1. Copyright.

2. *la_expr* **grammar.**
Evaluate the lookahead expression making sure that the elements are well formed. It's output is a set of enumerated values of Terminals that is deposited into *T_parallel_la_boundary*'s *la_first_set_*. *la_expr_lexical* creates its input token container. As a help to my outputting of the CWEB document, i rebuild the source la expression while parsing it and also deposit it back into *T_parallel_la_boundary*'s *cweb_la_srce_expr*. This source expression gets outputted as part of a thread's cweb doc's banner.
Let's review the problem:
There are "+" or "-" operators that add or remove elements from the set. How can the removal operator exist when Terminal elements are played with? There are 2 situations that define groups of elements:

1) eolr—a terminal representing all terminals including self
2) rule—where 1st element of each subrule is a terminal or a rule

The interesting part is recursion of a rule containing a rule. Of course recurse is devine. A subtlty creeps in on the use of the "eolr" terminal. It was created to lower fat-city of lookahead sets and to save the Carpal Tunnel Syndrome caused by typing. If the "eolr" is the only terminal present in the lookahead expression then it's singular. When operators are applied to it, it deconsolidates all its contents minus itself to allow for the operators to become effective. For example A - eolr + B is a round about way to just have B in its set. This could be taken as an error but i'm in a "benign state of mind" as the song goes.

Let's review what "eolr" can stand for. If the lookahead boundary takes on all contexts, then it represents all terminals: no refinement to the run context is needed. This is a lookahead(0) context. This could be represented by the empty set but i didn't want to complicate the parsing algorithm to deal with this situation and so it's a xxx(1) lookahead where u plug in your favorite bottomup algo descriptin in place of the hootch. When refinement to the contents within the lookahead boundary is required, the "eolr" is followed by the subtraction operator to remove the ambiguous component(s) from the boundary: ambiguity only shows up when lring the grammar. The "eolr" coupled with the removal operator is not a universal rule but is dictated by the amount of effort the grammar writer needs to declare the lookahead boundary. For the explicit coders, each specific terminal can be stated by the addition operator but as an aid to the grammar writer a rule definition can do the same thing and reduce the size of the written lookahead expression. This coding indirection can be more explicit by intent of a well chosen rule's name. It costs a bit more in coding a rule definition but it is more flexible in modification and in the reading of a grammar. A rule definition is allowed where its only purpose is use within a lookahead expression. One can still define rules and not use them: a bit of junk code or dangling thoughts... but to each their own. O_2 only honks when an undefined rule is referenced in a subrule.

Bugs:
The *set_difference* algorithm is buggy with MS. So write my own! The bugs are:

1) does not work A − B where B is empty
2) A − B have data and the resulting set is empty

Item 2 throws an error: "map/set iterators are not incrementable" from VS2005.
set_difference(A_set→begin(),A_set→end() ,B_set→begin(),B_set→end(),fset_.begin());
Another misfortune sequential reading of a set is not gauranteed to be in ascending order. So use find probe to determine membership.

Exploding "eolr":
It means use-all-terminals. What is considered a Terminal? Every terminal except itself and the meta terminals: `|||`, `|+|`,`|?|`, `|.|`, `|r|`, `|t|`. In parsing, the metaterminals represent contextual positions within the fsa state that act as action triggers. They only get actioned if they are present within the current fsa's parse state. Their actions are ordered: shifts take precedence over reduce. The first shift tried is the current token in the parse stream. After this the metaterminals are tried in the following order:

`|.|` acts as an epsilon
`|+|` as the all shift
`|r|` represents in a reduced LA set the `|||` operator
`|t|` is the transience operator used by O_2's linker

`|?|` is the wild error catching operator; the unexpected Tes

Added support for Rule recycling as an new/delete optimization.

As rules are recycled, make sure that any temporary variables are initialed properly or the part residues will haunt u. Basically cleared the temporary set variable *fset_* in *Rt* and *Ra* rules before its use.

3. Fsm Cla_expr class.

4. Cla_expr constructor directive.

⟨ Cla_expr constructor directive 4 ⟩ ≡

```
ddd_idx_ = 0;
ddd_[ddd_idx_] = 0;
gps_for_error_reporting_ = 0;
```

5. Cla_expr op directive.

⟨ Cla_expr op directive 5 ⟩ ≡

```
ddd_idx_ = 0;
ddd_[ddd_idx_] = 0;
gps_for_error_reporting_ = 0;
```

6. Cla_expr user-declaration directive.

⟨ Cla_expr user-declaration directive 6 ⟩ ≡

```
public: char ddd_[1024 * 32];
  int ddd_idx_;
  yacco2::CAbs_lr1_sym * gps_for_error_reporting_;
  void copy_str_into_buffer(std::string * Str);
  void copy_kstr_into_buffer(const char *Str);
  void unionize_sets(T_IN_STBL_SET_type * Add_to_set, T_IN_STBL_SET_type * A_set,
      T_IN_STBL_SET_type * B_set);
  void set_differences(T_IN_STBL_SET_type * Add_to_set, T_IN_STBL_SET_type * A_set,
      T_IN_STBL_SET_type * B_set);
  void explode_eolr(T_IN_STBL_SET_type * A_set);
  void add_element_to_set(T_in_stbl * Elem, T_IN_STBL_SET_type * Set_to_add_to); void add_rule_to_set
      (NS_yacco2_terminals::rule_def * Rule, T_IN_STBL_SET_type * Set_to_add_to, std::set <
      int > *Rules_already_processed ) ;
```

7. Cla_expr user-implementation directive.

```
⟨ Cla_expr user-implementation directive 7 ⟩ ≡
  void Cla_expr::copy_str_into_buffer(std::string * Str)
  {
    const char *y = Str->c_str( );
    int x(0);
    for ( ; y[x] ≠ 0; ++x, ++ddd_idx_) ddd_[ddd_idx_] = y[x];
    ddd_[ddd_idx_] = 0;
  }
  void Cla_expr::copy_kstr_into_buffer(const char *Str)
  {
    const char *y = Str;
    int x(0);
    for ( ; y[x] ≠ 0; ++x, ++ddd_idx_) ddd_[ddd_idx_] = y[x];
    ddd_[ddd_idx_] = 0;
  }
  void Cla_expr::unionize_sets(T_IN_STBL_SET_type * Add_to_set, T_IN_STBL_SET_type * A_set,
          T_IN_STBL_SET_type * B_set)
  {
    if (A_set->find(STBL_T_ITEMS[LR1_Eolr]) ≠ A_set->end( )) {
      Add_to_set->insert(A_set->begin( ), A_set->end( ));
      return;
    }
    if (B_set->find(STBL_T_ITEMS[LR1_Eolr]) ≠ B_set->end( )) {
      Add_to_set->insert(B_set->begin( ), B_set->end( ));
      return;
    }
    Add_to_set->insert(A_set->begin( ), A_set->end( ));
    Add_to_set->insert(B_set->begin( ), B_set->end( ));
  }
  void Cla_expr::explode_eolr(T_IN_STBL_SET_type * A_set)
  {
    A_set->clear( );
    T_enum_phrase * e_p = O2_T_ENUM_PHASE;
    A_set->insert(STBL_T_ITEMS[LR1_Eog]);      /* eog */
    int tt = e_p->total_enumerate( );
    for (int x = 8; x < tt; ++x) {      /* balance of meta Ts bypassed. Start from RC */
      A_set->insert(STBL_T_ITEMS[x]);
    }
  }
  void Cla_expr::set_differences(T_IN_STBL_SET_type * Add_to_set, T_IN_STBL_SET_type * A_set,
          T_IN_STBL_SET_type * B_set)
  {
    if (A_set->find(STBL_T_ITEMS[LR1_Eolr]) ≠ A_set->end( )) {
      explode_eolr(A_set);
    }
    if (B_set->find(STBL_T_ITEMS[LR1_Eolr]) ≠ B_set->end( )) {
      explode_eolr(B_set);
    }
    if (A_set->empty( ) ≡ true) {
```

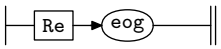
```
      CAbs_lr1_sym * sym = new Err_empty_set_removal_in_la_expr;
      sym→set_rc(*gps_for_error_reporting_, __FILE__, __LINE__);
      parser__→set_stop_parse(true);
      ADD_TOKEN_TO_ERROR_QUEUE_FSM(*sym);
      parser__→set_abort_parse(true);
    }
    T_IN_STBL_SET_ITER_type Ai = A_set→begin( );
    T_IN_STBL_SET_ITER_type Aie = A_set→end( );
    T_IN_STBL_SET_ITER_type Bie = B_set→end( );
    T_IN_STBL_SET_ITER_type AinBi;
    for ( ; Ai ≠ Aie; ++Ai) {
      AinBi = B_set→find(*Ai);
      if (AinBi ≠ Bie) {
        continue;      /* bypass */
      }
      Add_to_set→insert(*Ai);
    }
  }
  void Cla_expr::add_element_to_set(T_in_stbl * Elem, T_IN_STBL_SET_type * Set_to_add_to)
  {
    if (Set_to_add_to→find(STBL_T_ITEMS[LR1_Eolr]) ≠ Set_to_add_to→end( )) return;
    if (Elem ≡ STBL_T_ITEMS[LR1_Eolr]) {
      Set_to_add_to→clear( );
    }
    Set_to_add_to→insert(Elem);
  }
  void Cla_expr ::add_rule_to_set (NS_yacco2_terminals::rule_def * Rule,
      T_IN_STBL_SET_type * Set_to_add_to, std::set < int > *Rules_already_processed ) { using
      namespace NS_yacco2_terminals;
  int r_id = Rule→enum_id( );
  if (Rules_already_processed→find(r_id) ≠ Rules_already_processed→end( )) return;
  Rules_already_processed→insert(r_id);
  AST * r_t = Rule→rule_s_tree( );
  set < yacco2::INT > elem_filter;
  elem_filter.insert(NS_yacco2_T_enum::T_Enum::T_T_subrule_def_);
  tok_can_ast_functor ast_functor;
  ast_prefix_wbreadth_only pwbo(*r_t, &ast_functor, &elem_filter, true);
  tok_can < AST *> tok_can(pwbo); for (int xxx = 0; tok_can.operator[](xxx) ≠ yacco2::PTR_LR1_eog__;
      ++xxx) { T_subrule_def * sr_def = ( T_subrule_def * ) tok_can.operator[](xxx);
  vector < CAbs_lr1_sym *> *elems_list = sr_def→subrule_elems( );
  CAbs_lr1_sym * sym = (*elems_list)[0];
  int id = sym→enumerated_id__; switch (id) {
case NS_yacco2_T_enum::T_Enum::T_T_eosubrule_:
  break; case NS_yacco2_T_enum::T_Enum::T_refered_T_: { refered_T * rt = ( refered_T * ) sym;
  add_element_to_set(rt→t_in_stbl( ), Set_to_add_to);
  break; } case NS_yacco2_T_enum::T_Enum::T_refered_rule_: { refered_rule * rt = ( refered_rule * ) sym;
  rule_def * r = rt→Rule_in_stbl( )→r_def( );
  add_rule_to_set(r, Set_to_add_to, Rules_already_processed);
  break; }
default: { } } } }
```

8. Cla_expr user-prefix-declaration directive.

⟨ Cla_expr user-prefix-declaration directive 8 ⟩ ≡
```
#include "o2_externs.h"
  extern void XLATE_SYMBOLS_FOR_cweave(const char *Sym_to_xlate, char *Xlated_sym);
```

9. *Rla_expr* **rule.**

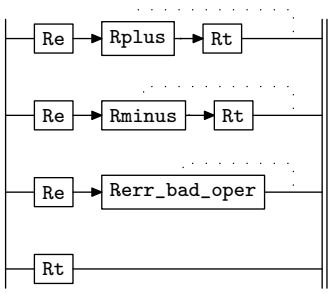
Rla_expr



⟨ Rla_expr subrule 1 op directive 9 ⟩ ≡
```
  Cla_expr * fsm = ( Cla_expr * ) rule_info__.parser__→fsm_tbl__;
  if (sf→p1__→fset_.empty( ) ≡ true) {
    CAbs_lr1_sym * sym = new Err_la_expr_calc_empty_set;
    sym→set_rc(*fsm→gps_for_error_reporting_, __FILE__, __LINE__);
    ADD_TOKEN_TO_ERROR_QUEUE(*sym);
    rule_info__.parser__→set_abort_parse(true);
  }
  T_parallel_parser_phrase * la_ph = O2_PP_PHASE;
  T_parallel_la_boundary * la = la_ph→la_bndry( );
  la→la_first_set(sf→p1__→fset_);
  la→cweb_la_srce_expr((const char *) &fsm→ddd_);
```

10. *Re* **rule.**

Re



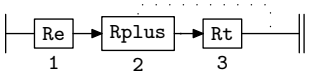
11. Re constructor directive.

⟨ Re constructor directive 11 ⟩ ≡
```
  fset_.clear( );
```

12. Re user-declaration directive.

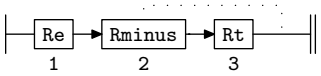
⟨ Re user-declaration directive 12 ⟩ ≡
```
public: T_IN_STBL_SET_type fset_;
```

13. *Re*'**s subrule 1.**



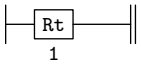
⟨ Re subrule 1 op directive 13 ⟩ ≡
```
  Cla_expr * fsm = ( Cla_expr * ) rule_info__.parser__→fsm_tbl__;
  fset_.clear( );
  fsm→unionize_sets(&fset_, &sf→p1__→fset_, &sf→p3__→fset_);
```

14. *Re*'**s subrule 2.**



⟨ Re subrule 2 op directive 14 ⟩ ≡
Cla_expr ∗ *fsm* = (*Cla_expr* ∗) *rule_info__.parser__→fsm_tbl__*;
fset_.clear( );
fsm→set_differences(&*fset_*, &*sf→p1__→fset_*, &*sf→p3__→fset_*);

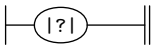
15. *Re*'**s subrule 4.**



⟨ Re subrule 4 op directive 15 ⟩ ≡
fset_.clear( );
fset_.insert(*sf→p1__→fset_.begin*( ), *sf→p1__→fset_.end*( ));

16. *Rerr_bad_oper* **rule.**

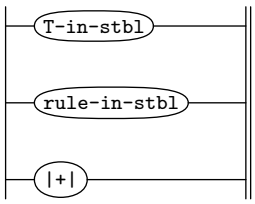
`Rerr_bad_oper`



⟨ Rerr_bad_oper subrule 1 op directive 16 ⟩ ≡
CAbs_lr1_sym ∗ *sym* = **new** *Err_bad_operator_in_la_expr*;
sym→set_rc(∗*sf→p1__*, `__FILE__`, `__LINE__`);
`ADD_TOKEN_TO_ERROR_QUEUE`(∗*sym*);
rule_info__.parser__→set_abort_parse(*true*);

17. *Rt* **rule.**

`Rt`



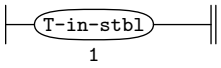
18. Rt constructor directive.

⟨ Rt constructor directive 18 ⟩ ≡
fset_.clear( );

19. Rt user-declaration directive.

⟨ Rt user-declaration directive 19 ⟩ ≡
public: *T_IN_STBL_SET_type fset_*;

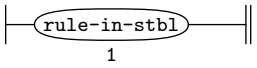
20. *Rt*'s subrule 1.



⟨ Rt subrule 1 op directive 20 ⟩ ≡

```
fset_.clear( );
using namespace NS_yacco2_terminals; Cla_expr *fsm = ( Cla_expr * ) rule_info__.parser__->fsm_tbl__;
T_terminal_def * dt = sf->p1__->t_def( );
fsm->copy_kstr_into_buffer("␣");
char xlate_file[Max_cweb_item_size];
XLATE_SYMBOLS_FOR_cweave(dt->t_name( )->c_str( ), xlate_file);
fsm->copy_kstr_into_buffer(xlate_file);
fset_.clear( );
fset_.insert(sf->p1__);
```

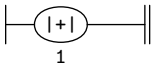
21. *Rt*'s subrule 2.



⟨ Rt subrule 2 op directive 21 ⟩ ≡

```
fset_.clear( );
using namespace NS_yacco2_terminals; Cla_expr *fsm = ( Cla_expr * ) rule_info__.parser__->fsm_tbl__;
rule_def * rt = sf->p1__->r_def( );
fsm->copy_kstr_into_buffer("␣");
char xlate_file[Max_cweb_item_size];
XLATE_SYMBOLS_FOR_cweave(rt->rule_name( )->c_str( ), xlate_file);
fsm->copy_kstr_into_buffer(xlate_file); std::set < int > already_processed_rules;
fsm->add_rule_to_set(rt, &fset_, &already_processed_rules);
```

22. *Rt*'s subrule 3.

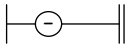


⟨ Rt subrule 3 op directive 22 ⟩ ≡

```
fset_.clear( );
using namespace NS_yacco2_terminals;
int id = sf→p1__→enumerated_id__; switch (id) { case T_Enum :: T_LR1_all_shift_operator_: {
    Cla_expr * fsm = ( Cla_expr * ) rule_info__.parser__→fsm_tbl__;
fsm→copy_kstr_into_buffer("␣$|+|$");
fsm→add_element_to_set(STBL_T_ITEMS[id], &fset_);
break; } case T_Enum :: T_LR1_invisible_shift_operator_: { Cla_expr * fsm = ( Cla_expr * )
    rule_info__.parser__→fsm_tbl__;
fsm→copy_kstr_into_buffer("␣$|.|$");
fsm→add_element_to_set(STBL_T_ITEMS[id], &fset_);
break; } case T_Enum :: T_LR1_parallel_operator_: { Cla_expr * fsm = ( Cla_expr * )
    rule_info__.parser__→fsm_tbl__;
fsm→copy_kstr_into_buffer("␣$|||$");
fsm→add_element_to_set(STBL_T_ITEMS[id], &fset_);
break; } case T_Enum :: T_LR1_fset_transience_operator_: { Cla_expr * fsm = ( Cla_expr * )
    rule_info__.parser__→fsm_tbl__;
fsm→copy_kstr_into_buffer("␣$|t|$");
fsm→add_element_to_set(STBL_T_ITEMS[id], &fset_);
break; } case T_Enum :: T_LR1_reduce_operator_: { Cla_expr * fsm = ( Cla_expr * )
    rule_info__.parser__→fsm_tbl__;
fsm→copy_kstr_into_buffer("␣$|r|$");
fsm→add_element_to_set(STBL_T_ITEMS[id], &fset_);
break; }
default:
  {
    CAbs_lr1_sym * sym = new Err_bad_term_in_la_expr;
    sym→set_rc(*sf→p1__, __FILE__, __LINE__);
    rule_info__.parser__→set_stop_parse(true);
    ADD_TOKEN_TO_ERROR_QUEUE(*sym);
    rule_info__.parser__→set_abort_parse(true);
  }
  }
```

23. *Rminus* **rule.**

Rminus

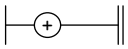


⟨ Rminus subrule 1 op directive 23 ⟩ ≡

```
Cla_expr * fsm = ( Cla_expr * ) rule_info__.parser__→fsm_tbl__;
fsm→gps_for_error_reporting_ = sf→p1__;
fsm→copy_kstr_into_buffer("␣$-$");
```

24. *Rplus* **rule.**

`Rplus`



⟨ Rplus subrule 1 op directive 24 ⟩ ≡

```
Cla_expr * fsm = ( Cla_expr * ) rule_info__.parser__→fsm_tbl__;
fsm→copy_kstr_into_buffer("␣$+$");
```

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

```
/*
 File: la_expr.fsc
 Date and Time: Fri Jan  2 15:33:39 2015
*/
transitive   n
grammar-name "la_expr"
name-space   "NS_la_expr"
thread-name  "Cla_expr"
monolithic   y
file-name    "la_expr.fsc"
no-of-T      569
list-of-native-first-set-terminals 3
   LR1_all_shift_operator
   T_in_stbl
   rule_in_stbl
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
"Parse the lookahead expression after chaffe removed."
```

26. Lr1 State Network.

⇒ State: 1 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rt		4	3	1	|+|		1	2	2	
c	Rt		4	1	1	T-in-stbl		1	3	3	
c	Rt		4	2	1	rule-in-stbl		1	4	4	
c	Rla_expr		1	1	1	Re *eog*		1	5	7	
c	Re		2	1	1	Re *Rplus*		1	5	14	
c	Re		2	3	1	Re *Rerr_bad_oper*		1	5	10	
c	Re		2	2	1	Re *Rminus*		1	5	12	
c	Re		2	4	1	Rt		1	15	15	

⇒|+| State: 2 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rt		4	3	2			1	0	2	1

⇒$^{T-in-stbl}$ State: 3 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rt		4	1	2			1	0	3	1

⇒$^{rule-in-stbl}$ State: 4 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rt		4	2	2			1	0	4	1

⇒^{Re} State: 5 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rerr_bad_oper		3	1	1	|?|		5	6	6	
t	Rla_expr		1	1	2	eog		1	7	7	
c	Rplus		6	1	1	+		5	8	8	
c	Rminus		5	1	1	-		5	9	9	
t	Re		2	3	2	Rerr_bad_oper		1	10	10	
t	Re		2	2	2	Rminus *Rt*		1	11	12	
t	Re		2	1	2	Rplus *Rt*		1	13	14	

⇒|?| State: 6 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rerr_bad_oper		3	1	2			5	0	6	1

⇒^{eog} State: 7 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rla_expr		1	1	3			1	0	7	2

⇒$^{+}$ State: 8 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rplus		6	1	2			5	0	8	3

⇒$^{-}$ State: 9 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rminus		5	1	2			5	0	9	3

⇒$^{Rerr\_bad\_oper}$ State: 10 state type: ^{r}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Re			2	3	3				1	0	10	1

⇒*Rminus* State: 11 state type: *s*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rt			4	3	1	|+|			11	2	2	
c	Rt			4	1	1	T-in-stbl			11	3	3	
c	Rt			4	2	1	rule-in-stbl			11	4	4	
t	Re			2	2	3	Rt			1	12	12	

⇒*Rt* State: 12 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Re			2	2	4				1	0	12	1

⇒*Rplus* State: 13 state type: *s*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rt			4	3	1	|+|			13	2	2	
c	Rt			4	1	1	T-in-stbl			13	3	3	
c	Rt			4	2	1	rule-in-stbl			13	4	4	
t	Re			2	1	3	Rt			1	14	14	

⇒*Rt* State: 14 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Re			2	1	4				1	0	14	1

⇒*Rt* State: 15 state type: *r*

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Re			2	4	2				1	0	15	1

27. Index.

la_expr Grammar

Date: January 2, 2015 at 15:36

File: la_expr.lex

Ns: NS_la_expr

Version: 1.0 Debug: false

Grammar Comments: Type: Monolithic

Parse the lookahead expression after chaffe removed.