Rsubrules* State: 27 state type: *s/r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rrule			15	1	3				21	0	27	14
c	Rsubrule_def			19	1	1	subrule-def			27	26	26	
t	Rsubrules			17	2	2	Rsubrule			25	28	28	
c	Rsubrule			18	1	1	Rsubrule_def *Relements*			27	29	38	

⇒*Rsubrule* State: 28 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rsubrules			17	2	3				25	0	28	16

⇒*Rsubrule_def* State: 29 state type: *s*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Relement			21	4	1	refered-T			29	30	34	
c	Relement			21	5	1	refered-T			29	30	35	
c	Relement			21	6	1	refered-T			29	30	33	
c	Relement			21	1	1	refered-T			29	30	31	
c	Relement			21	2	1	refered-rule			29	36	36	
c	Relement			21	3	1	eosubrule			29	37	37	
t	Rsubrule			18	1	2	Relements			27	38	38	
c	Relements			20	2	1	Relements *Relement*			29	38	39	
c	Relements			20	1	1	Relement			29	40	40	

⇒*refered−T* State: 30 state type: *s*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Relement			21	1	2	`|.|`			29	31	31	
t	Relement			21	4	2	refered-T			29	32	34	
t	Relement			21	5	2	refered-T			29	32	35	
t	Relement			21	6	2	refered-T			29	32	33	

⇒`|.|` State: 31 state type: *r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Relement		21	1	3			29	0	31	17

⇒*refered−T* State: 32 state type: *s*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Relement		21	6	3	`|.|`		29	33	33	
t	Relement		21	4	3	called thread eosubrule		29	34	34	
t	Relement		21	5	3	null call thread eosubrule		29	35	35	

⇒`|.|` State: 33 state type: *r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Relement		21	6	4			29	0	33	17

⇒*calledthreadeosubrule* State: 34 state type: *r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Relement		21	4	4			29	0	34	17

⇒*nullcallthreadeosubrule* State: 35 state type: *r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Relement		21	5	4			29	0	35	17

⇒*refered−rule* State: 36 state type: *r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Relement		21	2	2			29	0	36	17

⇒*eosubrule* State: 37 state type: *r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Relement		21	3	2			29	0	37	17

⇒*Relements* State: 38 state type: *s/r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rsubrule		18	1	3			27	0	38	16
c	Relement		21	4	1	refered-T		38	30	34	
c	Relement		21	5	1	refered-T		38	30	35	
c	Relement		21	6	1	refered-T		38	30	33	
c	Relement		21	1	1	refered-T		38	30	31	
c	Relement		21	2	1	refered-rule		38	36	36	
c	Relement		21	3	1	eosubrule		38	37	37	
t	Relements		20	2	2	Relement		29	39	39	

⇒*Relement* State: 39 state type: *r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Relements		20	2	3			29	0	39	17

⇒*Relement* State: 40 state type: *r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Relements		20	1	2			29	0	40	17

⇒*Rsubrule* State: 41 state type: *r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rsubrules		17	1	2			25	0	41	16

⇒*cweb−marker* State: 42 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rk			5	1	2				3	0	42	18

⇒*Rks* State: 43 state type: *s/r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rks_epi			3	1	2				3	0	43	2
c	Rk			5	1	1	cweb-marker			43	42	42	
t	Rks			4	1	2	Rk			3	44	44	

⇒*Rk* State: 44 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rks			4	1	3				3	0	44	18

⇒*Rk* State: 45 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rks			4	2	2				3	0	45	18

⇒*fsm−phrase* State: 46 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rfsm_phrase			6	1	2				4	0	46	19

⇒`|.|` State: 47 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rparallel_phrase			7	2	2				6	0	47	20

⇒*parallel−parser−phrase* State: 48 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rparallel_phrase			7	1	2				6	0	48	20

⇒*T−enum−phrase* State: 49 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	RT_enum_phrase			8	1	2				8	0	49	21

⇒*lr1−k−phrase* State: 50 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rlr1_k_phrase			11	1	2				10	0	50	22

⇒*rc−phrase* State: 51 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rrc_phrase			10	1	2				12	0	51	23

⇒*terminals−phrase* State: 52 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rterms_phrase			12	1	2				14	0	52	24

⇒*error−symbols−phrase* State: 53 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rerr_phrase			9	1	2				16	0	53	25

⇒*rules−phrase* State: 54 state type: *r*

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rrule_phrase		13	1	2			18	0	54	26

$\Rightarrow^{Rks\_epi}$ State: 55 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rrule_def		16	1	1	rule-def		55	23	23	
t	Rrules		14	1	2	Rrule		19	56	56	
c	Rrule		15	1	1	Rrule_def *Rsubrules*		55	25	27	

$\Rightarrow^{Rrule}$ State: 56 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rrules		14	1	3			19	0	56	14

37. Index.

⟨ Cmpost_output constructor directive 4 ⟩
⟨ Cmpost_output op directive 5 ⟩
⟨ Cmpost_output user-declaration directive 6 ⟩
⟨ Cmpost_output user-prefix-declaration directive 7 ⟩
⟨ Relement subrule 1 op directive 29 ⟩
⟨ Relement subrule 2 op directive 30 ⟩
⟨ Relement subrule 3 op directive 31 ⟩
⟨ Relement subrule 4 op directive 32 ⟩
⟨ Relement subrule 5 op directive 33 ⟩
⟨ Relement subrule 6 op directive 34 ⟩
⟨ Rerr_phrase subrule 1 op directive 16 ⟩
⟨ Rfsm_phrase subrule 1 op directive 13 ⟩
⟨ Rk subrule 1 op directive 12 ⟩
⟨ Rlr1_k_phrase subrule 1 op directive 18 ⟩
⟨ Rmpost_output subrule 1 op directive 8 ⟩
⟨ Rrule_def subrule 1 op directive 23 ⟩
⟨ Rrule_phrase subrule 1 op directive 20 ⟩
⟨ Rsubrule_def subrule 1 op directive 26 ⟩
⟨ Rterms_phrase subrule 1 op directive 19 ⟩

mpost_output Grammar

Date: January 2, 2015 at 15:37

File: mpost_output.lex

Ns: NS_mpost_output

Version: 1.0 Debug: false

Grammar Comments: Type: Monolithic

Output grammar rules railroad diagrams for mpost that cweb program uses.